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

19741

6481

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

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

git version-control git-commit undo

edited May 29 at 17:56

community wiki
81 revs, 53 users 14%
Peter Mortensen

175 Warning: you should only do this if you have not yet pushed the commit to a remote, otherwise you will mess up the history of others who have already pulled the commit from the remote! – thSoft May 13 '15 at 21:18

75 Here's a very clear and thorough post about undoing things in git, straight from Github. – Nobita Jun 8 '15 at 19:39

54 Before you post a new answer, consider there are already 65+ answers for this question. Make sure that your answer contributes what is not among existing answers. – Sazzad Hissain Khan Jun 15 '17 at 15:26

29 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 '18 at 1:45

3 @YiminRong That can be done with Git's `alias` feature: git-scm.com/book/en/v2/Git-Basics-Git-Aliases – Edric Oct 5 '18 at 14:50

## 78 Answers

1 2 3 next

```
$ git add ...                                   # (4)
$ git commit -c ORIG_HEAD                       # (5)
```

1. This is what you want to undo.
2. This leaves your working tree (the state of your files on disk) unchanged but undoes the commit and leaves the changes you committed unstaged (so they'll appear as "Changes not staged for commit" in `git status`, so you'll need to add them again before committing). If you *only* want to *add* more changes to the previous commit, or change the commit message[1], you could use `git reset --soft HEAD~` instead, which is like `git reset HEAD~` (where `HEAD~` is the same as `HEAD~1`) but leaves your existing changes staged.
3. Make corrections to working tree files.
4. `git add` anything that you want to include in your new commit.
5. Commit the changes, reusing the old commit message. `reset` copied the old head to `.git/ORIG_HEAD`; `commit` with `-c ORIG_HEAD` will open an editor, which initially contains the log message from the old commit and allows you to edit it. If you do not need to edit the message, you could use the `-C` option.

Beware however that if you have added any new changes to the index, using `commit --amend` will add them to your previous commit.

If the code is already pushed to your server and you have permissions to overwrite history (rebase) then:

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

You can also look at this answer:

How to move HEAD back to a previous location? (Detached head) & Undo commits

The above answer will show you `git reflog` which is used to find out what is the SHA-1 which you wish to revert to. Once you found the point to which you wish to undo to use the sequence of commands as explained above.

[1] Note, however, that you don't need to reset to an earlier commit if you just made a mistake in your *commit message*. The easier option is to `git reset` (to unstage any changes you've made since) and then `git commit --amend`, which will open your default commit message editor pre-populated with the last commit message.

435 And if the commit was to the wrong branch, you may `git checkout theRightBranch` with all the changes stages. As I just had to do. – Frank Shearar Oct 5 '10 at 15:44

455 If you're working in DOS, instead of `git reset --soft HEAD^` you'll need to use `git reset --soft HEAD~1` . The ^ is a continuation character in DOS so it won't work properly. Also, `--soft` is the default, so you can omit it if you like and just say `git reset HEAD~1` . – Ryan Lundy Apr 13 '11 at 14:15

26 Also, in zsh you have to quote ^, so `git reset --soft 'HEAD^'` ... at least I did – jberryman Oct 27 '11 at 18:24

150 (Correction to what I wrote above; `--mixed` is the default. `--mixed` means to keep the changed files, but not keep them in the index. `--soft` would keep the changed files and keep them in the index as they were just before the changed commit. Sorry for the confusion.) – Ryan Lundy Nov 17 '11 at 2:40

101 zsh users might get: `zsh: no matches found: HEAD^` - you need to escape ^ i.e. `git reset --soft HEAD\^` – tnajdek Feb 21 '13 at 17:47

10293

Undoing a commit is a little scary if you don't know how it works. But it's actually amazingly easy if you do understand.

Say you have this, where C is your HEAD and (F) is the state of your files.

```
   (F)
A-B-C
    ↑
  master
```

You want to **nuke commit C and never see it again.** You do this:

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

The result is:

```
 (F)
A-B
  ↑
master
```

```
   (F)
A-B-C
    ↑
  master
```

You can do this, leaving off the `--hard`:

```
git reset HEAD~1
```

In this case the result is:

```
   (F)
A-B-C
  ↑
master
```

In both cases, HEAD is just a pointer to the latest commit. When you do a `git reset HEAD~1`, you tell Git to move the HEAD pointer back one commit. But (unless you use `--hard`) you leave your files as they were. So now `git status` shows the changes you had checked into C. You haven't lost a thing!

For the lightest touch, you can even **undo your commit but leave your files and your index**:

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

This not only leaves your files alone, it even leaves your *index* alone. When you do `git status`, you'll see that the same files are in the index as before. In fact, right after this command, you could do `git commit` and you'd be redoing the same commit you just had.

One more thing: **Suppose you destroy a commit** as in the first example, **but then discover you needed it after all**? Tough luck, right?

