How do I undo 'git add' before commit?

Asked 14 years, 2 months ago Modified 11 days ago Viewed 4.9m times

10890

Want to improve this post? Provide detailed answers to this question, including citations and an explanation of why your answer is correct. Answers without enough detail may be edited or deleted.

I mistakenly added files to Git using the command:

```
git add myfile.txt
```

I have not yet run `git commit`. How do I undo this so that these changes will not be included in the commit?

git undo git-add

Share Edit Follow Flag

edited Jul 25, 2022 at 2:07
Mateen Ulhaq
23.1k ● 16 ● 89 ● 132

asked Dec 7, 2008 at 21:57
oz10
150k ● 27 ● 93 ● 127

48 Starting with Git v1.8.4, all the answers below that use `HEAD` or `head` can now use `@` in place of `HEAD` instead. See this answer (last section) to learn why you can do that. – user456814 Jul 26, 2013 at 2:04

5 I made a little summery which shows all ways to unstage a file: stackoverflow.com/questions/6919121/... – Daniel Alder Apr 26, 2014 at 12:09

8 If you use Eclipse, it is as simple as unchecking the files in the commit dialogue box – Hamzahfrq Nov 17, 2016 at 12:49

2 This is a great resource straight from Github: How to undo (almost) anything with Git – jasonleonhard Feb 3, 2017 at 21:13

2 Before you post a new answer, consider there are already 25+ answers for this question. Make sure that your answer contributes what is not among existing answers – Sazzad Hissain Khan Jun 15, 2017 at 15:29

38 Answers

Sorted by: Highest score (default)

1 2 Next

12941

Undo `git add` for uncommitted changes with:

```
git reset <file>
```

That will remove the file from the current index (the "about to be committed" list) without changing anything else.

To unstage all changes for all files:

```
git reset
```

In old versions of Git, the above commands are equivalent to `git reset HEAD <file>` and `git reset HEAD` respectively, and will fail if `HEAD` is undefined (because you haven't yet made any commits in your repository) or ambiguous (because you created a branch called `HEAD`, which is a stupid thing that you shouldn't do). This was changed in Git 1.8.2, though, so in modern versions of Git you can use the commands above even prior to making your first commit:

> "git reset" (without options or parameters) used to error out when you do not have any commits in your history, but it now gives you an empty index (to match non-existent commit you are not even on).

Documentation: **git reset**

Share Edit Follow Flag

edited Jul 5, 2022 at 0:18
Mateen Ulhaq
23.1k ● 16 ● 89 ● 132

answered Dec 7, 2008 at 22:30
genehack
133k ● 1 ● 23 ● 24

---

147 Of course, this is not a true undo, because if the wrong `git add` overwrote a previous staged uncommited version, we can't recover it. I tried to clarify this in my answer below. – leonbloy May 6, 2013 at 19:10

11 `git reset HEAD *.ext` where `ext` is the files of the given extension you want to unadd. For me it was `*.bmp` & `*.zip` – boulder_ruby Nov 26, 2013 at 14:25

28 @Jonny, the index (aka staging area) contains *all* the files, not just changed files. It "starts life" (when you check out a commit or clone a repo) as a copy of all the files in the commit pointed to by HEAD. So if you *remove* a file from the index (`git rm --cached`) it means you are preparing to make a commit that *deletes* that file. `git reset HEAD <filename>` on the other hand will copy the file from HEAD to the index, so that the next commit won't show any changes being made to that file. – Wildcard Mar 16, 2016 at 12:27

22 I just discovered that there is a `git reset -p` just like `git add -p`. This is awesome! – donquixote Jul 17, 2016 at 23:23

22 You actually **can recover overwriten previously staged but uncommited changes** but not in a userfriendly way and not 100% secure (at least none I had found): goto .git/objects, search for files created at the time of `git add` you want to recover (`61/3AF3...` -> object id `613AF3...`), then `git cat-file -p <object-id>` (might be worth it to recover several hours of work but also a lesson to commit more often...) – Peter Schneider Jul 31, 2017 at 14:03

2397

You want:

```
git rm --cached <added_file_to_undo>
```

Reasoning:

When I was new to this, I first tried

```
git reset .
```

(to undo my entire initial add), only to get this (not so) helpful message:

```
fatal: Failed to resolve 'HEAD' as a valid ref.
```

It turns out that this is because the HEAD ref (branch?) doesn't exist until after the first commit. That is, you'll run into the same beginner's problem as me if your workflow, like mine, was something like:

1. cd to my great new project directory to try out Git, the new hotness
2. `git init`
3. `git add .`
4. `git status`

   ... lots of crap scrolls by ...

   => Damn, I didn't want to add all of that.
5. google "undo git add"

   => find Stack Overflow - yay
6. `git reset .`

   => fatal: Failed to resolve 'HEAD' as a valid ref.

It further turns out that there's a bug logged against the unhelpfulness of this in the mailing list.

And that the correct solution was right there in the Git status output (which, yes, I glossed over as 'crap)

> ```
> ...
> # Changes to be committed:
> #   (use "git rm --cached <file>..." to unstage)
> ...
> ```

And the solution indeed is to use `git rm --cached FILE`.

Note the warnings elsewhere here - `git rm` deletes your local working copy of the file, but *not* if you use **--cached**. Here's the result of `git help rm`:

> --cached Use this option to unstage and remove paths only from the index. Working tree files, whether modified or not, will be left.

I proceed to use

```
git rm --cached .
```

to remove everything and start again. Didn't work though, because while `add .` is recursive, turns out `rm` needs `-r` to recurse. Sigh.

```
git rm -r --cached .
```

Okay, now I'm back to where I started. Next time I'm going to use `-n` to do a dry run and see what will be added:

```
git add -n .
```

I zipped up everything to a safe place before trusting `git help rm` about the `--cached` not destroying anything (and what if I misspelled it).

