How do I change the author and committer name/email for multiple commits?

Asked 15 years, 8 months ago Modified 28 days ago Viewed 1.2m times

3142

How do I change the author for a range of commits?

git version-control git-filter-branch git-rewrite-history

Share Improve this question Follow

edited Jul 11, 2022 at 6:58
Mateen Ulhaq
27.1k ● 21 ● 117 ● 152

asked Apr 15, 2009 at 3:09
Flávio Amieiro
44k ● 8 ● 33 ● 24

18 Question: does using git filter-branch preserve the SHA1's for previous tags, versions and objects? Or will changing the author name force change the associated SHA1's as well? – AndyL Aug 3, 2010 at 14:13

50 Hashes will change yes – Not Available Oct 14, 2010 at 15:16

3 Tangentially, I created a small script which finally fixed the root cause for me. gist.github.com/tripleee/16767aa4137706fd896c – tripleee May 30, 2014 at 8:51

2 @impinball The age of the question is hardly relevant. Creating a new duplicate question is out of the question. I suppose I could create a question which begs this particular answer but I'm not altogether convinced it would get all that much visibility. It's not like there is a shortage of Git questions here... Glad I could help, anyway. – tripleee Sep 1, 2014 at 14:50

3 The github script that @TimurBernikovich mentioned is great and works for me. But that github url has changed: docs.github.com/en/enterprise/2.17/user/github/using-git/... – Kaiwen Sun Oct 13, 2020 at 4:14

46 Answers

Sorted by: Highest score (default)

1 2 Next

2350

NOTE: This answer changes SHA1s, so take care when using it on a branch that has already been pushed. If you only want to fix the spelling of a name or update an old email, Git lets you do this without rewriting history using `.mailmap`. See my other answer.

Using Rebase

First, if you haven't already done so, you will likely want to fix your name in git-config:

```
git config --global user.name "New Author Name"
git config --global user.email "<email@address.example>"
```

This is optional, but it will also make sure to reset the committer name, too, assuming that's what you need.

To rewrite metadata for a range of commits using a rebase, do

```
git rebase -r <some commit before all of your bad commits> \
    --exec 'git commit --amend --no-edit --reset-author'
```

`--exec` will run the `git commit` step after each commit is rewritten (as if you ran `git commit && git rebase --continue` repeatedly).

If you also want to change your first commit (also called the 'root' commit), you will have to add `--root` to the rebase call.

This will change both the committer and the author to your `user.name` / `user.email` configuration. If you did not want to change that config, you can use `--author "New Author Name <email@address.example>"` instead of `--reset-author`. Note that doing so will *not* update the committer -- just the author.

Single Commit

If you just want to change the most recent commit, a rebase is not necessary. Just amend the commit:

```
git commit --amend --no-edit --reset-author
```

Entire project history

```
git rebase -r --root --exec "git commit --amend --no-edit --reset-author"
```

For older Git clients (pre-July 2020)

`-r,--rebase-merges` may not exist for you. As a replacement, you can use `-p`. Note that `-p` has serious issues and is now deprecated.

Share Improve this answer Follow

edited Apr 9 at 21:01
Josh Correia
4,308 ●3 ●40 ●61

answered Aug 24, 2009 at 3:08
asmeurer
91.2k ●29 ●180 ●253

33 Great for the odd commit though - useful if you're pairing and forget to change the author – mloughran Sep 25, 2009 at 11:14

39 +1 for mentioning the usecase for the typical one-mistake fix: git commit --amend --author=username – Nathan Kidd Mar 15, 2010 at 20:03

19 This is perfect, my most common usecase is that I sit down at another computer and forget to set up author and thus usually have < 5 commits or so to fix. – Zitrax Aug 21, 2010 at 11:34

74 `git commit --amend --reset-author` also works once `user.name` and `user.email` are configured correctly. – pts Jul 4, 2014 at 16:56

30 Rewrite author info on all commits after `<commit>` using `user.name` and `user.email` from `~/.gitconfig`: run `git rebase -i <commit> --exec 'git commit --amend --reset-author --no-edit'`, save, quit. No need to edit! – ntc2 Mar 6, 2015 at 23:47

1578

+50

> This answer uses `git-filter-branch`, for which the docs now give this warning:
>
> git filter-branch has a plethora of pitfalls that can produce non-obvious manglings of the intended history rewrite (and can leave you with little time to investigate such problems since it has such abysmal performance). These safety and performance issues cannot be backward compatibly fixed and as such, its use is not recommended. Please use an alternative history filtering tool such as git filter-repo. If you still need to use git filter-branch, please carefully read SAFETY (and PERFORMANCE) to learn about the land mines of filter-branch, and then vigilantly avoid as many of the hazards listed there as reasonably possible.

Changing the author (or committer) would require rewriting all of the history. If you're okay with that and think it's worth it then you should check out git filter-branch. The manual page includes several examples to get you started. Also note that you can use environment variables to change the name of the author, committer, dates, etc. -- see the "Environment Variables" section of the git manual page.

Specifically, you can fix all the wrong author names and emails **for all branches and tags** with this command (source: GitHub help):

```
#!/bin/sh

git filter-branch --env-filter '
OLD_EMAIL="your-old-email@example.com"
CORRECT_NAME="Your Correct Name"
CORRECT_EMAIL="your-correct-email@example.com"
if [ "$GIT_COMMITTER_EMAIL" = "$OLD_EMAIL" ]
then
    export GIT_COMMITTER_NAME="$CORRECT_NAME"
    export GIT_COMMITTER_EMAIL="$CORRECT_EMAIL"
fi
if [ "$GIT_AUTHOR_EMAIL" = "$OLD_EMAIL" ]
then
    export GIT_AUTHOR_NAME="$CORRECT_NAME"
    export GIT_AUTHOR_EMAIL="$CORRECT_EMAIL"
fi
' --tag-name-filter cat -- --branches --tags
```

