Principles of Software Construction

Version Control with Git

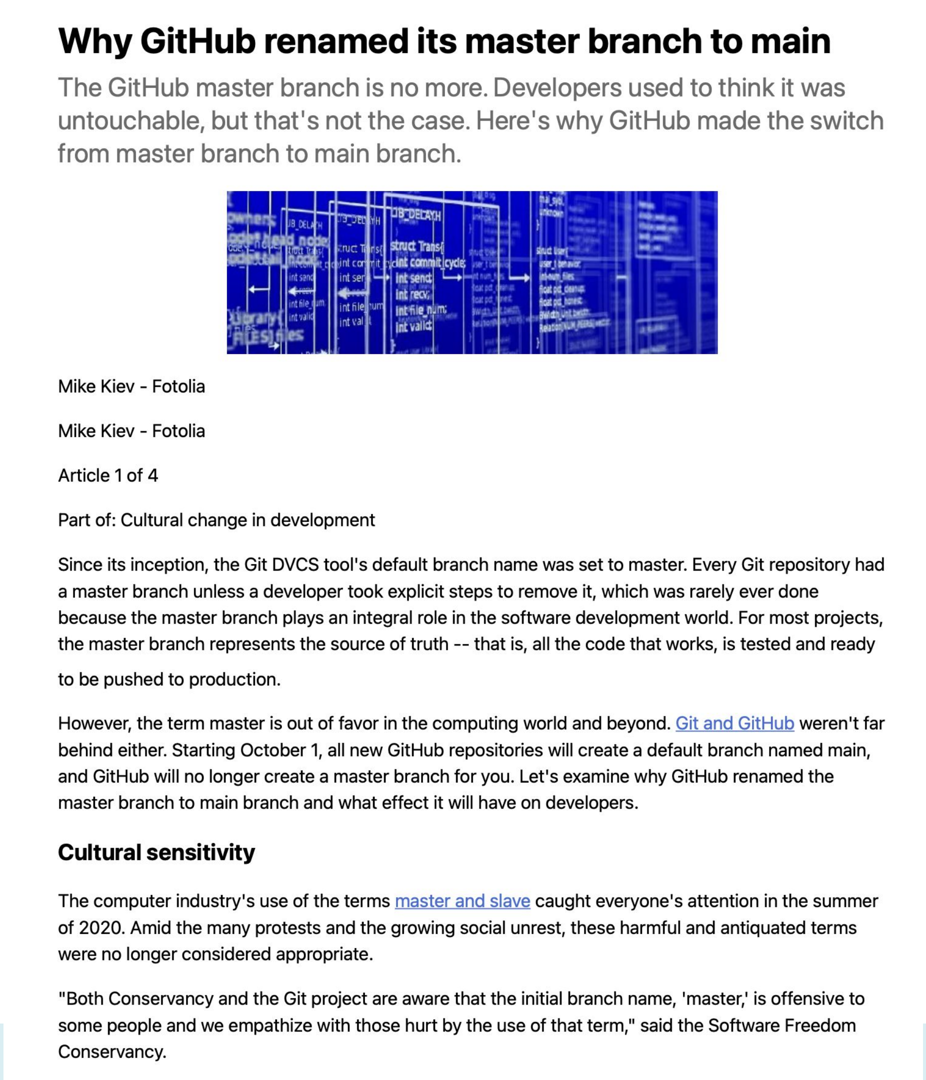
Claire Le Goues **Bogdan Vasilescu**

Carnegie Mellon University
School of Computer Science

institute for SOFTWARE RESEARCH

GIT BASICS

Graphics by https://learngitbranching.js.org



Why GitHub renamed its master branch to main

The GitHub master branch is no more. Developers used to think it was untouchable, but that's not the case. Here's why GitHub made the switch from master branch to main branch.

Mike Kiev - Fotolia

Mike Kiev - Fotolia

Article 1 of 4

Part of: Cultural change in development

Since its inception, the Git DVCS tool's default branch name was set to master. Every Git repository had a master branch unless a developer took explicit steps to remove it, which was rarely ever done because the master branch plays an integral role in the software development world. For most projects, the master branch represents the source of truth -- that is, all the code that works, is tested and ready to be pushed to production.

However, the term master is out of favor in the computing world and beyond. Git and GitHub weren't far behind either. Starting October 1, all new GitHub repositories will create a default branch named main, and GitHub will no longer create a master branch for you. Let's examine why GitHub renamed the master branch to main branch and what effect it will have on developers.

Cultural sensitivity

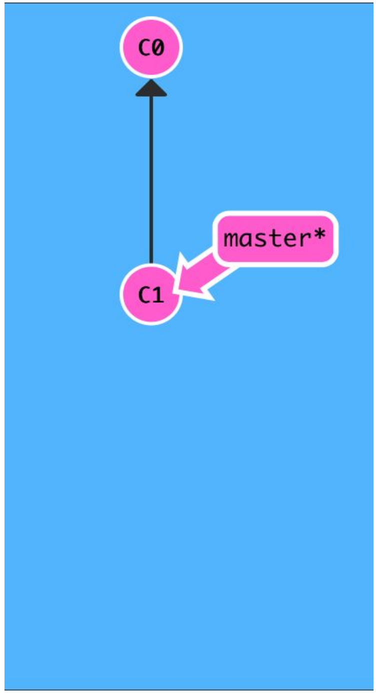
The computer industry's use of the terms master and slave caught everyone's attention in the summer of 2020. Amid the many protests and the growing social unrest, these harmful and antiquated terms were no longer considered appropriate.

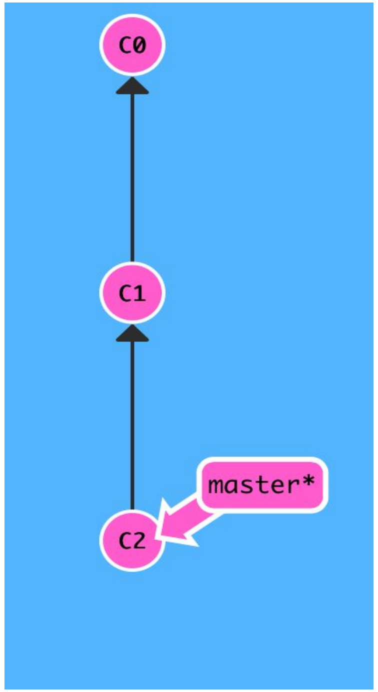
"Both Conservancy and the Git project are aware that the initial branch name, 'master,' is offensive to some people and we empathize with those hurt by the use of that term," said the Software Freedom Conservancy.

Note: Some slides and imagery use discouraged terminology. Sorry I didn't get a chance to update!

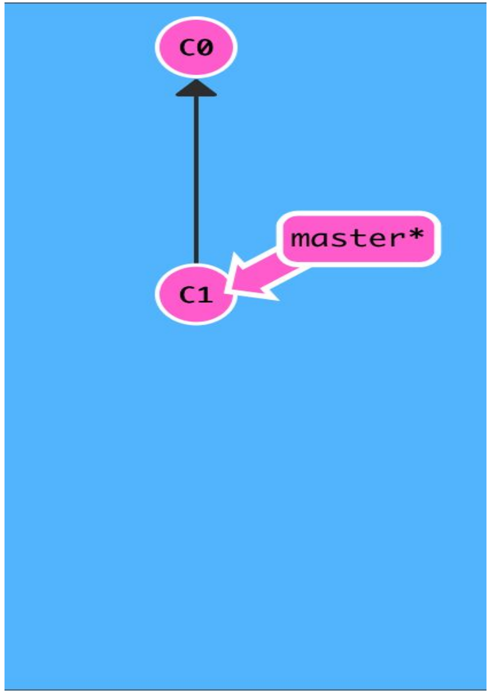
https://www.theserverside.com/feature/Why-GitHub-renamed-its-master-branch-to-main

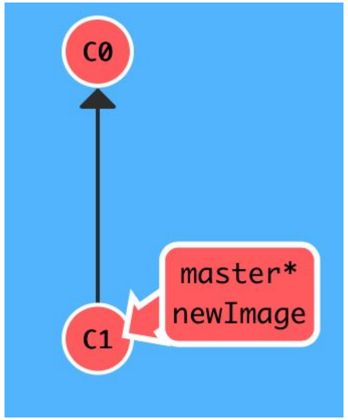
git commit



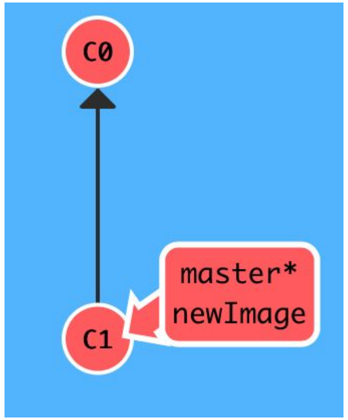


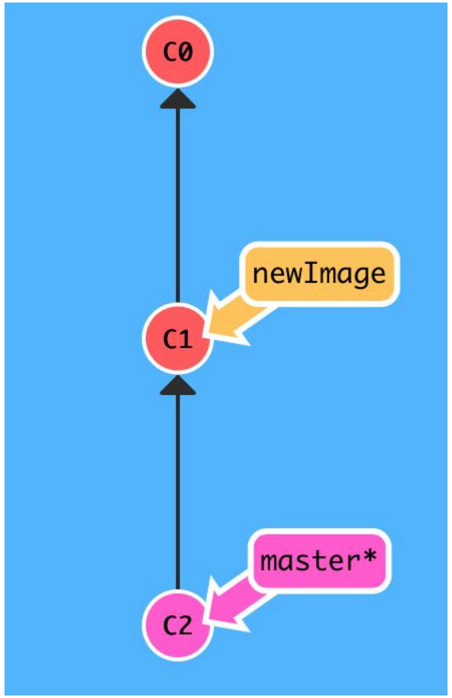
git branch newImage



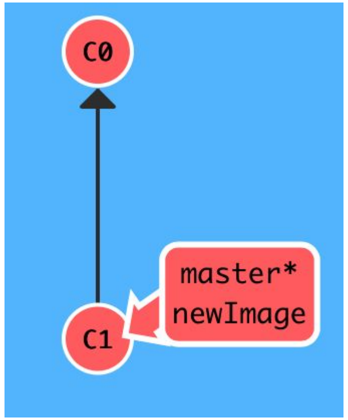


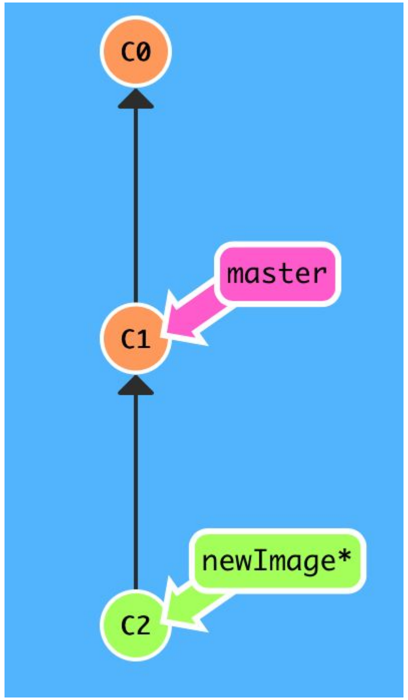
git commit





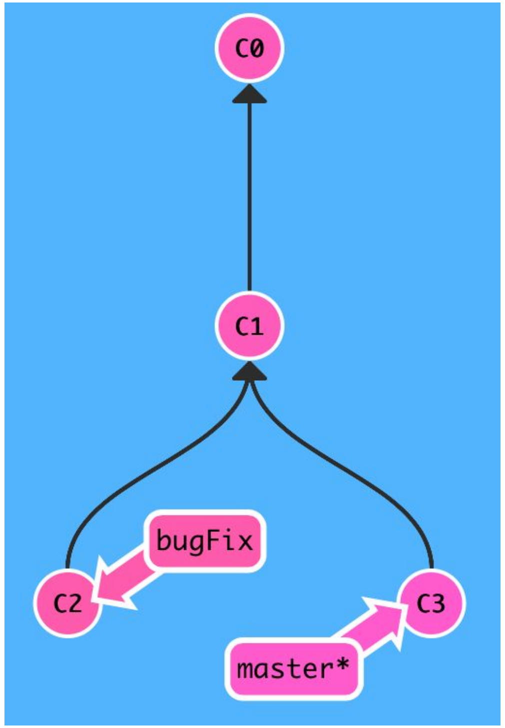
`git checkout newImage; git commit`

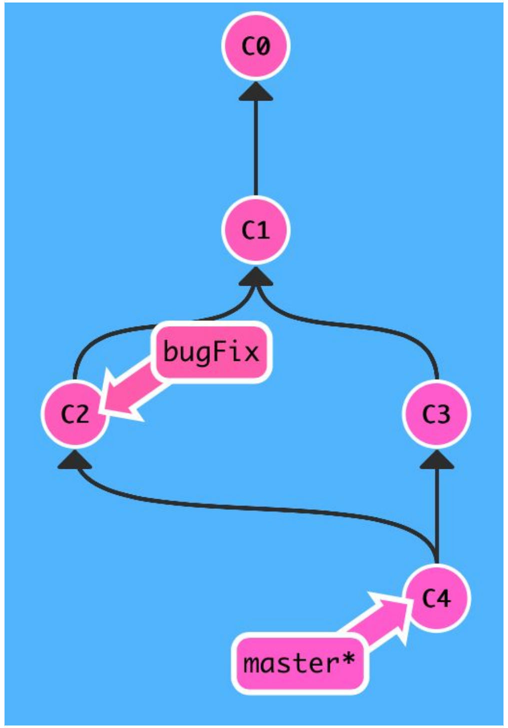




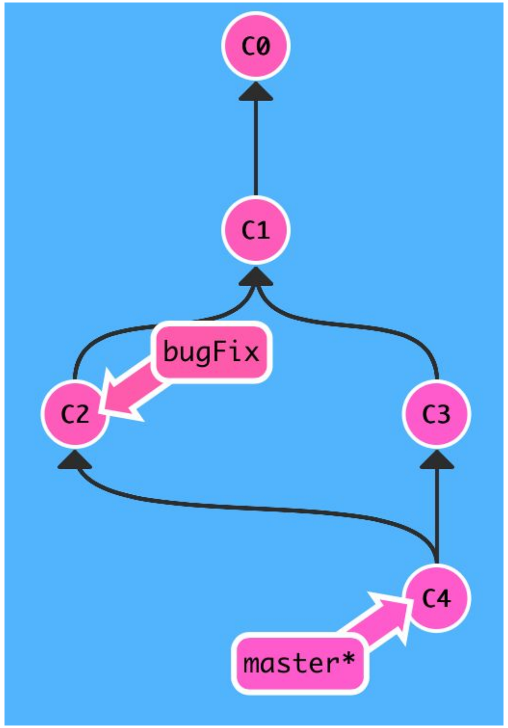
Three ways to move work around between branches

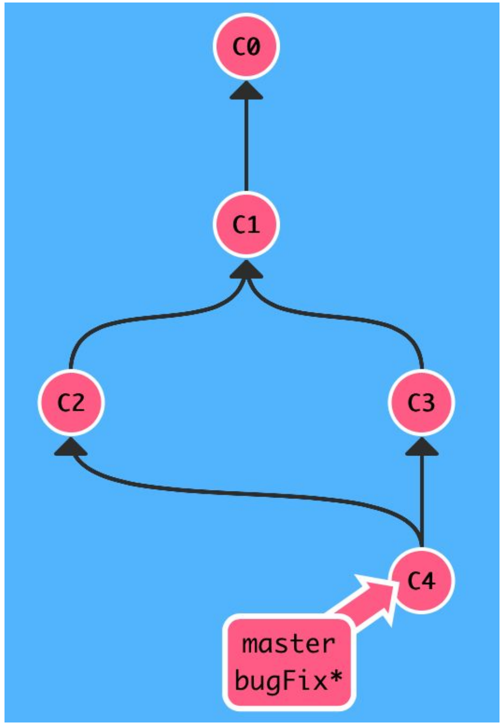
1) `git merge bugFix` (into `master`)



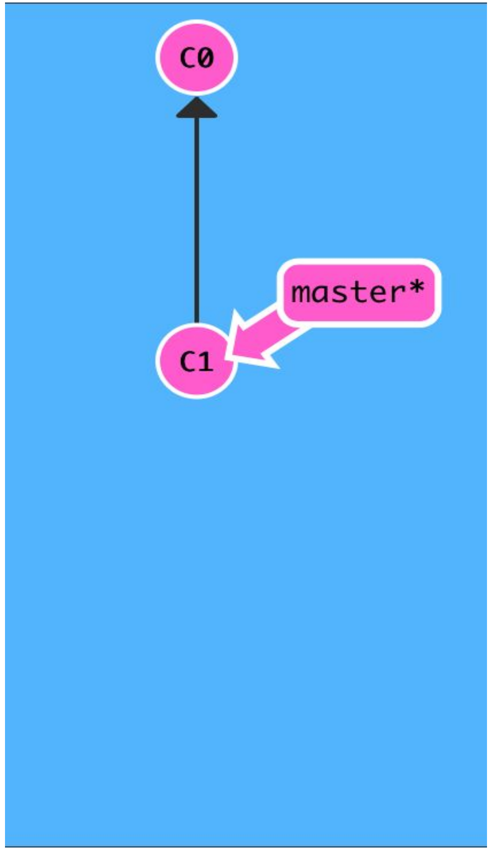


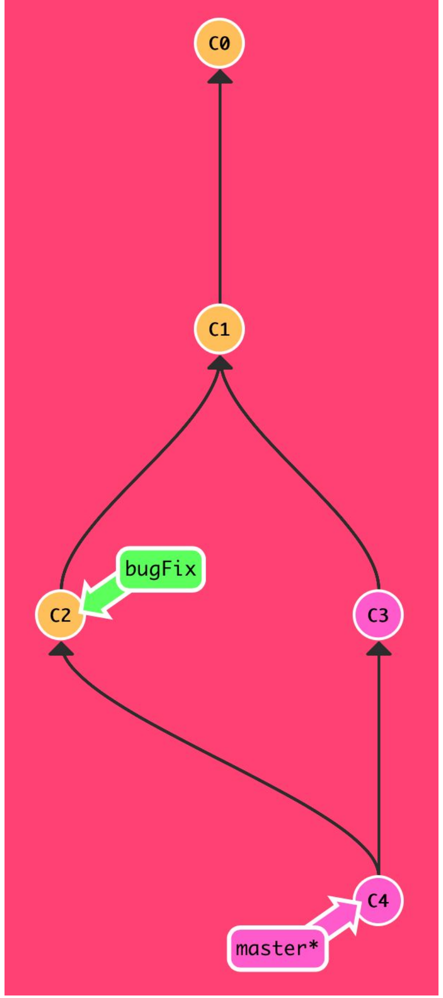
git checkout bugfix; git merge master (into bugFix)





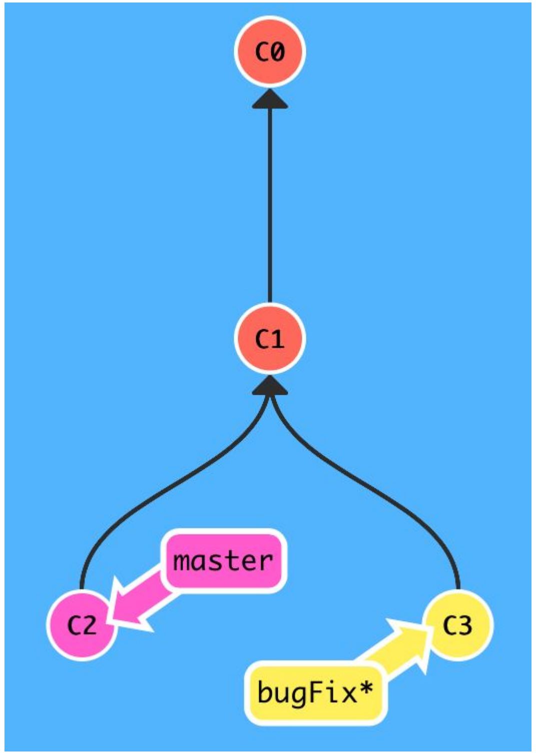
Activity:

