OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG

INF FAKULTÄT FÜR INFORMATIK

Collaborative Software Development: Git and Github

Sandro Schulze, 26.04.2016

Important Note

- Git is a tool for
  - YOU to enable more efficient development
  - ME to investigate contributions of team members

- You HAVE TO use git
  - commit frequently
  - use the README to spread the word (e.g., how to use your app)

Roadmap

- Git
  - What makes it different?
  - Basic concepts
- Github
  - Social Coding in-the-larg
  - how to use it for an SW development project
- Travis CI
  - Basic idea
  - How does it integrate with the development process

Git

A brief introduction

“I am an egoistical bastard, and I name all my projects after myself. First Linux, now git”

–Linus Torvalds

History

- originally, the Linux kernel has been developed by sending around patches or archive files (1991—2002)
- in 2002: use of proprietary DVCS called BitKeeper relationship broke down in 2005
- Hence, they (most notably, Linus Torvalds himself) decided to develop a proprietary tool
  —> Git was born

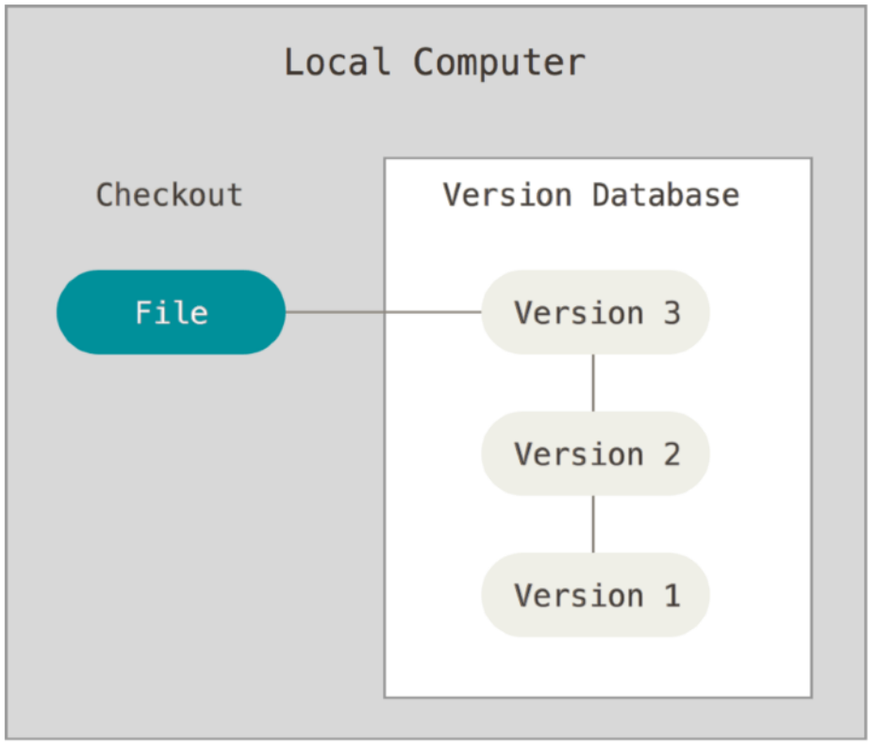
Goals

- Speed
- Simple Design
- support for non-linear development
- fully distributed
- able to handle large projects such as the Linux kernel efficiently

Git Basics

From Local Versioning to Distributed VCS

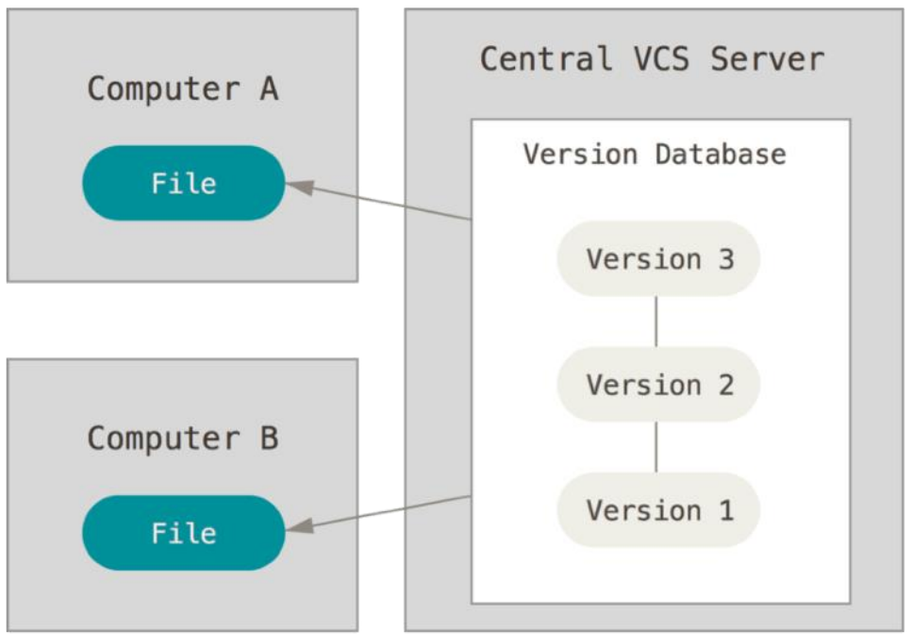
Version Control Systems (VCS)



- introduced to overcome simple copy&paste of files
- database to keep changes to files
- Tool: RCS (try rcs in your shell ;))
- Technique: storing of patch sets on disk

From Local Versioning to Distributed VCS (II)

Centralized Version Control Systems (CVCS)

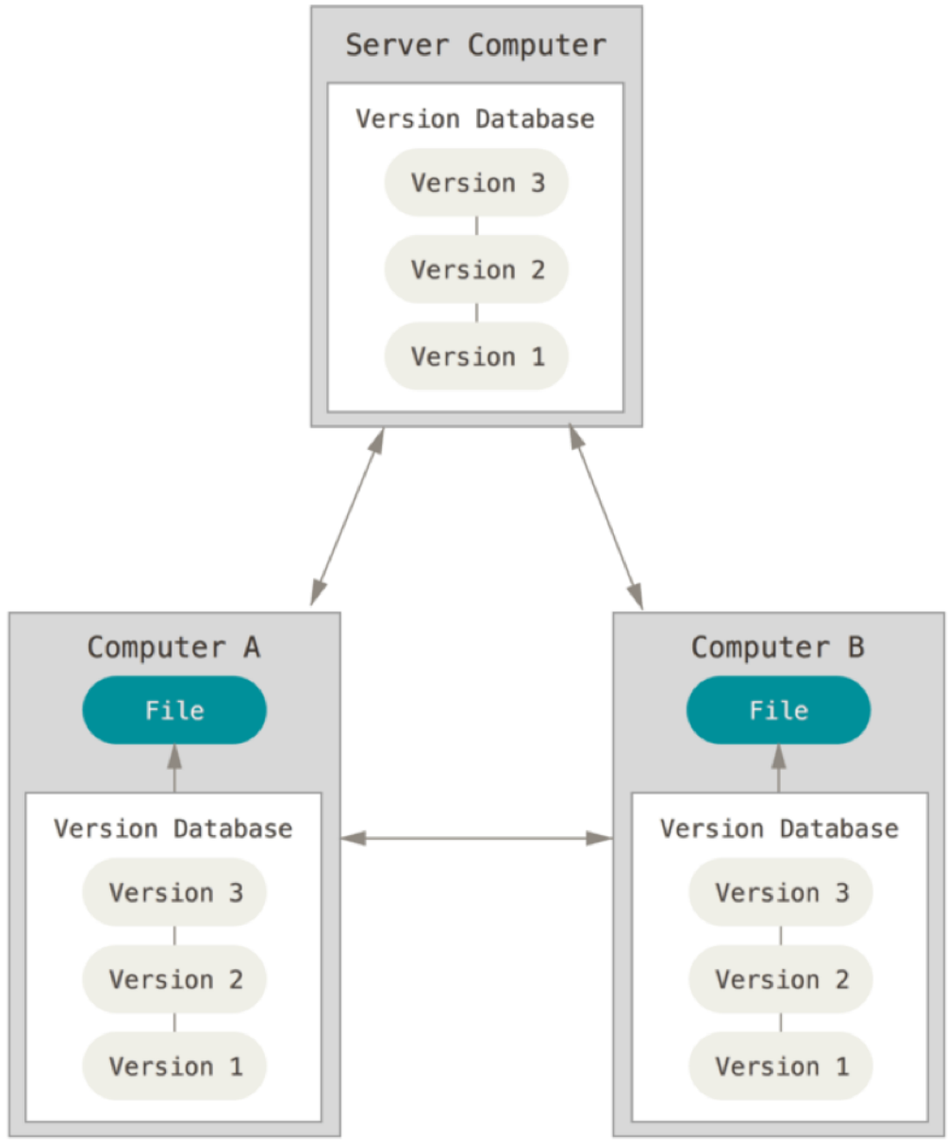


- Originated by need to collaboration between developers
- central server that contains DB with all change information (versions)
- de-facto standard for many years
- Tools: SVN, Perforce

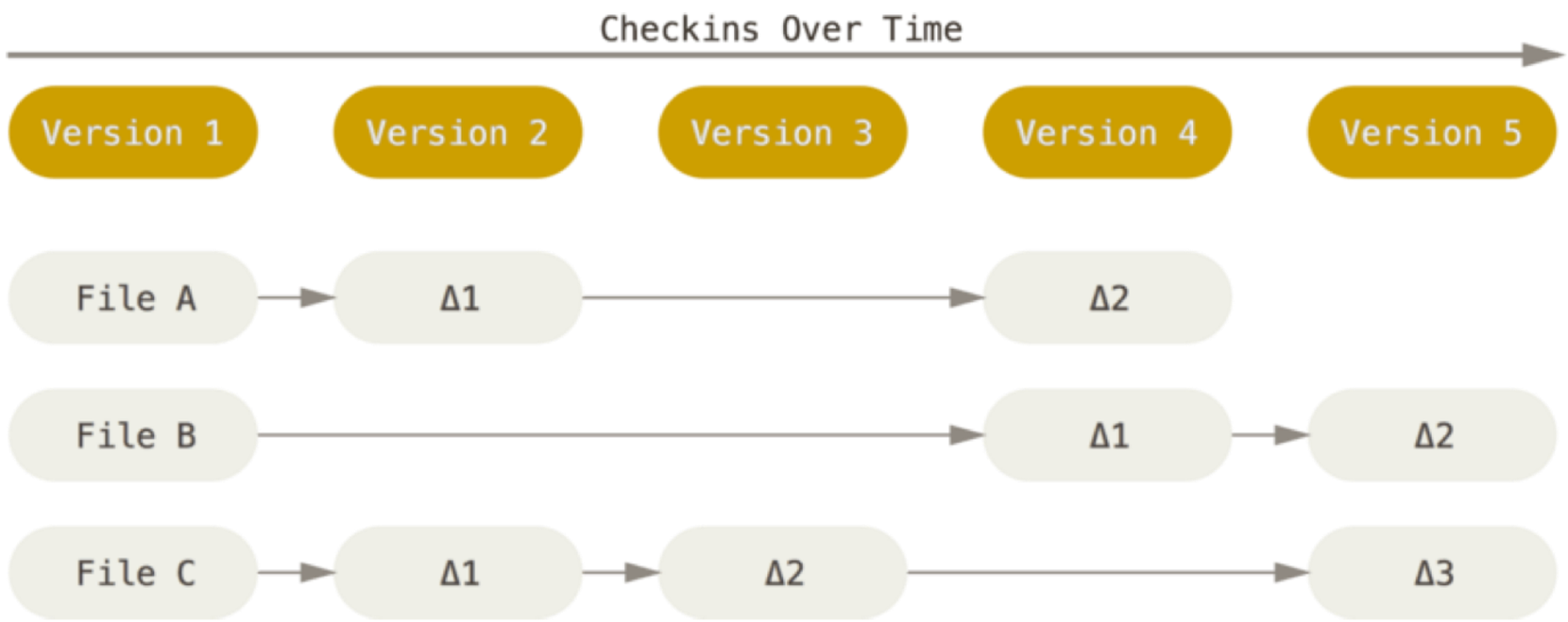
From Local Versioning to Distributed VCS (II)

Distributed Version Control Systems (DVCS)

- Clients not just check out, they fully mirror the repository (clone)
- Thus, can work even offline on the project
- support for having several repos of the same project —> enables new workflows (compared to CVCS)