For using alternative history filtering tool git filter-repo, you can first install it and construct a `git-mailmap` according to the format of gitmailmap.

```
Proper Name <proper@email.xx> Commit Name <commit@email.xx>
```

And then run filter-repo with the created mailmap:

```
git filter-repo --mailmap git-mailmap
```

Share Improve this answer Follow

edited Mar 20, 2022 at 0:20
Skully
3,087 ● 3 ● 27 ● 41

answered Apr 15, 2009 at 3:16
Pat Notz
214k ● 31 ● 94 ● 92

49 After executing the script you may remove the backup branch by executing "git update-ref -d refs/original/refs/heads/master". – D.R. Aug 14, 2013 at 16:47

13 @rodowi, it duplicates all my commits. – Rafael Barros Jun 17, 2014 at 17:43

12 @RafaelBarros the author info (just like anything else in the history) is part of the commit's sha key. Any change to the history is a rewrite leading to new id's for all commits. So don't rewrite on a shared repo or make sure all users are aware of it ... – johannes Jun 11, 2015 at 13:52

32 Solved using `git push --force --tags origin HEAD:master` – mcont Nov 13, 2016 at 11:50

26 **IMPORTANT!!!** Before executing the script, set your user.name and user.email git config parameter properly! And after executing the script you'll have some duplicate backup history called "original"! Delete it via `git update-ref -d refs/original/refs/heads/master` and then check if `.git/refs/original` folder structure is empty and then just remove it with `rm -rf .git/refs/original`. Lastly, you can verify the new rewritten log via: `git log --pretty=format:"[%h] %cd - Committer: %cn (%ce), Author: %an (%ae)"` ! One more thing: `.git/logs` has some log files that still have your old name! – user964843 Feb 3, 2017 at 22:23

780

One liner, but be careful if you have a multi-user repository - this will change *all* commits to have the same (new) author and committer.

```
git filter-branch -f --env-filter "GIT_AUTHOR_NAME='Newname'; GIT_AUTHOR_EMAIL='new@email'; GIT_COMMITTER_NAME='Newname';
GIT_COMMITTER_EMAIL='new@email';" HEAD
```

With linebreaks in the string (which is possible in bash):

```
git filter-branch -f --env-filter "
    GIT_AUTHOR_NAME='Newname'
    GIT_AUTHOR_EMAIL='new@email'
    GIT_COMMITTER_NAME='Newname'
    GIT_COMMITTER_EMAIL='new@email'
  " HEAD
```

Share Improve this answer Follow

edited Oct 20, 2015 at 21:10
kenorb
166k ●94 ●706 ●772

answered Apr 15, 2009 at 3:22
Brian Gianforcaro
27.1k ●11 ●59 ●78

8 Why does it rewrite all commits if you specify `HEAD` in the end of the command? – Nick Volynkin Jun 18, 2015 at 3:26

5 This does not work for my bitbucket repository, any idea ? I do a `git push --force --tags origin 'refs/heads/*'` after the advised command – Olórin Oct 5, 2016 at 21:46

13 The push command for this is : `$git push --force --tags origin 'refs/heads/master'` – HARSH NILESH PATHAK Jun 8, 2018 at 22:07

18 Neat; this keeps the old timestamps too. – DharmaTurtle Mar 6, 2020 at 21:19

5 @HARSHNILESHPATHAK Note that for recently created repositories the branch master has been renamed main, so the command becomes `$git push --force --tags origin 'refs/heads/main'` – Alex Gisi Jan 20, 2021 at 22:30

623

You can also do:

```
git filter-branch --commit-filter '
        if [ "$GIT_COMMITTER_NAME" = "<Old Name>" ];
        then
                GIT_COMMITTER_NAME="<New Name>";
                GIT_AUTHOR_NAME="<New Name>";
                GIT_COMMITTER_EMAIL="<New Email>";
                GIT_AUTHOR_EMAIL="<New Email>";
                git commit-tree "$@";
        else
                git commit-tree "$@";
        fi' HEAD
```

Note, if you are using this command in the Windows command prompt, then you need to use `"` instead of `'`:

```
git filter-branch --commit-filter "
        if [ "$GIT_COMMITTER_NAME" = "<Old Name>" ];
        then
                GIT_COMMITTER_NAME="<New Name>";
                GIT_AUTHOR_NAME="<New Name>";
                GIT_COMMITTER_EMAIL="<New Email>";
                GIT_AUTHOR_EMAIL="<New Email>";
                git commit-tree "$@";
        else
                git commit-tree "$@";
        fi" HEAD
```

Share Improve this answer Follow

edited Jun 23, 2014 at 18:14
user456814

answered May 15, 2009 at 19:15
Rognon
6,387 ●2 ●17 ●6

6 Isn't using env-filter the easier solution? Not sure why this is getting more votes, then. – stigkj Dec 9, 2011 at 9:21

3 Then link is broken. How do we push these changes to another repository? – Russell Feb 18, 2012 at 23:21

30 env-filter will change all the commits. This solution allows a conditional. – user208769 Apr 11, 2012 at 15:29

6 `"A previous backup already exists in refs/original/ Force overwriting the backup with -f"` sorry but where the `-f` -flag is going to be whene executing this script two times. Actually that is in Brian's answer, sorry about disturbance just after the filter-branch is the solution. – hhh May 4, 2012 at 22:11

2 @user208769 env-filter also allows a conditional; look at my answer :-) – stigkj Apr 11, 2013 at 8:52

254

It happens when you do not have a `$HOME/.gitconfig` initialized. You may fix this as:

```
git config --global user.name "you name"
git config --global user.email you@domain.example
git commit --amend --reset-author
```

Tested with Git version 1.7.5.4.

Note that this fixes only the last commit.