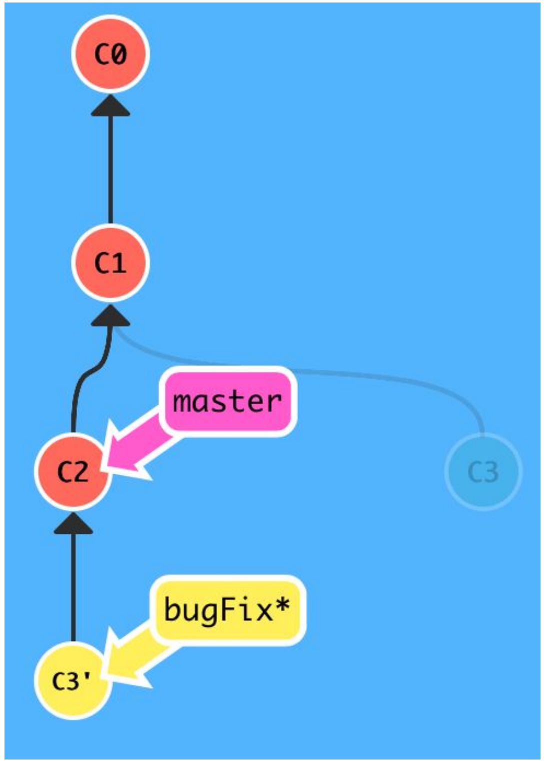




Move work from bugFix directly onto master

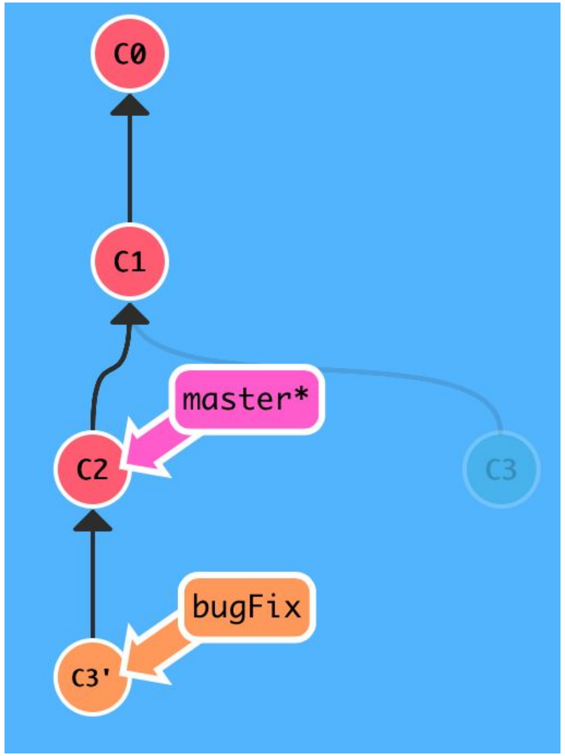
```
2) git rebase master
```

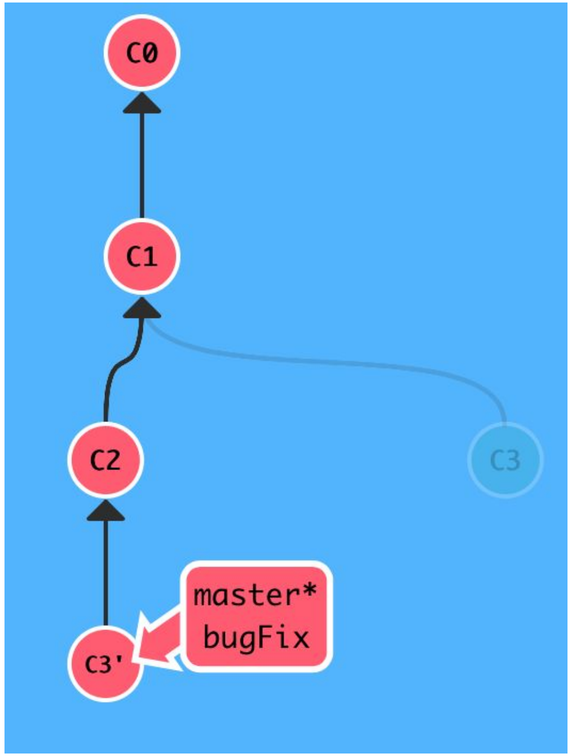




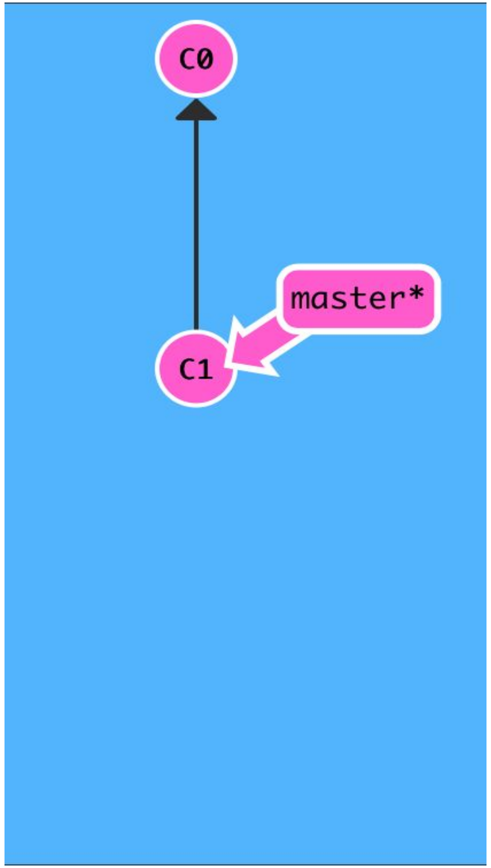
But master hasn't been updated, so:

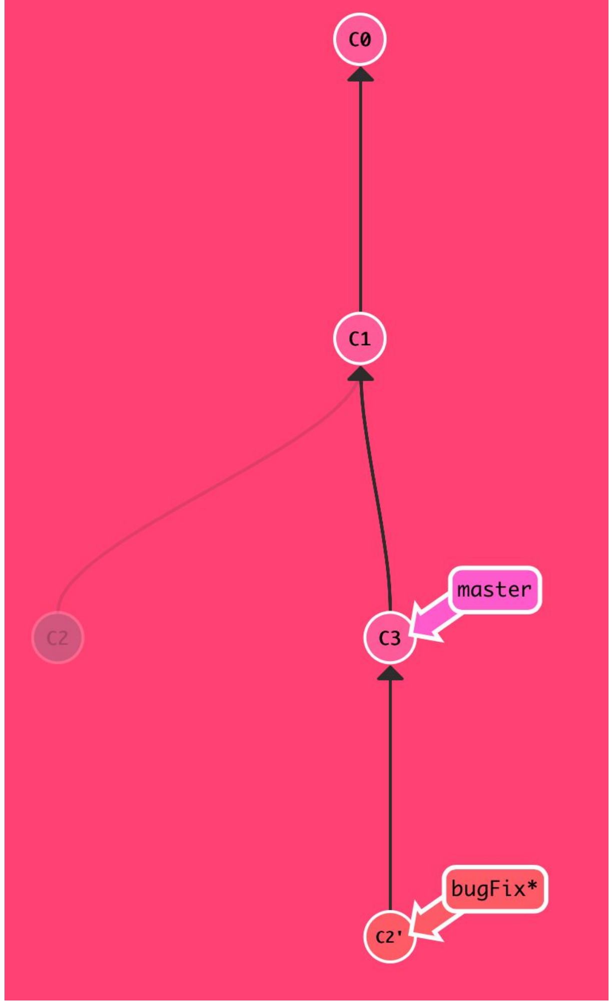
```
git checkout master; git rebase bugFix
```





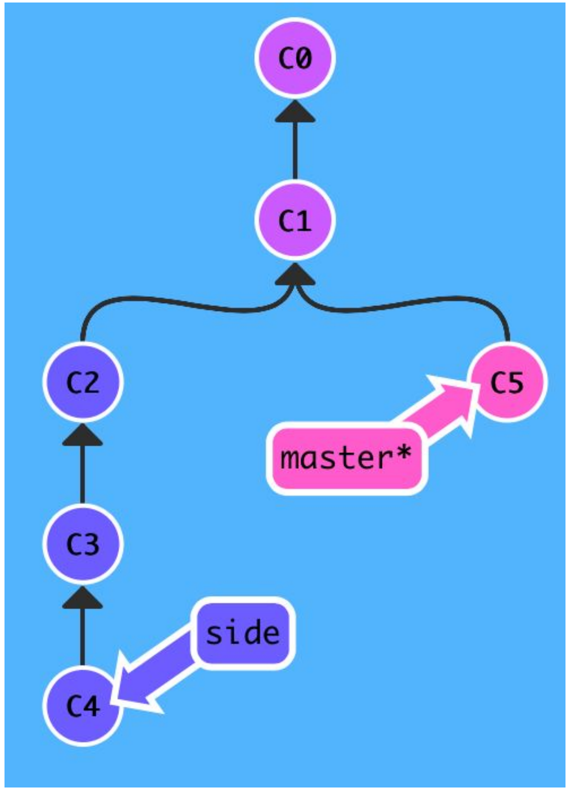
Activity:

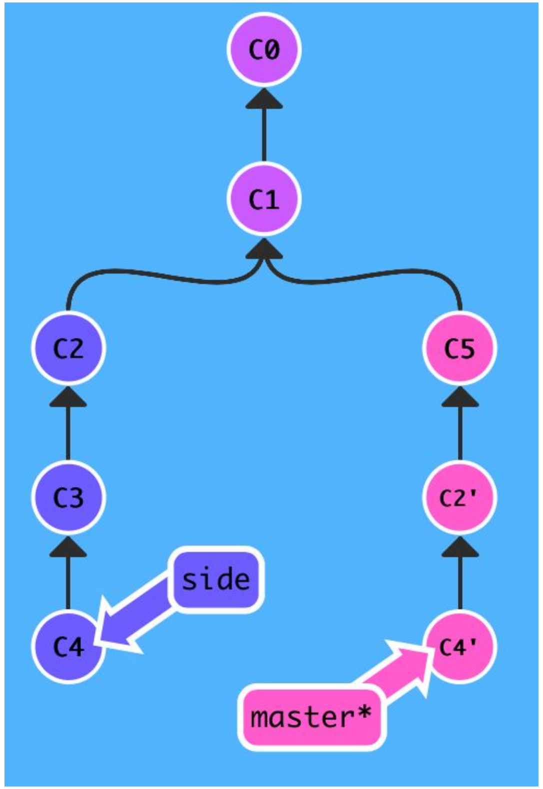




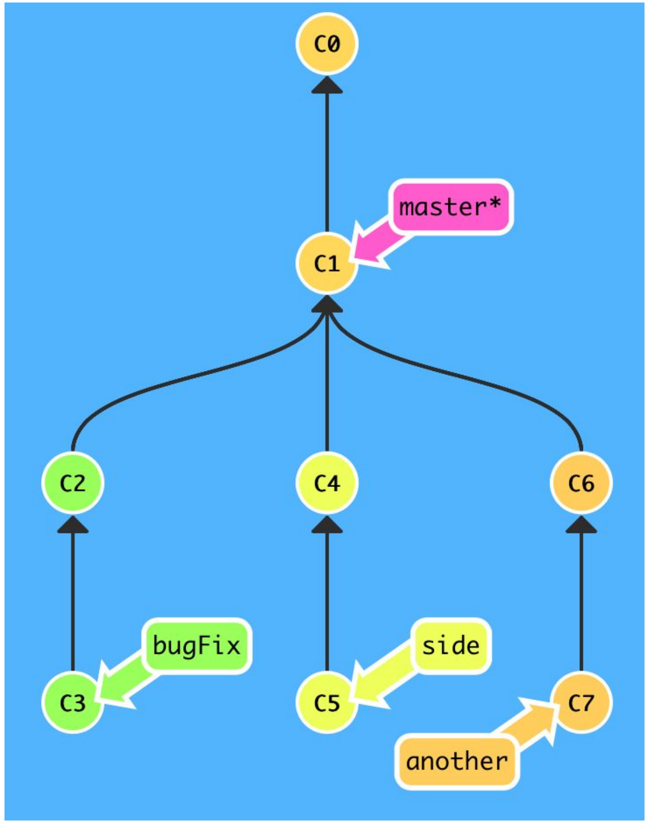
Copy a series of commits below current location

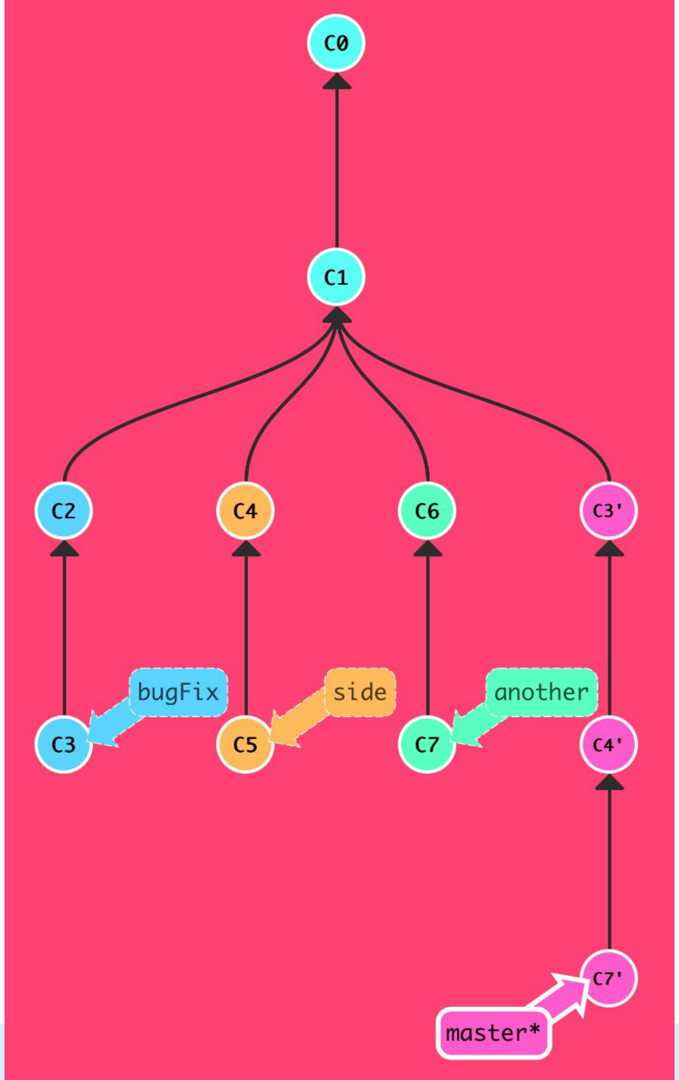
```
3) git cherry-pick C2 C4
```





Activity:

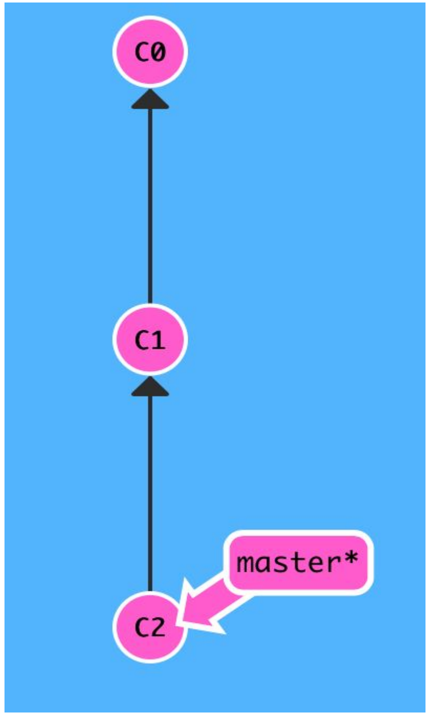

C0
master*
C1
C2
C4
C6
bugFix
side
C3
C5
C7
another

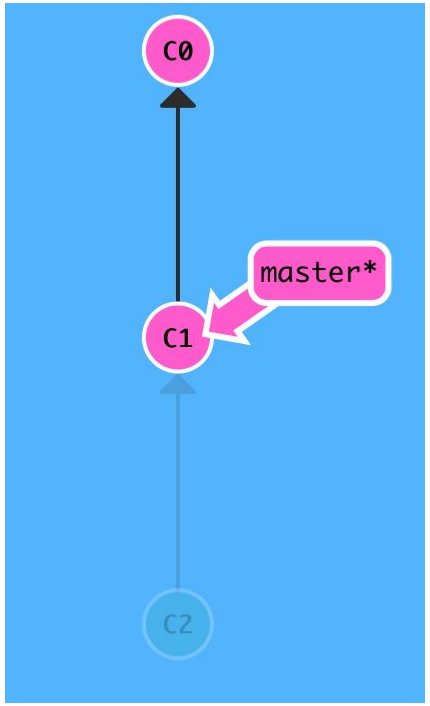

C0
C1
C2
C4
C6
C3'
bugFix
side
another
C3
C5
C7
C4'
C7'
master*

Ways to undo work (1)

```
git reset HEAD~1
```

HEAD is the symbolic name for the currently checked out commit

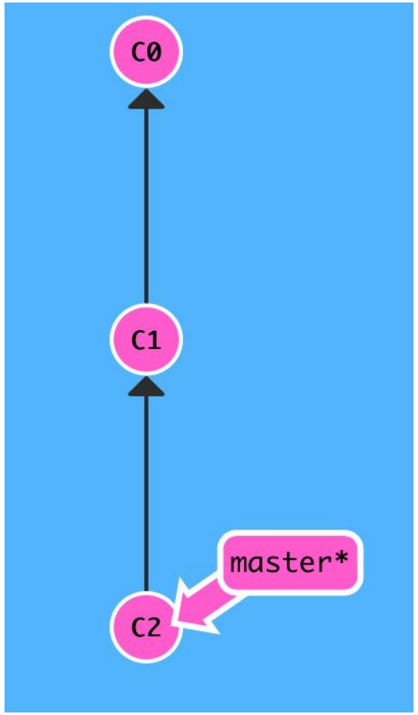


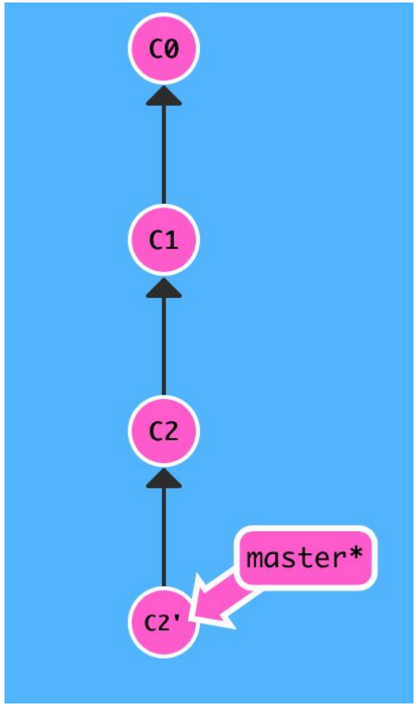


Ways to undo work (2)

