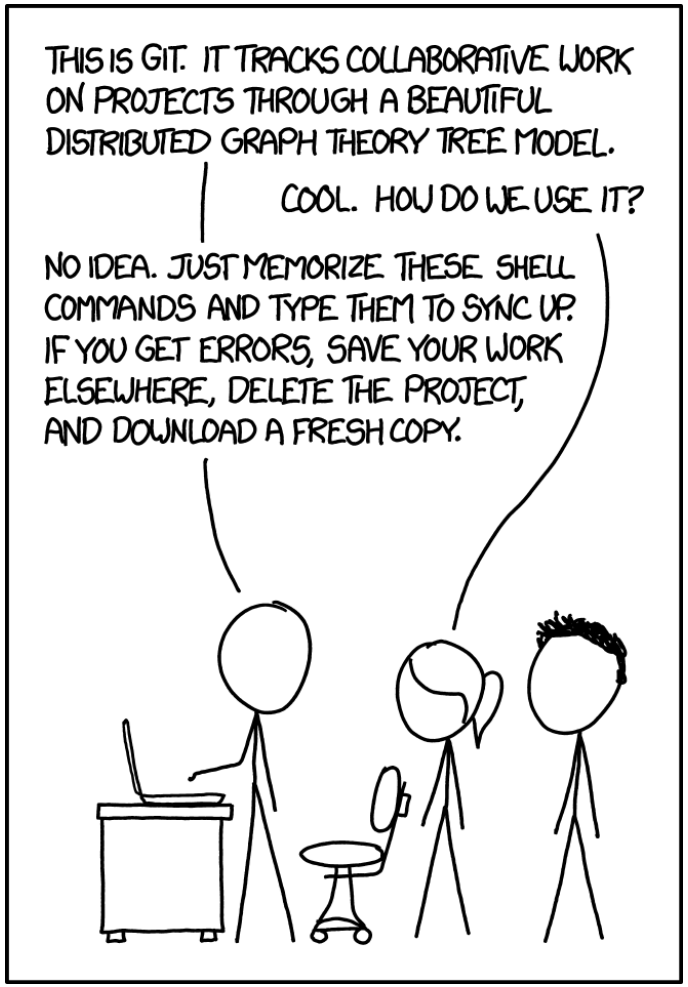
2. Git

What is Git?

- Fully distributed <u>version control</u> system
  - Although it's usually used in a centralized way
- Objective: Allows collaboration between programmers working on the same project
  - What problems would usually arise?
- Originally created by Linus Torvalds
- Other alternatives?

What even is Git?



