Introduction to version control with Git

Dr Maria del Mar Quiroga

Senior Research Data Specialist and HASS Taskforce Lead

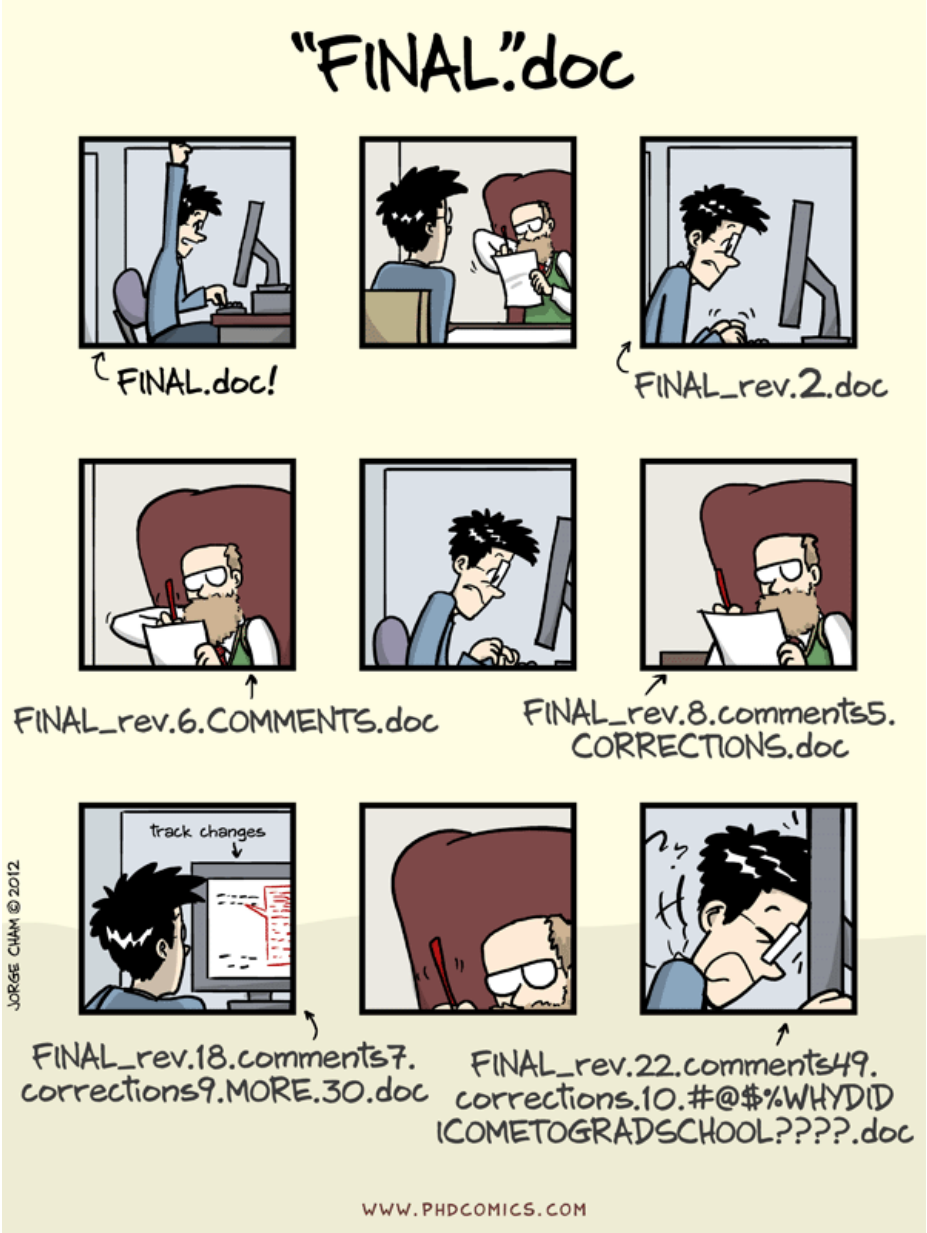
Melbourne Data Analytics Platform

University of Melbourne

Today we meet on the untreatied lands of the Wurundjeri Woi-wurrung and Bunurong peoples.

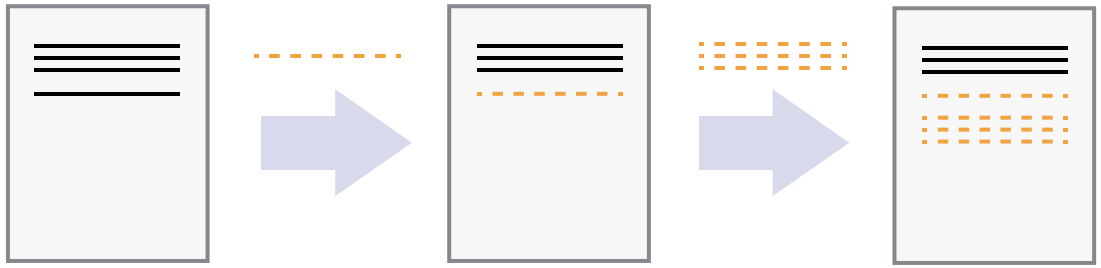
I acknowledge the Ancestors of this place. I acknowledge the stolen land and the stolen children. I acknowledge the stolen wealth of the Wurundjeri Woi-wurrung and Bunurong peoples. I acknowledge all those who continue to struggle for justice and recognition for Indigenous people.

At MDAP, we pay respect to the essential role that Indigenous led research, data governance, and knowledge sharing plays in empowering First Nations communities to shape their own local data priorities and sovereignty, and in our own research journeys.