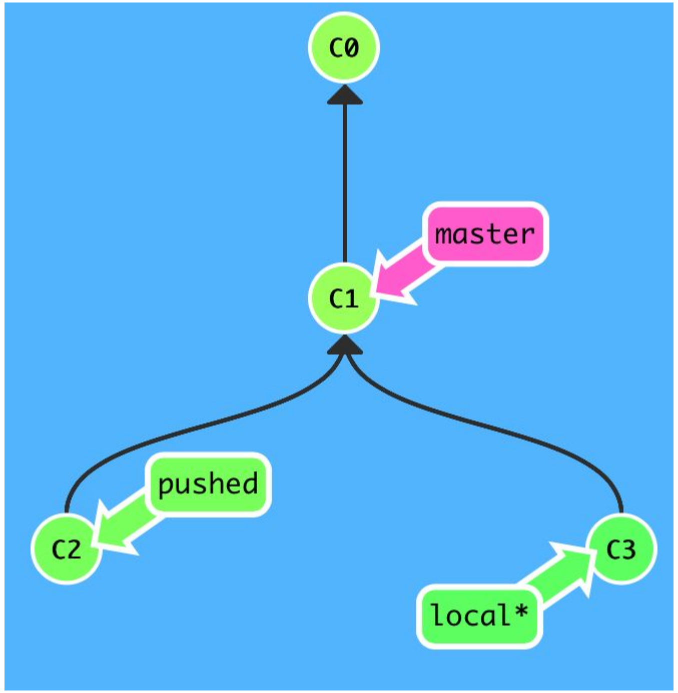
```
git revert HEAD
```

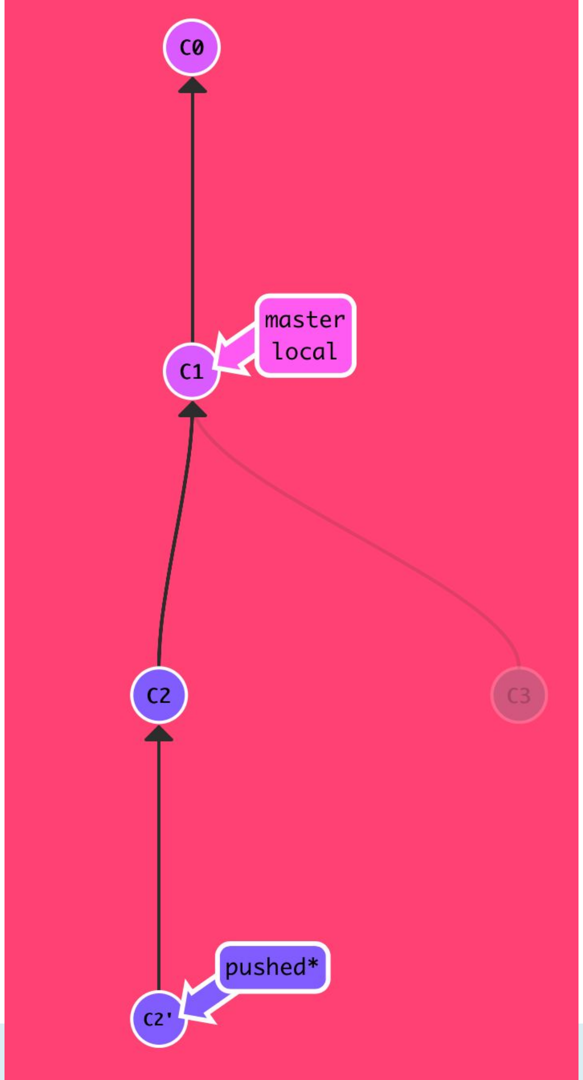
git reset does not work for remote branches





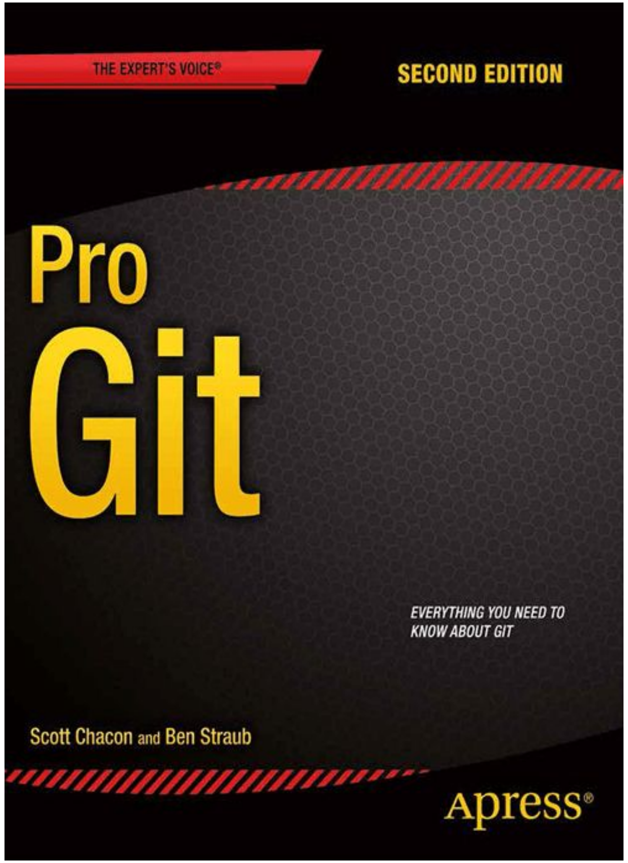
Activity:


C0
master
C1
pushed
C2
C3
local*


C0
master
local
C1
C2
C3
pushed*
C2'

Highly recommended

- (second) most useful life skill you will have learned in 214/514

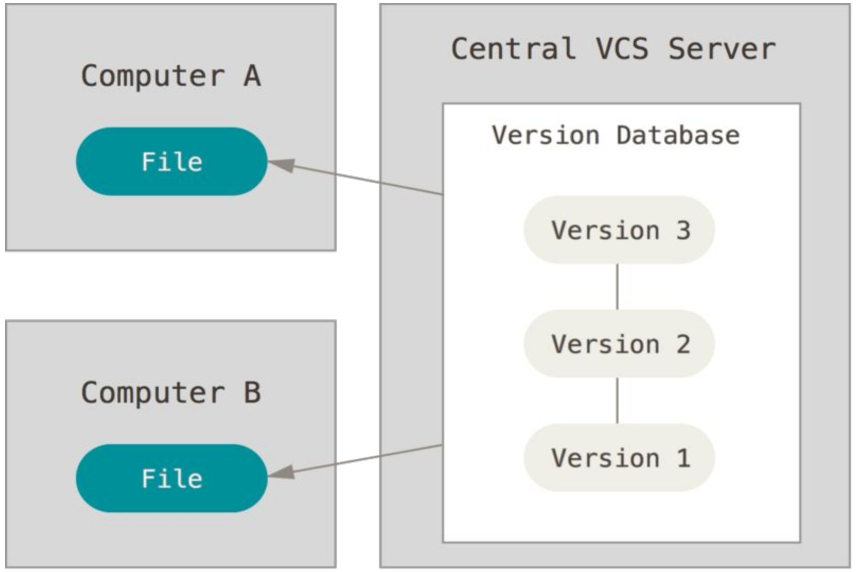

THE EXPERT'S VOICE®
SECOND EDITION
Pro Git
EVERYTHING YOU NEED TO KNOW ABOUT GIT
Scott Chacon and Ben Straub
Apress®

https://git-scm.com/book/en/v2

TYPES OF VERSION CONTROL

Centralized version control

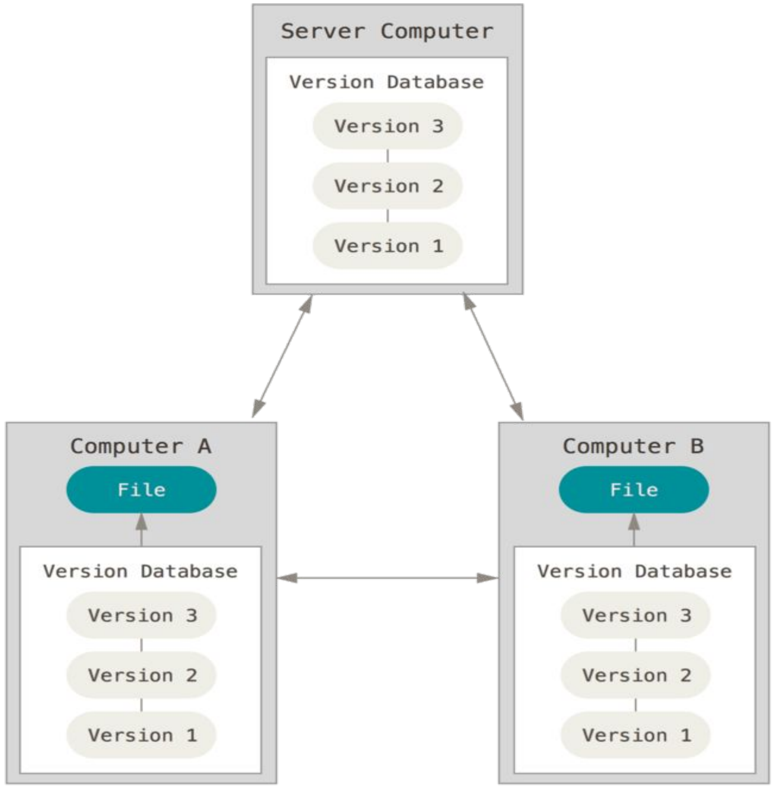
- Single server that contains all the versioned files
- Clients check out/in files from that central place
- E.g., CVS, SVN (Subversion), and Perforce



https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control

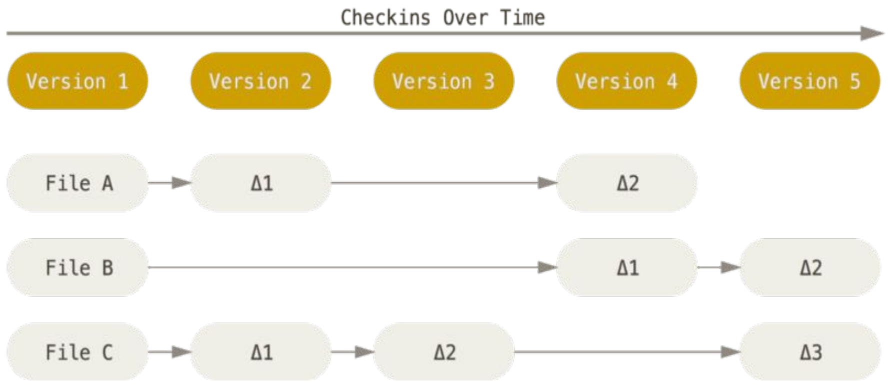
Distributed version control

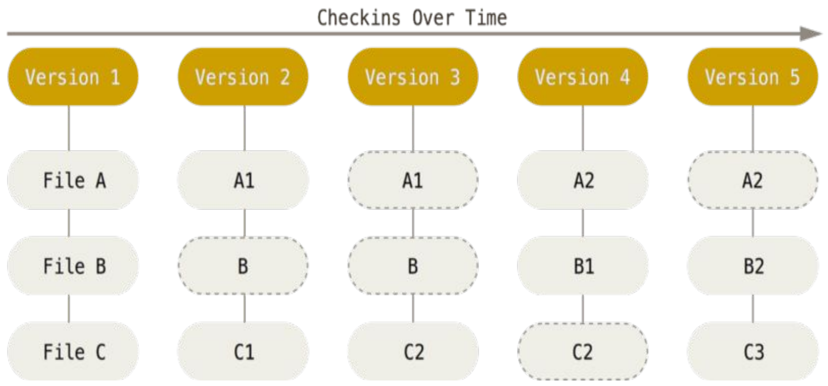
- Clients fully mirror the repository
  - Every clone is a full backup of ***all*** the data
- E.g., Git, Mercurial, Bazaar



https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control

SVN (left) vs. Git (right)



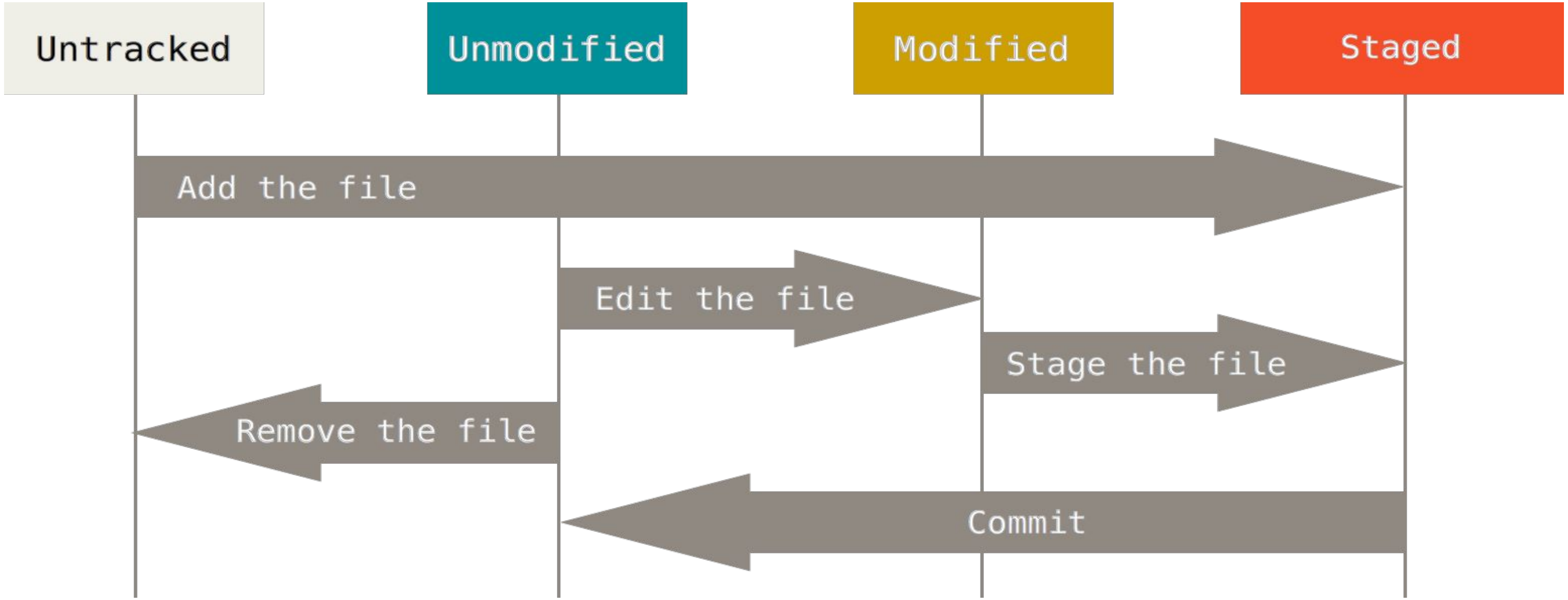


- SVN stores changes to a base version of each file
- Version numbers (1, 2, 3, …) are increased by one after each commit

- Git stores each version as a snapshot
- If files have not changed, only a link to the previous file is stored
- Each version is referred by the SHA-1 hash of the contents

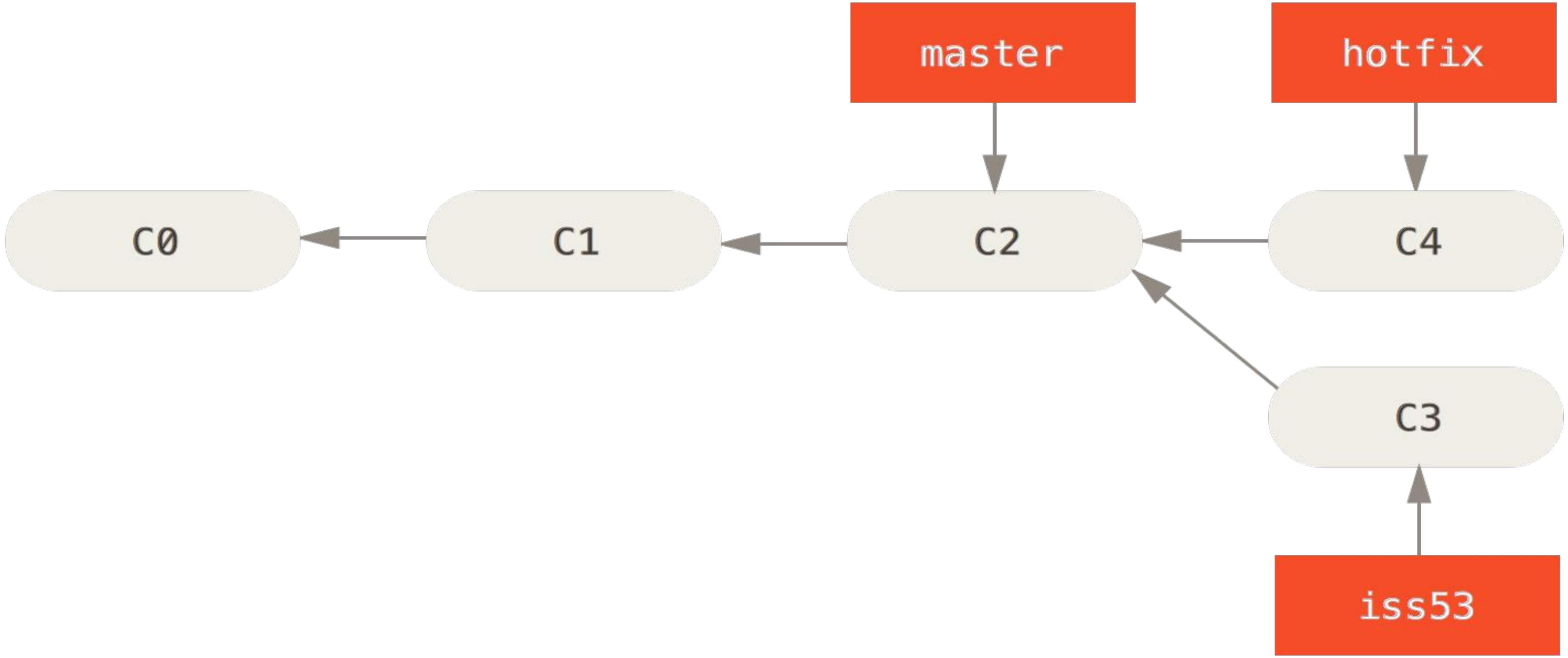
https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control

Aside: Git process



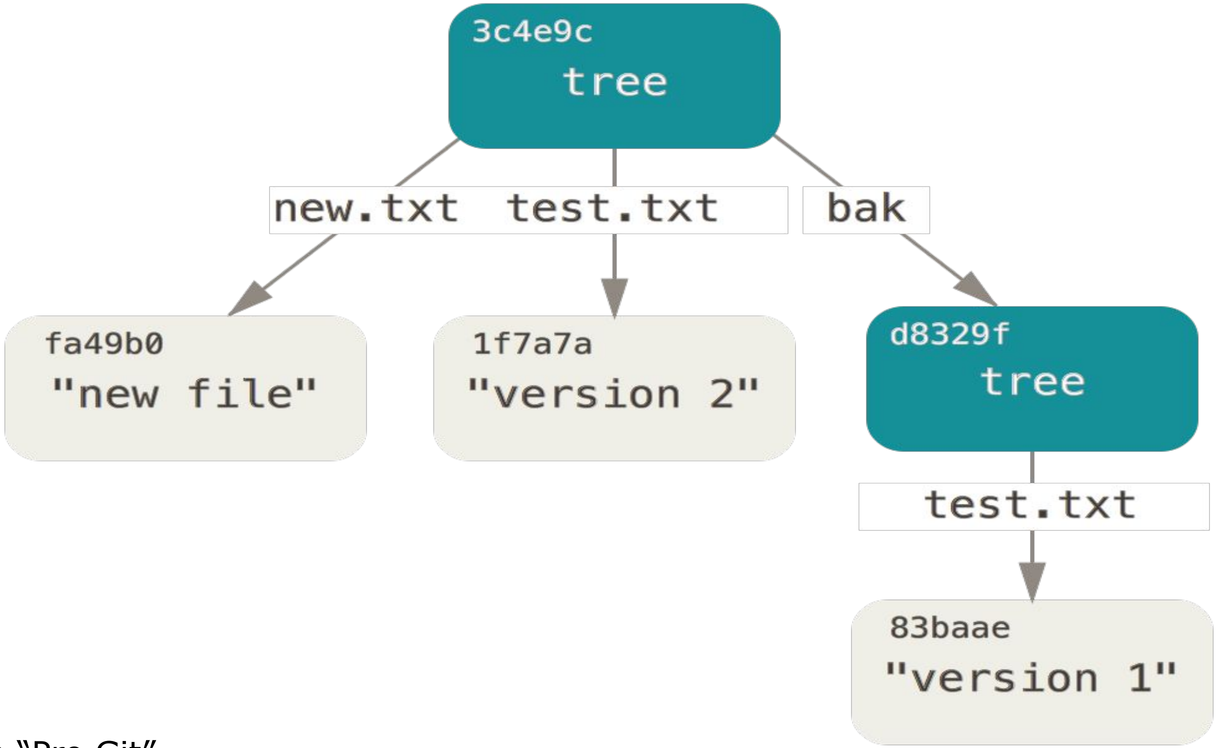
© Scott Chacon "Pro Git"