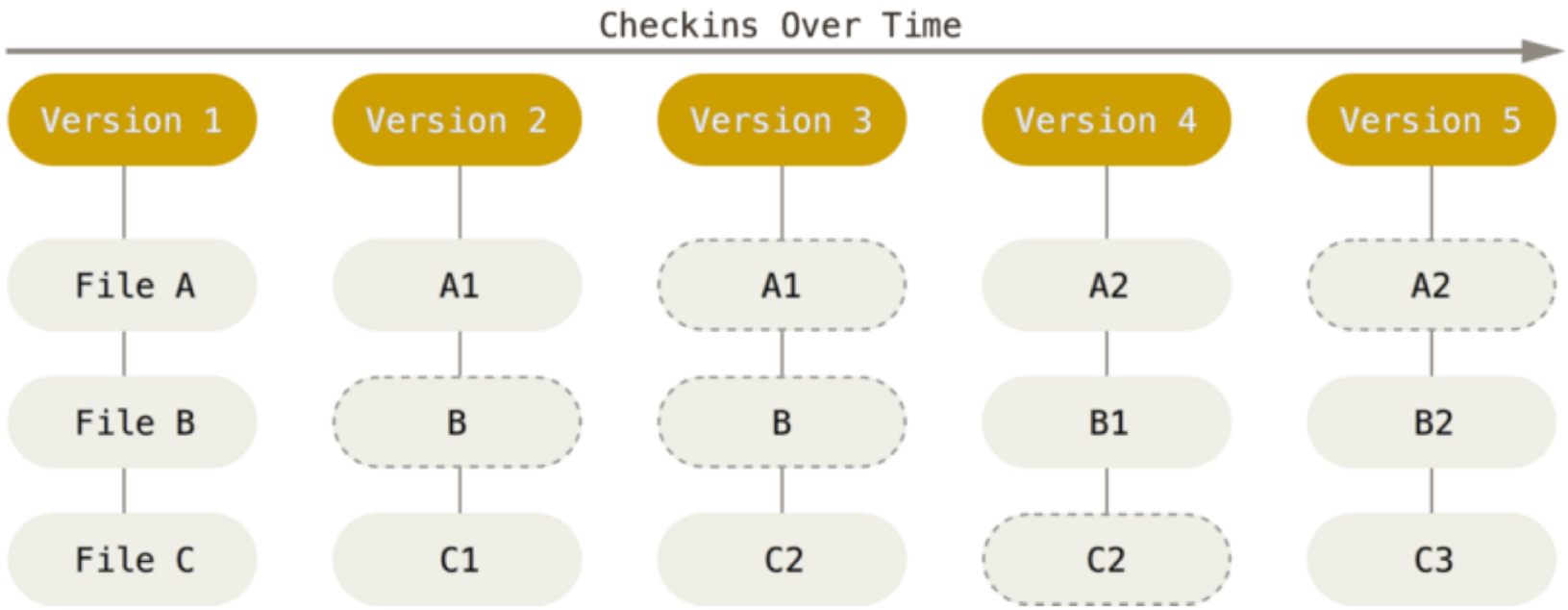


Data Handling — Snapshots, not differences



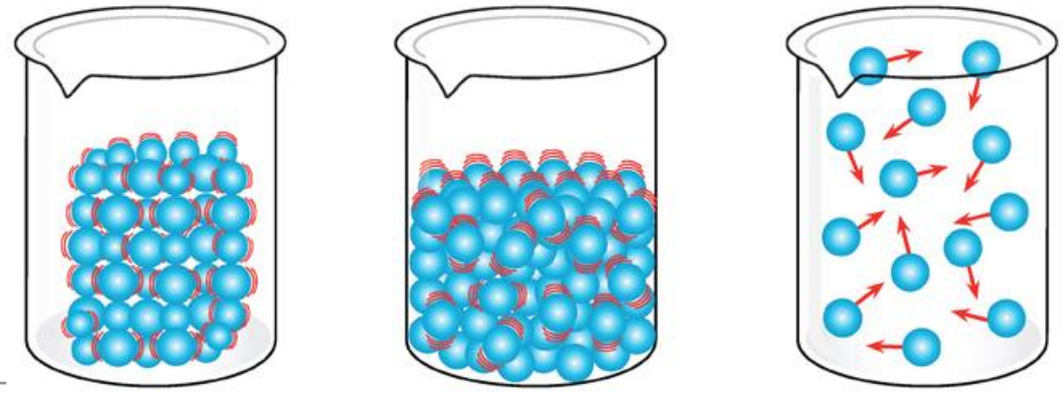
- In common version control systems (VCS) such as SVN, data is stored as changes to a base version of each file

Data Handling — Snapshots, not differences



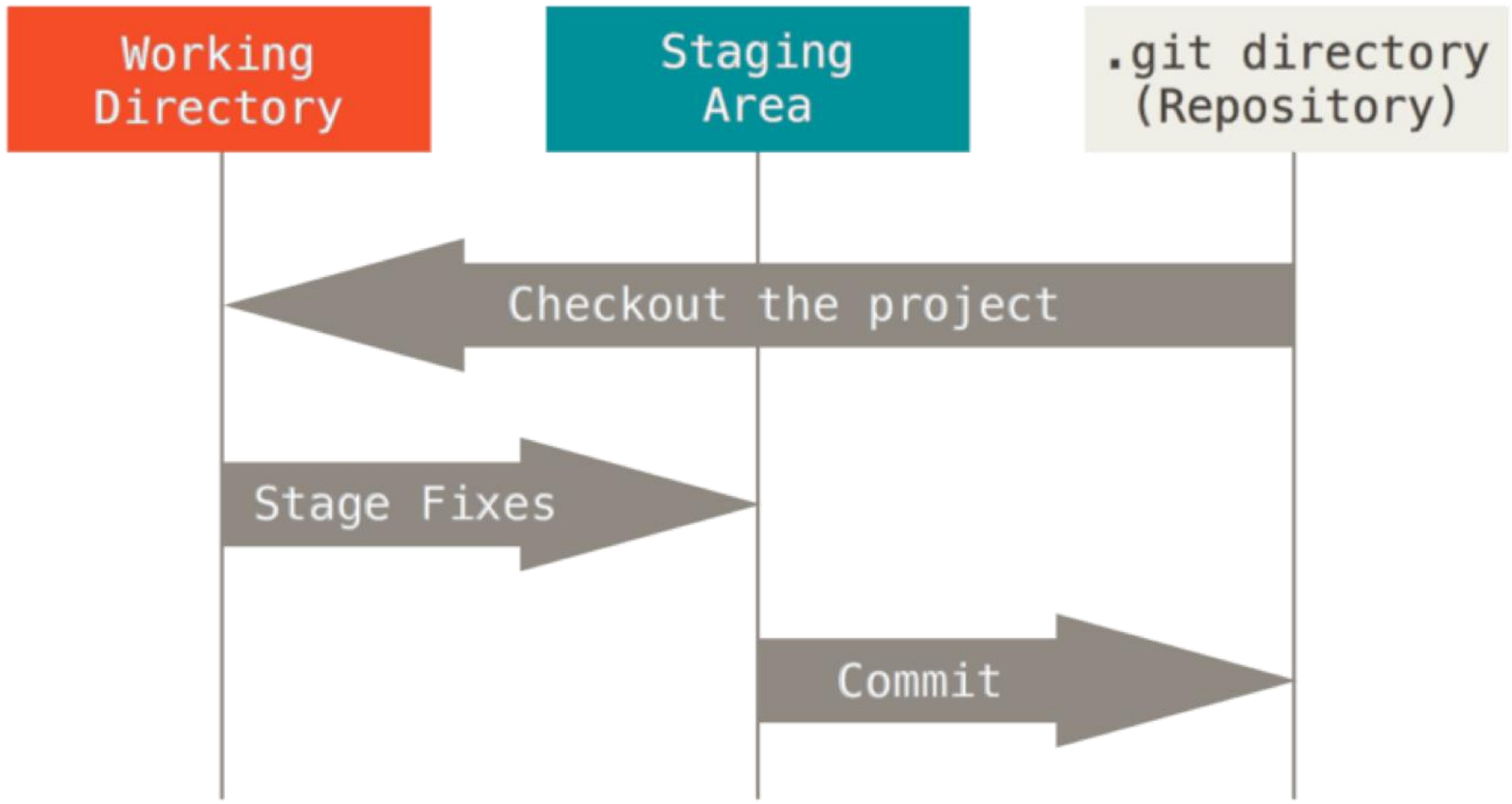
- In Git, data is handled rather as snapshots of a miniature file system
- On commit, the current “state” (i.e., snapshot) of your project is stored
- If files not change —> link to previous version of this file

The Three States



- Git has **three** main states your files can reside in: *committed, modified,* and *staged*
- committed —> safely stored in (local) database
- modified —> changed file, but not committed to DB yet
- staged —> marked file (current version) to be part of next commit snapshot
- These states are reflected by the actual workflow in git

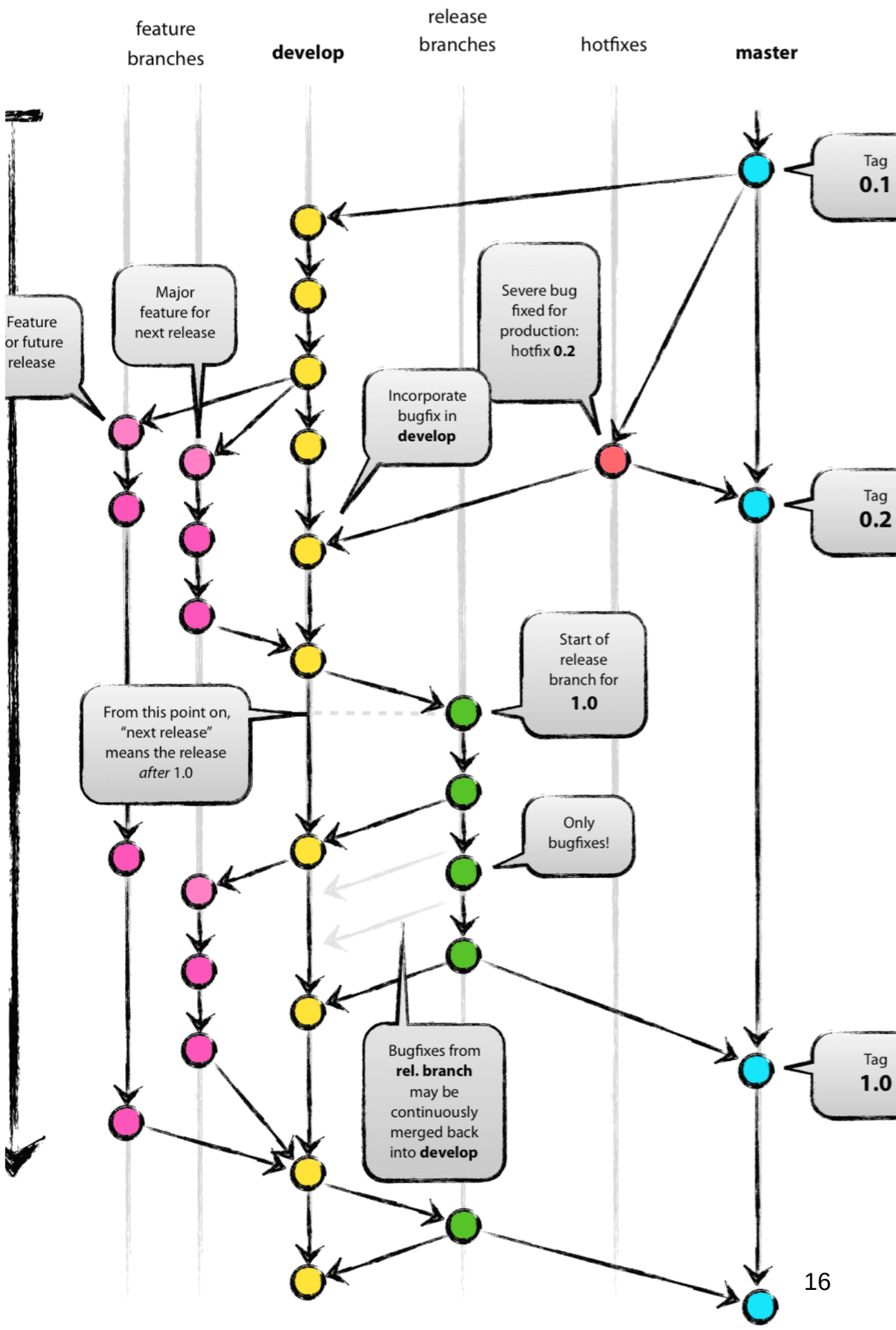
Structure of a Git Project



- Git directory —> stores metadata and object database
- working dir. —> single checkout of *one version* of the project
- staging areas —> file that stores information about what goes into the next commit

