How do I undo 'git add' before commit?

8199

1426

I mistakenly added files to git using the command:

```
git add myfile.txt
```

I have not yet run `git commit`. Is there a way to undo this, so these files won't be included in the commit?

git version-control git-commit git-stage

edited Jun 10 at 10:04
Zoe
15.9k 9 62 94

asked Dec 7 '08 at 21:57
paxos1977
63.6k 23 79 117

16 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 '13 at 2:04

2 I made a little summery which shows all ways to unstage a file: stackoverflow.com/questions/6919121/… – Daniel Alder Apr 26 '14 at 12:09

1 Why not git checkout? – Erik Reppen Sep 5 '16 at 14:57

8 @ErikReppen `git checkout` does not remove staged changes from the commit index. It only reverts un-staged changes to the last committed revision - which by the way is not what I want either, I want those changes, I just want them in a later commit. – paxos1977 Sep 6 '16 at 21:08

2 If you use Eclipse, it is as simple as unchecking the files in the commit dialogue box – Hamzahfrq Nov 17 '16 at 12:49

35 Answers

1 2 next

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

You can use

```
git reset
```

without any file name to unstage all due changes. This can come in handy when there are too many files to be listed one by one in a reasonable amount of time.

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 repo) 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).

edited Oct 29 '15 at 23:04

Mark Amery
68.8k 32 266 315

answered Dec 7 '08 at 22:30

genehack
96k 1 18 24

74 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 '13 at 19:10

5 `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 '13 at 14:25

14 @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 '16 at 12:27

8 I just discovered that there is a `git reset -p` just like `git add -p` . This is awesome! – donquixote Jul 17 '16 at 23:23

4 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 (`61/3AE3` -> object id `613AE3`), then

2063

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.

```
...
# 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).

edited Sep 13 '18 at 0:05

codeforester
19.7k 8 44 73

answered Mar 25 '09 at 16:20

Rhubarb
28.8k 2 35 31

13 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 '13 at 22:14

13 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 '13 at 2:33

4 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 '14 at 10:54

507

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)
```

edited Nov 9 '13 at 3:48

answered Dec 7 '08 at 23:22

Paul Beckingham

11.6k 4 26 64

15 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 '13 at 18:25

240

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

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 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
```

edited Apr 29 '13 at 13:05

answered Sep 10 '10 at 20:28

takeshin

30.9k 26 107 153

Actually, `git stage` is the alias for `git add`, which is the historic command, both on Git and other SCM. It has been added in december 2008 with commit 11920d28da in the "Git's git repository", if I can say. – Obsidian Sep 12 '18 at 18:13

161

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 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 up-voted 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 commited later.

In light of this, the question is slightly ambiguous:

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 resyncs 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 lost irrevocably "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

edited Oct 15 '18 at 17:23

answered May 18 '11 at 18:05

leonbloy

54.9k 17 110 154

3 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 '17 at 13:07

2 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 '17 at 13:22

1 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 '18 at 15:29

92

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

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

edited May 28 '13 at 15:18

fedorqui

178k 56 368 411

answered Dec 9 '09 at 21:19

braitsch

9,844 4 36 31

3 I wasn't looking to un-add everything, just ONE specific file. – paxos1977 Dec 9 '09 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 '13 at 3:46

5 No, this *adds* a *deletion* of everything in your current directory. Very different to just unstaging changes. – Mark Amery Oct 30 '15 at 1:33

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

edited Mar 9 '13 at 11:21 — Peter Mortensen — 14.2k 19 88 114

answered Oct 12 '11 at 1:12 — Khaja Minhajuddin — 5,243 5 36 44

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 '14 at 16:11

1 It says, "git-gui: command not found". I'm not sure if this works. – Parinda Rajapaksha Sep 13 '17 at 4:19

Wow, this is much simple then doing command lines which you don't understood. **This is definitely recommended for a beginner like me.** Thanks for writing this up! – Irfandy Jip Apr 11 at 4:27

85

Undo a file which already added is quite easy using **git**, for resetting `myfile.txt` which already added, use:

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

Explain:

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

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

edited Mar 21 at 9:34

answered Jun 28 '17 at 10:43

Alireza

57.5k 14 191 126

1 AMAZING!!!!!!!!!!!!!!!!!!!! – Thanveer Shah Mar 26 at 12:56

80

Git has commands for every action imaginable, but 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:

edited Oct 30 '15 at 1:37

Mark Amery
68.8k 32 266 315

answered Mar 29 '13 at 11:14

sjas
11.3k 9 62 72

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 '13 at 0:39

5 @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 '13 at 15:09

2 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 '14 at 19:54

73

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**:

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

edited Oct 30 '15 at 23:47 — Mark Amery — 68.8k 32 266 315

answered Jan 16 '14 at 19:54 — Michael_Scharf — 21.3k 12 48 78

5 +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 '15 at 23:44

one of the best answers on here, sadly it is quite low on the list – Creos Jun 10 at 1:10

62

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

edited Feb 17 at 12:01 — joppiesaus — 3,788 2 18 32

answered Nov 19 '09 at 16:39 — Paul Betts — 65.8k 15 121 192

5 One tip is to copy your .git/config file if you have added remote origin, before deleting the folder. – Tiago Mar 8 '10 at 23:15

3 @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 '13 at 4:20

55

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..

Now you can

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

edited Jul 13 '16 at 16:51

answered Oct 1 '10 at 14:54

electblake

1,579 15 23

45

`git remove` or `git rm` can be used for this, with the `--cached` flag. Try:

```
git help rm
```

edited Mar 9 '13 at 11:14

Peter Mortensen

14.2k 19 88 114

answered Dec 7 '08 at 22:00

gnud

63.8k 5 52 72

8 Isn't this going to remove the file altogether? – Willa Aug 26 '15 at 5:29

6 `git rm --cached ...` will remove files from a git repo. They'll still exist on your computer, but this is VERY different from unstaging *changes* to a file. For anyone stumbling upon this, it isn't a valid answer to the question. – Addison Dec 17 '18 at 20:02

42

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)
```

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)
$
```

37

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.

edited May 10 '12 at 18:59

answered Sep 24 '11 at 23:34

Ryan Lundy

160k 31 163 197

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 '12 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 '13 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 '15 at 23:38

@MarkAmery, you may be right (it'd be cool if you posted a source for your assertion), but there's still value in starting your repo with a clean commit or two. – Ryan Lundy Oct 31 '15 at 20:01

Maybe Git has evolved since you posted your question.

Now, you can try:

```
git reset HEAD .
```

This should be what you are looking for.

edited Mar 9 '13 at 11:17

Peter Mortensen
14.2k 19 88 114

answered Nov 19 '09 at 16:38

Kokotte23
347 3 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 '13 at 13:36

33

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)

edited Apr 5 '14 at 5:32
user456814

answered Jan 23 '12 at 16:57

powlo
1,265 1 16 29

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 '14 at 5:32

30

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

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

edited Jun 22 '09 at 19:46

answered Jun 22 '09 at 11:58

Ran

4,977 12 51 69

3 Mind that the `--cached` is a really important part here. – takeshin Apr 12 '13 at 12:21

-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 '15 at 23:42

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

24

```
git reset *
```

answered Jul 26 '12 at 7:50

Zorayr

15.4k 2 95 86

4 Actually, this does not reset every file because * uses shell expansion and it ignores dotfiles (and dot-directories). – Luc May 4 '14 at 23:20

You can run `git status` to see anything remaining and reset it manually i.e. `git reset file .` – Zorayr May 7 '14 at 15:23

use the `*` command to handle multiple files at a time

24

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

boulder_ruby
27.8k 6 59 76

2 Mind that * will usually not include dotfiles or 'dot-directories' unless you explicitly specify `.*` or `.*.prj` – Luc May 4 '14 at 23:21

22

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.

edited Apr 22 '15 at 10:57

Piyush
2,077 7 25 58

answered May 19 '10 at 3:49

Donovan
245 2 3

1 Won't work if there's no last commit. – davidA Apr 11 '12 at 22:07

As it happens, there was a last commit... but I was specifically asking about removing a single file from the commit, not every file from the commit. – paxos1977 Jan 31 '13 at 16:21

18

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 accidently, `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)
```

answered Oct 4 '16 at 11:02

Vidura Mudalige

595 2 11 23

Suppose I am at 1st pic meaning meaning I have not even did "git.add". Also, I not at all want all this change. I mean when I do git status, it should not show any red files. I mean it should be in sync as if there was not a single file altered since the last git push. how to achieve that. – Unbreakable Mar 26 '17 at 0:59

directories as well. – Vidura Mudalige Mar 26 '17 at 10:19

If you don't want to delete the untracked files, just ignore "-f" flag. – Vidura Mudalige Mar 26 '17 at 10:20

17

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**.**

edited Sep 28 '18 at 18:11

Jonathan Leffler

584k 96 704 1055

answered Oct 28 '17 at 6:03

Hasib Kamal

1,006 9 21

3 Please explain the difference between `git reset <file>` and `git checkout <file>` . – Trent Jan 22 '18 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 '18 at 11:18

checkout change the codes in file and move to the last updated state. reset doesn't change the codes it just reset the header. As example, reset use for added or committed files resetting before push and checkout use for back to the last updated/committed stage before git add. – Hasib Kamal Mar 14 '18 at 11:38

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 '18 at 10:16

You can also use

```
git add -p
```

to add parts of files.