Share Edit Follow Flag

edited Sep 13, 2018 at 0:05
codeforester
37.4k ● 16 ● 107 ● 132

answered Mar 25, 2009 at 16:20
Rhubarb
34.1k ● 2 ● 47 ● 35

26 Hah. I followed this same process. Except I gave up and said `rm -rf .git`, `git init` because I didn't trust `git rm --cached` to keep my working copy. It says a little for how git is still overly complex in some places. `git unstage` should just be a stock standard command, I don't care if I can add it as an alias. – Adrian Macneil Mar 29, 2011 at 3:45

6 For me git says `git reset HEAD <File>...` – drahnr Sep 12, 2012 at 6:50

25 git rm --cached <file> is actually the correct answer, if it is the initial import of <file> into the repository. If you're trying to unstage a change to the file, git reset is the correct answer. People saying

that this answer is wrong are thinking of a different question. – Barry Kelly Feb 28, 2013 at 22:14

18 This will actually work, but **only** on the first commit, where the file didn't exist before, or where the `git add` command added new files, **but not** changes to existing files. – naught101 Apr 10, 2013 at 2:33

10 just goes to show how unintuitive and convoluted git is. instead of having parallel "undo" commands, you have to find out how to undo them. Like trying to free your leg in quick sand, and then getting your arm stuck, then getting your other arm stuck... every command should be done through GUI, with dropdown menus items for the options... Think of all the UI, productivity gains we've had, but we have this mess of a retro command line interface. It's not like the git GUI programs make this any more intuitive. – ahnbizcad May 24, 2014 at 10:54

616

If you type:

```
git status
```

Git will tell you what is staged, etc., including instructions on how to unstage:

```
use "git reset HEAD <file>..." to unstage
```

I find Git does a pretty good job of nudging me to do the right thing in situations like this.

Note: Recent Git versions (1.8.4.x) have changed this message:

```
(use "git rm --cached <file>..." to unstage)
```

Share Edit Follow Flag

edited Nov 3, 2019 at 13:08
Peter Mortensen
30.9k ● 21 ● 105 ● 125

answered Dec 7, 2008 at 23:22
Paul Beckingham
14.3k ● 5 ● 33 ● 67

26 The message will be different depending on whether the `add`ed file was already being tracked (the `add` only saved a new version to the cache - here it will show your message). Elsewhere, if the file was not previously staged, it will display `use "git rm --cached <file>..." to unstage` – leonbloy May 6, 2013 at 18:25

3 My git version 2.14.3 says `git reset HEAD` to unstage. – SilverWolf Apr 23, 2018 at 19:16

8 Since Git v2.23 the message has changed yet again. It now says `git restore --staged <file>`. See my answer below for an update. – prosoitos Nov 19, 2020 at 17:02

To clarify: `git add` moves changes from the current working directory to the *staging area* (index).

297

This process is called *staging*. So the most natural command to *stage* the changes (changed files) is the obvious one:

```
git stage
```

`git add` is just an easier-to-type alias for `git stage`

Pity there is no `git unstage` nor `git unadd` commands. The relevant one is harder to guess or remember, but it is pretty obvious:

```
git reset HEAD --
```

We can easily create an alias for this:

```
git config --global alias.unadd 'reset HEAD --'
git config --global alias.unstage 'reset HEAD --'
```

And finally, we have new commands:

```
git add file1
git stage file2
git unadd file2
git unstage file1
```

Personally I use even shorter aliases:

```
git a # For staging
git u # For unstaging
```

Share Edit Follow Flag

edited Nov 3, 2019 at 13:15
Peter Mortensen
30.9k ● 21 ● 105 ● 125

answered Sep 10, 2010 at 20:28
takeshin
48.2k ● 32 ● 118 ● 162

6 "moves"? This would indicate it has gone from the working directory. That's not the case. – Thomas Weller Jun 8, 2017 at 9:18

2 I agree, it's very annoying that Linus Torvalds, instead of creating simmetric commands, just created a new word for a different command. For simmetric i mean something like: commit - uncommit; stage-unstage . Or a keyword UNDO that can be used for many commands: git commit X - git UNDO commit x. Seems natural that one has to learn by heart a lot of words. The ones that are used not so often are easily forgotten... and here we all are on this page – fresko Dec 21, 2021 at 10:27

An addition to the accepted answer, if your mistakenly-added file was huge, you'll probably notice that, even after removing it from the index with '`git reset`', it still seems to occupy space

208

in the `.git` directory.

This is nothing to be worried about; the file is indeed still in the repository, but only as a "loose object". It will not be copied to other repositories (via clone, push), and the space will be eventually reclaimed - though perhaps not very soon. If you are anxious, you can run:

```
git gc --prune=now
```

Update (what follows is my attempt to clear some confusion that can arise from the most upvoted answers):

So, which is the real **undo** of `git add`?

`git reset HEAD <file>` ?

or

`git rm --cached <file>`?

Strictly speaking, and if I'm not mistaken: **none**.

`git add` **cannot be undone** - safely, in general.

Let's recall first what `git add <file>` actually does:

1. If `<file>` was **not previously tracked**, `git add` **adds it to the cache**, with its current content.
2. If `<file>` was **already tracked**, `git add` **saves the current content** (snapshot, version) to the cache. In Git, this action is still called **add**, (not mere *update* it), because two different versions (snapshots) of a file are regarded as two different items: hence, we are indeed adding a new item to the cache, to be eventually committed later.

In light of this, the question is slightly ambiguous:

> I mistakenly added files using the command...

The OP's scenario seems to be the first one (untracked file), we want the "undo" to remove the file (not just the current contents) from the tracked items. **If** this is the case, then it's ok to run `git rm --cached <file>`.

And we could also run `git reset HEAD <file>`. This is in general preferable, because it works in both scenarios: it also does the undo when we wrongly added a version of an already tracked item.

But there are two caveats.

First: There is (as pointed out in the answer) only one scenario in which `git reset HEAD` doesn't work, but `git rm --cached` does: a new repository (no commits). But, really, this a practically irrelevant case.

Second: Be aware that `git reset HEAD` can't magically recover the previously cached file contents, it just resynchronises it from the HEAD. If our misguided `git add` overwrote a previous staged uncommitted version, we can't recover it. That's why, strictly speaking, we cannot undo [*].

Example:

```
$ git init
$ echo "version 1" > file.txt
$ git add file.txt   # First add of file.txt
$ git commit -m 'first commit'
$ echo "version 2" > file.txt
$ git add  file.txt   # Stage (don't commit) "version 2" of file.txt
$ git diff --cached file.txt
-version 1
+version 2
$ echo "version 3" > file.txt
$ git diff  file.txt
-version 2
+version 3
$ git add  file.txt    # Oops we didn't mean this
$ git reset HEAD file.txt  # Undo?
$ git diff --cached file.txt  # No dif, of course. stage == HEAD
$ git diff file.txt   # We have irrevocably lost "version 2"
-version 1
+version 3
```

Of course, this is not very critical if we just follow the usual lazy workflow of doing 'git add' only for adding new files (case 1), and we update new contents via the commit, `git commit -a` command.

