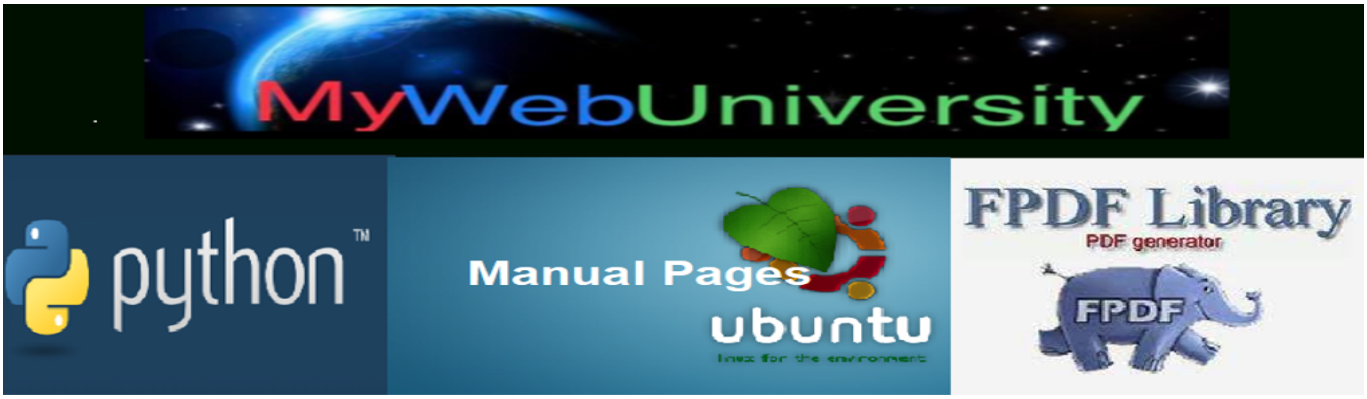




## *Rocky Enterprise Linux 9.2 Manual Pages on command 'git-rerere.1'*

***$ man git-rerere.1***

GIT-RERERE(1) Git Manual GIT-RERERE(1)

NAME

git-rerere - Reuse recorded resolution of conflicted merges

SYNOPSIS

git rerere [clear|forget <pathspec>|diff|remaining|status|gc]

DESCRIPTION

In a workflow employing relatively long lived topic branches, the developer sometimes needs to resolve the same conflicts over and over again until the topic branches are done (either merged to the "release" branch, or sent out and accepted upstream).

This command assists the developer in this process by recording conflicted automerge results and corresponding hand resolve results on the initial manual merge, and applying previously recorded hand resolutions to their corresponding automerge results.

Note

You need to set the configuration variable rerere.enabled in order to enable this command.

COMMANDS

Normally, git rerere is run without arguments or user-intervention. However, it has several commands that allow it to interact with its working state.

clear

Reset the metadata used by rerere if a merge resolution is to be aborted. Calling git am [--skip|--abort] or git rebase [--skip|--abort] will automatically invoke this command.

forget <pathspec>

Reset the conflict resolutions which rerere has recorded for the current conflict in <pathspec>.

diff

Display diffs for the current state of the resolution. It is useful for tracking what has changed while the user is resolving conflicts. Additional arguments are passed directly to the system diff command installed in PATH.

status

Print paths with conflicts whose merge resolution rerere will record.

remaining

Print paths with conflicts that have not been autoresolved by rerere. This includes paths whose resolutions cannot be tracked by rerere, such as conflicting submodules.

gc

Prune records of conflicted merges that occurred a long time ago. By default, unresolved conflicts older than 15 days and resolved conflicts older than 60 days are pruned. These defaults are controlled via the gc.rerereUnresolved and gc.rerereResolved configuration variables respectively.

