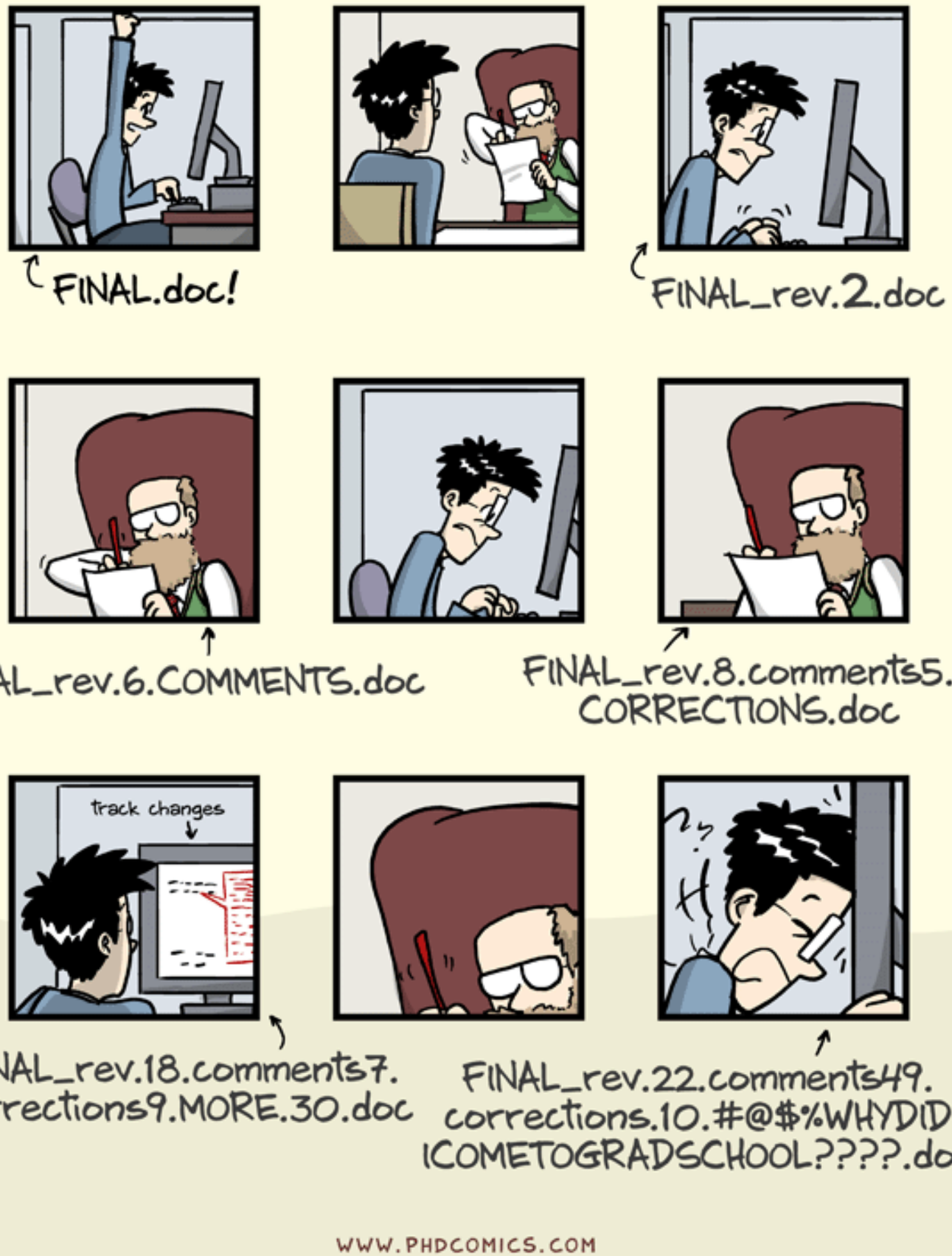
Modern Analysis Techniques for Large Data Sets

Miguel F. Morales
Bryna Hazelton

"FINAL".doc



JORGE CHAM © 2012

git and GitHub

- git
  - version control
  - local + remote

- GitHub
  - Tools for collaboration
    - Issue tracking, pull requests with code review, forking
  - hosting & public access

Why version control (and git)

- simplicity
  - only need one copy
  - always clear what the current version is
  - git just tracks changes
  - backup!
- freedom to delete code
  - git keeps the full history, you can always resurrect old code
  - don’t need to keep commented code around ‘just in case’
- provenance and reproducibility
  - makes it possible to track exactly what code was run for any analysis
  - fine-grained history
- good support for branching and merging
  - supports development separate from a stable ‘main’ branch
  - aids parallel development

GitHub collaboration tools

- Issue tracking
  - with labelling, assignments and links between issues and PRs
- Pull Requests (PRs)
  - build code reviews into the process of merging in new functionality
- integrations with other services
  - Continuous integration: tests and other checks run every time the repo is updated
  - Documentation hosting: rebuild the documentation every time the repo is updated
- user interface for exploring code changes
  - graphical diffs between any commits or branches

Reproducibility & Open Science

- public code
  - complementary (some times required) to publishing
  - documentation and readability are key
  - the analysis is the code — allows others to see what you did

- open source code
  - requires an open source license
  - API documentation
  - unit testing and continuous integration
  - building a user community

- reproducibility
  - capture of versions & settings
  - open data
  - full stack capture, containers (docker)

git basics

- local repository
  - a complete copy (with the full history) on your local machine
  - self-contained and self-sufficient

- remote repository
  - a complete copy hosted remotely (e.g. on GitHub) — the repository all collaborators have access to

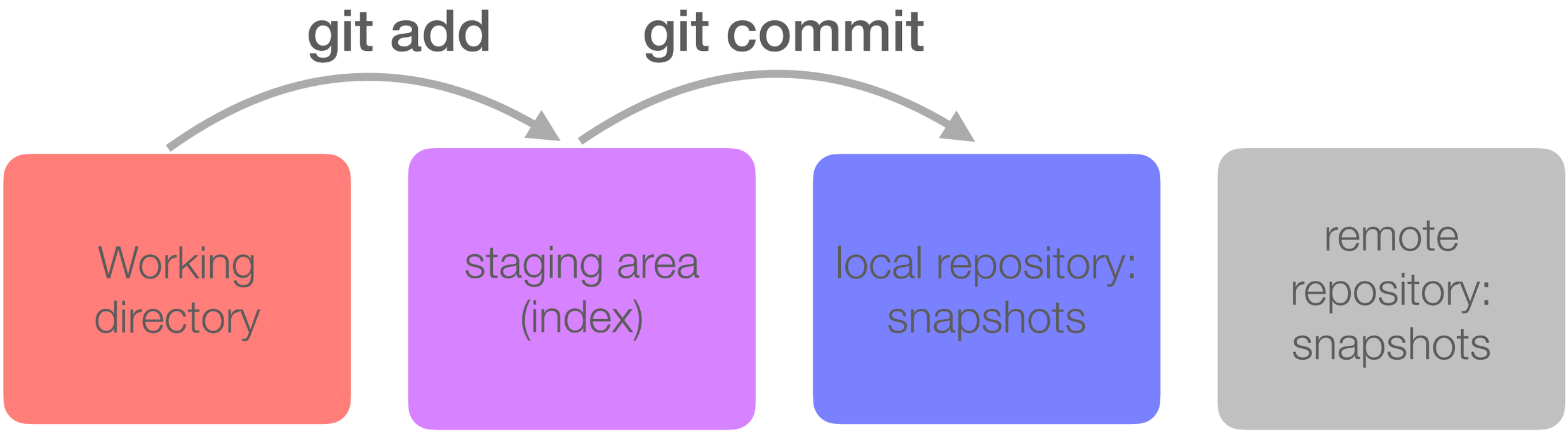
- snapshots (commits)
  - the unit of tracking within git — can be multiple changes to multiple files
  - should be used to identify ‘atomic’ changes — things that go together
  - commit early & often — fine-grained commits make the history more useful

