

# Git第二讲

清华大学

电子系科协软件部

李宗洹

[lizonghu@mails.tsinghua.edu.cn](mailto:lizonghu@mails.tsinghua.edu.cn)

[kevinli606@gmail.com](mailto:kevinli606@gmail.com)

## 上节回顾

- Git简介


Git是什么，有哪些特点，git在做什么.....

- 本地仓库操作

工作区和版本库，add和commit，版本回退，撤销修改.....

- 关联远程仓库

remote关联，从远程克隆clone，基本的pull和push.....



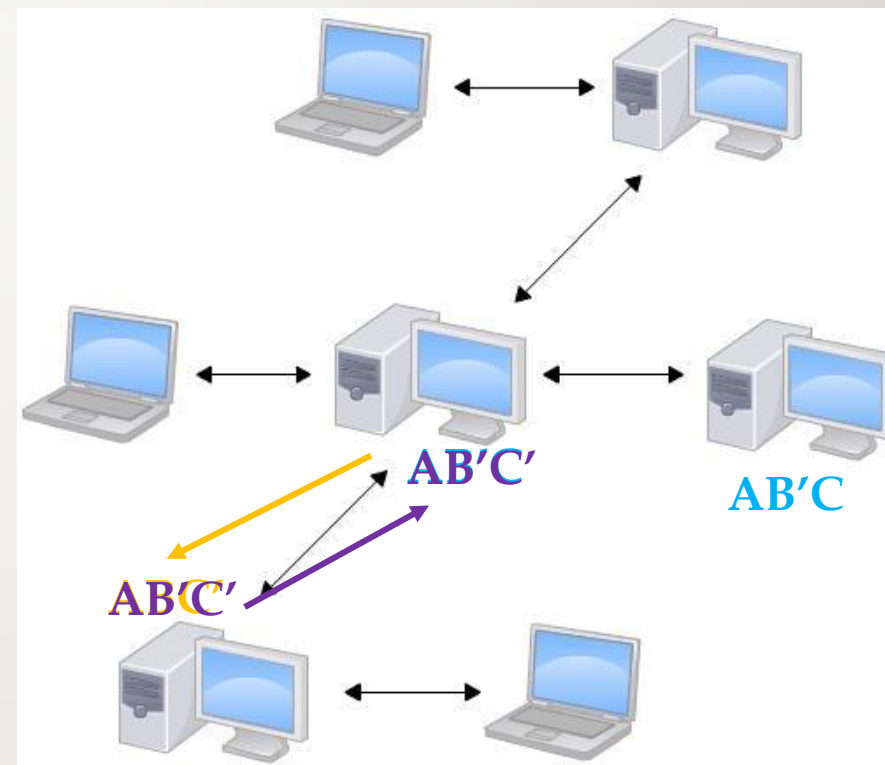
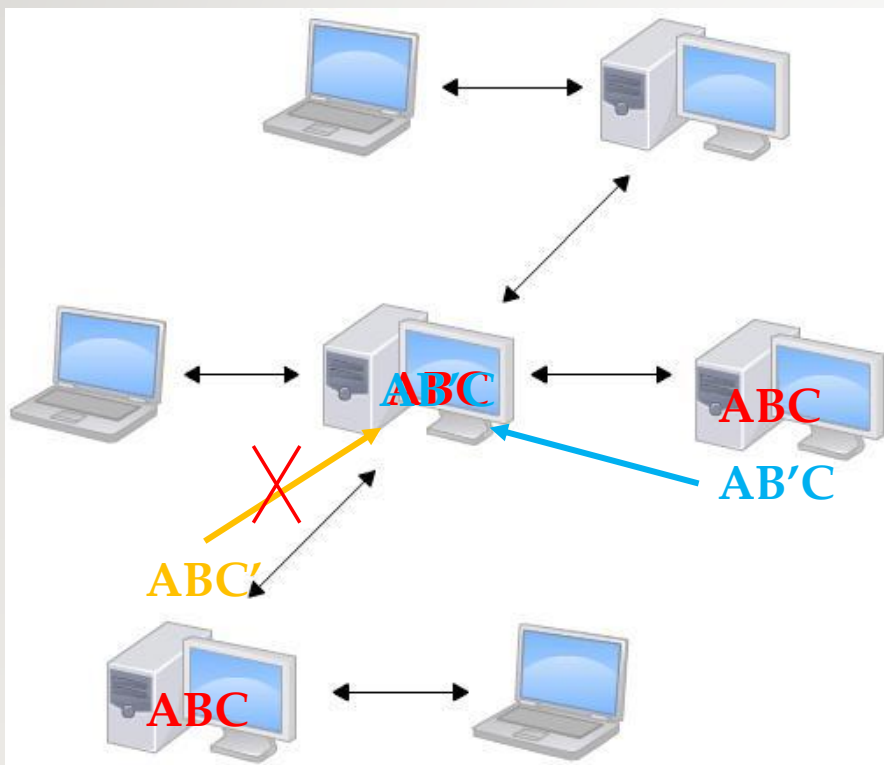
# 主要内容