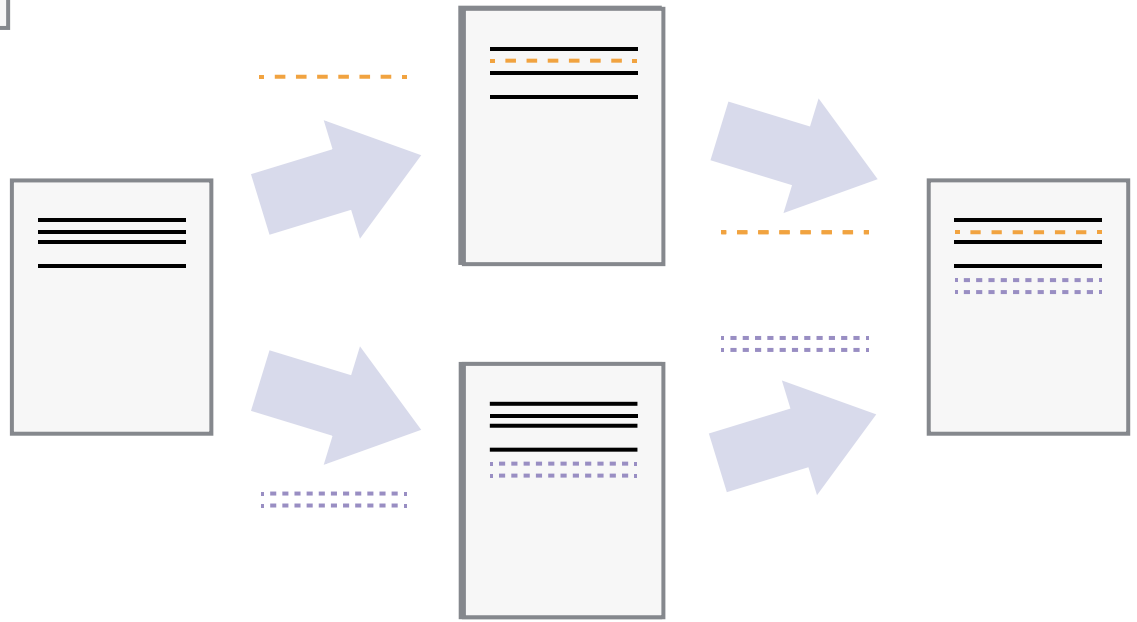


http://phdcomics.com/comics/archive.php?comicid=1531

Enabling data-led research

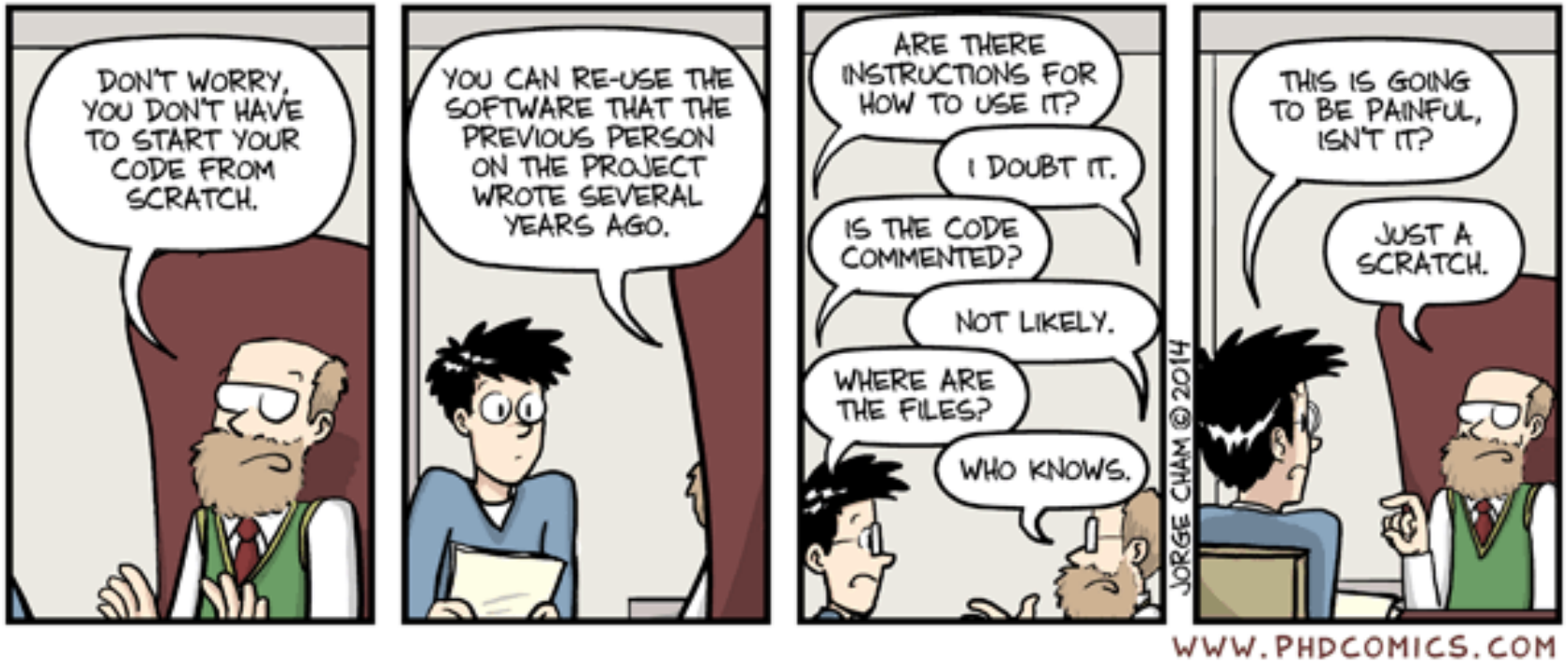


It’s like an unlimited ‘undo’



Allows many people to work in parallel

Code: Stata, R, Python, MATLAB, JavaScript, C, C#, C++, FORTRAN, perl, etc…



http://phdcomics.com/comics/archive.php?comicid=1689

Git: a distributed version control system

THE UNIVERSITY OF MELBOURNE

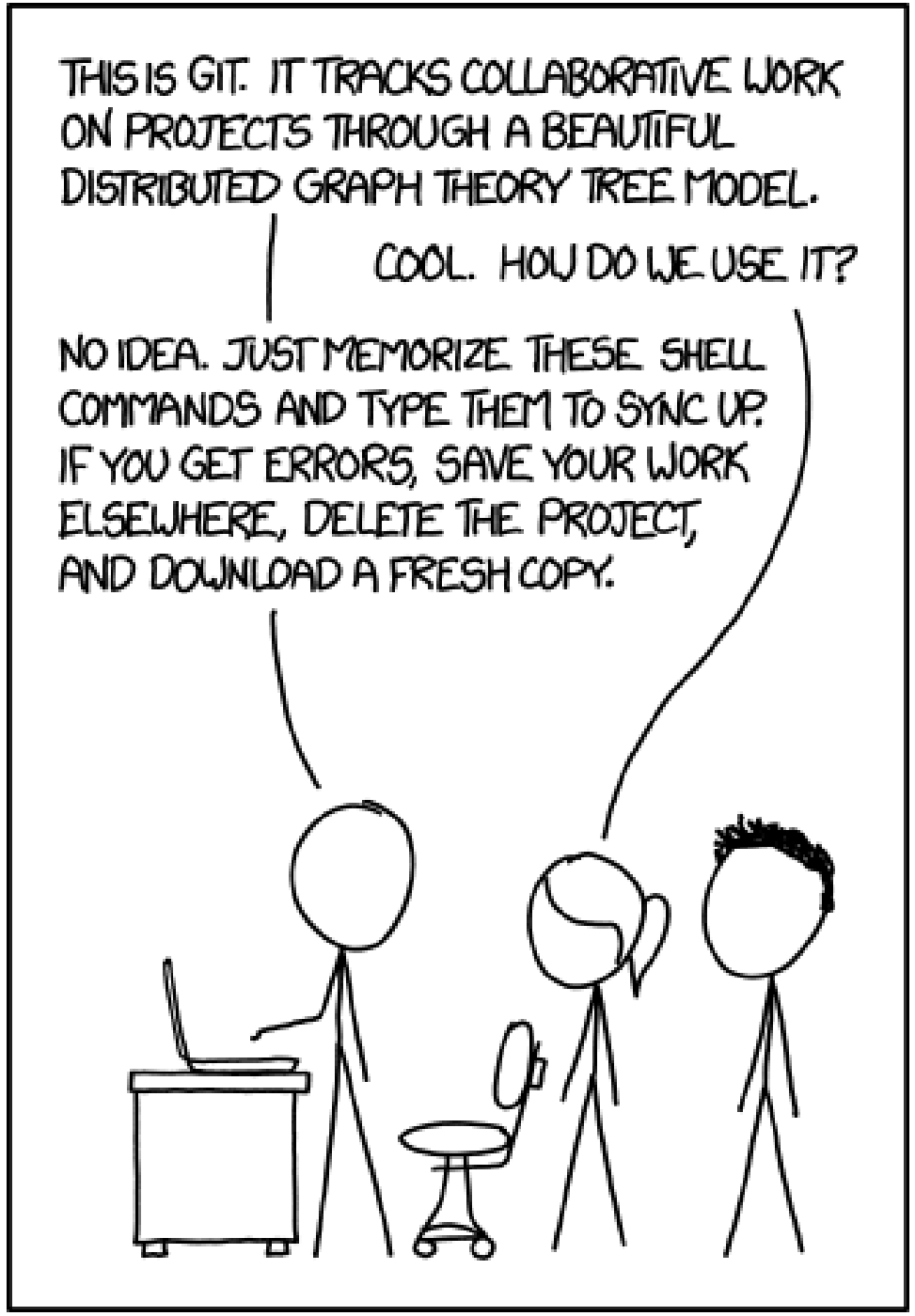
“Distributed” means that the complete history is contained in each instance of a repository

Git basics

- Git is anything but…
- Made for text files (i.e., doesn't work on binary files such as .docx, .xlsx, .pptx)
- How do I get set up to use Git?
  - Download from: https://git-scm.com/, follow all default settings

Where does Git store information?

