

# **GITHUB FOR COLLABORATIVE PROJECTS**

**CODE VERSIONING**

**GIT FLOW**

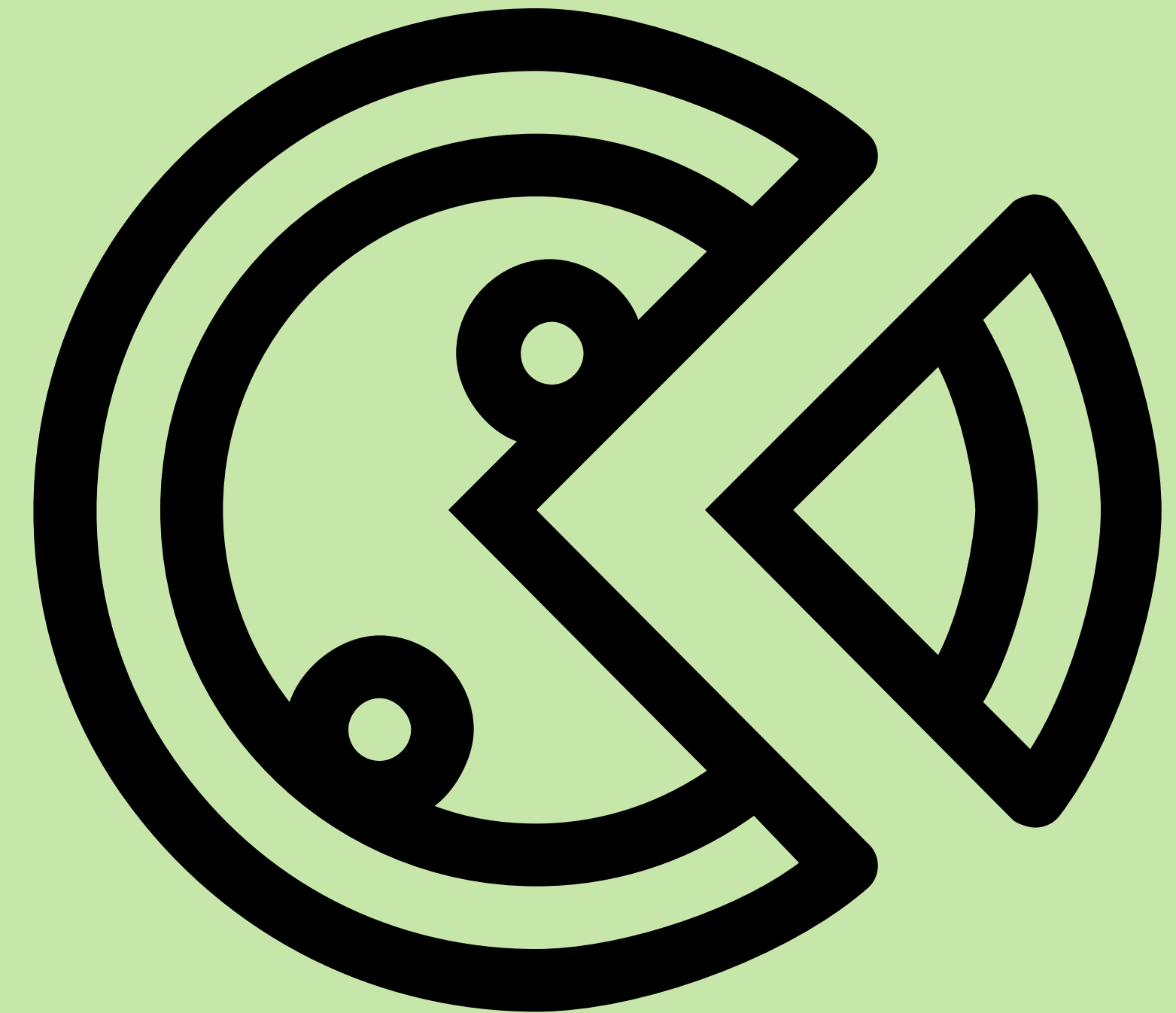
**GITHUB ISSUES**

**PROJECT MANAGEMENT**

# CODE VERSIONING

# Why do we Need Code Versioning?

- Lets say you want to make a pizza
- You make the dough and go out for the day
- When you come back the pizza is ready to go in the oven (roommate made it)
  - You like your pizza spicy so put extra spicy sauce on top
- Pizza goes in the oven and you go out, leaving a note for roommate to take it out the oven
- You come back to...
  - Burn pizza
  - A pizza that is as hot as the sun



# What is a Version Control System?

- A version control system, or VCS, tracks the history of changes as people and teams collaborate on projects together.
- As the project evolves, teams can run tests, fix bugs, and contribute new code with the confidence that any version can be recovered at any time.
- Developers can review project history to find out:
  - Which changes were made?
  - Who made the changes?
  - When were the changes made?
  - Why were changes needed?