---

* (Edit: the above is practically correct, but still there can be some slightly hackish/convoluted ways for recovering changes that were staged, but not committed and then overwritten - see the comments by Johannes Matokic and iolsmit)

Share Edit Follow Flag

edited Nov 3, 2019 at 13:21
Peter Mortensen
30.9k ●21 ●105 ●125

answered May 18, 2011 at 18:05
leonbloy
71.7k ●20 ●139 ●189

6 Strictly speaking there is a way to recover an already staged file that was replaced with git add. As you mention git add creates an git object for that file that will become a loose object not only when removing the file completely but also when being overwritten with new content. But there is no command to automatically recover it. Instead the file has to be identified and extracted manually or with tools written only for this case (libgit2 will allow this). But this will only pay out if the file is very important and big and could not be rebuild by editing the previous version. – Johannes Matokic Dec 6, 2017 at 13:07

4 To correct myself: Once the loose object file is found (use meta-data like creation date/time) `git cat-file` could be used to recover its content. – Johannes Matokic Dec 6, 2017 at 13:22

8 Another way to **recover changes that were staged but not committed and then overwritten** by e.g. another `git add` is via `git fsck --unreachable` that will list all unreachable obj, which you can then inspect by `git show SHA-1_ID` or `git fsck --lost-found` that will >Write dangling objects into `.git/lost-found/commit/` or `.git/lost-found/other/`, depending on type. See also `git fsck --help` – iolsmit Apr 27, 2018 at 15:29

1 @FilipSavic If you add the same file multiple times, without committing. E.g. `echo 1 > a.txt` followed by `git add a.txt` - do not commit yet - `echo 2 > a.txt` followed by `git add a.txt`; now commit e.g. `git commit -m "create unreachable blob"` and run `git fsck --unreachable` – iolsmit Oct 26, 2022 at 18:46

178

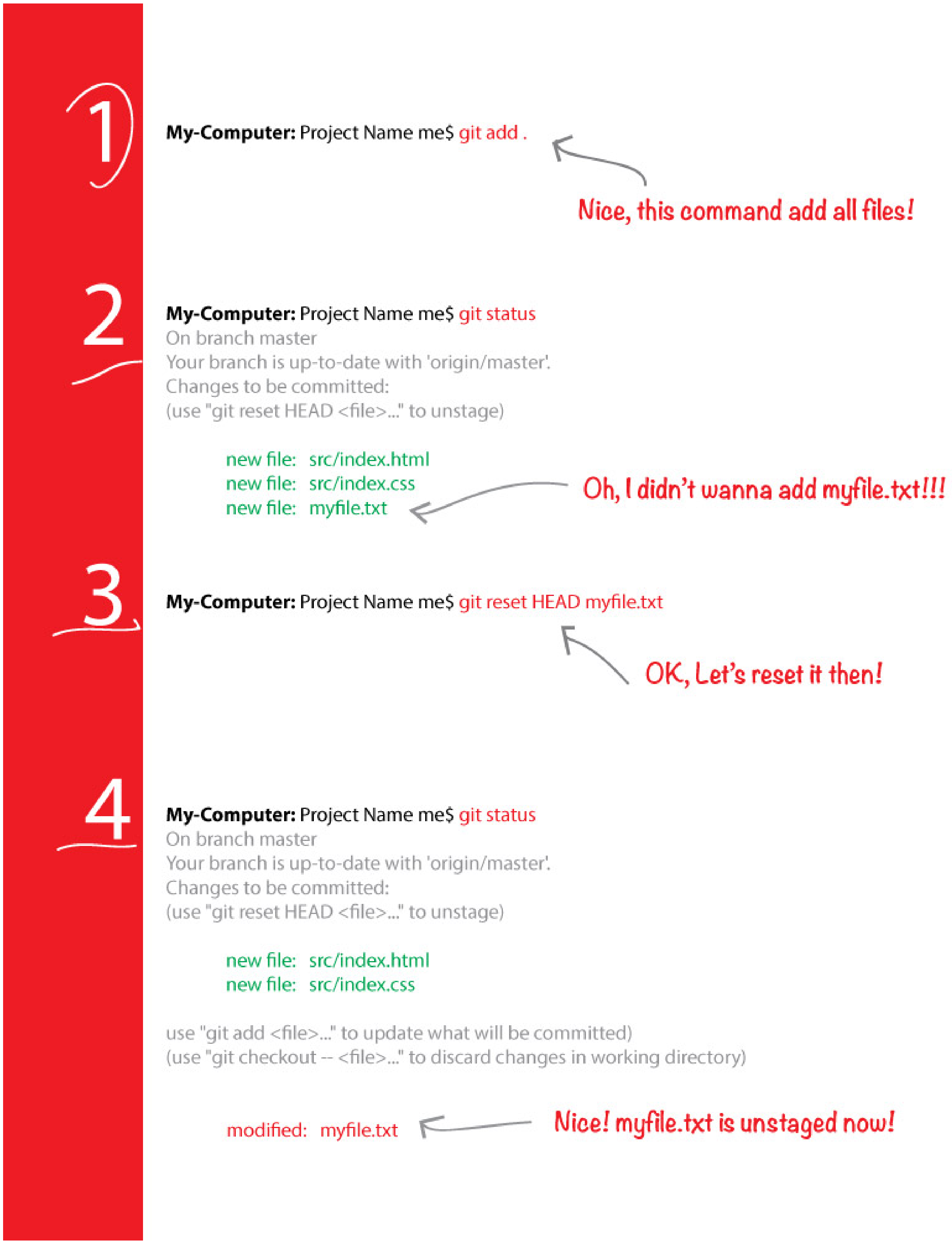
Undo a file which has already been added is quite easy using Git. For resetting `myfile.txt`, which have already been added, use:

```
git reset HEAD myfile.txt
```

Explanation:

After you staged unwanted file(s), to undo, you can do `git reset`. `Head` is head of your file in the local and the last parameter is the name of your file.