Share Improve this answer Follow

edited Jun 21, 2022 at 12:41
Stephen Ostermiller ♦ 25.4k ● 16 ● 94 ● 114

answered Feb 16, 2012 at 9:46
lrkwz 6,516 ● 4 ● 40 ● 65

11 That works really well on the last commit. Nice and simple. Doesn't *have* to be a global change, using `--local` works too – Ben May 30, 2012 at 23:24

This one was the big winner for me! The `git commit --amend --reset-author --no-edit` command is especially useful if you created commits with the wrong author information, then set the correct author after-the-fact via `git config`. Saved my a$$ just now when I had to update my email. – ecbrodie Feb 8, 2019 at 19:36

The answers might be overkill. First check whether this satisfies your usecase - stackoverflow.com/a/67363253/8293309 – paradocslover May 3, 2021 at 3:48

211

In the case where just the top few commits have bad authors, you can do this all inside `git rebase -i` using the `exec` command and the `--amend` commit, as follows:

```
git rebase -i HEAD~6 # as required
```

which presents you with the editable list of commits:

```
pick abcd Someone else's commit
pick defg my bad commit 1
pick 1234 my bad commit 2
```

Then add `exec ... --author="..."` lines after all lines with bad authors:

```
pick abcd Someone else's commit
pick defg my bad commit 1
exec git commit --amend --author="New Author Name <email@address.example>" -C HEAD
pick 1234 my bad commit 2
exec git commit --amend --author="New Author Name <email@address.example>" -C HEAD
```

save and exit editor (to run).

This solution may be longer to type than some others, but it's highly controllable - I know exactly what commits it hits.

Thanks to @asmeurer for the inspiration.

Share Improve this answer Follow

edited Jun 21, 2022 at 12:44
Stephen Ostermiller ♦ 25.4k ● 16 ● 94 ● 114

answered Dec 8, 2011 at 17:05
Alex Brown 42.8k ● 10 ● 95 ● 109

32 Definitely awesome. Can you shorten it by setting user.name and user.email in the repo's local config, and then each line is only `exec git commit --amend --reset-author -C HEAD` ? – Andrew Nov 30, 2012 at 11:07

2 The canonical answer, to use filter-branch, just deleted refs/heads/master for me. So +1 to your controllable, editable solution. Thanks! – jmtd Jun 17, 2014 at 20:30

3 In place of `git rebase -i HEAD^^^^^^` you can also write `git rebase -i HEAD~6` – Patrick Schlüter Jun 9, 2015 at 13:09

4 Please note that this changes the timestamp of the commits. See stackoverflow.com/a/11179245/1353267 for reverting to the correct timestamps – Samveen Oct 10, 2017 at 11:50

2 For anyone else struggling with the same problem as me, if you are trying to include the initial commit and you get `fatal: Needed a single revision`, try `git rebase -i --root` instead – DJMcMayhem Nov 23, 2020 at 18:44

194

For a single commit:

```
git commit --amend --author="Author Name <email@address.example>"
```

(extracted from asmeurer's answer)

Share Improve this answer Follow

edited Jun 21, 2022 at 12:44
Stephen Ostermiller ♦
25.4k ● 16 ● 94 ● 114

answered Apr 26, 2010 at 22:50
blueyed
27.8k ● 4 ● 79 ● 71

19 but that's only if it's the most recent commit – Richard Jan 17, 2012 at 23:24

4 According to `git help commit`, `git commit --amend` changes the commit at the "tip of the current branch" (which is HEAD). This is normally the most recent commit, but you can make it any commit you want by first checking out that commit with `git checkout <branch-name>` or `git checkout <commit-SHA>`. – Rory O'Kane Apr 25, 2012 at 19:33

13 But if you do that, all of the commits that already have that commit as a parent will be pointing to the wrong commit. Better to use filter-branch at that point. – John Gietzen Jul 11, 2012 at 21:02

3 @JohnGietzen: You can rebase the commits back onto the one that's changed to fix that. However, if you're doing >1 commit, then as mentioned, filter-branch is probably going to be a lot easier. – Thanatos Oct 24, 2013 at 20:35

14 Note that this changes only commit `author` and not the `committer` – Nick Volynkin Jun 18, 2015 at 3:39

124

GitHub originally had a nice solution (broken link), which was the following shell script:

```
#!/bin/sh

git filter-branch --env-filter '

an="$GIT_AUTHOR_NAME"
am="$GIT_AUTHOR_EMAIL"
cn="$GIT_COMMITTER_NAME"
cm="$GIT_COMMITTER_EMAIL"

if [ "$GIT_COMMITTER_EMAIL" = "your@email.to.match.example" ]
then
    cn="Your New Committer Name"
    cm="Your New Committer Email"
fi
if [ "$GIT_AUTHOR_EMAIL" = "your@email.to.match.example" ]
then
    an="Your New Author Name"
    am="Your New Author Email"
fi

export GIT_AUTHOR_NAME="$an"
export GIT_AUTHOR_EMAIL="$am"
export GIT_COMMITTER_NAME="$cn"
export GIT_COMMITTER_EMAIL="$cm"
'
```

Share Improve this answer Follow

edited Jun 21, 2022 at 12:45
Stephen Ostermiller ♦
25.4k ● 16 ● 94 ● 114

answered Oct 7, 2010 at 9:54
Olivier Verdier
48.9k ● 30 ● 101 ● 92

5 Worked perfectly. Just had to `git reset --hard HEAD^` a couple of times on the other local repositories to get them to an earlier version, `git pull`-ed the amended version, and here I am without any lines containing `unknown <stupid-windows-user@.StupidWindowsDomain.local>` (got to love git's defaulting). – Alan Plum Jan 8, 2011 at 17:34

1 I cannot push after this. Do I have to use "-f"? – Fish Monitor Jul 30, 2012 at 7:01

9 I did `git push -f`. Also, local repos have to be recloned after this. – Fish Monitor Jul 30, 2012 at 7:23

If you need to run the shell script on a specific branch you can change the last line into: "' master..your-branch-name" (assuming you branched of master). – Robert Kajic May 29, 2013 at 18:23

