ENGR 301 *Project Management*

Lecture 8 — git II

James Quilty

School of Engineering and Computer Science
Victoria University of Wellington

Basic git

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

- Pro Git `https://git-scm.com/book/en/v2`
- Git: Mastering Version Control `https://learning.oreilly.com/library/view/git-mastering-version/9781787123205/`
- Version Control (Git) `https://missing.csail.mit.edu/2020/version-control/`

git's Workflow

Basic concepts

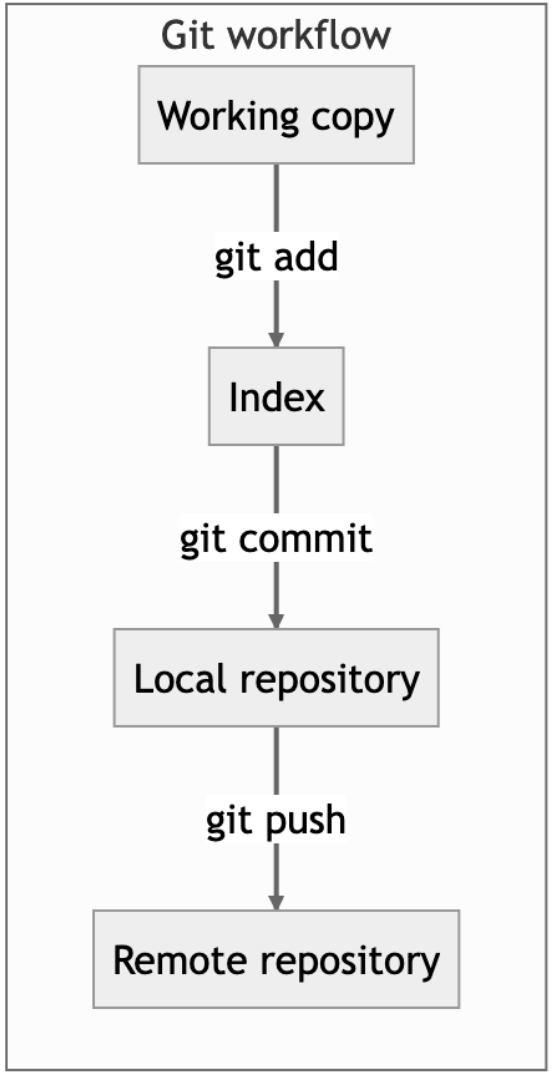
Working copy: the files you see and edit

Index: a.k.a. the "staging area"

Local repo: your clone of the remote repo

Remote repo: the *Single Source of Truth*

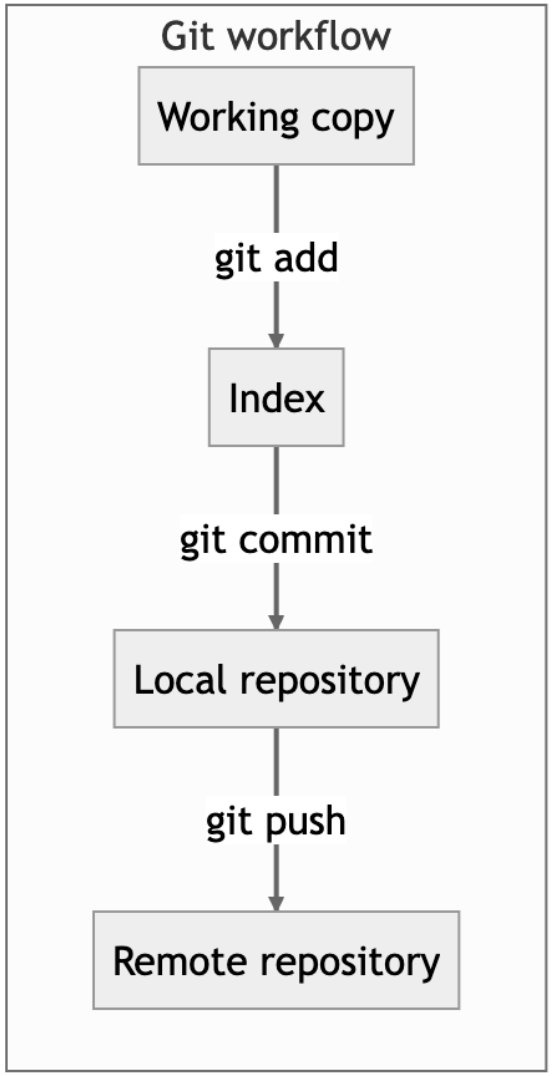
Typically centrally served, e.g. GitLab, GitHub, Bitbucket, *etc.*



