# Félag tölvunar- og kerfisfræðikennara í framhaldsskólum

Hannes Pétursson

3. desember 2016

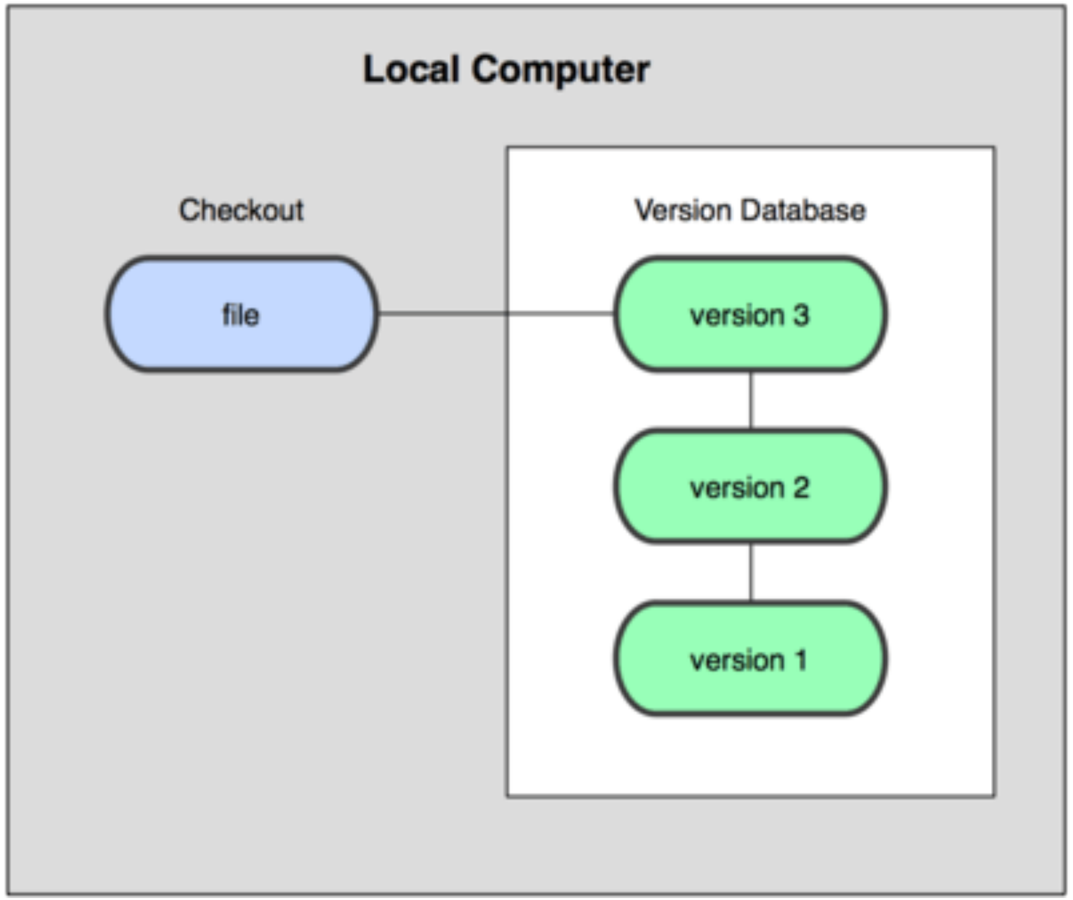
## History

- Gran-daddy of all version control systems was SCCS
  - Written in 1972 by Bell Labs
- Most of open source vcs evolve from SCCS like:
  - RCS, CVS and Subversion
- Example of commercial tools:
  - Perforce, StarTeam, ClearCase, AccuRev and Team Foundation System (TFS), BitKeeper

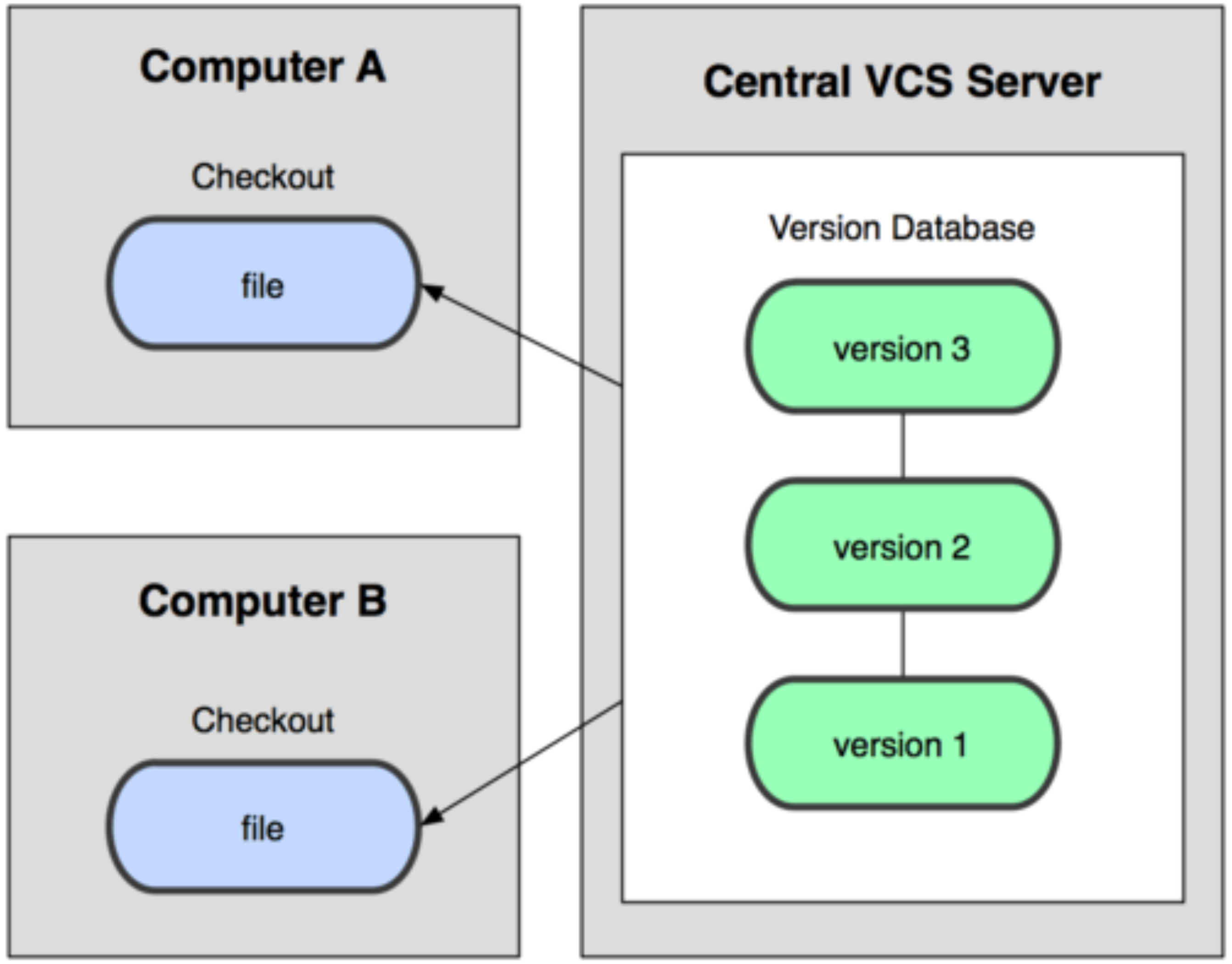
## Version Control Types

- Local Version Control
  - rcs
- Central server
  - Subversion
  - CVS
  - TFS
- Distributed
  - Git
  - Mercurial
  - Bazaar