Click on the link <nice solution> as the script has been updated – gxpr Apr 10, 2018 at 11:23

103

A **single command** to change the author for the last N commits:

```
git rebase -i HEAD~N -x "git commit --amend --author 'Author Name <author.name@mail.example>' --no-edit"
```

NOTES

- replace `HEAD~N` with the reference until where you want to rewrite your commits. This can be a hash, `HEAD~4`, a branch name, ...
- the `--no-edit` flag makes sure the `git commit --amend` doesn't ask an extra confirmation
- when you use `git rebase -i`, you can manually select the commits where to change the author,

the file you edit will look like this:

```
pick 897fe9e simplify code a little
exec git commit --amend --author 'Author Name <author.name@mail.example>' --no-edit
pick abb60f9 add new feature
exec git commit --amend --author 'Author Name <author.name@mail.example>' --no-edit
pick dc18f70 bugfix
exec git commit --amend --author 'Author Name <author.name@mail.example>' --no-edit
```

You can then still modify some lines to see where you want to change the author. This gives you a nice middle ground between automation and control: you see the steps that will run, and once you save everything will be applied at once.

Note that if you already fixed the author information with `git config user.name <your_name>` and `git config user.email <your_email>`, you can also use this command:

```
git rebase -i HEAD~N -x "git commit --amend --reset-author --no-edit"
```

Share Improve this answer Follow

edited Jun 21, 2022 at 12:45
Stephen Ostermiller ♦
25.4k ● 16 ● 94 ● 114

answered Jan 25, 2019 at 10:14
Chris Maes
37.6k ● 15 ● 116 ● 153

I used HEAD~8 and it shows way more than the last 8 commits. – Bryan Bryce Jan 17, 2020 at 0:05

2 @BryanBryce if there are merge commits involved, things get complicated :) – Chris Maes Jan 17, 2020 at 8:04

5 You use `--root` instead of `HEAD~N` to edit the entire history (including initial commit), and use `--reset-author` to take the current committer instead of `--author ...` – Allan Deamon Feb 25, 2021 at 12:26

3 My use case was that I had to change all past commits in some private repositories because my pushes were under a different username with no email attached. The first bit allowed me to change the author and email for the first N commits but it did not preserve the commit timestamps, those got updated along with it. I solved this by using this script. It is nice and clean and allows me to change the entire commit history to a single username and email while preserving the commit timestamps. – Ajay Pillay Jun 7, 2021 at 12:18

1 @PedroHenrique: you need to replace `HEAD~4` with the reference until where you want to rewrite your commits... I'll try to make this a little clearer in my answer. As I mentioned before: beware for merge commits where you will get into complicated stuff – Chris Maes Sep 8, 2021 at 6:37

90

As docgnome mentioned, rewriting history is dangerous and will break other people's repositories.

But if you really want to do that and you are in a bash environment (no problem in Linux, on Windows, you can use git bash, that is provided with the installation of git), use git filter-branch:

```
git filter-branch --env-filter '
  if [ $GIT_AUTHOR_EMAIL = bad@email ];
    then GIT_AUTHOR_EMAIL=correct@email;
  fi;
export GIT_AUTHOR_EMAIL'
```

To speed things up, you can specify a range of revisions you want to rewrite:

```
git filter-branch --env-filter '
  if [ $GIT_AUTHOR_EMAIL = bad@email ];
    then GIT_AUTHOR_EMAIL=correct@email;
  fi;
export GIT_AUTHOR_EMAIL' HEAD~20..HEAD
```

Share Improve this answer Follow

edited Jun 23, 2014 at 18:29
user456814

answered Aug 4, 2010 at 0:52
svick
244k ● 53 ● 403 ● 526

2 Do note that this will leave any tags pointing at the old commits. `--tag-name-filter cat` is the "make it work" option. – Roman Starkov Mar 27, 2014 at 16:46

@romkyns any idea on how to change tags as well? – Nick Volynkin Jun 18, 2015 at 3:30

@NickVolynkin Yes, you specify `--tag-name-filter cat`. This really should have been the default behaviour. – Roman Starkov Jun 18, 2015 at 21:36

The answers might be overkill. First check whether this satisfies your usecase - stackoverflow.com/a/67363253/8293309 – paradocslover May 3, 2021 at 3:49

71

I should point out that if the only problem is that the author/email is different from your usual, this is not a problem. The correct fix is to create a file called `.mailmap` at the base of the directory with lines like

```
Name you want <email you want> Name you don't want <email you don't want>
```

And from then on, commands like `git shortlog` will consider those two names to be the same (unless you specifically tell them not to). See https://schacon.github.io/git/git-shortlog.html for more information.

This has the advantage of all the other solutions here in that you don't have to rewrite history, which can cause problems if you have an upstream, and is always a good way to accidentally lose data.

Of course, if you committed something as yourself and it should really be someone else, and you don't mind rewriting history at this point, changing the commit author is probably a good idea for attribution purposes (in which case I direct you to my other answer here).

Share Improve this answer Follow

edited Mar 20, 2022 at 0:27
Skully
3,087 ● 3 ● 27 ● 41

answered Feb 28, 2012 at 23:57
asmeurer
91.2k ● 29 ● 180 ● 253

Actually this is a very interesting answer. In my case I made some commits from home and it may be confusing an extra author so this is all I needed. – Òscar Raya Sep 8, 2020 at 10:31

2 Also, notice this does not works for web side on Gitea. – Òscar Raya Sep 8, 2020 at 10:39

@iuliu.net I'm not sure. This question stackoverflow.com/questions/53629125/... seems to suggest it does, but I haven't confirmed it. Certainly if they don't then they ought to, because it's a standard part of git. – asmeurer Jan 11, 2021 at 9:20

As of today (2024-05-11) and according to the comment [1], the mailmap functionality is no longer supported by Github. [1] - stackoverflow.com/a/77616449/6158542 – JonyD May 11 at 2:03

55

You can use this as a **alias** so you can do:

```
git change-commits GIT_AUTHOR_NAME "old name" "new name"
```

or for the last 10 commits:

```
git change-commits GIT_AUTHOR_EMAIL "old@email.com" "new@email.com" HEAD~10..HEAD
```

Add to ~/.gitconfig:

```
[alias]
    change-commits = "!f() { VAR=$1; OLD=$2; NEW=$3; shift 3; git filter-branch --env-filter \"if [[ \\\"$`echo $VAR`\\\" =
'$OLD' ]]; then export $VAR='$NEW'; fi\" $@; }; f "
```

Source: https://github.com/brauliobo/gitconfig/blob/master/configs/.gitconfig

Hope it is useful.

