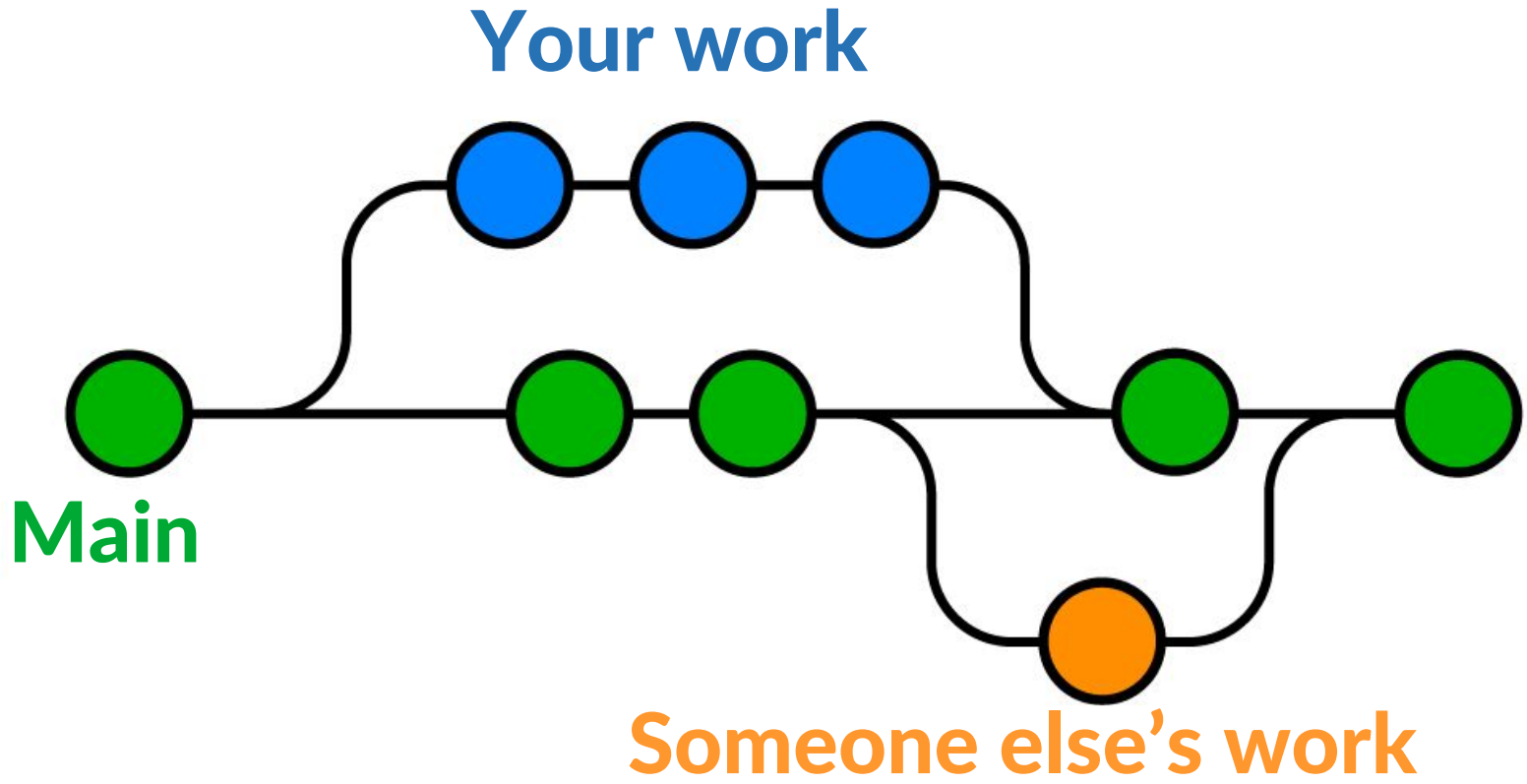
# 141B Lecture 3
# Git and github

# Versioning Systems (git)



# Overview

- maintaining versions of human readable files
- facilitate collaboration
- **git** is distributed - as opposed to cvs and subversion
- **.git** folder contains history of changes made to “versioned” files
- **github.com** is a service that provides servers with repo and some other features: issues, project boards, pull requests, renders ipynb & md
- Some IDEs (pycharm, RStudio) have built in git (I don’t use this)

