15-441: Computer Networks

Recitation 4

Agenda

1. Project 1 Quick Reminder
2. Partners and Group
3. Dividing work in a team
4. Writing code in a team
5. Activity: Play with Git
6. Q&A

Project 1: Quick reminder

15-441

CP	Grade	Deadline
2	75%	Sep 27

15-641

CP	Grade	Deadline
3	35%	Sep 27

Start early! Do not wait until the last day!

Partners and Group

In Project 2 and Project 3, you will need to complete the project in **group of 2**.

641 Students are allowed to form team with 441 students, and 441 students are also allowed to form team with 641 students :)

How to find partners?
There is a piazza post for you to search partner!

Dividing Work in a Team

1. Read through the project writeup and list tasks that you need to finish as a team
2. Divide those tasks based on their relevance
3. It is better that each student could write and test their code individually, and then combine their code together to form an entire system

Some Suggestions When Dividing Work

1. Keep modularity in mind
2. Divide the work properly before start developing
3. It is suggested that each student contributes equally to the project
4. If your project partner is not contributing to the project, please contact the course instructor

Writing Code in a Team

Some suggestions… [1]

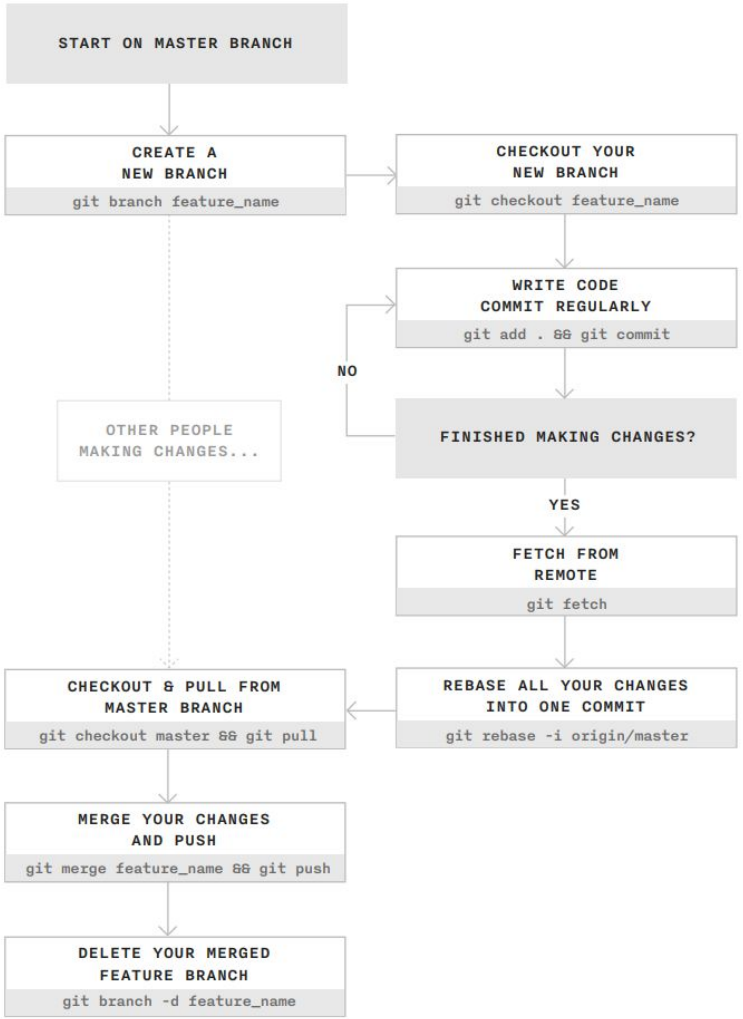
1. Use version control
2. Each developer should update and commit to the version control after finishing one task
3. Each developer should test the new feature or bug fix one is adding before commit to the repository (make sure it doesn't contain trivial bugs before commit! Or it will be a disaster for your teammates...)