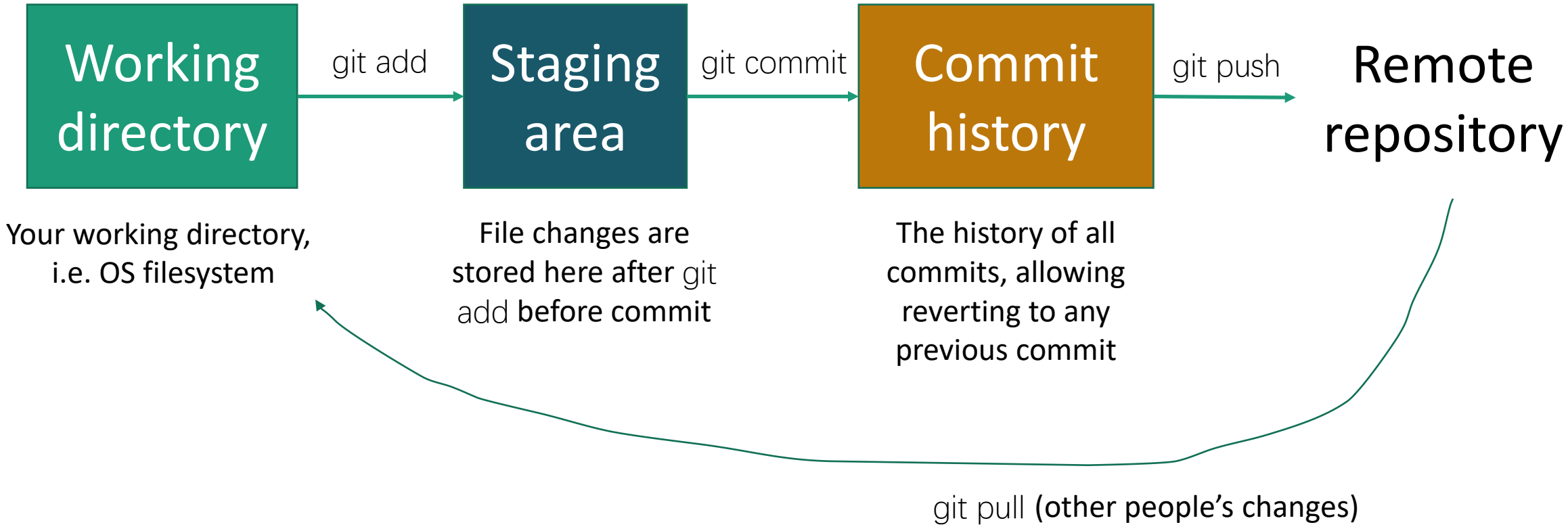
xkcd

Startup

- Two ways to do so:
  - Go to a website (e.g. Github), create a repository there, follow instructions to clone it
  - git init on a server, then git clone on your local machine
- git init sets up the current directory as a git repository
- git clone <repo> sets up the current directory as a git repository by linking it to a repository that has already been set up
  - repo link can be ssh://, http(s://, git://
- git config to set up your name and e-mail address

The Three Trees of Git

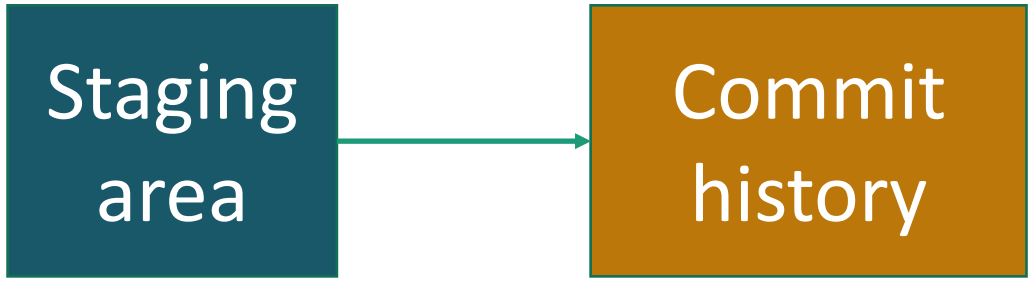
- *Three* stages to push changes to remote…



git add

- git add <file> to add a specific file to the staging area
  - git add <directory> to add all files in a folder
  - If you change a file after adding it, these changes would *not* be committed
- What types of files should be in a git repository?
  - What types of files should not be in a git repository?
- The staging area is purely local
- What's the point of the staging area?
  - You often want to divide your changes into several commits
  - You want to have exact control over what changes you're committing

git commit



- git commit **to commit all added changes to history**
  - git commit -a **automatically adds** all changes in this directory **to files that have been added at some point in the past, then commits**
  - A commit is a permanent "save point" – you can revert to any commit at any point
  - Next step: write a commit comment
- What/when should you commit?
  - Key word: **atomic**
  - e.g. one bugfix = one commit, one feature = one commit
  - Generally, smaller is better
  - Don't commit: unfinished code, untested code, code that doesn't compile

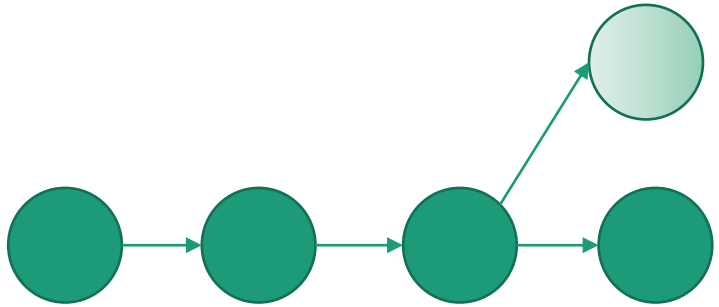
Commit message best practices

- git commit –m “fixed bug” ✕
- First line: title (<50 characters) – what and why
  - Use *imperative mood*. “Add support for tensorflow 2.7.0”
- Second line: empty
- Third line onwards: description, if necessary (<70 characters/line)

```
commit ae878fc8b9761d099a4145617e4a48cbeb390623
Author: [removed]
Date:   Fri Jun 1 01:44:02 2012 +0000

  Refactor libvirt create calls

   - Minimize duplicated code for create

   - Make wait_for_destroy happen on shutdown instead of undefine

   - Allow for destruction of an instance while leaving the domain
```