# What should be tracked?

**Definitely:** code, markdown documentation, bash scripts/makefiles, ...

**Possibly:** logs, jupyter notebooks, images (that won’t change), ...

**Questionable:** processed data, static pdfs, ...

**Definitely not:** full data, continually updated pdfs (other things compiled from source code), ...

# Versioning commands

`git init : initialize empty .git`

`git add ___ : track the changes to file (put in staging area called the “index”)`

`git commit -m “message here” : commit the changes made`

`git status : see what you tracked and untracked changes since commit`

`git log : see history`

`git diff : see differences within files`

# Messages

- concise but descriptive, add body if more explanation is needed
- 50 char limit recommended
- imperative mode
- don’t overthink it, but try

# Collaboration commands

`git clone : copy repo`

`git pull repo-name branch-name: fetch changes (puts them into .git) and merges changes to branch`

`git remote add repo-name http://github.com/path repo-name is now a shorthand for the full path`

`git merge : merge two branches (e.g. origin/main and main)`

`git push : push changes to remote repository (typically github), will complain if can’t be merged`

# Resolving conflicts

```
<<<<<<< HEAD:mergetest
 This is my third line (your commit)
 =======
 This is a fourth line I am adding (commit from merge)
 >>>>>>> 4e2b407f501b68f8588aa645acafffa0224b9b78:mergetest
```

# Daily operations

```
git add code.py

git commit -m “add header flag to df reader”

git pull or git fetch; git merge

(resolve conflicts if any and commit again)

git push
```

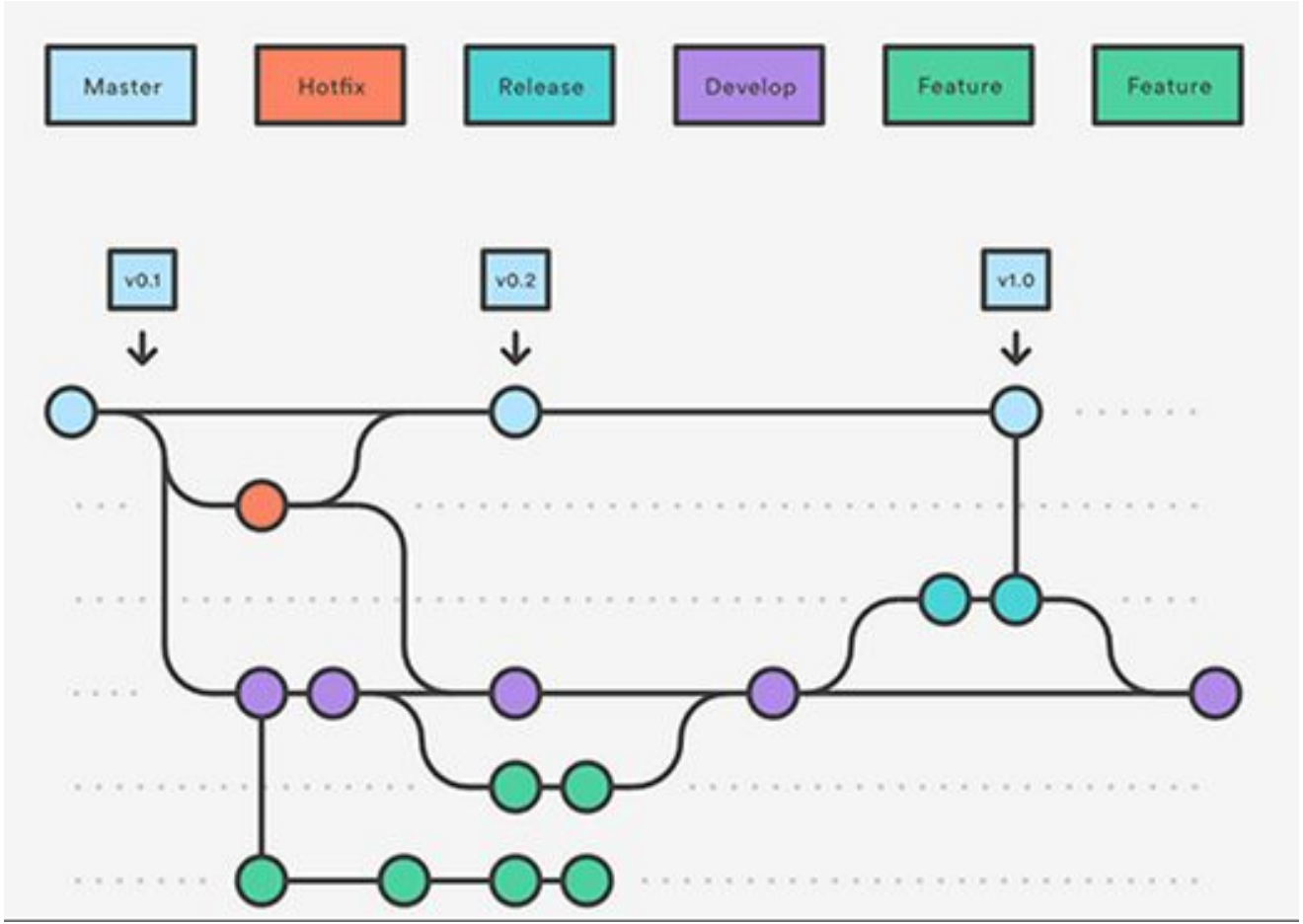
- commit often (small “atomic” commits)
- take messages (your own, and others) seriously
- commit complete, tested code