Branching

How data is stored in git

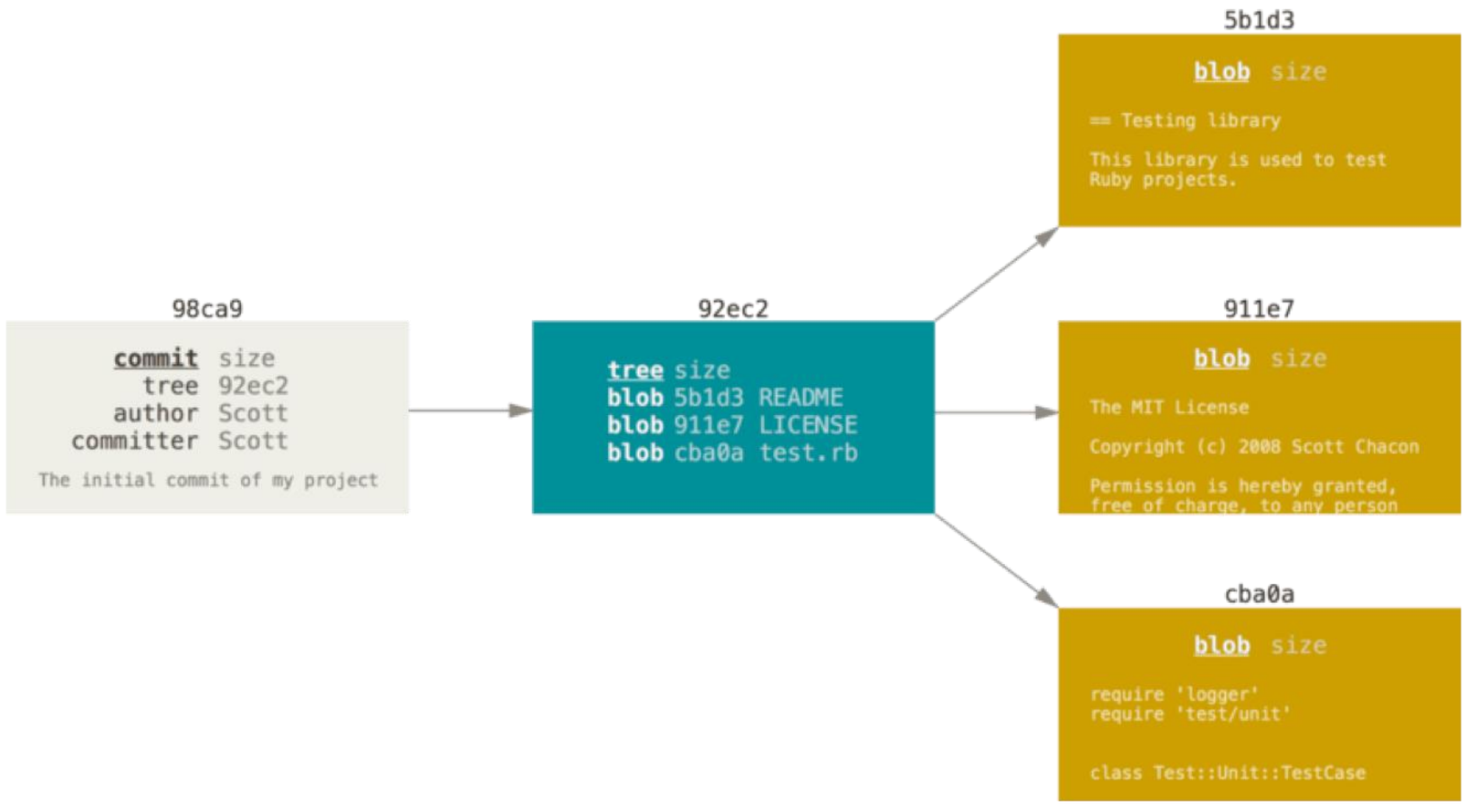


General Process

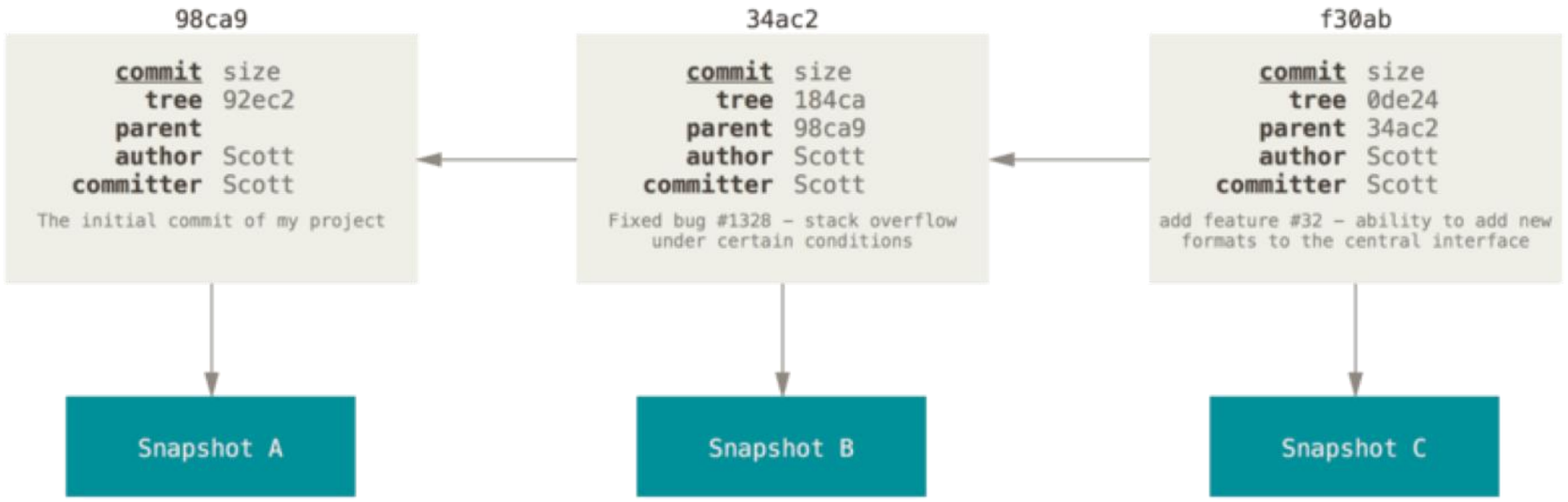
- When committing, git stores a *commit object*
  - contains pointer to the snapshot you staged
  - includes metadata (author, message, etc.)
  - contains also pointer to previous commit(s)
    —> multiple parents in case of previous merge
- on commit, a checksum is computed —> ID of commit

Example: initial commit

```
$ git add README test.rb LICENSE
$ git commit -m 'The initial commit of my project'
```

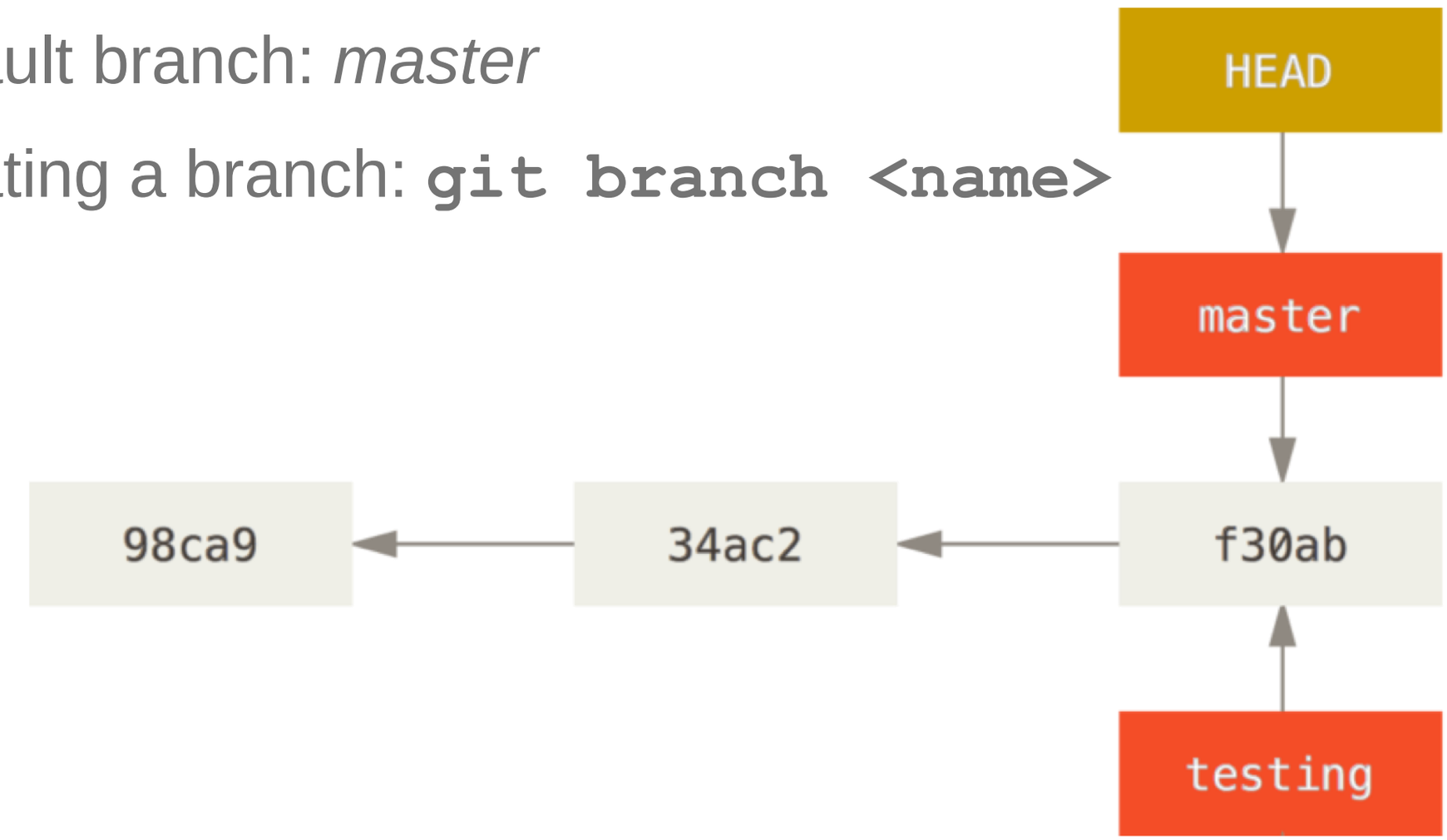


Example: multiple commits



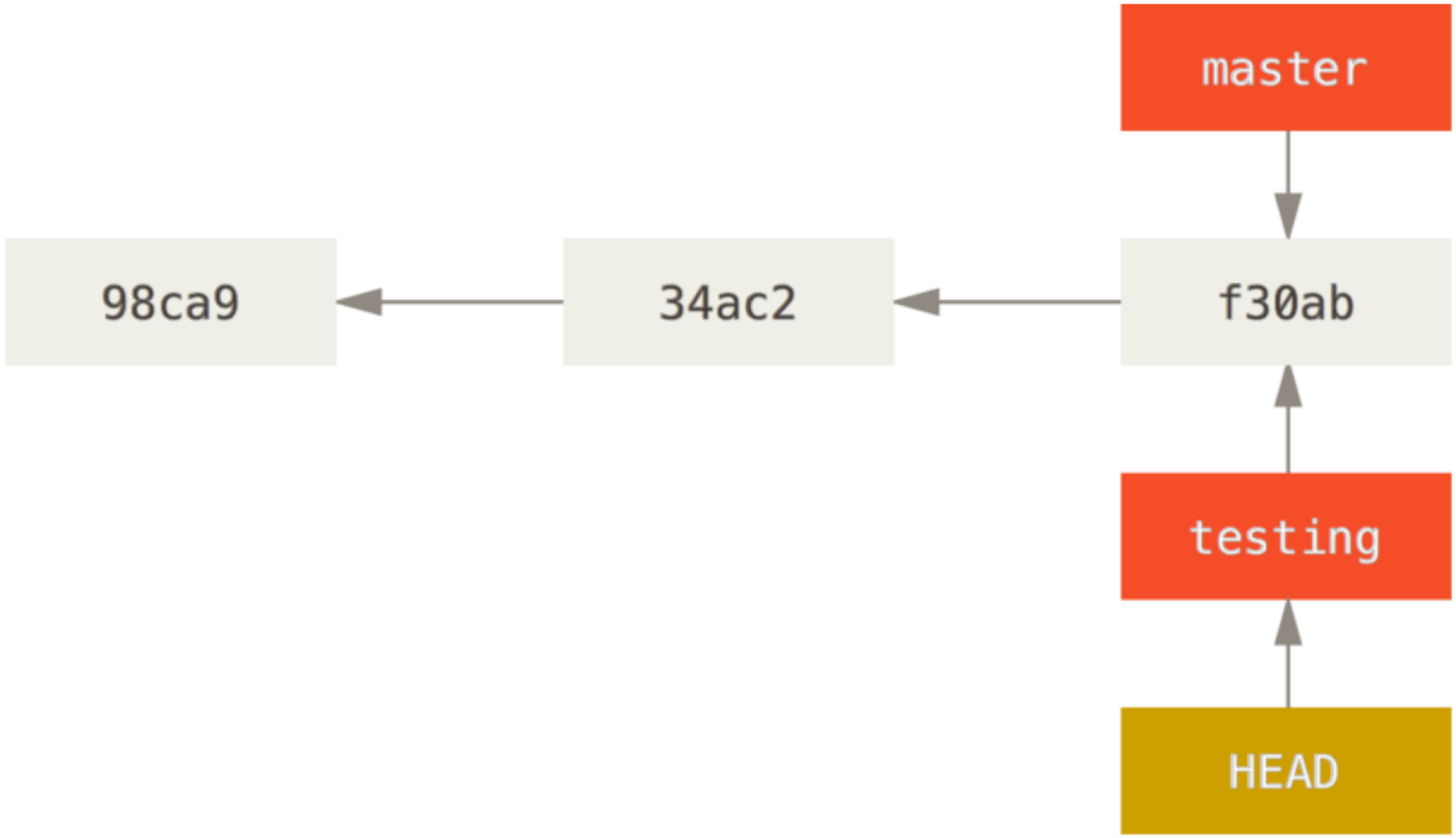
Example: Branching

- A branch is just a movable pointer to one of the previous commits
- default branch: *master*
- creating a branch: **`git branch <name>`**