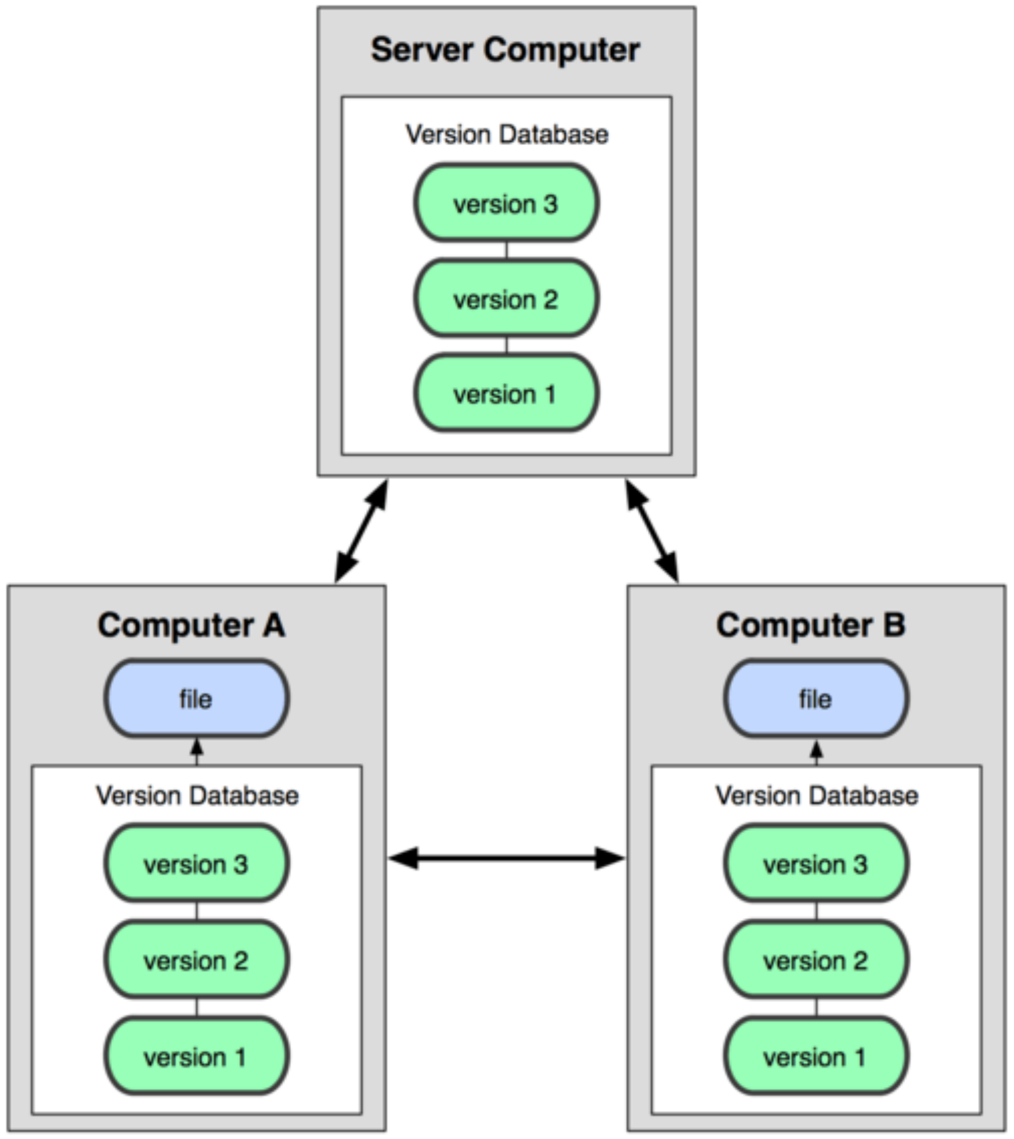
## Local Version Control System



# Centralized Version Control System



# Distributed Version Control System



# Distributed Version Control

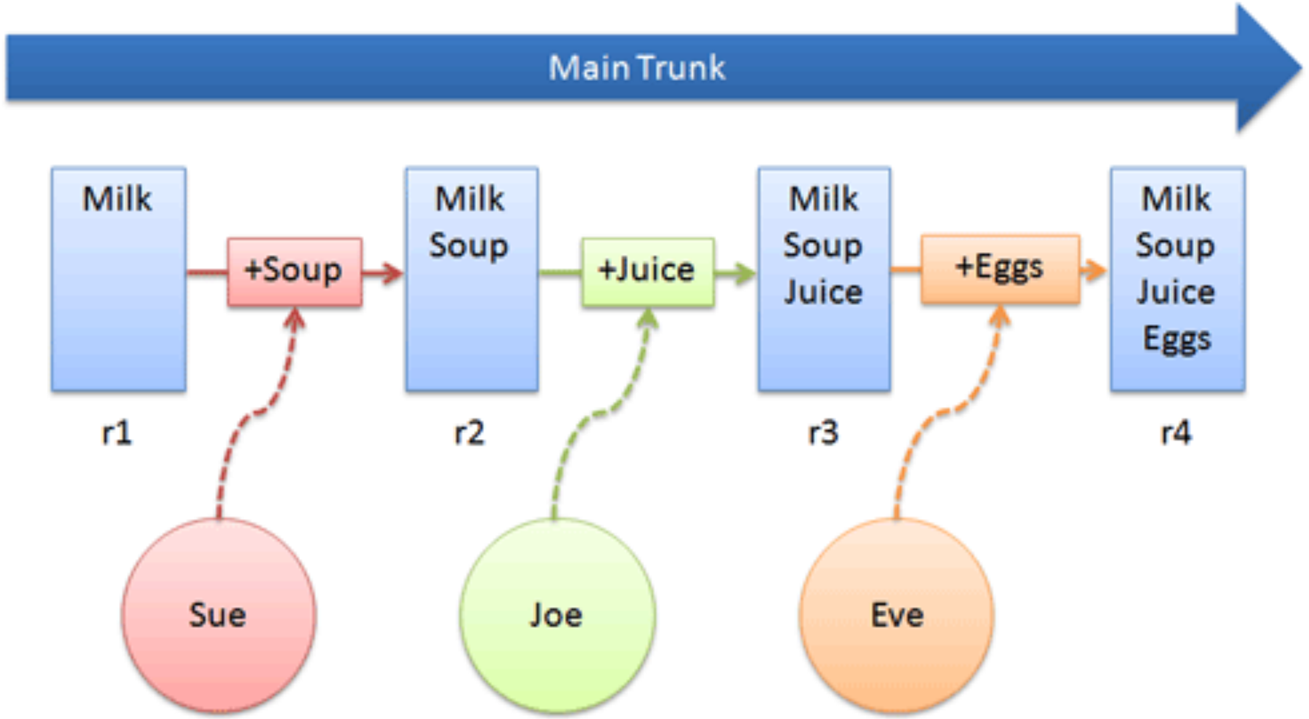
- Centralized version control focuses on **synchronizing, tracking, and backing up files**
- Distributed version control focuses on **sharing changes**; every change has a guid or unique id.

- Done right, you can get the best of both worlds: **simple merging** and **centralized releases**.

## Distributed Version Control

- Centralized VCS are not good in merging and branching
- Distributed systems make branching and merging painless because they rely on it.

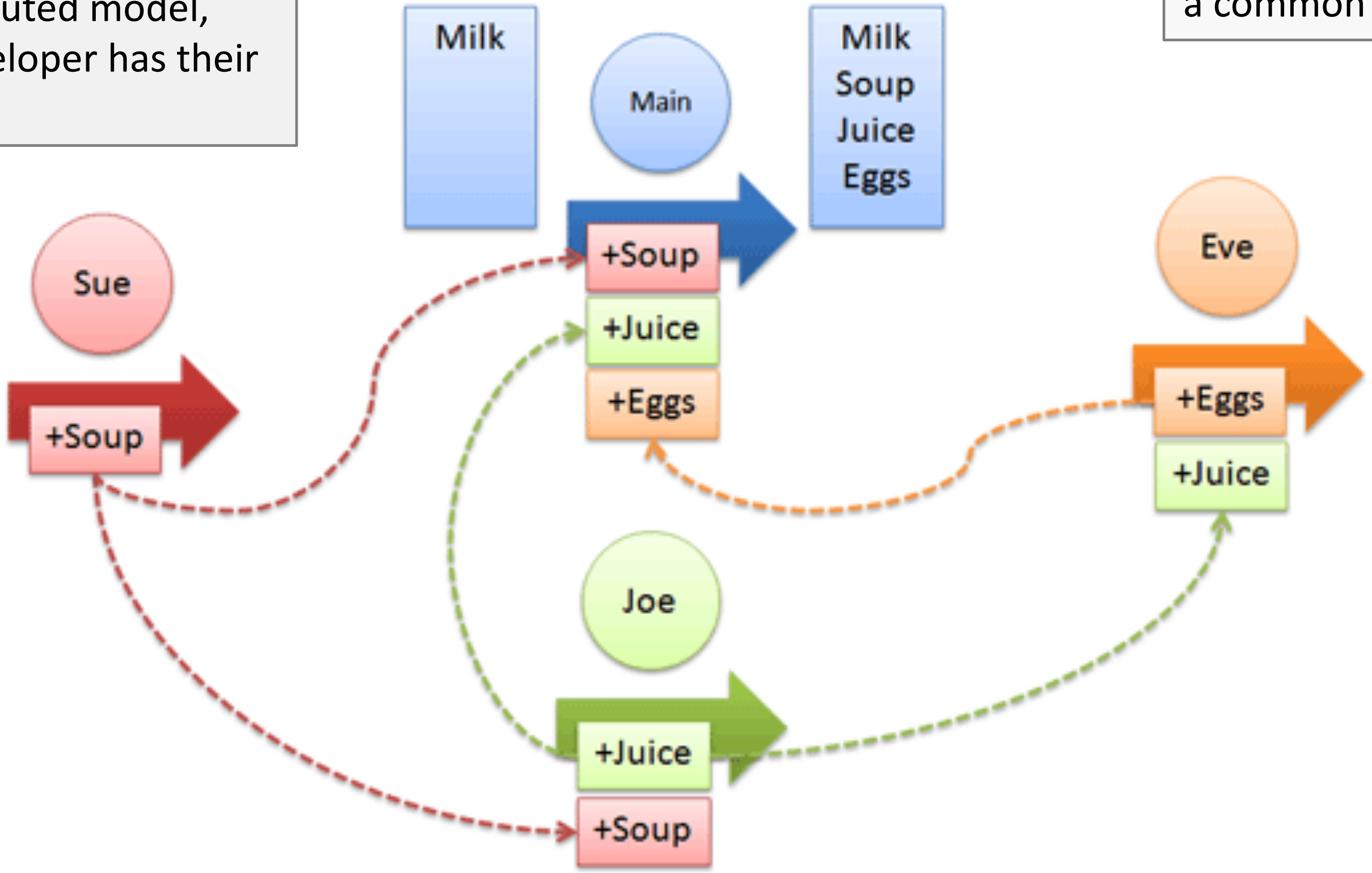
# Centralized VCS



# Distributed VCS

In a distributed model, every developer has their own repo

If desired, everyone can push changes into a common repo



## New Terminology

- **push**: send a change to another repository (may require permission)
- **pull:** grab a change from a repository