- 远程仓库操作
- 分支管理
- 标签管理
- GitHub使用

## 远程仓库操作

- pull和push操作都有可能遇到问题，也就是**冲突(conflict)**
- 先看push
- 如果push时远程仓库已经被更新过了怎么办呢？
- 先用pull命令获取远程更新并与本地合并后，再push
- 思考：什么时候会出现这种情况？

## 远程仓库操作

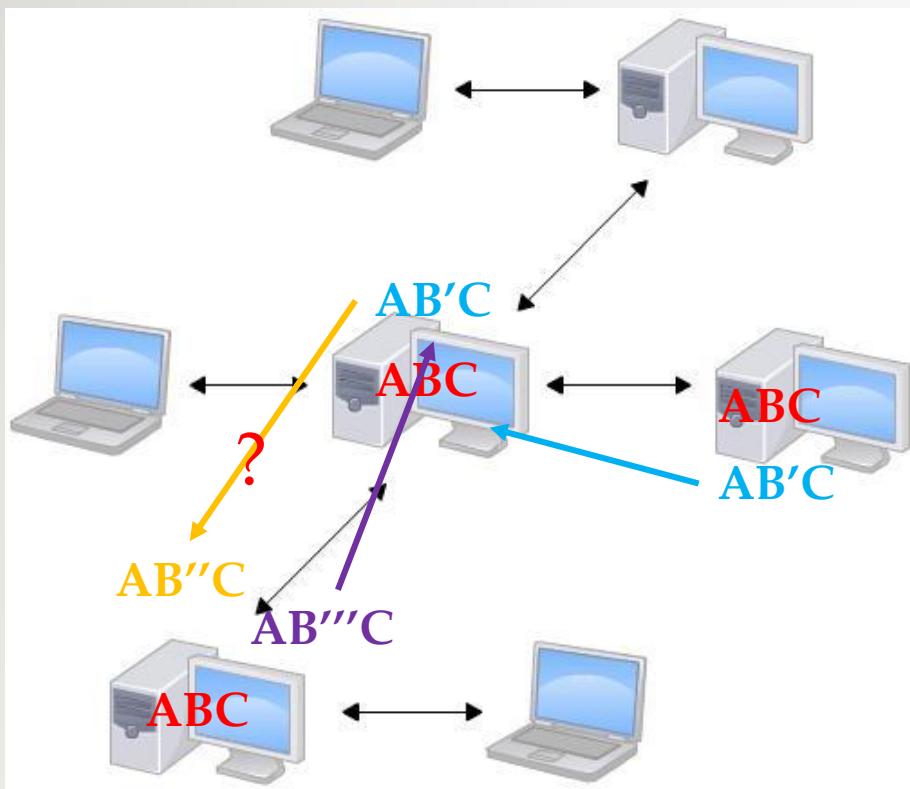


## 远程仓库操作

- 再看pull
- 即使先pull，也有可能遇到问题
- 在pull时，可能远程仓库的修改和本地修改矛盾怎么办？
- 一方面，git会自动合并不冲突的修改
- 另一方面，冲突的修改需要手动处理，之后再commit
- 思考：什么时候会出现这种情况？



## 远程仓库操作



- git会自动合并不冲突的修改
- 有冲突的修改需要手动处理
- 编辑器会显示冲突的内容，你来选择

```
>>>>>XXXXX
  XXXXXXX
=====XXXXX
  YYYYY
<<<<<XXXXX
```

# 远程仓库操作

- 小结
- pull或push时都可能遇到冲突
- 先pull再push是好习惯
- 要及时push，不要存着很多修改，否则会很麻烦
- 一定要学会手动修改冲突，不要“删库解决”



# 分支管理

- 为什么需要分支？
  - 多人开发中，写了不完整的代码
  - 直接提交？整个项目会因此出问题
  - 写完再提交？不保险，可能丢失进度
- 
- **分支(branch)**可以很好地解决这个问题
  - 在自己的分支上工作，互不影响