Using git in a Team

Basic Idea: [2]

It is suggested that each developer should

- Checkout a new branch
- Work on new branch
- Merge changes back to master branch



Step1: Create a new branch to work on

When naming feature branches, a good best practice is to start with you initials, then the feature name (e.g. myname_newfeature)

```
# Create a new feature branch
git branch myname_newfeature

# Checkout your new feature branch
git checkout myname_newfeature
```

Step2: Write Some Code and Commit

Commit your code after finishing some milestones. But make sure to check for trivial bugs before commit.

```
# Add needed files to the stage
git add files

# Commit files
git commit -m "Description of this commit"

# Optional (but recommended) push local branch to remote
git push origin jc_feature_name
```

Step3: Fetch When You’re Done

When you’re ready to merge your features back into the master branch, you could do fetch. Fetching makes sure you’re up to date when merging changes back into master.

```
# Fetch
git fetch
```

Step4: Squash Your Commits and Get Ready to Merge (Optional)

Now you'll rebase your changes into the master branch. This effectively condenses down all the commits you've made on your feature branch (myname_featurename) into one commit.

```
# Open the interactive rebase tool
git rebase -i origin/master
```

This command will open an interactive rebase tool, and you can work on that.

Step5: Merge Your Changes

Switch to the master branch in preparation of merging your changes. After merging you could push your local master branch to remote.

```
# Checkout the master branch
git checkout master

# Merge your new feature branch INTO master
git merge myname_newfeature

# Push your local master branch to remote
git push origin master
```

Step6: Cleanup

With your changes merged into the master branch, you can safely delete your feature branches.

```
# Delete remote feature branch (the colon is important!)
git push origin :myname_newfeature

# Delete local branch
git branch -d myname_newfeature
```

Activity: Play with Git!

- **Set up a new repository or clone from a existing one**:

```
$ cd target_dir/
$ git clone https://github.com/vvchd/Computer-Networks-Playground.git
```

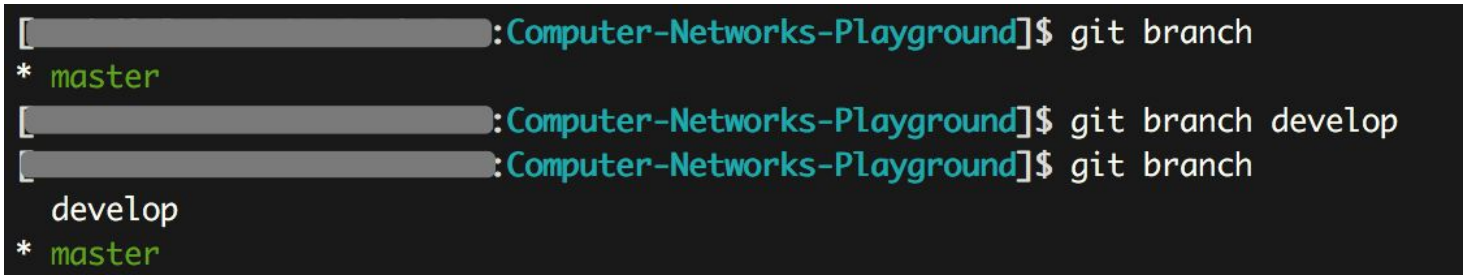
  This will clone a repository into your target-dir.

- **View local branches:**

```
$ git branch
```

- **Create a new local branch:**

```
$ git branch develop
```



Activity: Play with Git! (2)

- **Checkout to your own branch and develop!**

```
$ git checkout develop
$ echo "this is a new line" >> README.md
```