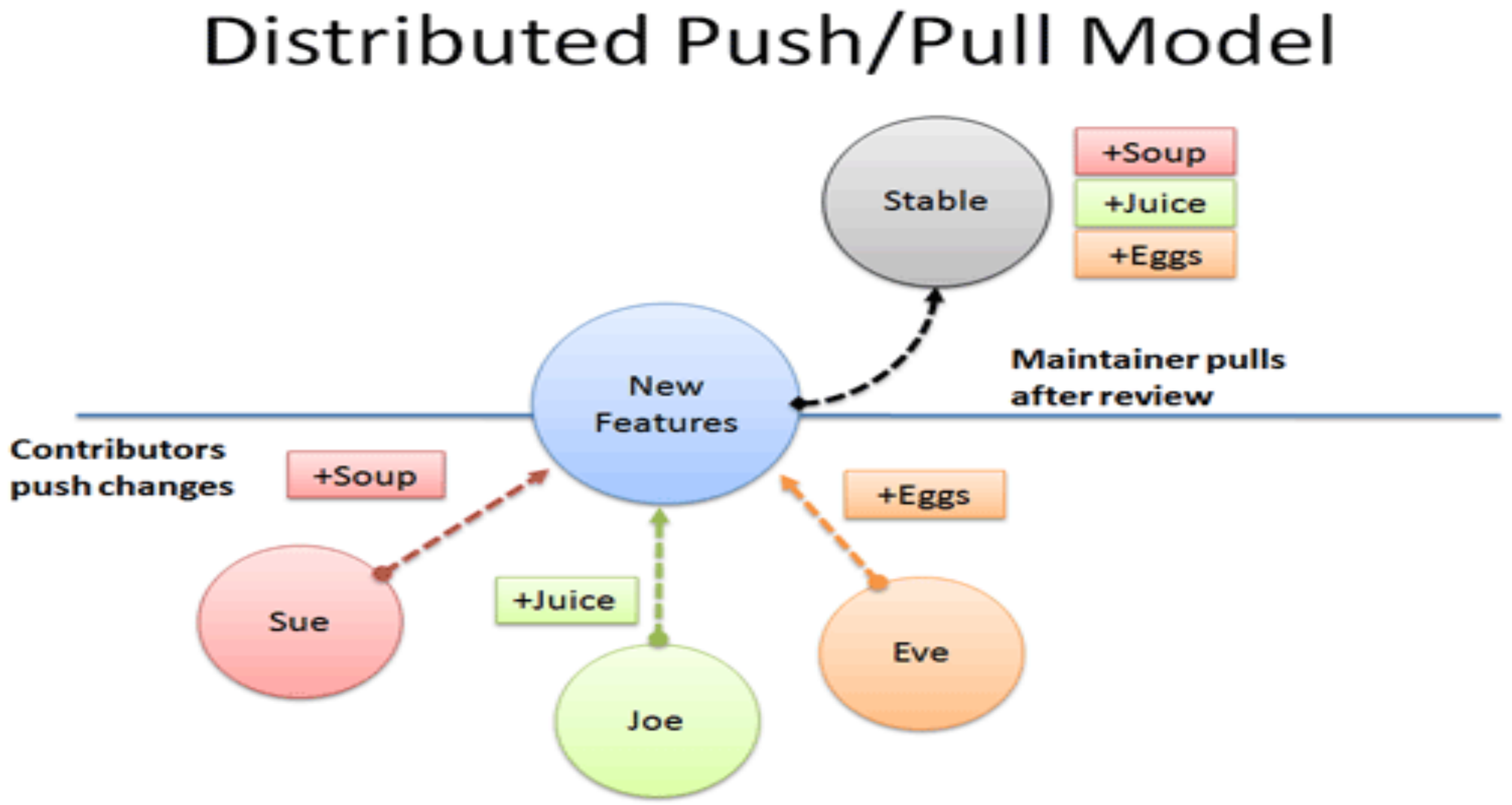
## Key Advantages

- Everyone has a local sandbox
- It works offline
- It’s fast
  - Diff, commits and reverts are all done locally
- It handles changes well
  - Built around sharing changes (each change has guid)
- Branching and merging is easy
  - every developer “has their own branch”
- Less management
  - No “always-running” server software to install
  - DVCSes may not require you to “add” new users (pull URLs)

## Key Disadvanges

- You still need a backup
- There’s not really a “latest version”
  - A central location helps clarify what the latest “stable” release is.
- There aren’t really revision numbers
  - Every repo has its own revision numbers, instead, people refer to change numbers (guid’s)
  - Thankfully, you can tag releases with meaningful names
- Distributed systems have no forced structure
  - You can create “centrally administered” locations or keep everyone as peers

# One way to organize a distributed project


Distributed Push/Pull Model
Stable
+Soup
+Juice
+Eggs
New Features
Maintainer pulls after review
Contributors push changes
+Soup
+Eggs
+Juice
Sue
Joe
Eve

git

## Git History

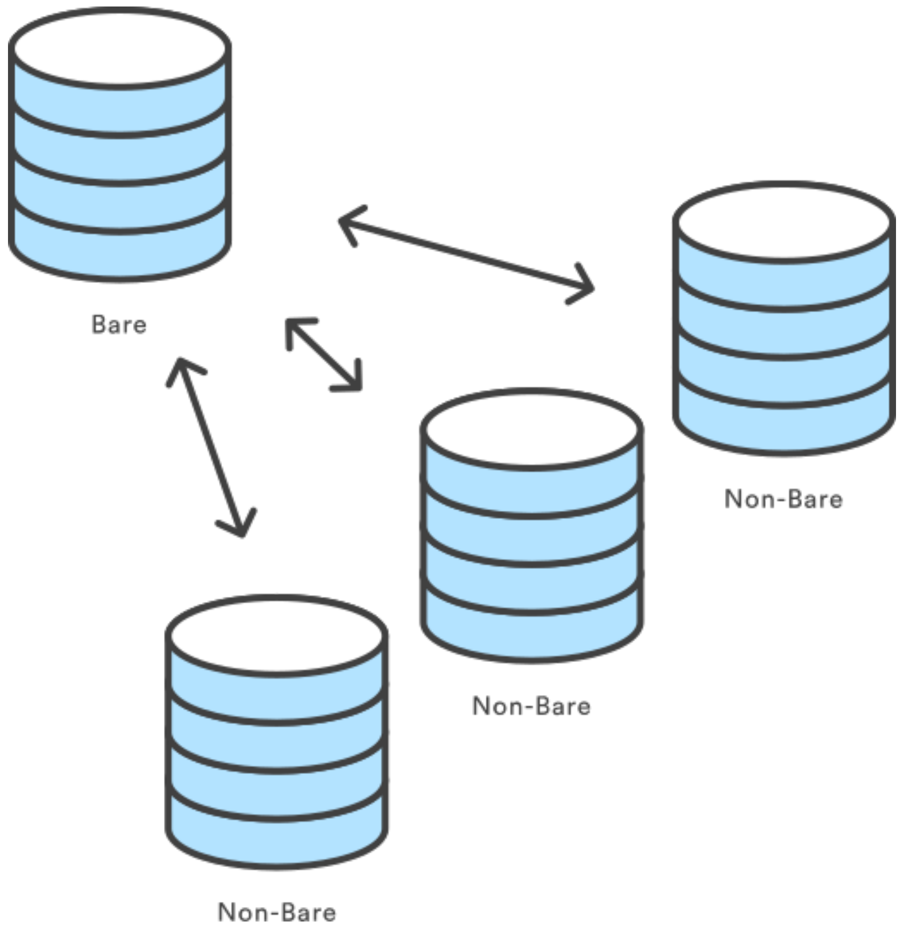
- Originally the Linux kernal was developed without vcs.
- In december 1999, started to use BitKeeper
- BitKeeper stoped supporting Linux
- Linus Thorvalds started on Git to maintain the Linux kernel