Working directory

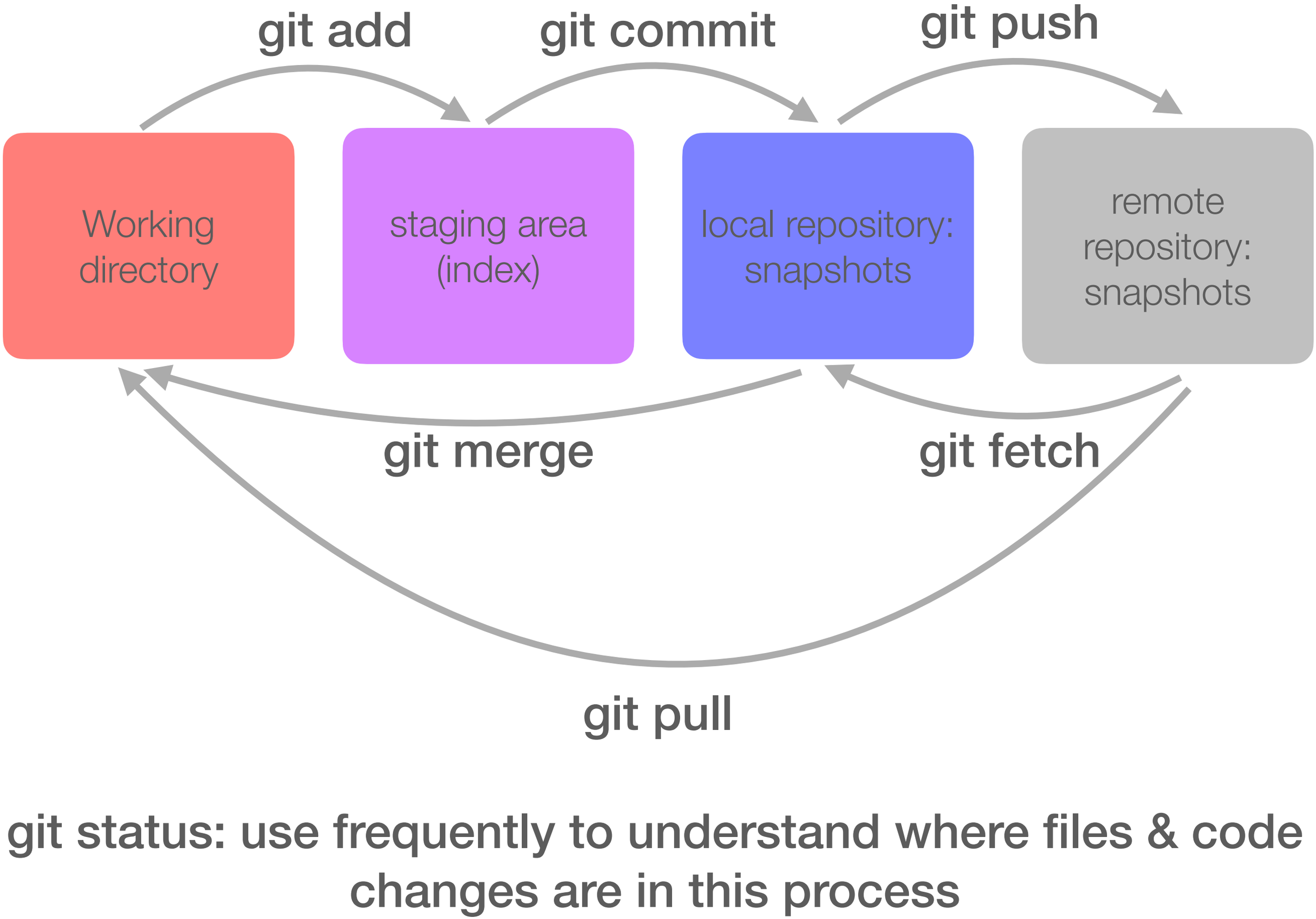
staging area (index)

local repository: snapshots

remote repository: snapshots



git status: use frequently to understand where files & code changes are in this process



git status: use frequently to understand where files & code changes are in this process

making changes

- Check the status
  - use `git status` to see what things have changed
  - use this command liberally — it's always safe and helps you know what's going on
- Identify all the changes you want to snapshot together
  - use `git diff` to see what the changes are
  - use `git add <file>` to move changes to the staging area
  - only include changes that go together
- make the snapshot
  - use `git commit` to make the snapshot: brings up a browser to add a commit message
  - or `git commit -m 'your message here'`
  - commit messages should be **descriptive**
- view the history
  - `git log`, `git show`

syncing with the remote

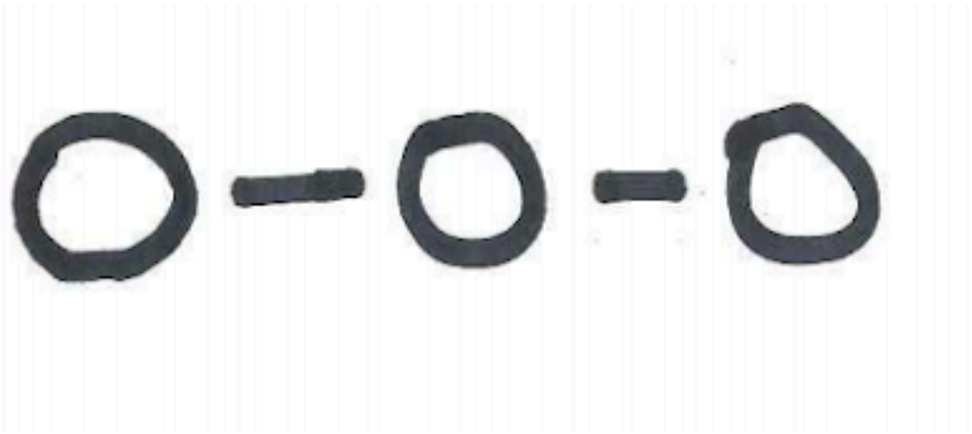
- get snapshots from the remote
  - use **`git fetch`** to get the snapshots but not apply them to the local repo
  - use **`git status`** to see differences between the local and remote
  - use **`git merge`** to apply the snapshots to the local repo
  - **`git pull`** is **`git fetch`** immediately followed by **`git merge`** but doesn't let you examine the snapshots before applying them
- send your snapshots to the remote
  - use **`git push`** to send your local snapshots to the remote
  - git will not let you push if there are snapshots on the remote that you have not yet merged into your local repository
- view the history
  - **`git log`**, **`git show`**

Understanding the Git Tree

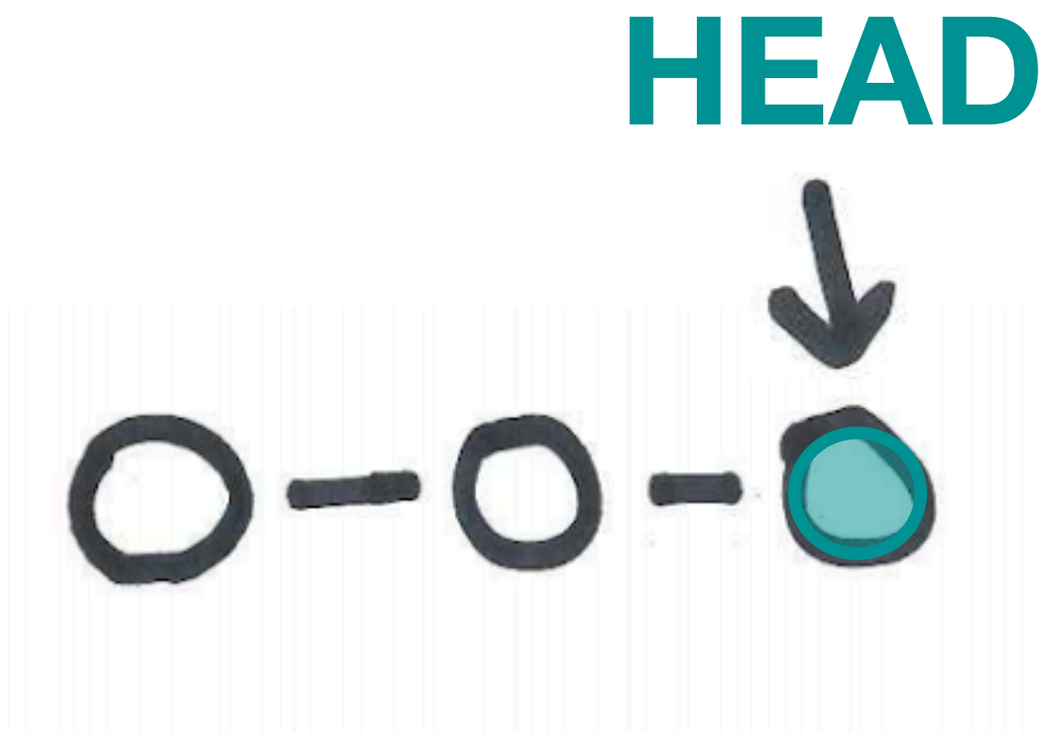
Representation

a single snapshot:

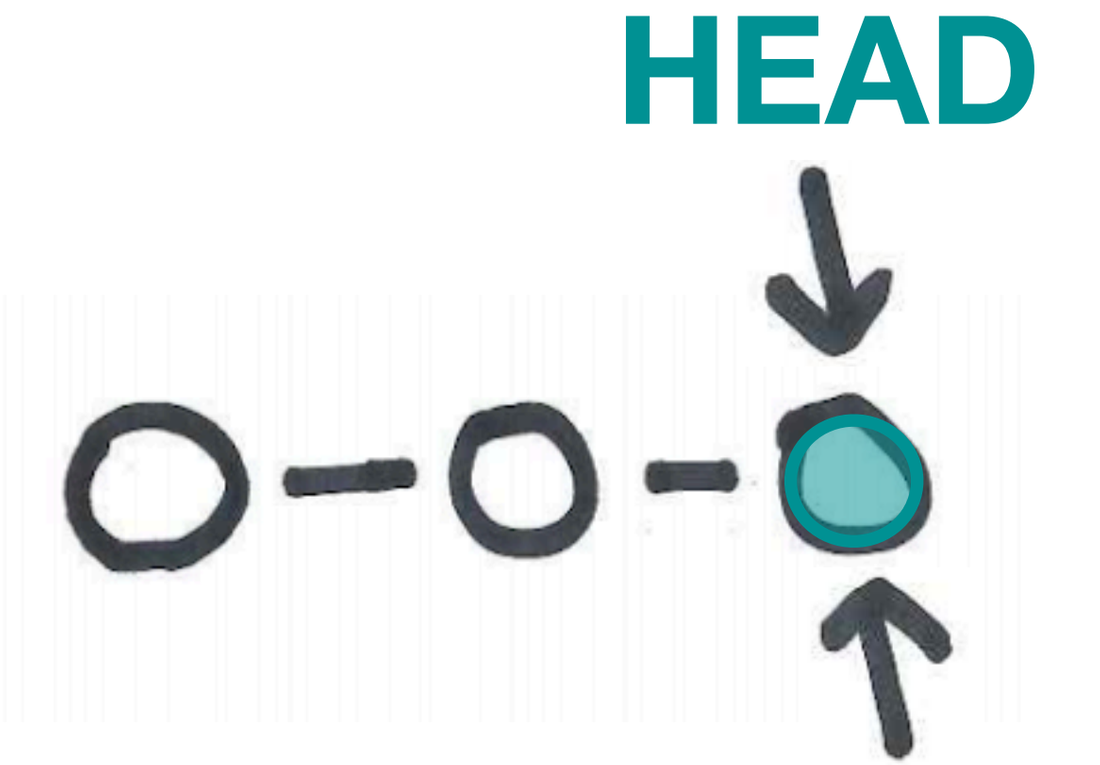
a series of snapshots:



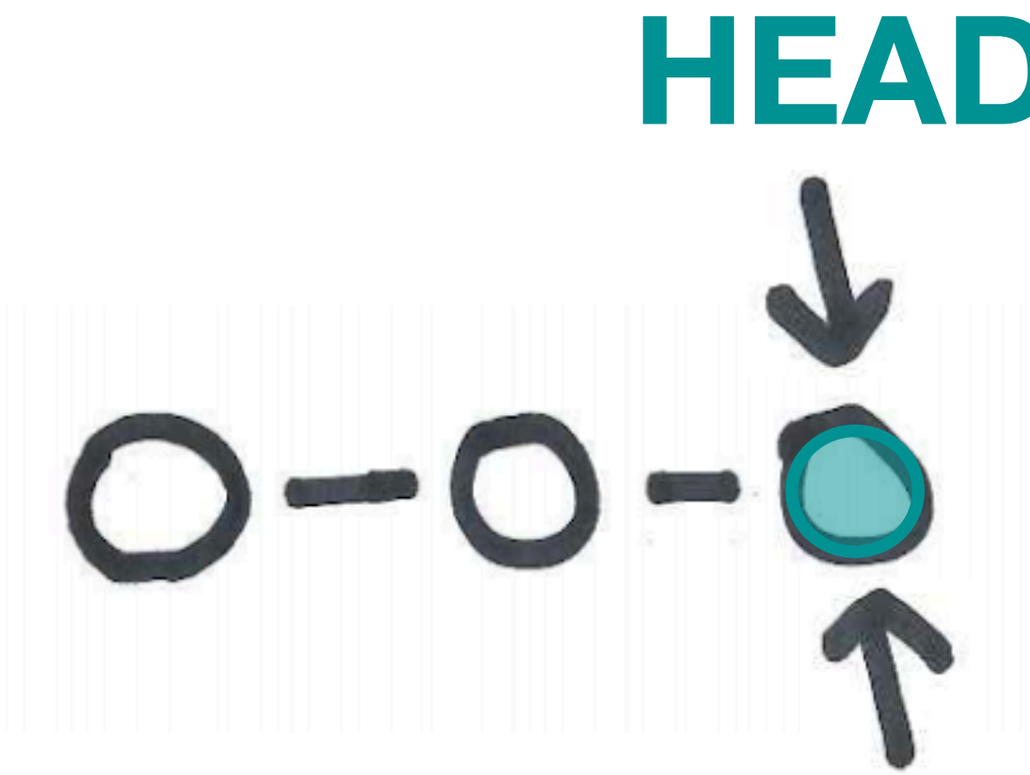
Your working directory and HEAD pointer



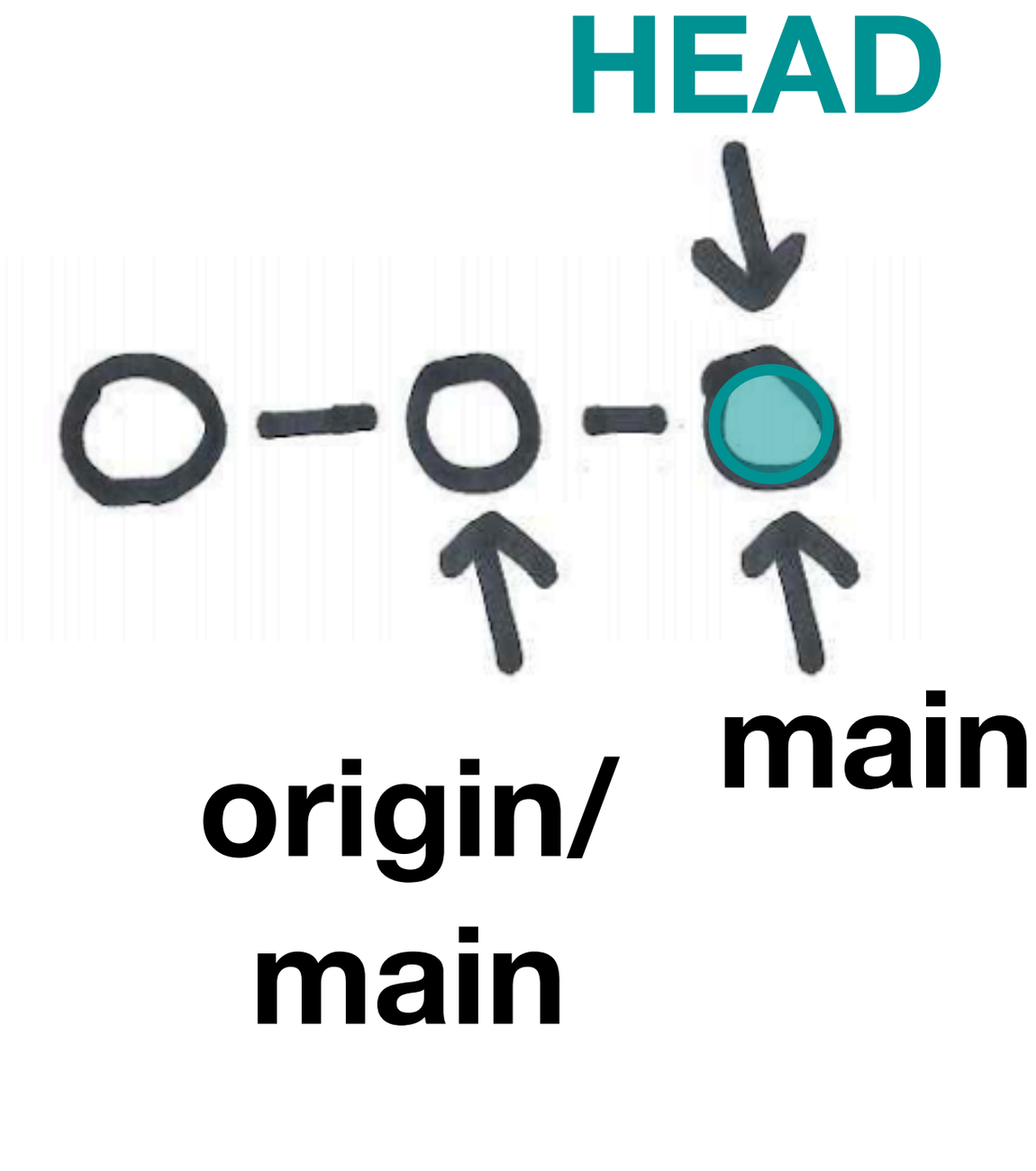
Think of branches as pointers as well



Remote branches included



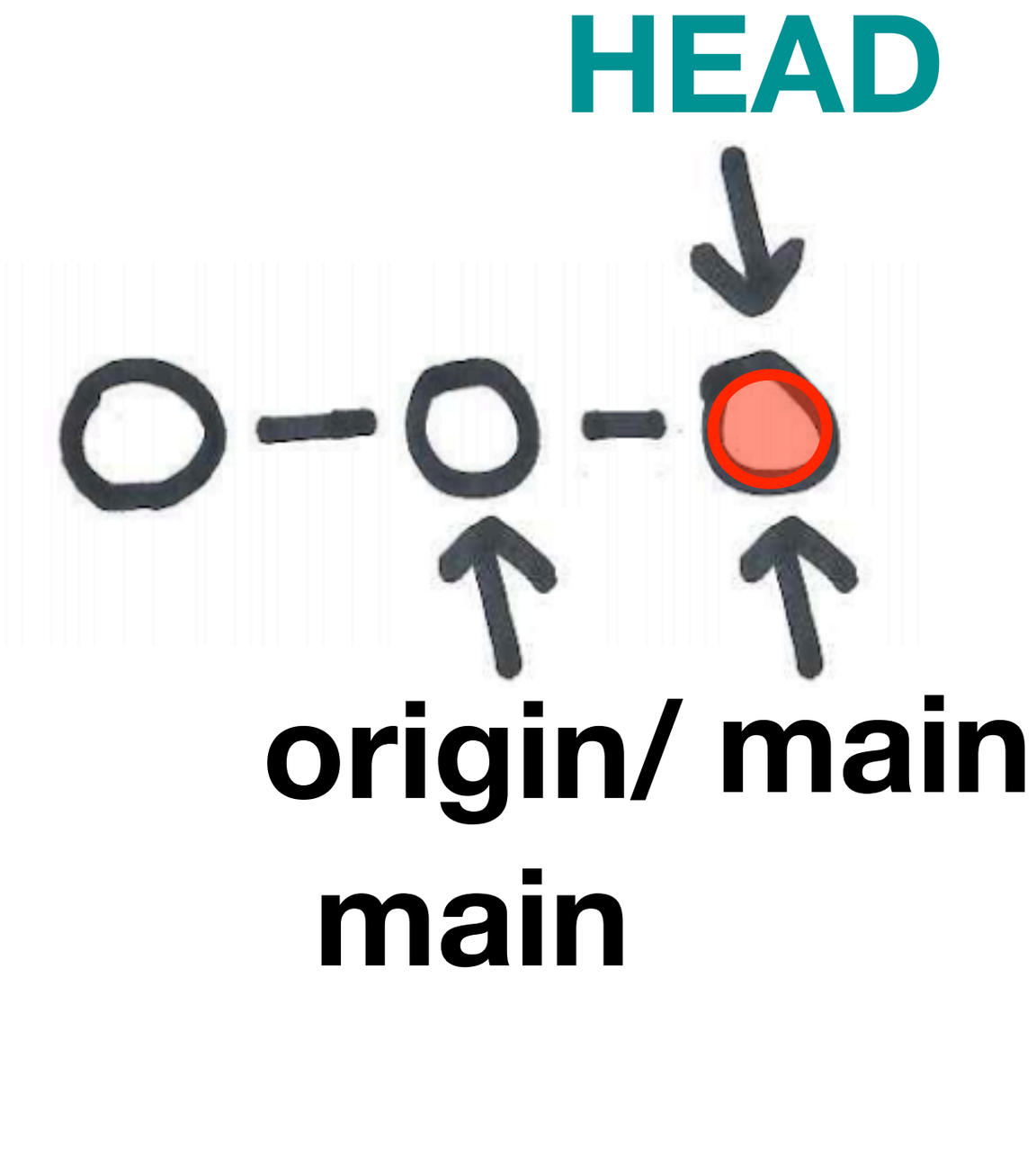
Local snapshot, before pushing to remote



