ENGR 301 *Project Management*

Lecture 10 — git III

James Quilty

School of Engineering and Computer Science
Victoria University of Wellington

Introduction

Today's lecture concludes our discussion of basic usage of git with branches. References you may find useful are:

- git documentation `https://git-scm.com/docs`
- Pro Git book
  `https://git-scm.com/book/en/v2`
- Git: Mastering Version Control (O'reilly)
  `https://bit.ly/4accAng`
- Version Control (Git)
  `https://missing.csail.mit.edu/2020/version-control/`

Branches

Branches are an integral part of most SCM tools

- `git branch` and `git switch` are the main commands
- `git branch` allows branch creation, renaming, deletion, etc.
- `git checkout` will switch to a branch and update the working copy
- `git switch` (introduced in 2.23) is a “friendlier”, and thus recommended, version of `git checkout`

Branching Strategies

A *branching strategy* is necessary to keep the situation with `git` from spiralling out of control. There are *many* branching strategies, the one we will use a simplified version of GitLab Workflow. Branches are:

- created from an *Issue* via a *Merge Request*
- directed toward a *single purpose*
- typically *short-lived* on the time scale of the project
- schematics and PCB layout branches will be longer-lived — *exceptio probat regulam*

Merging Branches

Branches are all good-and-well, but how do we merge branches?

- Use GitLab's Merge Request support to merge branches to `main`
- Avoid `git merge` manually unless you know what you're doing
- Never `git merge main` on a branch

Avoid Foxtrots! `https://bit.ly/43liksE`

Interactive Rebasing

`git rebase` is a very powerful command which can perform *several* different actions. Let's look at the simplest application: interactively reordering, dropping and squashing *local* commits

- on a branch invoke by `git rebase --interactive`
- use your editor to reorder, squash, drop, etc. commits
- *have confidence*: you *should* be able to recover from mistakes by using the RefLog

Fundamental Rule of Rebasing: rebasing commits which have not been pushed to remote is *safe*, but rebasing commits pushed to remote can disrupt others.

Cherry-picking

Sometimes you'll want to move a commit, or series of commits, from one branch to another.

- On the target branch use `git cherry-pick <sha>`
- Use `git cherry-pick --no-commit <sha>` if you just want the files without the commit
- Perform an interactive rebase on the source branch to drop moved commits.

The local vs. remote rules apply: local is *safe[-ish]* but working with commits pushed to remote should be approached with caution.

Pruning Branches

Keeping your local copy tidy sometimes requires some manual intervention, particularly regarding branches.

```
git remote prune origin
git branch --all
git branch --delete <branch>
```

will help remove merged and deleted branches from your local copy.

Again, reference to a git GUI like GitKraken will be helpful.