## Good Material on Git

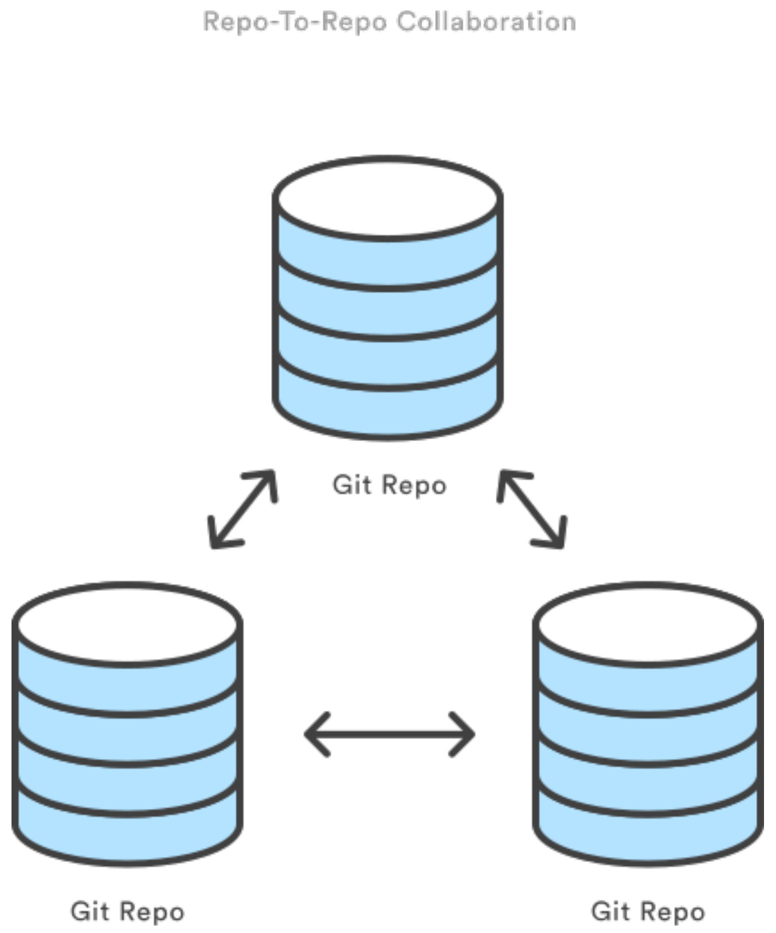
- Open Source leiðbeiningar á íslensku
  - https://github.com/gaui/git
- https://www.atlassian.com/git/tutorials/
- https://git-scm.com/book/en/v2