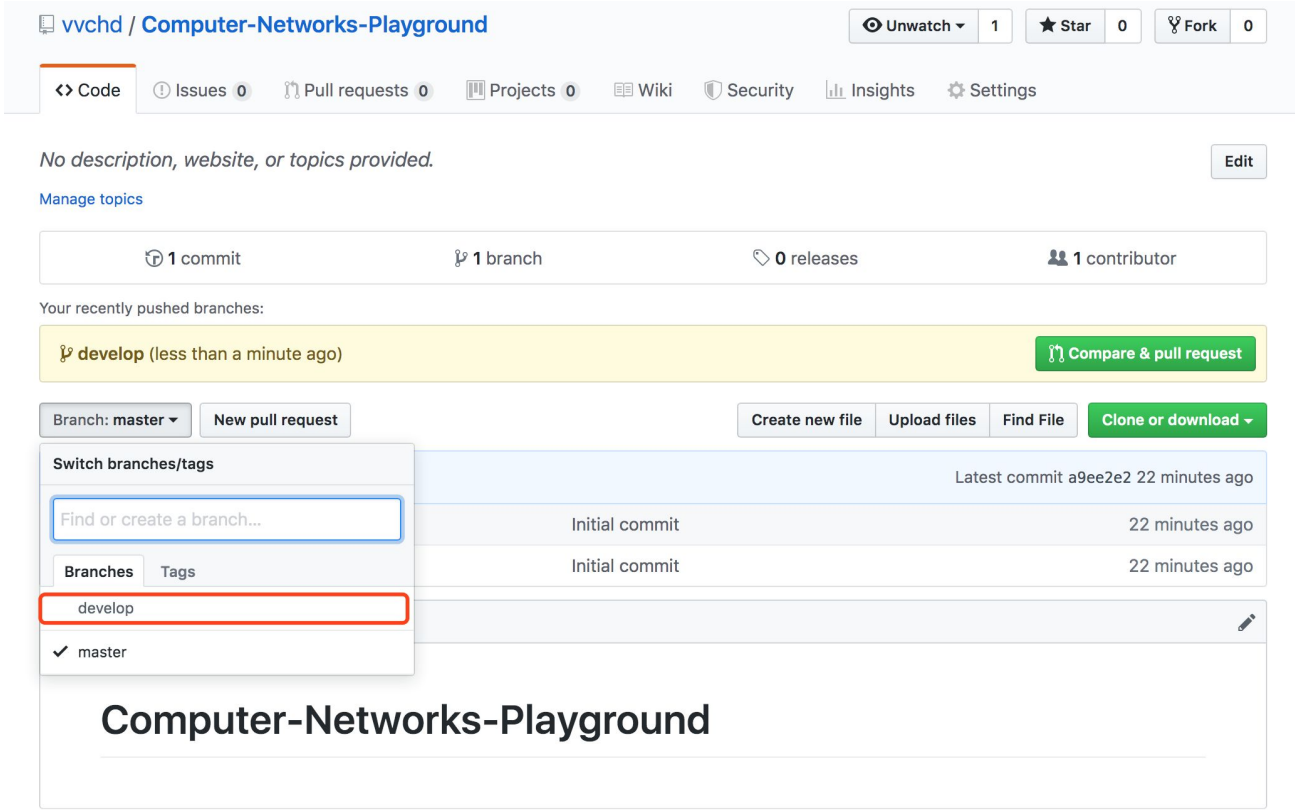
- **View and commit your changes**

```
$ git status
$ git add .
$ git commit -m “first commit to develop branch”
$ git push origin develop
```

The changes on your local `develop` branch will be pushed to remote repository. Then go to GitHub to check your new branch as well as the changes.