- Repository: A storage area where a version control system stores the full history of commits of a project and information about who changed what and when (different to a data repository!)
- Hosting services: GitHub cloud (Microsoft), Bitbucket cloud or self-hosted (Atlassian), GitLab cloud or self-hosted
- Main differences:
  - where repositories are stored
  - whether they're private/public/internal
- UoM has self-managed GitLab (activate your account by logging in at https://gitlab.unimelb.edu.au/) BUT is moving towards GitHub…



https://xkcd.com/1597/

Git GUI clients

Command line can be intimidating, and the built-in GUI is not very user friendly…

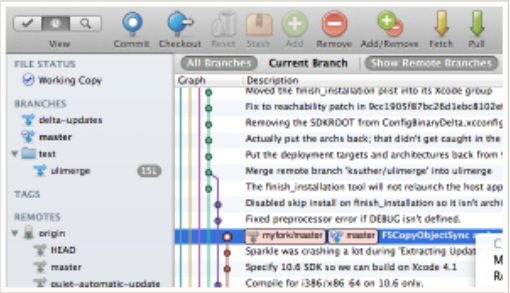
There are heaps of third-party options: https://git-scm.com/downloads/guis/

GUI Clients

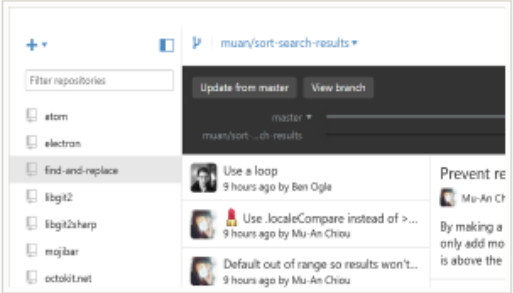
Git comes with built-in GUI tools for committing (git-gui) and browsing (gitk), but there are several third-party tools for users looking for platform-specific experience.

If you want to add another GUI tool to this list, just follow the instructions.

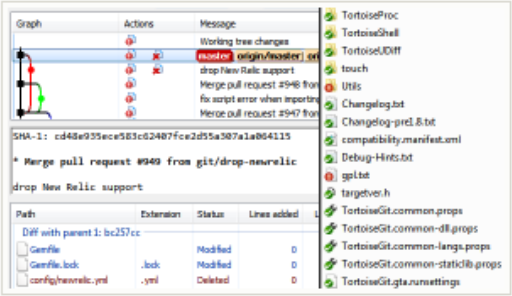
All | Windows | Mac | Linux | Android | iOS



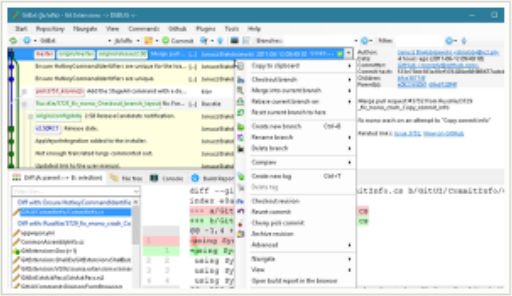
SourceTree
Platforms: Mac, Windows
Price: Free
License: Proprietary



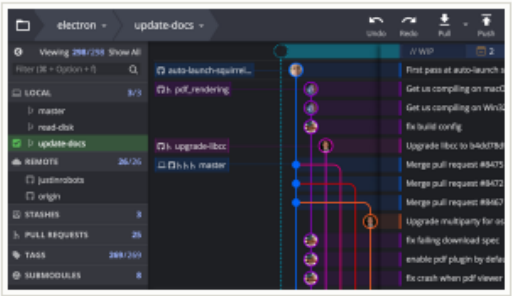
GitHub Desktop
Platforms: Mac, Windows
Price: Free
License: MIT



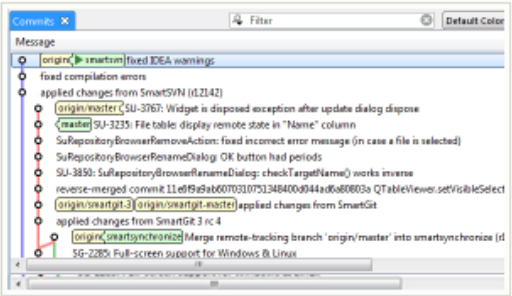
TortoiseGit
Platforms: Windows
Price: Free
License: GNU GPL



Git Extensions
Platforms: Linux, Mac, Windows
Price: Free
License: GNU GPL



GitKraken
Platforms: Linux, Mac, Windows
Price: Free / $29 / $49
License: Proprietary



SmartGit
Platforms: Linux, Mac, Windows
Price: $79/user / Free for non-commercial use
License: Proprietary

So, what's the difference between git, GitHub, and GitKraken, and why do I need all three?

- **git:** software that enables you to apply version-control to any project, i.e., make a repository

- Git **hosting**: your repository will not be backed up, shareable, or collaborative unless you host it somewhere!

  - Cloud-based hosting: GitHub, GitLab (cloud), BitBucket (cloud)

  - On-campus server hosting: Unimelb GitLab – not for long!

- Git **client** (GUI): facilitate interactions with repositories

  - GitHub Desktop, GitKraken, SourceTree, IDEs like RStudio or VSCode have their own built-in client…