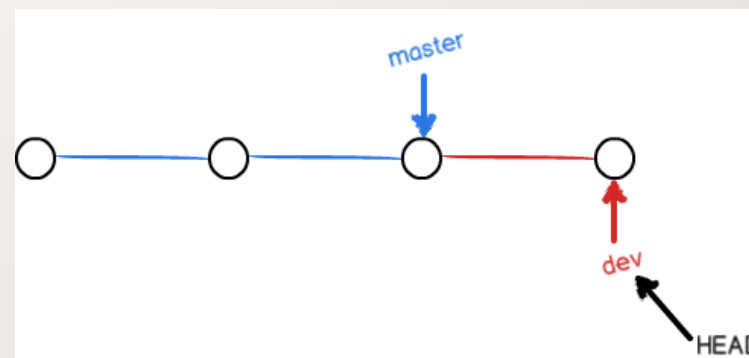
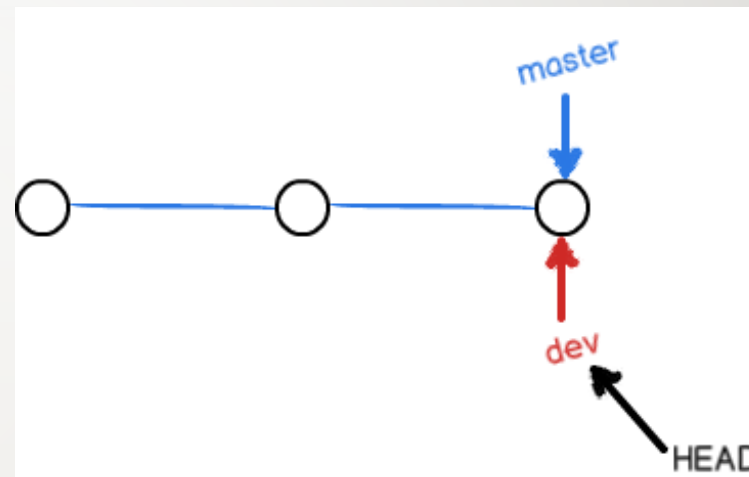
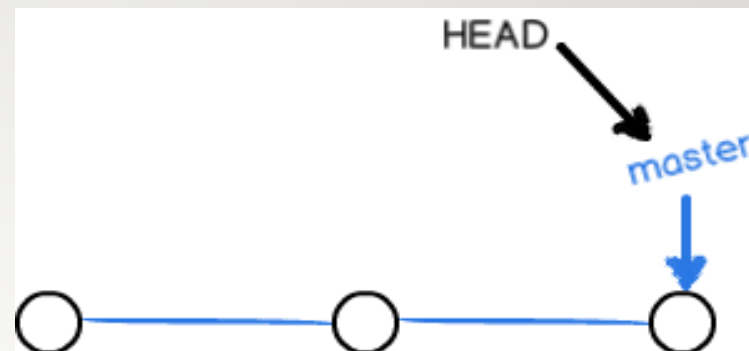
# 分支管理

- 分支长什么样？



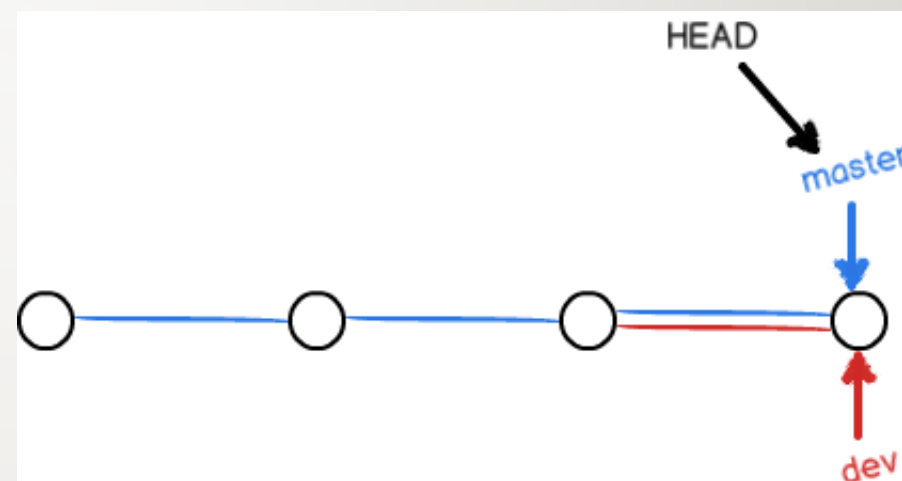
## 分支管理

- Git已有的分支：master
- 创建新分支：
- `$ git checkout -b dev`
- -b表示创建同时转到dev
- 此时就可以在dev分支上工作



