What is git?

Git is **version control**.

Additionally:

- Content tracker
- Collaboration tool
- Notekeeping system
- (Backup - if used correctly)

This session:

- Git
  - Concepts
  - Set up a repository
  - Add (“track”) files
  - Change files
  - Branches
  - Merging
- Git**Hub**
  - Set up a repository
  - Push, pull

caveat emptor

Git is both very simple *and* very complicated.

But it is only as complicated as you need it to be.

Distinction: *Git* vs *GitHub*

Git

The core software which actually *creates and maintains repositories*
Fundamentally a command line tool with a very rudimentary GUI.
Lots and lots of third party graphical interfaces.

- Suggested: SmartGit, GitHub Desktop, **RStudio**
- Others: TortoiseGit, Tower, SourceTree, GitKraken
- List: https://git-scm.com/downloads/guis/

NB: Many of the GUIs simplify interacting with GitHub

Distinction: *Git* vs *GitHub*

GitHub

An online service to *store your repositories, collaborate with others* and various other 'value added' services.

Needs an account to use.

Alternatives: BitBucket, GitLab, self-hosted

It is perfectly possible and reasonable to use Git without using GitHub

- But for simplicity. . . use GitHub.
- NB: **GitHub** operates a student programme where 'premium' features and extras available for students for free for a year or so.

Git basics

- Fundamentally, git creates and stores **snapshots** of files and folders.
- Git must be *told* to track files as you add them.
- Adding new files is a *two step process*, involving (1) staging and (2) committing.

Git basics 2

- When you *delete* a file, git notes it this: you will no longer *see* it in your folders. But *previous* versions of the file live in gits repository.
- Everything is stored in the `.git` folder - normally **hidden** in Explorer/Finder.

Aside: hidden files

- Mac: Cmd+Shift+Dot to view hidden files
- Windows:
  1. Open File Explorer from the taskbar.
  2. Select View $>$ Options $>$ Change folder and search options.
  3. Select the View tab and, in Advanced settings, select Show hidden files, folders, and drives and OK.