Branches

- This is where things get complicated



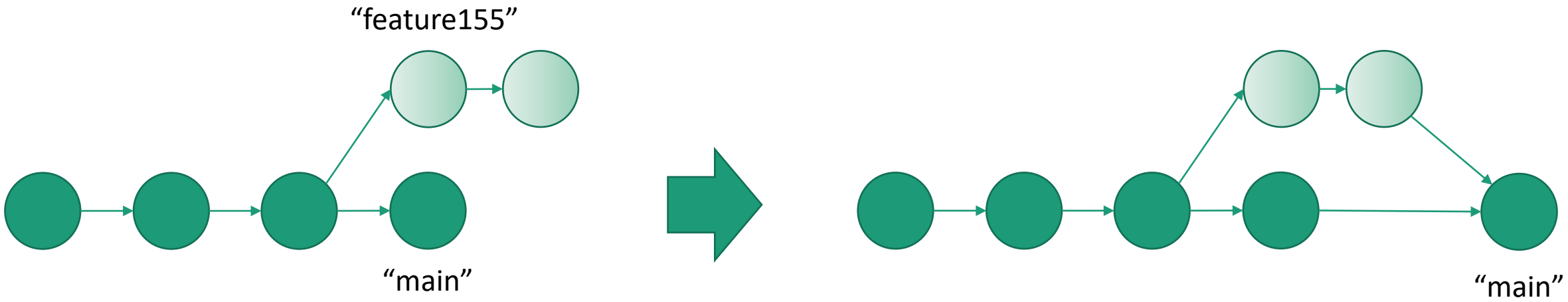
- Branches are useful for feature implementations that require multiple commits
- git branch lists all branches
- git branch <name> creates a branch <name>
- git branch –d <name> deletes that branch
- git checkout <name> switches to that branch
  - Note that creating a branch does not switch you to that branch automatically

git merge

- You finished your feature! Now you want to merge your branch back into main…
- Case 1: fast-forward merge
  - git checkout main
  - git merge feature155
  - git branch –d feature155

git merge

- Case 2: three-way merge
  - Same commands



- Conflicts may arise… (when?)
- After fixing them, git commit to finalize the merge

git fetch

- Everything before this was local
- Think of origin/main and main as <u>two different branches</u>
  - Origin is set up to be the repository you cloned from
  - The former is a "remote-tracking branch", the latter is a local branch
- git fetch <remote> updates all your remote-tracking branches by checking a remote repository (you can also specify one branch)
  - Default <remote> is origin
  - None of your local branches are affected – use git merge if you want to update them
- git pull <remote> does a fetch, then merges all remote changes with your local ones, creating merge commits (or fast-forward merges)

git fetch

- If someone has created a new branch, you will get it with git fetch, and you can check it out with git checkout
- This puts you in a detached HEAD state, because you should only work on local branches
  - git branch –b <local_name> to create the corresponding local branch
- git pull can fail (when?)
  - Commit your untracked files and resolve the merge conflict
  - Or, stash your untracked files (stashes are purely local and do not interact with commits)
    - Later, you can re-apply or delete your stash

git push

- This is how you push to remote
- git push <remote>
  - Can specify one branch or all branches
  - Default is origin
  - You can also see all remote aliases with git remote -v
- A push fails if someone has pushed to that branch after you pulled
  - Exception: a fast-forward merge is possible
  - Another good reason to use branches
  - git push --force ignores the check (**don't do this**)