Git Internals



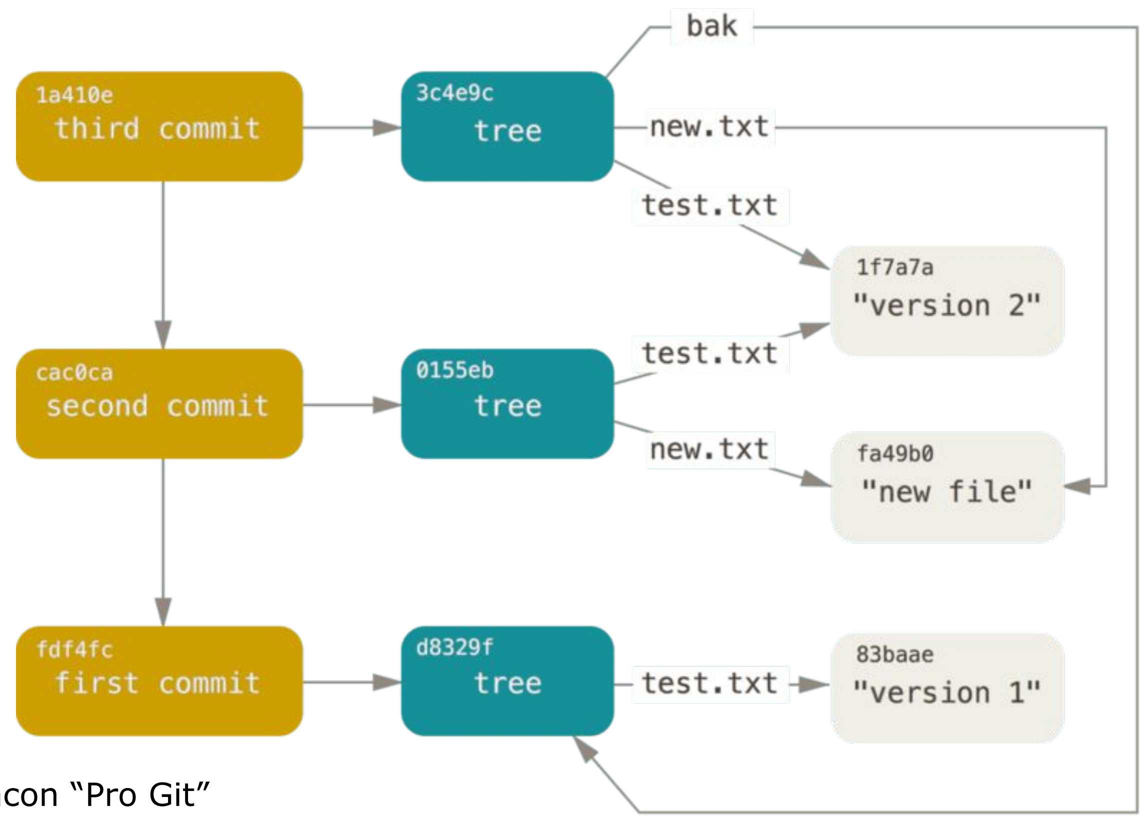
© Scott Chacon "Pro Git"

Git Internals



© Scott Chacon "Pro Git"

Aside: Git object graph



© Scott Chacon "Pro Git"

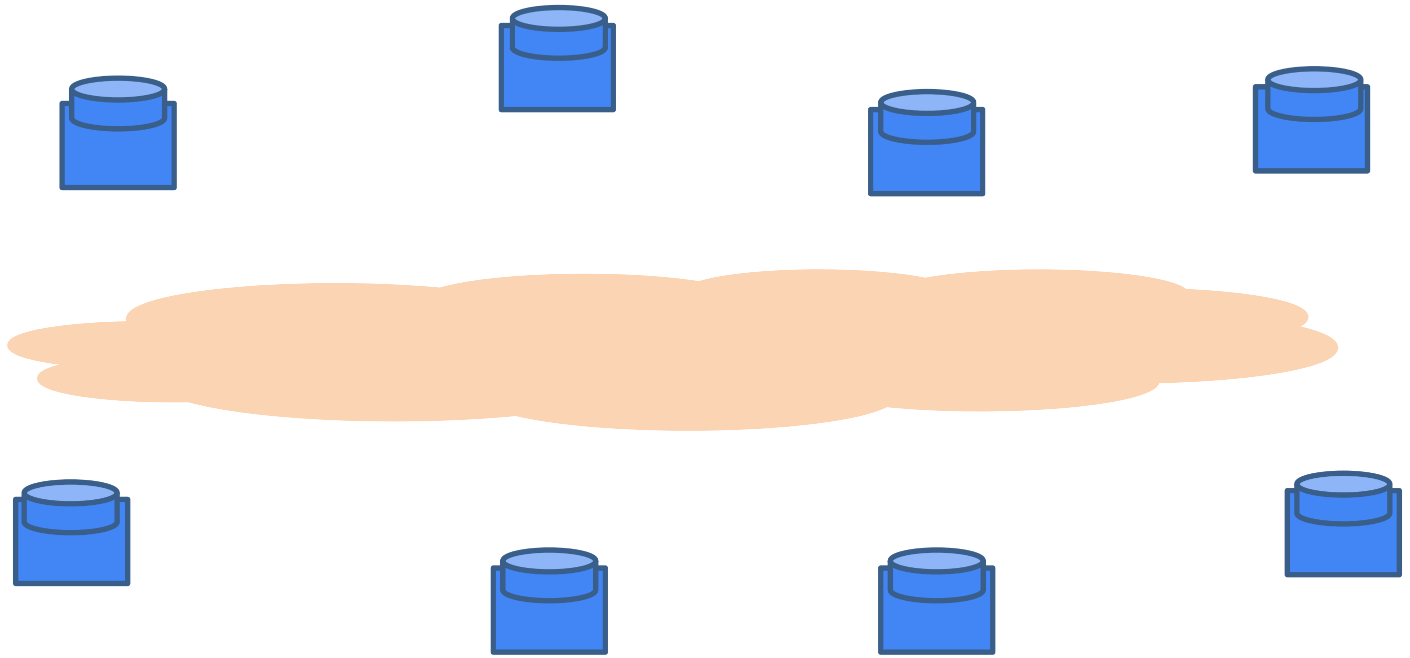
Aside: Which files to manage

- All code and noncode files
  - Java code
  - Build scripts
  - Documentation
- Exclude generated files (.class, …)
- Most version control systems have a mechanism to exclude files (e.g., .gitignore)

SYNCING LOCAL <--> REMOTE

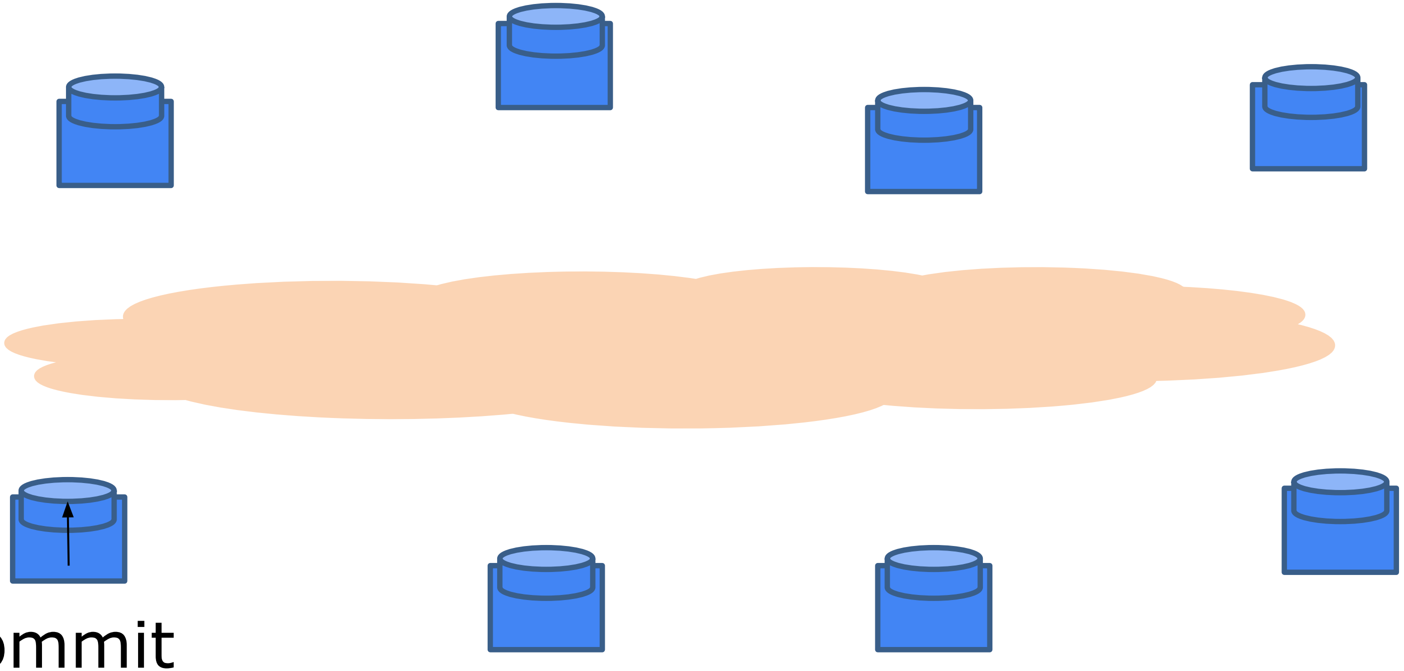
Git

Every computer is a server and version control happens locally.



Git

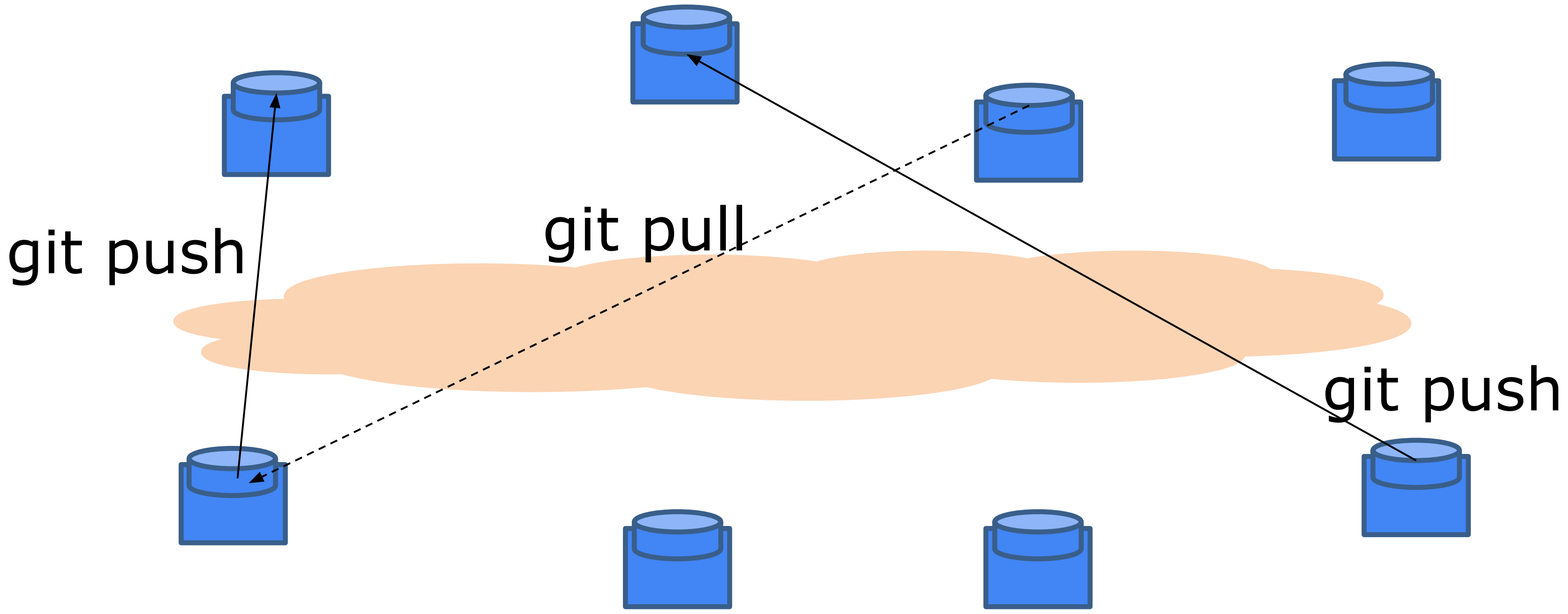
How do you share code with collaborators if commits are *local*?



git commit

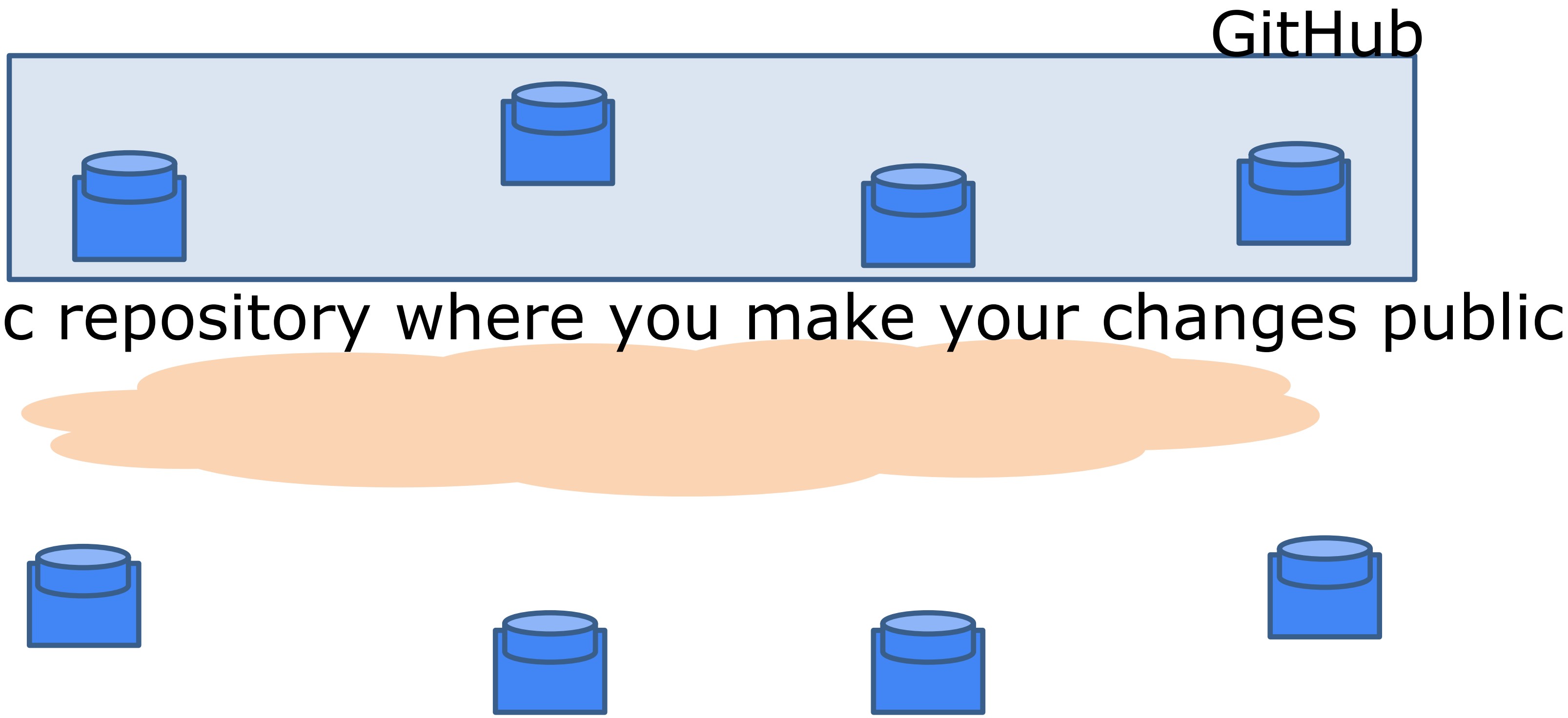
Git

You *push* your commits into their repositories /
They *pull* your commits into their repositories