# What is a Distributed Version Control System?

- Git is an example of a distributed version control system (DVCS) commonly used for open source and commercial software development.
- DVCSs allow full access to every file, branch, and iteration of a project, and allows every user access to a full and self-contained history of all changes.
- Unlike once popular centralized version control systems, DVCSs like Git don't need a constant connection to a central repository.
- Developers can work anywhere and collaborate asynchronously from any time zone

# VCS Terminology

- A **repository**, or Git project, encompasses the entire collection of files and folders associated with a project, along with each file's revision history.
- The file history appears as snapshots in time called **commits**.
- The commits exist as a linked-list relationship, and can be organised into multiple lines of development called **branches**.

# Anatomy of a Repository





# Order of Operations

FIRST WE STAGE THE  
MODIFICATION THAT  
WE'VE MADE

**git** ADD

THEN WE COMMIT THE  
MODIFICATIONS TO THE  
REPOSITORY

**git** COMMIT

# Being Selective with GIT

THINGS CAN BE ADDED ACROSS SEVERAL FILES IN  
THE 1 COMMIT

```
git commit -m "New eMail Address Added"
```



# Committing items logically

- ✓ `git commit -m "Typo in readme.md"`
- ✓ `git commit -m "New layout for homepage"`
- ✗ `git commit -m "Type in readme.md, new layout for homepage, told Mike how awesome he is"`

IF A FEATURE GETS ROLLED BACK THEN YOU RE-INTRODUCE THE TYPO!

# Where does GitHub fit into this?

- GitHub is a Git hosting repository that provides developers with tools to ship better code through:
  - Issues (threaded discussions),
  - Pull requests,
  - Code review,
  - Project management
- GitHub builds collaboration directly into the development process.
  - Work is organized into repositories, where developers can outline requirements or direction and set expectations for team members.
  - Developers create a branch to work on updates, commit changes to save them, open a pull request to propose and discuss changes, and merge pull requests once everyone is on the same page.



## **Creating a GitHub Repository and adding Collaborators**

For these activities you are going to be working with the people around your table

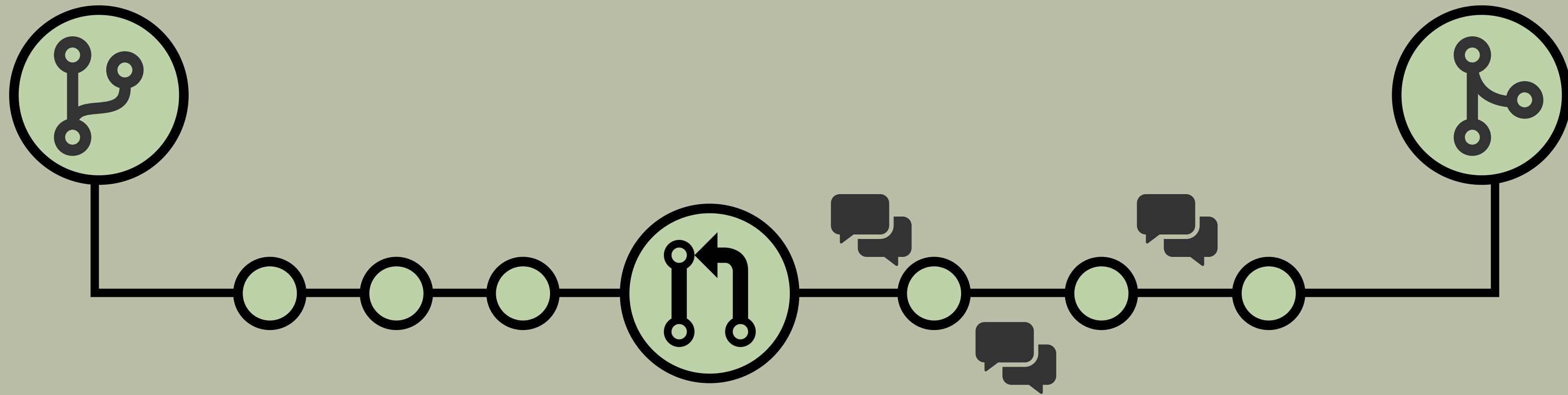
1. Sign up for a gitHub account if you haven't already created one
2. Get one person at your table to create a new repository
3. Add everyone else at the table as a collaborator to this repository