Other (very) useful tools

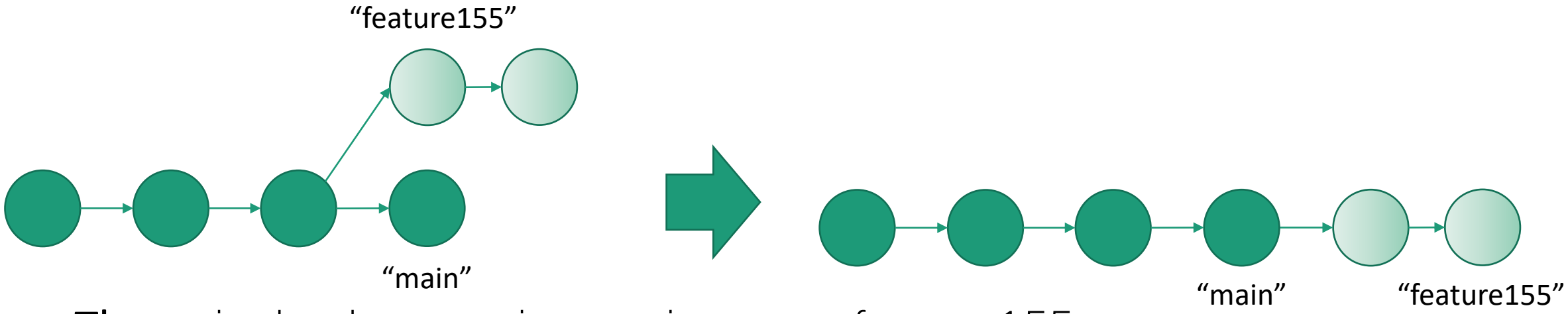
- git status – shows what files you haven't added or haven't committed
  - Also helps with merge conflict resolution
- git log – shows a log of all commits before this branch
  - Other branches are not shown – unless you add -a
- .gitignore – a local file that dictates what files should be ignored by git, such as by git add –a, git status and git stash
- git diff – show you differences between two commits, or two files, or two branches

Reverting history

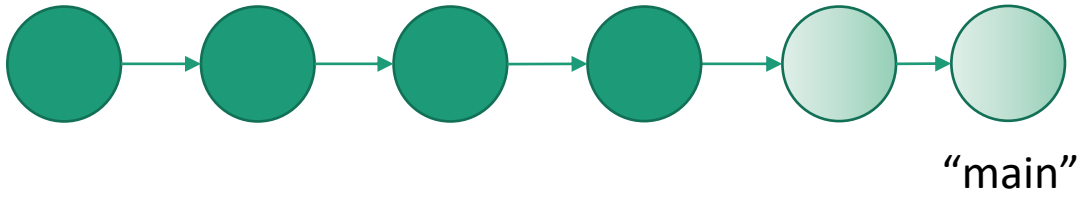
- #1 Rule: Don’t revert public history
  - But reverting local history can be helpful for organization, making good commits, etc.
- git commit --amend
  - Rewrites the previous commit with all changes made since that commit
  - Logical workflow: git add, git commit, work on some more files, git add, git commit --amend
- git reset – undo an add, or undo a commit
  - Beware: this can cause you to permanently lose changes!

Reverting history

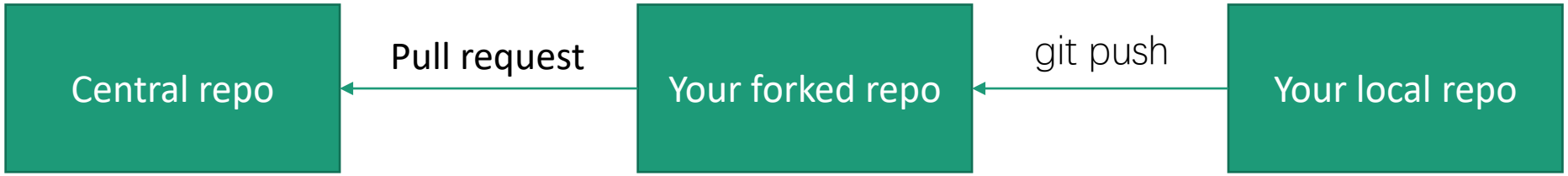
- git rebase - **an alternative to merging**
- git checkout feature155 -> git rebase main