Using git's workflow Effectively

Guidelines

- Greatest freedom at the start, least freedom at the end
- Use staging to collect changes for a commit *while working*
- Local commits are easy to change, both content and order
- Remote is *shared* and commits should be changed *with caution*



Working with the Index (staging)

Basic concepts:

- Files are *untracked* until *tracked* using `git add`
- Untrack files with `git rm` *but* will also remove from working copy
- Remove files from staging with `git restore --cached`
- Add changed files with `git add` and piece-by-piece with `git add --patch`

Finding what's changed

Two main commands:

- `git status` shows what git knows about changes
- By default, ignored files are not shown by `git status`
- `git diff` shows the changes between the working copy and staging
- `git diff --cached` shows the changes between staging and local

Restoring changes

Can restore (undo) changes in the Working Copy and Index (staging area) [see also GitKraken]:

- `git restore` will undo changes in the Working Copy tree or a specific file
- `git restore --cached` will undo changes added to the staging area, i.e. undo the `git add`
- `git reset` is an older and *far more dangerous* command for undoing changes
- `git reset` flags: `--soft`, `--mixed` and `--hard`
- `git reset` will also operate on *commits*

Committing to Local

The basic command is `git commit`

- `git commit [--message] [--message]`
- `git commit --amend` will merge changes in the index with the most recent commit
- `git commit --amend --no-edit` will do the same but not ask for an update to the commit summary/title

Pushing and Pulling

- `git push` merges your commits from *local* to *remote*
- `git pull --rebase` merges commits from *remote* to your *working copy*

A common situation on pull is to receive an error message like:

```
$ git pull –rebase
error: cannot pull with rebase: You have unstaged changes.
error: please commit or stash them.
```

What is “stashing“?

Pushing and Pulling

A common situation on push is to receive an error message like:

```
$ git push
To gitlab.ecs.vuw.ac.nz:Courses/SWEN326/SWEN326.git
 ! [rejected]        41-... -> 41-... (fetch first)
error: failed to push some refs to '<path-to-repo>'
hint: Updates were rejected because the remote
hint: contains work that you do not have locally.
hint: This is usually caused by another repository
hint: pushing to the same ref. You may want to first
hint: integrate the remote changes (e.g., 'git pull ...')
hint: before pushing again. See the 'Note about
hint: fast-forwards' in 'git push --help' for details.
```

Need to pull before pushing. Always use the `rebase` merge strategy!

Stashing changes

Basic concepts:

- The stash is a queue, thus: `git stash [push]` and `git stash pop`
- `git stash list` shows the list of stashes;
  select a stash with `git stash pop stash@{<number>}`
- Show the contents of a stash with
  `git stash show [stash@{<number>}]`
- Apply the contents of a stash without removing it with
  `git stash apply [stash@{<number>}]`
- Give stashes better messages with
  `git stash --message "..."`

Pushing and Pulling

A common situation when changes have been made “outside” git:

```
$ git pull –rebase
...
error: The following untracked working tree files would be overwritten
by merge:
<some files>
Please move or remove them before you merge.
Aborting
```

git hooks

`git` hooks execute scripts on specific events.

pre-commit `https://pre-commit.com/`

> *"A framework for managing and maintaining multi-language pre-commit hooks."*

> *"...we recognised that sharing our pre-commit hooks across projects is painful."*

- pre-commit not limited to one hook only
- Incorporated in the data-recorder repository