I have created the steps in the image below in more details for you, including all steps which may happen in these cases:



```
1
My-Computer: Project Name me$ git add .
```
Nice, this command add all files!

```
2
My-Computer: Project Name me$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
(use "git reset HEAD <file>..." to unstage)

        new file:   src/index.html
        new file:   src/index.css
        new file:   myfile.txt
```
Oh, I didn't wanna add myfile.txt!!!

```
3
My-Computer: Project Name me$ git reset HEAD myfile.txt
```

```
4
My-Computer: Project Name me$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
(use "git reset HEAD <file>..." to unstage)

        new file:   src/index.html
        new file:   src/index.css

use "git add <file>..." to update what will be committed)
(use "git checkout -- <file>..." to discard changes in working directory)

        modified:   myfile.txt
```

Share Edit Follow Flag

edited Nov 3, 2019 at 13:50
Peter Mortensen
30.9k ●21 ●105 ●125

answered Jun 28, 2017 at 10:43
Alireza
98k ●27 ●266 ●171

```
git rm --cached . -r
```

115

will "un-add" everything you've added from your current directory recursively

Share Edit Follow Flag

edited May 28, 2013 at 15:18
fedorqui
267k ● 101 ● 536 ● 590

answered Dec 9, 2009 at 21:19
braitsch
14.5k ● 5 ● 42 ● 35

4 I wasn't looking to un-add everything, just ONE specific file. – oz10 Dec 9, 2009 at 22:35

3 Also helpful if you don't have any previous commits. In absence of previous commit, `git reset HEAD <file>` would say `fatal: Failed to resolve 'HEAD' as a valid ref.` – Priya Ranjan Singh Jun 2, 2013 at 3:46

9 No, this *adds* a *deletion* of everything in your current directory. Very different to just unstaging changes. – Mark Amery Oct 30, 2015 at 1:33

115

Git has commands for every action imaginable, but it needs extensive knowledge to get things right and because of that it is counter-intuitive at best...

What you did before:

- Changed a file and used `git add .`, or `git add <file>`.

What you want:

- Remove the file from the index, but keep it versioned and left with uncommitted changes in working copy:

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

- Reset the file to the last state from HEAD, undoing changes and removing them from the index:

```
# Think `svn revert <file>` IIRC.
git reset HEAD <file>
git checkout <file>

# If you have a `<branch>` named like `<file>`, use:
git checkout -- <file>
```

  This is needed since `git reset --hard HEAD` won't work with single files.

- Remove `<file>` from index and versioning, keeping the un-versioned file with changes in working copy:

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

- Remove `<file>` from working copy and versioning completely:

```
git rm <file>
```

Share Edit Follow Flag

edited Nov 27, 2020 at 14:10

answered Mar 29, 2013 at 11:14
sjas
18.1k ● 12 ● 85 ● 92

1 I can't under stand the difference of 'git reset head <file>' and 'git rm --cached <file>. Could you explain it? – jeswang Aug 14, 2013 at 0:39

8 @jeswang files are either 'known' to git (changes in them are being tracked.), or they are not 'versioned'. `reset head` undoes your current changes, but the file is still being monitored by git. `rm --cached` takes the file out of versioning, so git no longer checks it for changes (and also removes eventually indexed present changes, told to git by the prior `add`), but the changed file will be kept in your working copy, that is in you file folder on the HDD. – sjas Aug 15, 2013 at 15:09

3 The difference is `git reset HEAD <file>` is temporary - the command will be applied to the next commit only, but `git rm --cached <file>` will unstage untill it gets added again with `git add <file>`. Also, `git rm --cached <file>` means if you push that branch to the remote, anyone pulling the branch will get the file ACTUALLY deleted from their folder. – DrewT Aug 10, 2014 at 19:54

1 just what i searched `git checkout -- <file>` thanx ! – Vladimir Ch Aug 10, 2021 at 18:31

104

Run

```
git gui
```

and remove all the files manually or by selecting all of them and clicking on the *unstage from commit* button.

Share Edit Follow Flag

edited Mar 9, 2013 at 11:21
Peter Mortensen
30.9k ● 21 ● 105 ● 125

answered Oct 12, 2011 at 1:12
Khaja Minhajuddin
6,573 ● 7 ● 45 ● 45

1 Yes I understand that. I only wanted to implicitly suggest that your indicate that on your answer like "You can use `git-gui` ...." :) – Alexander Suraphel Aug 1, 2014 at 16:11

1 It says, "git-gui: command not found". I'm not sure if this works. – Parinda Rajapaksha Sep 13, 2017 at 4:19

102

The question is not clearly posed. The reason is that `git add` has two meanings:

1. adding a **new file** to the staging area, then undo with `git rm --cached file`.
2. adding a **modified** file to the staging area, then undo with `git reset HEAD file`.

If in doubt, use

```
git reset HEAD file
```

Because it does the expected thing in both cases.

Warning: if you do `git rm --cached file` on a file that was **modified** (a file that existed before in the repository), then the file will be removed on `git commit`! It will still exist in your file system, but if anybody else pulls your commit, the file will be deleted from their work tree.

`git status` will tell you if the file was a **new file** or **modified**:

```
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   my_new_file.txt
    modified:   my_modified_file.txt
```

Share Edit Follow Flag

edited Nov 3, 2019 at 13:33
Peter Mortensen
30.9k ● 21 ● 105 ● 125

answered Jan 16, 2014 at 19:54
Michael_Scharf
32.2k ● 20 ● 71 ● 94

9 +1. An extraordinary number of highly-upvoted answers and comments on this page are just flat-out wrong about the behaviour of `git rm --cached somefile`. I hope this answer makes its way up the page to a prominent position where it can protect newbies from being misled by all the false claims.
– Mark Amery Oct 30, 2015 at 23:44

91

As per many of the other answers, you can use `git reset`

BUT:

I found this great little post that actually adds the Git command (well, an alias) for `git unadd`: see *git unadd* for details or..

Simply,

```
git config --global alias.unadd "reset HEAD"
```

Now you can

```
git unadd foo.txt bar.txt
```

Alternatively / directly:

```
git reset HEAD foo.txt bar.txt
```

Share Edit Follow Flag

edited Apr 17, 2022 at 3:33
tagurit
478 ● 6 ● 13

answered Oct 1, 2010 at 14:54
electblake
1,987 ● 17 ● 25

78

```
git reset filename.txt
```

will remove a file named `filename.txt` from the current index (also called the “staging area”, which is where changes “about to be committed” are saved), without changing anything else (the working directory is not overwritten).