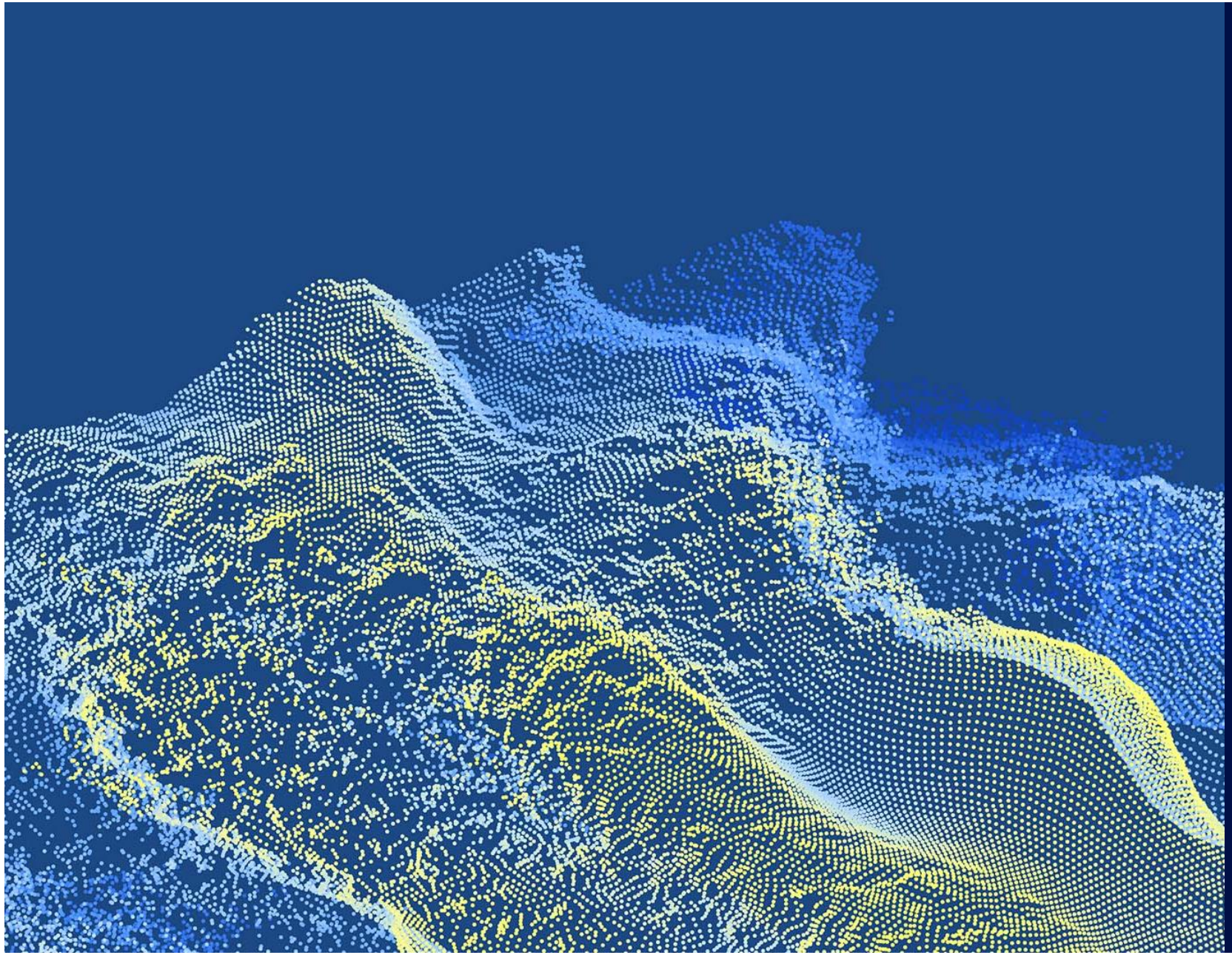
What should and shouldn’t be tracked with Git?

Remember: Git is *distributed,* which means the whole history is contained in every instance!

Better to exclude:

- Big data files (use GitLFS or DVC instead)
- Passwords, private ssh keys, etc.
- System files (i.e., Mac's .DS_Store)
- App configuration files (i.e., app.config, .env, etc.)
- Build artifacts (i.e., *.pyc)
- Installed dependencies (i.e., node_modules)
- Non-documentation and personal text files (i.e., todo.txt)
- Application data and logs (i.e., *.log, *.sqlite, etc.)

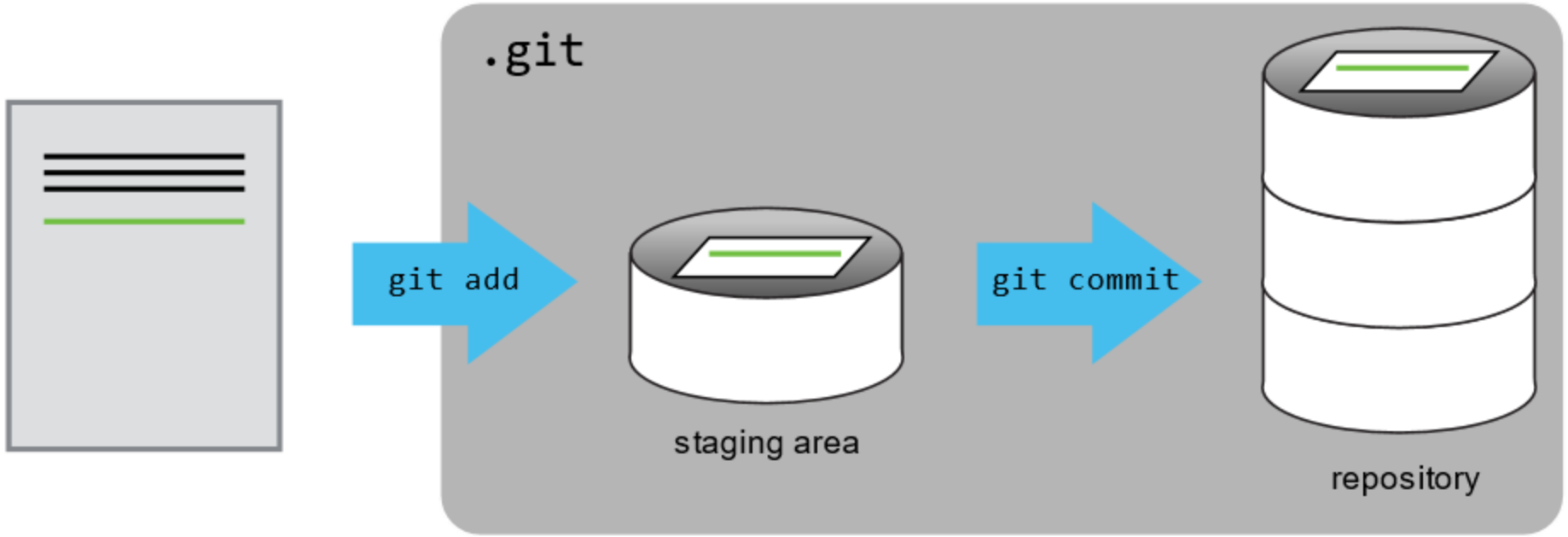
Set up .gitignore first thing (works prospectively, not retrospective)