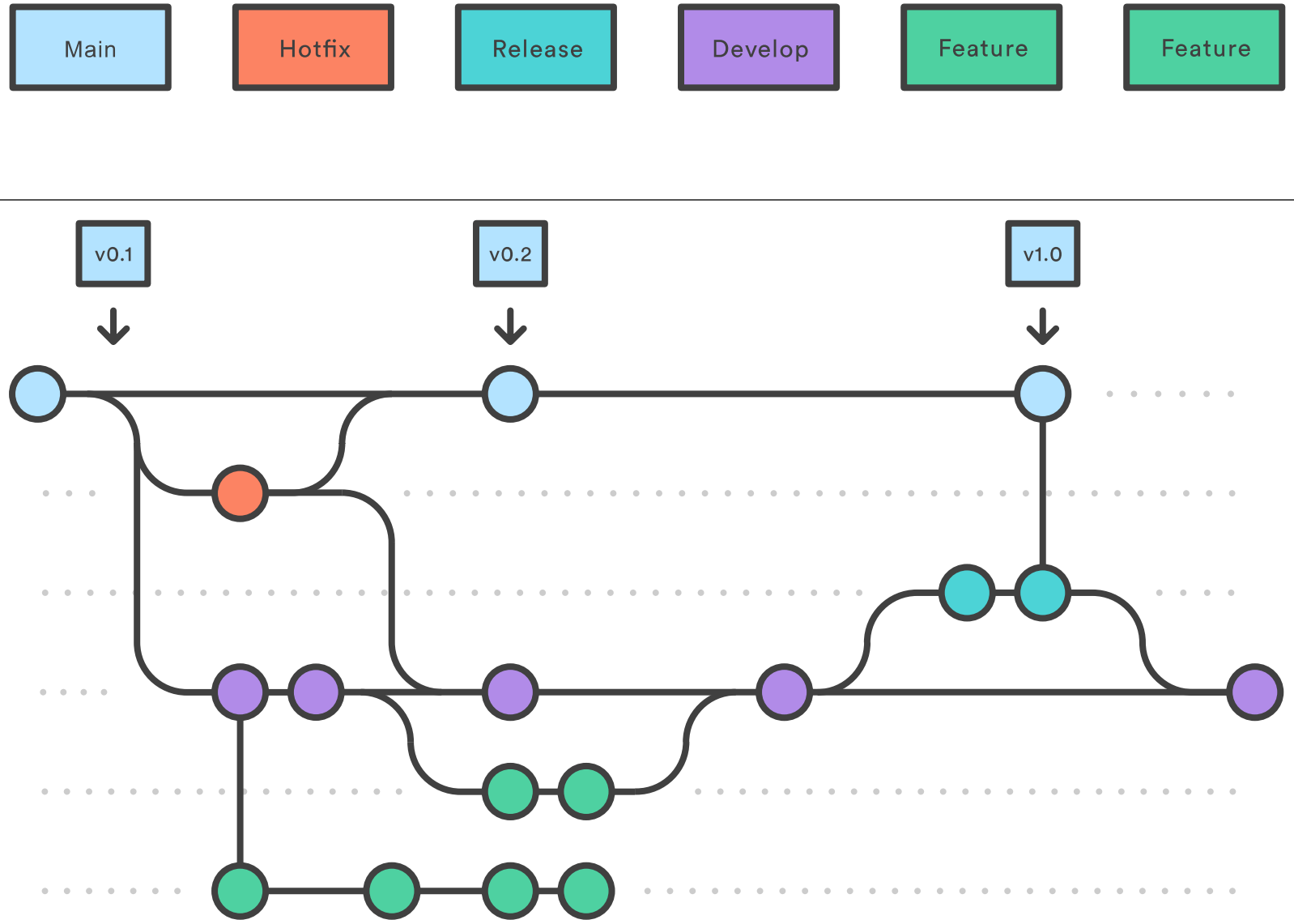
- **Then** git checkout main -> git merge feature155



Pull request



- For public collaboration projects, you won't be able to push directly to the central repository
- Steps:
  - Fork the central repo into your own server-side repository
  - Clone the forked repo into your local machine
  - Finish your work, push to your forked repo
  - Make a pull request to the central repo
  - Reviewer checks your code, possibly requests changes, and git pulls



from Atlassian Bitbucket

Project details

- You will make a board game ☺
- If your team cannot agree on a language, Java is recommended as a default
- You can implement an existing game
  - You may add or remove features of this game as desired
- The following components are necessary:
  - A GUI, to play and interact with the game (cannot be pure text UI)
  - Different game modes, challenges and achievements
  - Implementation of game rules and game elements
  - Testing and debug features
  - A rational AI
- Detailed submission instructions for Phase 1 to be posted