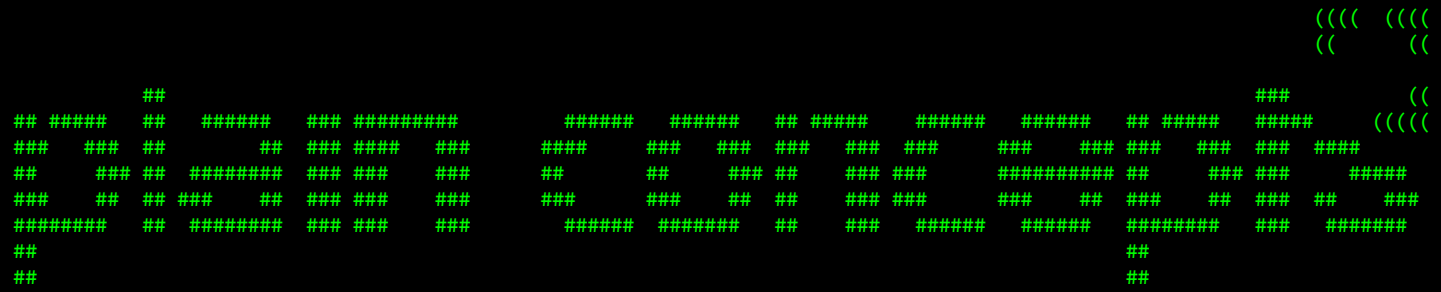


> git --rewrite --the --history
> Author: Joan Jané
> Date: 19/10/2017

>> DANGER ZONE <<

```
#  If you don’t know what are you doing, keep your hands off the
#  console :P

#  Changing the history of a repo should be only applied on
#  private commits, and never to public commits. This operations
#  create a new timeline that doesn’t share the same common
#  ancestors.

#  This means THEY ARE NEW COMMITS and will create conflicts with
#  existing remotes.

   > BE ADVISED, PRESS KEY TO CONTINUE...
```

Oh no! I’ve just committed a wrong message

Ok, don’t worry, this is an easy one, git amend *combines staged*
changes with the previous commit and also allows edit the
commit message.

```
> git commit --amend
```

This creates a new commit in your history, replacing the
previous one.

But what if I want to change the message of a commit that is
not the last one?

Rebase interactive

```
#  Rebase interactive command lets you apply this operations to a
#  range of commits:

#       * squash (melt a commit with previous)
#       * reword (change log message)
#       * edit (use commit and stop for amending)
#       * pick (use commit)
#       * fixup (squash but with new log message)
#       * exec (run command)
#       * drop (remove commit)
        * reorder commits (crazy)
```

Squash, WHY?

When working on a feature branch, you may create a large amount
of tiny commits (baby steps).

On many projects, especially on big Open Source Projects, it is
common that feature branches integrate to master squashing
commits into a single commit or few commits.

Rebase interactive

Example of changing the last 3 commits:

```
> git rebase -i HEAD~3
```

Perform operations and continue process until finishes:

```
> git rebase --continue
```

And remember, you can abort and start again:

```
> git rebase --abort
```

Deleting a commit of git history

Sometimes, you would like to delete a concrete commit that you
regret to have done. You can do it knowing its commit id:

```
# replace SHA with commit id like git rebase -p --onto abcde12^ abcde12
> git rebase -p --onto SHA^ SHA
```

Ups, I’ve committed on a wrong branch :P

Commit in a wrong branch is something that happens easily, when
this is the case, you can **cherry-pick** a commit and copy it to a
desired branch.

```
# checkout the target branch you want to have the commit
> git cherry-pick <sha1-commit-id>
```

Force clean of orphaned commits

When applying changes on git history, git still stores them on
file system. Git Garbage Collector is suposed to run clean tasks
periodically, but if you wish to force clean unreachable commits
that don't have any branch associated, you can do it so:

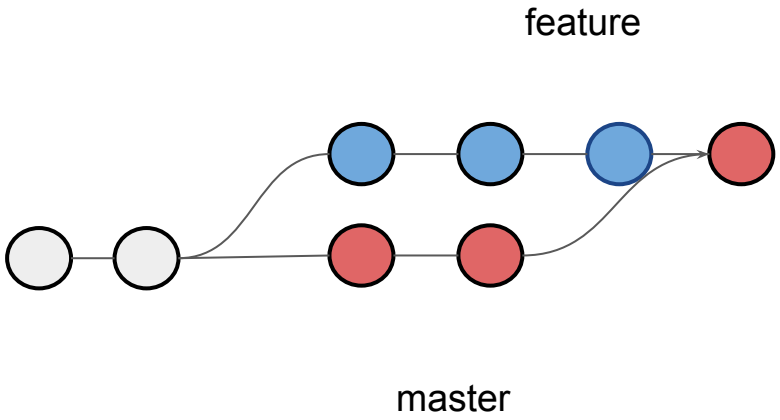
```
> git reflog expire --expire-unreachable=now --all
> git gc --prune=now
```

Merge vs. Rebase

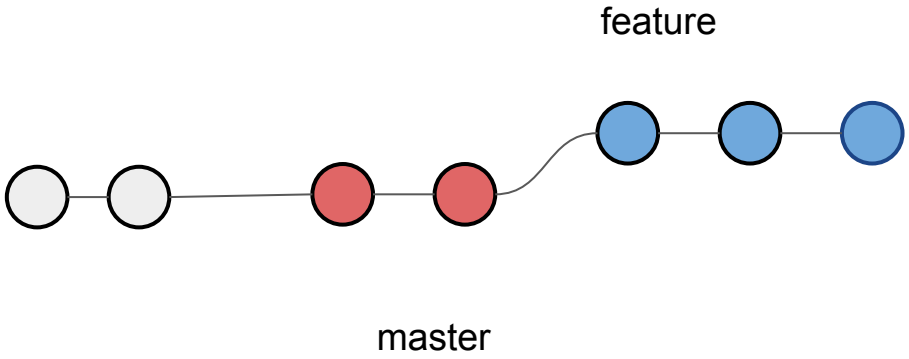
```
#    Rebase puts your commits on your branch on top of the incoming rebase branch,
#    rewriting the history.
#
#    Merge creates a new commit with incomming changes on history.
#
#    When to use merge:
#
#         * You are working on a public branch
#
#         * You may have outdated branch with many conflicts
#
#    When to use rebase:
#
#         * Otherwise and when you like a clean history
#
#         * You are not afraid of conflicts
#
```

Merge vs. Rebase

Merge:



Rebase:



Merge vs. Rebase

```
#   Merge:

    > git checkout feature # your branch

    > git merge master # incoming branch

#   Rebase:

    > git checkout feature # your branch

    > git rebase master # incoming branch

    #   If your branch is public, BAD BOY, now you have to push forced.

    > git push --force
```

Challenge 1

* Clone repo https://github.com/joanjane/git-challenge.git

* Open settings.txt file to see that there’s no relevant content.

* 2nd commit contains a password, after an security audit, we need to make it disappear from history for security reasons.

Challenge 1 - One solution

```
> git rebase -p --onto HEAD~2^ HEAD~2

> git rebase --continue
```

Challenge 2

* Checkout “test” branch on the same repo

* This branch contains 3 commits that we need to melt together in order to integrate to master.

Challenge 2 - Solution

```
> git rebase -i HEAD~3
```

And then, choose commits to squash:

```
# pick ******* Added 10mb file

# squash ******* Update settings

# squash ******* Updated 10mb file
```

More info / handy links

* https://git-scm.com/book/en/v2/Git-Tools-Rewriting-History
* https://www.atlassian.com/git/tutorials/rewriting-history
* https://www.atlassian.com/git/tutorials/merging-vs-rebasing
* https://gist.github.com/davfre/8313299