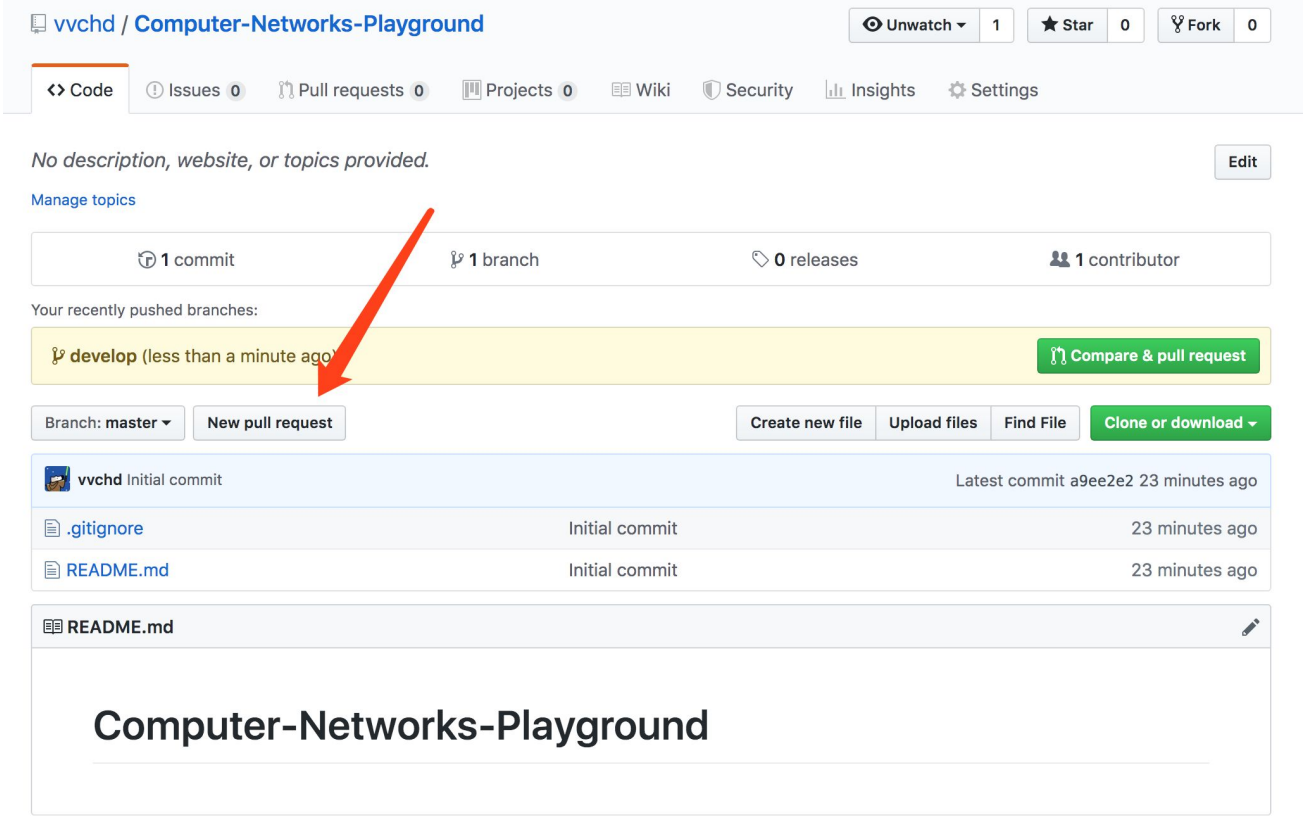
Activity: Play with Git! (3)

- **Navigate to the remote repository**



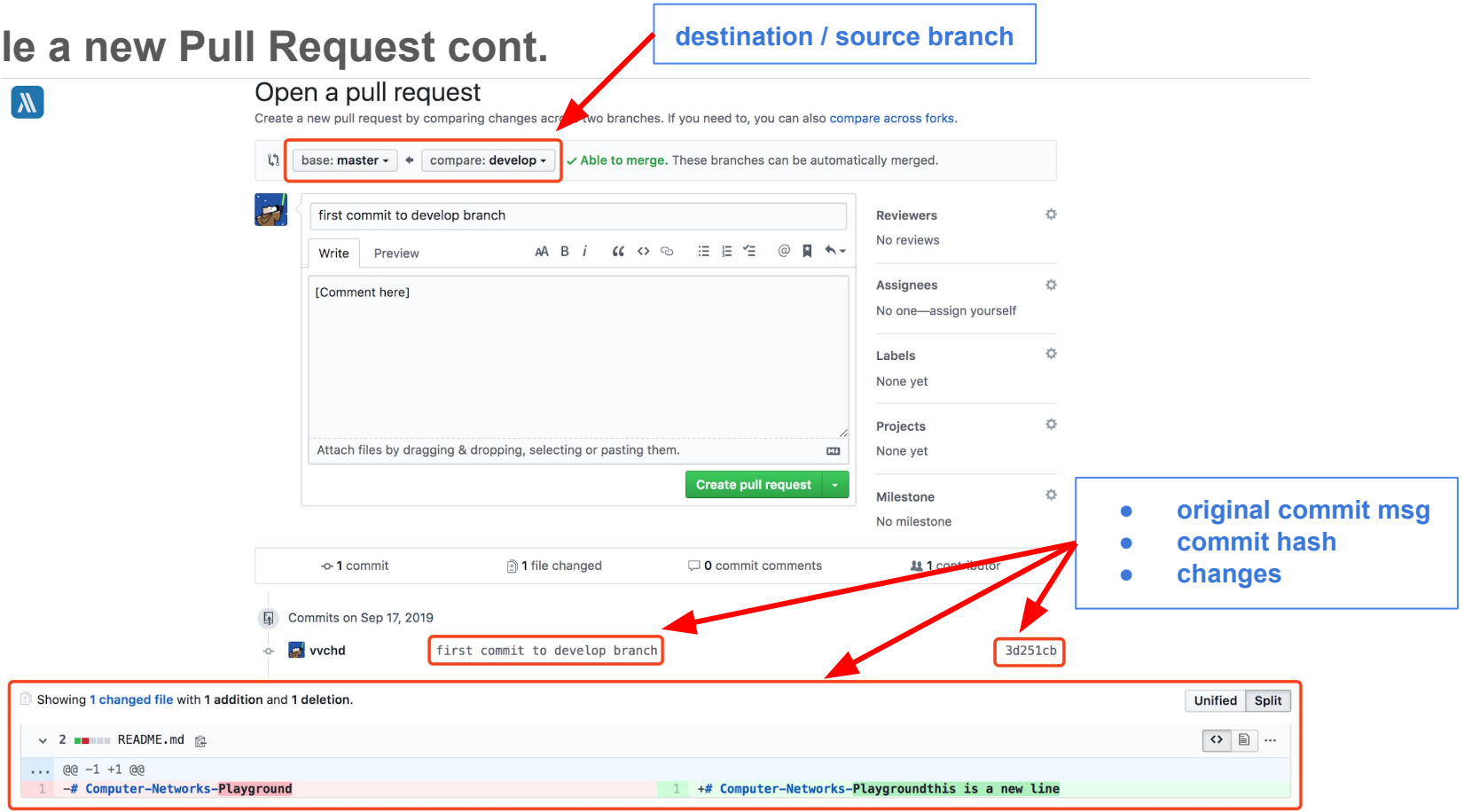
Activity: Play with Git! (4)