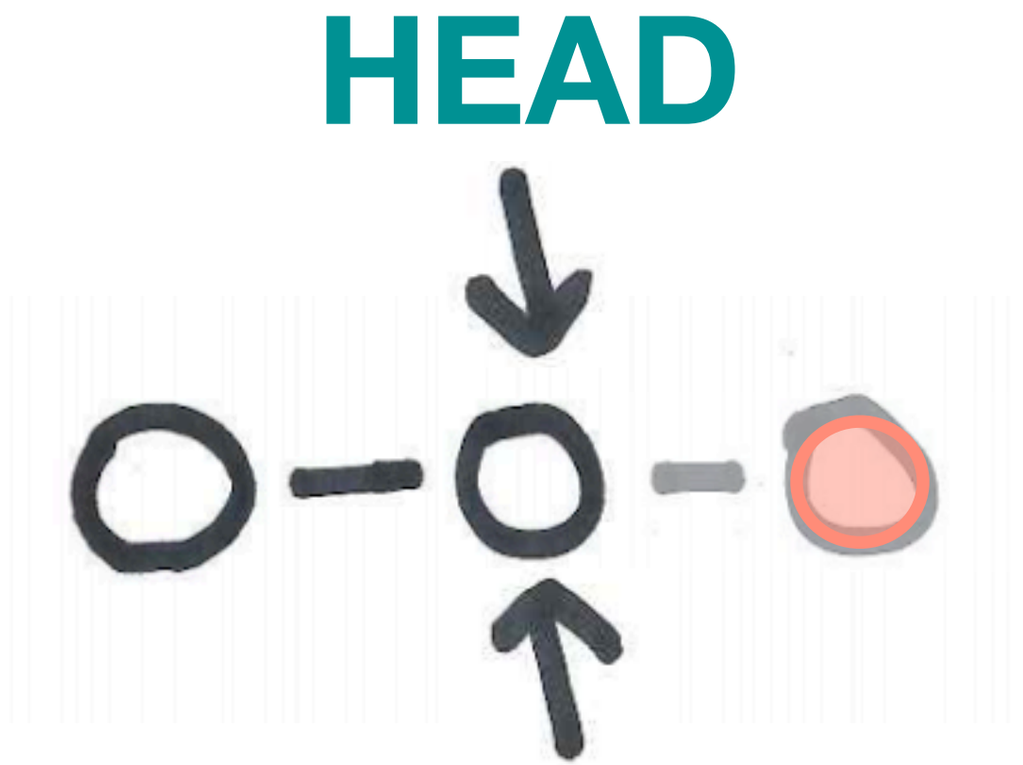
editing and deleting snapshots (commits)

- amend commits
  - use **`git commit --amend`** to add new changes to the last snapshot. More options with rebase
- rebase
  - use **`git rebase [-i]`** to combine, squash, delete snapshots, or to move the base of a branch
- reset:
  - use **`git reset`** to remove a snapshot and optionally staged changes and working directory changes
- revert
  - use **`git revert`** to create an additional snapshot that reverses changes for specified snapshots. Good for main branches of shared repos.

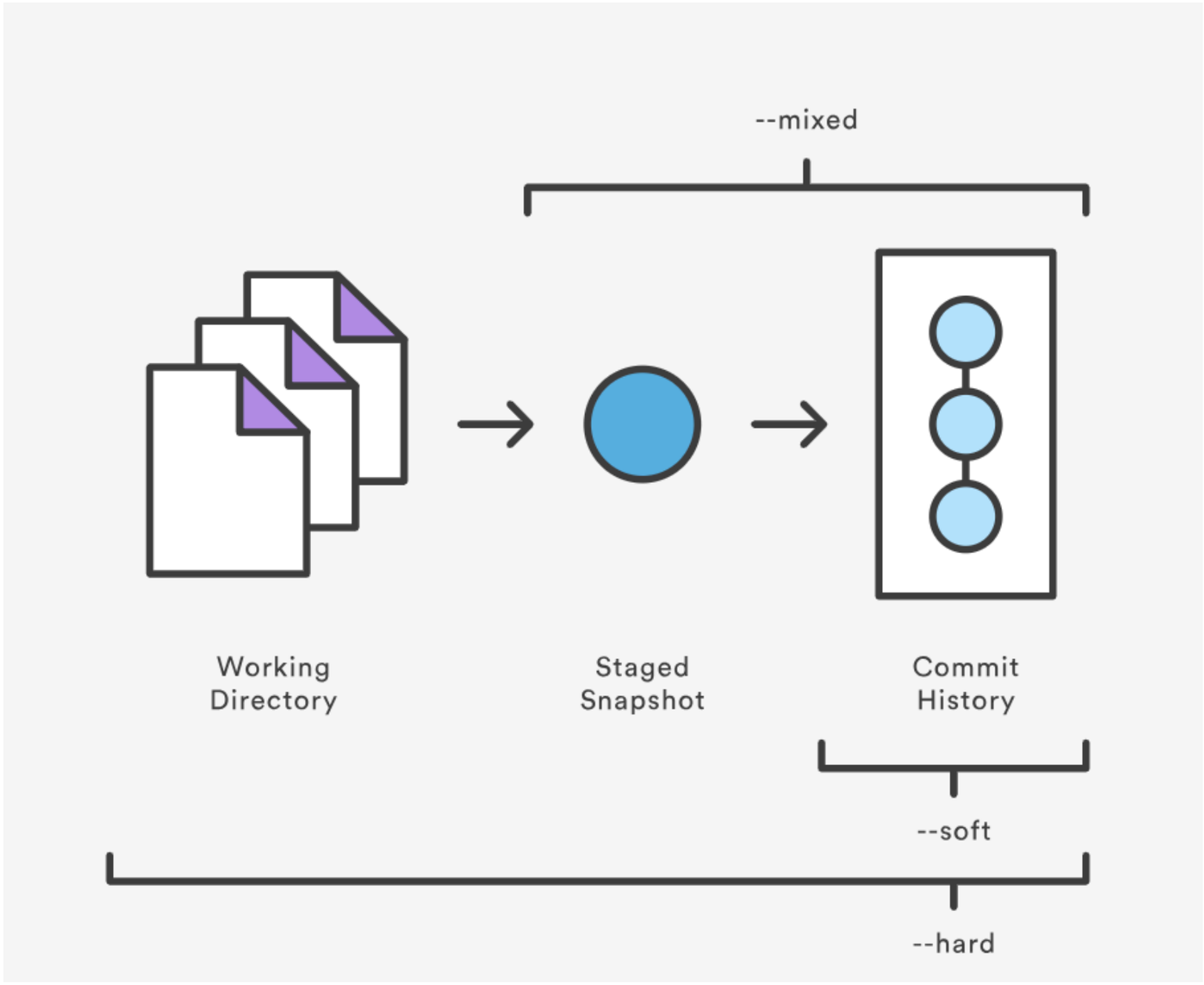
You’ve made an unwanted snapshot locally



reset your snapshot



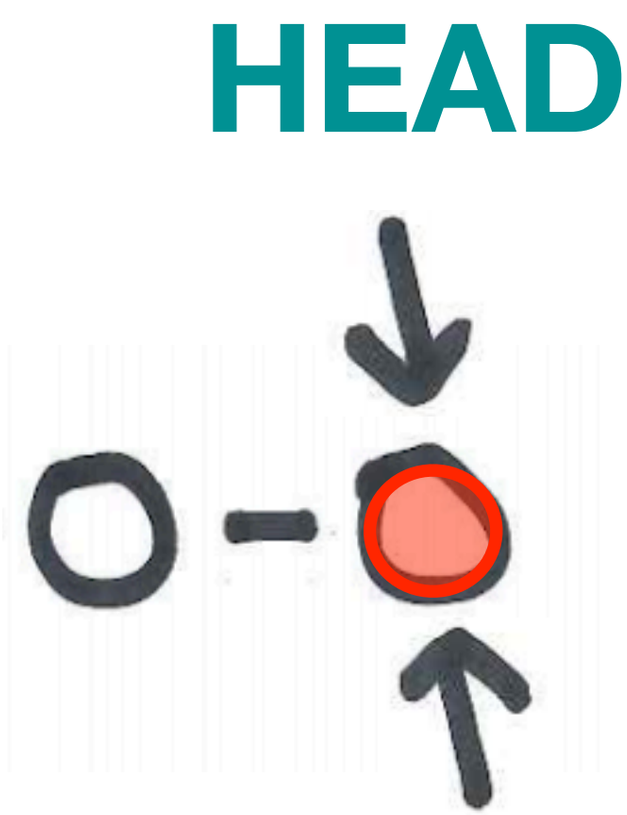
differing levels of reset



--soft:
leave changes in directory and staged area
--mixed:
leave changes in directory, unstage everything (default)
--hard:
remove changes from working directory