**Remember and turn your big screens to a single person so everyone can see what is happening for step 2 and 3.**

# **GIT FLOW**

# Git & GitHub

## The GitHub Workflow

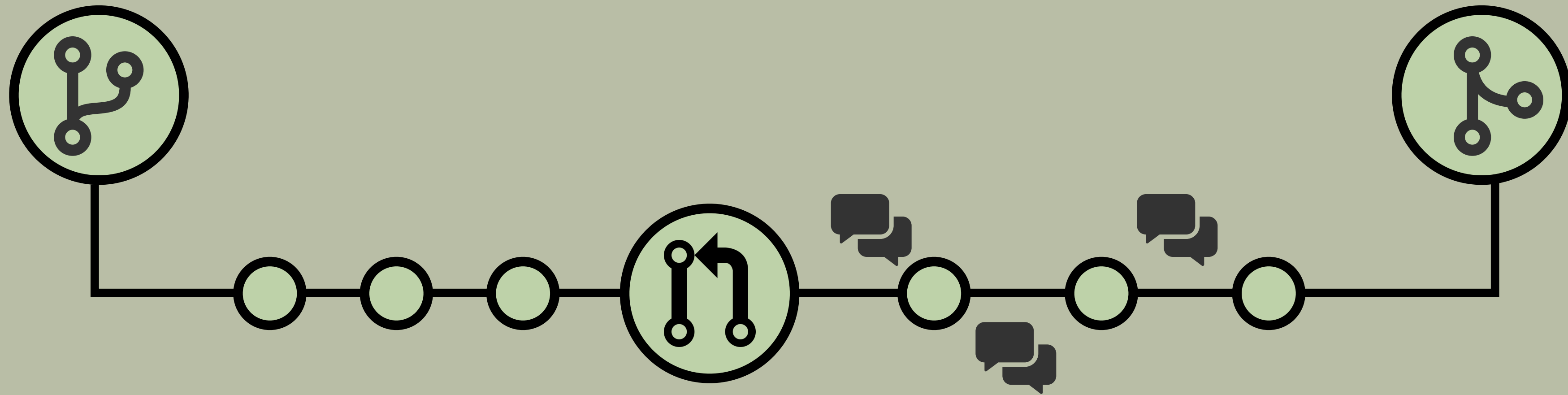


1. *Journal of Management Studies*, 1996, 33, 1, 1-14.

# Create a branch

- Branching is a core concept in Git, and the entire GitHub flow is based upon it.
- There's only one rule: anything in the master branch is always deployable.





# 2

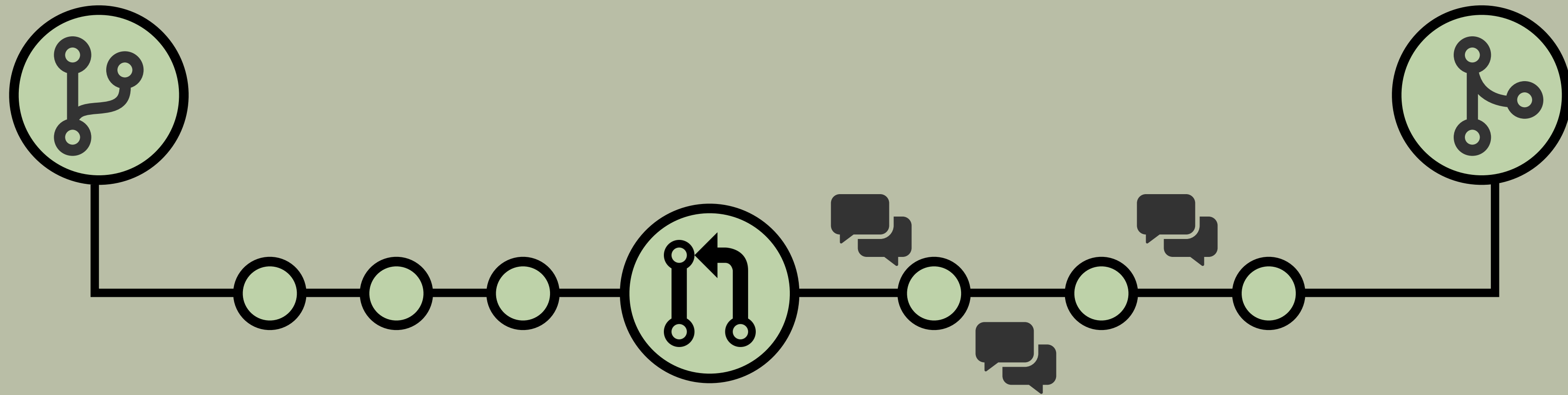
## Add commits

- Whenever you add, edit, or delete a file, you're making a commit, and adding them to your branch.
- Commit messages are important. By writing clear commit messages, you can make it easier for other people to follow along and provide feedback



**Creating a .md file  
for all group  
members**

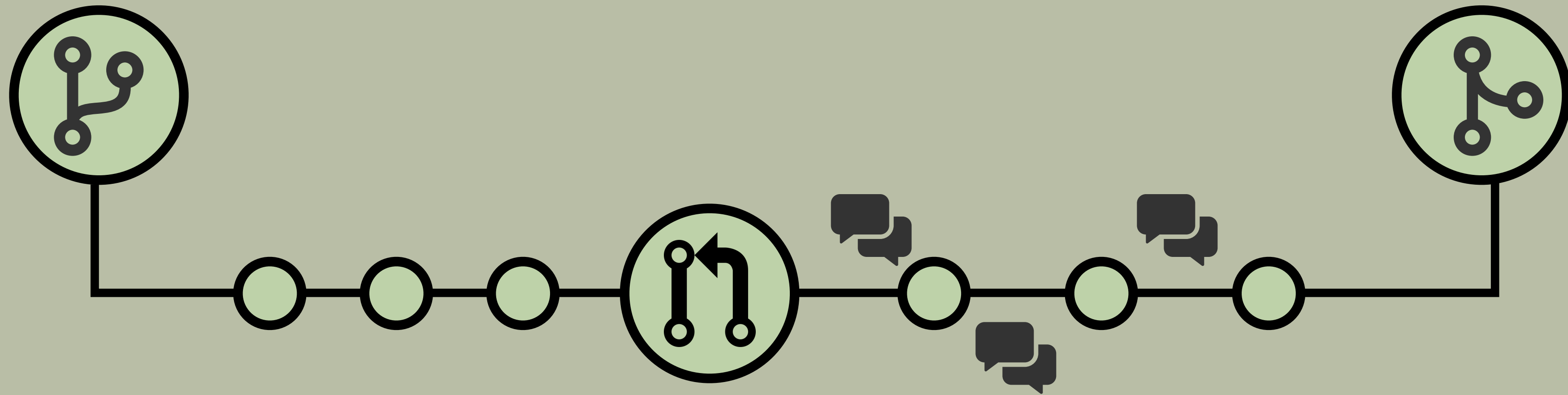
1. Get your project working locally via atom
  - a. Open ATOM
  - b. Command Pallete Cmd/Ctrl+Shift+P
  - c. Git Clone
2. Create a new branch
3. Create a file in this branch that contains:
  - a. Your name
  - b. Your favourite pizza topping
  - c. 1 person in your team should lie about their pizza topping...work out who between your group
4. Upload this branch to gitHub