Nope, there's *still* a way to get it back. Type `git reflog` and you'll see a list of (partial) commit shas (that is, hashes) that you've moved around in. Find the commit you destroyed, and do this:

edited Dec 7 '18 at 9:56

answered Jul 28 '11 at 22:22

Ryan Lundy
**160k** 31 163 197

40 @dma_k, yes. Or you could do `git reset --hard HEAD^^` once. I use the tilde (~) notation because the caret (^) notation doesn't work in DOS. – Ryan Lundy Feb 25 '12 at 15:02

18 Another nice tip: You can re-attach the branch to the commit that you removed it from with `git branch -f <branch> <commit-id>` . Saves having to re-create commits! – naught101 Jun 22 '12 at 13:11

191 For a git beginner, it isn't obvious what the difference is between the last two options (--soft and the one above it). Mentioning the index doesn't help, we don't really know what that means yet. @nessur's connection between soft and Ctrl-Z really helped! But I still don't quite understand the difference between the two options. – Stomp Jun 26 '12 at 15:56

129 It's much better to be told 'why' something works, than just to be told the answer. Kudos to this description - it helped be 'get' git. – Chris Nash Jul 3 '12 at 19:13

61 Missing a crucial point: If the said commit was previously 'pushed' to the remote, any 'undo' operation, no matter how simple, will cause enormous pain and suffering to the rest of the users who have this commit in their local copy, when they do a 'git pull' in the future. So, if the commit was already 'pushed', do this instead: git revert <bad-commit-sha1-id> git push origin : – FractalSpace Nov 8 '13 at 23:43

1967

This took me a while to figure out, so maybe this will help someone...

There are two ways to "undo" your last commit, depending on whether or not you have already made your commit public (pushed to your remote repository):

## How to undo a local commit

Let's say I committed locally, but now want to remove that commit.

```
git log
    commit 101: bad commit    # latest commit, this would be called 'HEAD'
    commit 100: good commit   # second to last commit, this is the one we want
```

To restore everything back to the way it was prior to the last commit, we need to `reset` to the commit before `HEAD` :

Now `git log` will show that our last commit has been removed.

## How to undo a public commit

If you have already made your commits public, you will want to create a new commit which will "revert" the changes you made in your previous commit (current HEAD).

```
git revert HEAD
```

Your changes will now be reverted and ready for you to commit:

```
git commit -m 'restoring the file I removed by accident'
git log
    commit 102: restoring the file I removed by accident
    commit 101: removing a file we don't need
    commit 100: adding a file that we need
```

For more info, check out Git Basics - Undoing Things

edited Jan 30 '18 at 23:27
YakovL
3,350 10 31 45

answered Jun 16 '11 at 17:27
Andrew
103k 164 465 647

90 I found this answer the clearest. `git revert HEAD^` is not the previous, is the previous of the previous. I did : `git revert HEAD` and then push again and it worked :) – nacho4d Jul 14 '11 at 8:32

1685

Add/remove files to get things the way you want:

```
git rm classdir
git add sourcedir
```

The previous, erroneous commit will be edited to reflect the new index state - in other words, it'll be like you never made the mistake in the first place.

Note that you should only do this if you haven't pushed yet. If you have pushed, then you'll just have to commit a fix normally.

edited Oct 12 '16 at 6:45

Vikrant
**4,039** 15 39 58

answered May 29 '09 at 18:16

bdonlan
**181k** 23 224 302

23 Does this work when I did a `git commit --amend` and what I really meant to do is a `git commit` ? – dbm May 18 '11 at 13:07

65 @dbm, if you accidentally amended, use `git reset --soft <oldref>` , where oldref is the commit ID before the amend. You can use `git reflog` to identify the old commit ID. This will undo the effects of the amend, but leave changes staged. Then just do `git commit` to commit as a regular commit. – bdonlan May 18 '11 at 14:20

16 @Dennis, git commit –amend turns the current tree (ie, staged changes) into a commit, overwriting current HEAD. After that point, they're not considered staged anymore because they're part of the commit (ie, git diff –cached is blank), but they're not "removed" or "lost". – bdonlan Feb 1 '12 at 3:08

The option `--amend` was the tricks, to avoid a fake commit! – Laurent Mar 22 at 19:58

954

```
git rm yourfiles/*.class
git commit -a -m "deleted all class files in folder 'yourfiles'"
```

or

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

**Warning: The above command will permanently remove the modifications to the `.java` files (and any other files) that you wanted to commit.**

The `hard reset` to `HEAD-1` will set your working copy to the state of the commit before your wrong commit.