edited Mar 9 '13 at 11:22

Peter Mortensen
14.2k 19 88 114

answered Jan 31 '13 at 15:43

wallerjake
3,056 2 17 29

13

I'm surprised that no one mention interactive mode:

```
git add -i
```

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

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

Git staging files

edited Oct 26 '15 at 12:20

answered Oct 22 '15 at 13:03

Jonathan
189 2 9

"I'm surprised that no one mention interactive mode" - they did: stackoverflow.com/a/10209776/1709587 – Mark Amery Oct 30 '15 at 23:52

To undo git add use

9

```
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.

answered Jul 11 '16 at 18:40

Rahul Sinha
641 7 13

9

```
git add myfile.txt
```
this will add your file into to be committed list

Quite opposite to this command is,

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

so, you will be in 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

answered Jun 27 '17 at 13:58

Mohideen bin Mohammed
8,481 4 55 66

8

In SourceTree you can do this easily via the gui. You can check which command sourcetree uses to unstage a file.

I created a new file and added it to git. Then I unstaged it using the SourceTree gui. This is the result:

> Unstaging files [08/12/15 10:43]

answered Dec 8 '15 at 9:58

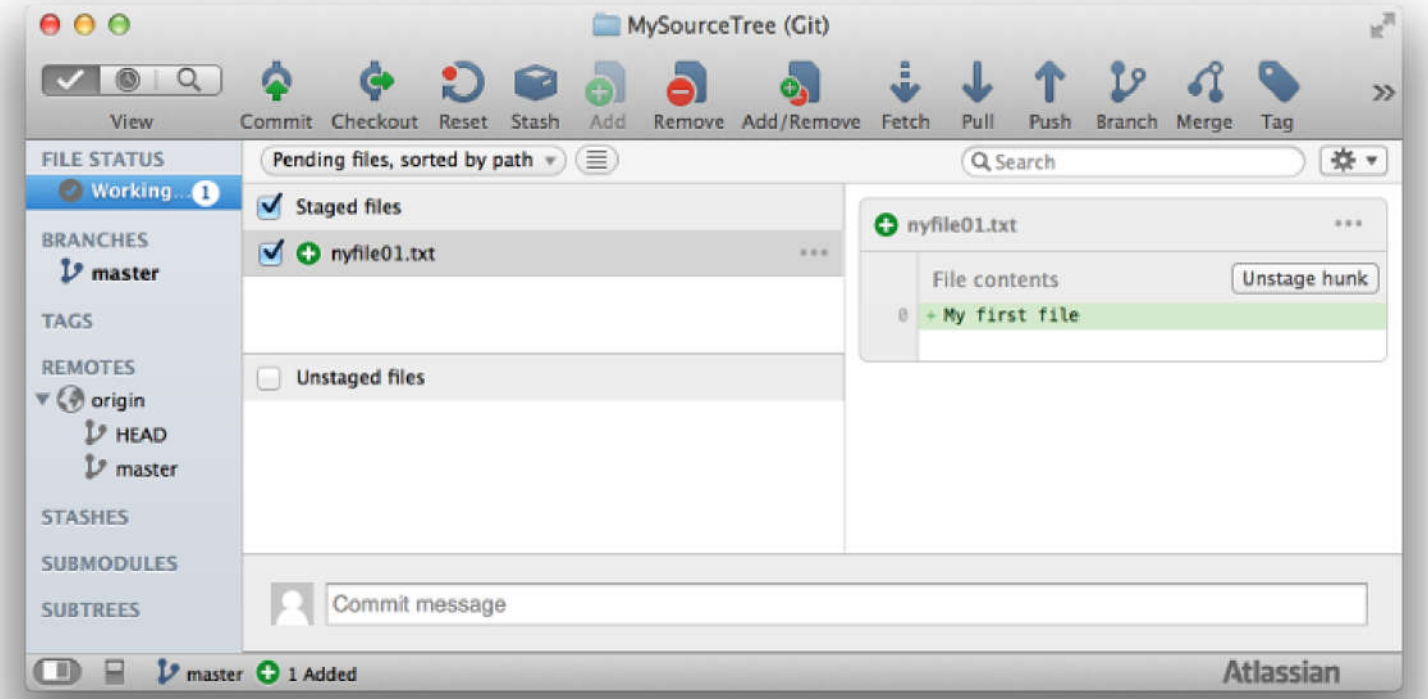
miva2

1,779 16 29

7

One of the most intuitive solutions is using SourceTree.

You can just drag and drop files from staged and unstaged



answered May 26 '17 at 8:32

1 2 next

protected by paxos1977 Jan 31 '13 at 16:20

Thank you for your interest in this question. Because it has attracted low-quality or spam answers that had to be removed, posting an answer now requires 10 reputation on this site (the association bonus does not count).

Would you like to answer one of these unanswered questions instead?