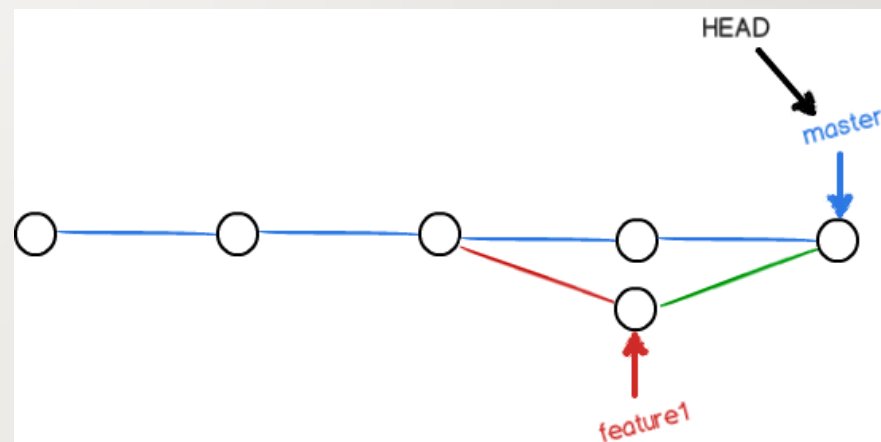
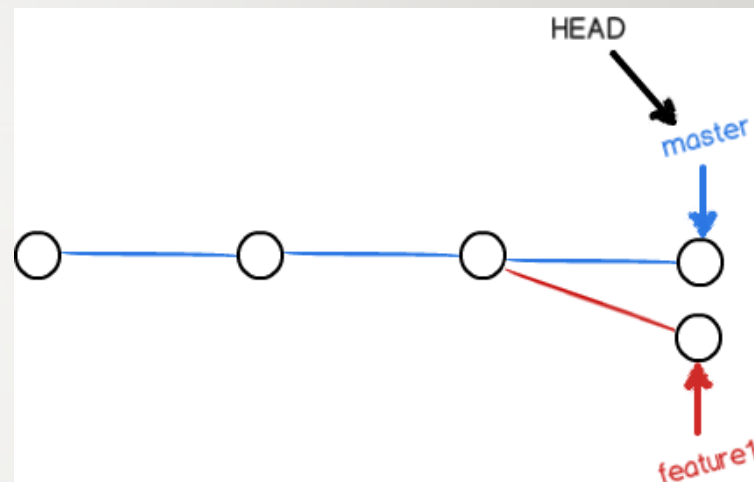
## 分支管理

- 工作完成，如何切换回原来的分支？
- `$ git checkout master`
- 在刚切回master分支时,会发现工作区内容和原来master分支内容一样,看不到在dev分支以上的修改
- 想合并两个分支？
- `$ git merge dev`
- 这样master分支就吸收了dev分支的修改！



## 分支管理

- 但别忘了，你开发dev时别人也在开发自己的分支
- 想merge时master已经被改变，怎么办？
- 类比一下pull时的冲突
- 先用 `$ git status` 查看冲突的文件
- 找到冲突位置手动处理冲突，再add和commit，即可完成本次合并

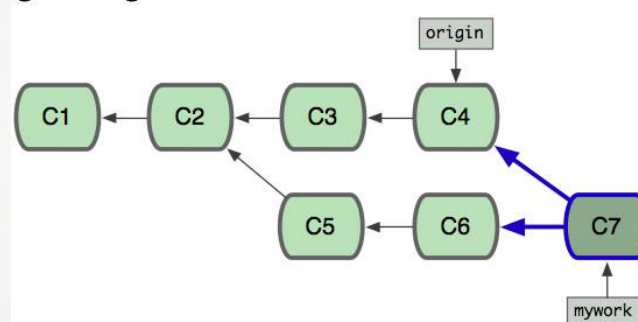




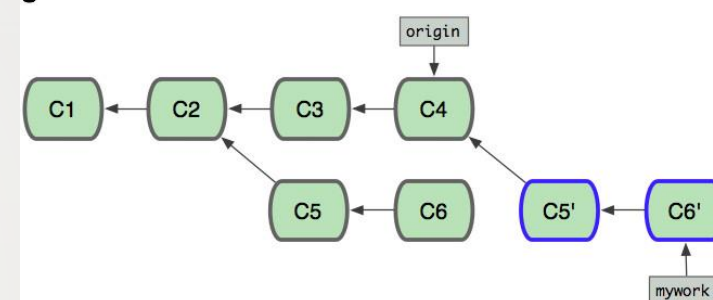
# 分支管理

- `$ git merge` 一般会出现的情况
- 想让版本树看起来像没有分支一样?