73 "--hard" will get rid of the modified .java files in the working directory that he wanted to commit. – Esko Luontola May 29 '09 at 18:26

25 You can "git stash save" working copy changes, do a hard reset and then "git stash pop" to get them back, though I suppose a soft reset would be simpler. – Asad R. Apr 15 '11 at 13:33

14 `git commit -a -m ""` or `git commit -am ""` naturally! :] – trejder Jun 21 '14 at 16:31

Another 'shortcut' use of stash; if you want to unstage everything (undo git add), just `git stash` , then `git stash pop` – seanriordan08 Dec 8 '15 at 22:30

## To change the last commit

724

Replace the files in the index:

```
git rm --cached *.class
git add *.java
```

Then, if it's a private branch, **amend** the commit:

```
git commit --amend
```

Or, if it's a shared branch, make a new commit:

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

*(**to change a previous commit**, use the awesome interactive rebase)*

ProTip™: Add `*.class` to a gitignore to stop this happening again.

```
git reset @~N
```

Where `N` is the number of commits before `HEAD`, and `@~` resets to the previous commit.

So, instead of amending the commit, you could use:

```
git reset @~
git add *.java
git commit -m "Add .java files"
```

Check out `git help reset`, specifically the sections on `--soft` `--mixed` and `--hard`, for a better understanding of what this does.

## Reflog

If you mess up, you can always use the reflog to find dropped commits:

```
$ git reset @~
$ git reflog
c4f708b HEAD@{0}: reset: moving to @~
2c52489 HEAD@{1}: commit: added some .class files
$ git reset 2c52489
... and you're back where you started
```

edited May 23 '17 at 12:03
Community ♦
1 1

answered Jul 31 '10 at 9:39

Zaz
26.8k 9 60 85

For those reading in future - please note that `git revert` is a separate command - which basically 'resets' a single commimt. – BKSpurgeon Aug 8 '18 at 7:11


Eugen Konkov
**6,931** 3 42 68


Jaco Pretorius
**20.8k** 9 51 89

28 If you committed to the wrong branch: once reverted, switch to the correct branch and cherry-pick the commit. – Kris Jun 27 '12 at 11:02

11 What does that mean, cherry pick the commit? In my case, I was on the wrong branch when I edited a file. I committed it then realized I was in the wrong branch. Using "git reset --soft HEAD~1" got me back to just before the commit, but now if I checkout the correct branch, how do I undo the changes to the file in wrong branch but instead make them (in the same named file) in the correct branch? – astronomerdave Jan 13 '15 at 22:05

I just utilized `git revert commit-id` worked like a charm. Of course then you will need to push your changes. – Casey Robinson Jan 25 '16 at 21:07

6 I believe that would be `git cherry-pick <<erroneous-commit-sha>>` @astronomerdave. From, Mr. Almost-2-Years-Late-to-the-Party. – Tom Howard Oct 20 '16 at 18:19

@Kris: Instead of cherry-pick use rebase. Because it is advanced cherry-picking – Eugen Konkov Nov 10 '18 at 9:38

494

If you are planning to undo a local commit entirely, whatever you change you did on the commit, and if you don't worry anything about that, just do the following command.

```
git reset --hard HEAD^1
```

(This command will ignore your entire commit and your changes will be lost completely from your local working tree). If you want to undo your commit, but you want your changes in the staging area (before commit just like after `git add`) then do the following command.

```
git reset --soft HEAD^1
```

Now your committed files come into the staging area. Suppose if you want to upstage the files, because you need to edit some wrong content, then do the following command

```
git reset HEAD
```

Now committed files to come from the staged area into the unstaged area. Now files are ready to edit, so whatever you change, you

user10075100

Madhan Ayyasamy
**11k** 3 14 18

11 @SMR, In your example, all are pointing into current HEAD only. HEAD^ = HEAD^1. As well as HEAD^1 = HEAD~1. When you use HEAD~2, there is a difference between ~ and ^ symbols. If you use ~2 means “the first parent of the first parent,” or “the grandparent”. – Madhan Ayyasamy Dec 14 '15 at 15:34

466

If you have Git Extras installed, you can run `git undo` to undo the latest commit. `git undo 3` will undo the last 3 commits.

answered Dec 13 '11 at 10:18

nickf
**384k** 175 590 691

433

I wanted to undo the lastest 5 commits in our shared repository. I looked up the revision id that I wanted to rollback to. Then I typed in the following.

```
prompt> git reset --hard 5a7404742c85
HEAD is now at 5a74047 Added one more page to catalogue
prompt> git push origin master --force
Total 0 (delta 0), reused 0 (delta 0)
remote: bb/acl: neoneye is allowed. accepted payload.
To git@bitbucket.org:thecompany/prometheus.git
 + 09a6480...5a74047 master -> master (forced update)
prompt>
```

answered Apr 6 '12 at 13:58

neoneye
**32.1k** 19 138 126

21 Rewriting history on a shared repository is generally a very bad idea. I assume you know what you're doing, I just hope future readers do too. – Brad Koch Dec 7 '12 at 16:02

7 '13 at 10:10

412

I prefer to use `git rebase -i` for this job, because a nice list pops up where I can choose the commits to get rid of. It might not be as direct as some other answers here, but it just *feels right*.