# 3

## Create a Pull Request

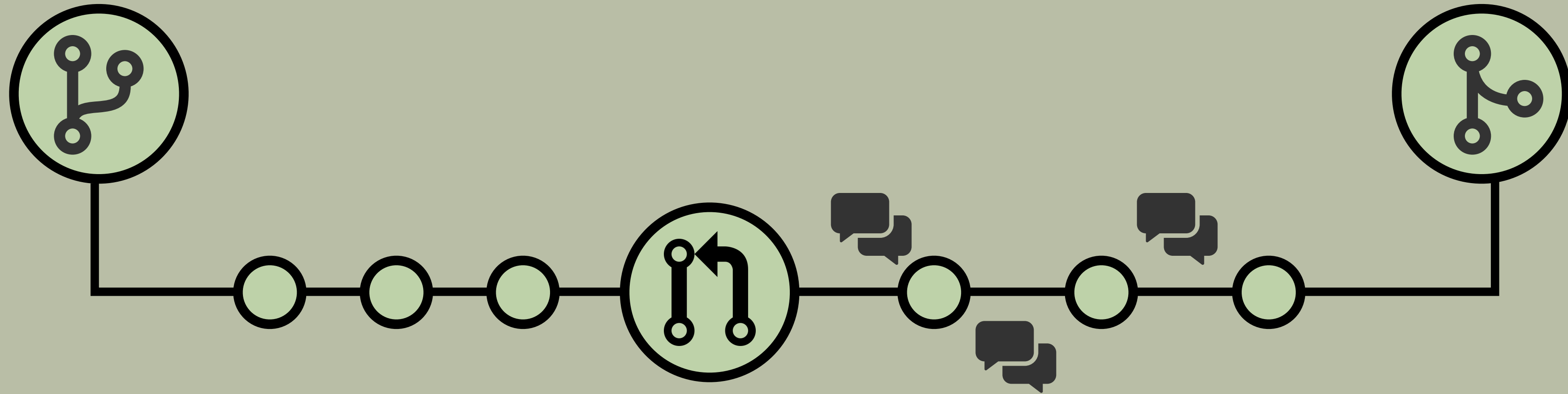
- Pull Requests provide a way to notify project maintainers about the changes you'd like them to consider.
- Pull Requests initiate discussion about your commits.



# 4

## Discuss and Review Code

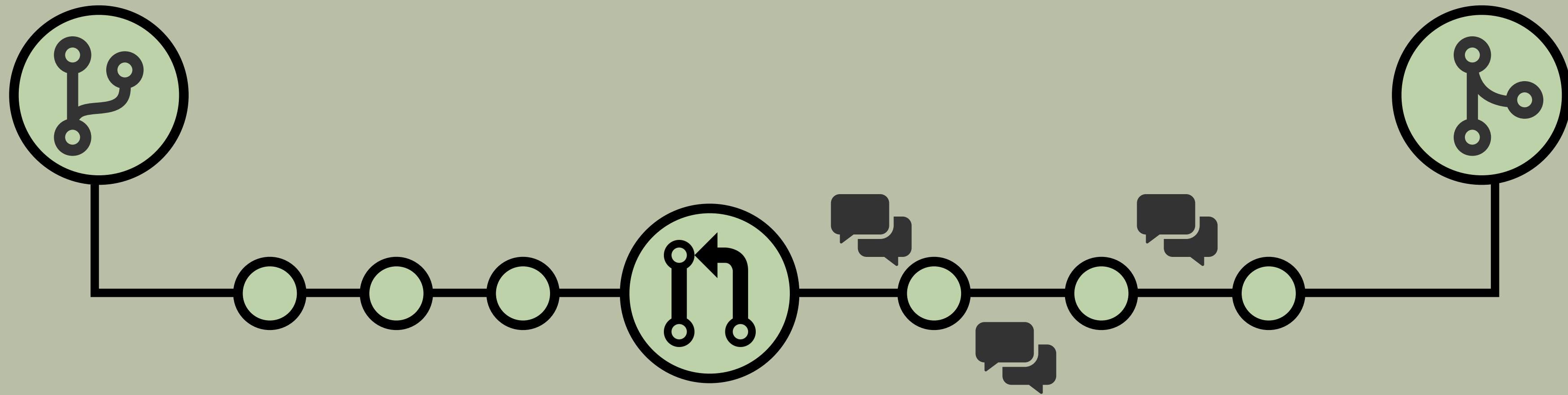
- Once a Pull Request has been opened, the person or team reviewing your changes may have questions or comments
- You can also continue to push to your branch in light of discussion and feedback about your commits.