Share Edit Follow Flag

edited Oct 11, 2021 at 12:50
Maëlan
3,343 ● 1 ● 14 ● 33

answered Jul 11, 2016 at 18:40
Rahul Sinha
1,879 ● 13 ● 17

73

If you're on your initial commit and you can't use `git reset`, just declare "Git bankruptcy" and delete the `.git` folder and start over

Share Edit Follow Flag

edited Feb 17, 2019 at 12:01
joppiesaus
5,351 ● 3 ● 25 ● 36

answered Nov 19, 2009 at 16:39
Ana Betts
73.5k ● 16 ● 140 ● 206

5 One tip is to copy your .git/config file if you have added remote origin, before deleting the folder. – Tiago Mar 8, 2010 at 23:15

4 @ChrisJohnsen comment is spot on. Sometimes, you want to commit all files except one: `git add -A && git rm --cached EXCLUDEFILE && git commit -m 'awesome commit'` (This also works when there's no previous commits, re `Failed to resolve 'HEAD'` problem) – user246672 Mar 29, 2013 at 4:20

61

2019 update

As pointed out by others in related questions (see here, here, here, here, here, here, and here), you can now **unstage a single file** with:

```
git restore --staged <file>
```

and **unstage all files** (from the root of the repo) with:

```
git restore --staged .
```

Notes

`git restore` was introduced in July 2019 and released in version 2.23.
With the `--staged` flag, it restores the content of the index (what is asked here).

When running `git status` with staged uncommitted file(s), this is now what Git suggests to use to unstage file(s) (instead of `git reset HEAD <file>` as it used to prior to v2.23).

Share Edit Follow Flag edited Feb 18, 2021 at 21:30 answered Jun 25, 2020 at 6:23

prosoitos
6,429 ● 5 ● 30 ● 41

54

Use `git add -i` to remove just-added files from your upcoming commit. Example:

Adding the file you didn't want:

```
$ git add foo
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   foo
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
# [...]#
```

Going into interactive add to undo your add (the commands typed at git here are "r" (revert), "1" (first entry in the list revert shows), 'return' to drop out of revert mode, and "q" (quit):

```
$ git add -i
           staged     unstaged path
  1:        +1/-0      nothing foo

*** Commands ***
  1: [s]tatus     2: [u]pdate     3: [r]evert     4: [a]dd untracked
  5: [p]atch      6: [d]iff       7: [q]uit       8: [h]elp
What now> r
           staged     unstaged path
  1:        +1/-0      nothing [f]oo
Revert>> 1
           staged     unstaged path
* 1:        +1/-0      nothing [f]oo
Revert>>
note: foo is untracked now.
reverted one path

*** Commands ***
```

```
  1: [s]tatus     2: [u]pdate     3: [r]evert     4: [a]dd untracked
  5: [p]atch      6: [d]iff       7: [q]uit       8: [h]elp
What now> q
Bye.
$
```

That's it! Here's your proof, showing that "foo" is back on the untracked list:

```
$ git status
# On branch master
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
# [...]
#       foo
nothing added to commit but untracked files present (use "git add" to track)
$
```

Share Edit Follow Flag

edited Dec 21, 2012 at 22:14
the Tin Man
157k ● 41 ● 211 ● 300

answered Apr 18, 2012 at 12:53
Alex North-Keys
4,140 ● 1 ● 19 ● 22

46

Here's a way to avoid this vexing problem when you start a new project:

- Create the main directory for your new project.
- Run `git init`.
- Now create a .gitignore file (even if it's empty).
- Commit your .gitignore file.

Git makes it really hard to do `git reset` if you don't have any commits. If you create a tiny initial commit just for the sake of having one, after that you can `git add -A` and `git reset` as many times as you want in order to get everything right.

Another advantage of this method is that if you run into line-ending troubles later and need to refresh all your files, it's easy:

- Check out that initial commit. This will remove all your files.
- Then check out your most recent commit again. This will retrieve fresh copies of your files, using your current line-ending settings.

Share Edit Follow Flag

edited May 10, 2012 at 18:59

answered Sep 24, 2011 at 23:34
Ryan Lundy
202k ● 37 ● 182 ● 209

1 Confirmed! Tried a git reset after a git add . and git was complaining about corrupt HEAD. Following your advice, I could git add & reset back and forth with no problems :) – Kounavi Oct 3, 2012 at 21:32

1 The second part works, but it is a bit clumsy. How line endings are handled, depends on `autocrlf` value... This won't work in every project, depending the settings. – sjas Mar 29, 2013 at 11:26

1 This answer was reasonable at the time it was posted, but is now obsolete; `git reset somefile` and `git reset` both work prior to making the first commit, now. This has been the case since several Git releases back. – Mark Amery Oct 30, 2015 at 23:38

43

Maybe Git has evolved since you posted your question.

```
$> git --version
git version 1.6.2.1
```

Now, you can try:

```
git reset HEAD .
```

This should be what you are looking for.

Share Edit Follow Flag

edited Mar 9, 2013 at 11:17
Peter Mortensen
30.9k ● 21 ● 105 ● 125

answered Nov 19, 2009 at 16:38
Kokotte23
447 ● 4 ● 2

2 Sure, but then you have the followup question of how one should unadd one of *two* (or more) files added. The "git reset" manual does mention that "git reset <paths>" is the opposite of "git add <paths>", however. – Alex North-Keys May 15, 2013 at 13:36

43

Note that if you fail to specify a revision then you have to include a separator. Example from my console:

```
git reset <path_to_file>
fatal: ambiguous argument '<path_to_file>': unknown revision or path not in the working
tree.
Use '--' to separate paths from revisions

git reset -- <path_to_file>
Unstaged changes after reset:
M    <path_to_file>
```

(Git version 1.7.5.4)

Share Edit Follow Flag

edited Nov 3, 2019 at 13:23
Peter Mortensen
30.9k ● 21 ● 105 ● 125

answered Jan 23, 2012 at 16:57
powlo
2,478 ● 3 ● 28 ● 37

2 I tried `git reset <path>` and it works just fine without a separator. I'm also using git 1.9.0. Maybe it doesn't work in older versions? – user456814 Apr 5, 2014 at 5:32

39

To remove new files from the staging area (and only in case of a new file), as suggested above:

```
git rm --cached FILE
```

Use rm --cached only for new files accidentally added.