# .gitignore

`.gitignore : tells git what to ignore during git add`

- git status will not mention ignored files
- git commit -a will not add them
- easy mistake is to delete .gitignore
- https://git-scm.com/docs/gitignore

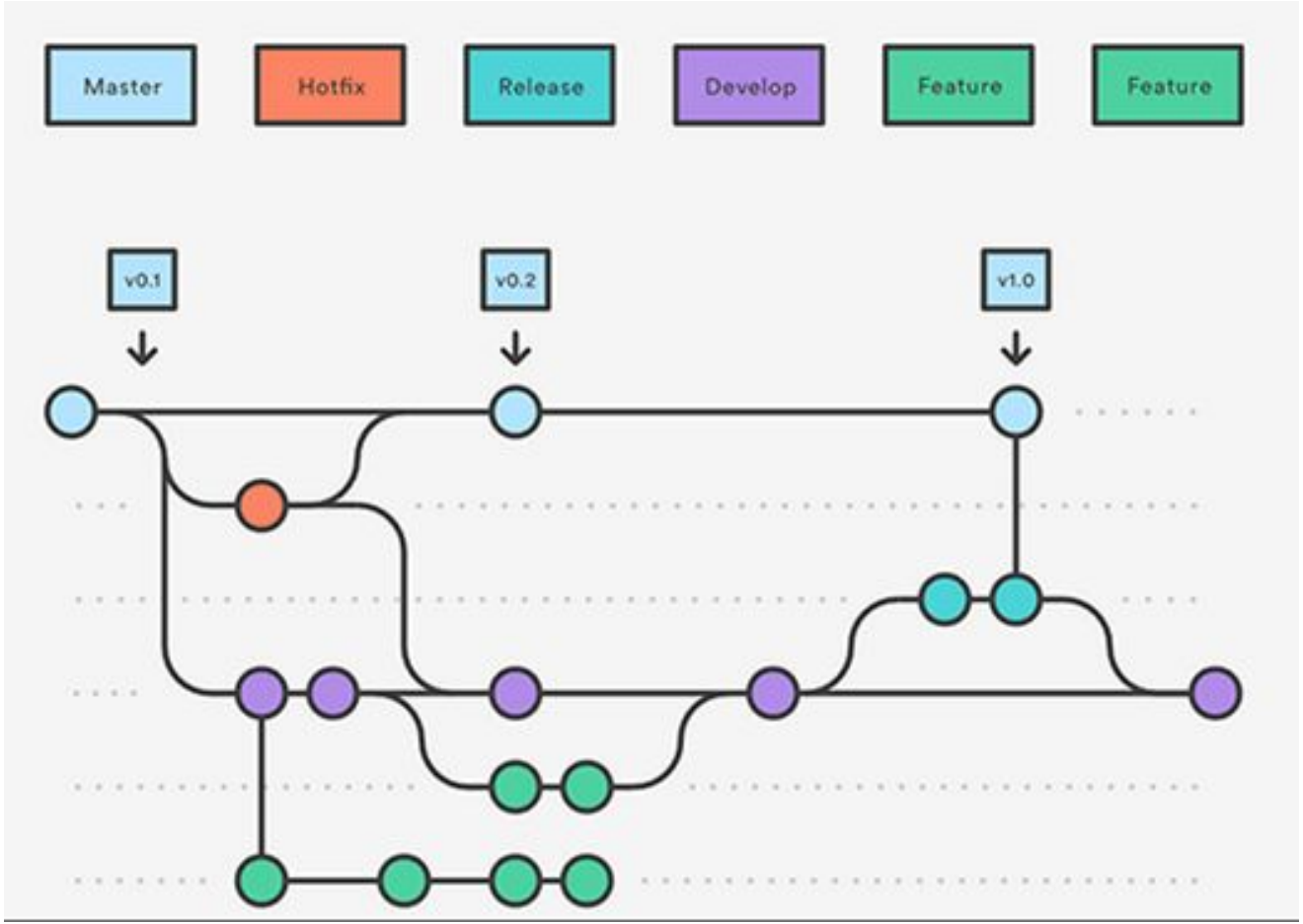
# Gitflow

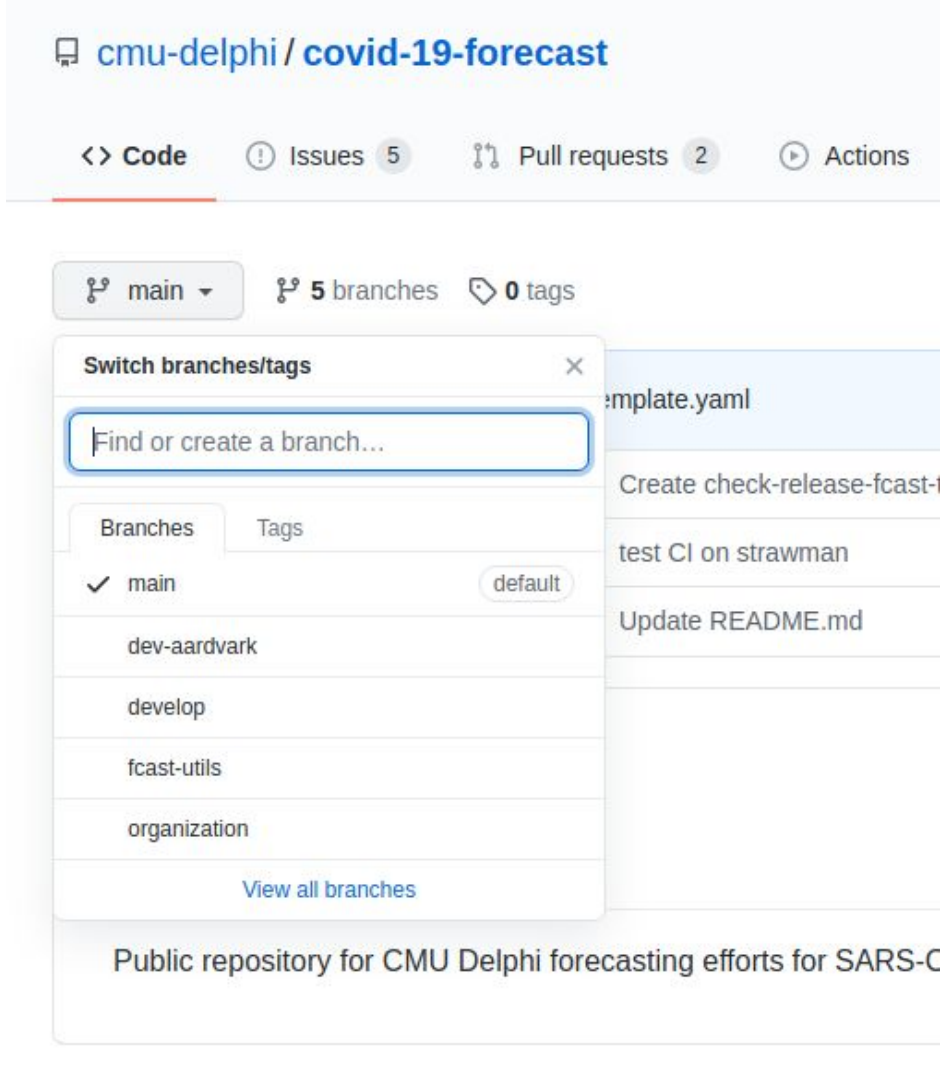


# Merging branches without a Pull Request

```
