Source control

Dr Peadar Grant

January 31, 2024

Contents

1 Motivation

Have you ever:

Broke something that was working and been unsure how to fix it?

Needed to see differences compared to a previously working version?

Wanted the history of how code / data got to its current state?

Needed to show evolutionary progress of work for academic integrity reasons.

Hoarded files in a uncontrollable sprawl?

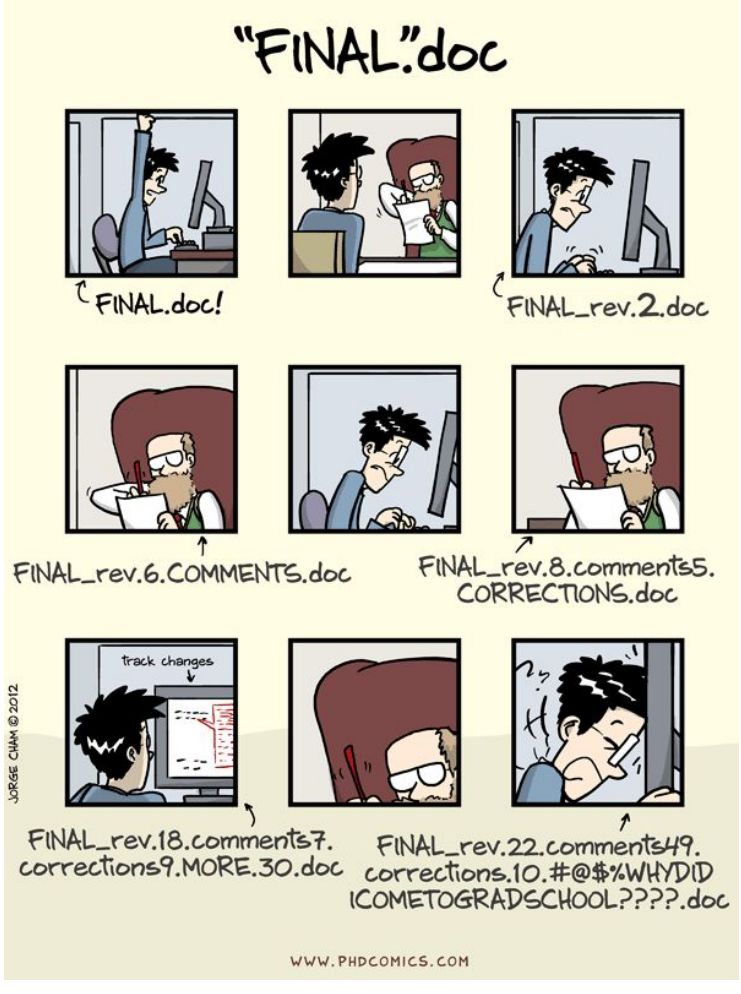

Figure 1: Versioning without source control

2 Source control

- Source Control Systems keep all files in a Code 'Repository'.
- A code repository is a file system that stores all the files.
  - Adds a time dimension to the file system.
- Atomic snapshots called "commits".
- The repository stores every committed version of each file.
- Can show differences from, or revert to, previous versions.
- Version history is tracked with explanatory messages.
- Later on (very soon!) can work with remote repositories.

2.1 Available systems

- Source control systems have been around for many years.
- Historically: RCS, CVS
- Still used: SVN, others...
- Git is by far the most commonly used tool.

3 Git

From the git website:

> Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.
>
> Git is easy to learn and has a tiny footprint with lightning fast performance. It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like cheap local branching, convenient staging areas, and multiple workflows.

3.1 Starting off

- Git works at a per-project level
- Primarily used via Command line
  - Support built into many IDEs and other tooling however!
- Use `git init .` in a folder to start tracking it
  - Includes subfolders as well
  - Creates a `.git` folder where history is stored
- Git creates a `main` or `master` *branch* by default.

3.2 Recording changes

- Use `git status` to show the project status at any time
  - Use it liberally as you try out git!
  - File paths relative to current working directory
- Untracked files have no history in git (yet)
- Use `git add filename.txt` to track file
  - Not committed yet, instead in **staging area**.
- Use `git commit -m "message"` to **commit** changes.