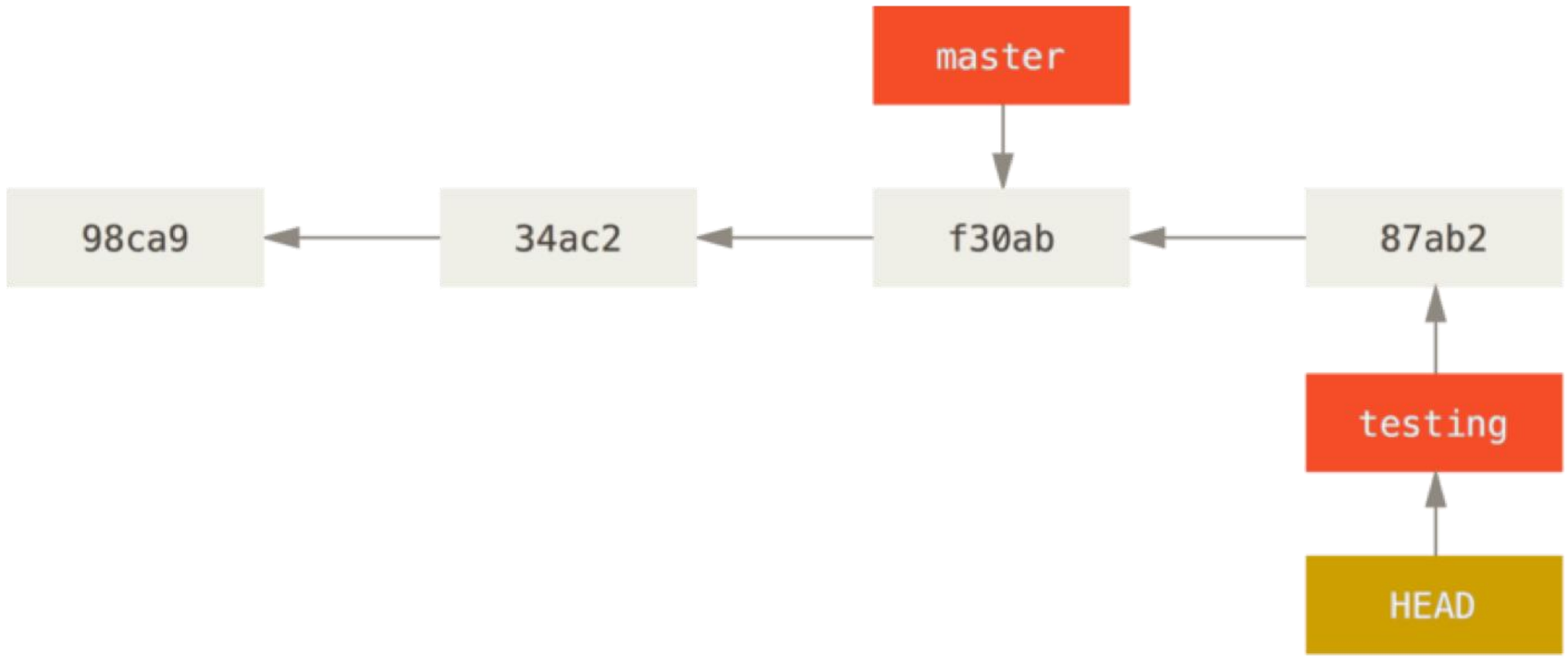
Example: Switch Branches

- switching branches: **`git checkout <name>`**

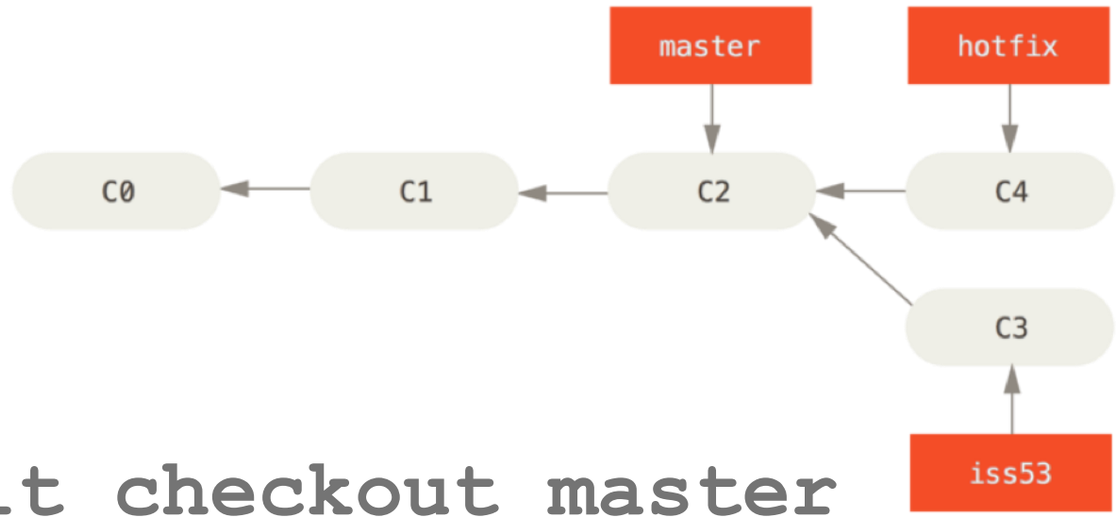


Example: Committing to a Branch

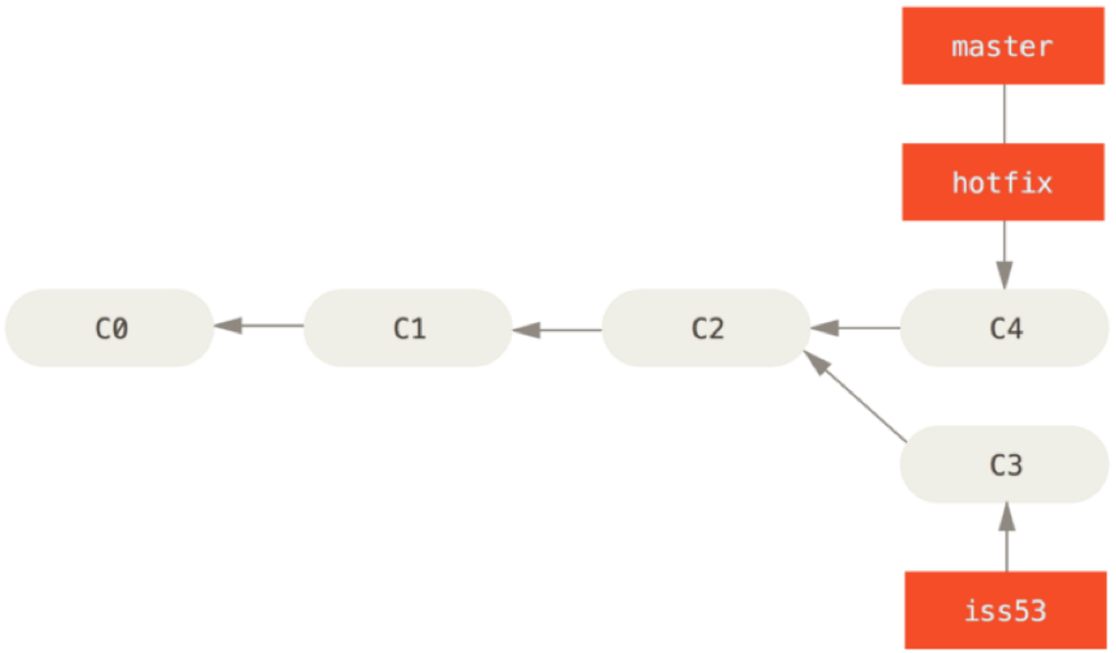
- Assume we change file test.rb and commit the change to the *testing* branch



Unifying Branches — Merging



- change to master branch: **`git checkout master`**
- merge with hotfix branch: **`git merge hotfix`**



Github

A toolkit on top of Git

1,067,856 people hosting over 3,012,331 git repositories

jQuery, reddit, Sparkle, curl, Ruby on Rails, node.js, ClickToFlash, Erlang/OTP, CakePHP, Redis, and many more

Find any repository

twitter facebook rackspace HOSTING digg YAHOO! shopify EMI six apart

git /'gɪt/

Git is an extremely fast, efficient, distributed version control system ideal for the collaborative development of software.

git·hub /'gɪt,hʌb/

GitHub is the best way to collaborate with others. Fork, send pull requests and manage all your **public** and **private** git repositories.

Plans, Pricing and Signup

Unlimited public repositories are **free!**

Free public repositories, collaborator management, issue tracking, wikis, downloads, code review, graphs and much more…

Team management

30 seconds to give people access to code. No SSH key required. Activity feeds keep you updated on progress.

More about collaboration

Code review

Comment on changes, track issues, compare branches, send pull requests and merge forks.

More about code review

Reliable code hosting

We spend all day and night making sure your repositories are **secure**, **backed up** and **always available**.

More about code hosting

Open source collaboration

Participate in the most important open source community in the world today—online or at one of our meetups.

More about our community

1,067,856 people hosting over 3,012,331 git repositories

jQuery, reddit, Sparkle, curl, Ruby on Rails, node.js, ClickToFlash, Erlang/OTP, CakePHP, Redis, and many more

Find any repository

twitter facebook rackspace HOSTING digg YAHOO! shopify EMI six apart

git /'gɪt/

Git is an extremely fast, efficient, distributed version control system ideal for the collaborative development of software.

git·hub /'gɪt,hʌb/

GitHub is the best way to collaborate with others. Fork, send pull requests and manage all your **public** and **private** git repositories.

Plans, Pricing and Signup

Unlimited public repositories are **free!**

Free public repositories, collaborator management, issue tracking, wikis, downloads, code review, graphs and much more…

Team management

30 seconds to give people access to code. No SSH key required. Activity feeds keep you updated on progress.

More about collaboration

Code review

Comment on changes, track issues, compare branches, send pull requests and merge forks.

More about code review

Reliable code hosting

We spend all day and night making sure your repositories are **secure**, **backed up** and **always available**.

More about code hosting

Open source collaboration

Participate in the most important open source community in the world today—online or at one of our meetups.