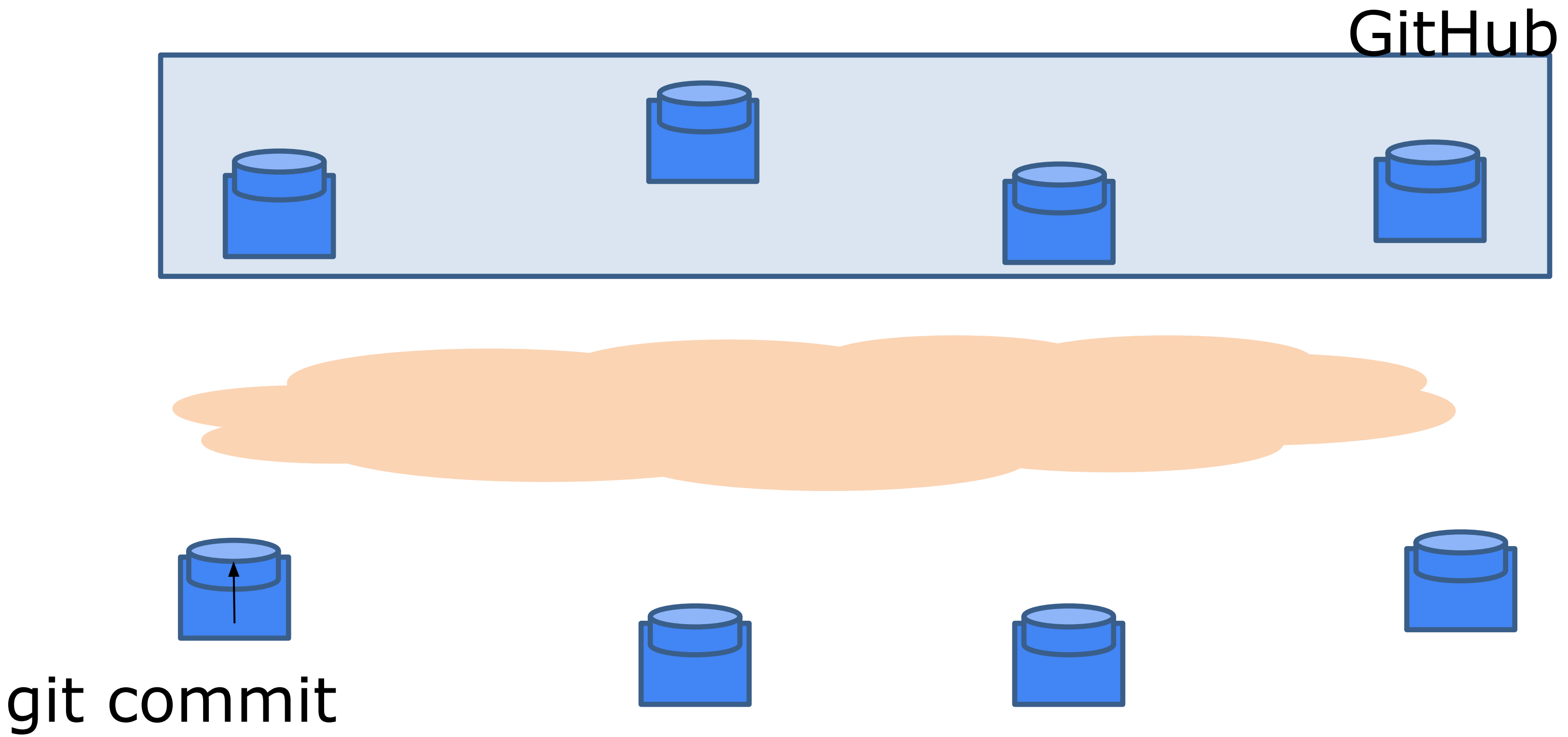


… But requires host names / IP addresses

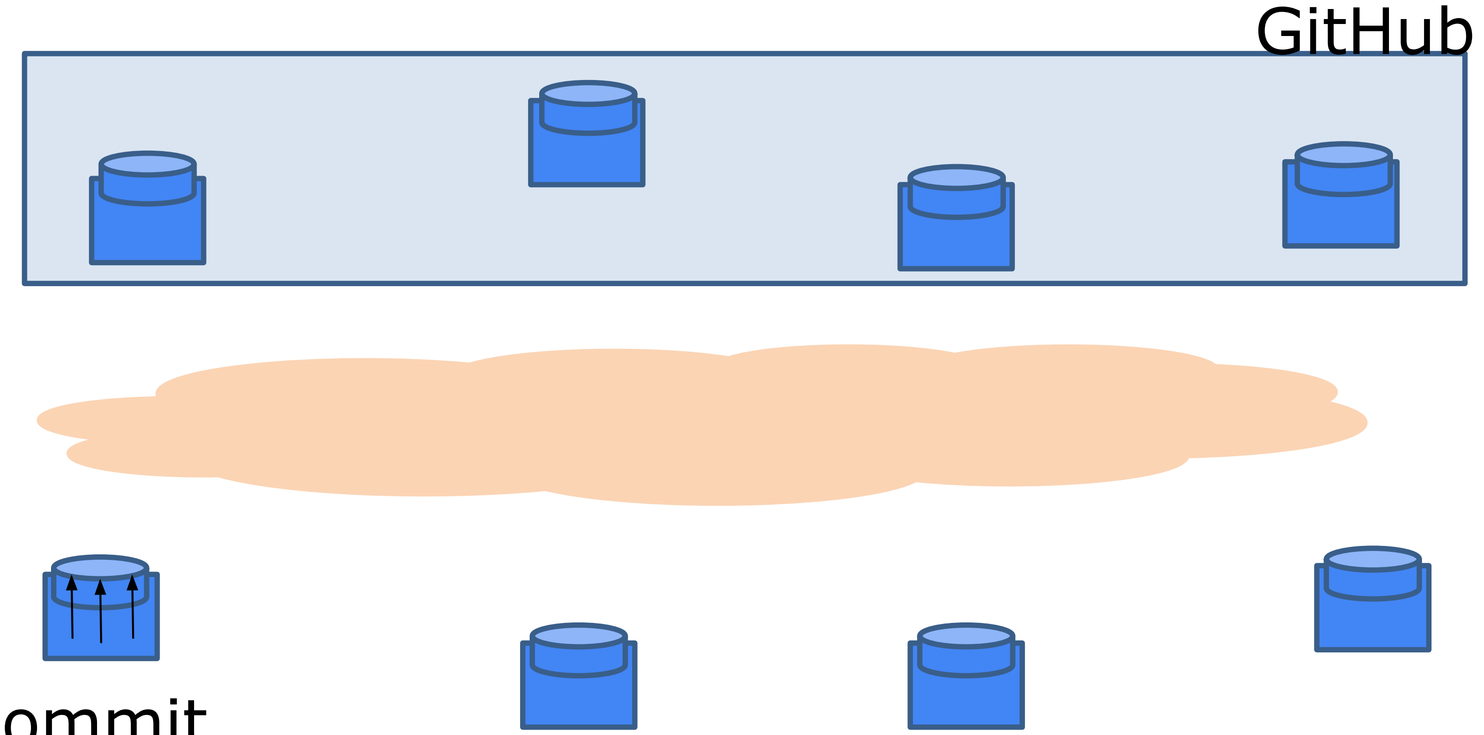
GitHub typical workflow



GitHub typical workflow

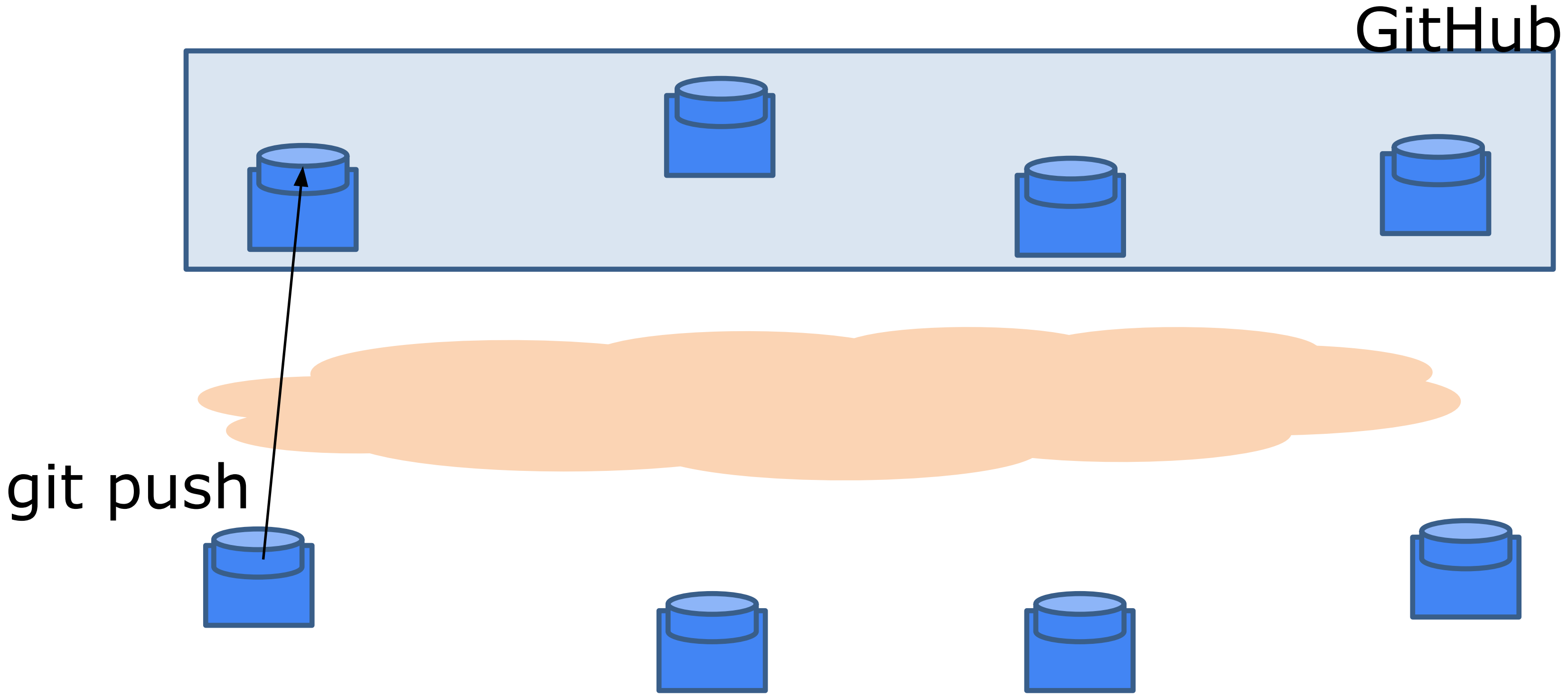


GitHub typical workflow



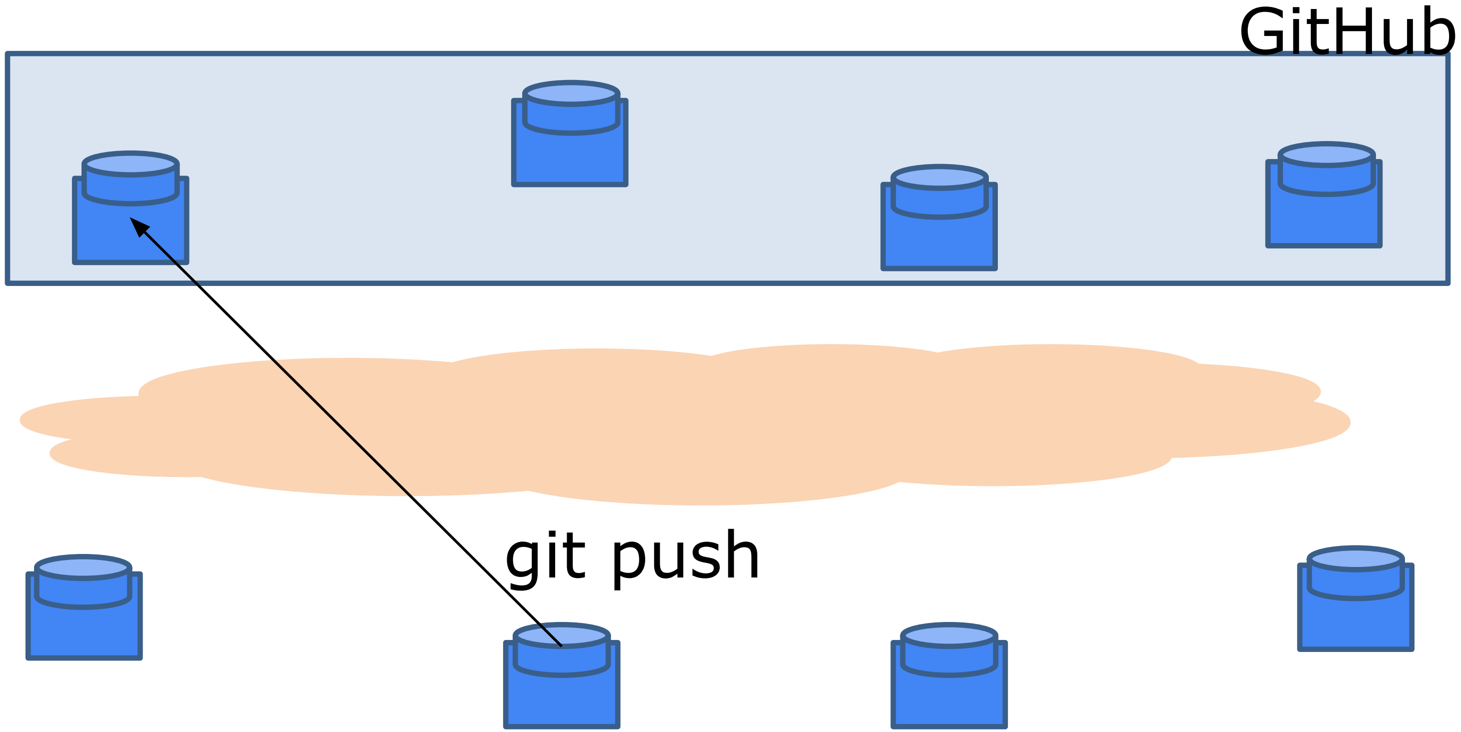
git commit

GitHub typical workflow



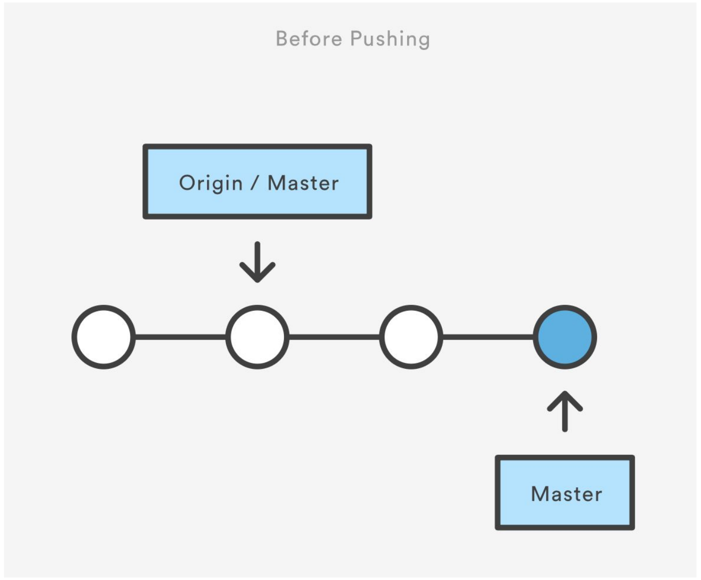
push your local changes into a remote repository.

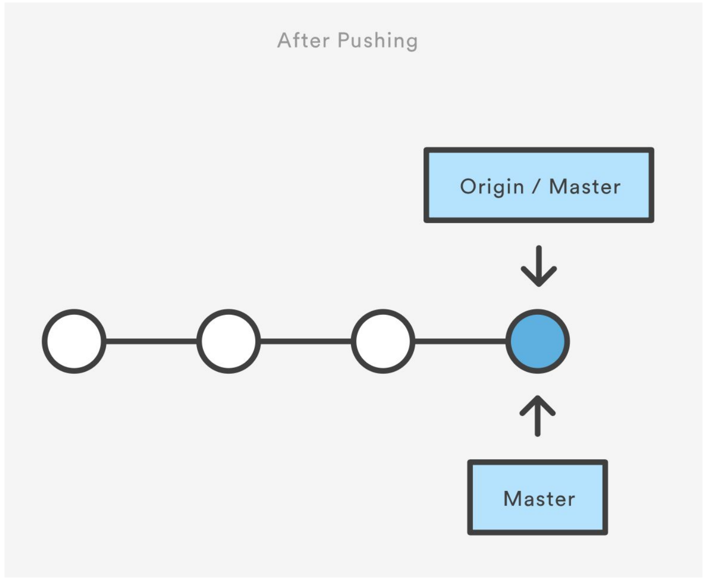
GitHub typical workflow



Collaborators can push too if they have access rights.

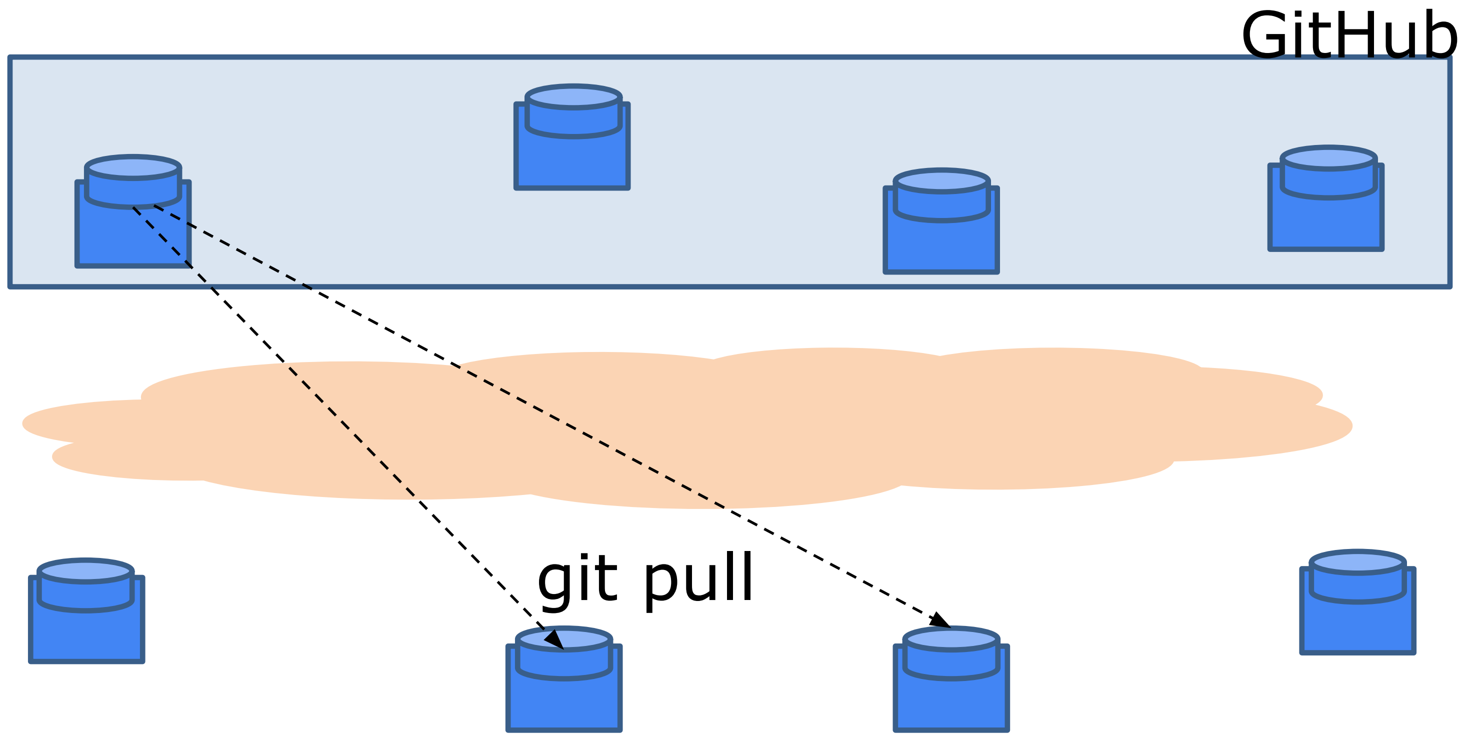
`git push <remote> <branch>`: upload local repository content to a remote repository





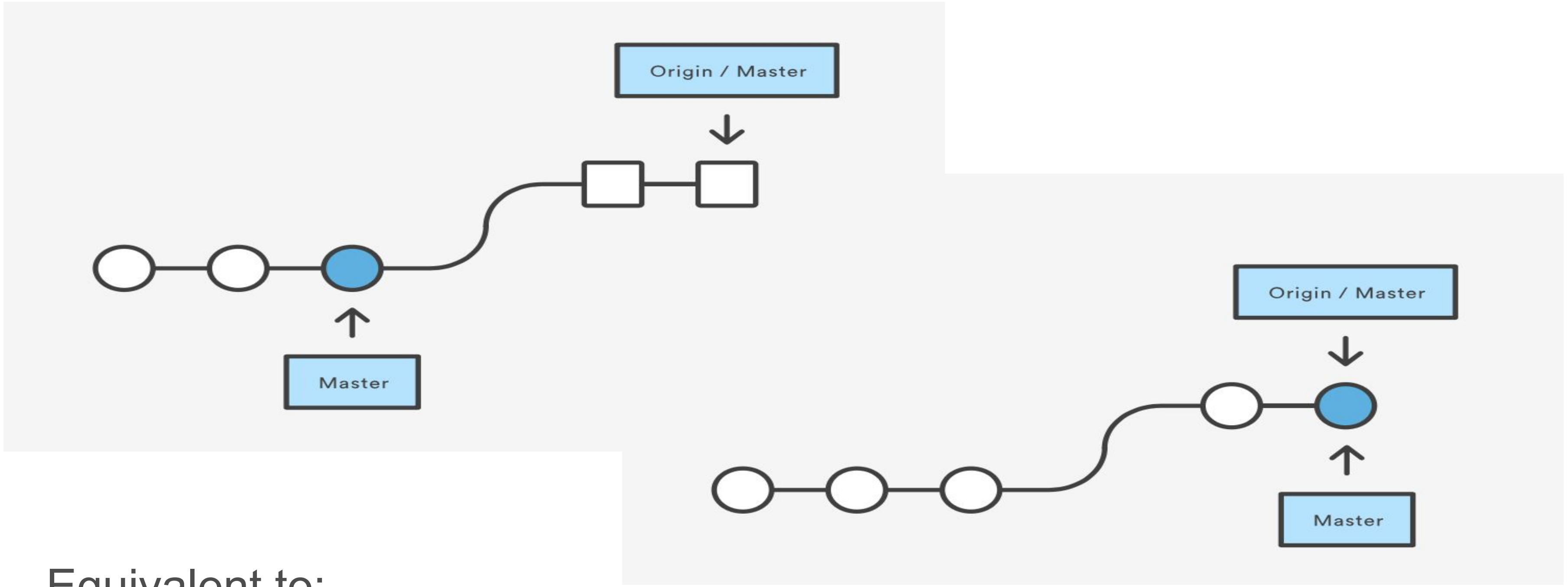
https://www.atlassian.com/git/tutorials/syncing/git-push

GitHub typical workflow



Without access rights, “don’t call us, we’ll call you” (*pull* from trusted sources) … But again requires host names / IP addresses.