# 5

## Deploy

- Not used in this module, but important to check that your code meets pre-determined tests and won't break everything once it is launched



# 6

## Merge

- Now that your changes have been verified in production, it is time to merge your code into the master branch.
- Once merged, Pull Requests preserve a record of the historical changes to your code. Because they're searchable, they let anyone go back in time to understand why and how a decision was made.



## **Creating Pull Requests and Discussing**

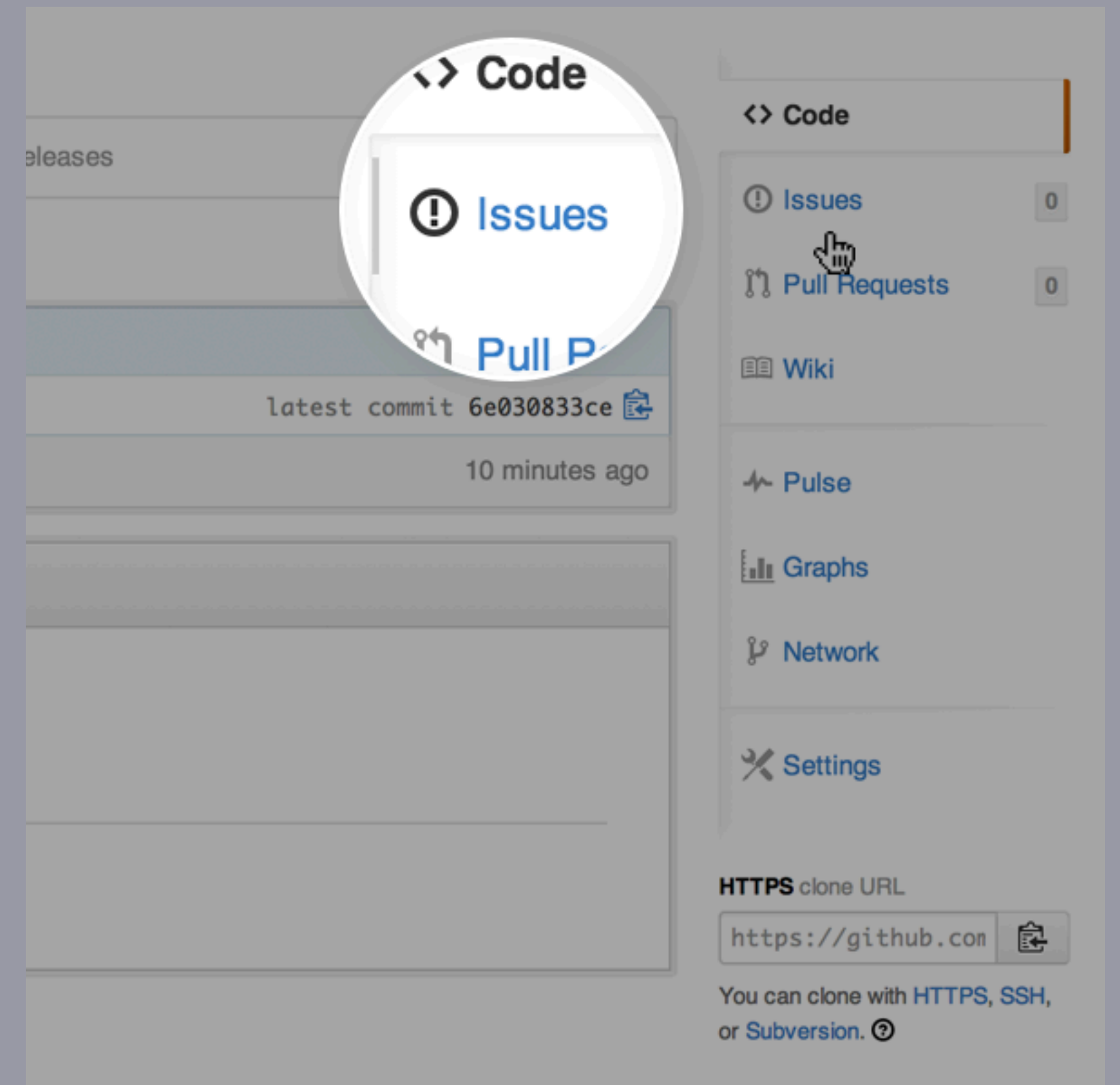
1. On the GitHub website create a pull request for your branch
  1. You might want to use a pull request template...
  2. [https://raw.githubusercontent.com/embeddedartistry/templates/master/oss\\_docs/PULL\\_REQUEST\\_TEMPLATE.md](https://raw.githubusercontent.com/embeddedartistry/templates/master/oss_docs/PULL_REQUEST_TEMPLATE.md)
2. Discuss your pull request within your group
3. Merge in the request when you (as a group) are happy with this