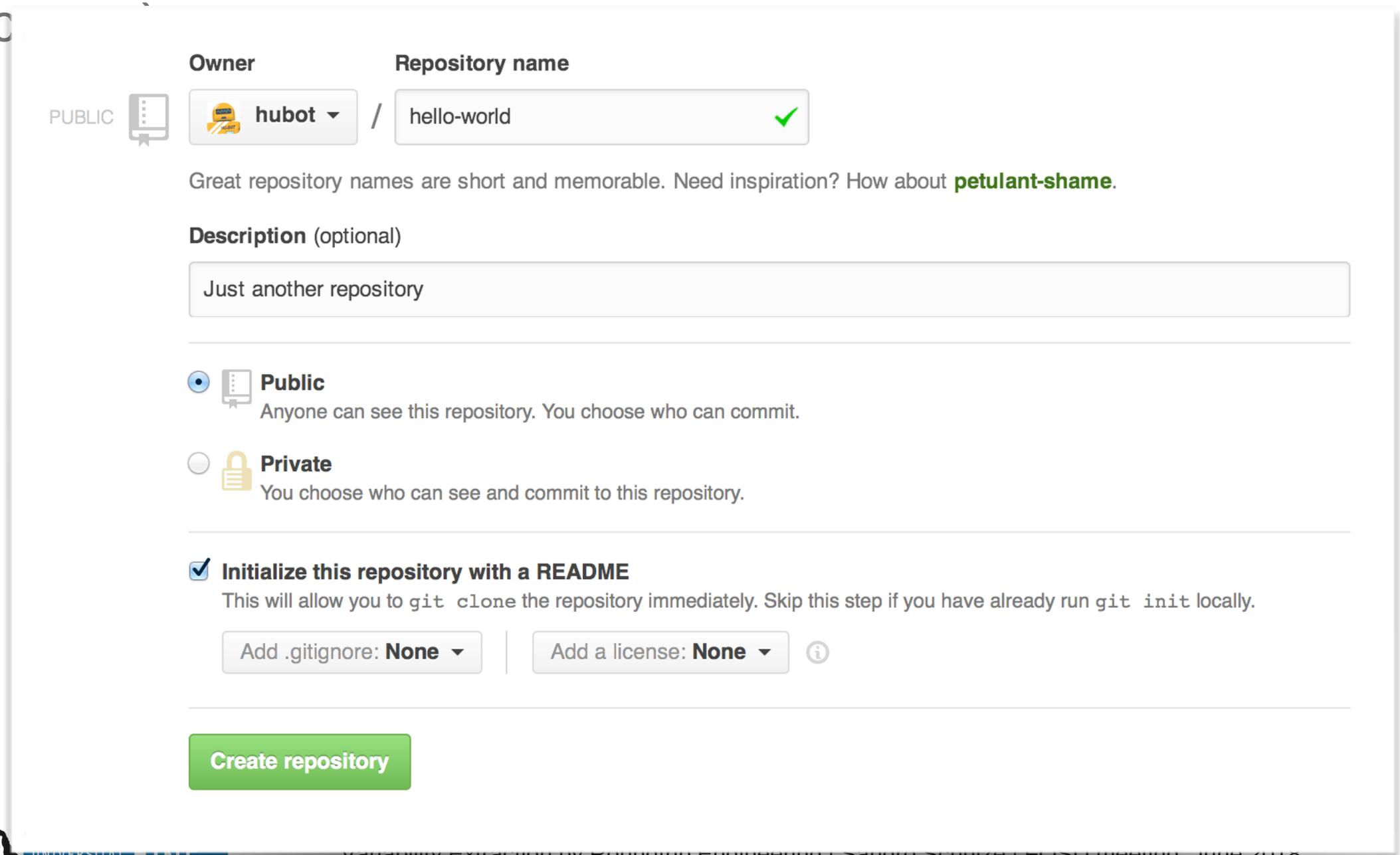
More about our community

Overview

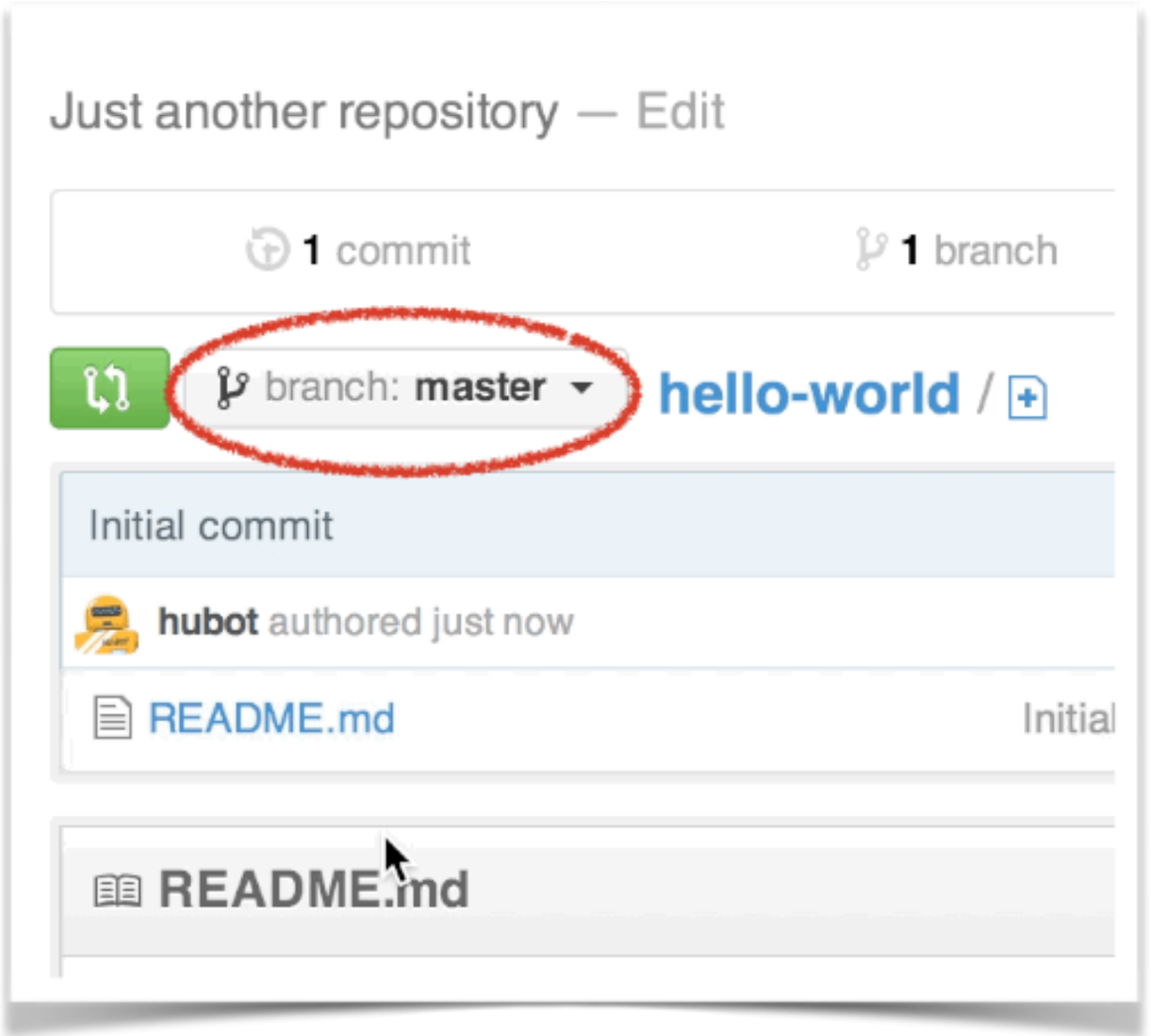
- Github provides remote hosting of git repositories
- Beyond that, they provide rich, web-based tool support for collaborative development
  - forking
  - issue tracking
  - pull requests
  - markdown-supported wiki, github page etc.
- provide integration with additional services, e.g., Continuous Integration