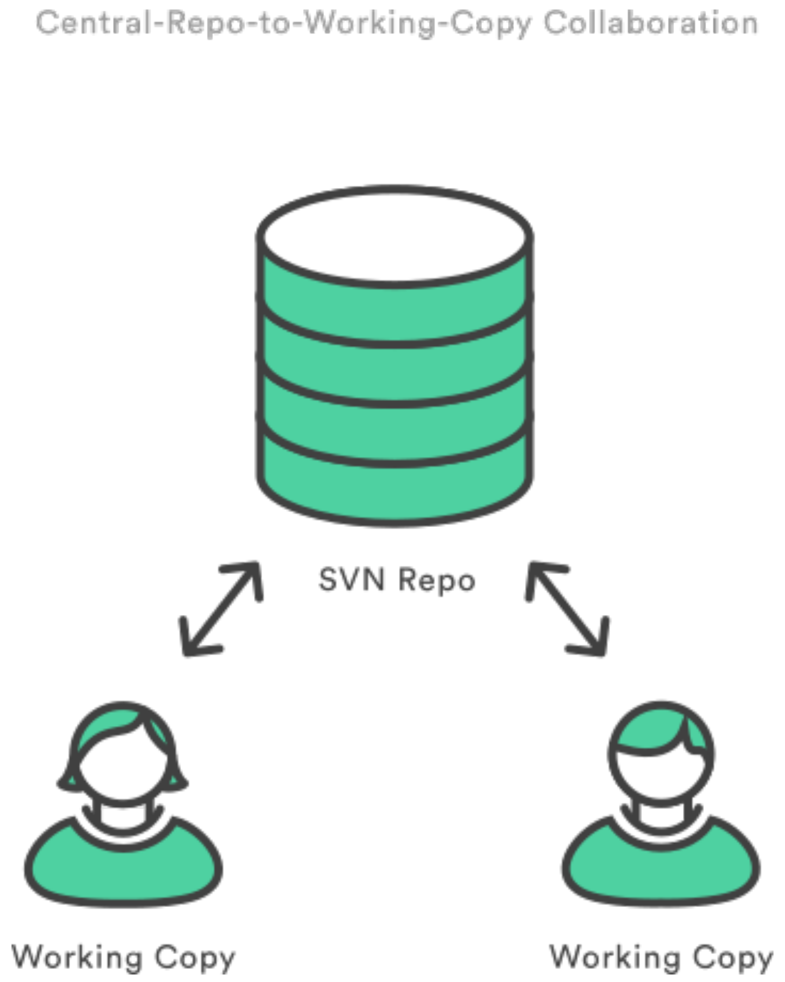
## Setting up repository

## Bare Repository



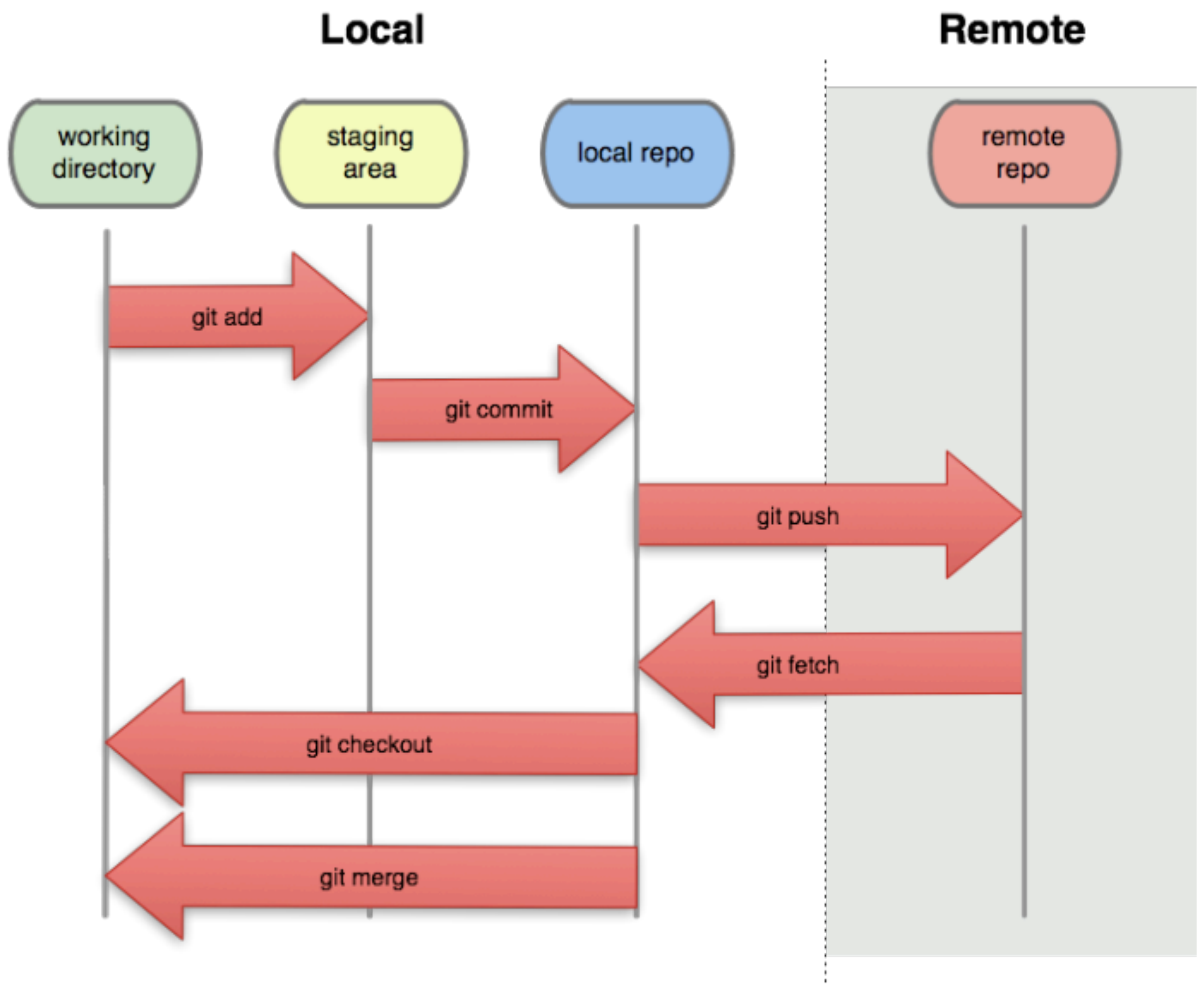
# Distributed vs centralized



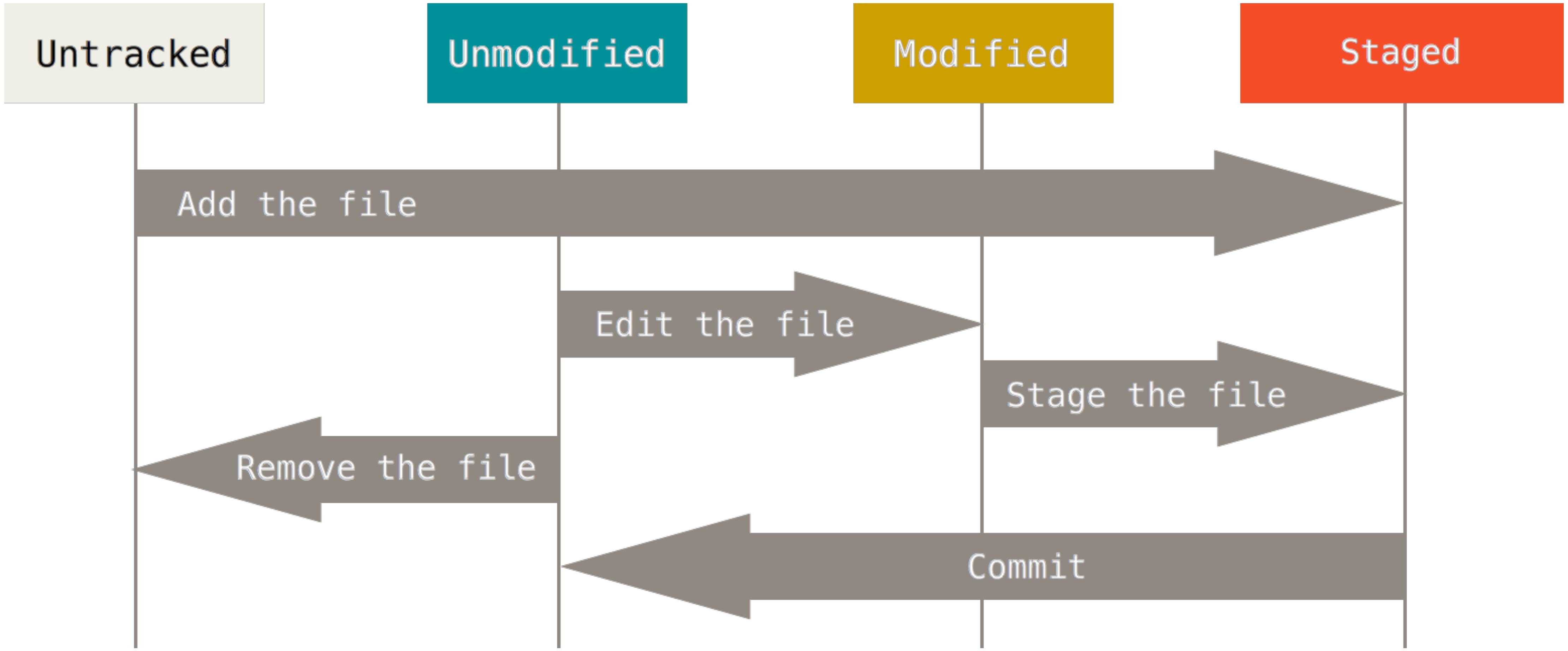


## Saving changes

# Local vs Remote repository



## Statuses in local repository



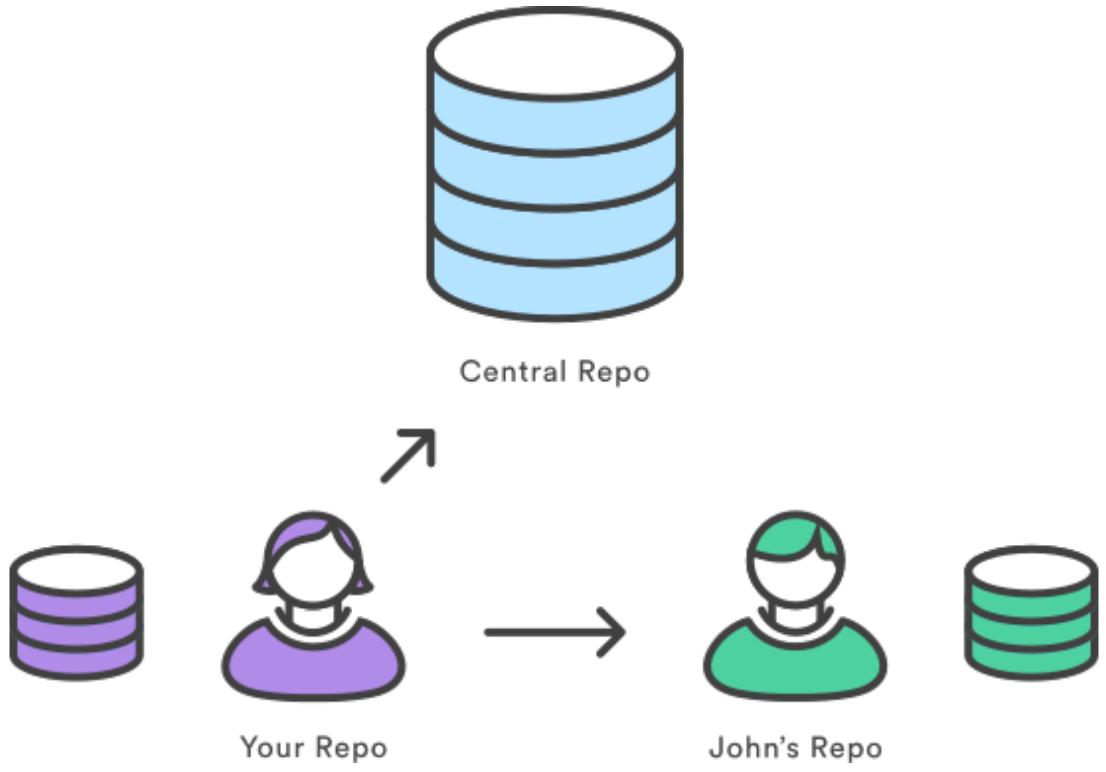
## Syncing

## Working with Remotes

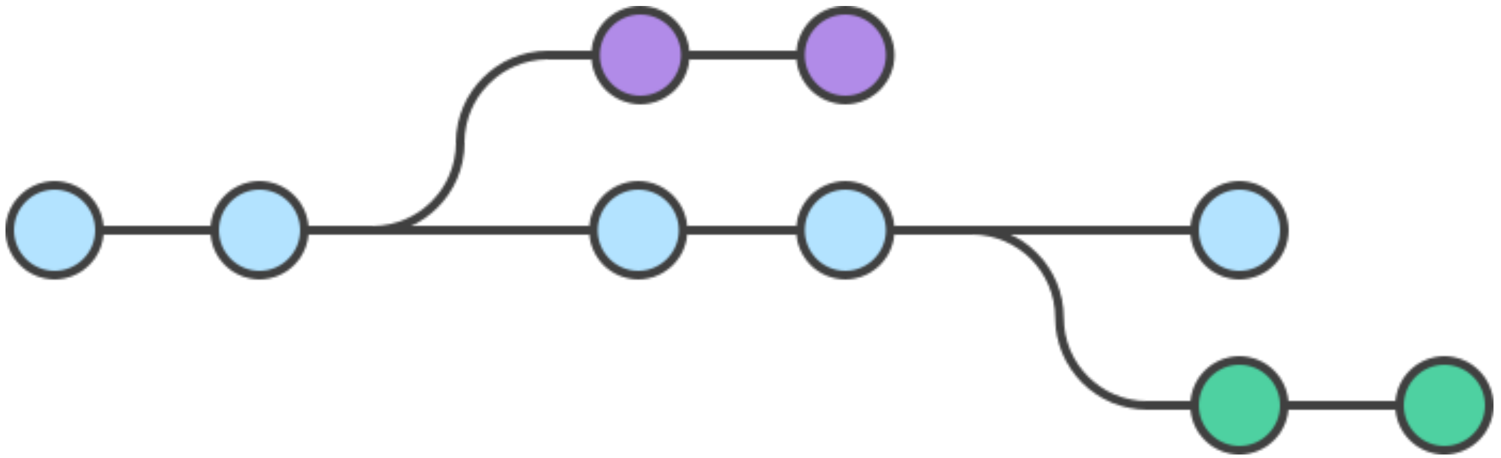
- Remote repositories are versions of your project that are hosted on the Internet or network somewhere. You can have several of them, each of which generally is either read-only or read/write for you.
- Each developer has an entirely isolated development environment
  - Needs to manually pull upstream commit into their local repository or push their local commits back up to the central repository
- git clone automatically creates a remote connection called origin
  - Most Git-based projects call their central repository origin

# git remote

- Create ,view and delete connections to other repositories
- git remote
  - List the remote connections
- git remote add <name> <url>
  - create new connection to a remote repository
- git remote rm <name>
- git remote rename <old> <new>

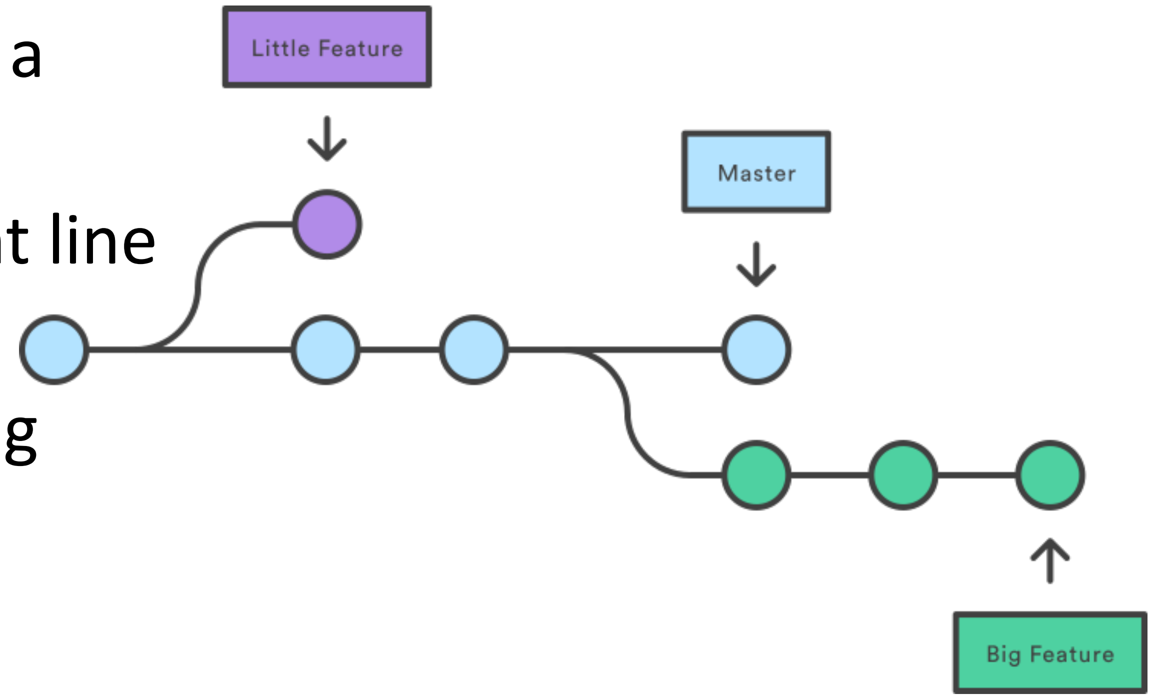


## Using Branches



# Branches

- Creating branches is like requesting a new project history.
- A branch represents an independent line of development.
- Way to request a brand new working directory, staging area and project history
  - A fork in the history of the project