# GITHUB ISSUES



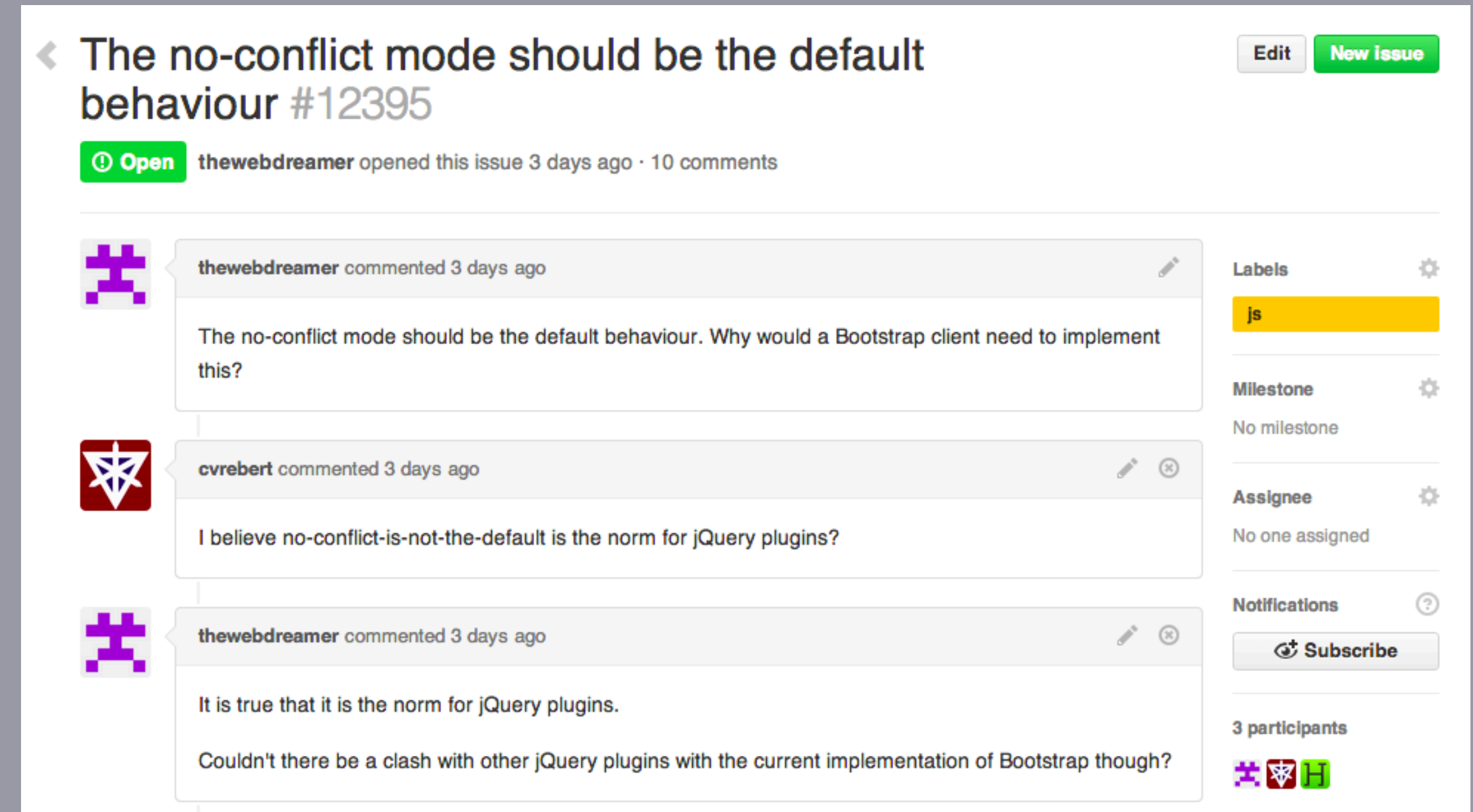
# Using Issues in GitHub

- Issues are a great way to keep track of tasks, enhancements, and bugs for your projects.
- They're kind of like email—except they can be shared and discussed with the rest of your team.
- Most software projects have a bug tracker of some kind.
- GitHub's tracker is called Issues, and has its own section in every repository.