`git pull <remote>`: Fetch the specified remote's copy of the current branch and immediately merge it into the local copy

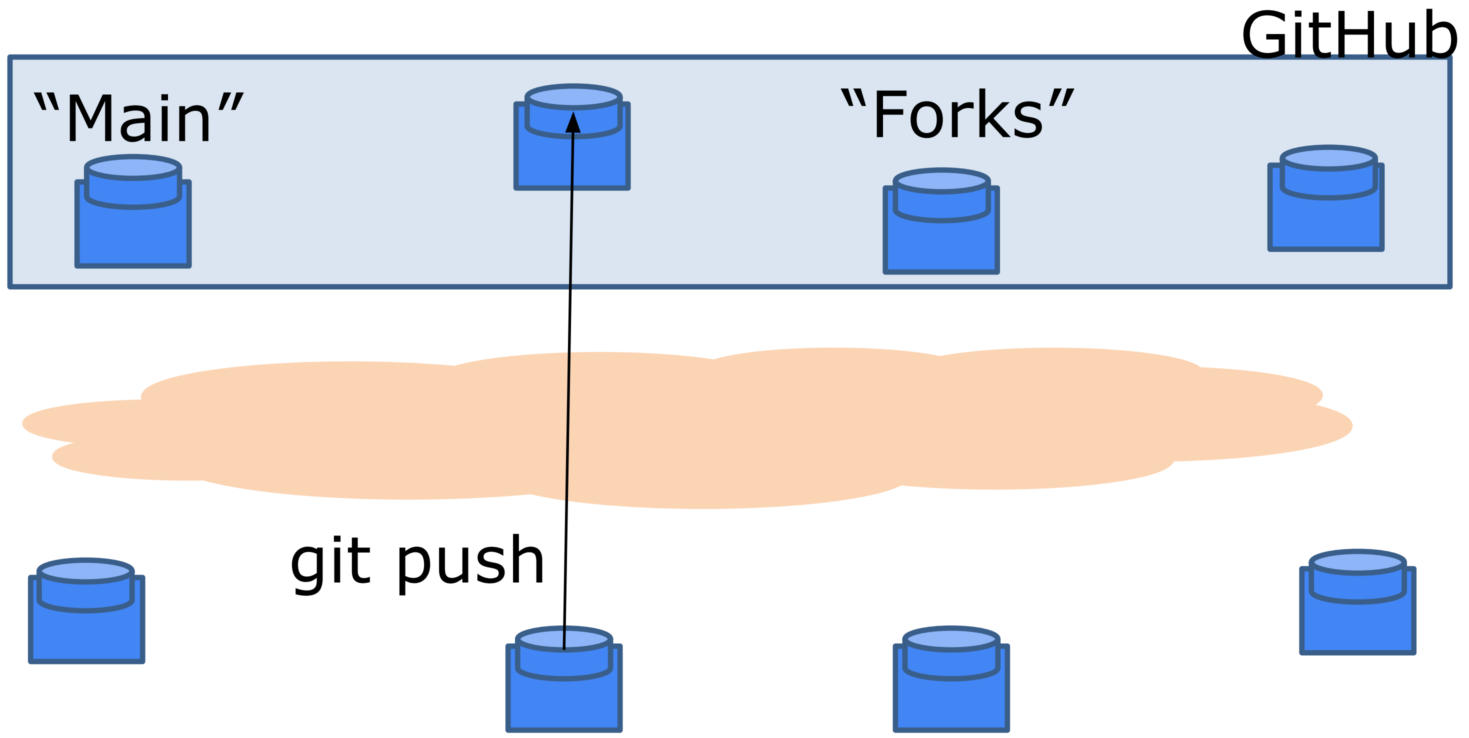


Equivalent to:

`git fetch origin HEAD` + `git merge HEAD`

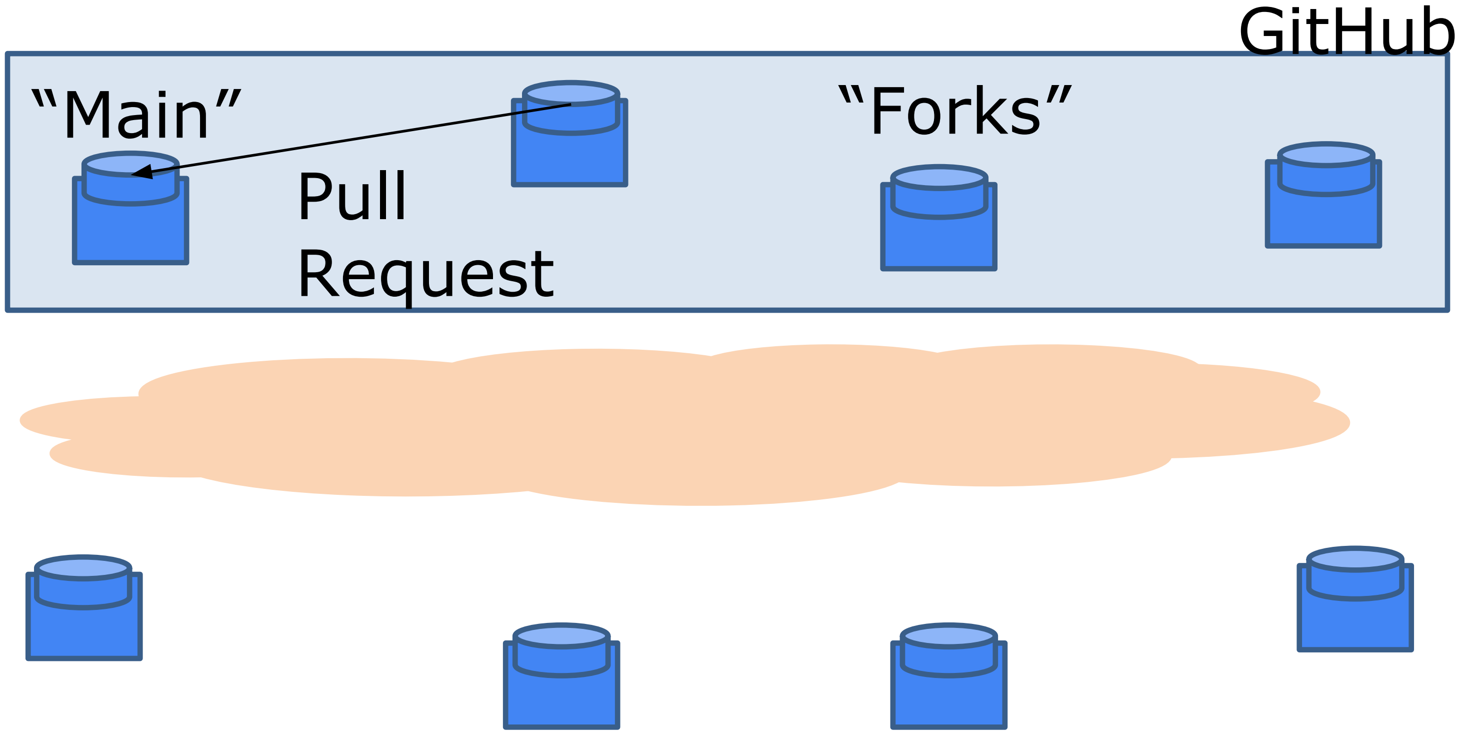
Also possible: `git pull --rebase origin`

GitHub typical workflow



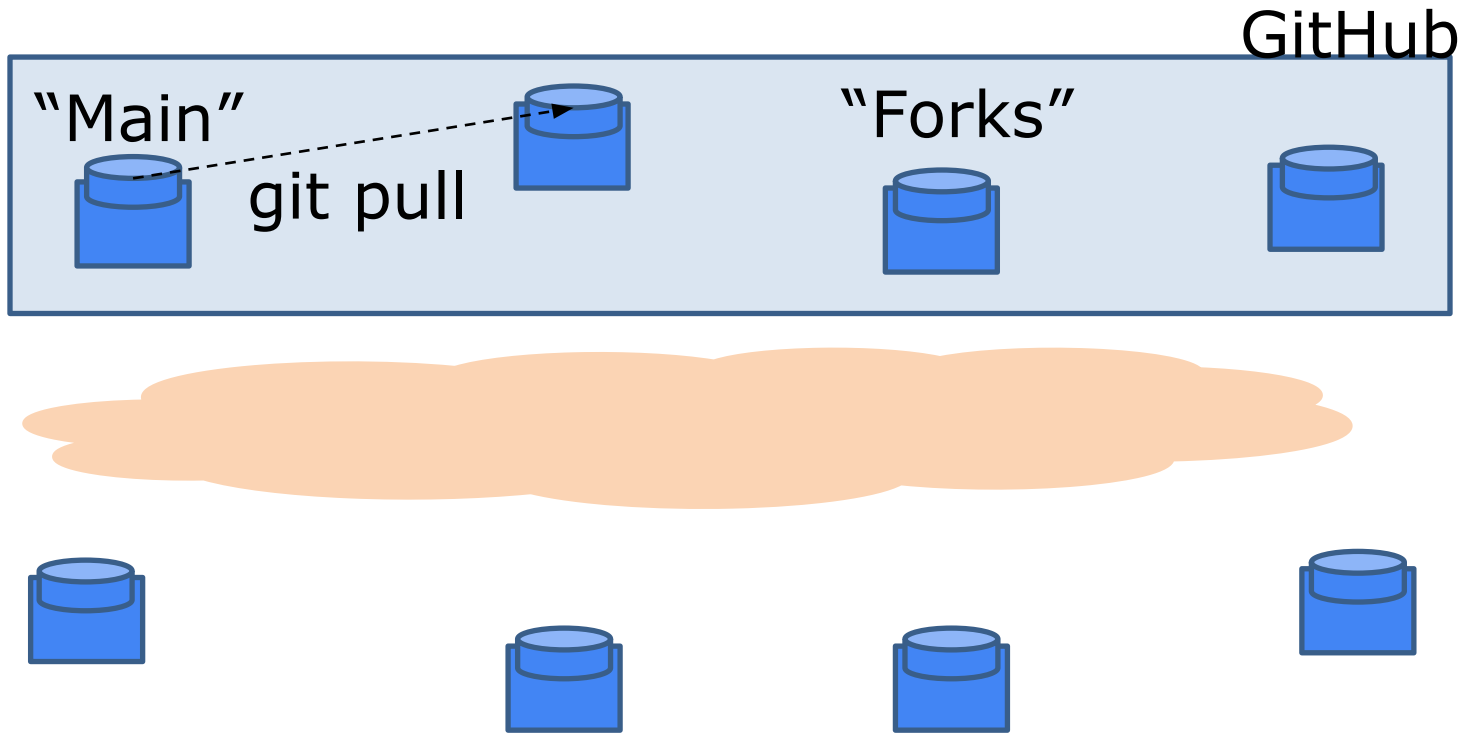
Instead, people maintain public remote "forks" of "main" repository on GitHub and push local changes.

GitHub typical workflow



Availability of new changes is signaled via "Pull Request".

GitHub typical workflow



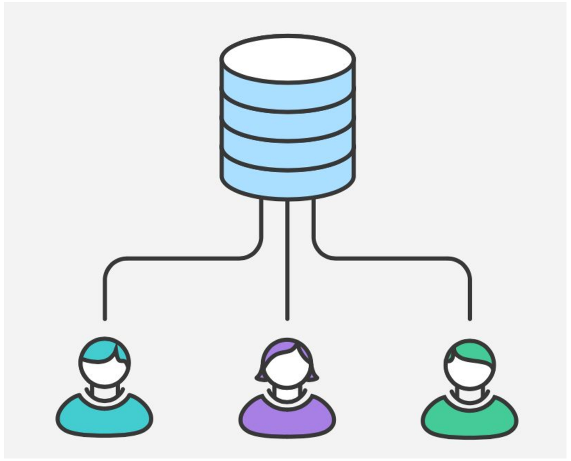
Changes are pulled into main if PR accepted.

BRANCH WORKFLOWS

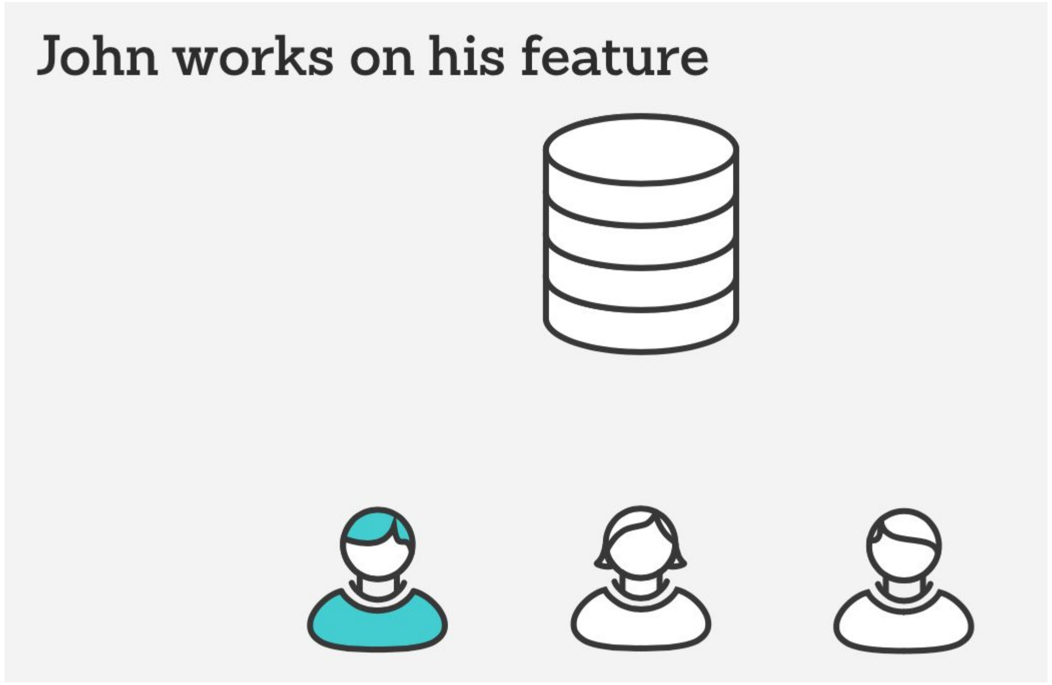
https://www.atlassian.com/git/tutorials/comparing-workflows

1. Centralized workflow

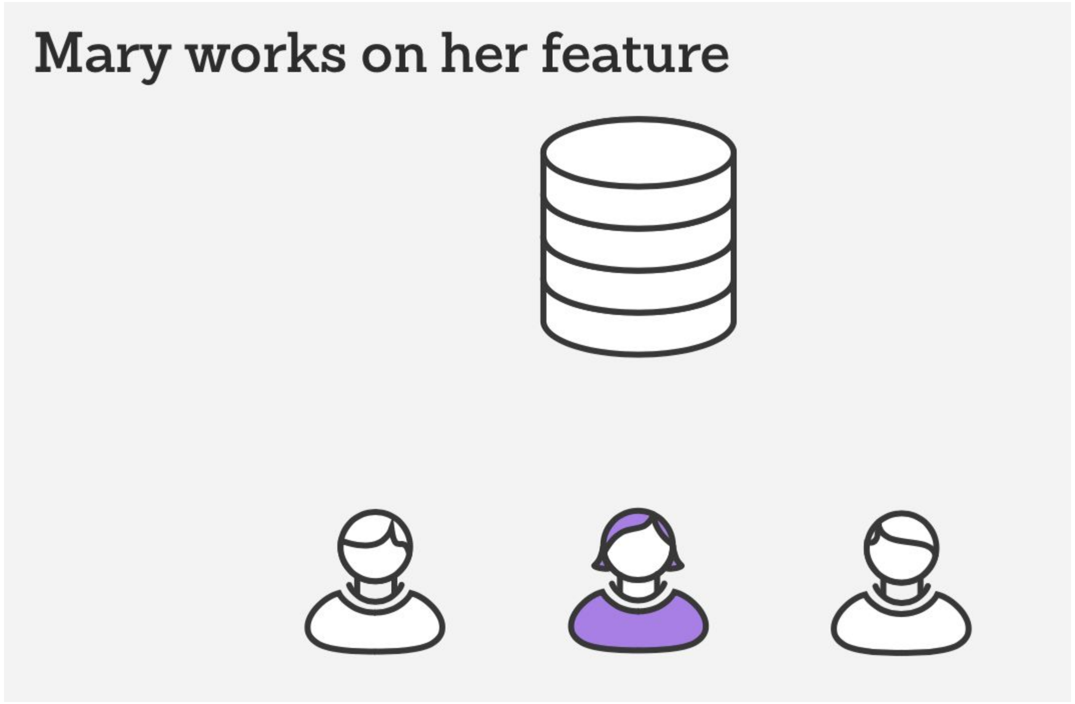
- Central repository to serve as the single point-of-entry for all changes to the project
- Default development branch is called **main**
    - all changes are committed into **main**
    - doesn't require any other branches