## Branching

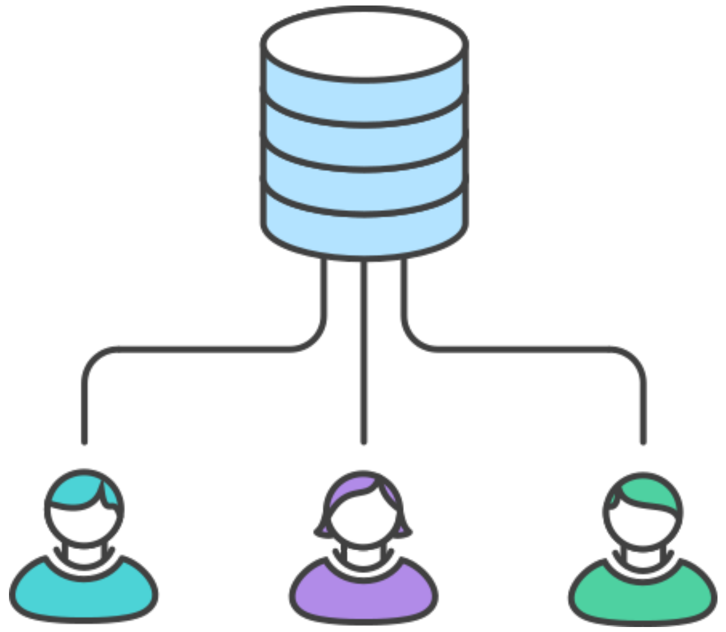
- Recommended to spawn a new branch to work on new feature or a bugfix to encapsulate your changes.
- Unstable code is never committed to the main code base.
- Gives you change to clean up your feature's history before merging it into the main branch.
- It makes it ridiculously easy to try new experiments without the fear of destroying existing functionality,

## Comparing Workflows in Source Control Systems

## Comparing Workflows

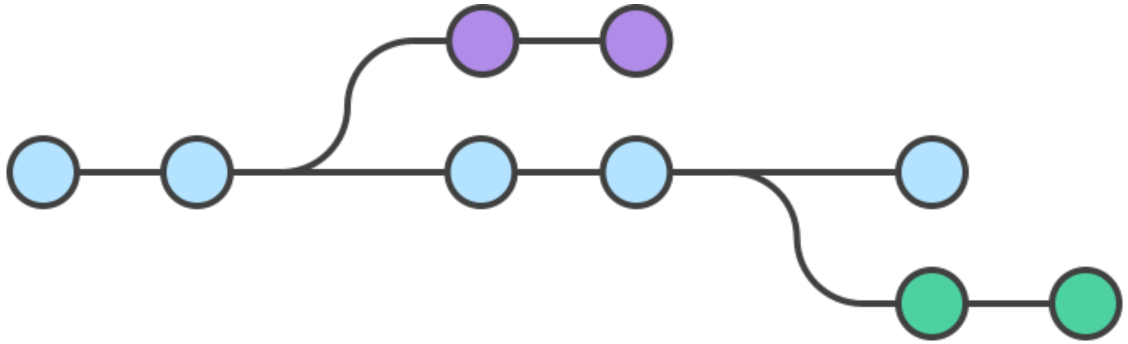
- Centralized Workflow
- Feature Based Workflow
- Gitflow Workflow
- Forking Workflow

## Centralized Workflow



- Single point-of-entry for all changes to the project.
- Default development branch is called master
  - All changes are committed into master
- Commits are stored locally
  - *Pushed* when convenient for developer to the central repository

## Feature Branch Workflow



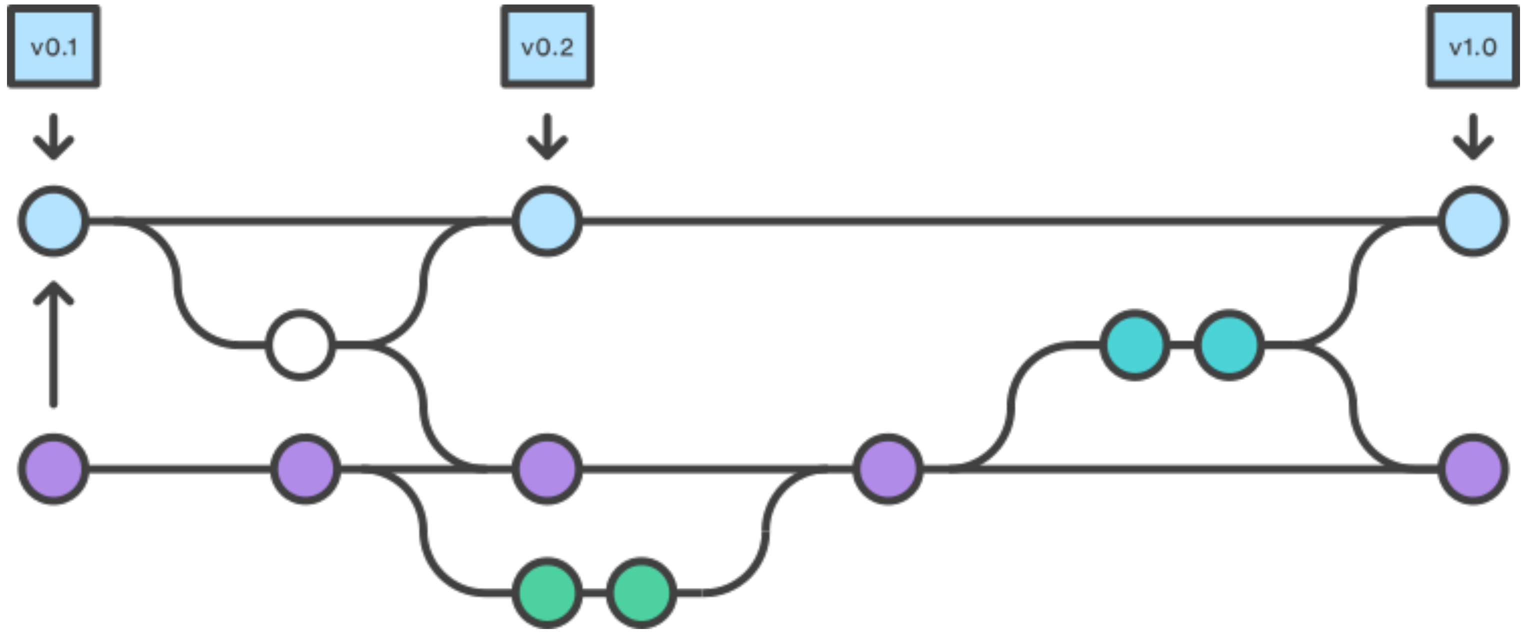
## Feature Branch Workflow

- Adding feature branches to our development process encourages collaboration and streamline communication between developers
- The core idea it that all feature development should take place in a dedicated branch instead of the *master* branch.
  - Makes it easy for multiple developers to work on a particular feature without disturbing the main codebase.
  - Master will never contain broken code, which is a huge advantage for continuous integration environments.
- Makes it possible to leverage *Pull Requests*