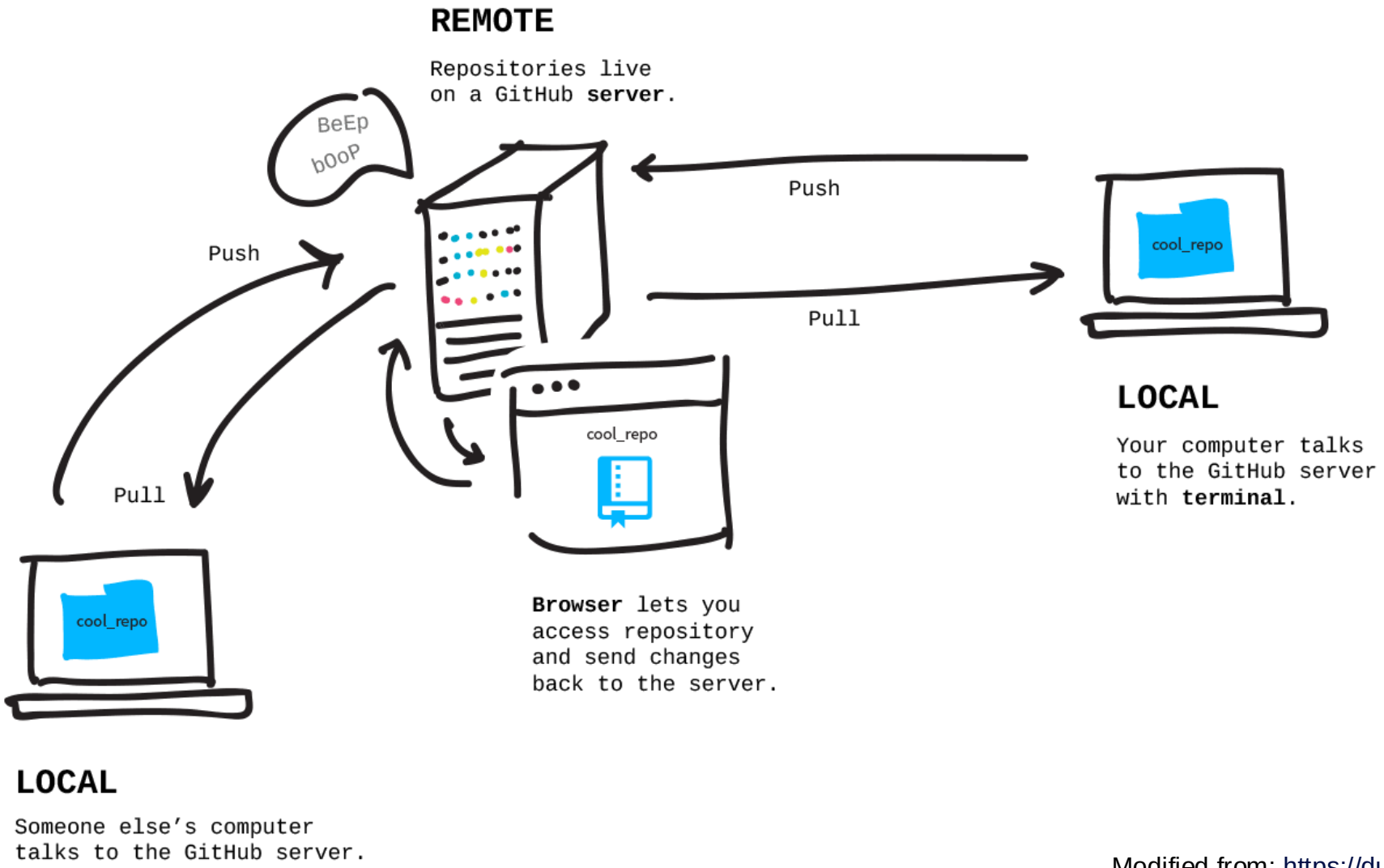
THE UNIVERSITY OF MELBOURNE

Let’s tackle some jargon!

