Durham University

Version Control: Collaboration

COMP51915 – *Collaborative Software Development*
Michaelmas Term 2024

Christopher Marcotte[1]

[1] For errata and questions please contact christopher.marcotte@durham.ac.uk

Outline

- Pull Requests
- Team Management
- Distributed Workflows
- Summarizing `git`

Learning Outcomes

By the end of this section you should:

- Understand when and how to use a Pull Request,
- Critically understand some distributed collaboration workflows

Review: `git push` & `git pull`

We've already discussed how to use `git push` & `git pull`.

- `git push` reflects your local changes in a remote repository, and
- `git pull` updates your local repository with the changes in the remote.

But these only work in the typical way if the remote has a matching branch.

Review: `git push` & `git pull`

We've already discussed how to use `git push` & `git pull`.

- `git push` reflects your local changes in a remote repository, and
- `git pull` updates your local repository with the changes in the remote.

But these only work in the typical way if the remote has a matching branch.

One or the other can fail due to merge conflicts. You should call `git pull --rebase` to *rebase* your local on the commits to the remote.

Pull Requests

Let's imagine you're in a team, and you have been charged with solving an issue.

You fork the public repository at version `v1.0` into your local, and fix the problem and commit it to your local repository.

Pull Requests

Let's imagine you're in a team, and you have been charged with solving an issue.

You fork the public repository at version `v1.0` into your local, and fix the problem and commit it to your local repository.

In `git`, you might use the command

```
git request-pull v1.0 <project_url> <local_branch_name>
```

Personally, I think the GitHub UI is easier – it prompts you for each of these pieces of information.

Pull Request Etiquette

As will all things collaborative – this is a people-problem, and technical solutions will only go so far.

You may be reviewed, or may be asked to review a contribution before merging into a project.

This is an opportunity to be direct and both give and take criticism gracefully.

Pull Request Etiquette

As will all things collaborative – this is a people-problem, and technical solutions will only go so far.

You may be reviewed, or may be asked to review a contribution before merging into a project.

This is an opportunity to be direct and both give and take criticism gracefully.

Importantly, pull request review is *not* an opportunity to be rude or insulting to people; least of all people who are contributing their time, effort, and expertise to your project.

Team Management

Frequently you will find yourself working in a team to develop a piece of software – like you will for this coursework.

If you are collaborating on GitHub, then you have a long list of tools to use to organize everyones efforts.

Issues, Pull Requests, Tags – these are all effective means of labeling bugs, fixes, and particular focuses of the development process, which makes the repository more legible to a team.

Distributed Workflows

Any number of distributed workflows are possible. Some common ones:

- **centralized workflow**
- **integration-manager**
- **dictator & lieutenants**

Distributed Workflows

Any number of distributed workflows are possible. Some common ones:

- **centralized workflow** – one central repository hosts the accepted version of the code base, and every developer synchronizes their local version with it to maintain consistency.
- **integration-manager**
- **dictator & lieutenants**

Distributed Workflows

Any number of distributed workflows are possible. Some common ones:

- **centralized workflow**
- **integration-manager** – one developer is selected to have (read-only) access to everyone's repository; this developer is then responsible for managing pull requests from everyone's repositories into the central repository.
- **dictator & lieutenants**

Distributed Workflows

Any number of distributed workflows are possible. Some common ones:

- **centralized workflow**
- **integration-manager**
- **dictator & lieutenants** – for very large projects, you might find the earlier options limiting; in this workflow, segments of the repository are under the stewardship of different lieutenants, whose work is integrated by the integration manager.

Distributed Workflows

Any number of distributed workflows are possible. Some common ones:

- **centralized workflow**
- **integration-manager**
- **dictator & lieutenants**

Each of these places further distance between the *one true repository* and the developers doing the work, introducing more hierarchy into the organization, and thus less *productive* work per developer effort.

I.e., if you have **5** collaborators – it's less wasted effort if everyone manages consistency themselves. But if you have **100** collaborators, it may be better to delegate that effort to a handful of people.

Summarizing `git`

We've covered a number of topics today, but not all[2], and most without the detail needed to achieve deep understanding.

I encourage you to experiment with `git` development processes, and find what works for you in a development community.

`git` is a lot like *C* – very easy basics, very difficult to understand the implications of how they combine.

[2] E.g., look up `git rebase` and `git cherry-pick`