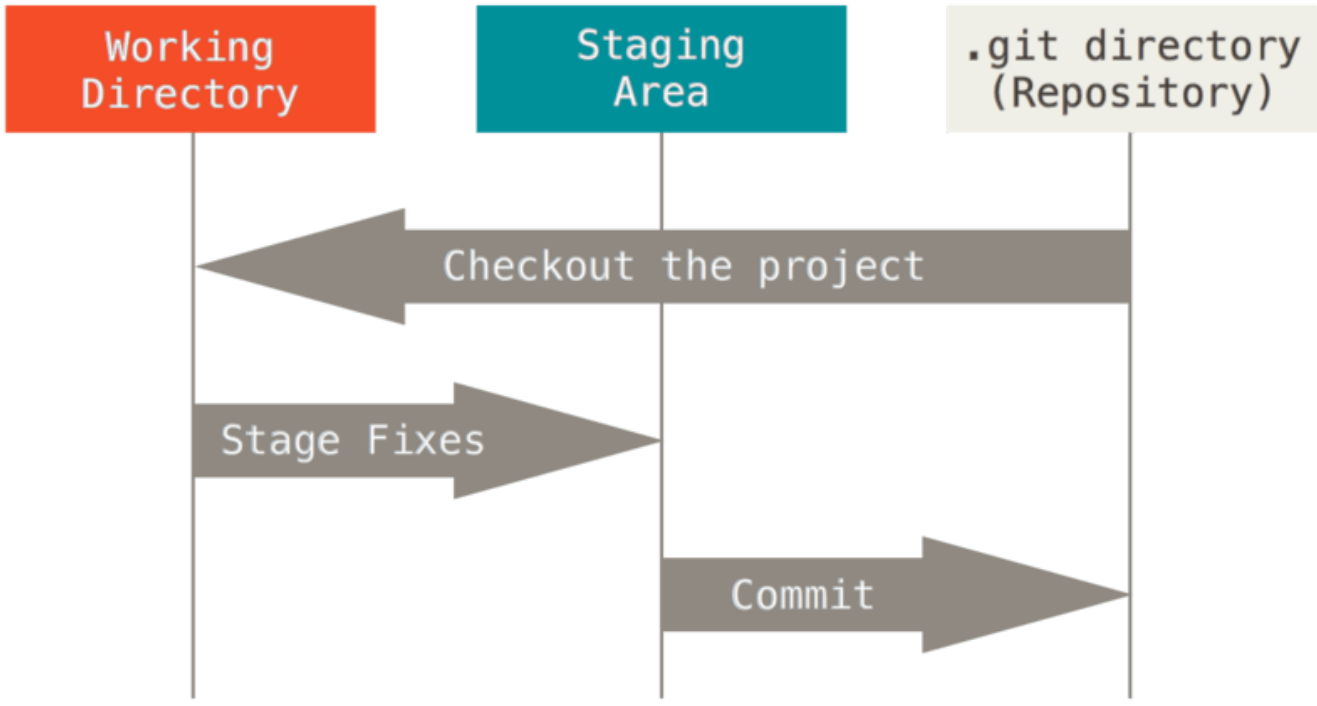
Some concepts

Git 'thinks' in terms of 3 'trees' (fancy word for directory structures)

- `working directory` : what you see when you open Explorer/Finder and look at your project folder.
- `index` : a.k.a staging area - an 'intermediate' area where changes are stored *before* committing.
- `HEAD` : this points to the *last/current commit**.

All comparisons related to files - e.g. whether a file exists, deleted, changed - are done between these three trees.

Unless you actively make it git will never permanently delete anything which has been **committed** (great safety feature!)



Chacon S. Pro Git. Second edition. New York, NY: Apress; 2014. 426 p. URL: https://git-scm.com/book/en/v2

Some terminology

- `stage` : add new files, or changes to already tracked files, to the *staging area* (aka index) - an intermediate step.
- `commit` :
  - verb: add new files/changes to the history - create a 'snapshot'.
  - noun: a previous 'snapshot' in history.

NB: `add` and `stage` are interchangeable. `stage` more common in GUIs.

[Walkthrough 1]

My First Commit: Initialising a repository and adding a file

[Walkthrough 2]

Be The Change You Want To Be: Making Changes and Storing Them

The commit history

- The commit history is a list of all the commits, in order, of a given branch.
- Command: `git log`
- But most GUIs will show you a tree visually, which is easier to follow.

Undoing changes - `git revert`, `git reset`

- The key benefit of version control is the ability to *undo* changes - rollback in time.
- But this requires more thought than at first glance.

! Git Revert Doesn't Do What You Think !

- When you `git revert` you don't go *backwards*. Instead, you go forwards by making an *inverse* commit to where you want to go.
  - Consider this in terms of snapshots and keeping all the snapshots. Going backwards means *losing snapshots*.
  - Going forwards making an inverse commit *keeps* the intermediate snapshots.
  - Often a source of confusion.

git revert

Consider:

Commit A: Files `+a`, `+b`, `+c` ====> Result: Files `a`, `b`, `c`

Commit B: Files `+d`, `+e`, changes `a'` ====> Result: Files `a'`, `b`, `c`, `d`, `e`

[revert!]

Commit C: Files `-a'`, `-d`, `-e` ====> Result: Files `a`, `b`, `c`

If you ever need to, you can *revert* to commit **B**. Files `d` and `e` are in the history as well

Reset

- `git reset` : this *resets* the *working directory* OR *index* back to their last state.
- This is probably what you want most of the time - used to undo changes/adding files into the index.
- `git restore filename` : this will restore the last *committed* version of the file into the working directory.