Stage and commit

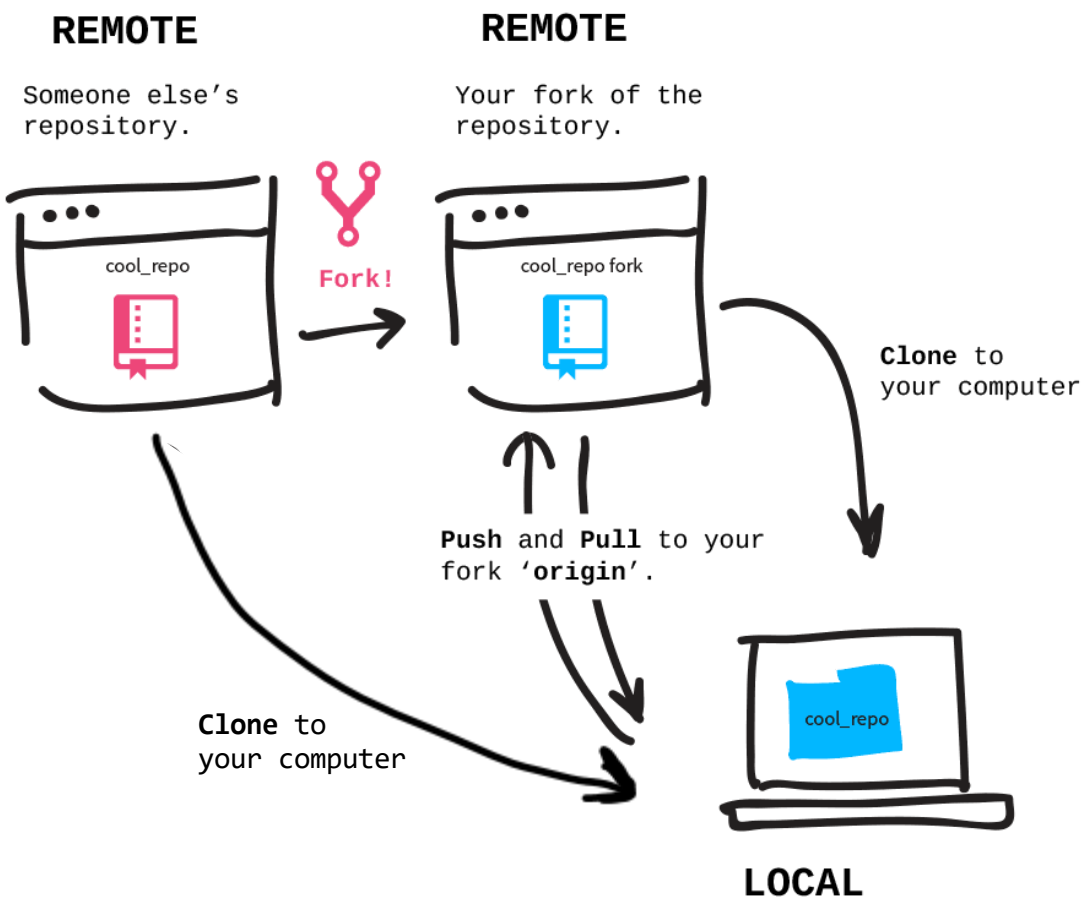


Local vs remote/origin



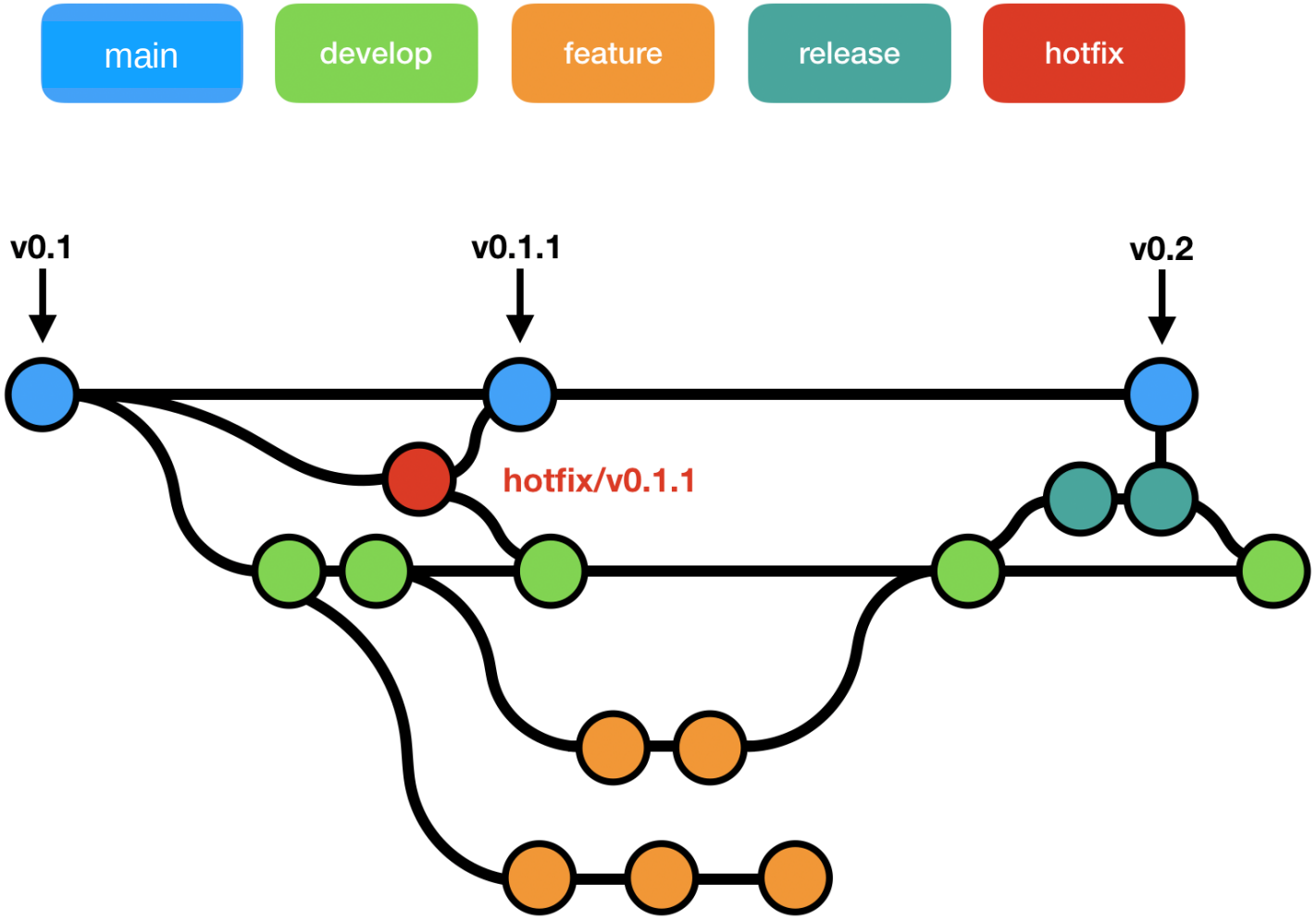
Modified from: https://duzun.me/tips/git

Clone and fork



Modified from: https://duzun.me/tips/git

Branches



Branching strategies:

- GitFlow
- GitHub Flow
- GitLab Flow
- Trunk-based development (CI and CD)

Example of GitFlow branching model, modified from: https://www.codewall.co.uk/a-git-flow-explainer-how-to-tutorial/

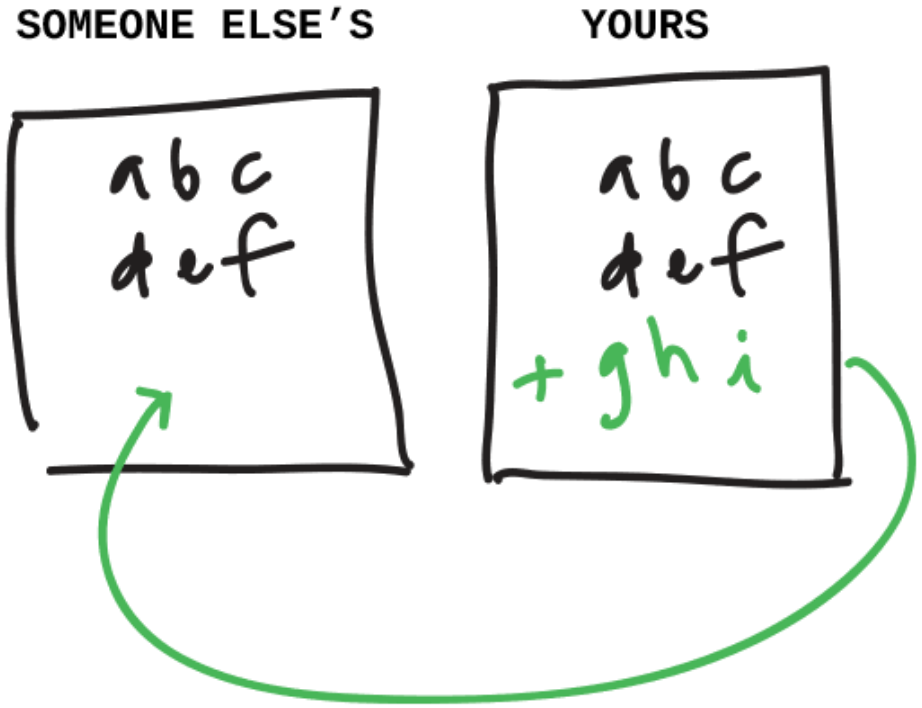
Pull request

Dear someone else,

I'd like to request you pull in the changes I've made to this branch.

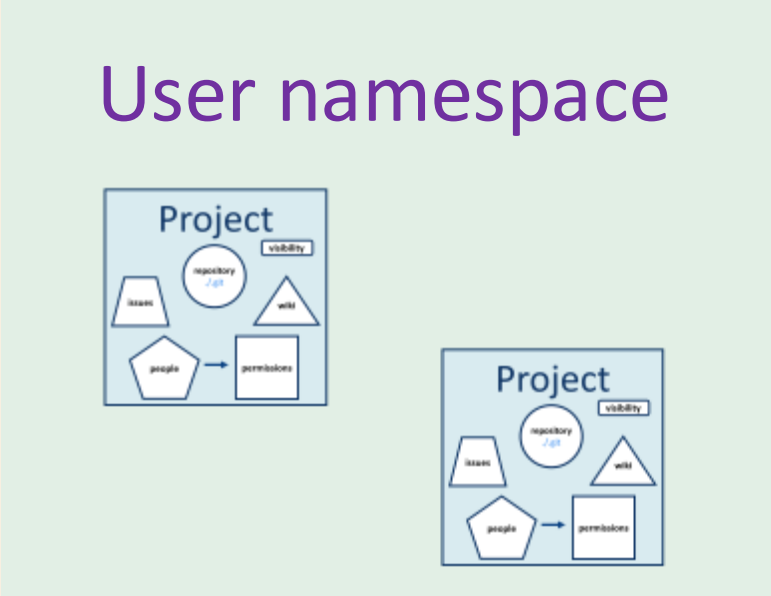
As you can see I've made an addition that I believe many will find useful.