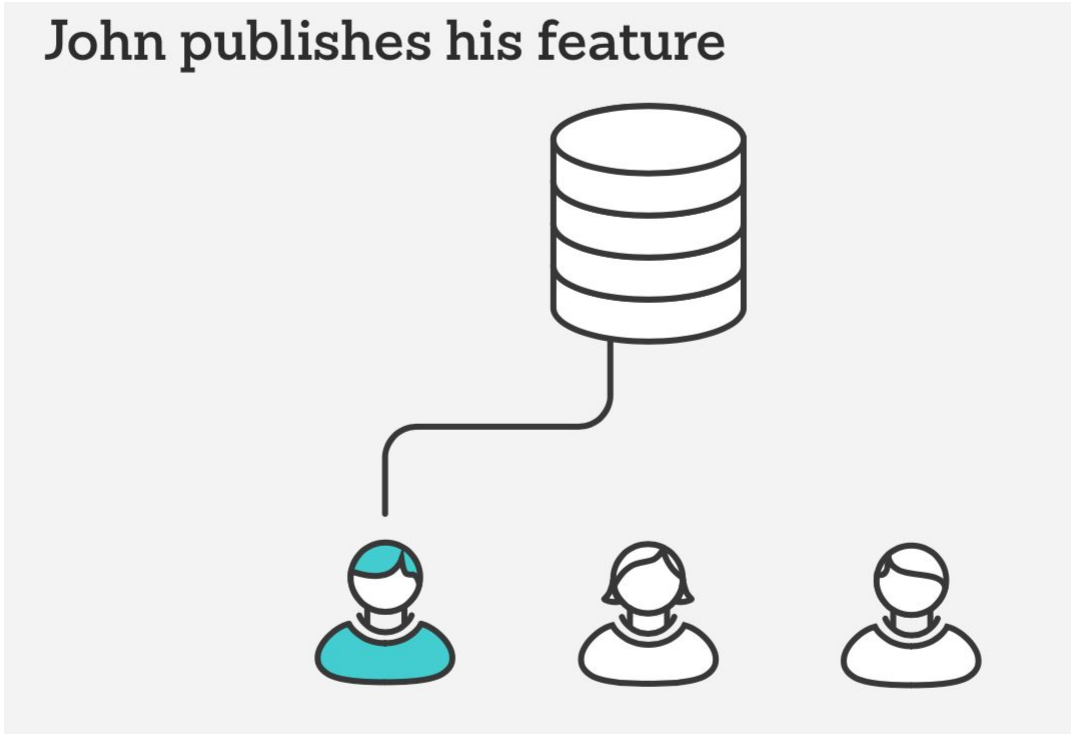
Example



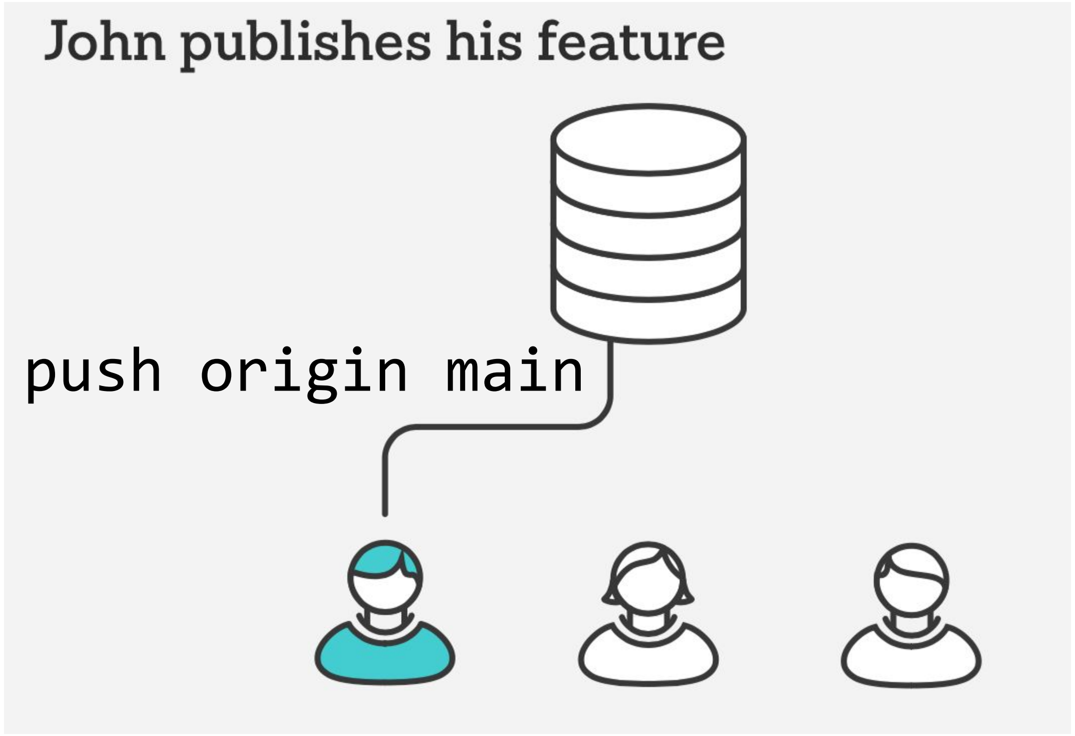
Example



Example

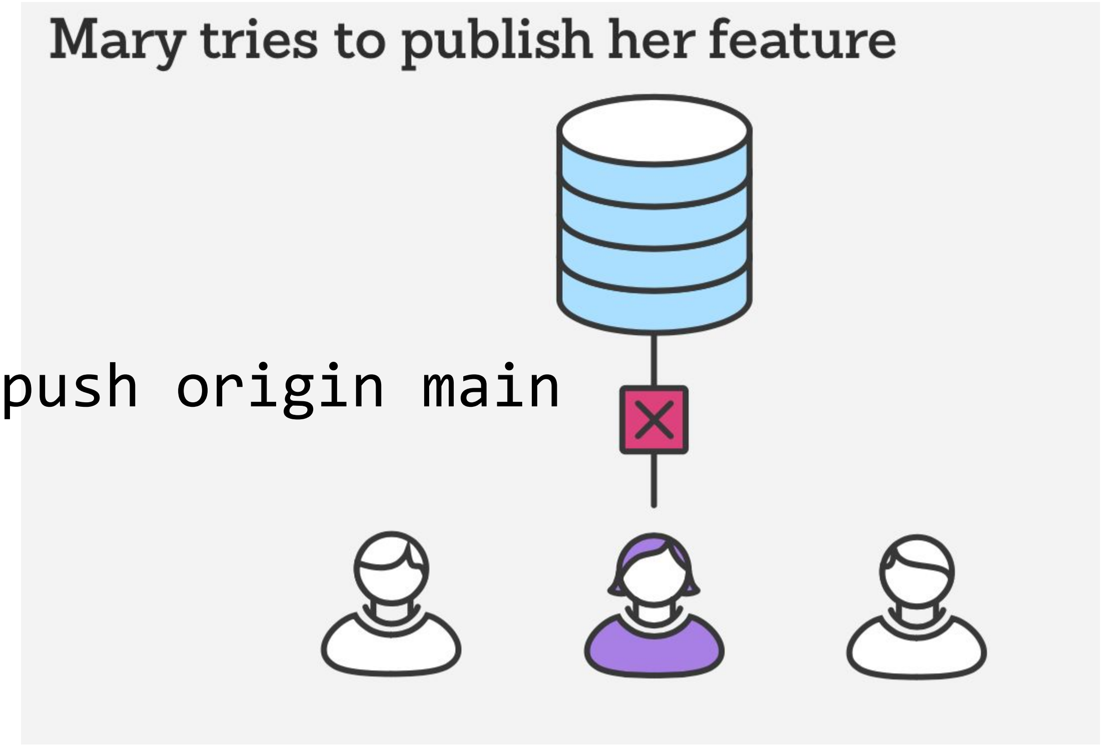


Example



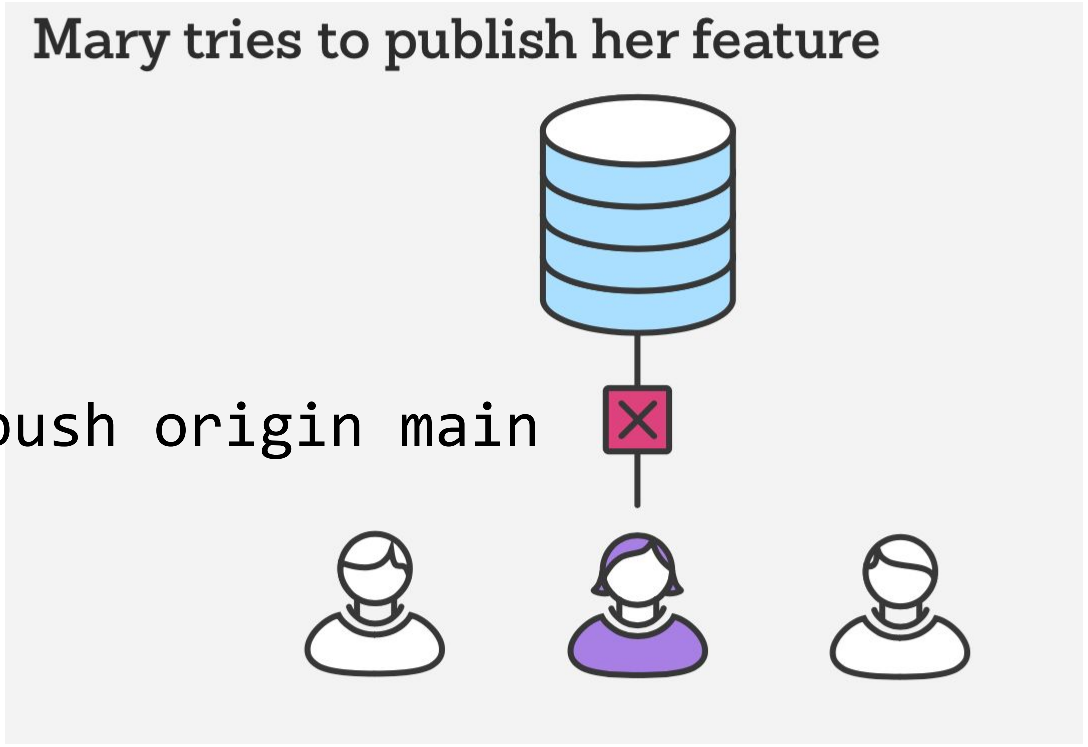
```
git push origin main
```

Example



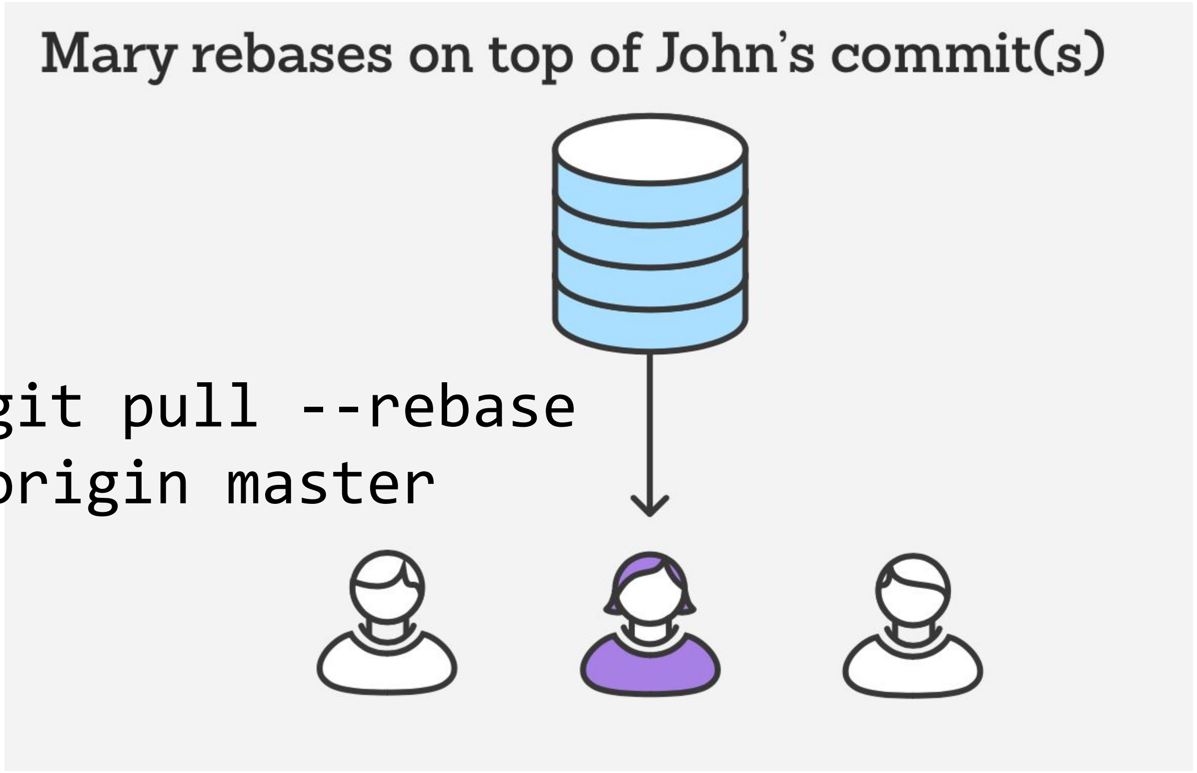
```
git push origin main
```

```
error: failed to push some refs to '/path/to/repo.git’
hint: Updates were rejected because the tip of your current branch is behind its
remote counterpart. Merge the remote changes (e.g. 'git pull') before pushing again.
See the 'Note about fast-forwards' in 'git push --help' for details.
```

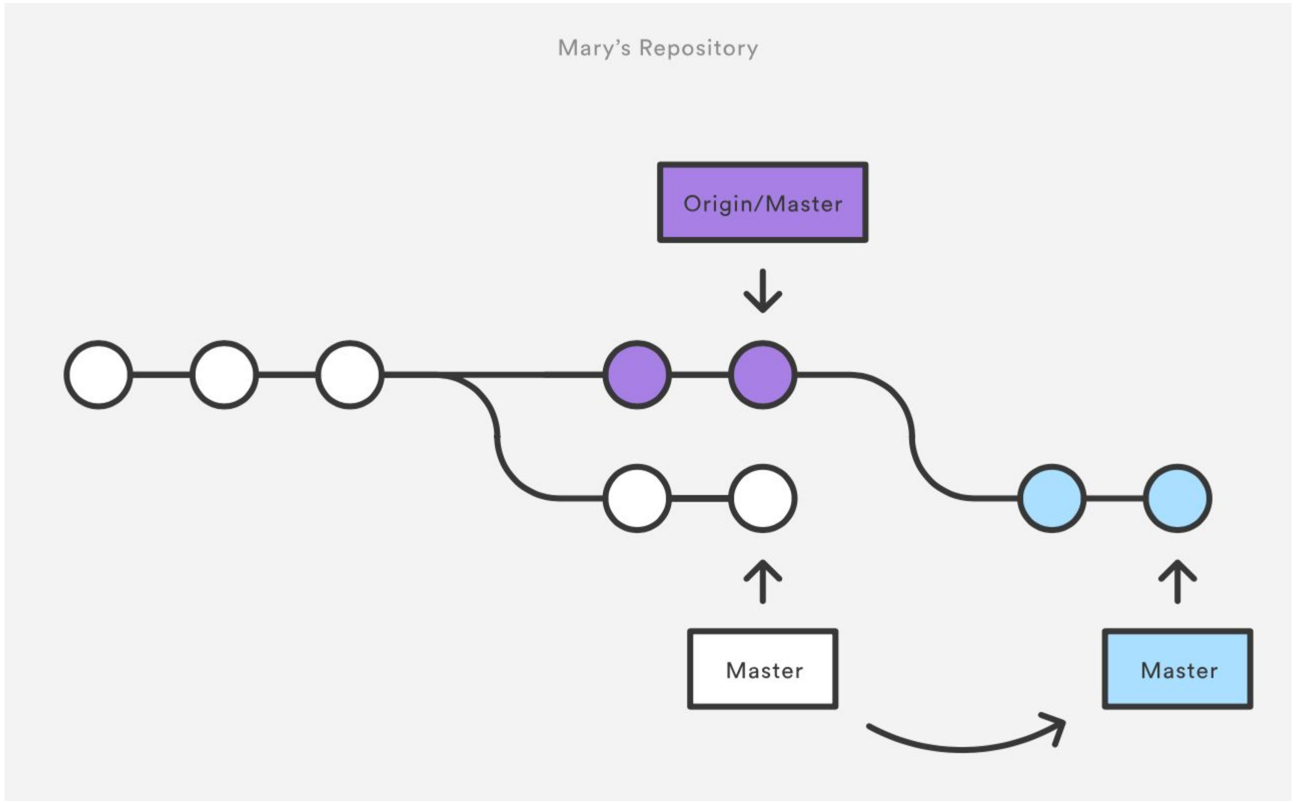


```
git push origin main
```

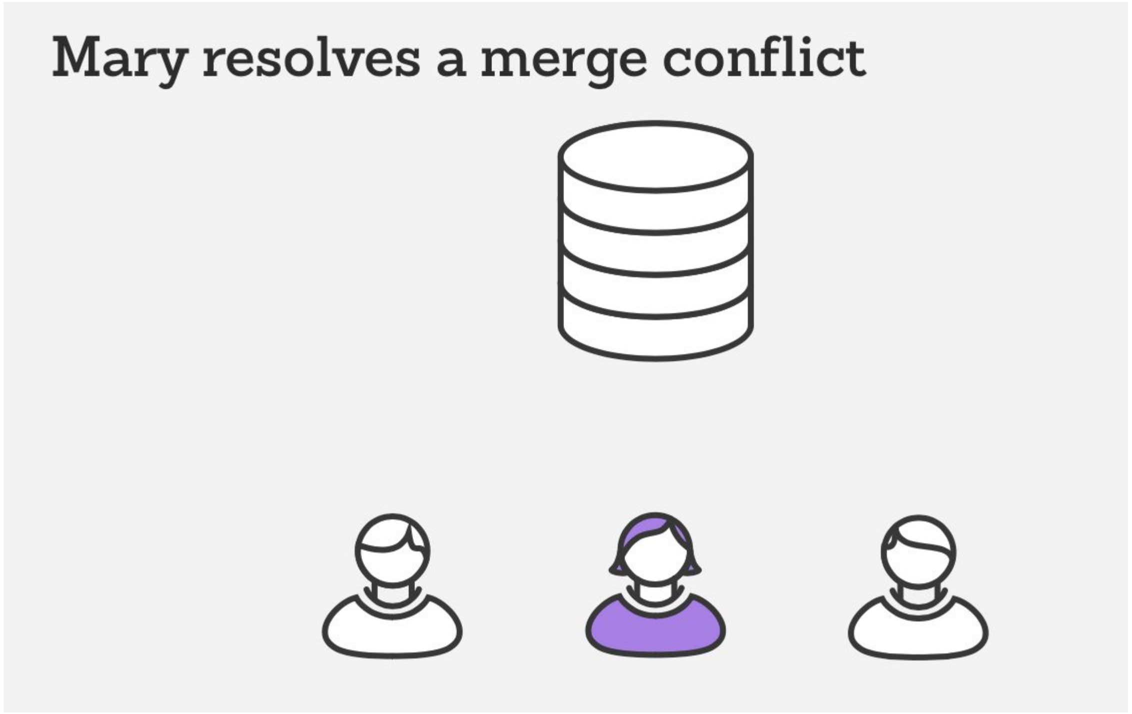
Example



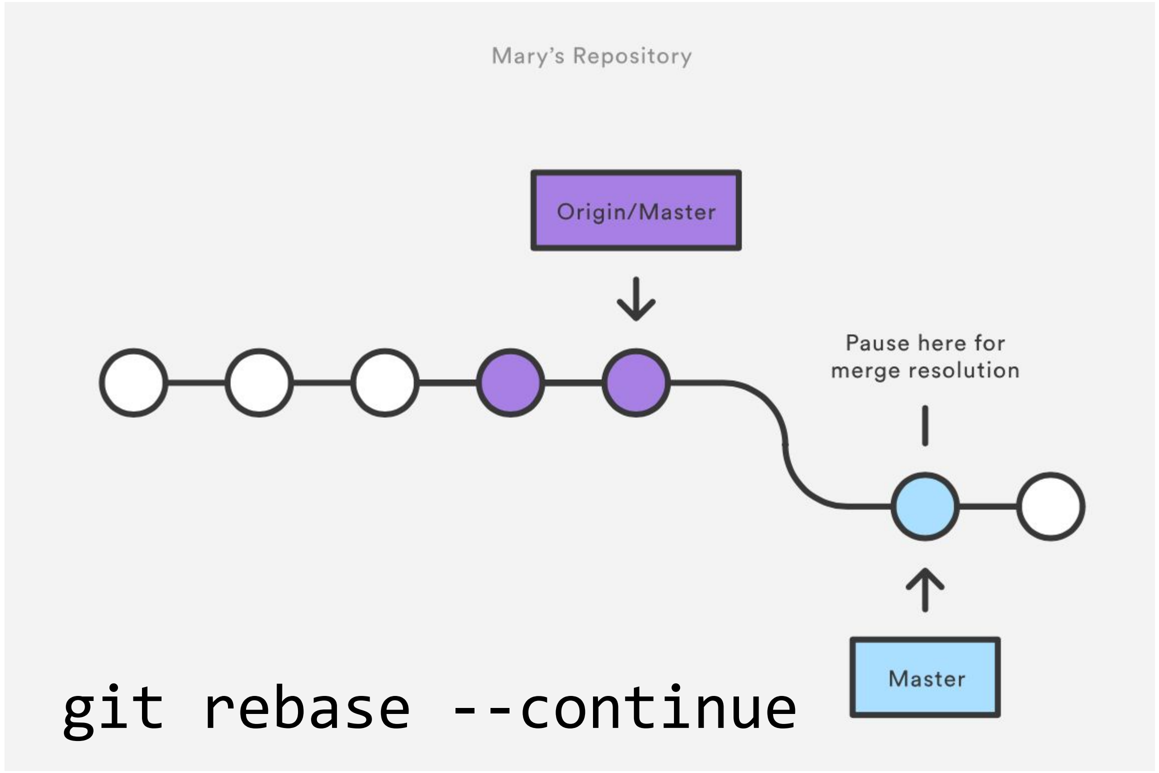
```
git pull --rebase
origin master
```


Mary's Repository
Origin/Master
Master
Master

Example

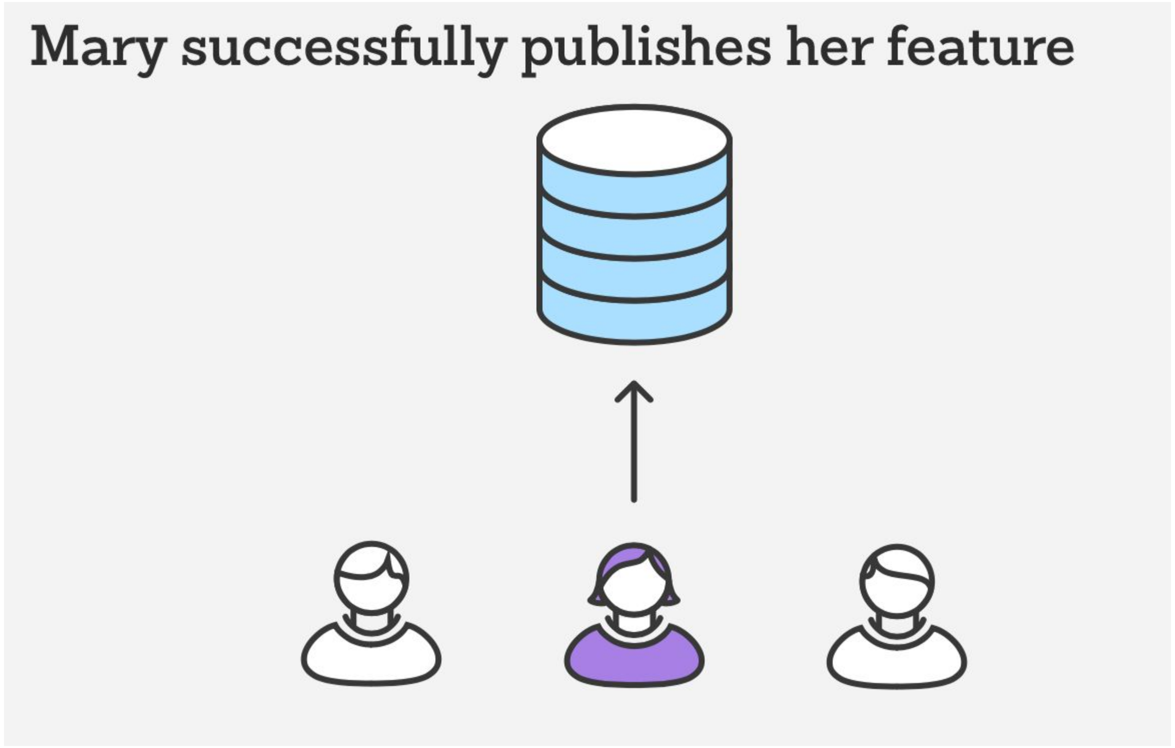


Example



```
git rebase --continue
```

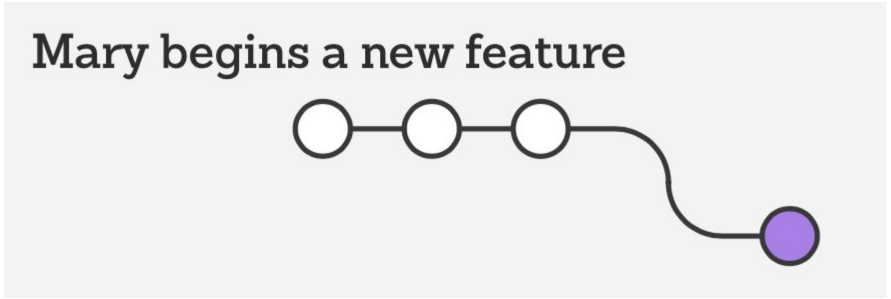
Example



2. Git Feature Branch Workflow

- *All* feature development should take place in a dedicated branch instead of the main branch
- Multiple developers can work on a particular feature without disturbing the main codebase
  - main branch will never contain broken code (enables CI)
  - Enables pull requests (code review)