git merge



git rebase



- `$ git rebase`
- 把分叉的提交历史“整理”成一条直线

Assume the following history exists and the current branch is "topic":

```
      A---B---C topic
     /
D---E---F---G master
```

From this point, the result of either of the following commands:

```
git rebase master
git rebase master topic
```

would be:

```
      A'---B'---C' topic
     /
D---E---F---G master
```



## 分支管理

- 问题：现在正在工作的dev分支没写完，需要立刻修改master分支的内容，怎么办？
- `$ git stash`
- 作用：储存当前工作区的内容
- 将工作区变“干净”，可以放心转到master分支并修改bug
- 就像已经保存了修改一样
- 修复bug完成后，再转到dev分支继续工作

## 分支管理

- `$ git stash list`
- 查看当前储存的进度，例如：

```
$ git stash list  
stash@{0}: WIP on dev: ee659f4 dev
```

- `$ git stash pop`
- 恢复工作区，可以继续工作（“弹出”）

# 分支管理

- 有关分支的一些命令
- `$ git checkout -b newbranch` 创建名为newbranch的新分支并切换到新分支
- `$ git checkout abranch` 切换到名为abranch的分支
- `$ git checkout -d newbranch` 删除名为newbranch的分支
- `$ git checkout -h` 帮助文档

# 标签管理

- 标签是什么?
- **标签(tag)**是一个指向对应版本的指针
- 项目发布的时候，可以会发很多版本，如果我们给版本打上个标签，我们就能很方便地找到它。
- 每一次commit都可以加标签
- 比版本号更直观，如v1.0， v2.0等等

## 标签管理

- 如何为一个版本建立标签?
  - `$ git tag 标签名 版本号`
  - 可以不写版本号,默认是HEAD
  - 还可以用-a指定标签名, -m指定说明文字
- 
- 注意: 标签总是和某个commit挂钩。如果这个commit既出现在master分支, 又出现在dev分支, 那么在这两个分支上都可以看到这个标签。

# 标签管理

- 标签的相关命令
- `$ git tag` 查看已有标签
- `$ git show 标签名` 查看相应标签的信息
- `$ git tag -d 标签名` 删除相应标签
  
