Git Rebase

A deep dive

A brownbag workshop at

by Seth House

@whiteinge
seth@eseth.com

Rebase or merge?

- Commits communicate *intent* to your teammates.
  What are you trying to communicate with a given commit/branch?
  - Individual commits that tell the story of a feature addition.
  - A record of when and who updated a branch?
- ...But: don't rebase shared branches.

Amend a commit

```
git commit --amend
```

Amend a commit

```
git commit --amend
```

```
git reset --soft HEAD~1
# git add -p or VSCode open changes view and right-click.
git commit -c ORIG_HEAD
```

Amend a commit

```
git commit --amend
```

```
git reset --soft HEAD~1
# git add -p or VSCode open changes view and right-click.
git commit -c ORIG_HEAD
```

```
# Reuse commit message/authorship.
git commit -C ORIG_HEAD
```

Amend a commit

```
git commit --amend
```

```
git reset --soft HEAD~1
# git add -p or VSCode open changes view and right-click.
git commit -c ORIG_HEAD
```

```
# Reuse commit message/authorship.
git commit -C ORIG_HEAD
```

```
# Amend a commit message
git commit --amend -o
```

`git cherry-pick`

- Sort of a "manual" rebase (order matters).

`git commit --fixup`

Find the commit message:

- Manually.
- Use history-search ref syntax `:/Foo`.
- Use a fuzzy-finder.
- Others can send you a fixup commit. (Similar to a "suggestion" comment in GitHub PRs.)

Rebase

Steps:

1. Does a hard reset on the current branch to the specifed target SHA.
2. Commits (saved to a temporary directory), are then replayed on the new SHA one-by-one.
3. Merge conflicts are possible, as per a merge.

Rebase

Steps:

1. Does a hard reset on the current branch to the specifed target SHA.
2. Commits (saved to a temporary directory), are then replayed on the new SHA one-by-one.
3. Merge conflicts are possible, as per a merge. Use cases:

- Choose a new base SHA to copy commits to a new place in the DAG (and optionally combine/split/modify/drop commits).
- Choose the *same* base SHA to only combine/split/modify/drop commits.

`git rebase`

- Defaults to the upstream tracking branch.
- `--continue` and `--abort` flags while in-progress.

`git rebase --onto`

- Use case: change a branch based off `develop` to be based off `master` instead.

`git rebase --onto`

- Use case: change a branch based off `develop` to be based off `master` instead.
- `<new base> <old base> <branch to modify>`

`git rebase --onto`

- Use case: change a branch based off `develop` to be based off `master` instead.
- `<new base> <old base> <branch to modify>`
- Often a bigger move and more conflict-prone.

Interactive rebase: introduction

- Open with `-i`.
- Abort with no changes, empty changes, or non-zero exit code (`:cq` in Vim, feature request in VSCode)

Interactive rebase: introduction

- Open with `-i`.
- Abort with no changes, empty changes, or non-zero exit code (`:cq` in Vim, feature request in VSCode)

```
git config --global commit.verbose 2
```

Interactive rebase: reorder commits

Interactive rebase: squash and fixup

Interactive rebase: reword

Interactive rebase: pause and edit

- Amend current commit.
- Add brand new commit.

`git rebase --autosquash`

- `git commit --fixup`
- `git commit --amend`
- `git commit --reword`
- Often helpful with `-p` patch commits.