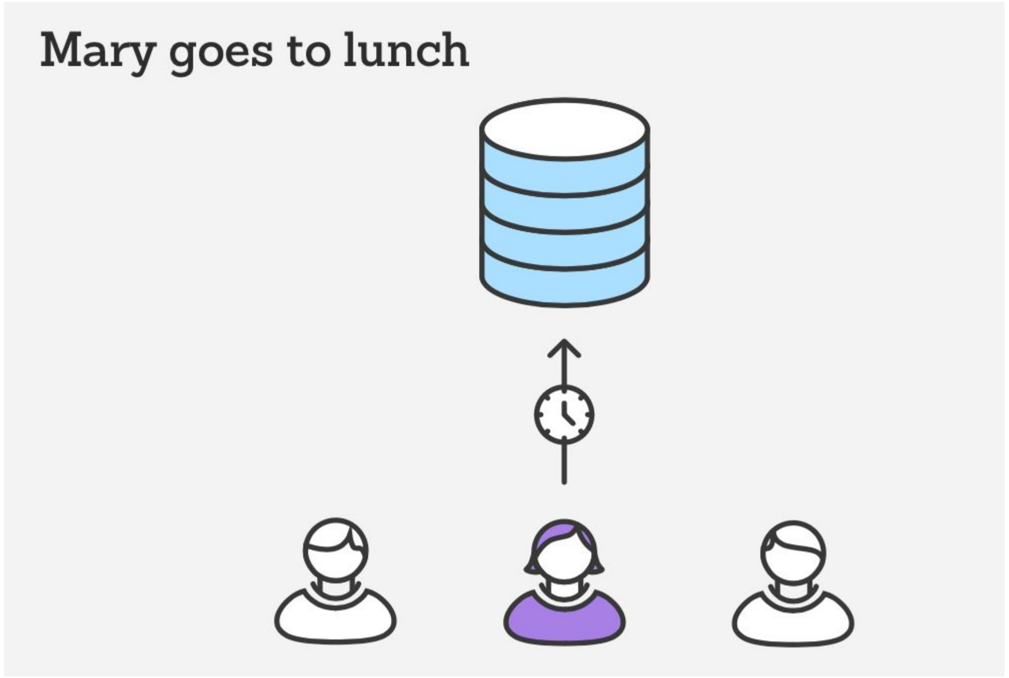
Example



```
git checkout -b marys-feature master
```

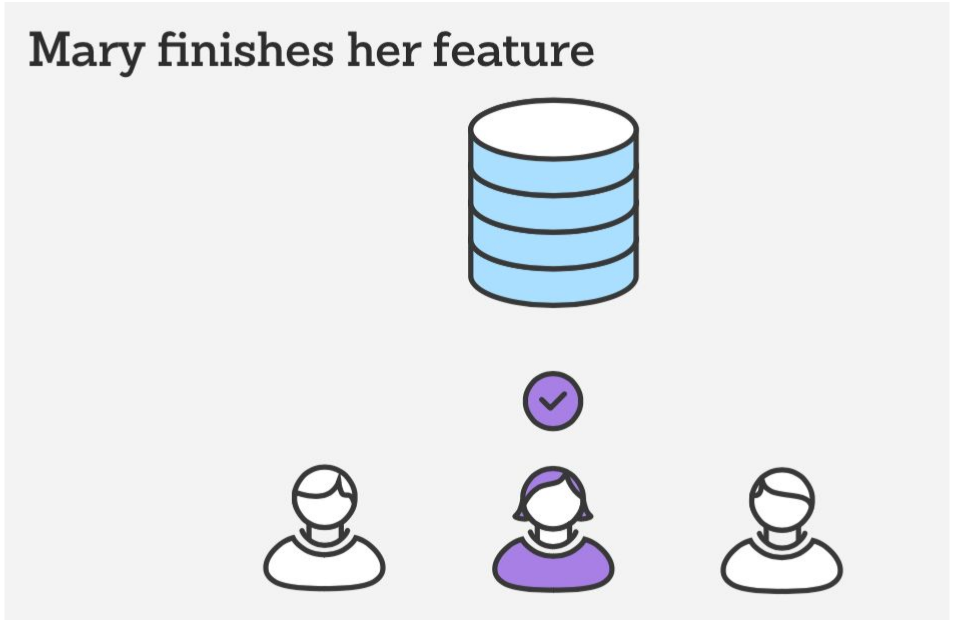
```
git status
git add <some-file>
git commit
```

Example



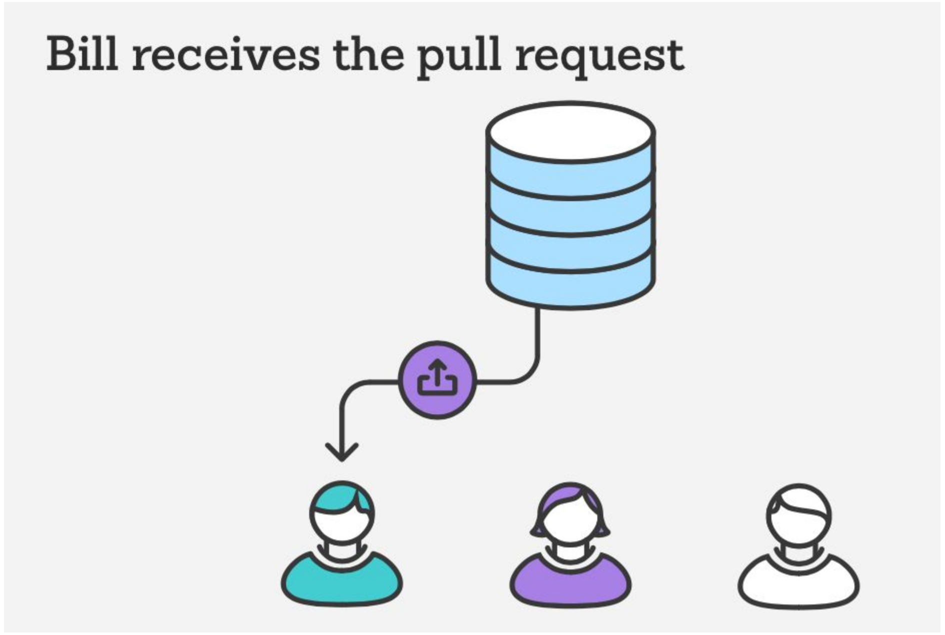
```
git push -u origin marys-feature
```

Example

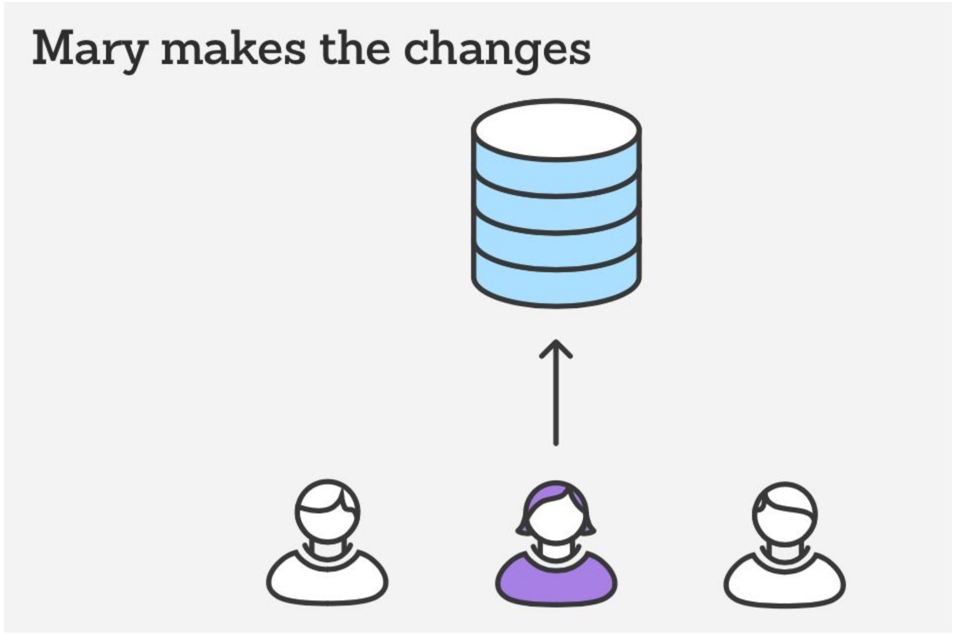


```
git push
```

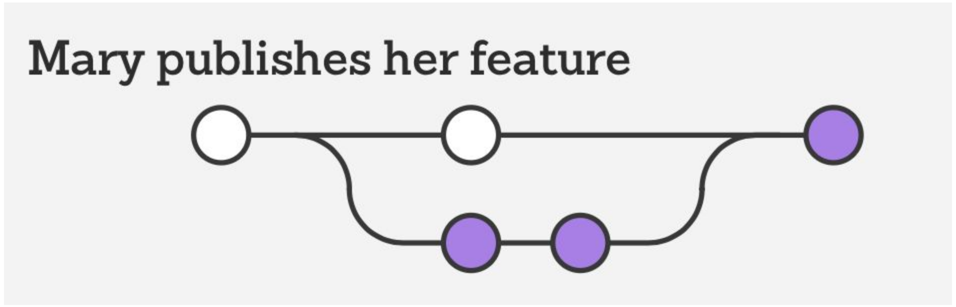
Example



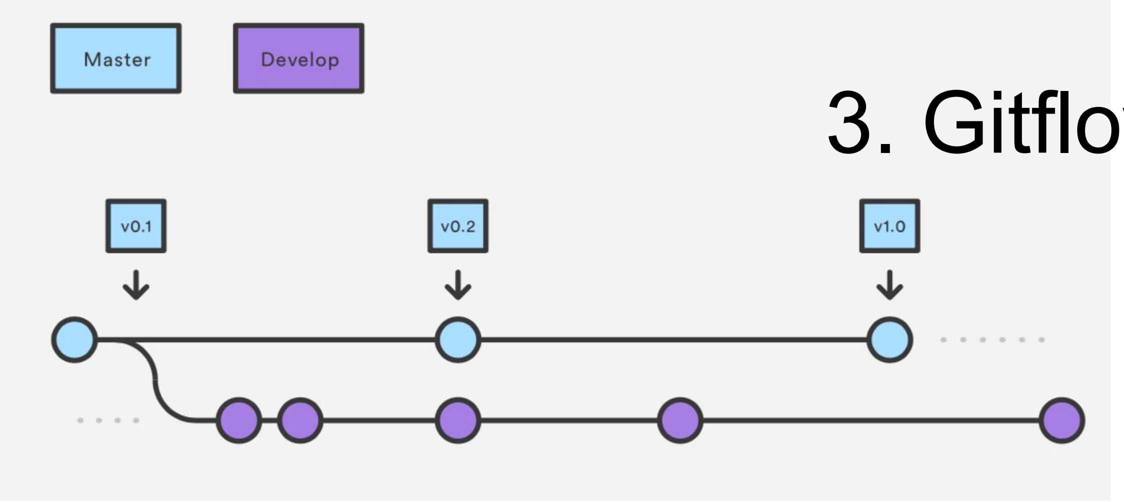
Example



Example - Merge pull request



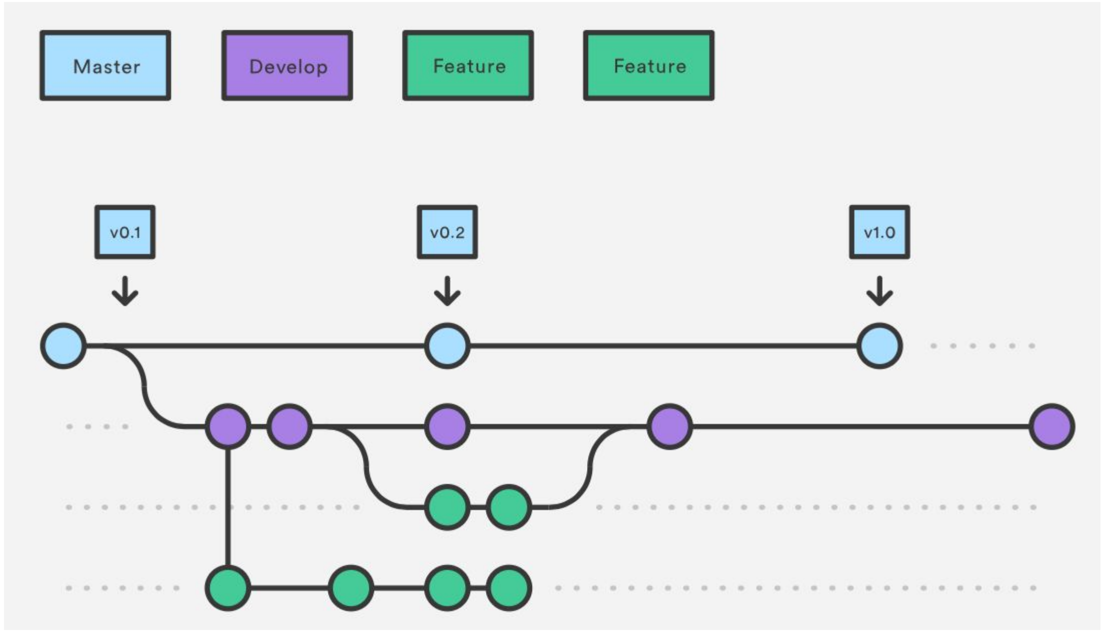
```
git checkout master
git pull
git pull origin marys-feature
git push
```



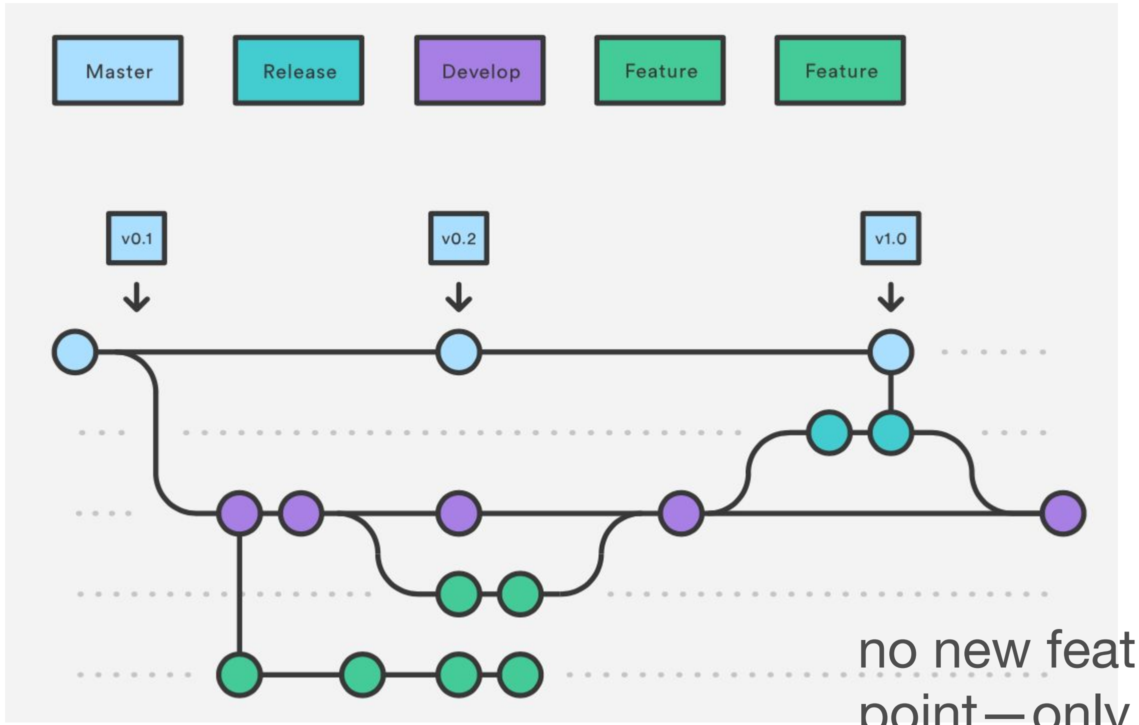
3. Gitflow Workflow

- Strict branching model designed around the project release
  - Suitable for projects that have a scheduled release cycle
- Branches have specific roles and interactions
- Uses two branches
  - master stores the official release history; tag all commits in the master branch with a version number
  - develop serves as an integration branch for features

GitFlow feature branches (from develop)



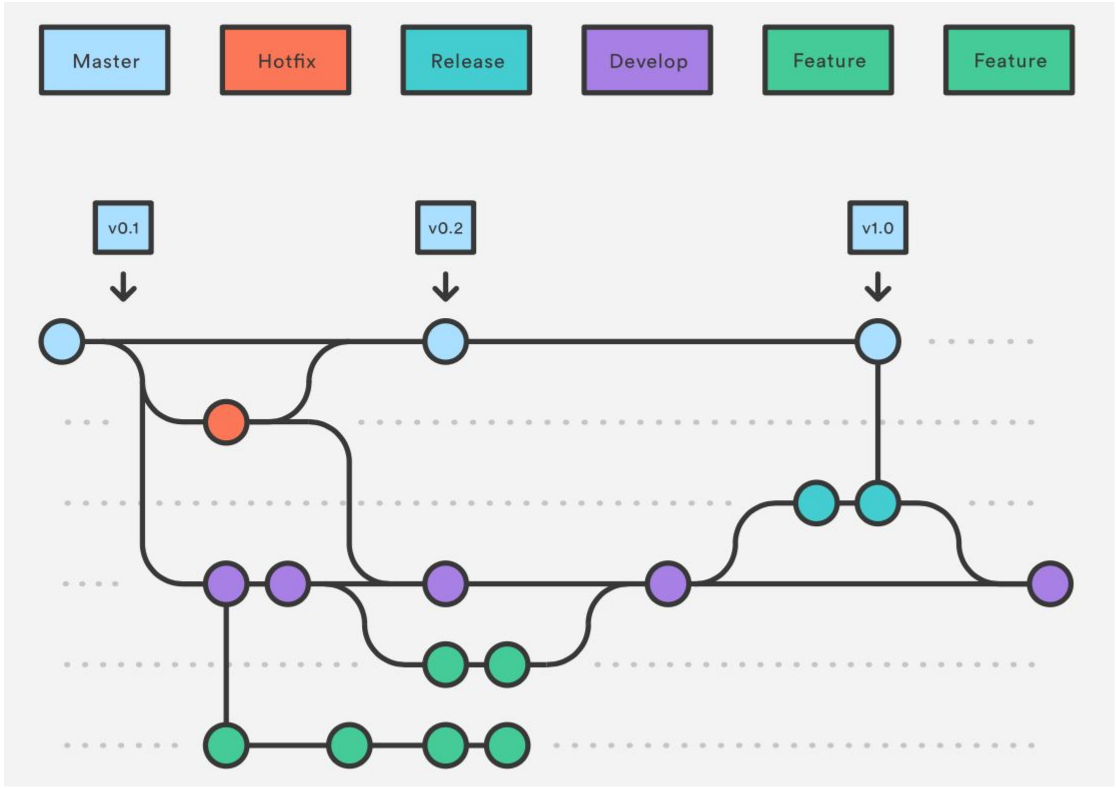
GitFlow release branches (eventually into master)



no new features after this point—only bug fixes, docs, and other release tasks

GitFlow hotfix branches

used to quickly patch production releases



Summary

- Version control has many advantages
  - History, traceability, versioning
  - Collaborative and parallel development
- Collaboration with branches
  - Different workflows
- From local to central to distributed version control