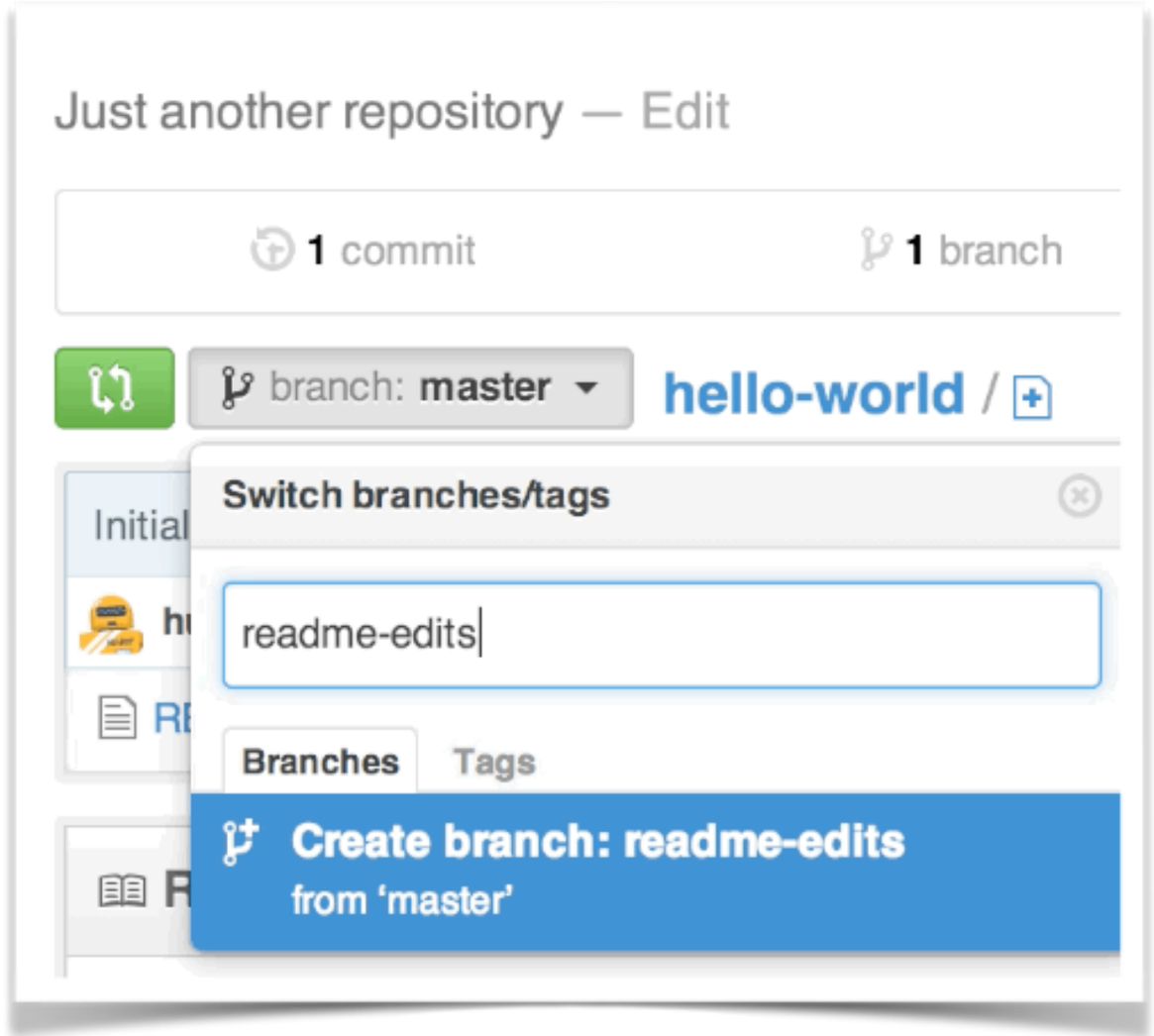
Create a Repository

- Initiate a new repo by clicking “new repository” (upper right c



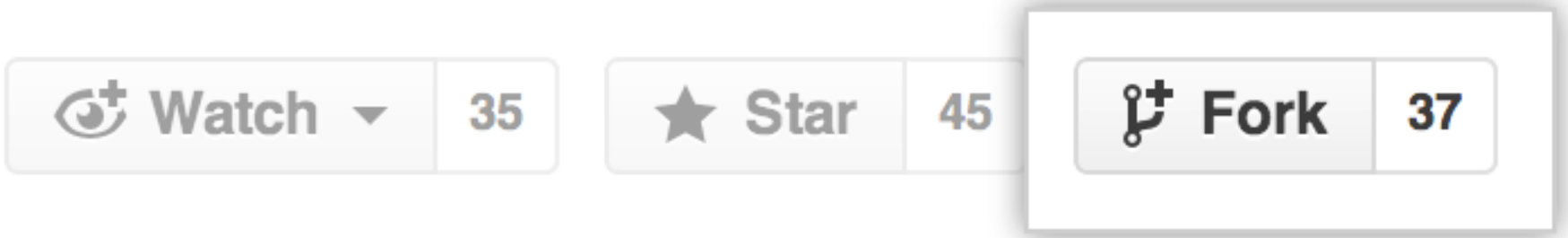
Branching

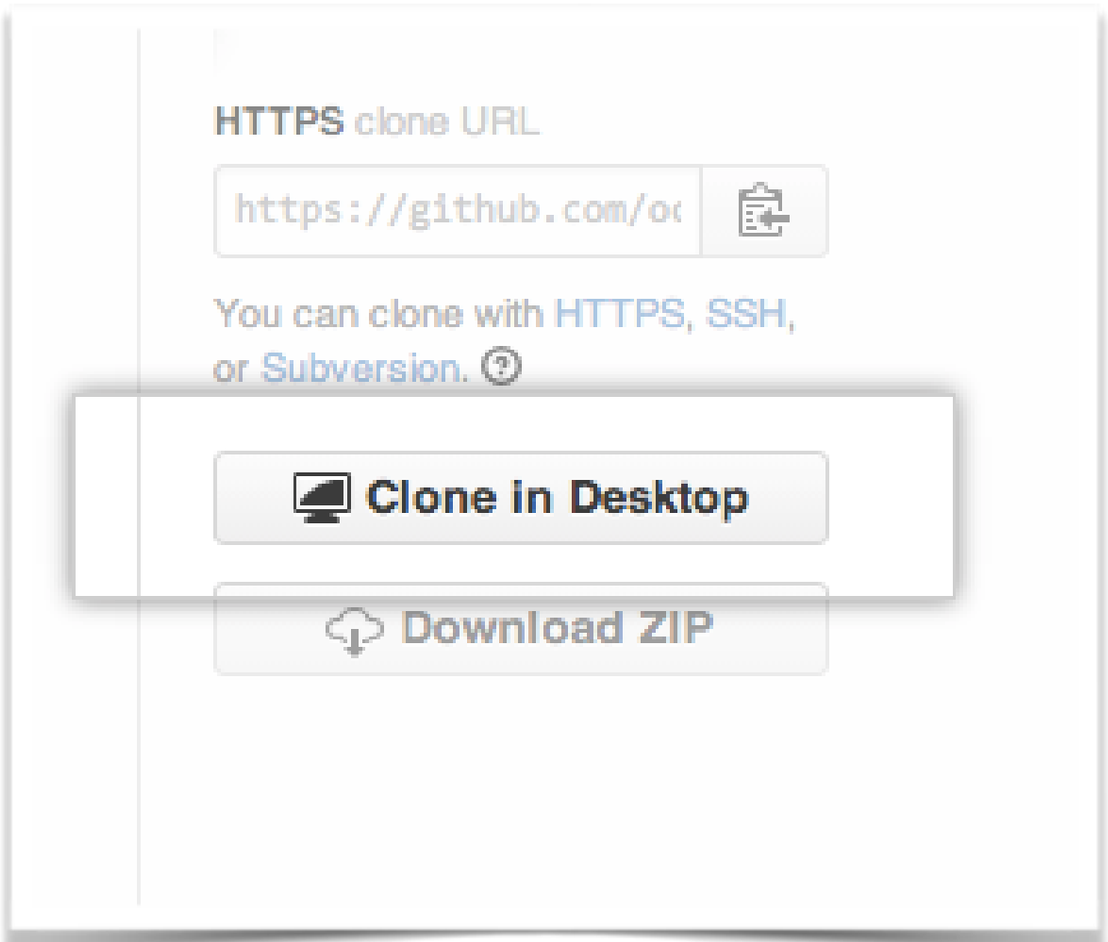

Just another repository — Edit
1 commit
1 branch
branch: master
hello-world /
Initial commit
hubot authored just now
README.md
Initial
README.md


Just another repository — Edit
1 commit
1 branch
branch: master
hello-world /
Switch branches/tags
readme-edits
Branches
Tags
Create branch: readme-edits
from 'master'

Forking

- Afte
- com

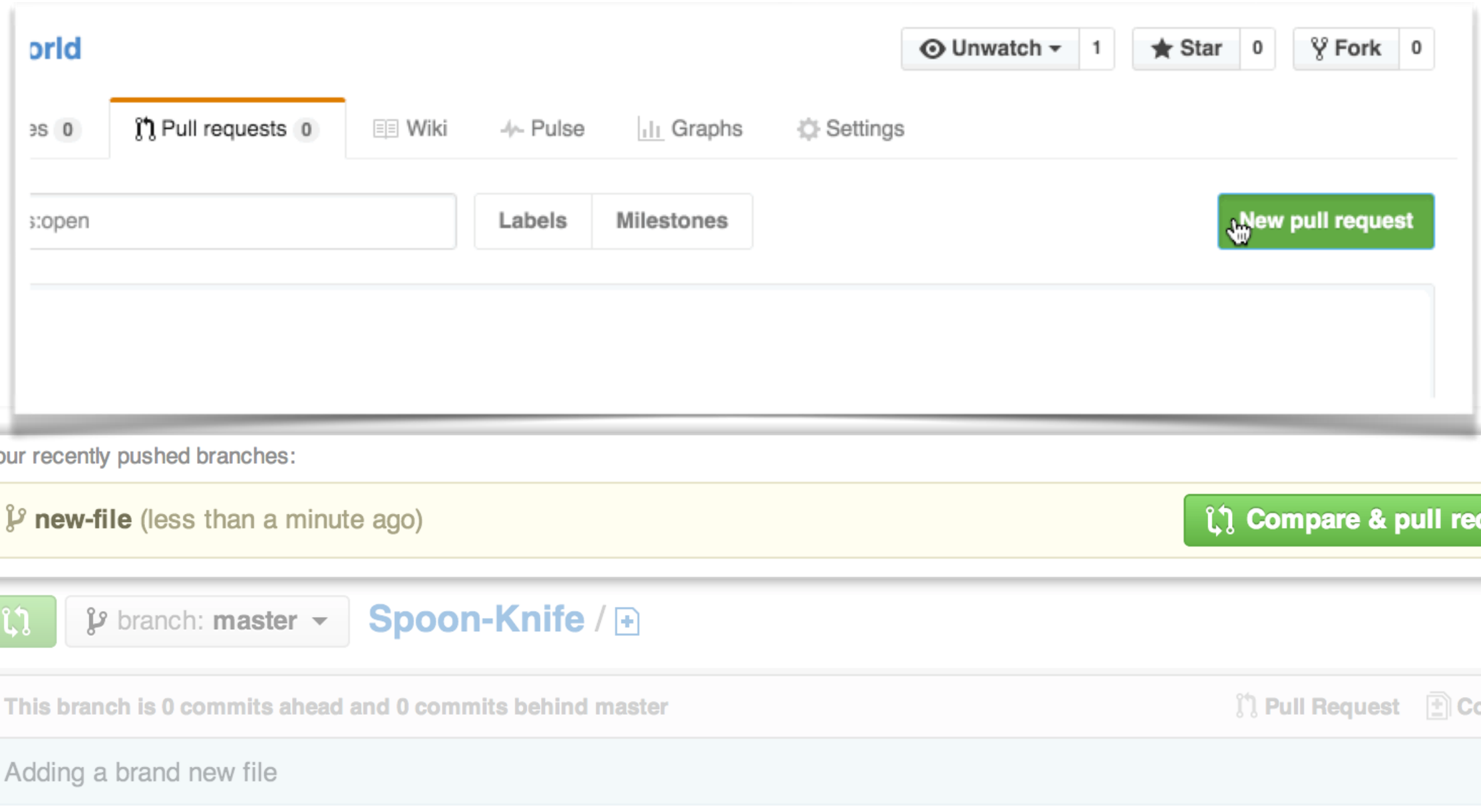




Pull Request

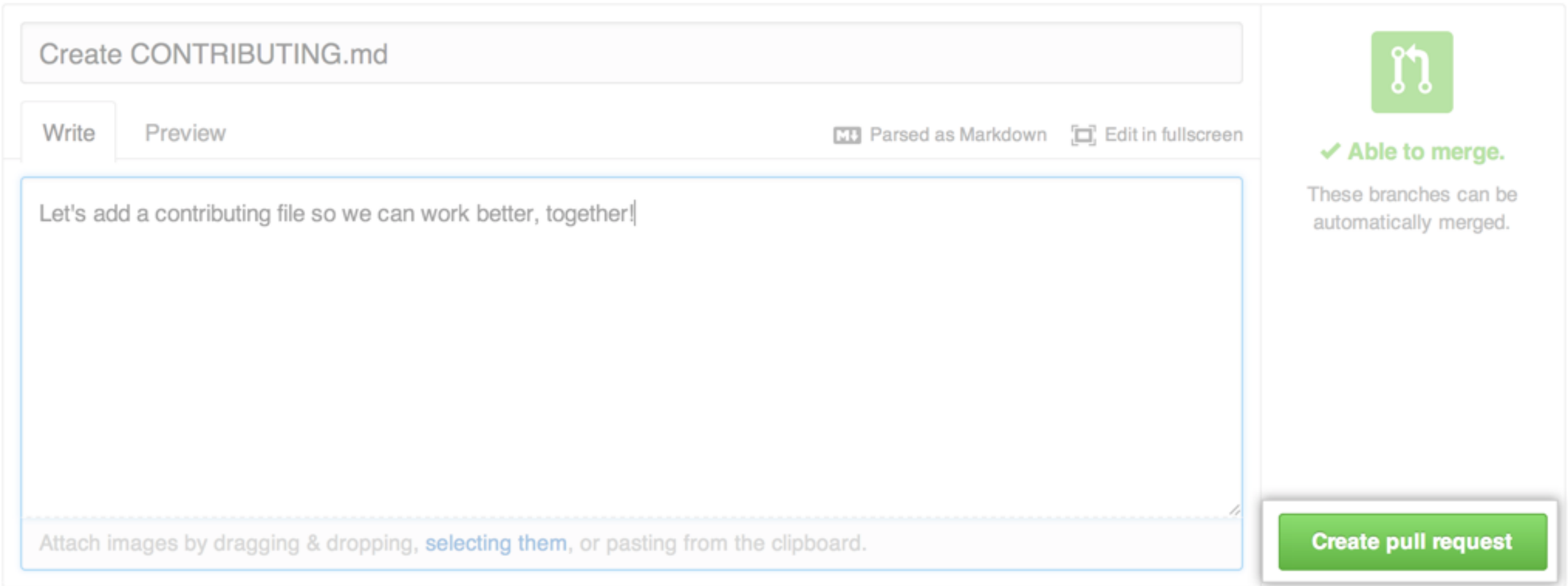
- Getting the changes of forks in the main project is the highest benefit of distributed development/social coding
- Challenge: how to get changes from (possibly) hundreds of forks back to the main project without sacrificing stability?
- Pull Requests provide a structured workflow for doing so
- is also applicable between branches of the same fork!

Creating a Pull Request

- 

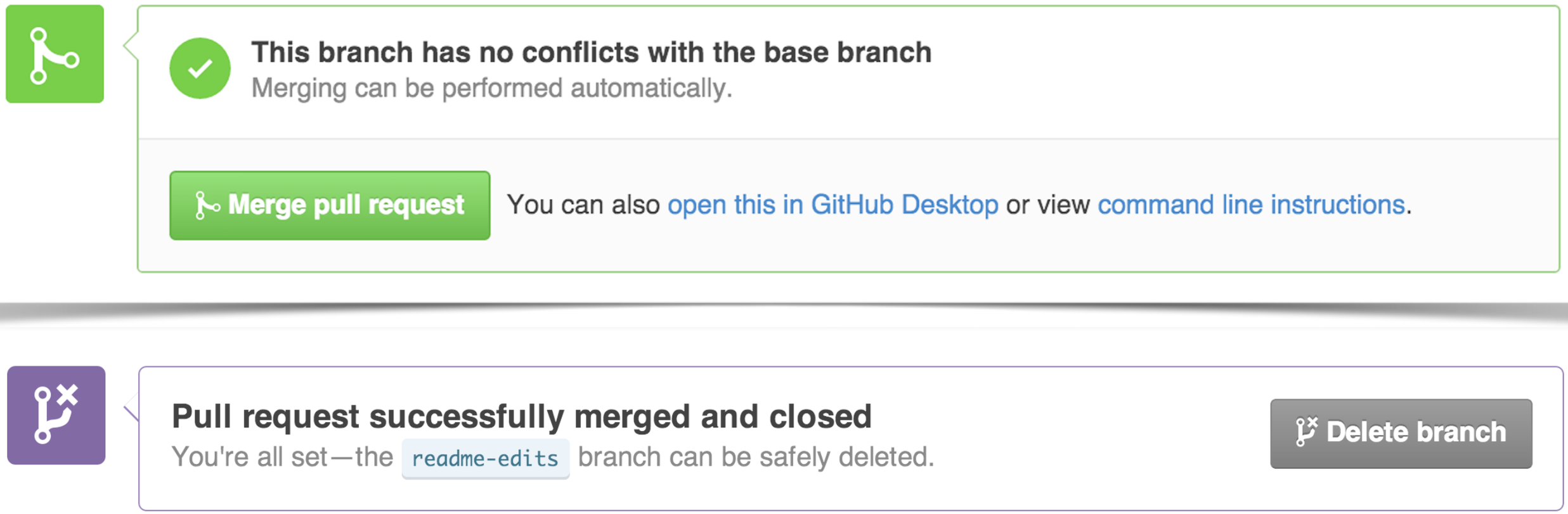
Creating a Pull Request

- description uses markdown: you can do formatting, adding pics etc. to make clear WHY you make this pull request
- Use @mention to get feedback from specific persons