Choose how many commits you want to list, then invoke like this (to enlist last three)

```
git rebase -i HEAD~3
```

Sample list

```
pick aa28ba7 Sanity check for RtmpSrv port
pick c26c541 RtmpSrv version option
pick 58d6909 Better URL decoding support
```

Then Git will remove commits for any line that you remove.

edited Dec 6 '18 at 21:59

Peter Mortensen
**14.2k** 19 88 114

answered Oct 25 '12 at 3:41

Steven Penny
**1**

398

## How to fix the previous local commit

Use git-gui (or similar) to perform a `git commit --amend`. From the GUI you can add or remove individual files from the commit. You can also modify the commit message.

## How to undo the previous local commit

Just reset your branch to the previous location (for example, using `gitk` or `git rebase`). Then reapply your changes from a saved copy. After garbage collection in your local repository, it will be like the unwanted commit never happened. To do all of that in a single

## How to undo a public commit

Perform a reverse cherry pick (git-revert) to undo the changes.

If you haven't yet pulled other changes onto your branch, you can simply do...

```
git revert --no-edit HEAD
```

Then push your updated branch to the shared repository.

*The commit history will show both commits, separately*.

## Advanced: Correction of the *private* branch in public repository

### *This can be dangerous -- be sure you have a local copy of the branch to repush.*

*Also note: You don't want to do this if someone else may be working on the branch.*

```
git push --delete (branch_name) ## remove public version of branch
```

Clean up your branch locally then repush...

```
git push origin (branch_name)
```

*In the normal case, you probably needn't worry about your private-branch commit history being pristine. Just push a followup commit (see 'How to undo a public commit' above), and later, do a squash-merge to hide the history.*

edited Dec 7 '18 at 5:57

community wiki
12 revs, 4 users 76%
nobar

See also: stackoverflow.com/a/30598953 – nobar Apr 5 '16 at 4:20

I think 'private'/'public' would more correctly be 'local'/'remote'. – nobar Mar 28 '18 at 14:59

Correcting a private branch in remote repository can also be done by simply `git push origin (branch_name) --force` – nobar Sep 7 '18 at 12:09

319

If you have committed junk but not pushed,

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

> **HEAD~1** is a shorthand for the commit before head. Alternatively you can refer to the **SHA-1** of the hash if you want to reset to. *--soft* option will delete the commit but it will leave all your changed files "Changes to be committed", as git status would put it.
>
> If you want to get rid of any changes to tracked files in the working tree since the commit before head use "**--hard**" instead.

OR

> If you already pushed and someone pulled which is usually my case, you can't use *git reset*. You can however do a *git revert*,

```
git revert HEAD
```

> This will create a new commit that reverses everything introduced by the accidental commit.

edited Sep 25 '14 at 7:58

community wiki
2 revs, 2 users 87%
santos_mgr

I'm in the 2nd case, but when I do "git revert HEAD" it says "error: Commit [ID] is a merge but no -m option was given. fatal: revert failed". Any suggestions? – metaforge Nov 12 '14 at 19:36

317

If you want to permanently undo it and you have cloned some repository

The commit id can be seen by

```
git log
```

Then you can do -

```
git reset --hard <commit_id>

git push origin <branch_name> -f
```

edited Jun 24 '15 at 9:34

community wiki
2 revs, 2 users 97%
poorva

What if you do not use "<commit_id>" and simply use "git reset –hard"? I typically just want to get rid of my latest updates that I have not committed yet and got back to the latest commit I made, and I always use "git reset --hard". – Jaime Montoya Sep 27 '17 at 23:30

3 @JaimeMontoya To undo latest changes you can use `git reset --hard` , but if you have to hard remove last "n" commits you specify a SHA – poorva Sep 28 '17 at 13:10

269

On SourceTree (GUI for GitHub), you may right-click the commit and do a 'Reverse Commit'. This should undo your changes.

On the terminal:

You may alternatively use:

```
git revert
```

Or:

5 revs, 5 users 56%
Varun Parakh

A single command:

250

```
git reset --soft 'HEAD^'
```

It works great to undo the last local commit!

edited Jul 21 '14 at 20:13

community wiki
3 revs, 3 users 67%
Manish Shrivastava

9 I needed to write git reset --soft "HEAD^" with double quotes, because I write it from Windows command prompt. – Ena Apr 23 '14 at 9:13

2 It should work without any quotes. – jkulak Nov 10 '16 at 19:19

Just reset it doing the command below using `git` :

241

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

**Explain:** what `git reset` does, it's basically `reset` to any commit you'd like to go back to, then if you combine it with `--soft` key, it will go back, but keep the changes in your file(s), so you get back to the stage which the file was just added, `HEAD` is the head of the branch and if you combine with `~1` (in this case you also use `HEAD^` ), it will go back only one commit which what you want...