- 标签还可以push!
- `$ git push [远程仓库名] --tag/标签名`
- 这样就可以把标签推到远程仓库去



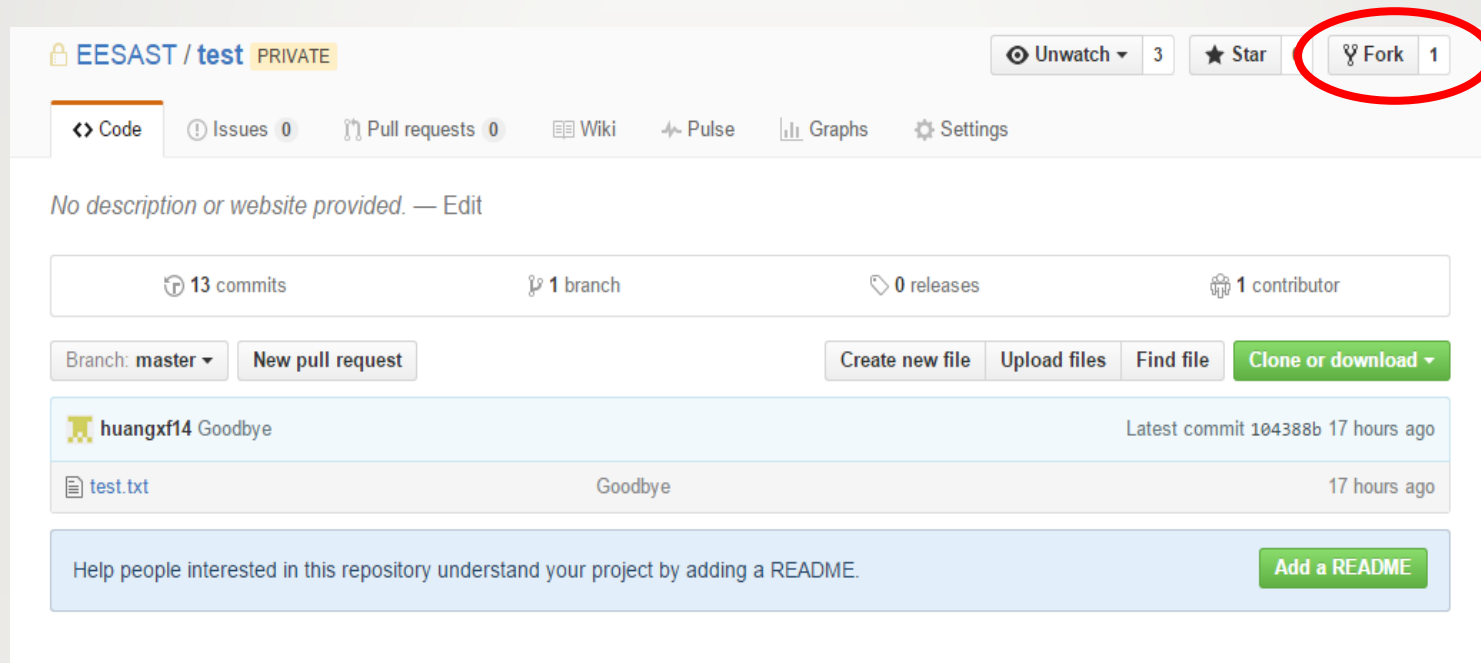
# GitHub使用

- GitHub界面.....



# GitHub使用

- GitHub上拥有众多的仓库，代码都是开源的
- 如果喜欢某个仓库的项目，可以**fork**到自己这里

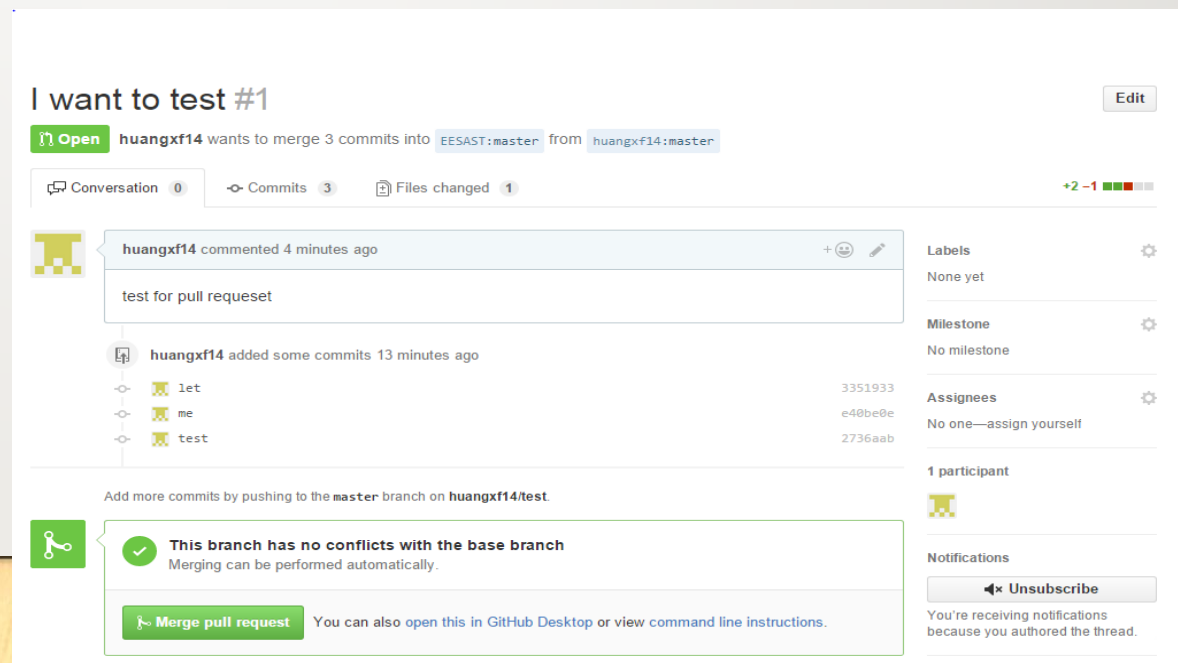
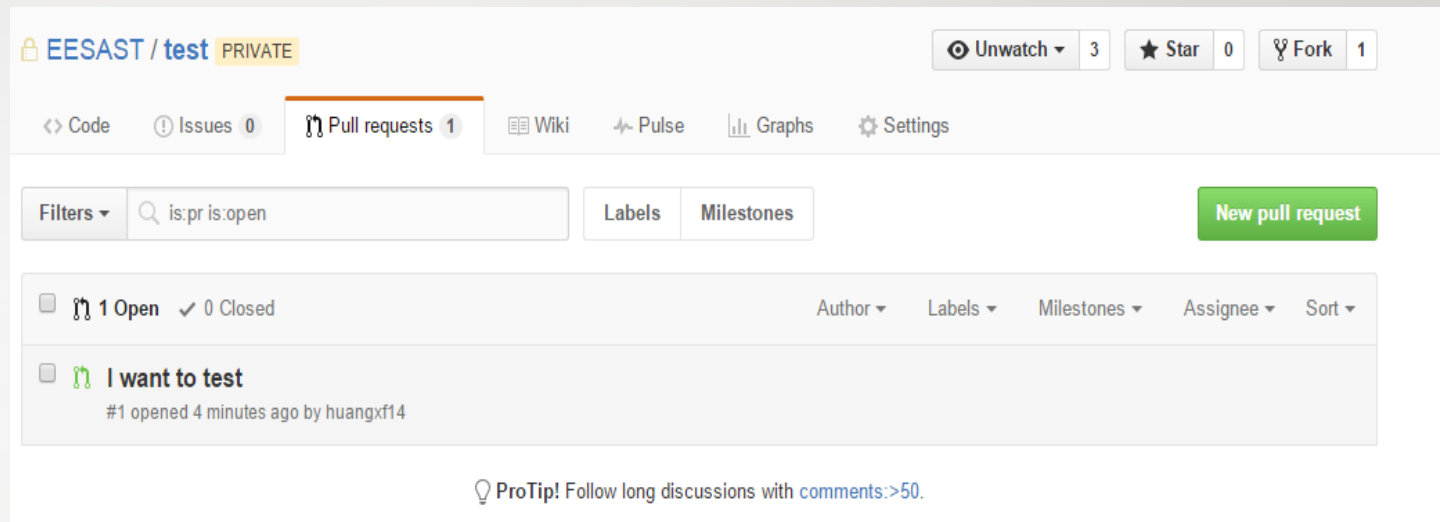


