git rm * doesn't remove all files in one go

Asked 8 years, 4 months ago Active 4 years, 1 month ago Viewed 30k times

8

3

I was trying out some sample instructions of git and came across this peculiar case that when we do a `git rm *`, it doesn't delete the `.*` files in the first attempt. Why is it so?

```
mkdir -p test_repo1/folder && cd test_repo1
touch .testfile1 .testfile2 testfile3 folder/testfile4 folder/.testfile5
git init && git add . && git commit -m "test commit"
```

If now I do a `git rm` as follows, I have to do it a second time to remove all the files

```
$ git rm -r *
rm 'folder/.testfile5'
rm 'folder/testfile4'
rm 'testfile3'

$ git rm -r *
rm '.testfile1'
rm '.testfile2'
```

Why doesn't git remove all files in the first attempt itself? Also, why is this happening for files withing the repo root only?

Interestingly, if all I have are those `.testfiles`, then git removes them in the first attempt itself.

I am using git version `1.7.9.5`

git

Share Follow

asked Oct 14 2013 at 15:31

Anshul Goyal
65.7k 34 140 171

6 the * will be interpreted by your shell, Check its globbing rules – exussum Oct 14 2013 at 15:37

control x will expand the * so you can see it – exussum Oct 14 2013 at 15:37

1 stackoverflow.com/a/1586569/1301972 – Todd A. Jacobs Oct 14 2013 at 15:38

2 None of the answers mention the solution: If you want the `*` to be interpreted by git, you need to quote or escape it: `git rm -r '*'` – Keith Thompson Oct 14 2013 at 18:09

4 Answers

Active | Oldest | Score

25

The wildcard gets expanded by your shell, and the expansion does not include dot files, by default, in most shells.

So by the time `git` executes, the first command has become

```
git rm -r folder testfile3
```

and the second probably a literal

```
git rm -r *
```

which `git` then expands by itself.

As remarked by Keith, to remove everything in one go, prevent the shell from expanding the wildcard so that `git` does the expansion the first time already. This can be done with double or single quotes, or with a backslash before the asterisk. I tend to prefer single quotes:

```
git rm -r '*'
```

Share Follow

edited Jan 21 2017 at 12:38

answered Oct 14 2013 at 15:37

tripleee
153k 26 227 284

6

Shell Globbing

When you run `git rm *` you are actually using the shell's globbing rules to pass arguments to git-rm(1). Many shells don't include hidden files (e.g. files with leading dots) by default. In Bash,

you can change the globbing rules with dotglob and GLOBIGNORE. For example:

- Using the *shopt* builtin:

```
# set dotglob
shopt -s dotglob

git rm *

# unset dotglob
shopt -u dotglob
```

- Using the *GLOBIGNORE* variable (which also sets *dotglob*):

```
# Run git within a modified environment that includes GLOBIGNORE.
GLOBIGNORE='.git' git rm *
```

Share Follow

answered Oct 14 2013 at 17:00

Todd A. Jacobs
75.2k 14 133 185

4

As user1281385 noted in a comment, the shell expands `*` the first time around.

The reason the *second* time removes the dot-files is that once there are no files to match, the shell leaves the literal asterisk as an argument (depending on your shell anyway, at least one variant errors out instead) and git then does its own matching.

Share Follow

answered Oct 14 2013 at 15:39

torek
373k 47 492 627

0

Note that the right syntax should be `git rm -r .` ('dot')

More worrying is `git rm -r` (empty pathspec string) which does the same until Git 2.11!

See commit d426430 (22 Jun 2016) by Emily Xie (`emilyxxie`).
(Merged by Junio C Hamano -- `gitster` -- in commit 3b1e135, 26 Oct 2016)

> **`pathspec` : warn on empty strings as pathspec**
>
> An empty string as a pathspec element matches all paths.
> A buggy script, however, could accidentally assign an empty string to a variable that then gets passed to a Git command invocation, e.g.:

```
path=... compute a path to be removed in $path ...
git rm -r "$paht"
```

> which would unintentionally remove all paths in the current directory.
>
> The fix for this issue requires a two-step approach.
>
> - As there may be existing scripts that knowingly use empty strings in this manner, the first step simply gives a warning that (1) tells that an empty string will become an invalid pathspec element and (2) asks the user to use " . " if they mean to match all.
> - For step two, a follow-up patch several release cycles later will remove the warning and throw an error instead.

Update for Git 2.15.x/2.16 (Q1 2018):

The message "will be made invalid in upcoming releases" disappear, and becomes:

```
empty string is not a valid pathspec.
please use . instead if you meant to match all paths
```

See commit 9e4e8a6 (07 Jun 2017) by Emily Xie (`emilyxxie`).
See commit 229a95a (23 Jun 2017) by Junio C Hamano (`gitster`).
(Merged by Junio C Hamano -- `gitster` -- in commit 728c573, 06 Nov 2017)

> An empty string as a pathspec element matches all paths.
> A buggy script, however, could accidentally assign an empty string to a variable that then gets passed to a Git command invocation, e.g.:

```
path=... compute a path to be removed in $path ...
git rm -r "$path"
```

> which would unintentionally remove all paths in the current directory.

Share Follow

edited Nov 7 2017 at 20:29

answered Oct 28 2016 at 21:07

VonC
1.1m 467 3938 4627