git checkout -b myfeature dev-aardvark : create branch

… do some work…

git checkout dev-aardvark

git merge myfeature : to be replaced with pull request for devel
and main

git branch -d myfeature : delete feature branch

git push origin dev-aardvark : push develop changes to github
```

# Gitflow

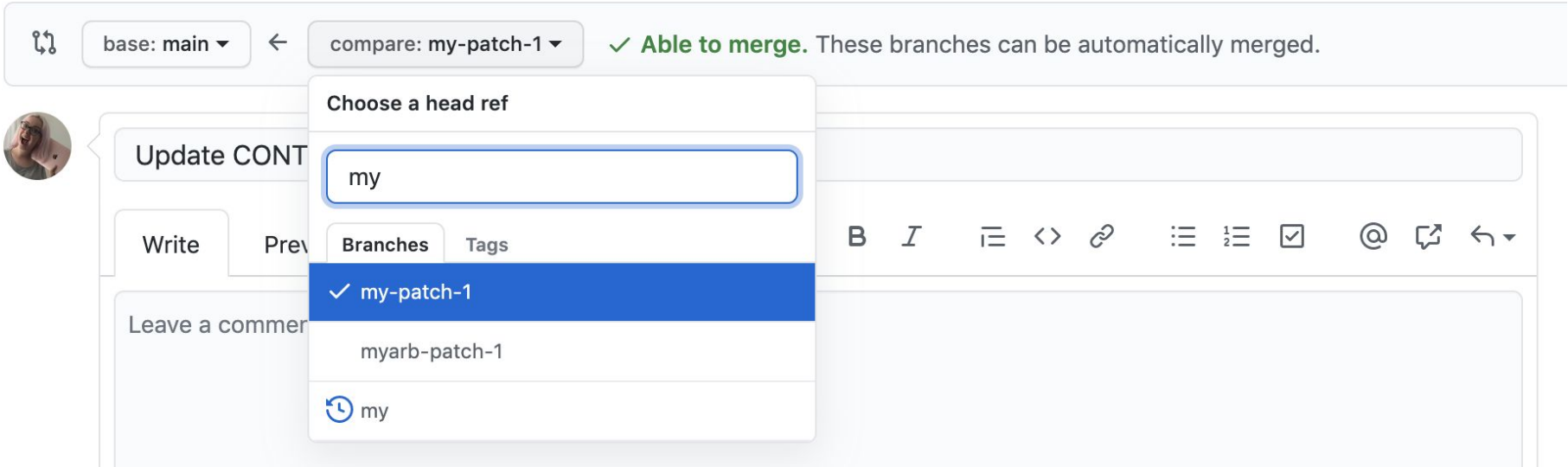



cmu-delphi / covid-19-forecast
Code
Issues 5
Pull requests 2
Actions
main
5 branches
0 tags
Switch branches/tags
Find or create a branch...
Branches
Tags
main
default
dev-aardvark
develop
fcast-utils
organization
View all branches
emplate.yaml
Create check-release-fcast-
test CI on strawman
Update README.md
Public repository for CMU Delphi forecasting efforts for SARS-C