- **File a new Pull Request**



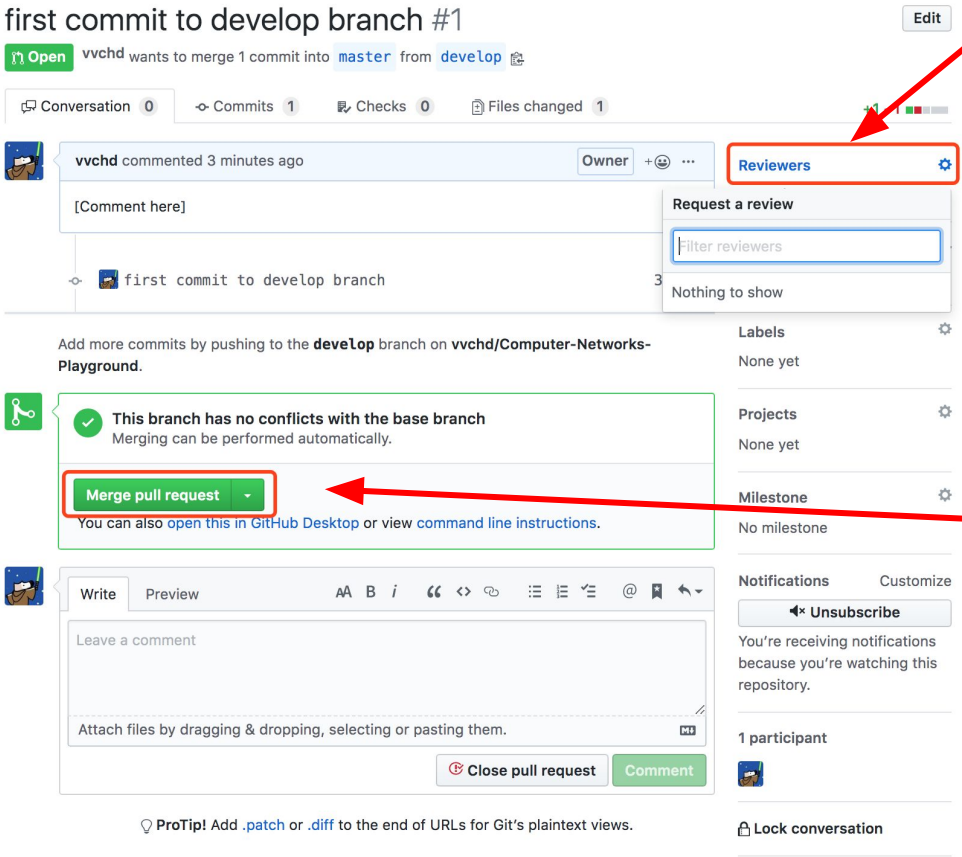
Activity: Play with Git! (5)