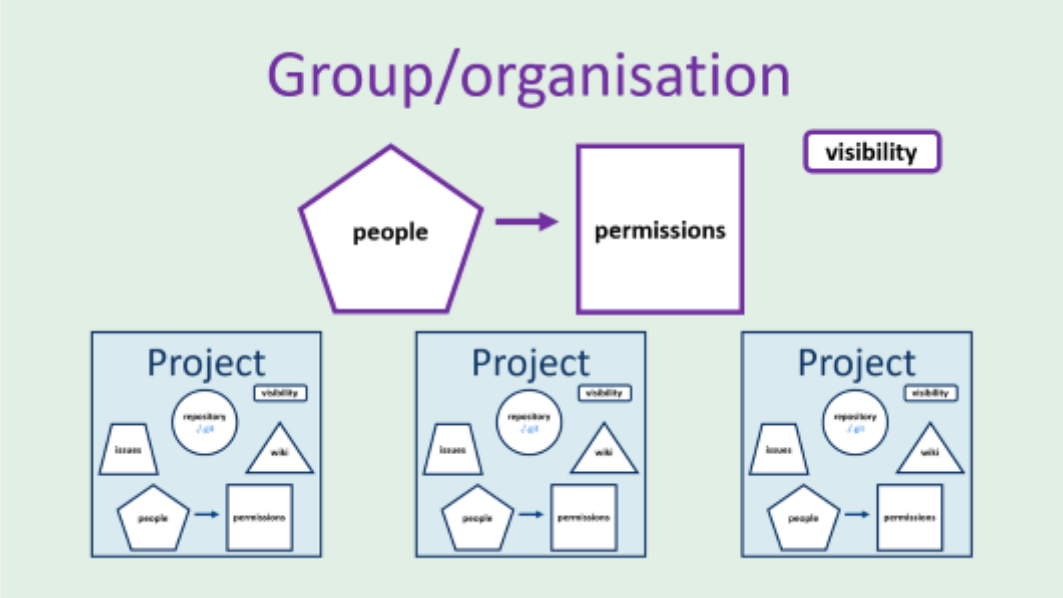
Thanks much,
Me

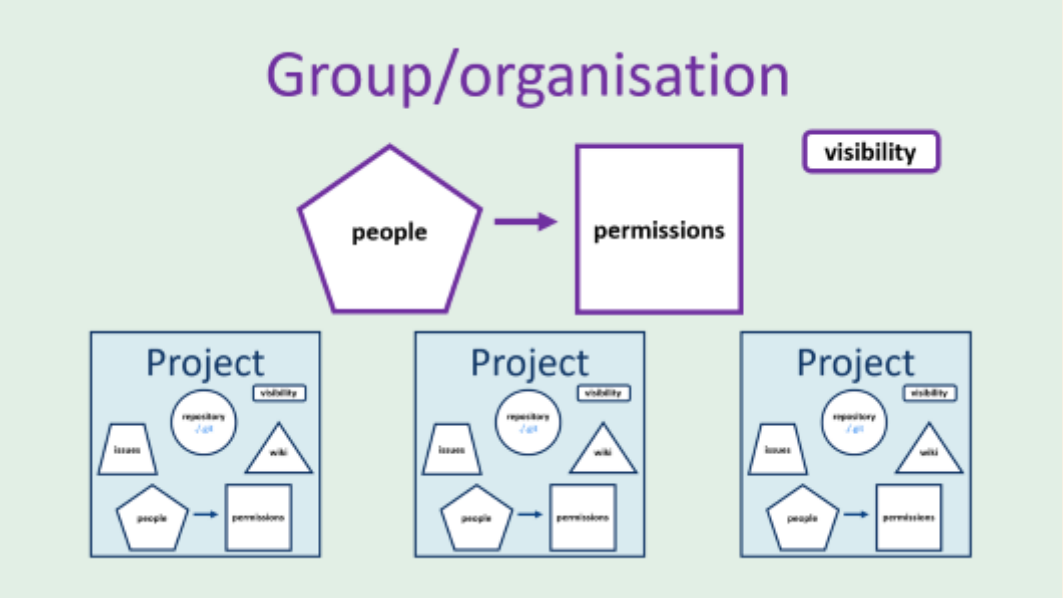


https://duzun.me/tips/git

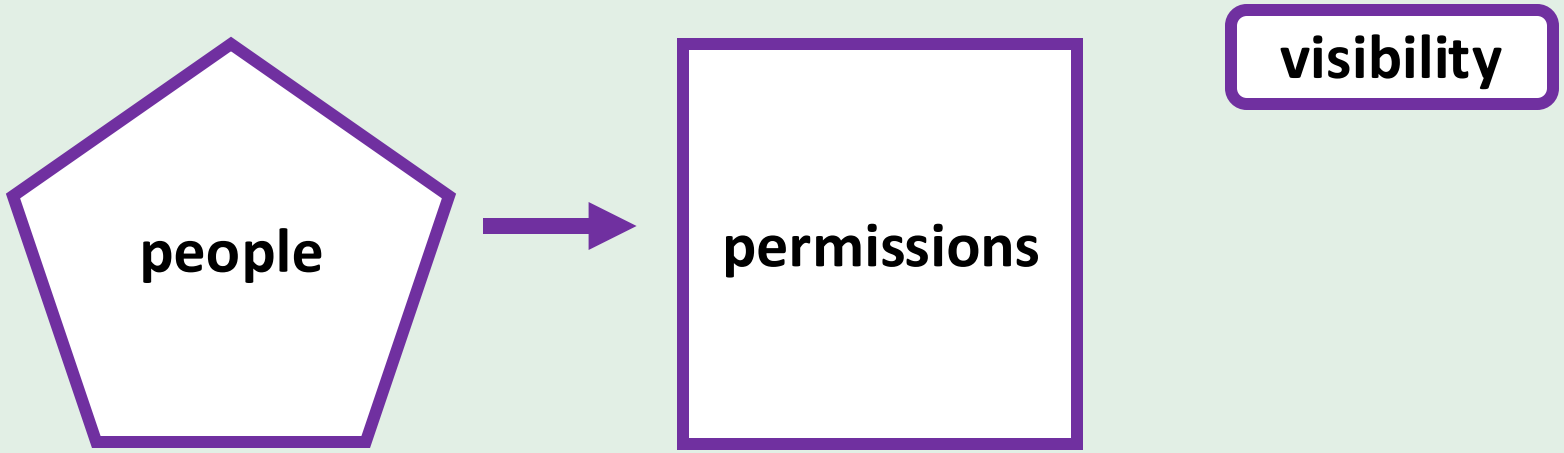
GitLab/GitHub instance

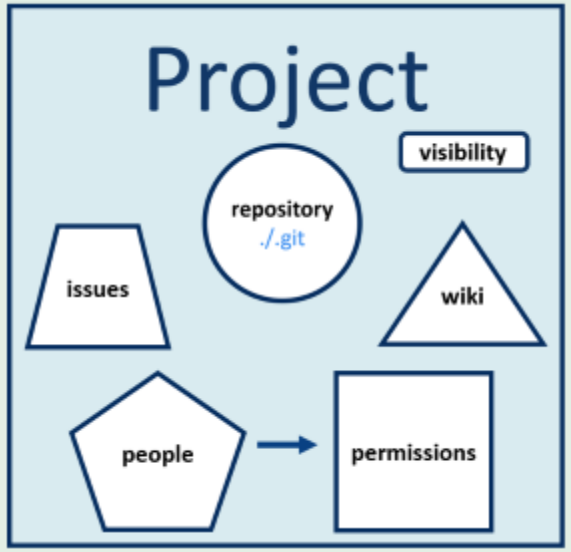


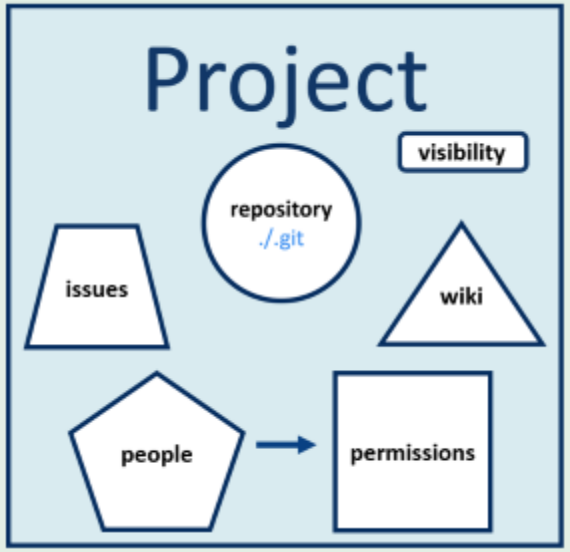


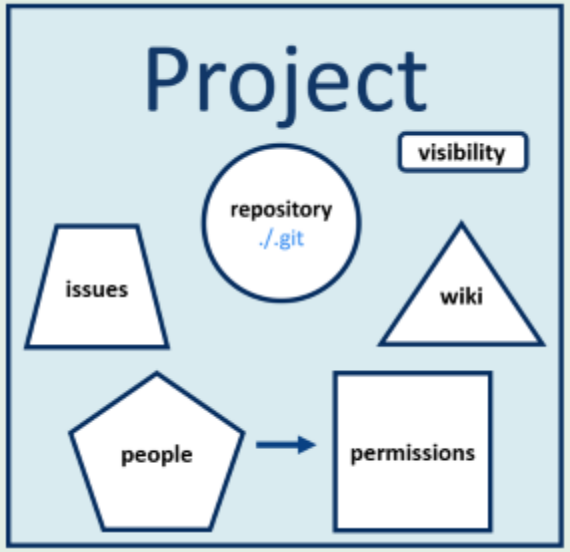


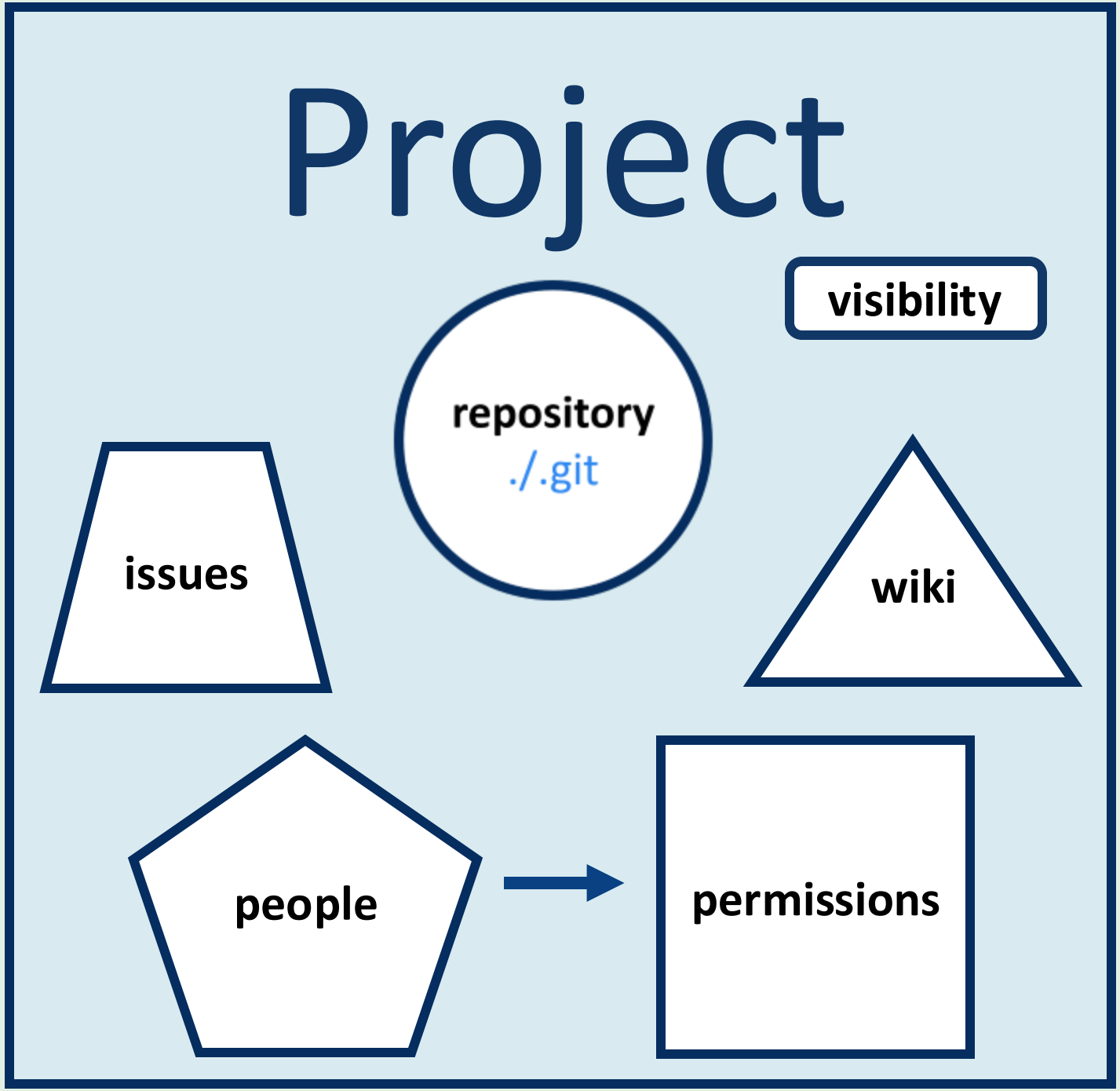
Group/organisation










Project
visibility
repository
./.git
issues
wiki
people
permissions

repository

./.git

Learn more! (an opinionated list of resources)

- *Happy Git and GitHub for the useR*: https://happygitwithr.com/
- The Carpentries (command-line based): https://swcarpentry.github.io/git-novice/
- The Turing Way:

  https://the-turing-way.netlify.app/reproducible-research/vcs.html
- Learn Git branching: https://learngitbranching.js.org/

In case of fire

1. git commit
2. git push
3. leave building

Thank you

THE UNIVERSITY OF MELBOURNE

MELBOURNE DATA ANALYTICS PLATFORM

unimelb.edu.au/mdap
mdap-info@unimelb.edu.au
@MDAP_Unimelb

Mar Quiroga
mar.quiroga@unimelb.edu.au
@_marstudio

Glossary

- Version control: A tool for managing changes to a set of files. Each set of changes creates a new commit of the files. Allows users to recover old commits reliably and helps manage conflicting changes made by different users.
- Repository: A storage area where a version control system stores the full history of commits of a project and information about who changed what, when.
- Remote (of a repository): A version control repository connected to another, in such way that both can be kept in sync exchanging commits.
- Branch: A "branch" is an active line of development.
- Master: The default development branch. Whenever you create a Git repository, a branch named "master" is created and becomes the active branch.
- Changeset: A group of changes to one or more files that are or will be added to a single commit in a version control repository.
- Conflict: A change made by one user of a version control system that is incompatible with changes made by other users. Helping users resolve conflicts is one of version control’s major tasks.

Glossary

- Commit: To record the current state of a set of files (a changeset) in a version control repository. As a noun, the result of committing, i.e., a recorded changeset in a repository. If a commit contains changes to multiple files, all the changes are recorded together.
- Checkout: The action of updating all or part of the working tree with a tree object or blob from the object database and updating the index and HEAD if the whole working tree has been pointed at a new branch.
- Fetch: Fetching a branch means to get the branch’s head ref from a remote repository, to find out which objects are missing from the local object database, and to get them, too.
- Pull: Download commits that don't exist on your local version from a remote repository.
- Push: Add your local changes to the remote repository.
- Merge (a repository): To reconcile two sets of changes to a repository.
- Resolve: To eliminate the conflicts between two or more incompatible changes to a file or set of files being managed by a version control system.