## DISCUSSION

When your topic branch modifies an overlapping area that your master branch (or upstream) touched since your topic branch forked from it, you may want to test it with the latest master, even before your topic branch is ready to be pushed upstream:

```
            o---*---o topic
           /
      o---o---o---*---o---o master
```

For such a test, you need to merge master and topic somehow. One way to do it is to pull master into the topic branch:

```
$ git switch topic
$ git merge master
```

```
            o---*---o---+ topic
           /           /
      o---o---o---*---o---o master
```

The commits marked with * touch the same area in the same file; you need to resolve the conflicts when creating the commit marked with +. Then you can test the result to make sure your work-in-progress still works with what is in the latest master.

After this test merge, there are two ways to continue your work on the topic. The easiest is to build on top of the test merge commit +, and when your work in the topic branch is finally ready, pull the topic branch into master, and/or ask the upstream to pull from you. By that time, however, the master or the upstream might have been advanced since the test merge +, in which case the final commit graph would look like this:

```
$ git switch topic
$ git merge master
$ ... work on both topic and master branches
$ git switch master
$ git merge topic
```

```
      o---*---o---+---o---o topic
     /         /        \
o---o---o---*---o---o---o---o---+ master
```

When your topic branch is long-lived, however, your topic branch would end up having many such "Merge from master" commits on it, which would unnecessarily clutter the development history. Readers of the Linux kernel mailing list may remember that Linus complained about such too frequent test merges when a subsystem maintainer asked to pull from a branch full of "useless merges".

As an alternative, to keep the topic branch clean of test merges, you could blow away the test merge, and keep building on top of the tip before the test merge:

```
$ git switch topic
$ git merge master
$ git reset --hard HEAD^ ;# rewind the test merge
$ ... work on both topic and master branches
$ git switch master
$ git merge topic
```

```
      o---*---o-------o---o topic
     /                    \
o---o---o---*---o---o---o---o---+ master
```

This would leave only one merge commit when your topic branch is finally ready and merged into the master branch. This merge would require you to resolve the conflict, introduced by the commits marked with *. However, this conflict is often the same conflict you resolved when you created the test merge you blew away. git rerere helps you resolve this

final conflicted merge using the information from your earlier hand resolve.

Running the git rerere command immediately after a conflicted automerge records the conflicted working tree files, with the usual conflict markers <<<<<<<, =======, and >>>>>>> in them. Later, after you are done resolving the conflicts, running git rerere again will record the resolved state of these files. Suppose you did this when you created the test merge of master into the topic branch.

Next time, after seeing the same conflicted automerge, running git rerere will perform a three-way merge between the earlier conflicted automerge, the earlier manual resolution, and the current conflicted automerge. If this three-way merge resolves cleanly, the result is written out to your working tree file, so you do not have to manually resolve it. Note that git rerere leaves the index file alone, so you still need to do the final sanity checks with git diff (or git diff -c) and git add when you are satisfied.

As a convenience measure, git merge automatically invokes git rerere upon exiting with a failed automerge and git rerere records the hand resolve when it is a new conflict, or reuses the earlier hand resolve when it is not. git commit also invokes git rerere when committing a merge result. What this means is that you do not have to do anything special yourself (besides enabling the rerere.enabled config variable).

In our example, when you do the test merge, the manual resolution is recorded, and it will be reused when you do the actual merge later with the updated master and topic branch, as long as the recorded resolution is still applicable.

The information git rerere records is also used when running git rebase. After blowing away the test merge and continuing development on the topic branch:

```
            o---*---o-------o---o topic
           /
    o---o---o---*---o---o---o---o   master
```

```
$ git rebase master topic
```

```
                        o---*---o-------o---o topic
                       /
    o---o---o---*---o---o---o---o   master
```

you could run git rebase master topic, to bring yourself up to date before your topic is ready to be sent upstream. This would result in falling back to a three-way merge, and it would conflict the same way as the test merge you resolved earlier. git rerere will be run by git rebase to help you resolve this conflict.

[NOTE] git rerere relies on the conflict markers in the file to detect the conflict. If the file already contains lines that look the same as lines with conflict markers, git rerere may fail to record a conflict resolution. To work around this, the conflict-marker-size setting in gitattributes(5) can be used.

## GIT

Part of the git(1) suite