I create the steps in the image below in more details for you, including all steps that may happens in real situations and committing the code:

edited Oct 27 '17 at 12:46

community wiki
17 revs
Alireza

---

223

**How to undo the last Git commit?**

To restore everything back to the way it was prior to the last commit, we need to reset to the commit before HEAD.

1. If you don't want to keep your changes that you made:

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

2. If you want to keep your changes:

```
git reset --soft HEAD^
```

Now check your git log. It will show that our last commit has been removed.

edited Mar 25 '17 at 8:14

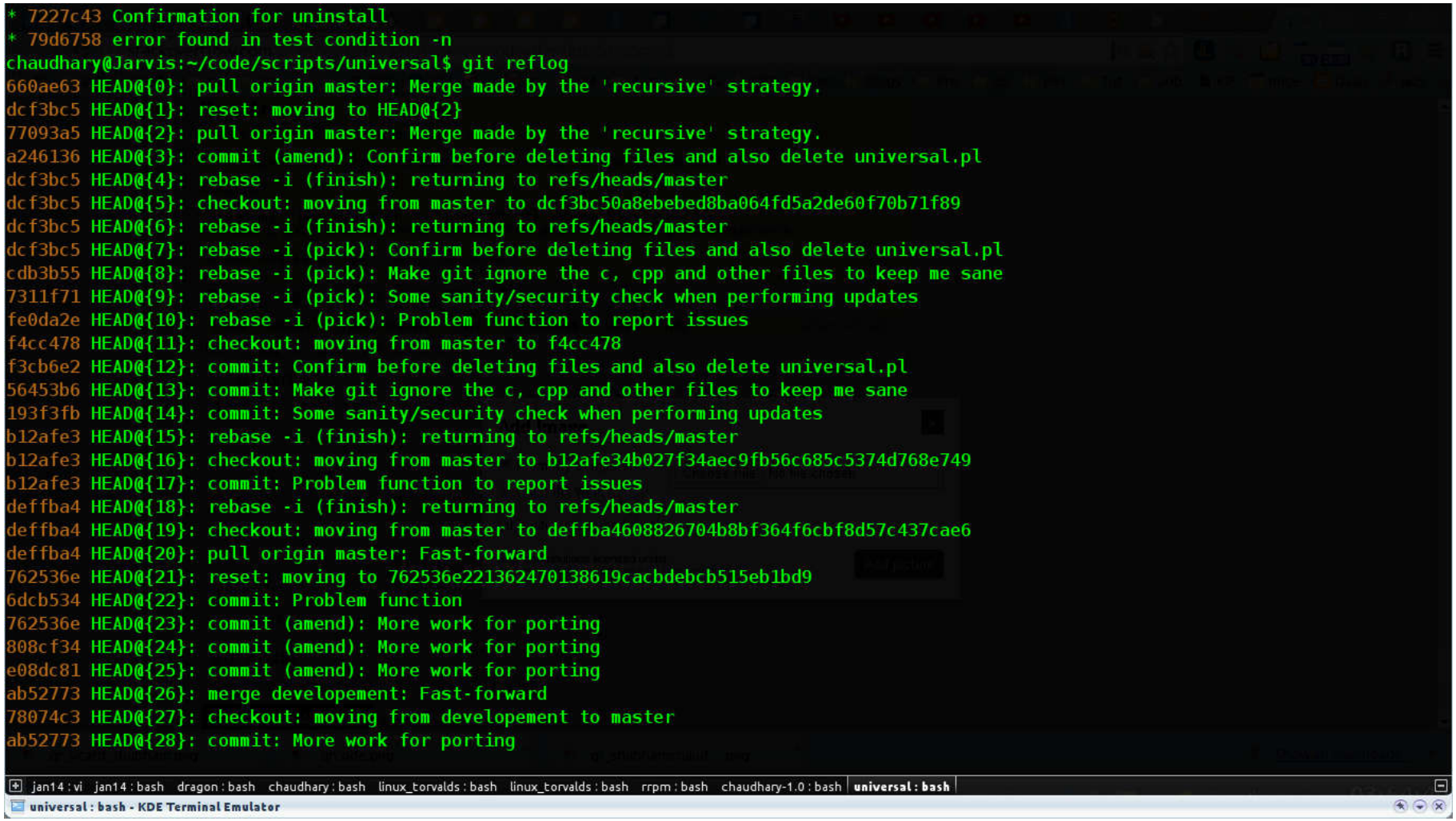
community wiki
3 revs, 3 users 74%
Ranjithkumar Ravi

