Why would one use "git merge -s ours"?

Asked 8 years, 9 months ago Active 1 month ago Viewed 9k times

16

6

I understand that the "ours" merge strategy (hear the quotes around *merge*?) actually does not use any commits from the other branch.

The "ours" strategy is sometimes mentioned as "keeping the other's history". But it records that history as "applied". Weird. **Are there cases where I do want this behaviour?**

As an alternative, you could just as well import the other branch and let its history there, where it belongs.

Note that I am asking about the "-s ours" strategy, not the "-X ours" option to "-s recursive" (which is different in that it applies those commits that don't conflict).

git merge

edited Feb 22 '11 at 13:31
Paŭlo Ebermann
64.2k 14 127 189

asked Feb 22 '11 at 11:49
Prestel Nué
641 6 12

Although very late: Have a look here – Christoph Jan 11 '17 at 14:16

6 Answers

12

One use is to "skip" a commit made on a maintenance branch that isn't intended to go back into your main/trunk branch of development. See this previous question for an example: git - skipping specific commits when merging

edited May 23 '17 at 12:24
Community ♦

answered Feb 22 '11 at 13:48
araqnid

✓

0

In short: **common ancestor**.

Detail

Whenever you do a merge, `git` will find a common ancestor of the current branch and the branch to be merged. Then `git` merges **commits after the common ancestor** from the other branch into current branch.

`git merge -s ours` *ignores any content* from the other branch entirely. It's just creating a new common ancestor.

It **changes the commit range to be merged** for the future merges.

Here's a use case in the "Advanced Merging" chapter from the book *Pro Git*

> For example, say you branched off a `release` branch and have done some work on it that you will want to merge back into your `master` branch at some point. In the meantime some bugfix on `master` needs to be **backported** into your `release` branch. You can merge the bugfix branch into the `release` branch and also `merge -s ours` the same branch into your `master` branch (even though the fix is already there) so when you later merge the `release` branch again, there are no conflicts from the bugfix.

In this example, `git merge -s ours` is used to *skip commits* related to bugfix commits merged in `release` branch.

edited Sep 29 at 15:03

answered Sep 29 at 14:58
Simba
3,089 11 19

0

One use for this is as a preparation for another merge.

Lets say you have some downstream changes that you want to merge with a new upstream version, but the new upstream version is not a git descendent of the old upstream version (maybe because the old upstream version was a stable branch that had diverged from trunk, maybe because the upstream versions were actually the results of an import tool).

What you can do is.

1. Check out the new upstream version.

3. Merge the new downstream version.

In this way your local changes will be merged, but git will not try to merge the old upstream changes because it considers them to be already merged.

answered Jun 7 '18 at 15:47
plugwash
5,337 1 18 31

6

Another reason is when you have a branch that is very out of sync with your active branch, and you wish to update the old one with the active one, but possible merge conflicts prevent this from being done easily with a normal merge.

For example, my use case was:

You have a master and dev branch, and for some reason, you have not updated your master branch in weeks, and your attempting to merge your dev branch into master now results in merge conflicts. What you may want to do is overwrite your master branch with the current state of your dev branch, so bring them both in sync.

What you can do in this case is as follows, using steps which have outlined by another SO user in another SO answer here. **But please see my warning note below**.

```
git checkout dev
git merge -s ours master
git checkout master
git merge dev
```

Worth a quick note: At the end of it, my master was up to date with my dev, but dev showed 4 commits to be pushed to the remote, which is strange. There should be 0 to push, as I just wanted to update master. Nevertheless, the code seemed fine. Just worth noting, as I had never used `git merge -s ours` myself before this so im not 100% on its usage.

I also found another answer using `ours` which could be useful to people: https://stackoverflow.com/a/13307342/339803

edited May 23 '17 at 12:01
Community ♦
1 1

answered Apr 28 '16 at 5:23
redfox05
1,674 1 18 28

1

Another use is as an alternative to a force-push that doesn't cause non-fast-forward changes for others.

E.g. you want to revert an incoming (stupid) commit, but you don't want to do `git push -f` because then you'll spend the next half-hour guiding your client through recovering from that, and you want a shorter alternative to

```
git checkout -b temp remote/origin
git revert HEAD
git push temp:origin
git checkout master
git merge origin/master
```