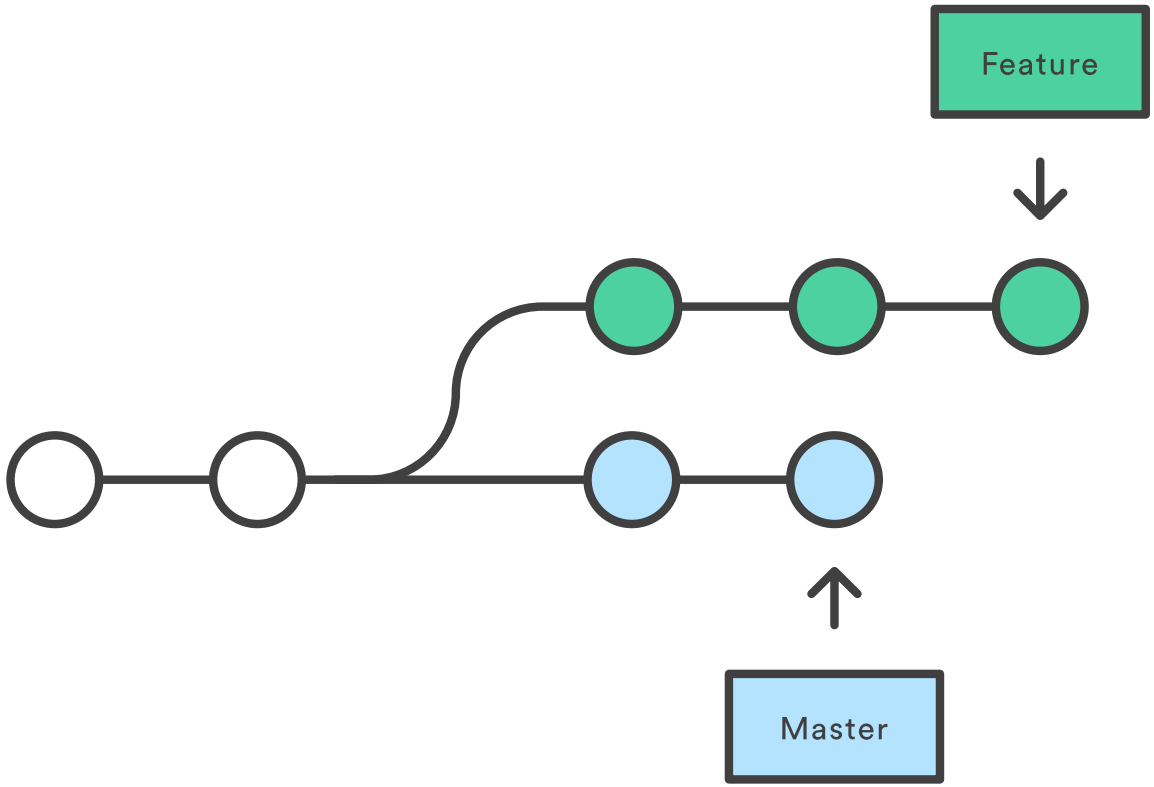
[Walkthrough 3]