Share Improve this answer Follow

edited Jan 23, 2019 at 23:33
Borzh
5,195 ● 2 ● 52 ● 66

answered Aug 1, 2012 at 23:09
brauliobo
6,296 ● 4 ● 30 ● 36

6 "git: 'change-commits' is not a git command. See 'git --help'." – Native_Mobile_Arch_Dev Apr 3, 2017 at 2:05

After this command & sync with master all commits in the history are duplicated! Even of other users :(– Vladimir Feb 26, 2019 at 13:30

@Vladimir that is expected, please study about changing history in git – brauliobo Feb 28, 2019 at 10:26

1 For me it seems to run in /bin/sh, so I had to replace the bash-specific test `[[ ]]` with sh-compatible test `[ ]` (single brackets). Besides that it works very well, thanks! – Steffen Schwigon Jun 5, 2020 at 10:22

@Native_Mobile_Arch_Dev You need this: git config --global alias.change-commits '!'"f() { VAR=\$1; OLD=\$2; NEW=\$3; shift 3; git filter-branch --env-filter \"if [[\\\"\$`echo \$VAR`\\\" = '\$OLD']]; then export \$VAR='\$NEW'; fi\" \$@; }; f" – Amir Hajiha Mar 29, 2021 at 7:45

52

A safer alternative to git's `filter-branch` is `filter-repo` tool as suggested by git docs here.

```
git filter-repo --commit-callback '
  old_email = b"your-old-email@example.com"
  correct_name = b"Your Correct Name"
  correct_email = b"your-correct-email@example.com"

  if commit.committer_email == old_email :
    commit.committer_name = correct_name
    commit.committer_email = correct_email

  if commit.author_email == old_email :
    commit.author_name = correct_name
    commit.author_email = correct_email
  '
```

The above command mirrors the logic used in this script but uses `filter-repo` instead of `filter-branch`.

The code body after `commit-callback` option is basically python code used for processing commits. You can write your own logic in python here. See more about `commit` object and its attributes here.

Since `filter-repo` tool is not bundled with git you need to install it separately.

See Prerequisties and Installation Guide

If you have a python env >= 3.5, you can use `pip` to install it.

```
pip3 install git-filter-repo
```

Note: It is strongly recommended to try `filter-repo` tool on a fresh clone. Also remotes are removed once the operation is done. Read more on why remotes are removed here. Also read the limitations of this tool under INTERNALS section.

Share Improve this answer Follow

edited Nov 26, 2021 at 2:43
Wenfang Du
11.1k ● 13 ● 73 ● 111

answered Apr 22, 2020 at 15:03
Saurabh P Bhandari
6,672 ● 1 ● 20 ● 54

2 This seems to be the new kid on the block and I cherish this answer like gold. remember the fields have to be binary and then remove the == lines, and You can unconditionally change everything before pushing. Did I say I like this answer? It should be the accepted one. – user2692263 Feb 27, 2021 at 9:08

Thank you for sharing the links in the `Note` section. – hustnzj Aug 31, 2022 at 9:28

To get this to work on Windows, I had to escape all the double quotes: `old_email = b\"your-old-email@example.com\"` – RapidIce Sep 1, 2022 at 6:46

Worked like a charm. Thanks! – xenoid Aug 1 at 17:09

52

This is a more elaborated version of [@Brian's version](#):

To change the author and committer, you can do this (with linebreaks in the string which is possible in bash):

```
git filter-branch --env-filter '
    if [ "$GIT_COMMITTER_NAME" = "<Old name>" ];
    then
        GIT_COMMITTER_NAME="<New name>";
        GIT_COMMITTER_EMAIL="<New email>";
        GIT_AUTHOR_NAME="<New name>";
        GIT_AUTHOR_EMAIL="<New email>";
    fi' -- --all
```

You might get one of these errors:

1. The temporary directory exists already
2. Refs starting with *refs/original* exists already
   (this means another filter-branch has been run previously on the repository and the then original branch reference is backed up at *refs/original*)

If you want to force the run in spite of these errors, add the `--force` flag:

```
git filter-branch --force --env-filter '
    if [ "$GIT_COMMITTER_NAME" = "<Old name>" ];
    then
        GIT_COMMITTER_NAME="<New name>";
        GIT_COMMITTER_EMAIL="<New email>";
        GIT_AUTHOR_NAME="<New name>";
        GIT_AUTHOR_EMAIL="<New email>";
    fi' -- --all
```

A little explanation of the `-- --all` option might be needed: It makes the filter-branch work on all revisions on *all refs* (which includes all branches). This means, for example, that tags are also rewritten and is visible on the rewritten branches.

A common "mistake" is to use `HEAD` instead, which means filtering all revisions on just the *current branch*. And then no tags (or other refs) would exist in the rewritten branch.

Share Improve this answer Follow

edited Jan 30, 2023 at 0:01
Lucas
541 ●2 ●10 ●23

answered Dec 9, 2011 at 10:23
stigkj
1,529 ●14 ●10

Kudos for supplying a procedure that changes commits on *all* refs/branches. – Johnny Utahh May 17, 2015 at 22:05

1 Doesn't work on Windows fatal: $GIT_COMMITTER_NAME: no such path in the working tree. – Cherona Nov 10, 2023 at 12:04

49

When taking over an unmerged commit from another author, there is an easy way to handle this.