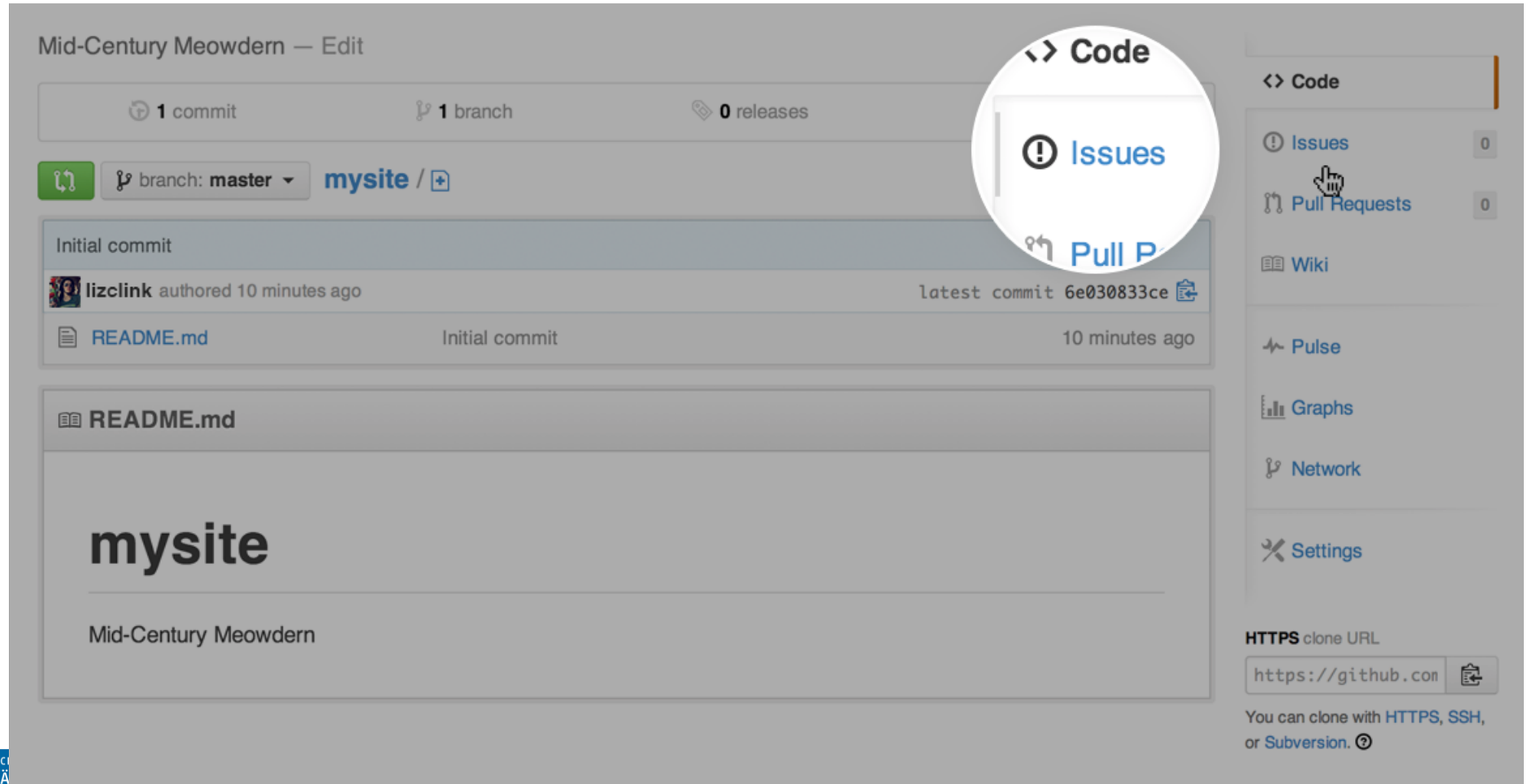
Merging your PR

- This is usually done by project owner —> PRs between fork and main project need to be merged by owners of the original repo

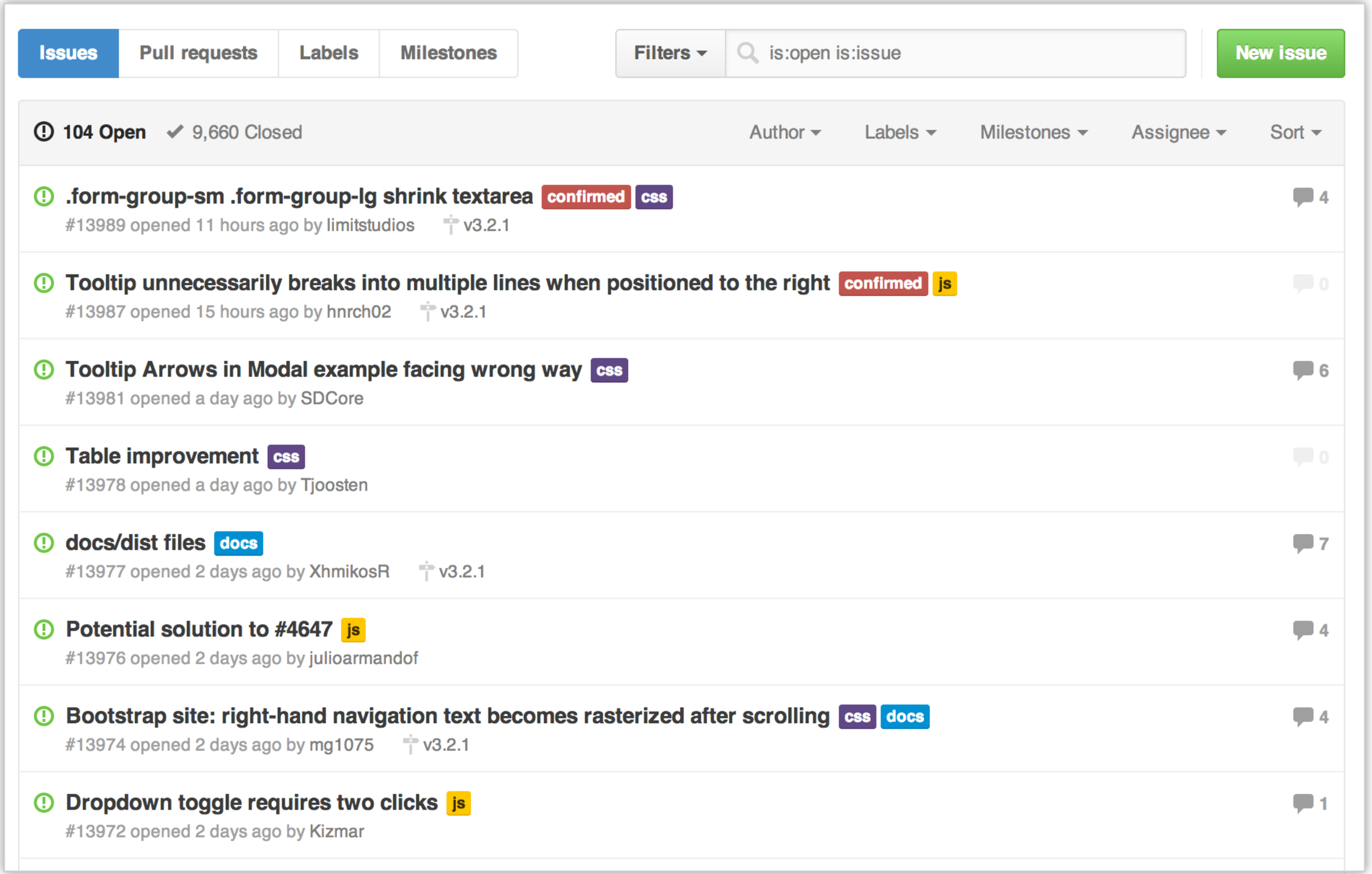


Issue tracking with Github

- best way to keep track of any progress
  —> tasks, features, enhancements, bugs
- you may think of using it to structure and document your project management



Example: A project using Github's Issues



A Typical Issue — Overview

The no-conflict mode should be the default behaviour #12395

Edit | New issue

Open **thewebdreamer** opened this issue 3 days ago · 10 comments

thewebdreamer commented 3 days ago

The no-conflict mode should be the default behaviour. Why would a Bootstrap client need to implement this?

cvrebert commented 3 days ago

I believe no-conflict-is-not-the-default is the norm for jQuery plugins?

thewebdreamer commented 3 days ago

It is true that it is the norm for jQuery plugins.

Couldn't there be a clash with other jQuery plugins with the current implementation of Bootstrap though?

Labels

js

Milestone

No milestone

Assignee

No one assigned

Notifications

Subscribe

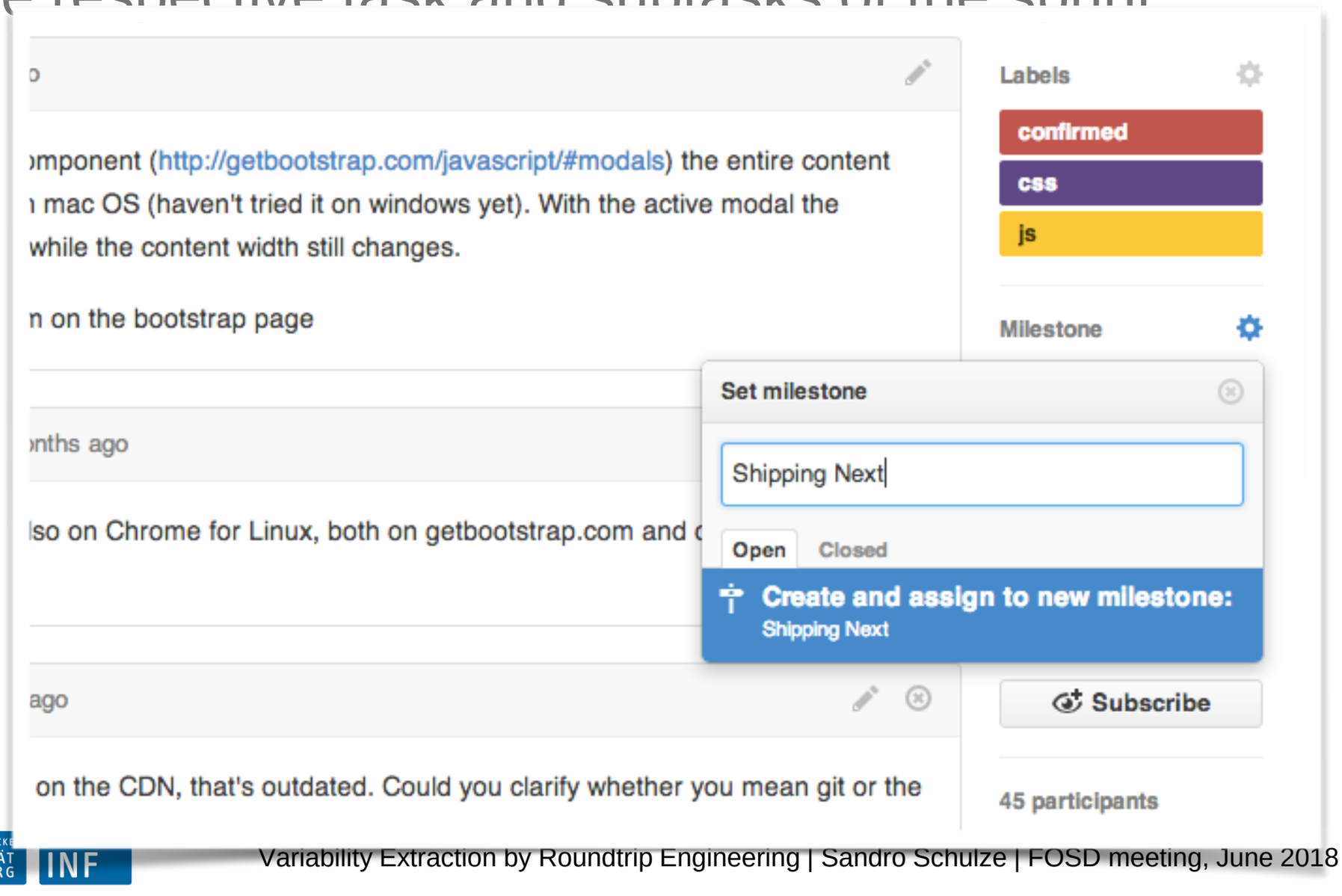
3 participants

Milestones, Labels, and Assignees

- after a while, you may get lost in all your issues
- Use milestones and labels to manage and structure them
- Moreover, assignees are useful to directly assign seem work to be done to a team member (e.g., to track who is working on what)
- add or change all of them on the right side of your issue