Share Edit Follow Flag edited Jun 22, 2009 at 19:46 answered Jun 22, 2009 at 11:58

Ran
7,481 ● 12 ● 59 ● 72

4 Mind that the `--cached` is a really important part here. – takeshin Apr 12, 2013 at 12:21

1 -1; no, this doesn't un-stage the file, it stages a deletion of the file (without actually deleting it from your work tree). – Mark Amery Oct 30, 2015 at 23:42

34

You can unstage or undo using the *git* command or GUI Git.

Single file

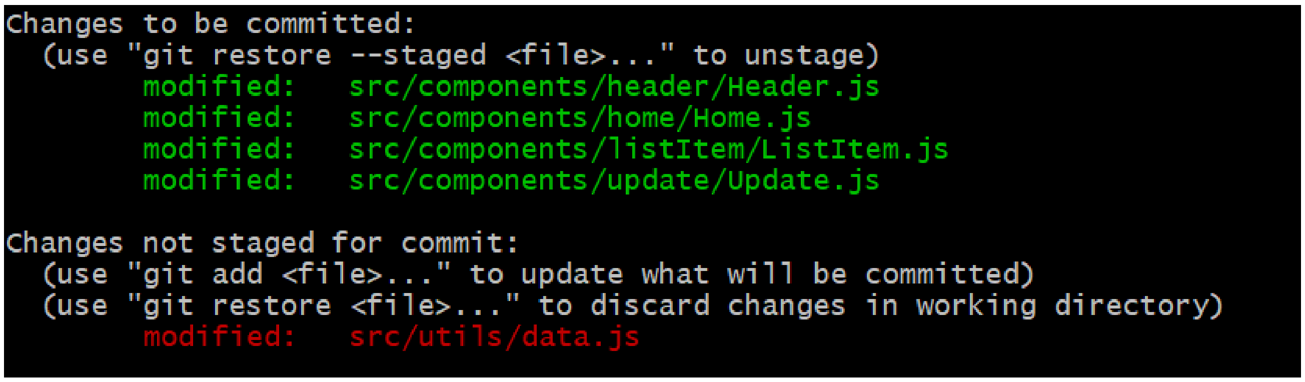
```
git reset File.txt
```

Multiple files

```
git reset File1.txt File2.txt File3.txt
```

Example

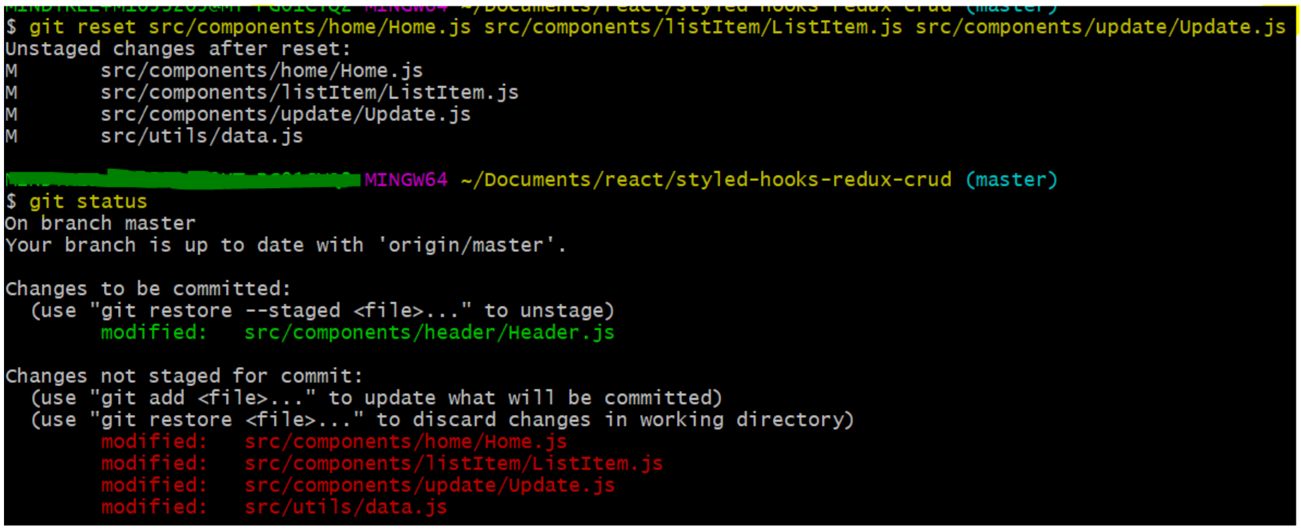
Suppose you have added **Home.js**, **ListItem.js**, **Update.js** by mistake,



```
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   src/components/header/Header.js
        modified:   src/components/home/Home.js
        modified:   src/components/listItem/ListItem.js
        modified:   src/components/update/Update.js

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:   src/utils/data.js
```

and want to undo/reset =>

```
git reset src/components/home/Home.js src/components/listItem/ListItem.js
src/components/update/Update.js
```

