**How to Merge Git Branches and Preserve Histories**

More specifically, in cases where content is moved inside a git repo/branch and histories aren’t readily available from surface-level commands or in GitHub/Lab blame and history UIs.

Preserving histories has the benefit of only requiring a single move across multiple branch merges, for example: I map src to libs/project/src while preserving git histories in branch A — merging branch B will automatically map changes made to src to libs/project/src.

# Skipping to the Answer

1. Back up everything you’re working on — repo, branches, anything that could get modified during this. These are one-way changes and are destructive, there is no way to undo these steps other than to wipe, clone, and begin again.
2. Install <https://github.com/newren/git-filter-repo> — this will give you access to filter-repo as a git command ala git filter-repo
3. Consider two branches, base-branch, and new-feature. Checkout base, merge new-feature git merge origin/new-feature . If the branch lives in another repo you can git remote add ~~~, git remote update, git merge repo/branch.
4. git filter-repo --path-rename DIR\_TO\_BE\_MV/:MV\_TARGET\_DIR/ --force  
   This command will select a directory or file and move its contents into the target location. The content changes and history are being moved so the directory or file needs to be a part of the target, ie:  
   git filter-repo --path-rename app1/models:app2/models --force  
   git filter-repo --path-rename bin/page1.php:pages/page1.php --force
5. git remote add origin [git@github.com](mailto:git@github.com):some-org/some-repo.git  
   git branch --set-upstream-to=origin/base-branch base-branch  
   As these steps are destructive the utility automatically disconnects the local branch from the remote to prevent accidental overwrites. The above steps reconnect our local branch to remote so we can push our changes.
6. git pull  
   git fetch --tags -f  
   git push

## Considerations and Tips

**You need to merge both the base and the target.** It is not possible to clone in another repo or a group of files and run git filter-repo. There needs to be an established history for moves/renames to be mapped — new files don’t have history until they’re merged.

**Utilize scripting.** You definitely don’t want to be running these commands by hand; one single typo will cause the process to fail and you will need to restart the process with a full wipe, clone, merge, etc.

Prepopulate an entire list of commands and review stringently. git-filter-repo is fairly efficient and should only take seconds on a mid-sized application repository.

**If you make a mistake start over immediately.** If you accidentally map a directory to a file, ie app1/models/model\_a.php:app2/models the singular file’s history will get mapped to the models/ directory. This is a corrupted history that will either get entirely rejected, wiped, or actually merged when pushed to the remote. You will corrupt your branch, don’t waste your own time.

**Backup your work-in-progress branches.** I, on more than one occasion, made a mistake and had to start from scratch. Eventually, I realized I could have made a backup branch after the merge step, that way I would only have to wipe, clone, and checkout a new branch all without dealing with merge/merge conflict steps.

# Before Continuing You Should Ask Yourself: Is This Really Necessary?

* Why can’t you create another repository (or leave a ‘backup’ branch) and refer to it for blame history?
* Is the historical/blame information so crucial it needs to be readily available through the GitHub UI?

In my discovery I’ve seen numerous forum posts explicitly asking “how to move a file and preserve git history”; even though you won’t see your history on the “history” wheel in GitHub all changes including moves are tracked via [git logs](http://git-scm.com/docs/git-log) git log --folow ./path/to/file.

However this really isn’t the proper outcome, right? You can’t view your history through the GitHub UI and while logs will show the original file and history, this isn’t a conventional way to follow a file’s history. [Or is it?](https://github.com/git/git/blob/4d34dffbdd2226034df9d7612f1e221b7143fc6a/Documentation/technical/directory-rename-detection.txt)

To git, once a file is renamed or removed it no longer exists and git is correct in its assertion. Only as an external observer can we understand the context that a brand new file is the same file that was just renamed; git isn’t a file manager it’s a change manager.

# Why Did I Find This Necessary?

Not everyone is strong with git nor does anyone want to open up their ide, pull the correct repo/branch, and run through git logs.

I almost exclusively use git blame as a visual element during code reviews on GitHub and with a growing team I want to avoid cascading inefficiency. The time I spent researching, testing, failing, and performing git surgery on our core frontend application repo will easily be paid back in the time saved from having future developers wrangle with the aforementioned alternative process.

The benefit of being able to map our existing history to our newly formatted repository then merging branches based off the existing format without any manual process / moving is incredible. Our team was stunned when moving a development branch up to a quality assurance branch without needing to move \*any\* files.

# Exploring Alternatives

Take the path of least resistance. A certain git author pointed directly to the filter-repo tool and all other solutions have obvious pitfalls.

## git’s built-in tool: git-filter-branch

Origin: <https://git-scm.com/docs/git-filter-branch#_description>

The gist: git filter-branch is git’s utility for rewriting individual branch history.

Ironically the document opens up with WARNING and the author acknowledges that filter-branch is not performant and inherently lossy. One of the warnings is explicitly “git-filter-branch is riddled with gotchas resulting in various ways to easily corrupt repos or end up with a mess worse than what you started with”.

Honestly, it’s a hilarious read, the author has a clear disdain for the tool.

## Extract git history in email format, merge, convert back

Origin: <https://stackoverflow.com/a/33901284>

The gist: git’s email utility has an unintended side effect of being able to merge git history in a lossless way.

The method follows the described tool-less way git glosses over in their [directory rename detection](https://github.com/git/git/blob/4d34dffbdd2226034df9d7612f1e221b7143fc6a/Documentation/technical/directory-rename-detection.txt) document however overcoming history gaps by using the git email format. I feel like any developer that gives this a read would be hesitant to depend on something that requires data conversion and, more importantly, so many steps.