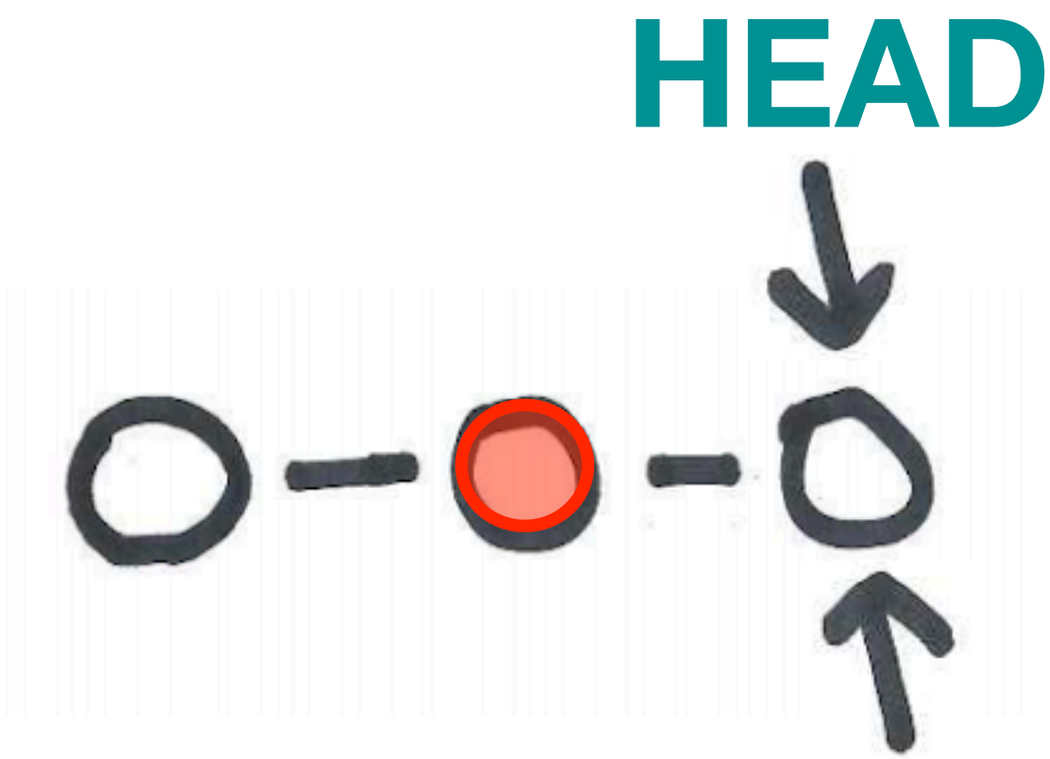
You’ve made an unwanted snapshot on the main branch of a shared repo