`git commit --amend --reset-author`

Share Improve this answer Follow

answered Mar 23, 2016 at 22:23
Ryanmt
3,245 ●3 ●23 ●23

1 For a single commit, and if you wanna put your username, this is most easy way. – Pedro Benevides Apr 6, 2016 at 17:08

7 You can add `--no-edit` to make this even easier, as generally most people will want to update only the email address and not the commit message – Debajit Jun 13, 2016 at 18:15

Can you guys please share the git command for just to update last commit's email/username with the new one – Adil Aug 3, 2016 at 11:52

Did you try this? That should be a side effect of this, if not [stackoverflow.com/a/2717477/654245](#) looks like a good path. – Ryanmt Aug 4, 2016 at 3:08

Note that this changes the author date – unkulunkulu Mar 10, 2023 at 21:18

29

1. run `git rebase -i <sha1 or ref of starting point>`
2. mark all commits that you want to change with `edit` (or `e`)
3. loop the following two commands until you have processed all the commits:

   `git commit --amend --reuse-message=HEAD --author="New Author <new@author.email>"` ; `git rebase --continue`

This will keep all the other commit information (including the dates). The `--reuse-message=HEAD` option prevents the message editor from launching.

Share Improve this answer Follow

edited Jun 20, 2020 at 9:12
Community Bot
1 ●1

answered Oct 4, 2012 at 2:22
sporsh
387 ●4 ●3

This doesn't update the committer. If you want to update the author and committer while keeping the dates, you may be interested in my answer – DharmaTurtle Aug 11, 2022 at 1:44

28

I use the following to rewrite the author for an entire repository, including tags and all branches:

```
git filter-branch --tag-name-filter cat --env-filter "
  export GIT_AUTHOR_NAME='New name';
  export GIT_AUTHOR_EMAIL='New email'
" -- --all
```

Then, as described in the MAN page of filter-branch, remove all original refs backed up by `filter-branch` (this is destructive, backup first):

```
git for-each-ref --format="%(refname)" refs/original/ | \
xargs -n 1 git update-ref -d
```

Share Improve this answer Follow

edited Aug 2 at 20:53

answered Feb 16, 2011 at 15:27
Ton van den Heuvel
10.5k ●6 ●45 ●89

3 It's very important to use `--tag-name-filter cat`. Otherwise your tags will remain on the original chain of commits. The other answers fail to mention this. – jeberle Mar 30, 2014 at 17:22

21

I adapted this solution which works by ingesting a simple `author-conv-file` (format is the same as one for git-cvsimport). It works by changing all users as defined in the `author-conv-file` across all branches.

We used this in conjunction with `cvs2git` to migrate our repository from cvs to git.

i.e. Sample `author-conv-file`

```
john=John Doe <john.doe@hotmail.com>
jill=Jill Doe <jill.doe@hotmail.com>
```

The script:

```
#!/bin/bash

export $authors_file=author-conv-file

git filter-branch -f --env-filter '

get_name () {
    grep "^$1=" "$authors_file" |
    sed "s/^.*=\(.*\) <.*>$/\1/"
}

get_email () {
    grep "^$1=" "$authors_file" |
    sed "s/^.*=.* <\(.*\)>$/\1/"
}

GIT_AUTHOR_NAME=$(get_name $GIT_COMMITTER_NAME) &&
    GIT_AUTHOR_EMAIL=$(get_email $GIT_COMMITTER_NAME) &&
    GIT_COMMITTER_NAME=$GIT_AUTHOR_NAME &&
    GIT_COMMITTER_EMAIL=$GIT_AUTHOR_EMAIL &&
    export GIT_AUTHOR_NAME GIT_AUTHOR_EMAIL &&
    export GIT_COMMITTER_NAME GIT_COMMITTER_EMAIL
' -- --all
```

Share Improve this answer Follow

edited Jun 23, 2014 at 18:20
user456814

answered Feb 4, 2011 at 22:46
leif.gruenwoldt
13.9k ● 5 ● 62 ● 66

Thanks, I wonder why this is not core git (or git-svn) functionality. This can be done with a flag for git svn clone, but not in git filter-branch...
– Daniel Hershcovich Feb 15, 2012 at 13:36

19

I found the presented versions way to aggressive, especially if you commit patches from other developers, this will essentially steal their code.

The version below does work on all branches and changes the author and comitter separately to prevent that.

Kudos to leif81 for the all option.

```
#!/bin/bash

git filter-branch --env-filter '
if [ "$GIT_AUTHOR_NAME" = "<old author>" ];
then
    GIT_AUTHOR_NAME="<new author>";
    GIT_AUTHOR_EMAIL="<youmail@somehost.ext>";
fi
if [ "$GIT_COMMITTER_NAME" = "<old committer>" ];
then
    GIT_COMMITTER_NAME="<new commiter>";
    GIT_COMMITTER_EMAIL="<youmail@somehost.ext>";
fi
' -- --all
```

Share Improve this answer Follow

edited Jun 23, 2014 at 18:19
user456814

answered Apr 23, 2012 at 8:46
drahnr
6,886 ● 5 ● 51 ● 75

19

1. Change commit `author name & email` by `Amend`, then replacing `old-commit with new-one`:

```
$ git checkout <commit-hash>                            # checkout to the commit need to modify
$ git commit --amend --author "name <author@email.com>" # change the author name and email

$ git replace <old-commit-hash> <new-commit-hash>      # replace the old commit by new one
$ git filter-branch -- --all                           # rewrite all futures commits based on the replacement

$ git replace -d <old-commit-hash>     # remove the replacement for cleanliness
$ git push -f origin HEAD              # force push
```

2. Another way `Rebasing`:

```
$ git rebase -i <good-commit-hash>      # back to last good commit

# Editor would open, replace 'pick' with 'edit' before the commit want to change author

$ git commit --amend --author="author name <author@email.com>"  # change the author name & email

# Save changes and exit the editor

$ git rebase --continue                # finish the rebase
```