# GitHub使用

- 什么是pull request?
- 代码虽然可见，但不是人人都有权限向仓库推送的
- 如果你对代码有修改的建议怎么办?
- fork到自己仓库下，然后自己改，最后在仓库开启**pull request**
- 有权限的用户来决定是否采纳

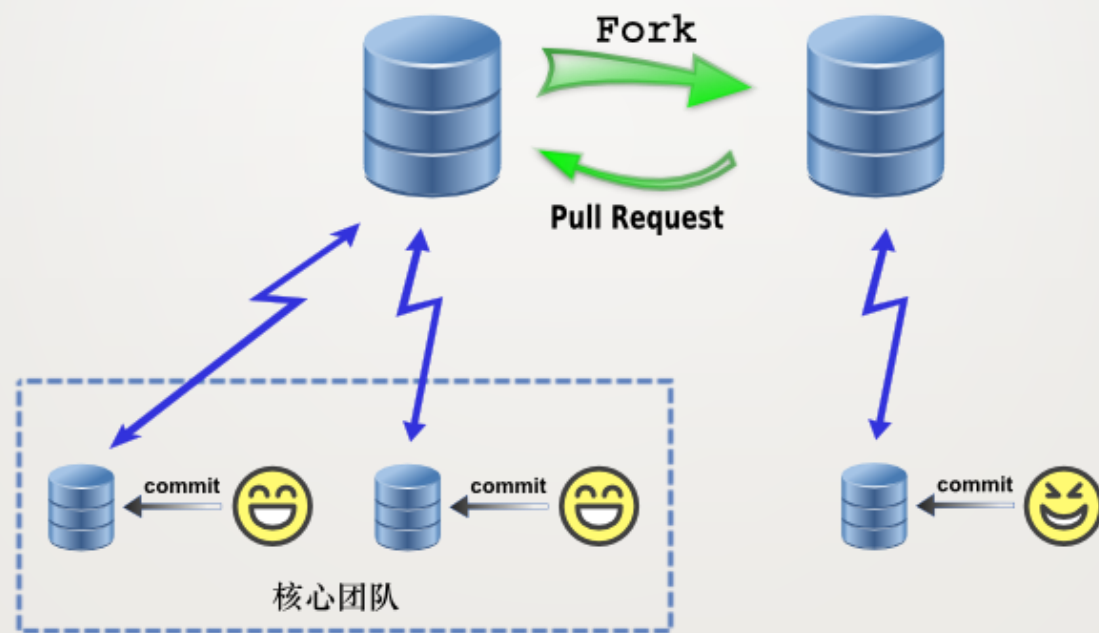
# GitHub使用

- 实际上是合并分支的操作
- 操作后可以close pull request
- pr给了所有用户为代码提出修改建议的渠道
- pr也是开发过程中应当遵守的规范



