ENGR 301 Lecture Notes

Source Control Management with `git`

These notes focus on the practicalities of setting-up and using `git` for source control management, a deeper understanding is left for another lecture.

- `git` on the command line
- GitKraken

Source Control Management

Source Control Management (SCM), also known as version control, has been around for a long time: the first full system was 1972 with precursor systems as far back as 1962.

Conceptually Simple:

- systematically store previous versions of files;
- each file has specific version or *revision* identifier;
- allow examination and retrieval of earlier versions;

Source control management *is not a backup system* however.

Centralised and Decentralised

Centralised: repository files served from a server, local copies of files only.

Decentralised: a copy of the entire repository is held locally *as well as* local copies of the files. There may be *many* remote copies of the repository.

Contemporary practice is that even decentralised SCM systems have a centrally served repository.

Why `git`?

Why are we learning `git`? A couple of reasons.

First, because it's effectively the only SCM system used today:

> [Stack Overflow Developer Survey 2022]
>
> **Version control systems**

> No other technology is as widely used as Git. Especially among Professional Developers. But for those learning to code, 17% still do not use a version control system.

Second, for all the reasons in the blog post *10 things I hate about Git* by Steve Bennett:

1. Complex information model
2. Crazy command line syntax
3. Crappy documentation
4. Information model sprawl
5. Leaky abstraction
6. Power for the maintainer, at the expense of the contributor
7. Unsafe version control
8. Burden of VCS maintenance pushed to contributors
9. Git history is a bunch of lies
10. Simple tasks need so many commands

Note: this blog post received 1,018 comments totalling more than **120,000 words**.

Setup

Download and install `git` if you don't have it already. On Windows, install the Windows Subsystem for Linux.

SSH Key Setup

Set up a SSH key to use with `git clone` so repositories can be cloned with SSH. This avoids the problem with HTTPS cloning: having to re-enter your password on the command line for access to the remote repository. See: https://docs.gitlab.com/ee/user/ssh.html.

In Linux and macOS, SSH keeps its user configuration files at `~/.ssh` (the `~` means "the user's home directory"). See if you have some files

```
$ ls -1 ~/.ssh
config
id_rsa
id_rsa.pub
known_hosts
```

If you don't have a key (they're not necessarily created by default) then create one with the `ssh-keygen` command; GitLab :

```
ssh-keygen -t ed25519 -C "your.email@ecs.vuw.ac.nz"
```

It's wisest to enter a password for your SSH key! If you become tired of entering it repeatedly, look into how to activate `ssh-agent` on your system.

Need to copy the *public* key to our profile in GitLab https://gitlab.ecs.vuw.ac.nz/-/profile/keys. Copy the *public* key information from the terminal into the field in GitLab's setting page. To do this manually:

```
less ~/.ssh/id_ed25519.pub
```

Alternate commands are detailed at https://docs.gitlab.com/ee/user/ssh.html#add-an-ssh-key-to-your-gitlab-account.

Note: GitKraken can also generate SSH keys for you, but it requires some configuration.

Configuration

The global configuration file is `~/.gitconfig` and contains things like your name and email address (logged when interacting with `git`). There are some things you'll want to set

```
git config --global user.name “Firstname Lastname”
git config --global user.email “Firstname.Lastname@ecs.vuw.ac.nz”
```

Note: configuration is a per-account kind of thing (stored in dot-files) so it must, effectively, be set on every computer you use.

Ignoring Files

There are two files which contain *patterns* for files which you *don't* want to place under version control

- a `.gitignore` file in the *working directory*
- the global `~/.gitignore_global` file.

Best to use simple wildcard patterns.

Generally want to ignore:

- intermediate files which are generated from source
- final results, text or binary, i.e. "artefacts"
- system files

Working with files

See https://docs.gitlab.com/ee/topics/gitlab_flow.html#git-workflow

The main way to obtain files from the remote repository is to *clone* it

```
git clone <ssh>
```

The main way to *update* your working copy is to *pull*

```
git pull --rebase
```

Note: Always use `--rebase` !

Obtaining the History of a specific file

```
git log --all --full-history -- <path>
```

Moving a Commit from One Branch to Another

On the target branch use `git cherry-pick <sha>` (possibly with the `--no-commit` flag) and then perform an interactive rebase on the source branch to delete the moved commit.

Selecting Which Parts of a Modified File to Commit

Use `git add --patch` for an interactive add which allows the selection/splitting of hunks to add.

Commit Messages

1. How (and why!) to keep your Git commit history clean (GitLab Blog)
2. How to keep your commits atomic
3. How to write a git commit message
4. GitLab Documentation on Closing issues with `git` commits:
    - Automatic issue closing
    - Issue closing patterns

Restoring a specific file to a previous version

More than one way to do this [1,2]:

- `git checkout <commit_hash_id> -- <file_path>`
- `git restore --source <commit_hash_id> <file_path>`

The contemporary way is to use `git restore` [3] although note the warning from the documentation:

```
THIS COMMAND IS EXPERIMENTAL. THE BEHAVIOR MAY CHANGE.
```

Pitfall: if the file has been moved to a different path in the tree subsequent to `<commit_hash_id>` the checkout/restore will put the file at the *old* path from `<commit_hash_id>`.

1. How do I reset or revert a file to a specific revision? https://stackoverflow.com/questions/215718/how-do-i-reset-or-revert-a-file-to-a-specific-revision
2. How to reset a file to an old revision https://gitbetter.substack.com/p/how-to-reset-a-file-to-an-old-revision
3. `git restore` https://git-scm.com/docs/git-restore

Opinion

1. 10 things I hate about Git by Steve Bennett https://stevebennett.me/2012/02/24/10-things-i-hate-about-git/
2. A year of using Git: the good, the bad, and the ugly https://ikriv.com/blog/?p=1905
3. A Short History of Git https://git-scm.com/book/en/v2/Getting-Started-A-Short-History-of-Git
4. A successful Git branching model https://nvie.com/posts/a-successful-git-branching-model/ **Note:** not recommended for students to follow, just history
5. git man page generator https://git-man-page-generator.lokaltog.net

    - git-hoist-remote
    - git-sponsor-status
    - git-ignore-backup
    - git-flatten-clone
    - git-fornicate-bug-report