---

180

Use reflog to find a correct state

```
git reflog
```

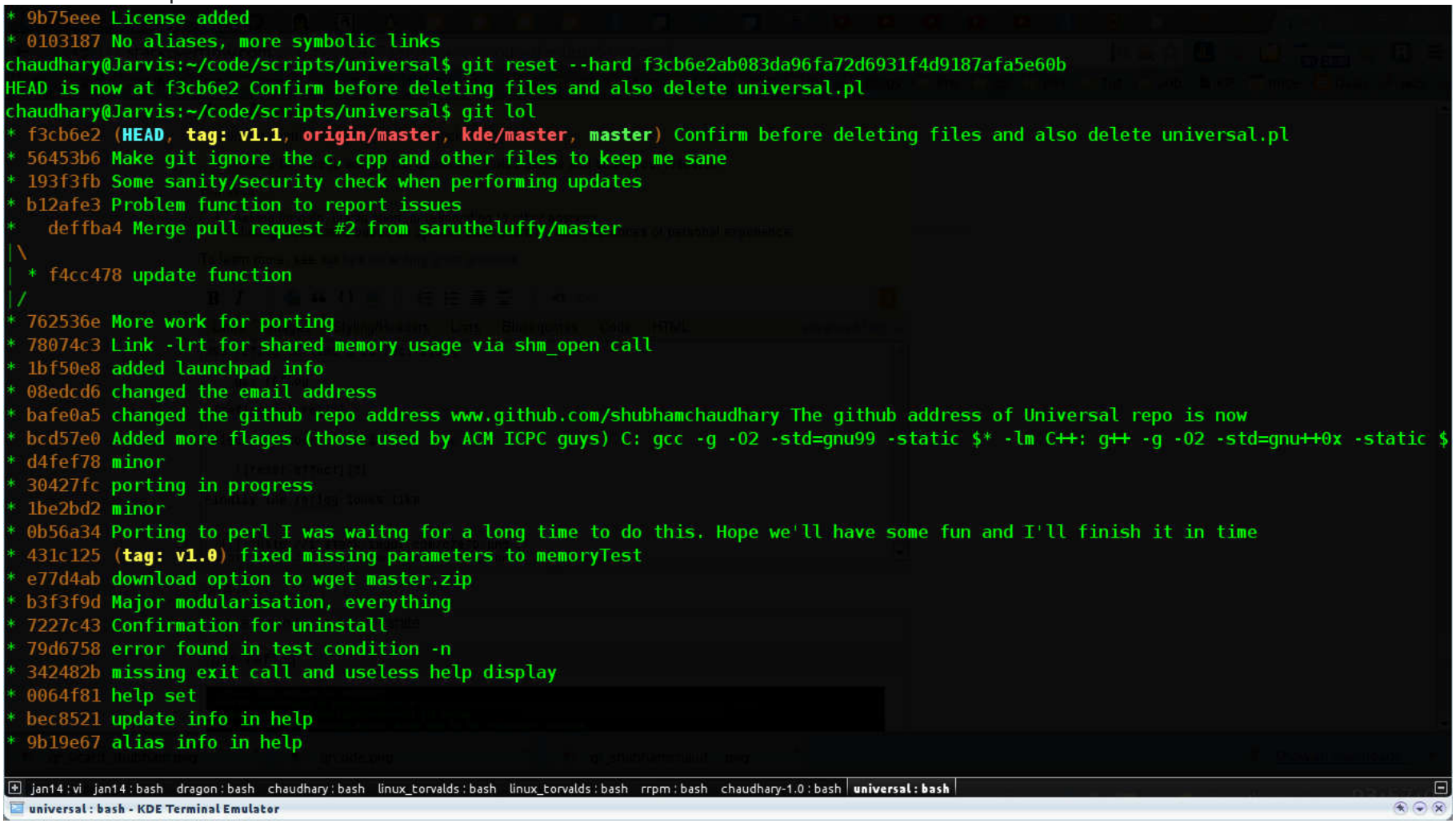



*REFLOG BEFORE RESET*

Select the correct reflog (f3cb6e2 in my case) and type

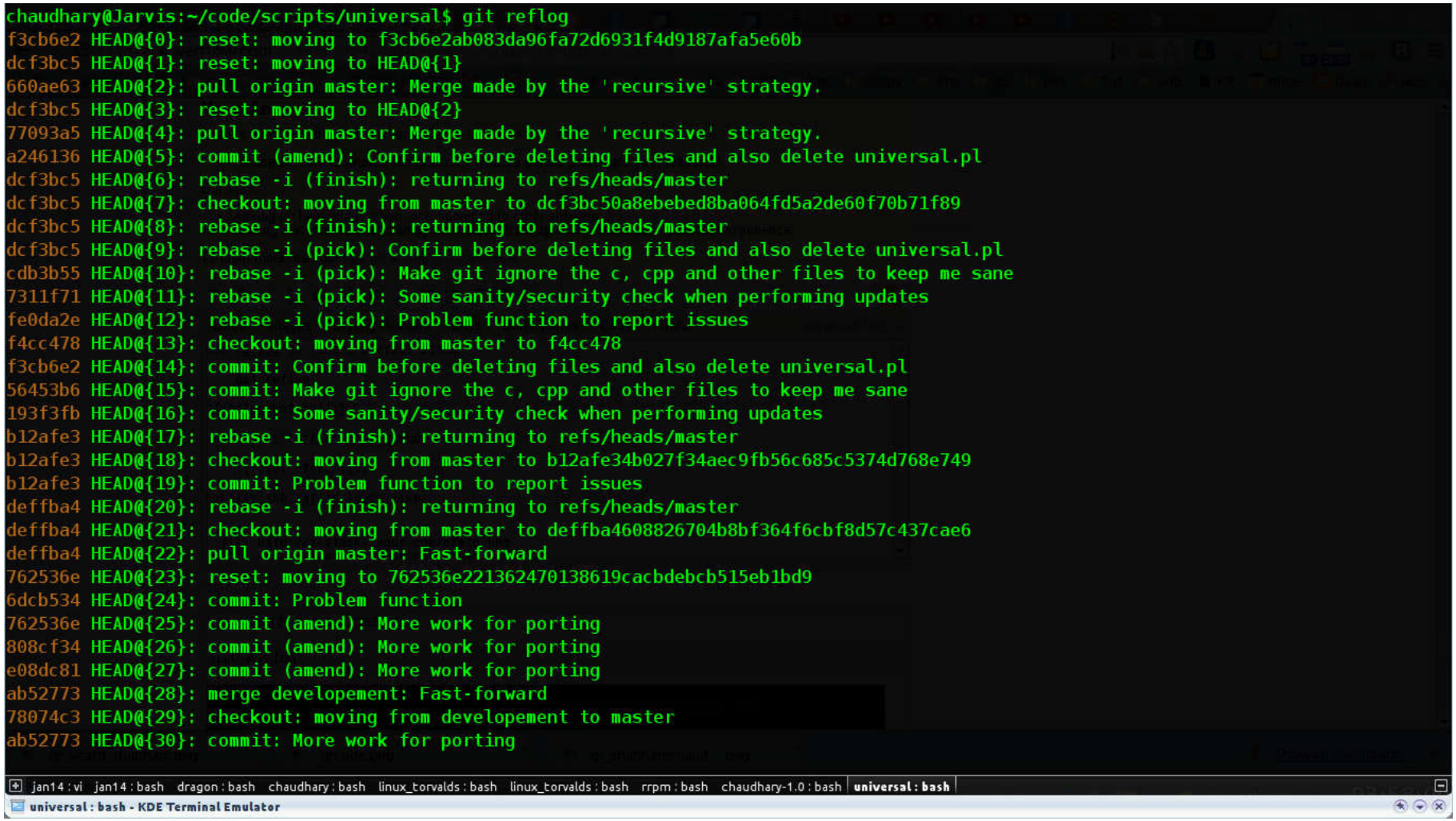
```
git reset --hard f3cb6e2
```

After that the repo HEAD will be reset to that HEADid




*LOG AFTER RESET*

Finally the reflog looks like the picture below




*REFLOG FINAL*

answered Jan 6 '14 at 22:34

community wiki
Shubham Chaudhary

"Clean unknown files from the working tree"

```
git clean
```

see - Git Quick Reference

**NOTE:** This command will delete your previous commit, so use with caution! `git reset --hard` is safer –

edited Dec 13 '17 at 22:04

community wiki
5 revs, 4 users 80%
Ravi_Parmar

---

156

First run:

```
git reflog
```

It will show you all the possible actions you have performed on your repository, for example, commit, merge, pull, etc.

Then do:

```
git reset --hard ActionIdFromRefLog
```

edited Jun 24 '15 at 9:33

community wiki
4 revs, 4 users 70%
U. Ali

---

148

## Undo last commit:

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

## Replace last commit to new commit:

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

It will replace the last commit with the new commit.

edited Dec 13 '17 at 22:05

community wiki
4 revs, 3 users 81%
akshay_rahar

---

143

Another way:

Checkout the branch you want to revert, then reset your local working copy back to the commit that you want to be the latest one on the remote server (everything after it will go bye-bye). To do this, in SourceTree I right-clicked on the and selected "Reset BRANCHNAME to this commit".