Share Improve this answer Follow

edited May 19, 2017 at 5:13

answered Dec 14, 2016 at 15:01
Sajib Khan
24k ●9 ●68 ●79

2 Very nice answer. I like that the changes are wrapped up from the very update to even cleaning up the git commits – Aleks May 7, 2017 at 20:41

19

The fastest, easiest way to do this is to use the --exec argument of git rebase:

```
git rebase -i -p --exec 'git commit --amend --reset-author --no-edit'
```

This will create a todo-list that looks like this:

```
pick ef11092 Blah blah blah
exec git commit --amend --reset-author --no-edit
pick 52d6391 Blah bloh bloo
exec git commit --amend --reset-author --no-edit
pick 30ebbfe Blah bluh bleh
exec git commit --amend --reset-author --no-edit
...
```

and this will work all automatically, which works when you have hundreds of commits.

Share Improve this answer Follow

answered Oct 25, 2018 at 16:19
Lie Ryan
64.7k ●14 ●102 ●149

You can replace `-p` with `--root` to change *all* commits in the history (The -p option is deprecated). And note that this only works after you have corrected the username and email via `git config user.name <yourname>` and `git config user.email <youremail>`. – Wim Deblauwe Mar 6, 2021 at 10:15

1 I have a repository that I've been working on with another contributor. I want to change all **my** commits' credentials. Is your suggestion safe to use in this case, to avoid any modification on the other contributor's commits? – bbasaran Oct 26, 2022 at 21:12

12

One-liner with `filter-repo`:

You can use the callbacks feature of `git-filter-repo` (a recommended replacement for `filter-branch`) to change the name and email associated with all the commits:

```
git filter-repo --name-callback 'return b"New Name"' --email-callback 'return
b"newemail@gmail.com"'
```

This is more performant and potentially more reliable than solutions using `filter-branch`.

Note that the above command changes the authors (and committer) of **all** commits, if you want to effectively "edit" a certain author, and only modify the commits of that specific author, then use the `--commit-callback` option like this:

```
git filter-repo --commit-callback '
old_email = b"oldemail@gmail.com"
new_email = b"newemail@gmail.com"
new_name = b"New Author"

if commit.author_email == old_email:
    commit.author_email = new_email
    commit.author_name = new_name

if commit.committer_email == old_email:
    commit.committer_email = new_email
    commit.committer_name = new_name
'
```

(Just change the `old_email`, `new_email`, and `new_name` variables in the command above to the right values.)

Share Improve this answer Follow

edited Nov 5, 2022 at 20:17

answered Nov 4, 2022 at 10:20
Arad
12.4k ● 10 ● 66 ● 90

9

If you are the only user of this repository, you can **rewrite history** using either `git filter-branch` (as svick wrote), or `git fast-export` / `git fast-import` plus filter script (as described in article referenced in docgnome answer), or interactive rebase. But either of those would change revisions from first changed commit onwards; this means trouble for anybody that based his/her changes on your branch pre-rewrite.

RECOVERY

If other developers didn't based their work on pre-rewrite version, simplest solution would be to re-clone (clone again).

Alternatively they can try `git rebase --pull`, which would fast-forward if there weren't any changes in their repository, or rebase their branch on top of re-written commits (we want to avoid merge, as it would keep pre-rewrite comits forever). All of this assuming that they do not have not comitted work; use `git stash` to stash away changes otherwise.

If other developers use feature branches, and/or `git pull --rebase` doesn't work e.g. because upstream is not set up, they have to *rebase* their work on top of post-rewrite commits. For example just after fetching new changes (`git fetch`), for a `master` branch based on / forked from `origin/master`, one needs to run

```
$ git rebase --onto origin/master origin/master@{1} master
```

Here `origin/master@{1}` is pre-rewrite state (before fetch), see gitrevisions.

Alternate solution would be to use **refs/replace/** mechanism, available in Git since version 1.6.5. In this solution you provide replacements for commits that have wrong email; then anybody who fetches 'replace' refs (something like `fetch = +refs/replace/*:refs/replace/*` refspec in appropriate place in *their* `.git/config`) would get replacements transparently, and those who do not fetch those refs would see old commits.

The procedure goes something like this:

1. Find all commits with wrong email, for example using

   ```
   $ git log --author=user@wrong.email --all
   ```

2. For each wrong commit, create a replacement commit, and add it to object database

   ```
   $ git cat-file -p <ID of wrong commit> |
     sed -e 's/user@wrong\.email/user@example.com/g' > tmp.txt
   $ git hash-object -t commit -w tmp.txt
   <ID of corrected commit>
   ```

3. Now that you have corrected commit in object database, you have to tell git to automatically and transparently replace wrong commit by corrected one using `git replace` command:

   ```
   $ git replace <ID of wrong commit> <ID of corrected commit>
   ```

4. Finally, list all replacement to check if this procedure succeded

   ```
   $ git replace -l
   ```

   and check if replacements take place

   ```
   $ git log --author=user@wrong.email --all
   ```

You can of course automate this procedure... well, all except using `git replace` which doesn't have (yet) batch mode, so you would have to use shell loop for that, or replace "by hand".

NOT TESTED! YMMV.

Note that you might encounter some rough corners when using `refs/replace/` mechanism: it is new, and not yet very well tested.

Share Improve this answer Follow

edited May 23, 2017 at 11:47
Community Bot
1 ● 1

answered Aug 4, 2010 at 9:41
Jakub Narębski
323k ● 65 ● 225 ● 232

8

Note that git stores **two** different e-mail addresses, one for the *committer* (the person who committed the change) and another one for the *author* (the person who wrote the change).

The committer information isn't displayed in most places, but you can see it with `git log -1 --format=%cn,%ce` (or use `show` instead of `log` to specify a particular commit).

