Git
Advanced Functionality I

Gaurav Mitra
Overview

- Merge
- Rebase
- Patch
- Reset
- Checkout
- Revert
- Log

Content adapted primarily from:
https://www.atlassian.com/git/tutorials/advanced-overview/
A new feature is implemented in a dedicated branch
Colleague makes further commits on master branch
These commits are relevant to the feature being implemented and must be added into the feature branch
Two options at this stage
  - Merge
  - Rebase
$ git checkout feature
$ git merge feature
---
$ git merge master feature

- Merge is *non-destructive*: Existing commits on each branch are preserved
- Creates an extra *merge commit* for each merge
$ git checkout feature
$ git rebase master

- Rebase the feature branch onto the master branch
- Moves the entire feature branch to the tip of the master branch
- Effectively incorporates all new feature commits on top of the master branch
- Performs a full re-write of the project history by creating new commits for each original commit that is moved
Advantages:

- Cleaner and more linear project history - removes branching
- No extra merge commit
- Makes navigating the project easier

Disadvantages:

- Safety: Consequences for collaborators
- Traceability: Loss of merge commit history i.e. when was a particular feature incorporated?

Interactive rebasing with -i provides complete control over commit history so that certain commits can be squashed
When to rebase?

Do not use it on *public* branches

Since rebasing creates new commits, git interprets your local branch as having diverged from the original master being used by other collaborators

Only way to sync two versions of master is to do a merge with commits being duplicated

Don’t push rebased master to remote with *--force*
Best practices

- Always use a dedicated branch to implement a new feature addition
- Use interactive rebasing to clean up *private* feature commit history

  ```
  $ git checkout feature
  $ git rebase -i HEAD~3
  ```

- This doesn't move the feature branch but interactively re-writes the 3 commits that follow the base HEAD~3
- No *git merge* alternative
- It does not incorporate upstream changes from master and as a result the resulting feature branch is clean and ready to be merged or rebased entirely
- Using *git merge-base* feature master provides the commit ID of the original base for the feature branch
Patching - Create a patch

- Useful when communicating code changes over a mailbox i.e. simply email a patch to a project maintainer

- Fork a branch from the one the patch is intended for

  ```bash
  $ git clone git://gitlab...
  $ git checkout -b feature
  ```

- Make the necessary changes and commits to the feature branch

- Generate the patch including all the commits in feature branch ahead of the master branch

  ```bash
  $ git format-patch master --stdout > feature.patch
  ```

- Send the patch to the project maintainer

---

Maintainer receives the feature.patch file

Of course you wouldn’t blindly apply a patch from an unknown person! Have a look at what the patch is trying to do and test run it:

```sh
$ git apply --stat feature.patch
$ git apply --check feature.patch
```

Once convinced that a patch is legitimate, apply it

```sh
$ git apply feature.patch
OR
$ git am --signoff < feature.patch
```

git am signs the patch application with your public key so authorship of the commit can be verified at a later stage!

git am was created to read email from a mailbox, split them into commit log messages, author info and patches, then apply the patches!

---