Then navigate to your repository's local directory and run this command:

```
git -c diff.mnemonicprefix=false -c core.quotepath=false push -v -f --tags
REPOSITORY_NAME BRANCHNAME:BRANCHNAME
```

This will erase all commits after the current one in your local repository but only for that one branch.

edited Jun 24 '15 at 9:34

community wiki
3 revs, 3 users 90%
CommaToast

---

Type `git log` and find the last commit hash code and then enter:

3 revs, 3 users 56%
Peter Mortensen

---

130

In my case I accidentally committed some files I did not want to. So I did the following and it worked:

```
git reset --soft HEAD^
git rm --cached [files you do not need]
git add [files you need]
git commit -c ORIG_HEAD
```

Verify the results with gitk or git log --stat

edited Jun 24 '15 at 9:34

community wiki
2 revs, 2 users 93%
egridasov

---

121

Simple, run this in your command line:

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

edited Jan 18 '16 at 21:58

community wiki
2 revs, 2 users 80%
ihue

---

119

There are many ways to do it:

Git command to undo the last commit/ previous commits:

**Warning:** Do Not use –hard if you do not know what you are doing. --hard is too **dangerous**, and it might **delete your files.**

or

```
$ git reset --hard HEAD~<n>
```

**COMMIT-ID**: ID for the commit

**n:** is number of last commits you want to revert

You can get the commit id as shown below:

```
$ **git log --oneline**
d81d3f1 function to subtract two numbers
be20eb8 function to add two numbers
bedgfgg function to mulitply two numbers
```

where **d81d3f1** and **be20eb8** are commit id.

**Now let's see some cases:**

*Suppose you want to revert the last commit 'd81d3f1'. Here are two options:*

```
$ git reset --hard d81d3f1
```

or

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

*Suppose you want to revert the commit 'be20eb8':*

```
$ git reset --hard be20eb8
```

edited Aug 19 '18 at 14:00

community wiki
5 revs, 5 users 86%
user3799762

5 `git reset --hard HEAD~1` is **too dangerous**! This will not just 'cancel last commit', but will revert repo completely back to the previous commit. So you will LOOSE all changes committed in the last commit! – Arnis Juraga Mar 21 '17 at 12:09

You right, to undo this you can use `git push -f <remote> HEAD@{1}:<branch>` – Benny Apr 24 '17 at 13:07

Unfortunately, I use –hard, and my files are deleted! I did not check the comment first because it is collapsed. Do not use –hard if you do not know what you are doing! – anonymous Aug 19 '18 at 13:53

117

## For a local commit

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

or if you do not remember exactly in which commit it is, you might use

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

## For a pushed commit

The proper way of removing files from the repository history is using `git filter-branch` . That is,

```
git filter-branch --index-filter 'git rm --cached <file>' HEAD
```

But I recomnend you use this command with care. Read more at *git-filter-branch(1) Manual Page*.

edited Jul 21 '14 at 20:15

community wiki
3 revs, 3 users 64%
geoom

117

If the problem was extra files you commited (and you don't want those on repository), you can remove them using `git rm` and then commiting with `--amend`

```
git rm <pathToFile>
```

You can also remove entire directories with `-r` , or even combine with other Bash commands

```
git rm -r <pathToDirectory>
git rm $(find -name '*.class')
```

After removing the files, you can commit, with **--amend** option

```
git commit --amend -C HEAD # the -C option is to use the same commit message
```

This will rewrite your recent local commit removing the extra files, so, these files will never be sent on push and also will be removed from your local .git repository by GC.

**You already pushed the commit**

You can apply the same solution of the other scenario and then doing `git push` with the `-f` option, but it is **not recommended** since it overwrites the remote history with a divergent change (it can mess your repository).

Instead, you have to do the commit without `--amend` (remember this about -amend`: That option rewrites the history on the last commit).

edited Jan 11 '16 at 23:50

community wiki
3 revs, 3 users 86%
dseminara

---

117

To reset to the previous revision, permanently deleting all uncommitted changes:

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

23 Maybe you could at a note/warning that his command will throw away the commit **and the changes in the working directory** without asking any further. – cr7pt0gr4ph7 Nov 24 '14 at 22:35

6 If you happen to do this by accident, not all is lost, though. See stackoverflow.com/questions/10099258/…, stackoverflow.com/questions/15479501/… and stackoverflow.com/questions/7374069/undo-git-reset-hard/7376959. – cr7pt0gr4ph7 Nov 24 '14 at 22:40

13 Use `--soft` to keep your changes as `uncommitted changes`, `--hard` to nuke the commit completely and revert back by one. Remember to do such operations only on changes, that are not pushed yet. – Yunus Nedim Mehel Mar 9 '15 at 9:11

@Zaz: You are right; maybe I should have clarified that. Only files/changes that have been either added to index (/staged) or have been committed can possibly be recovered. Uncommitted, unstaged changes *are*, as you said, completely thrown away by `git reset --hard .` – cr7pt0gr4ph7 Sep 13 '16 at 21:17

1 As a sidenote: Everytime a file is staged, `git` stores its contents in its object database. The stored contents are only removed when garbage collection is executed. It is therefore possible to recover the last staged version of a file that was not currently staged when `git reset --hard` was executed (see the posts linked above for more information). – cr7pt0gr4ph7 Sep 13 '16 at 21:22

1 2 3 next