## Feature Branch Workflow

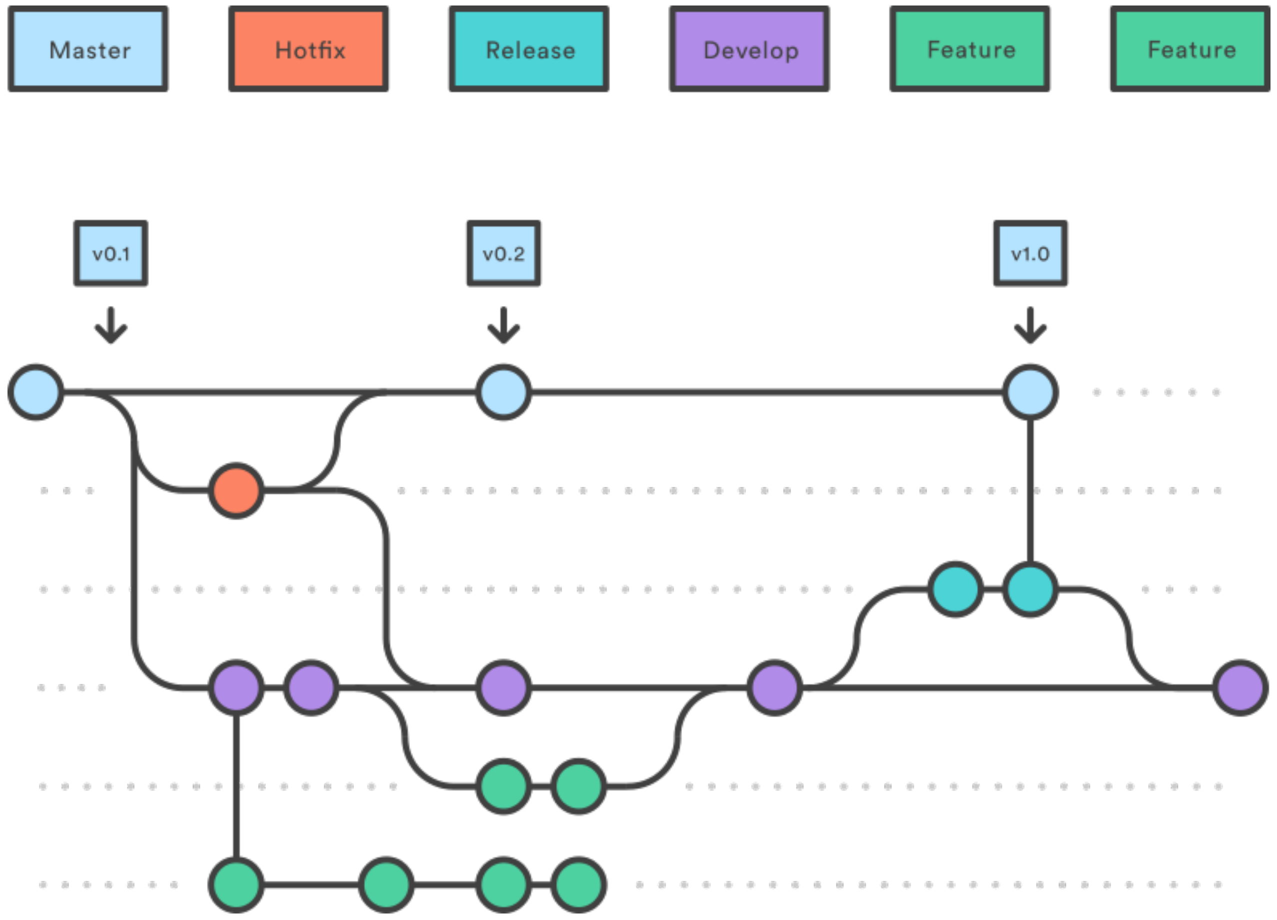
- Developers a new branch every time they start work on a new feature.
  - User descriptive names, the idea it to give a clear, highly-focused purpose to each branch.
- There is no technical distinction between the master branch and feature branches
  - Developers can edit, stage and commit changes to feature branch just as they did in the Centralized Workflow
- Feature branches can (and should) be pushed to central repository.
  - To share with other developers

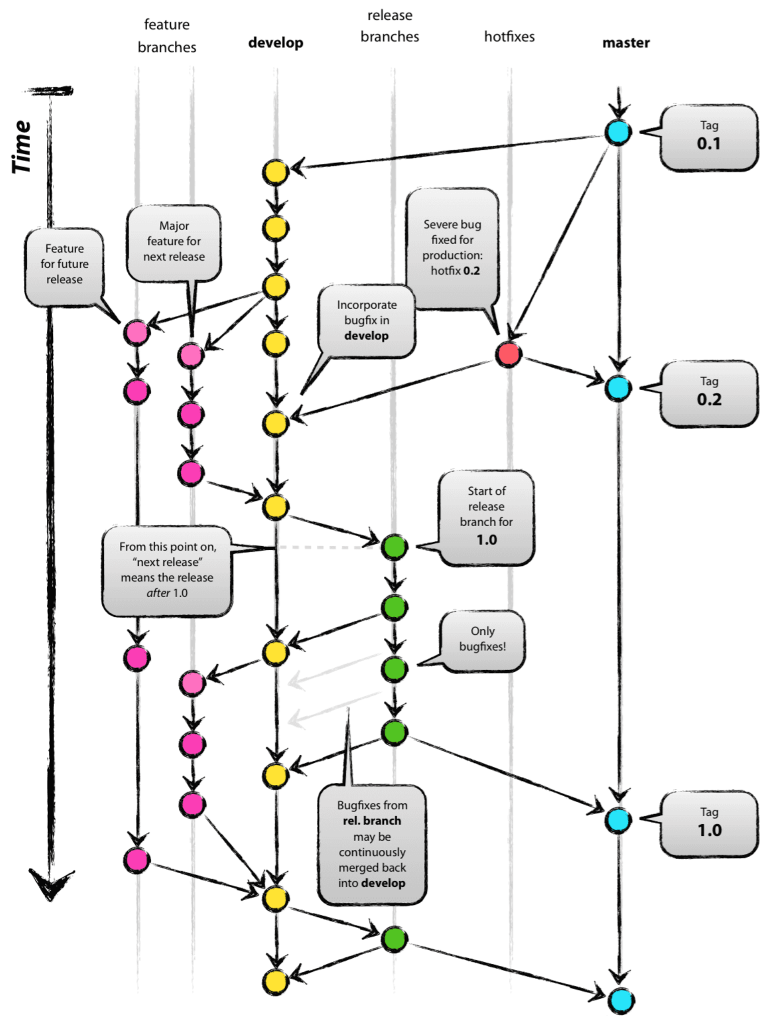
# Gitflow Workflow



# Gitflow Workflow

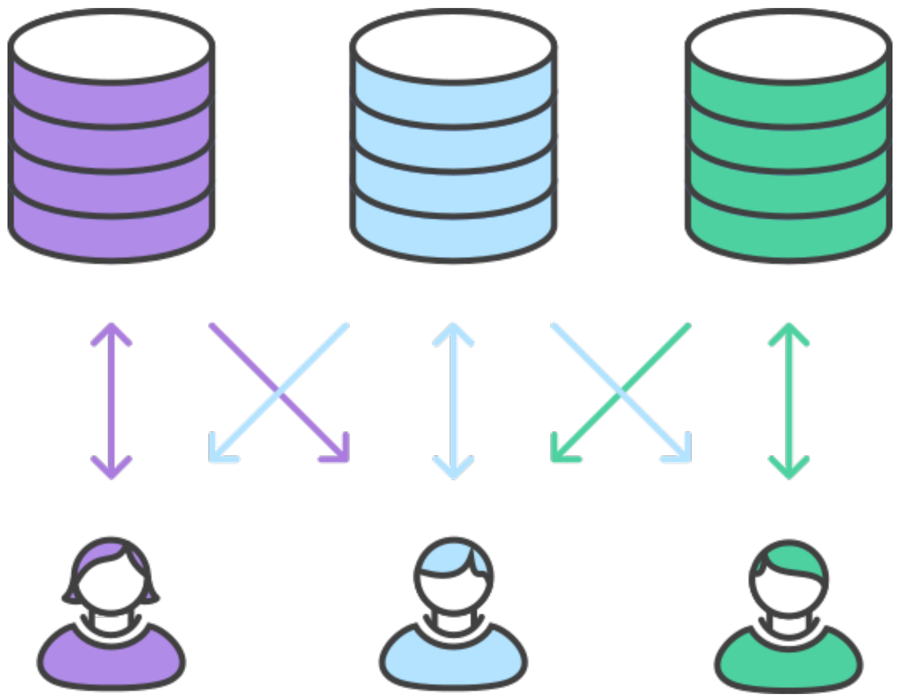
- The Feature Branch Workflow is an incredible flexible way to develop a project. Sometimes it's too flexible.
  - It's beneficial to assign more specific roles to different branches.
  - The Gitflow workflow is a common pattern for managing feature development, release preparation, and maintenance.
- The workflow is derived from Vincent Driessen at nvie
- Defines a strict branching model designed around the project release
- Robust framework for managing larger projects
- No new concepts or commands instead, it assigns very specific roles to different branches and defines how  and when they should interact.


Master
Hotfix
Release
Develop
Feature
Feature
v0.1
v0.2
v1.0


feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Tag 1.0

http://nvie.com/posts/a-successful-git-branching-model/

## Forking Workflow



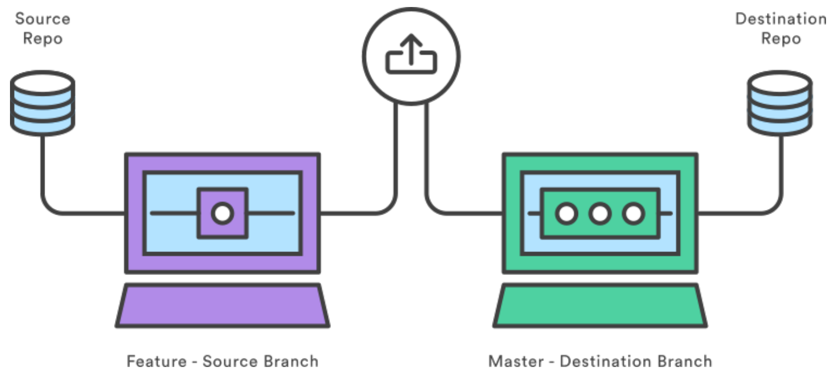
## Forking Workflow

- Instead of using a single server-side repository to act as the “central” codebase, it gives every developer a server-side repository. This means that each contributor has not one, but two Git repositories: a private local one and a public server-side one.
- The result is a distributed workflow that provides a flexible way for large, organic teams (including untrusted third-parties) to collaborate securely. This also makes it an ideal workflow for open source projects.