- **File a new Pull Request cont.**



Activity: Play with Git! (6)

- **File a new Pull Request cont.**

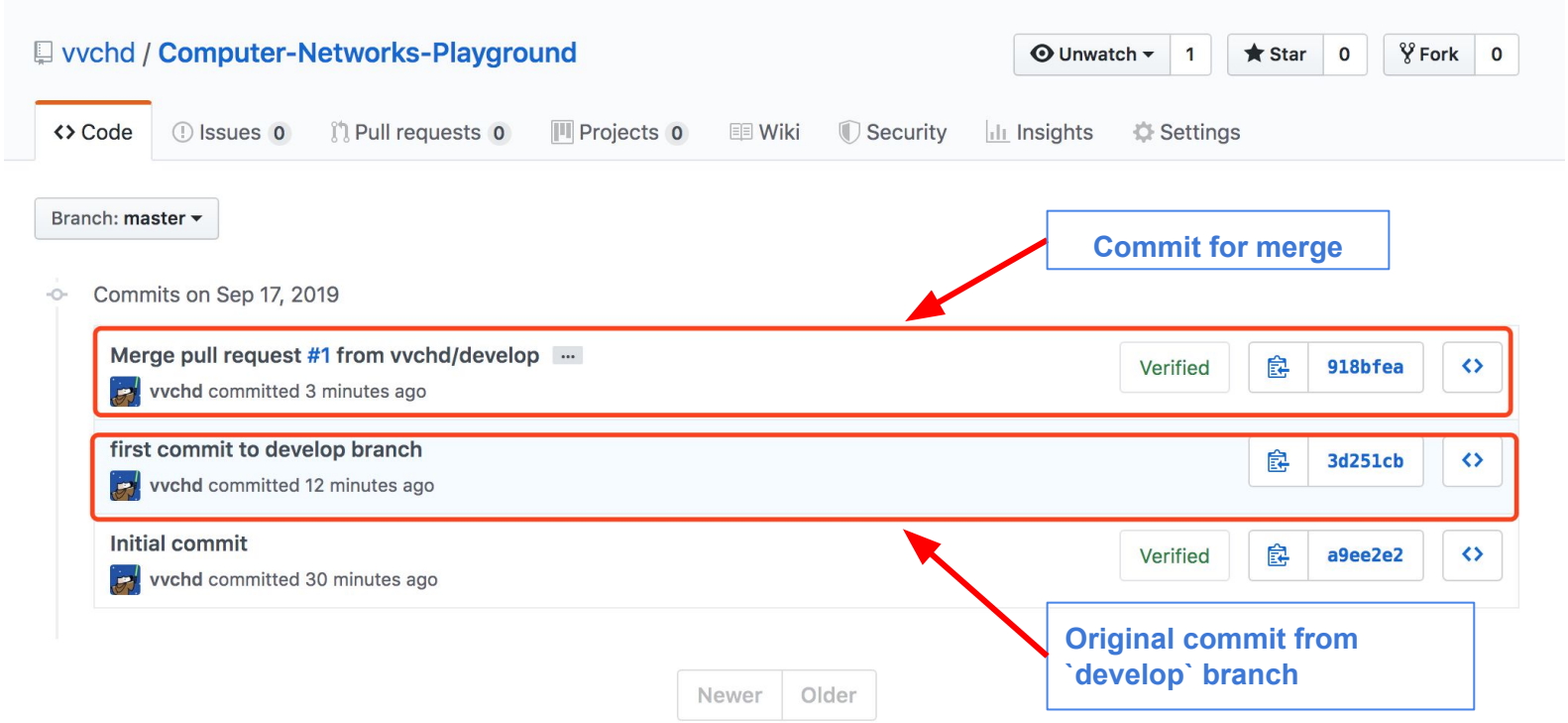


Add your teammate as reviewer

After reviewing code and resolving conflicts, if any, merge your PR here

Activity: Play with Git! (7)

- **You can check changes, after merging “develop” into “master” branch**



Q & A

References

1. How do programmers work together on a project: https://stackoverflow.com/questions/3000190/how-do-programmers-work-together-on-a-project
2. Using git in a team: a cheatsheet https://jameschambers.co/writing/git-team-workflow-cheatsheet/