---

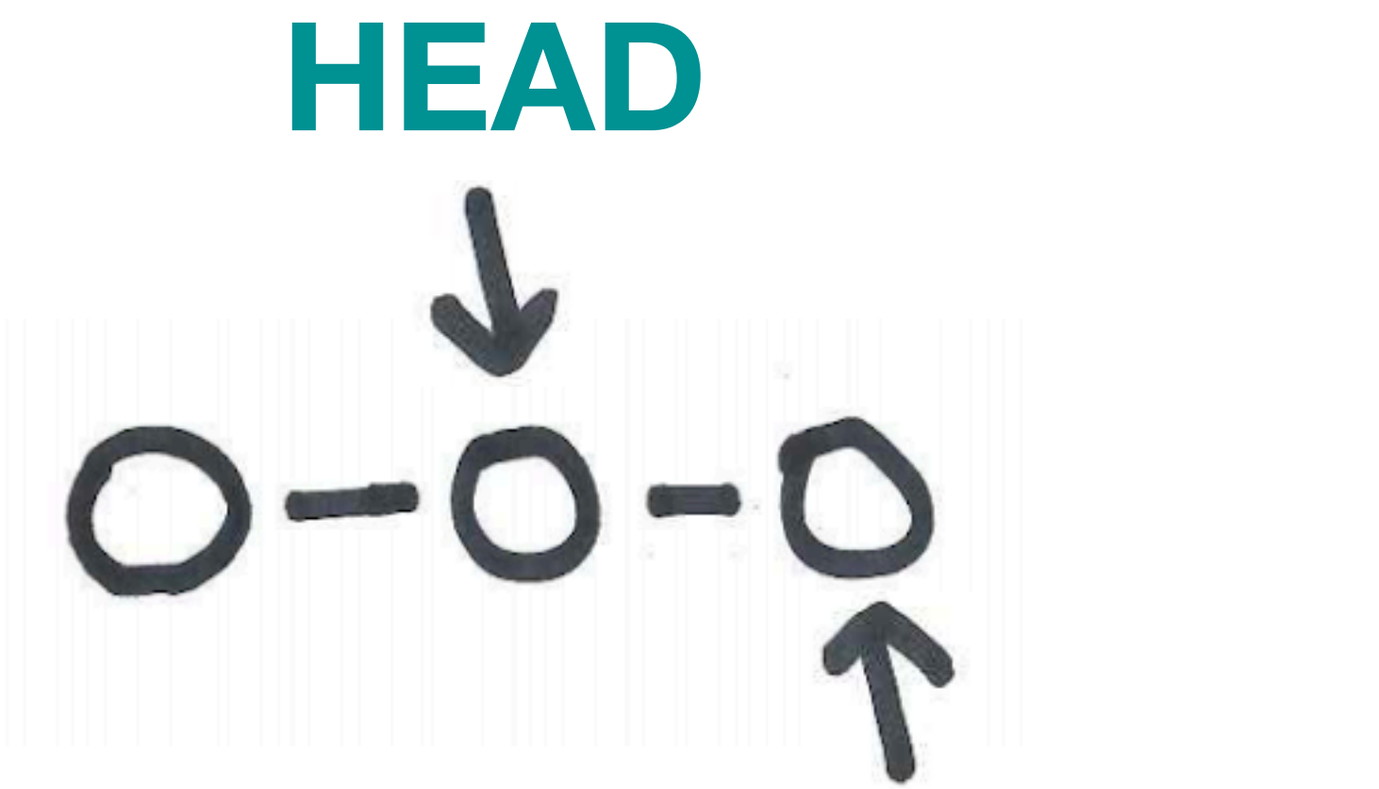


Use git revert to add a new snapshot that fixes (undoes) the bad snapshot

---

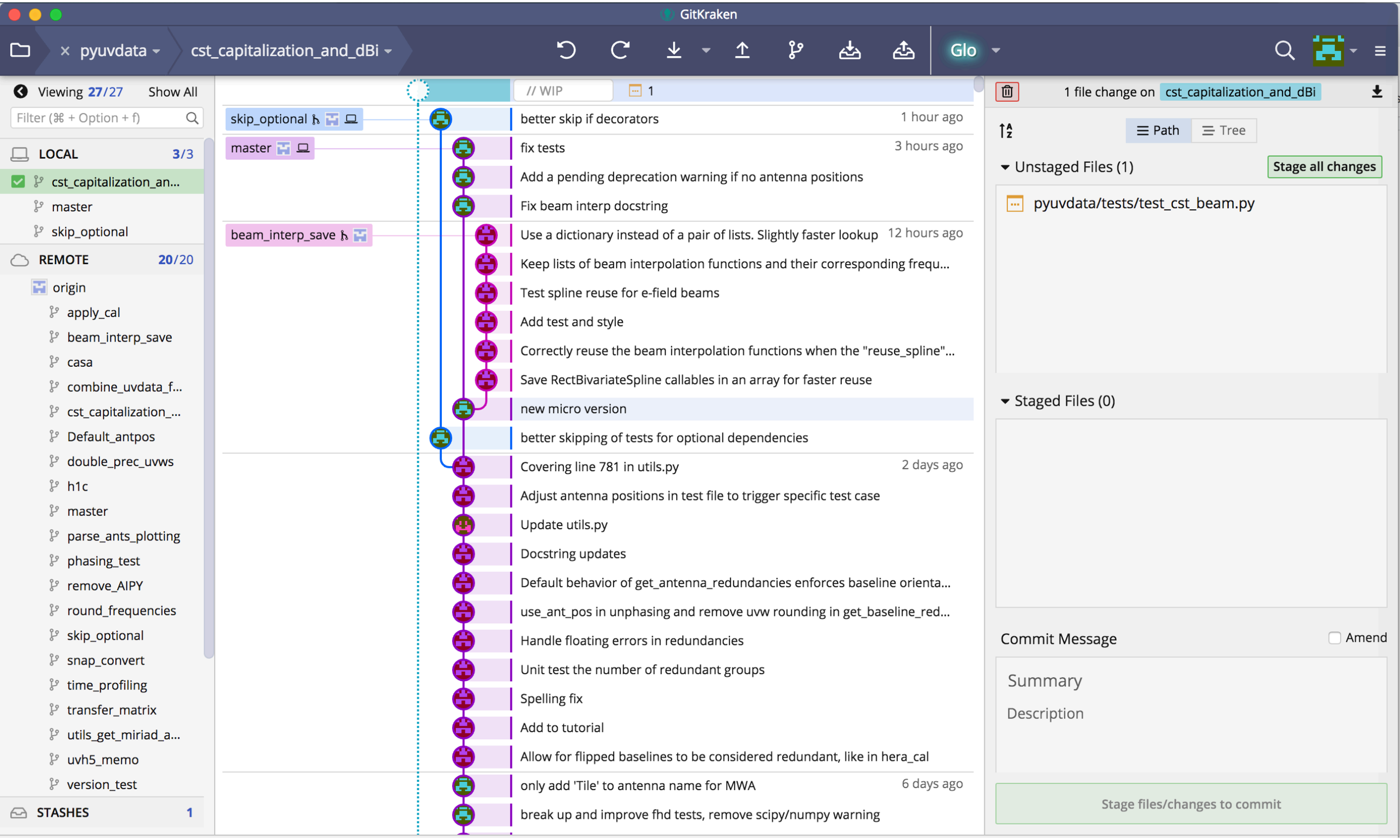


Checkout an earlier snapshot



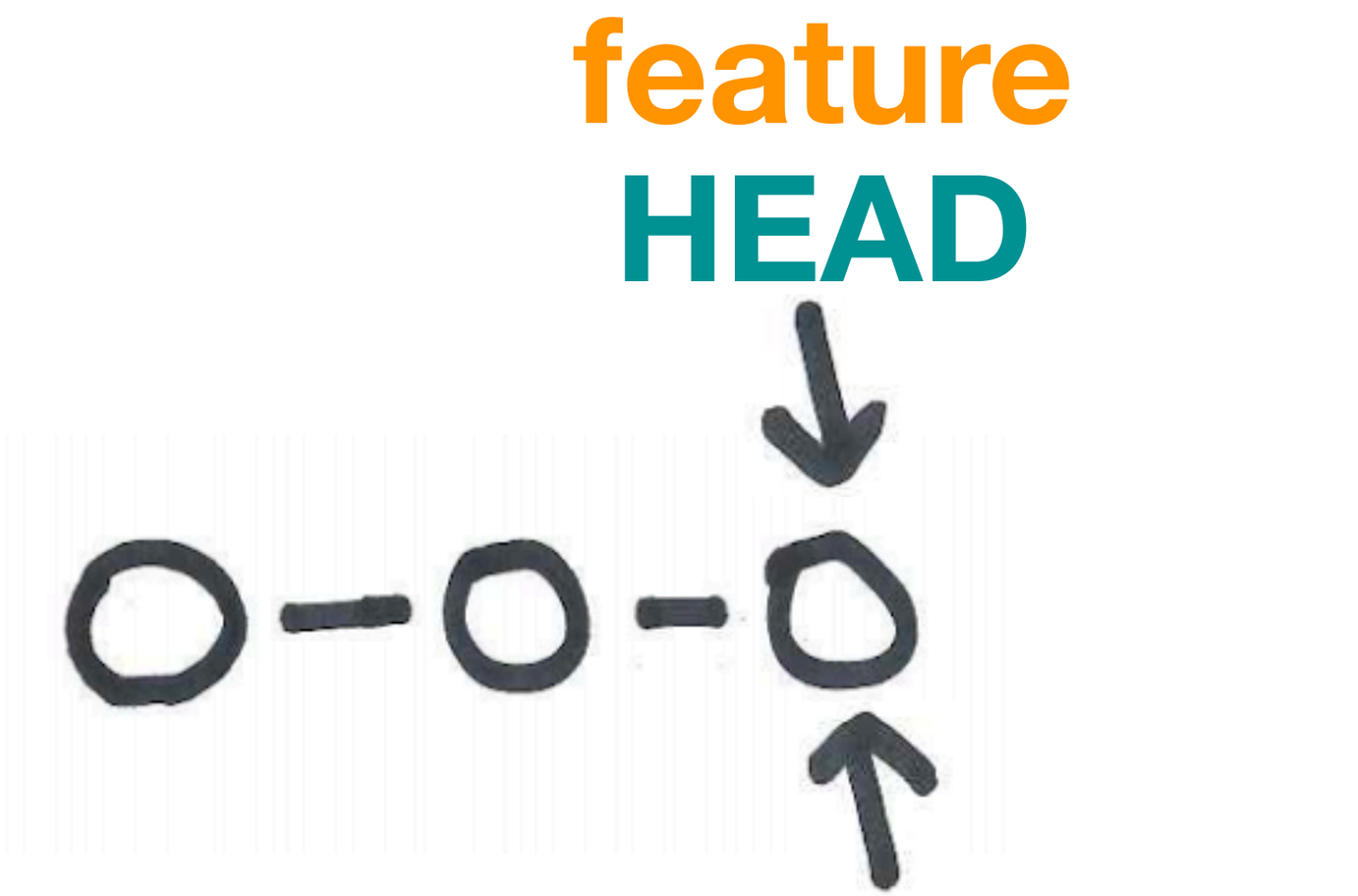
git user interfaces

- command line
  - most control and awareness of what you are doing
  - can be hard to visualize the process
- gui (GitKraken, SourceTree, Lazygit — terminal gui!)
  - good for visualizing the process
  - great interface for viewing history and diffs
  - encourages some good practices (viewing changes before adding)
  - easy to do powerful things (add parts of files, deal with merge conflicts, undo)
  - can obscure details or make it too easy to make mistakes
- GitHub
  - great interface for viewing history and diffs, but restricted to what's on the remote
  - required for issue tracking and pull request management