so you simply do

```
git merge -s ours origin/master
```

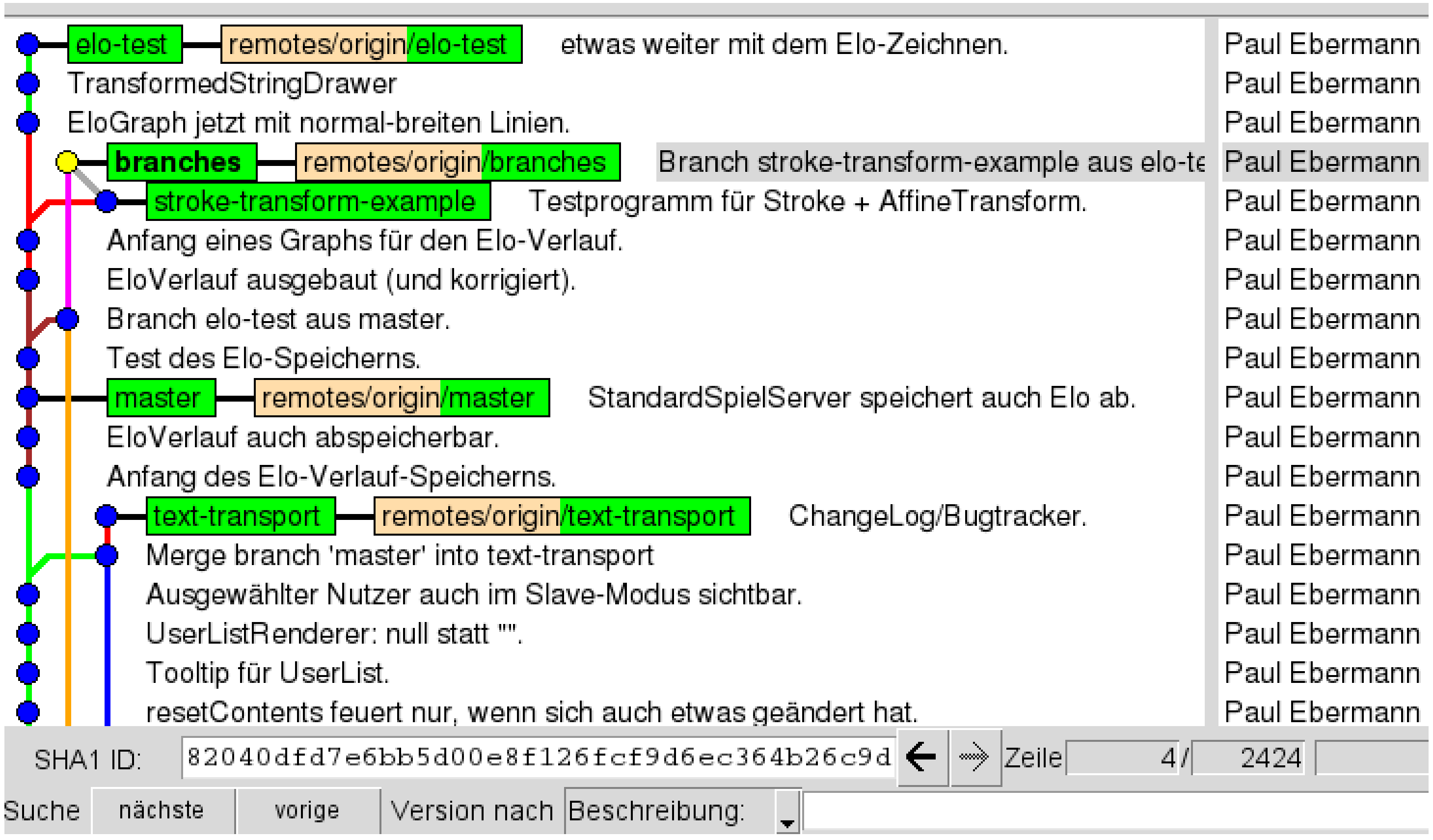
answered Oct 27 '14 at 10:00

Andrey Tarantsov
8,104 7 48 56

2

I'm using this pseudo-merge in an additional branch used to track the other branches, including abandoned ones. This additional branch has in fact only one file (called `branches-list`), which contains a text list of the active and inactive branches, but at the important points (mostly branch-points and "end-of-branch", either merge or abandon) the development branches are merged (with "-s ours") to this branch.

Here a recent screenshot of our project:



```
etwas weiter mit dem Elo-Zeichnen.                          Paul Ebermann
TransformedStringDrawer                                     Paul Ebermann
EloGraph jetzt mit normal-breiten Linien.                   Paul Ebermann
Branch stroke-transform-example aus elo-te                  Paul Ebermann
Testprogramm für Stroke + AffineTransform.                  Paul Ebermann
Anfang eines Graphs für den Elo-Verlauf.                    Paul Ebermann
EloVerlauf ausgebaut (und korrigiert).                      Paul Ebermann
Branch elo-test aus master.                                 Paul Ebermann
Test des Elo-Speicherns.                                    Paul Ebermann
StandardSpielServer speichert auch Elo ab.                  Paul Ebermann
EloVerlauf auch abspeicherbar.                              Paul Ebermann
Anfang des Elo-Verlauf-Speicherns.                          Paul Ebermann
ChangeLog/Bugtracker.                                       Paul Ebermann
Merge branch 'master' into text-transport                   Paul Ebermann
Ausgewählter Nutzer auch im Slave-Modus sichtbar.           Paul Ebermann
UserListRenderer: null statt "".                            Paul Ebermann
Tooltip für UserList.                                       Paul Ebermann
resetContents feuert nur, wenn sich auch etwas geändert hat. Paul Ebermann
SHA1 ID: 82040dfd7e6bb5d00e8f126fcf9d6ec364b26c9d   Zeile 4 / 2424
```

Thus, I lastly branched `elo-test` from master, and then from this I branched `stroke-transform-example` for my question here (and the answer - this should have been two different commits, though). At each such branch-point, and more importantly, at each point when a branch ends without it being merged to some other branch (which will happen to `stroke-transform-example` when I next clean up), I now merge this branch with `-s ours` to the meta-branch `branches`.

Then I can later delete the not-used-anymore branches without losing their history, so the output of `git branch -a` is always quite small. If I then ever want to look back at what I did then, I can easily find them.

(I think this is not really what this option is made for, but it works quite nice.)

edited May 23 '17 at 12:33
Community ♦
1 1

answered Feb 22 '11 at 13:30
Paŭlo Ebermann
64.2k 14 127 189

getting this right? – Prestel Nué Feb 22 '11 at 14:41

btw&ot, I think I will try this link occasionally (with paper&pencil)! – Prestel Nué Feb 22 '11 at 14:43

1 Yes, you are right. In my case I'm using these pseudo merges only on the `branches` branch, so it is easy to see. -- This game is much easier to play with computer support, since you shouldn't see the hidden points of your contrahent. Feel free to look at our program, it is free to play online :-) – Paŭlo Ebermann Feb 22 '11 at 14:47