Back to the Future

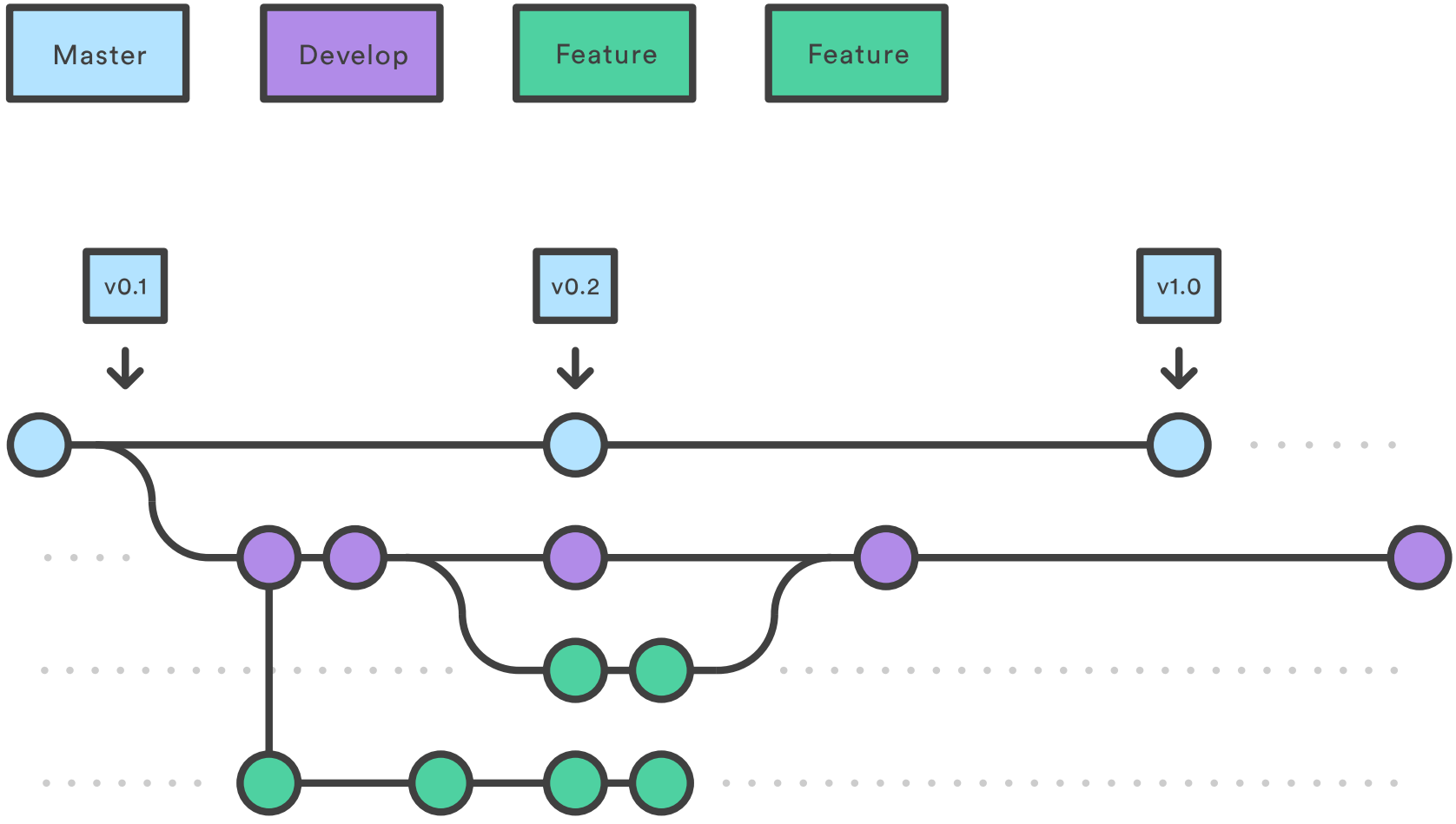
Git Branches

- A branch is a parallel workstream.
- Think of having a multiple working directories, each having independent work done on it.
- Useful for:
  - Collaboration - so multiple people can work on aspects of a project without messing with the ‘master’ copy.
  - Testing - quickly test a new bit of code/work, pause it, return to master, carry on, go back to new code etc.

A forked commit history



Jens Lechtenbörger https://oer.gitlab.io/oer-on-oer-infrastructure/Git-introduction.html#/sec-title-slide



Jens Lechtenbörger https://oer.gitlab.io/oer-on-oer-infrastructure/Git-introduction.html#/sec-title-slide

Git branching commands

- `git branch branchname` - create a new branch called branchname
- `git checkout branchname` - switch to branchname
  - New versions of git: also use `git switch branchname`
- `git checkout -b branchname` - shorthand for "create new branch branchname then switch to it".

[Walkthrough 4]

Remotes!

i.e. GitHub

Git can store repositories in so called ‘remotes’.

- A remote is just another location containing a copy of the **.git** folder.
- Can be another folder on your computer, a network drive, service such as GitHub.
- **But** your local copy *must be told* where the remote is.
- This is a bit confusing how to do, I think.
  - Recommend: *create* the empty repository remotely (on GitHub) **first**. THEN **clone** the empty repository to your computer. That was the “remote” related information is already set up.

NB You need to have a GitHub account

Pull

- Bring in any changes in the remote repository and *merge* them into the local one.
- Commands:
  - `git pull`
  - The very first time you want to copy a remote repository: `git clone`

Push

- Send your committed changes up into the remote repository
- Commands:
  - `git push`

[Walkthrough 5]