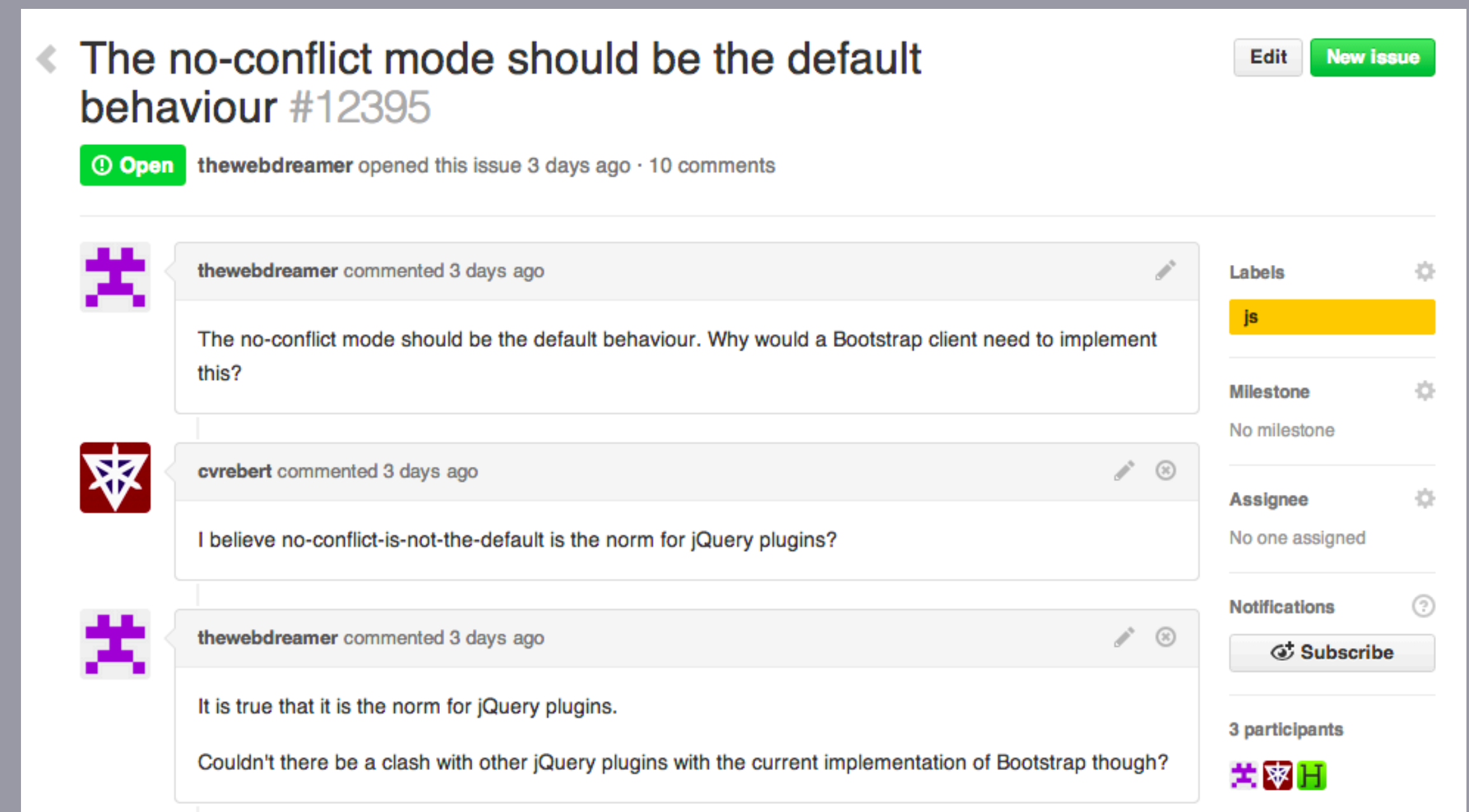
# What does an Issue look like?

- A title and description describe what the issue is all about.
- Colour-coded labels help you categorise and filter your issues (just like labels in email).
- A milestone acts like a container for issues. This is useful for associating issues with specific features or project phases (e.g. Weekly Sprint 9/5-9/16 or Shipping 1.0).
- One assignee is responsible for working on the issue at any given time.
- Comments allow anyone with access to the repository to provide feedback.














# Issues and Milestones

- Once you've collected a lot of issues, you may find it hard to find the ones you care about. Milestones, labels, and assignees are great features to filter and categorize issues.
- You can change or add a milestone, an assignee, and labels by clicking their corresponding gears in the sidebar on the right.
- Milestones are groups of issues that correspond to a project, feature, or time period.
  - Design Stage Complete
  - Beta Release
  - Final Release



# Labels

- Labels are a great way to organize different types of issues. Issues can have as many labels as you want, and you can filter by one or many labels at once.

	<b>Open modal is shifting body content to the left</b> <span>js</span> <span>css</span> <span>confirmed</span>	#9855
Opened by mat0r 3 months ago  62 comments		
	<b>Navbar issues</b> <span>js</span> <span>css</span> <span>confirmed</span>	#11243
Opened by Nugrata a month ago  36 comments		
	<b>Add support of extra styling class on collapse event</b> <span>js</span> <span>feature</span> <span>css</span>	#11350
Opened by zio gaschr 22 days ago  24 comments		
	<b>Redundant responsive utility styles</b> <span>css</span>	#11214
Opened by AlexYursha a month ago  24 comments		
	<b>Select tag not properly styled on stock android browser</b> <span>css</span> <span>confirmed</span>	#11055
Opened by ADmad a month ago  19 comments		
	<b>horizontal scrollbar on viewport 000px &lt; 1010px - full width hamburger</b>	#11000





## **Making an Issue**

1. Get one person in your group to set up issue templates
  - a. Settings / Features
2. Create a feature request for an additional piece of information that you would like to be added to each persons .md file (only one of these is needed)
3. Once this feature request is made, get one person to implement this on a branch and create a new pull request that cites the issue.