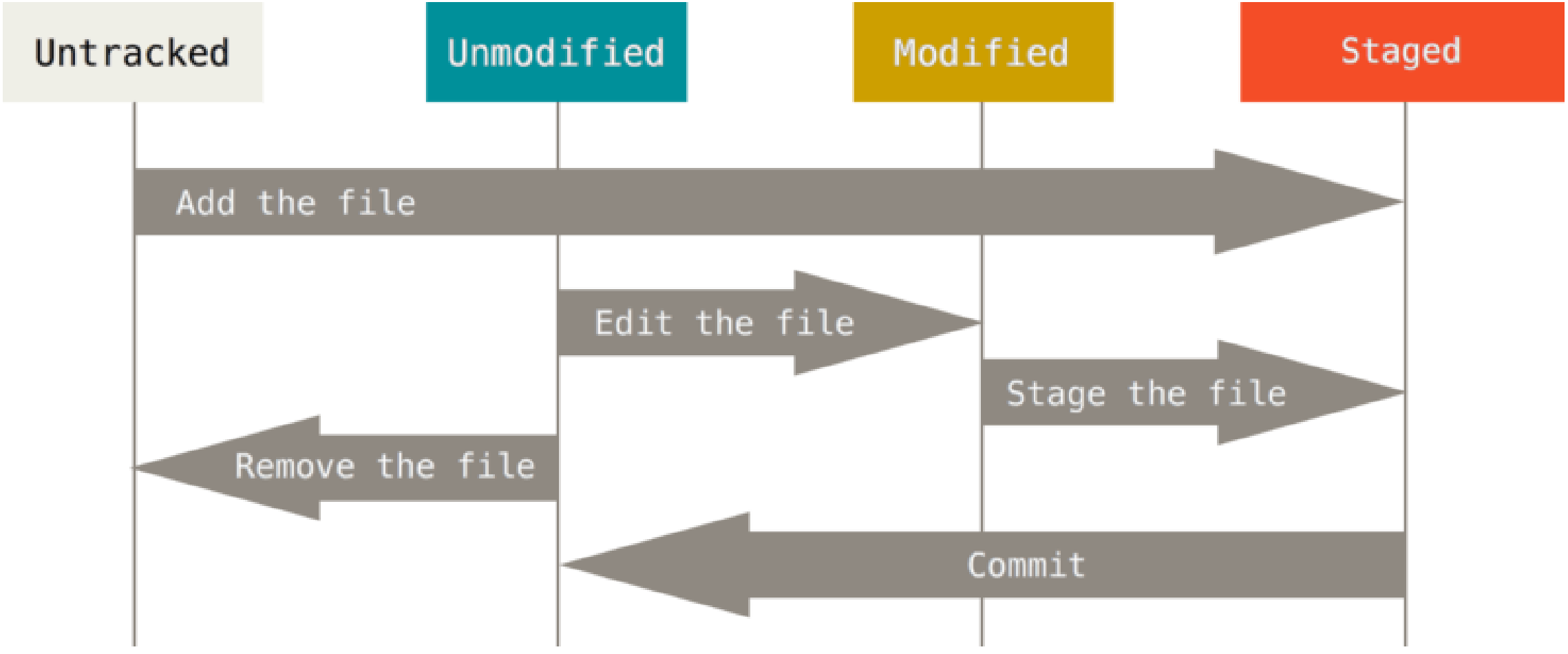

Figure 2: File lifecycle

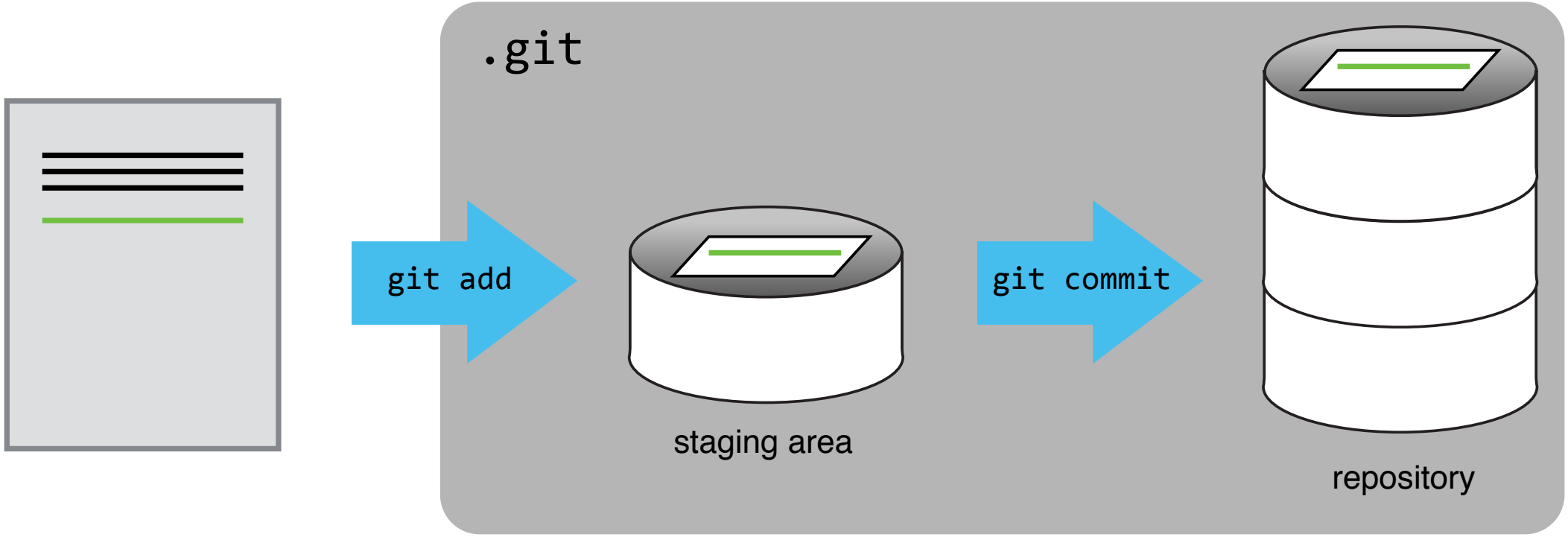

Figure 3: Git Staging Area (swcarpentry)

3.3 Showing differences

- `git diff filename.txt` will show changes to file since last committed.

3.4 File moving and deletion

- Git tracks removal of files using `git rm`.
  - Will remove the file on the filesystem
  - Will also record the change in the next commit
- Similarly can move or rename using `git mv`.
- Copying a file is different:
  - Use `cp` as normal to copy the file, then
  - `git add` to track it

3.5 History

- `git log` will show history.
- The most recent commit is known as the `HEAD`.
  - Previous commit is `HEAD~1`.
  - Similarly `HEAD~2`, `HEAD~3` etc.
- Can revert using `git checkout commitid`
  - Be careful making changes from this state!
- Can recover individual file using `git checkout commitid filename.txt`
  - This will just appear as a modified file in `git status`

4 Ignoring files

- Source control systems primarily manage text files
- Some things don't (generally) belong in source control:

  Large binary files like Word documents, Image files, videos etc.

  Generated output files, since we can re-generate them from source.
- Use `.gitignore` to ignore certain file types
  - Normally in top folder of the project
  - Can use `*` wildcards, e.g. `*.pdf`.