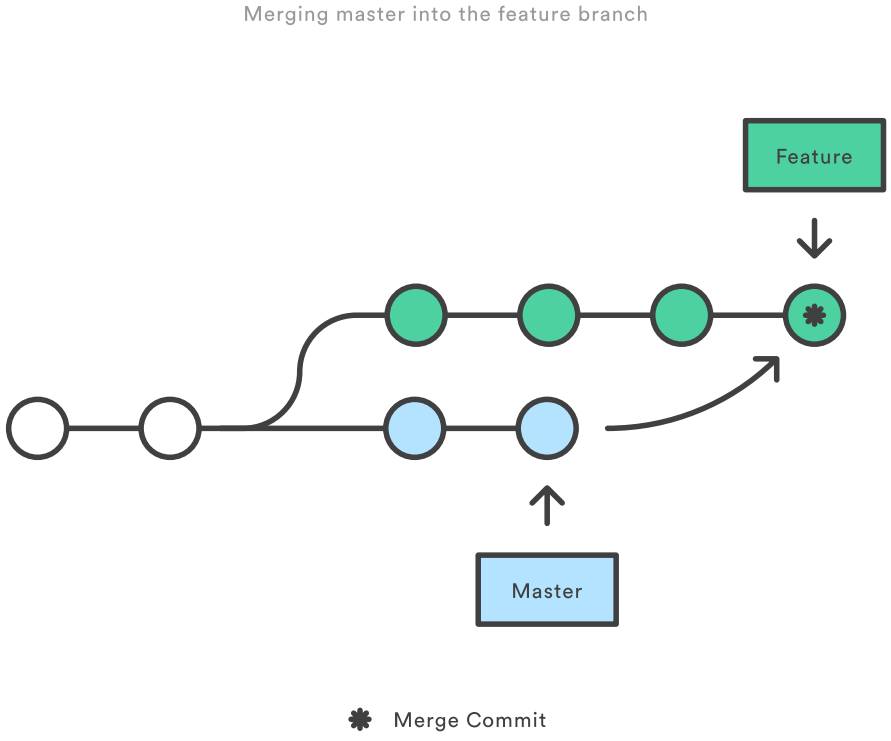
Tug o' War

Merging

- Two common scenarios:
  - You've made some changes in Branch-B, but would like to merge them back in to Branch-A.
  - You've been working on your working directory and made some commits. However, meanwhile, Person-B else has committed some other changes to the repository on GitHub. You need to merge your and Person-B's changes.

Merging 2

- `git merge` operates on *commits* NOT individual files - endless source of frustration for me.
- Merging happens from the POV of the *receiving* (destination) branch/commit.
- *All the changes* from the **source** are brought into the **destination**.



Jens Lechtenbörger https://oer.gitlab.io/oer-on-oer-infrastructure/Git-introduction.html#/sec-title-slide

[Walkthrough 6]

Merging lanes

Wrap up

- Use Git
- Learning curve but will repay itself many times over.
- Good research practice - keeping records.

Other useful topics

- Rebase - *changing history*
- Bisect - *made a mistake, now find the bug?*
- Git hooks - *before every commit, do X. Or before push/pull, do Y etc*
- Worktrees - *multiple branches of same repository in different folders*
- Cherry-pick - *did a thing in branchB, want it in branchA but only that very specific thing, not all the history of branchB*

Other Resources

- Git:
  - https://git-scm.org/ [NB GitHub Desktop will install **Git itself** on your computer]
- Tutorials:
  - http://learngitbranching.js.org - highly recommended, not just about branching
  - https://rogerdudler.github.io/git-guide/ - a simple introduction
  - https://www.atlassian.com/git/tutorials - a series of fairly detailed (but comprehensive) tutorials
  - https://git-rebase.io/ - how to change history with git rebase
  - https://github.com/jlord/git-it-electron - Git-it app to teach you git
  - https://try.github.io/ - various things from GitHub
  - https://marklodato.github.io/visual-git-guide/index-en.html - A visual git reference
  - https://gitimmersion.com/ - another tutorial, needs a Ruby interpreter installed

Other Resources

- Reference/Resources:
  - https://git-scm.com/docs - Git Reference *Manual*
  - https://git-scm.com/book/en/v2 - Git Reference Book
  - https://www.atlassian.com/git/tutorials/atlassian-git-cheatsheet - Quick reference
- Deep dive:
  - https://wyag.thb.lt/ - Write Yourself A Git [rewrite the entire core git program from scratch]
  - https://hackernoon.com/https-medium-com-zspajich-understanding-git-data-model-95eb16cc99f5 - an easy to read but very useful set of articles on git internals.
  - https://www.sbf5.com/~cduan/technical/git/ - Conceptual understanding

Other Resources

- Other VCS: SVN, Mercurial, Pijul, Fossil, DARCS, Bazaar
- SUGAR - these are like 'simplified' wrappers *on top* of Git:
  - https://gitless.com
  - https://frostming.github.io/legit/
- Misc:
  - https://keepachangelog.com/en/1.0.0/ - discretizing your work and good commit messages
  - https://www.conventionalcommits.org/en/v1.0.0/ - a framework for good commit messages