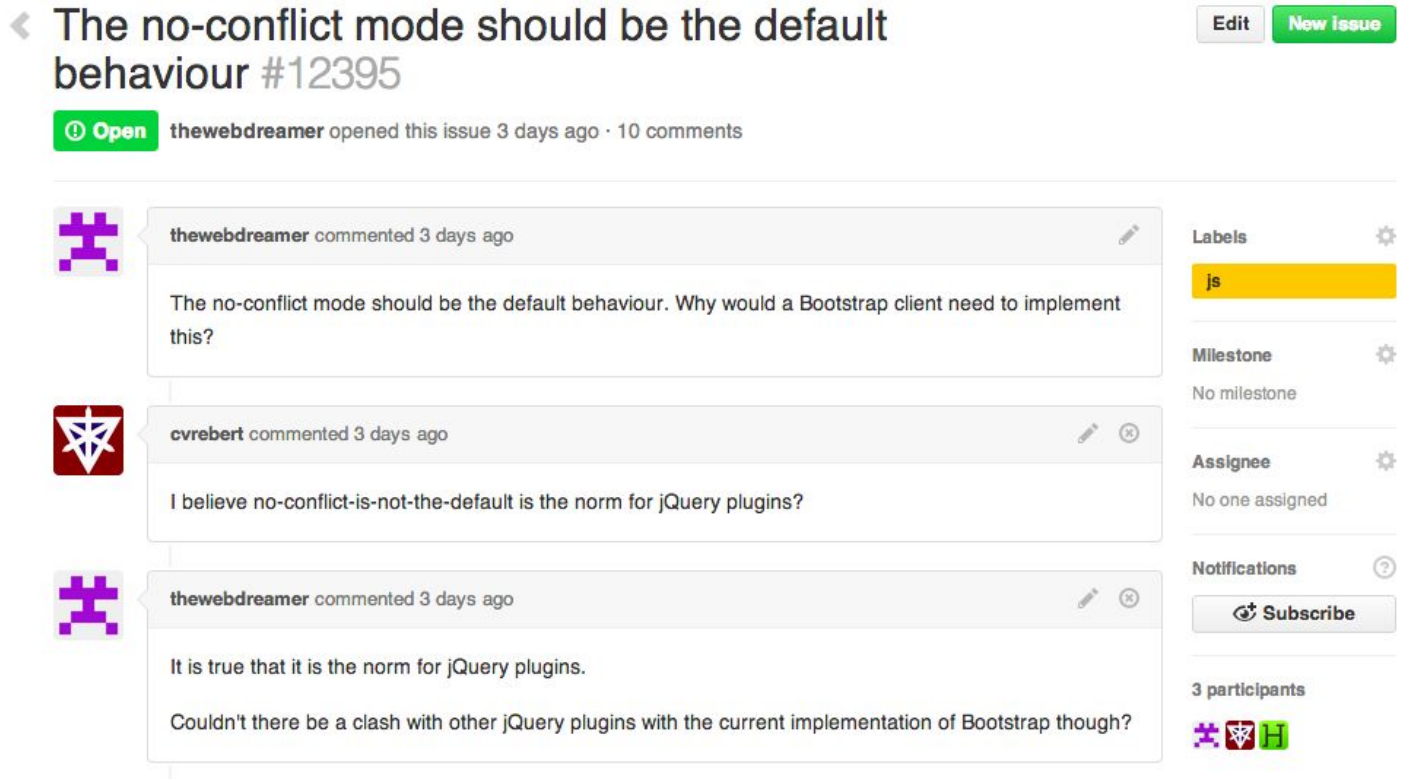
# github.com stuff : pull requests

## Open a pull request

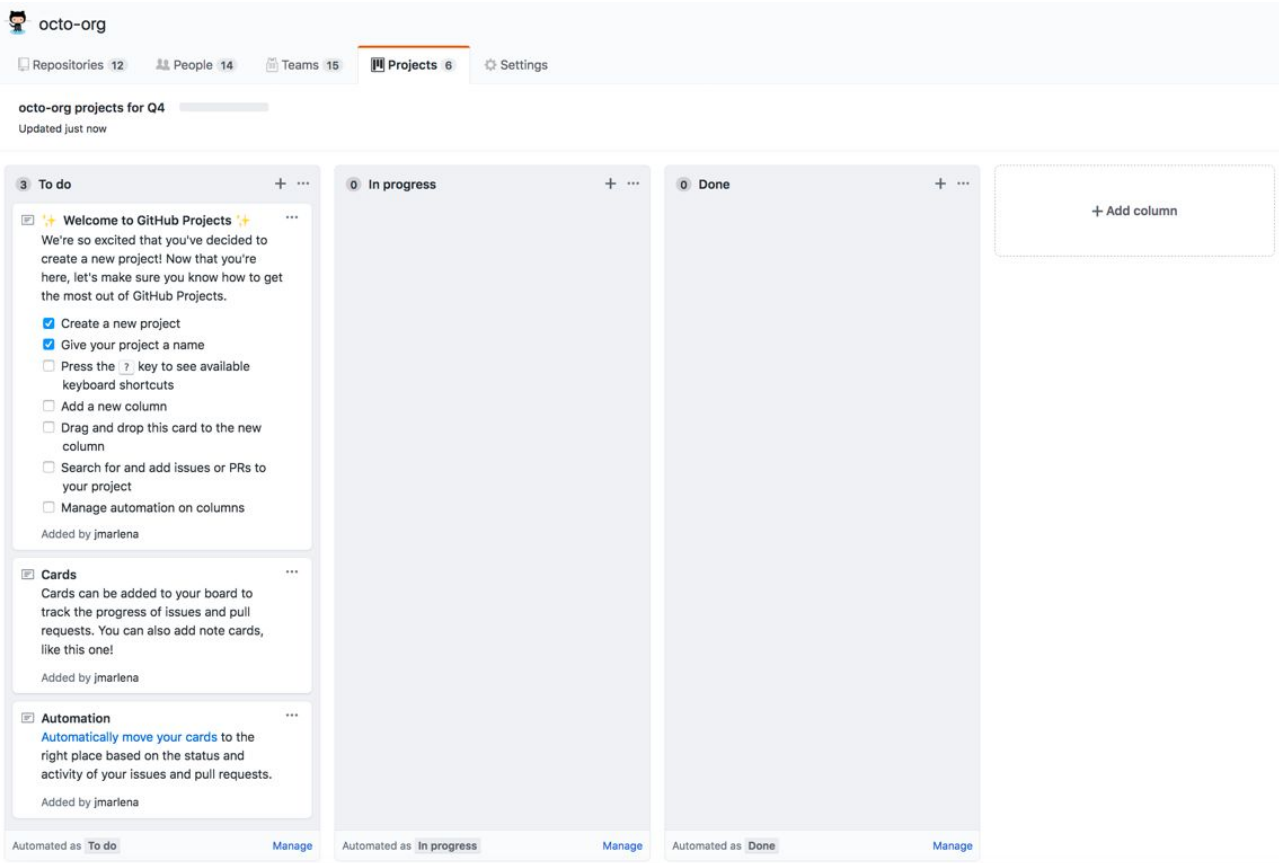
Create a new pull request by comparing changes across two branches. If you need to, you can also compare across forks.



# github.com stuff : issues



# github.com stuff : project boards



# good repo git standards

1. **Commit often with concise descriptive messages**, code bite-sized features, debugged and tested
2. **Develop and feature branches:** Add new features (forecasters, utilities) by branching off of develop, use naming scheme `dev-*`; merge with a PR and assign reviewer
3. **Main is for releases**, merging into main should be for hotfix, naming scheme is `hotfix-*`,or new release from develop (maintainer will make a PR against main)
4. **Issues** should be generated for desired features or bugs in release candidate (dev-*), bugs in main (hotfix-*), etc.
5. **Project board** tracks issues to be resolved in each release