Lecture 8
Integration-Manager Workflow and Rebasing

This is Git. It tracks collaborative work on projects through a beautiful distributed graph theory tree model.

Cool. How do we use it?

No idea. Just memorize these shell commands and type them to sync up. If you get errors, save your work elsewhere, delete the project, and download a fresh copy.

Sign in on the attendance sheet!
Remember the Centralized Workflow?

Problem: Every developer needs **push access** to the shared repository!

Integration-Manager Workflow

Github/“The cloud”

Local Computer

blessed repository

developer public

developer public

integration manager
developer private
developer private
Step 1. **Fork** the public repository

(make your own public copy)
Step 1. **Fork** the public repository
Step 2. Clone your public repository

$ git clone https://github.com/aperley/Autolab.git
Step 3. Create a **feature branch** and make some commits

$ git checkout -b my-feature
$ <do some work>
$ git commit -am "add my feature"

Then **push** your feature branch to your public repository

$ git push origin my-feature
Step 4. Create a pull request

Open a pull request
Create a new pull request by comparing changes across two branches. If you need to, you can also compare across forks.

Able to merge. These branches can be automatically merged.

Please review the guidelines for contributing to this repository.

Edit README
The integration manager can inspect and pull in your changes

As the integration manager:

$ git remote add aperleys-fork
https://github.com/aperley/Autolab.git
$ git checkout aperleys-fork/my-feature

If it looks good:

$ git checkout master
$ git merge aperleys-fork/my-feature
$ git push origin master
The integration manager can inspect and **pull in** your changes.
You need to keep your fork up to date

In the private developer repo

$ git remote add upstream
https://github.com/autolab/Autolab.git

$ git fetch upstream

$ git checkout master

$ git merge upstream/master

$ git push origin master
You need to keep your fork up to date
Git Rebase: Squashing Commits
Squashing Commits

Scenario:
Made some commits on a feature branch but want to “clean it up” before making a pull request or merging to master
Squashing Commits

$ git rebase -i master

Begins an interactive rebase of all of the commits since the branch split off of master.
Interactive Rebase

pick 40fd4c0 Implement the feature
pick 292a385 Fix this one tiny bug
pick 88e3570 Fix spelling mistake
pick 05fadb6 Add the tests
pick 86f3760 Fix that one failing test

# Rebase 8e0ff21..86f3760 onto 8e0ff21
#
# Commands:
# 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
#
# These lines can be re-ordered; they are executed from top to bottom.
# If you remove a line here THAT COMMIT WILL BE LOST.
# However, if you remove everything, the rebase will be aborted.
# Note that empty commits are commented out
Decide what you want to squash onto the commit above (before)

```
pick 40fd4c0 Implement the feature
s 292a385 Fix this one tiny bug
s 88e3570 Fix spelling mistake
pick 05fadbc Add the tests
s 86f3760 Fix that one failing test

# Rebase 8e0ff21..86f3760 onto 8e0ff21
#
# Commands:
# 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
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
#
# Note that empty commits are commented out
```
Now we need to edit the commit message for the first squash group

```bash
# This is a combination of 3 commits.
# The first commit's message is:
Implement the feature

# This is the 2nd commit message:
Fix this one tiny bug

# This is the 3rd commit message:
Fix spelling mistake

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# rebase in progress; onto 8e0ff21
# You are currently editing a commit while rebasing branch 'my-feature' on '8e0ff21'.
# Changes to be committed:
# modified: one.txt
```
Same for the second squash group
We are left with a new commit history

```
$ git log --pretty --oneline master...
09225c6 Add the tests
af5d8ad Implement the feature
```
What happened?

We **rewrote history** by replaying the patches for each commit.