git(102)

How to hide the (code) bodies

In Brief, we'll cover

- git rebase
  - -i
  - --continue
  - --abort
- git commit
  - --amend
- git reset
  - --hard
  - --soft
- Some other neat tools

git rebase

Rebasing is re-writing your git history.

$ git rebase [ref]

Fix your branch

$ git rebase -i [ref]

If you want to edit, smush or otherwise mess with lots of commits at once, interactive rebasing is your tool.

Interactive rebase will open an interactive terminal which will let you:

```
p, pick = use commit
r, reword = use commit, but edit the commit message
e, edit = use commit, but stop for amending
s, squash = use commit, but meld into previous commit
f, fixup = like "squash", but discard this commit's log message
x, exec = run command (the rest of the line) using shell
```

git rebase master

It is useful for having linear git history, which beyond being cleaner, actually lends itself better to tools like *git blame.*

```
$ git rebase [ref to bring in to current branch]
```

With a tool like GitHub and using rebasing as your branch deployment, you will usually be doing (for solving conflicts):

```
$ git rebase master
```

git rebase --continue

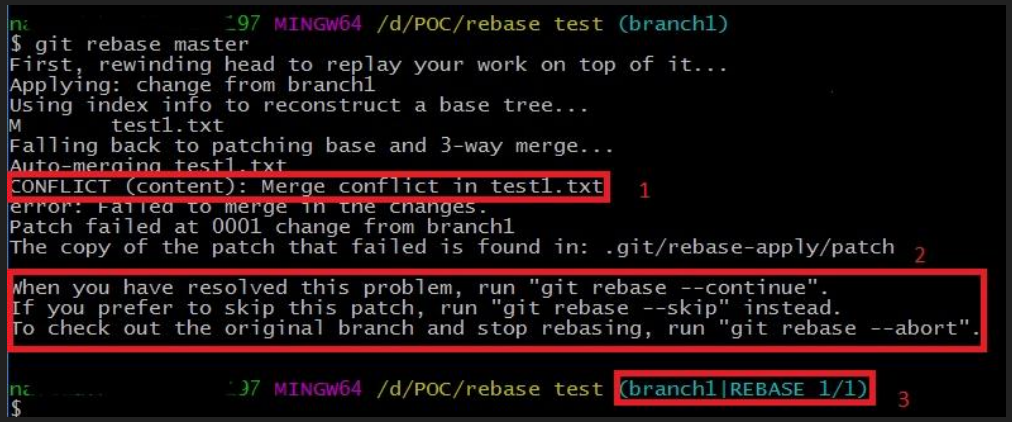
Sometimes your rebases don’t go to plan…

It’s ok. Rebase conflicts are the same as merge conflicts… just maybe a little messier.

$ git add [files]

$ git rebase --continue

All good.



Uhhh... what is happening?!

The reason that rebases can be a pain is that you have to fix conflicts with all of your commits.

Since merging is combining two trees, the conflicts are simpler since you are acknowledging a divergence happened. In rebases, you are firmly denying a divergence ever happened.

```
$ git add [files]
```

```
$ git rebase --continue
```

Ad nauseum...

git rebase --abort

No more! I want to go back to merging!

```
$ git rebase --abort
```

This will abort the rebase and throw away any changes you've made to your files during the rebase.

I don’t want to do this and I hate my team

$ git rebase master -X ours

This will rebase onto master but assume your change supersedes any change on master thereby instantly resolving conflicts

Alternatively:

$ git rebase master -X theirs

git commit --amend

Oh no! I did *git commit -am* but forgot an untracked file! I want to be cool and not create another commit!

```
$ git commit --amend
```

The flag is deceptive. The previous commit isn’t amended, it is replaced completely.

Caveat: You shouldn’t do this if another engineer has pulled from that branch since your previous commit. *git commit --amend* replaces history and fixing this is a huge pain (and this is coming from someone who enjoys fixing rebase conflicts)

I made a huge mistake...

If you want no responsibility for your mistakes, you can reset your branch with

$ git reset --hard [ref]

This will hard reset your current branch to the state of another branch. This means any committed changes between the current HEAD and that branch’s HEAD will be thrown away.

For smaller mistakes:

$ git reset --soft [ref]

Undo all commits between current HEAD and branch’s HEAD. Useful for undoing the previous commit (*git reset --soft HEAD*)

git merge vs git rebase

- Both move code from one branch to another
- Merging is being truthful, rebasing is lying
- Rebasing creates cleaner history, at the cost of possible inconsistency
- Merging creates ‘uglier’ history, at the cost of absolute consistency
- Rebasing re-writes history
- Merging keeps history

How to find the bodies

Rerere will **re**use **re**corded **re**solution.

```
$ git rerere
```

Reflog is a log of all reference changes.

```
$ git reflog
```

Blame is for finding out who to talk to about a change

```
$ git blame -L [line] [file]
```

Bisect is for finding what commit is the cause of a bug

```
$ git bisect
```