While changing the author of your last commit is as simple as `git commit --amend --author "Author Name <email@example.com>"`, there is no one-liner or argument to do the same to the committer information.

The solution is to (temporarily, or not) change your user information, then amend the commit, which will update the committer to your current information:

```
git config user.email my_other_email@example.com
git commit --amend
```

Share Improve this answer Follow

answered Dec 5, 2013 at 21:21
Sir Athos
9,847 ●2 ●24 ●26

Note that the old value is still in a few places in `path\to\repo\.git`. I'm not sure yet what you'd need to do to expunge it totally. Amends unfortunately (?) don't seem to erase. – ruffin Oct 8, 2014 at 15:02

8

For reset ALL commits (including first commit) to current user and current timestamp:

```
git rebase --root --exec "git commit --amend --no-edit --date 'now' --reset-author"
```

Share Improve this answer Follow

answered Dec 11, 2021 at 15:22
mixalbl4
3,917 ●1 ●34 ●47

this will work only for the current branch. – Chris Maes Dec 14, 2021 at 11:42

6

If the commits you want to fix are the latest ones, and just a couple of them, you can use a combination of `git reset` and `git stash` to go back an commit them again after configuring the right name and email.

The sequence will be something like this (for 2 wrong commits, no pending changes):

```
git config user.name <good name>
git config user.email <good email>
git reset HEAD^
git stash
git reset HEAD^
git commit -a
git stash pop
git commit -a
```

Share Improve this answer Follow

answered Sep 30, 2011 at 18:04
djromero
19.6k ●4 ●73 ●69

5

If you are using Eclipse with EGit, then there is a quite easy solution.
Assumption: you have commits in a local branch 'local_master_user_x' which cannot be pushed to a remote branch 'master' because of the invalid user.

1. Checkout the remote branch 'master'
2. Select the projects/folders/files for which 'local_master_user_x' contains changes
3. Right-click - Replace with - Branch - 'local_master_user_x'
4. Commit these changes again, this time as the correct user and into the local branch 'master'
5. Push to remote 'master'

Share Improve this answer Follow

answered Aug 24, 2011 at 17:54
paphko
51 ● 1 ● 1

5

Using interactive rebase, you can place an amend command after each commit you want to alter. For instance:

```
pick a07cb86 Project tile template with full details and styling
x git commit --amend --reset-author -Chead
```

Share Improve this answer Follow

edited Mar 19, 2013 at 21:02

answered Feb 26, 2013 at 13:19
j16r
309 ● 4 ● 8

4 The problem with this is that other commit metadata (e.g. date and time) is also amended. I just found that out the hard way `;-)`. – halfer Jul 7, 2013 at 20:31

5

We have experienced an issue today where a UTF8 character in an author name was causing trouble on the build server, so we had to rewrite the history to correct this. The steps taken were:

Step 1: Change your username in git for all future commits, as per instructions here: https://help.github.com/articles/setting-your-username-in-git/

Step 2: Run the following bash script:

```
#!/bin/sh

REPO_URL=ssh://path/to/your.git
REPO_DIR=rewrite.tmp

# Clone the repository
git clone ${REPO_URL} ${REPO_DIR}

# Change to the cloned repository
cd ${REPO_DIR}

# Checkout all the remote branches as local tracking branches
git branch --list -r origin/* | cut -c10- | xargs -n1 git checkout

# Rewrite the history, use a system that will preseve the eol (or lack of in commit messages) - preferably Linux not OSX
git filter-branch --env-filter '
OLD_EMAIL="me@something.com"
CORRECT_NAME="New Me"

if [ "$GIT_COMMITTER_EMAIL" = "$OLD_EMAIL" ]
then
    export GIT_COMMITTER_NAME="$CORRECT_NAME"
fi
if [ "$GIT_AUTHOR_EMAIL" = "$OLD_EMAIL" ]
then
    export GIT_AUTHOR_NAME="$CORRECT_NAME"
fi
' --tag-name-filter cat -- --branches --tags

# Force push the rewritten branches + tags to the remote
git push -f

# Remove all knowledge that we did something
rm -rf ${REPO_DIR}

# Tell your colleagues to `git pull --rebase` on all their local remote tracking branches
```

Quick overview: Checkout your repository to a temp file, checkout all the remote branches, run the script which will rewrite the history, do a force push of the new state, and tell all your colleagues to do a rebase pull to get the changes.

We had trouble with running this on OS X because it somehow messed up line endings in commit messages, so we had to re-run it on a Linux machine afterwards.

Share Improve this answer Follow

answered Oct 22, 2014 at 3:32
Miloš Ranđelović
444 ● 1 ● 5 ● 13

5

Your problem is really common. See "Using Mailmap to Fix Authors List in Git"

For the sake of simplicity, I have created a script to ease the process: git-changemail

After putting that script on your path, you can issue commands like:

- Change author matchings on current branch

```
$ git changemail -a old@email.com -n newname -m new@email.com
```

- Change author and committer matchings on <branch> and <branch2>. Pass `-f` to filter-branch to allow rewriting backups

```
$ git changemail -b old@email.com -n newname -m new@email.com -- -f &lt;branch> &lt;branch2>
```

- Show existing users on repo

```
$ git changemail --show-both
```

By the way, after making your changes, clean the backup from the filter-branch with: git-backup-clean

Share Improve this answer Follow

edited Feb 17, 2015 at 19:29
[illegible]
160k ● 44 ● 221 ● 306

answered Jan 3, 2015 at 12:19
albfan
12.9k ● 5 ● 64 ● 81

1 when i run your command, it says "fatal: cannot exec 'git-changemail': Permission denied" – Govind Sep 2, 2015 at 8:23

@Govind You need to set the execute permission for the script `chmod +x git-changemail` ! – the Tin Man Mar 19, 2021 at 0:45

1 2 Next

Highly active question. Earn 10 reputation (not counting the association bonus) in order to answer this question. The reputation requirement helps protect this question from spam and non-answer activity.