## Making a Pull Request

# Pull Request

- Pull Request is a mechanism for a developer to notify team members that they have completed a feature
- When you file a pull request, all you're doing is requesting that another developer (e.g., the project maintainer) pulls a branch from your repository into their repository



## Pull Request

- On a hosted Git service like GitHub and Bitbucket you get much more that a notification.
  - It’s a dedicated forum for discussing the proposed feature.
  - Teammates can post feedback in the pull request and even tweak the feature, everything is tracked directly inside of the pull request.
  - This encourages much more streamlined workflow

  - See more details on the GitHub help:
    - https://help.github.com/articles/using-pull-requests/

## Pull Request, general process

1. A developer creates the feature in a dedicated branch in their local repo.
2. The developer pushes the branch to a public repository (e.g. Github).
3. The developer files a pull request via Github
4. The rest of the team reviews the code, discusses it, and alters it.
5. The project maintainer merges the feature into the official repository and closes the pull request.

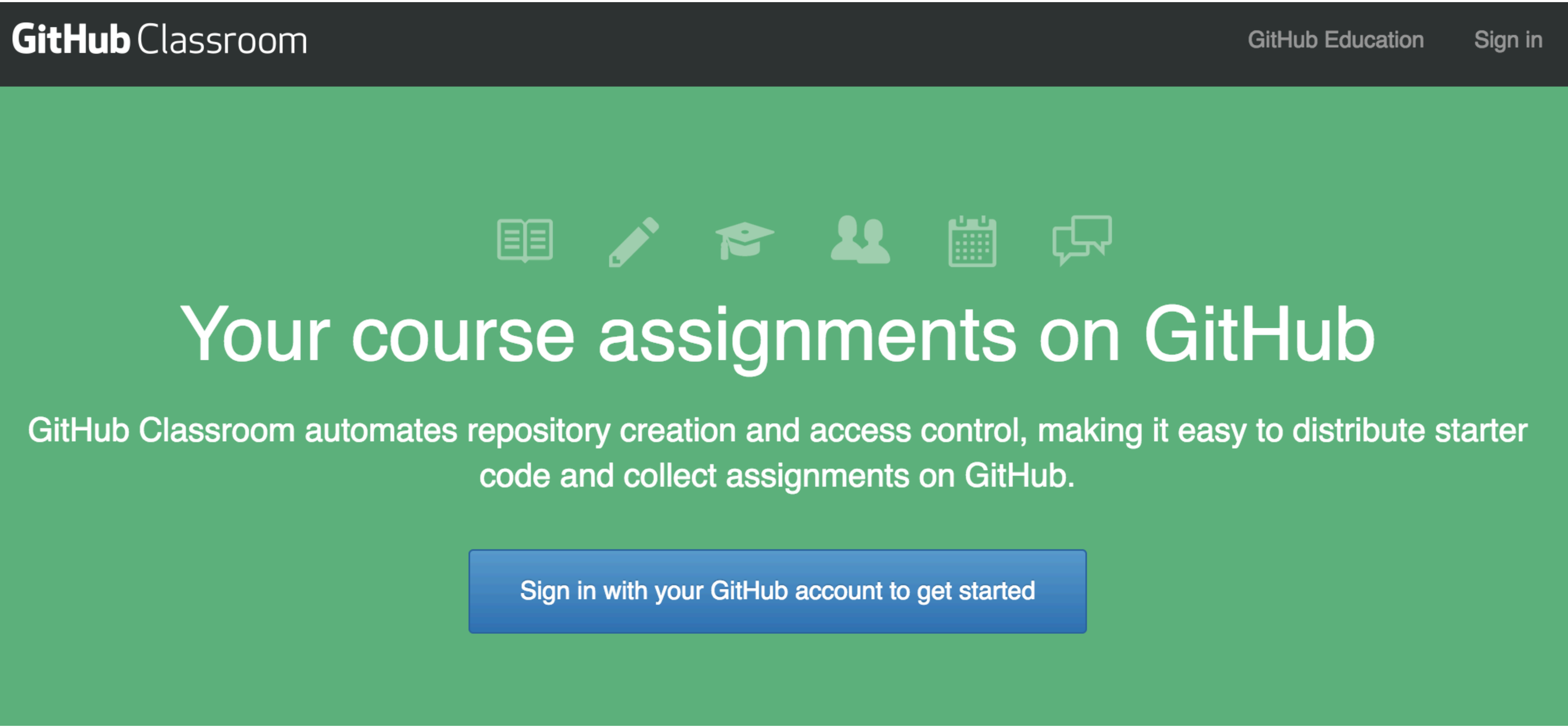
## Pull Requests

- Once someone completes a feature, they don’t immediately merge it into the master. Instead they push the feature branch to the central server and file a pull request asking to merge their additions into master.
- Developers can review the changes before they become part of the main codebase.
- You can think about pull requests as a discussion dedicated to a particular branch.

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

github

SOCIAL CODING



GitHub Classroom

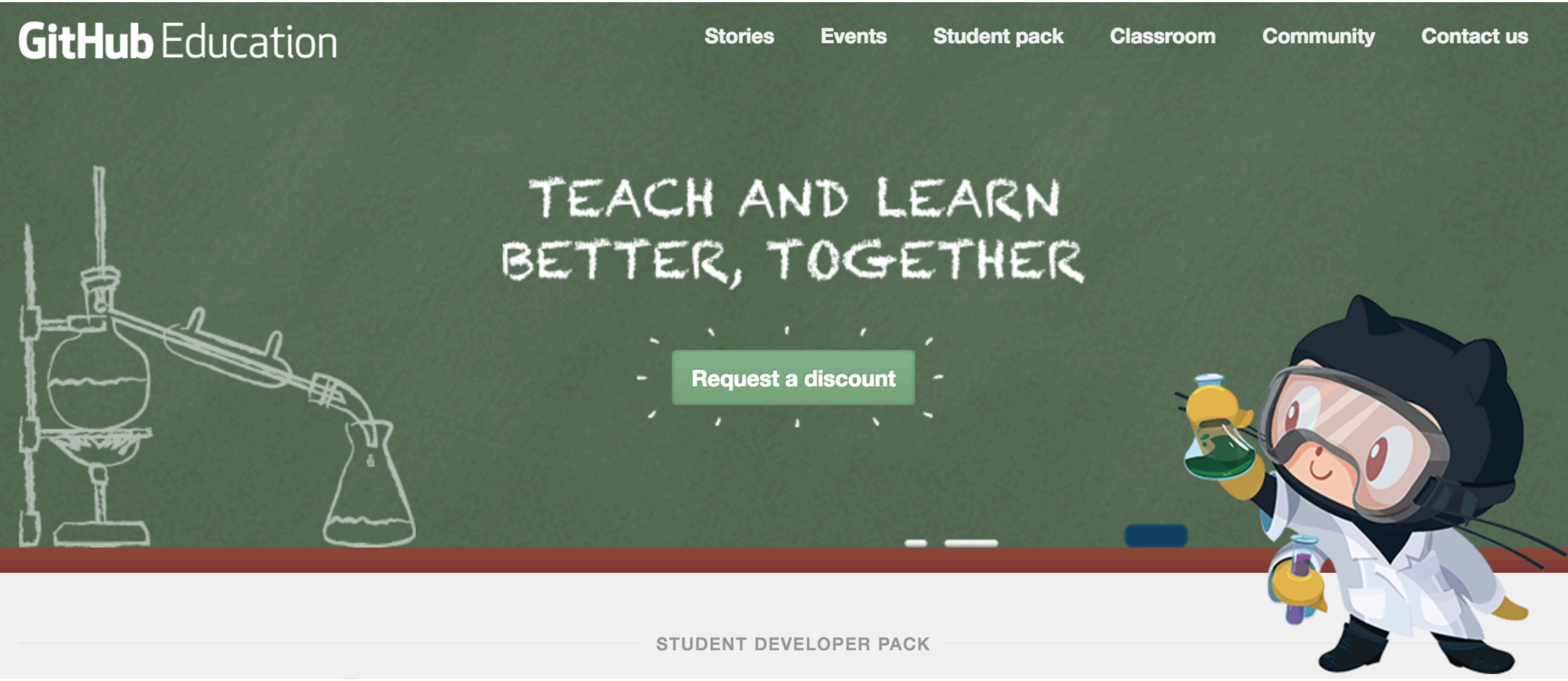
GitHub Education

Sign in

Your course assignments on GitHub

GitHub Classroom automates repository creation and access control, making it easy to distribute starter code and collect assignments on GitHub.

Sign in with your GitHub account to get started


GitHub Education
Stories
Events
Student pack
Classroom
Community
Contact us
TEACH AND LEARN BETTER, TOGETHER
Request a discount
STUDENT DEVELOPER PACK