**Remember and turn your big screens to a single person so everyone can see what is happening for step 2 and 3.**

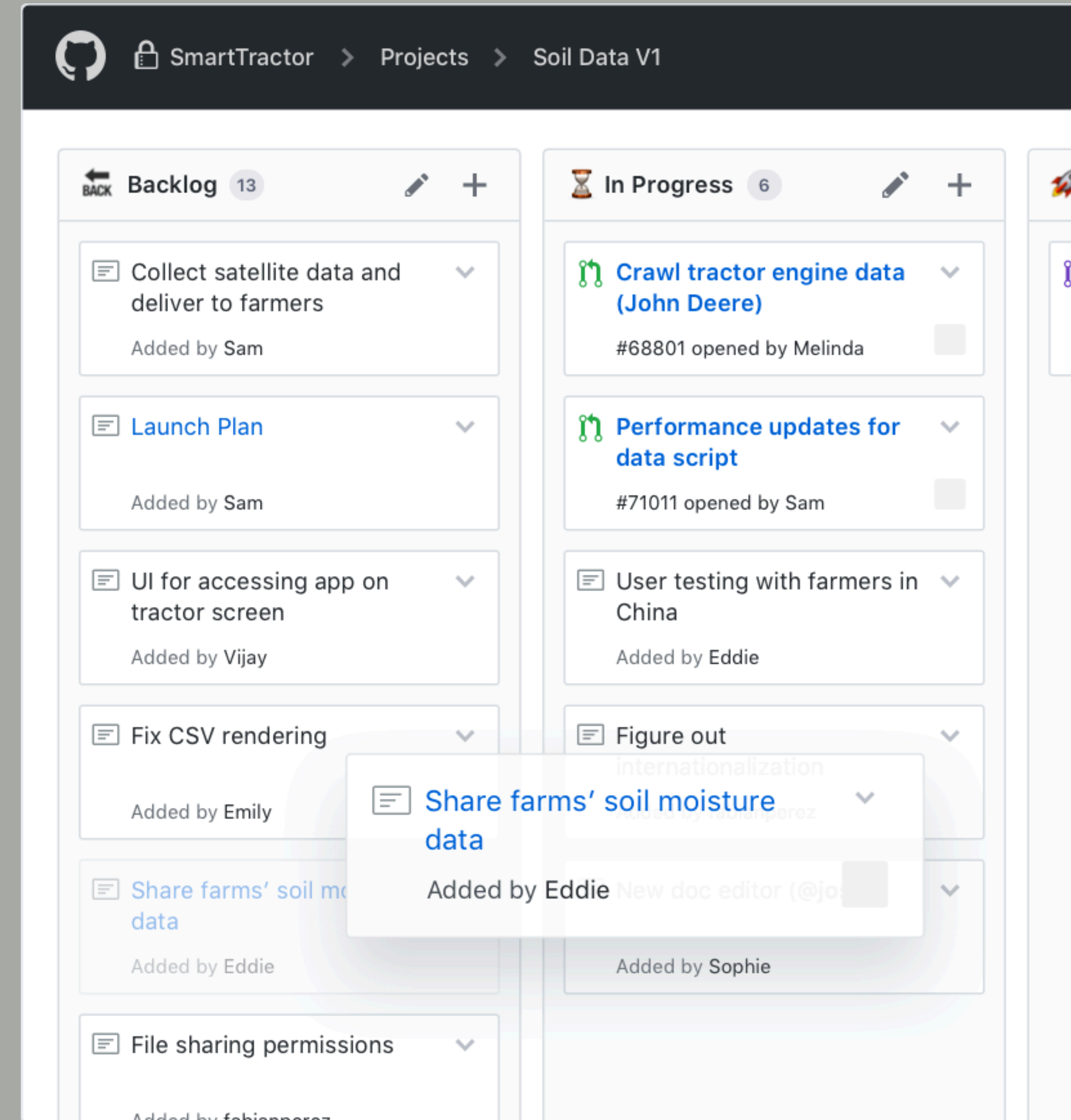
# PROJECT MANAGEMENT

# GitHub and Project Boards

- Project boards on GitHub help you organise and prioritise your work.
- You can create project boards for specific feature work, comprehensive roadmaps, or even release checklists.
- With project boards, you have the flexibility to create customised workflows that suit your needs.
- Project boards are made up of issues, pull requests, and notes that are categorised as cards in columns of your choosing.

# Project Board Automation

- We can automate our boards so that they update with our projects allowing us to keep track of how things are going.
- Simplest automation uses **Automated Kanban** template
  - Anytime an issue is made it goes into toDo
  - Anytime an issue is reopened it goes into inProgress
  - Any time a pull request is made it goes into inProgress
  - Any time an issue is closed it goes to Done
  - Any time a pull request is closed it goes into Done







## **Creating Project Boards**

1. Get one person in your team to create a new project board using the Automated KanBan method
2. Create 3 new issues / feature requests (each) and watch the board grow!
3. Decide which of these issues / feature requests you want to work on and create a solve it using gitFlow
  - a. Keep an eye on the board to see how it changes during this

**It may be worth pairing up and setting 1 computer to always show your project board on the big screen**

**CODE VERSIONING**

**GIT FLOW**

**GITHUB ISSUES**

**PROJECT MANAGEMENT**

# **GITHUB FOR COLLABORATIVE PROJECTS**