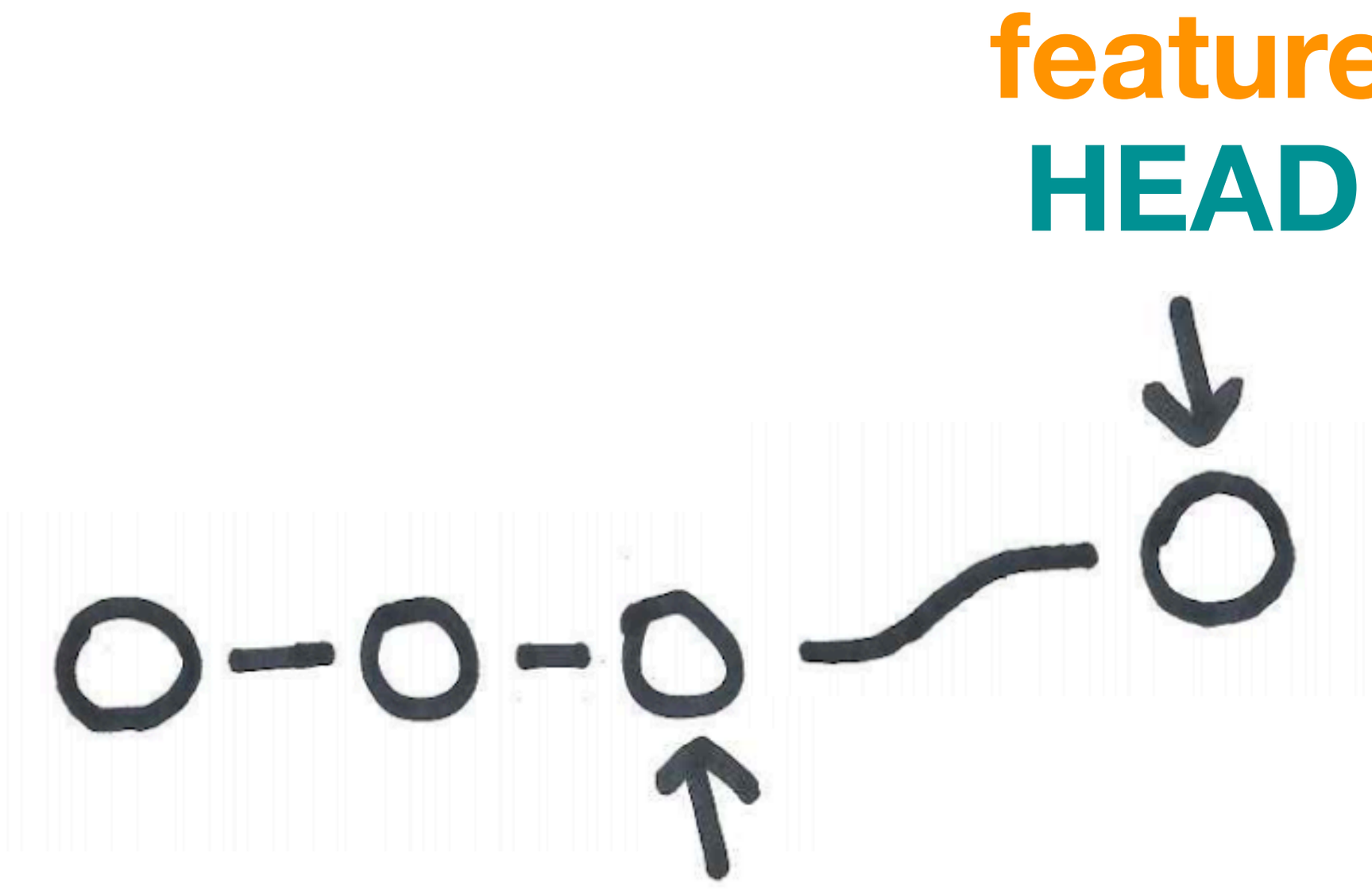

GitKraken
pyuvdata
cst_capitalization_and_dBi
Glo
Viewing 27/27
Show All
Filter (⌘ + Option + f)
LOCAL 3/3
cst_capitalization_an...
master
skip_optional
REMOTE 20/20
origin
apply_cal
beam_interp_save
casa
combine_uvdata_f...
cst_capitalization_...
Default_antpos
double_prec_uvws
h1c
master
parse_ants_plotting
phasing_test
remove_AIPY
round_frequencies
skip_optional
snap_convert
time_profiling
transfer_matrix
utils_get_miriad_a...
uvh5_memo
version_test
STASHES 1
// WIP
1
skip_optional
better skip if decorators
1 hour ago
master
fix tests
3 hours ago
Add a pending deprecation warning if no antenna positions
Fix beam interp docstring
beam_interp_save
Use a dictionary instead of a pair of lists. Slightly faster lookup
12 hours ago
Keep lists of beam interpolation functions and their corresponding frequ...
Test spline reuse for e-field beams
Add test and style
Correctly reuse the beam interpolation functions when the "reuse_spline"...
Save RectBivariateSpline callables in an array for faster reuse
new micro version
better skipping of tests for optional dependencies
Covering line 781 in utils.py
2 days ago
Adjust antenna positions in test file to trigger specific test case
Update utils.py
Docstring updates
Default behavior of get_antenna_redundancies enforces baseline orienta...
use_ant_pos in unphasing and remove uvw rounding in get_baseline_red...
Handle floating errors in redundancies
Unit test the number of redundant groups
Spelling fix
Add to tutorial
Allow for flipped baselines to be considered redundant, like in hera_cal
only add 'Tile' to antenna name for MWA
6 days ago
break up and improve fhd tests, remove scipy/numpy warning
1 file change on cst_capitalization_and_dBi
Path
Tree
Unstaged Files (1)
Stage all changes
pyuvdata/tests/test_cst_beam.py
Staged Files (0)
Commit Message
Amend
Summary
Description
Stage files/changes to commit

branching

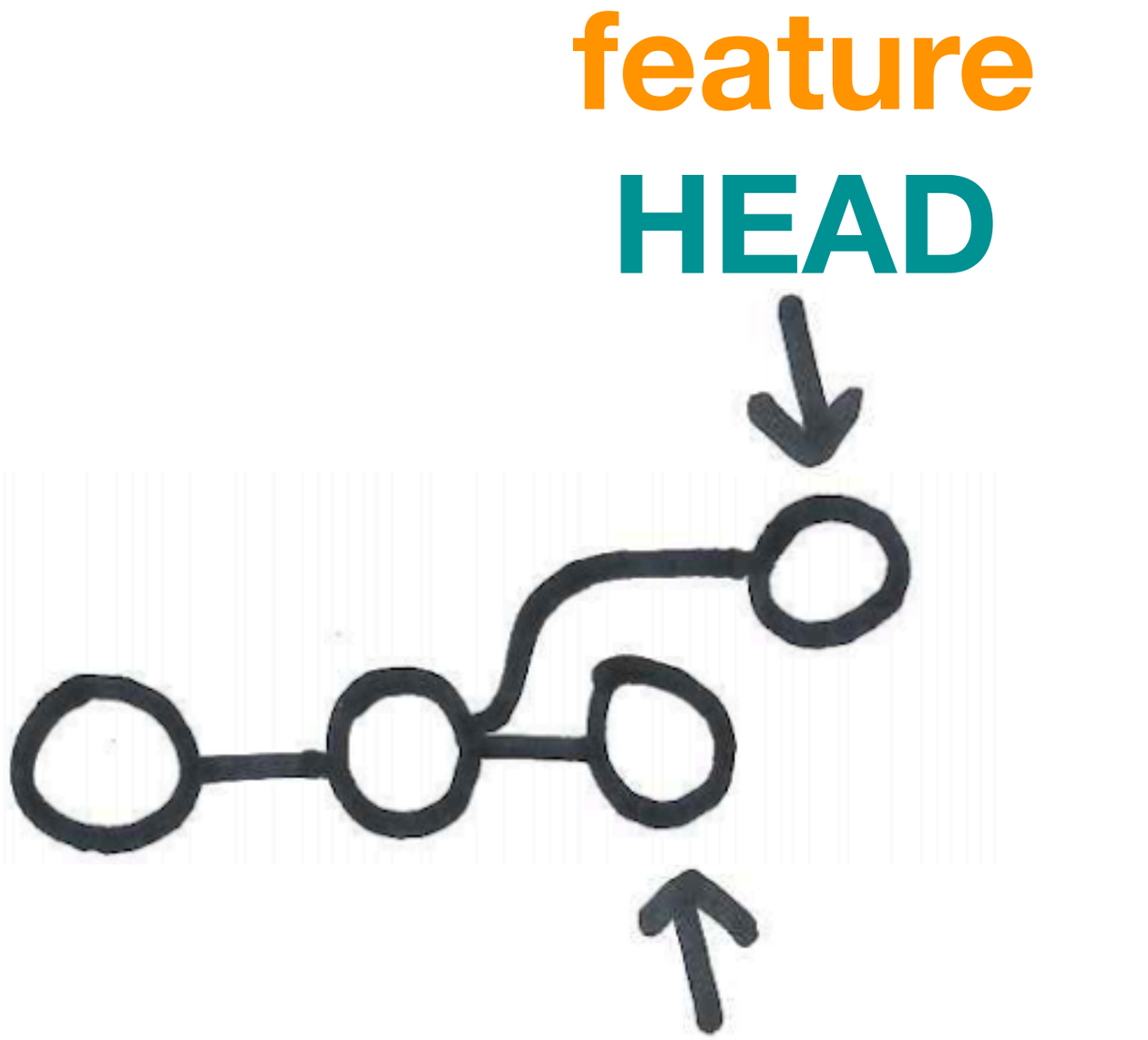
branching



branching



branching



branching

- create a new branch and switch to it
  - use `git checkout -b <branch_name>` to create the branch and switch to it
  - for existing branches, use `git checkout <branch_name>` to switch to that branch
- make changes and snapshots on that branch
- push the branch up to the remote
  - use `git push --set-upstream origin <branch_name>` to make a branch on the remote that tracks your new local branch
  - make more changes and snapshots and push/pull
- to merge the branch into the main, make a pull request
  - leads to a code review

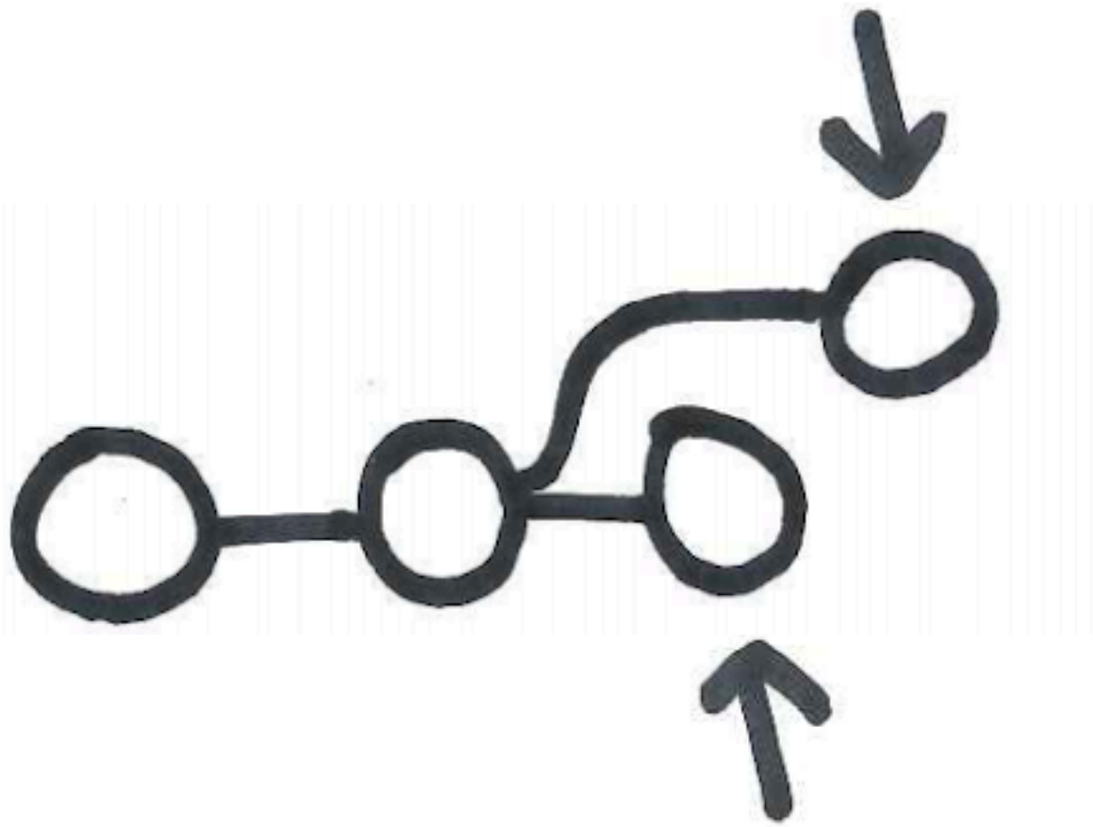
merging vs rebasing branches

- merging in branches is straightforward, but can result in a somewhat complicated graph
- rebasing is an alternative approach that results in a neat, linear graph at the expense of rewriting history
- rebasing effectively moves the location that a branch leaves the tree
  - can be used to place branches at the tip of the master branch to avoid having to merge
  - effectively replays the changes in the branch after the end of the master branch