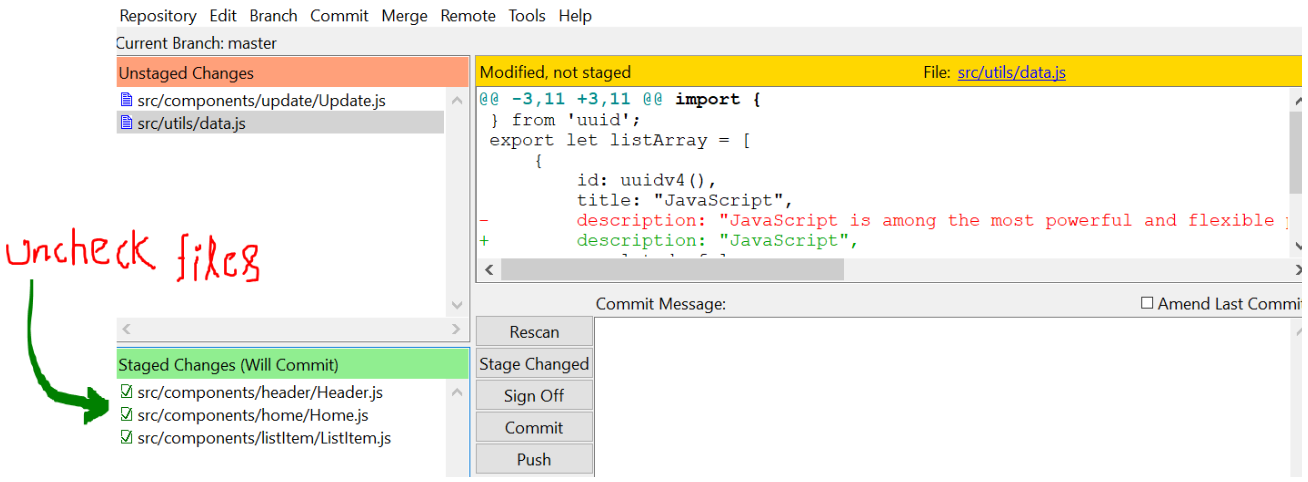


```
$ git reset src/components/home/Home.js src/components/listItem/ListItem.js src/components/update/Update.js
Unstaged changes after reset:
M       src/components/home/Home.js
M       src/components/listItem/ListItem.js
M       src/components/update/Update.js
M       src/utils/data.js

[redacted] MINGW64 ~/Documents/react/styled-hooks-redux-crud (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   src/components/header/Header.js

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:   src/components/home/Home.js
        modified:   src/components/listItem/ListItem.js
        modified:   src/components/update/Update.js
        modified:   src/utils/data.js
```

The same example using **Git GUI**

```
git gui
```

Opens a window. Uncheck your files from **Staged changes (will commit)**



Unstaged Changes:
- src/components/update/Update.js
- src/utils/data.js

```
@@ -3,11 +3,11 @@ import {
 } from 'uuid';
 export let listArray = [
     {
         id: uuidv4(),
         title: "JavaScript",
-        description: "JavaScript is among the most powerful and flexible
+        description: "JavaScript",
```

Staged Changes (Will Commit):
- ☑ src/components/header/Header.js
- ☑ src/components/home/Home.js
- ☑ src/components/listItem/ListItem.js

Share Edit Follow Flag

edited Sep 28, 2022 at 13:36
Peter Mortensen
30.9k ● 21 ● 105 ● 125

answered Sep 11, 2020 at 17:46
akhtarvahid
8,887 ● 2 ● 24 ● 28

31

To reset every file in a particular folder (and its subfolders), you can use the following command:

```
git reset *
```

Share Edit Follow Flag

answered Jul 26, 2012 at 7:50
Zorayr
23.1k ●7 ●133 ●123

5 Actually, this does not reset every file because * uses shell expansion and it ignores dotfiles (and dot-directories). – Luc May 4, 2014 at 23:20

30

Use the `*` command to handle multiple files at a time:

```
git reset HEAD *.prj
git reset HEAD *.bmp
git reset HEAD *gdb*
```

etc.

Share Edit Follow Flag

edited Nov 3, 2019 at 13:31
Peter Mortensen
30.9k ●21 ●105 ●125

answered Aug 27, 2013 at 21:15
boulder_ruby
37.5k ●9 ●74 ●97

4 Mind that * will usually not include dotfiles or 'dot-directories' unless you explicitly specify `.*` or `.*.prj` – Luc May 4, 2014 at 23:21

28

Just type `git reset` it will revert back and it is like you never typed `git add .` since your last commit. Make sure you have committed before.

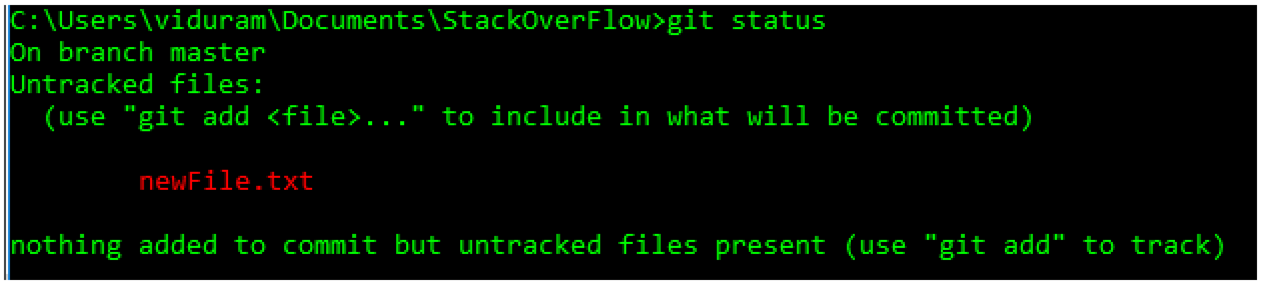
Share Edit Follow Flag

edited Apr 22, 2015 at 10:57
Piyush
3,937 ●8 ●35 ●69

answered May 19, 2010 at 3:49
Donovan
313 ●3 ●3

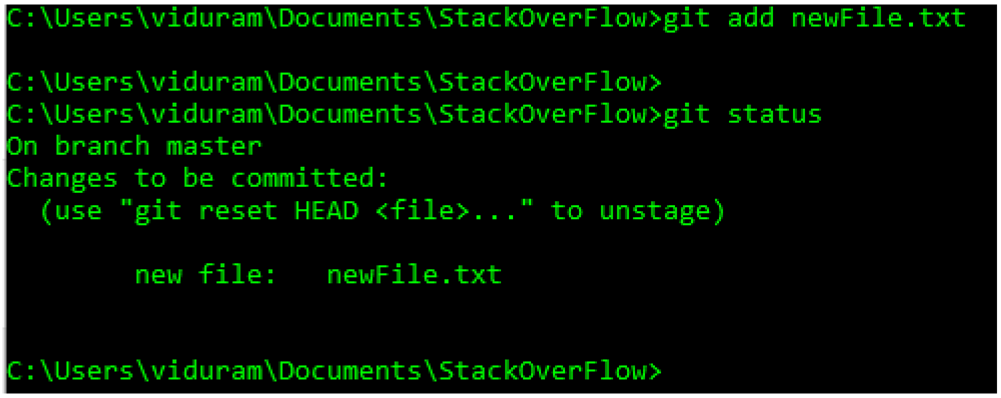
28

Suppose I create a new file, `newFile.txt`:



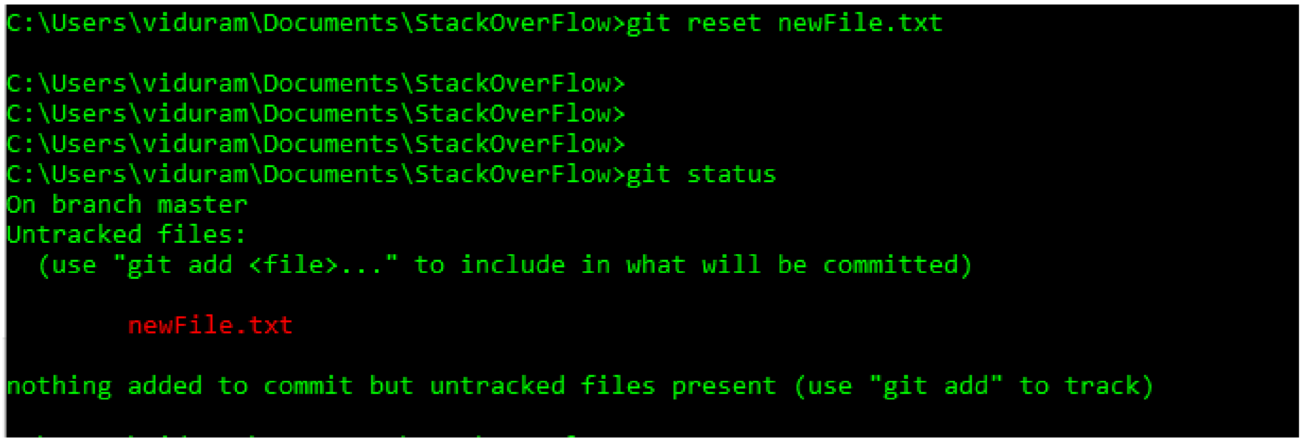
```
C:\Users\viduram\Documents\StackOverFlow>git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        newFile.txt

nothing added to commit but untracked files present (use "git add" to track)
```

Suppose I add the file accidentally, `git add newFile.txt`:



```
C:\Users\viduram\Documents\StackOverFlow>git add newFile.txt

C:\Users\viduram\Documents\StackOverFlow>
C:\Users\viduram\Documents\StackOverFlow>git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   newFile.txt


C:\Users\viduram\Documents\StackOverFlow>
```

Now I want to undo this add, before commit, `git reset newFile.txt`:



```
C:\Users\viduram\Documents\StackOverFlow>git reset newFile.txt

C:\Users\viduram\Documents\StackOverFlow>
C:\Users\viduram\Documents\StackOverFlow>
C:\Users\viduram\Documents\StackOverFlow>
C:\Users\viduram\Documents\StackOverFlow>git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        newFile.txt

nothing added to commit but untracked files present (use "git add" to track)
```

Share Edit Follow Flag

edited Nov 3, 2019 at 13:45
Peter Mortensen
30.9k ●21 ●105 ●125

answered Oct 4, 2016 at 11:02
Vidura Mudalige
810 ●2 ●18 ●30

23

For a specific file:

> - git reset my_file.txt
> - git checkout my_file.txt

For all added files:

> - git reset .
> - git checkout .

Note: **checkout** changes the code in the files and moves to the last updated (committed) state. **reset** doesn't change the codes; it just resets the header.

Share Edit Follow Flag

edited Sep 28, 2018 at 18:11

answered Oct 28, 2017 at 6:03

Jonathan Leffler
718k ● 138 ● 894 ● 1261

Hasib Kamal Chowdhury
2,436 ● 25 ● 28

4 Please explain the difference between `git reset <file>` and `git checkout <file>`. – Trent Jan 22, 2018 at 23:47

1 reset doesn't change the file, just put it away from the stage (=index, where it was put by git add) – franc Mar 12, 2018 at 11:18

1 reset = remove the file from stage however changes will still be there. checkout = gets the updated file from the repository and will overrides the current file – Imam Bux Sep 12, 2018 at 10:16

18

To undo `git add`, use:

```
git reset filename
```

Share Edit Follow Flag

edited Nov 3, 2019 at 13:43
Peter Mortensen
30.9k ● 21 ● 105 ● 125

answered Oct 2, 2016 at 15:54
Anirudh Sood
1,418 ● 12 ● 6

17

There is also interactive mode:

```
git add -i
```

Choose option 3 to un add files. In my case I often want to add more than one file, and with interactive mode you can use numbers like this to add files. This will take all but 4: 1, 2, 3, and 5

To choose a sequence, just type 1-5 to take all from 1 to 5.

Git staging files

Share Edit Follow Flag

edited Nov 3, 2019 at 13:36
Peter Mortensen
30.9k ● 21 ● 105 ● 125

answered Oct 22, 2015 at 13:03
Jonathan
369 ● 3 ● 9

16

This command will unstash your changes:

```
git reset HEAD filename.txt
```

You can also use

```
git add -p
```

to add parts of files.

Share Edit Follow Flag

edited Mar 9, 2013 at 11:22
Peter Mortensen
30.9k ● 21 ● 105 ● 125

answered Jan 31, 2013 at 15:43
wallerjake
4,079 ● 3 ● 26 ● 32

15

```
git reset filename.txt
```

> Will remove a file named filename.txt from the current index, the "about to be committed" area, without changing anything else.

Share Edit Follow Flag

answered Oct 26, 2017 at 18:15
Joseph Mathew
1,259 ● 14 ● 17

15

`git add myfile.txt` # This will add your file into the to-be-committed list

Quite opposite to this command is,

```
git reset HEAD myfile.txt  # This will undo it.
```

so, you will be in the previous state. Specified will be again in untracked list (previous state).

It will reset your head with that specified file. so, if your head doesn't have it means, it will simply reset it.

Share Edit Follow Flag

edited Nov 3, 2019 at 13:47
Peter Mortensen
30.9k ● 21 ● 105 ● 125

answered Jun 27, 2017 at 13:58
Mohideen bin Mohammed
18.1k ● 10 ● 109 ● 115

1 2 Next