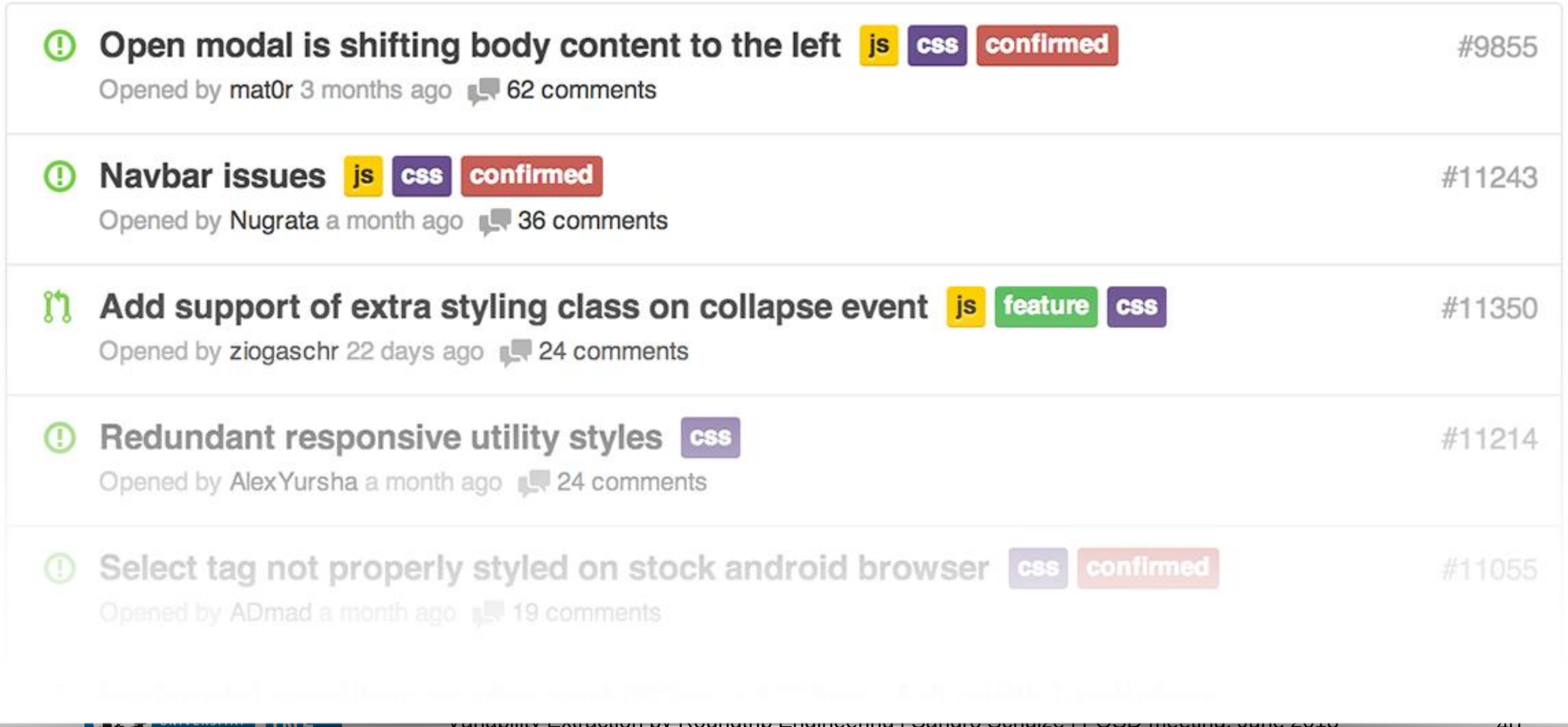
Milestones

- Milestones could be used to manage your sprints
  —> one milestone per sprint, afterwards you can open issues for the respective task and subtasks of the sprint



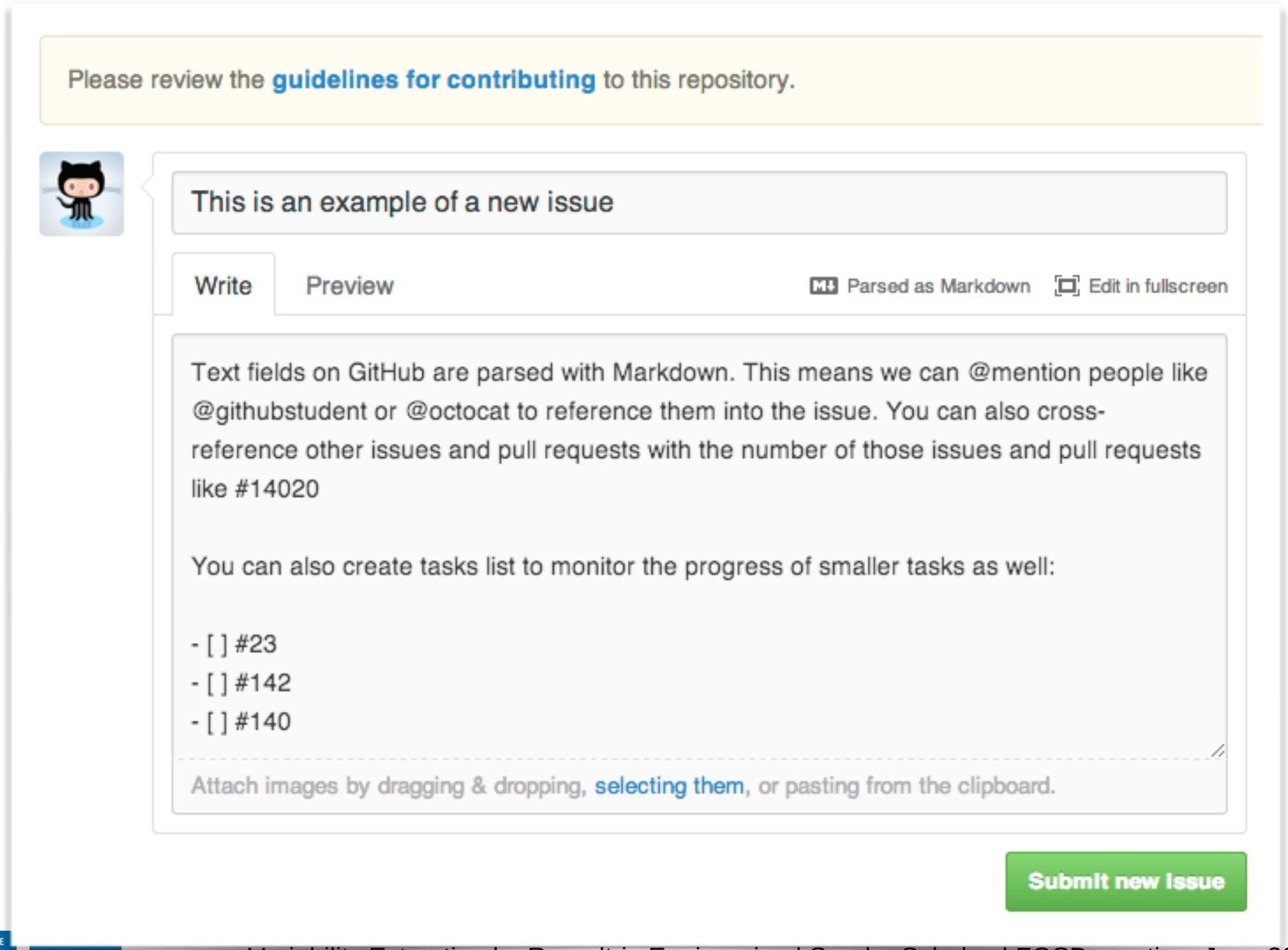
Labels

- Use labels to organise different types of issues

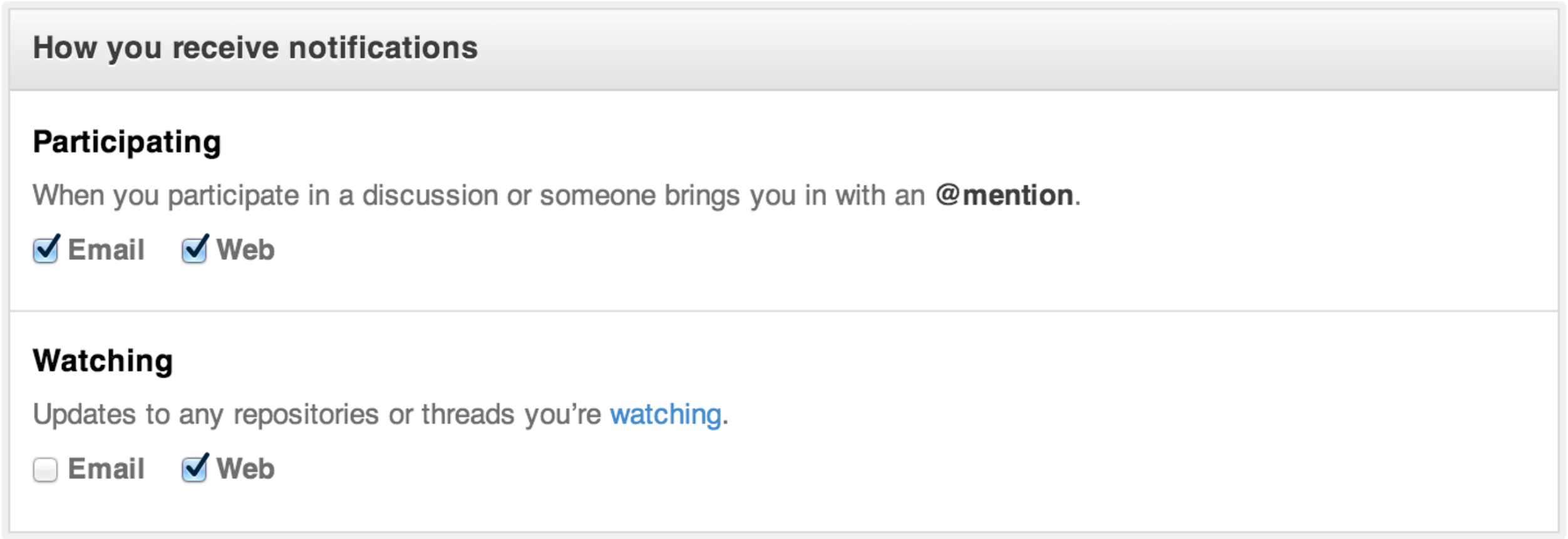


Make use of Markdown facilities

- @mention, references, but also text formatting

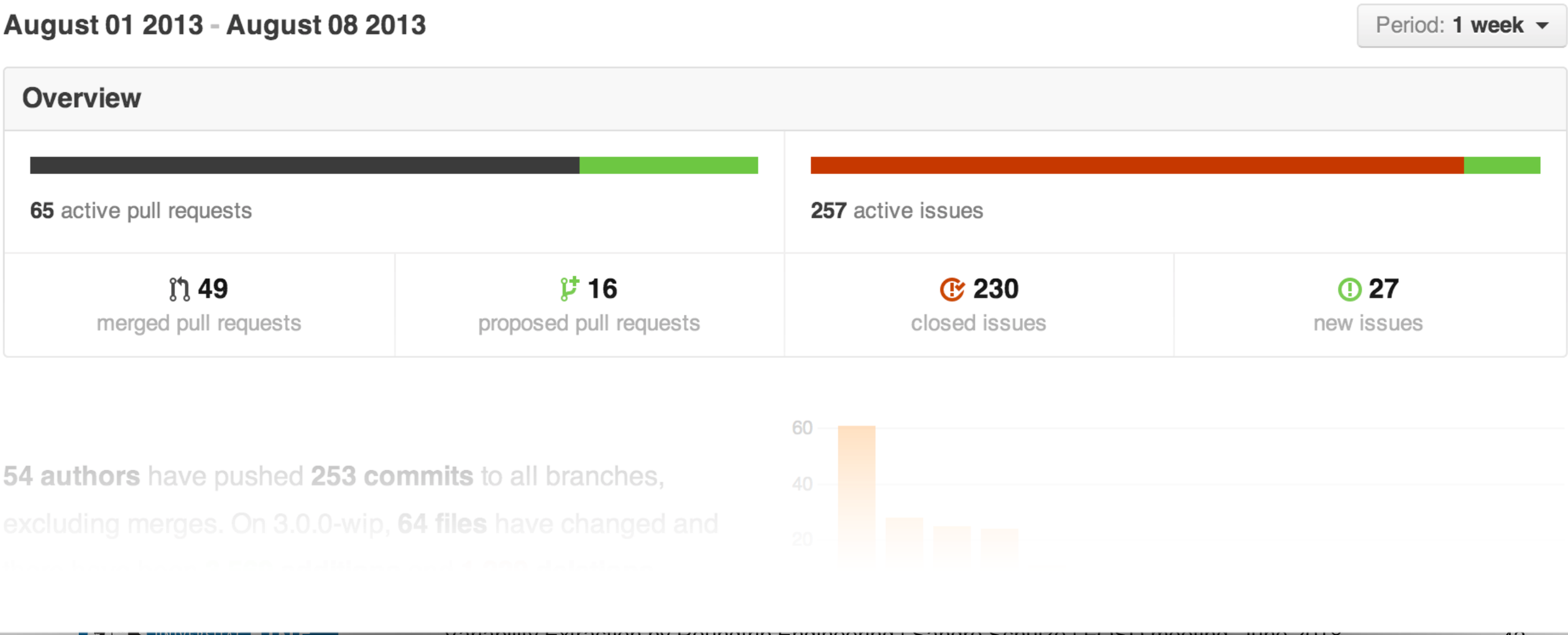


Never miss a change to Issues



The Big picture

- Make use of Issue Dashboard
- Check the Pulse feature on Github



More on this

---

https://guides.github.com/

Greetings fr(