branch setup

feature

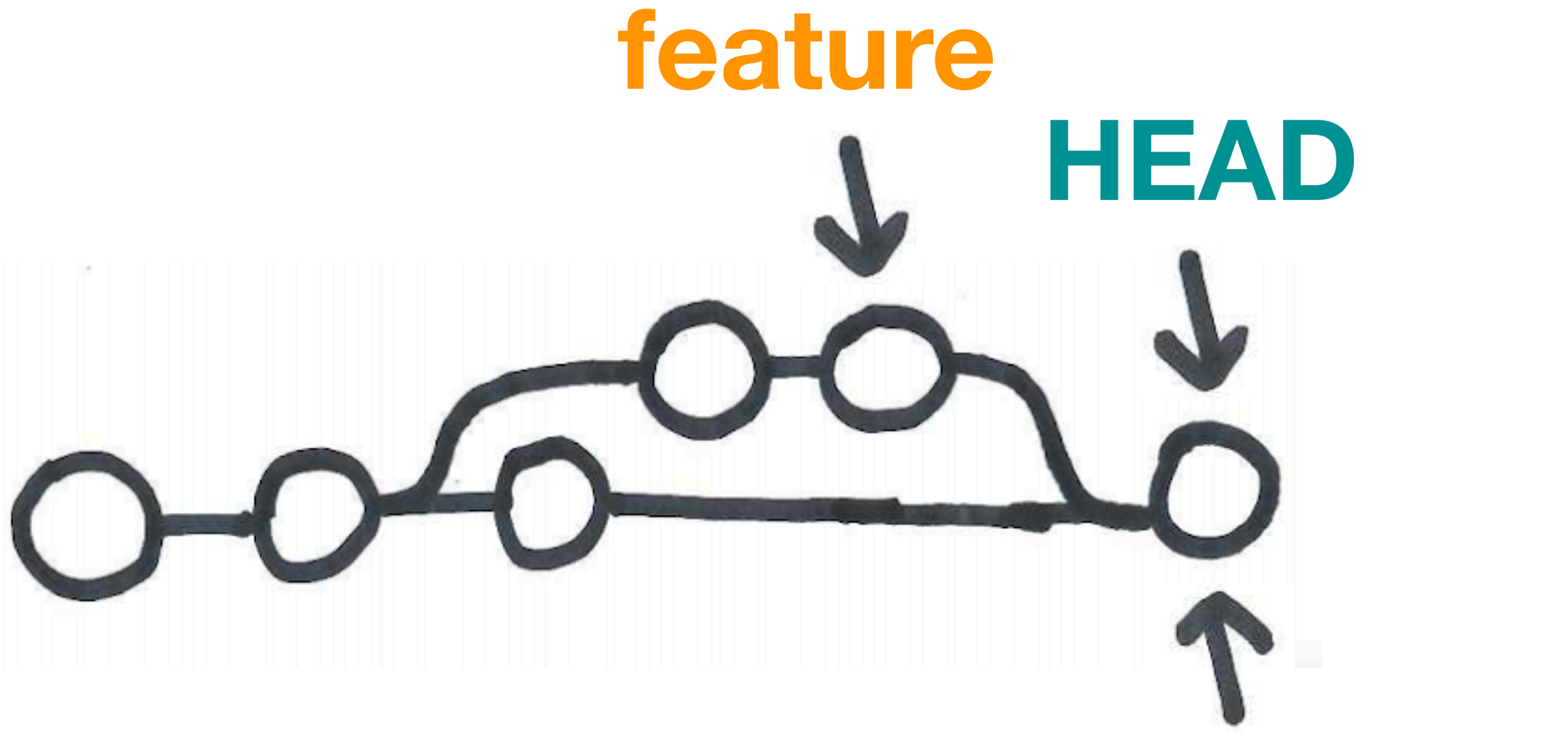
HEAD



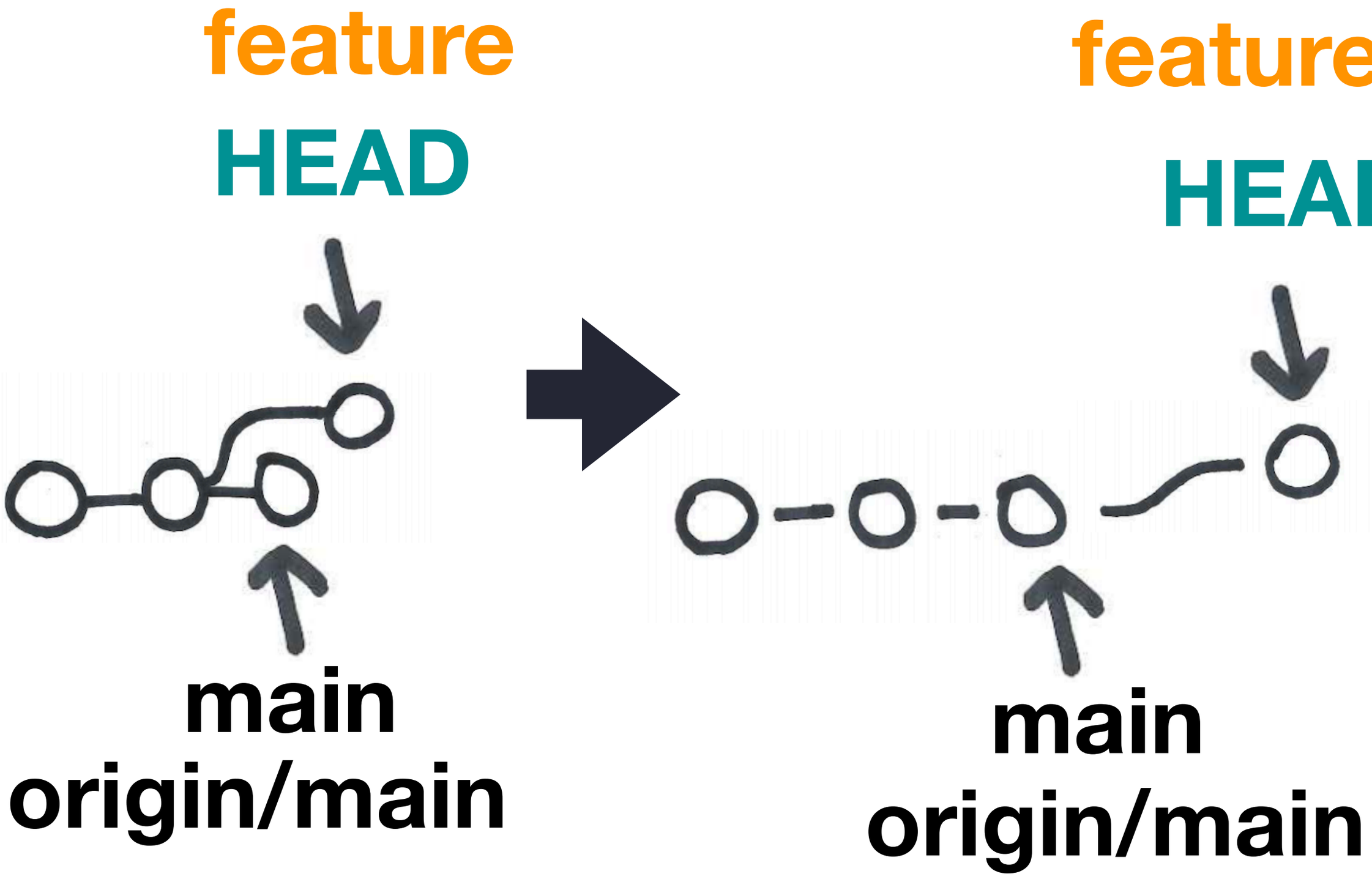
main

origin/main

merging branch



rebasing branch



Making a new repository

- On GitHub
  - choose public or private
  - initialize with a readme
  - choose a .gitignore (also see https://github.com/github/gitignore for more language options, including matlab)
  - choose a license

- Clone the repository locally
  - ssh vs https
  - `git clone <repo-address>`
  - `git remote -v` to see the address of the remote

git config

- Global settings for git
  - name and email address (to identify who made changes) — should match email associated with your GitHub account
  - preferred text editor (for commit messages)
- To see current settings:
  - `git config –list`
- To change these settings:
  - `git config --global user.name "Vlad Dracula"`
  - `git config --global user.email "vlad@tran.sylvan.ia"`
  - `git config --global core.editor "nano -w”`

Resources

- git parable (conceptually building up why git is the way it is): https://tom.preston-werner.com/2009/05/19/the-git-parable.html

- Software Carpentry hands-on tutorial: http://swcarpentry.github.io/git-novice/

- Lab-style git intro: https://github.com/HERA-Team/CHAMP_Bootcamp/blob/master/Lesson2_IntroToComputing/git-lab-handout.pdf

- eScience office hours (https://escience.washington.edu/office-hours/#eScienceDataScientists)