# GitHub使用

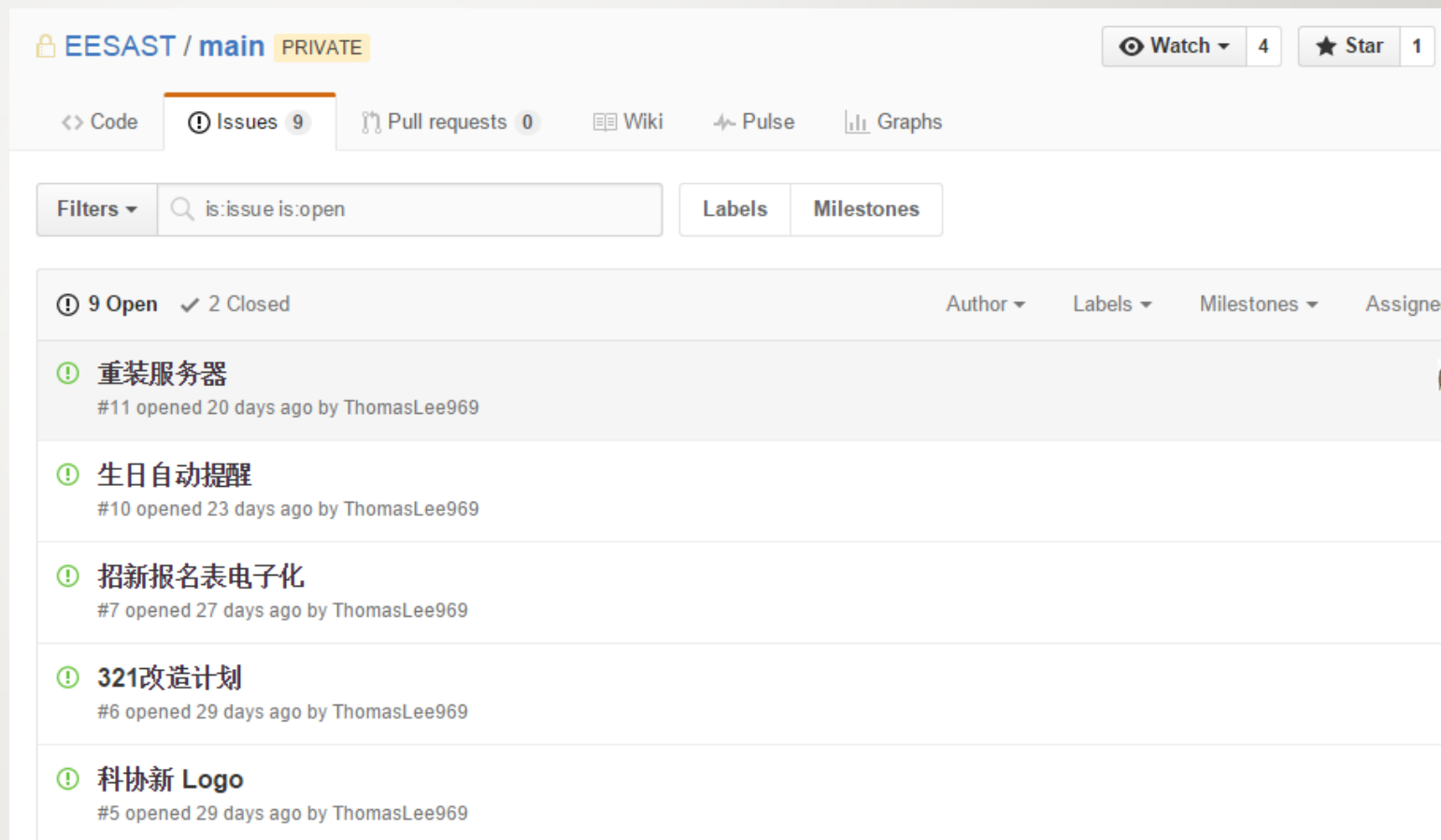
- 图示非核心团队成员贡献代码的流程





# GitHub使用

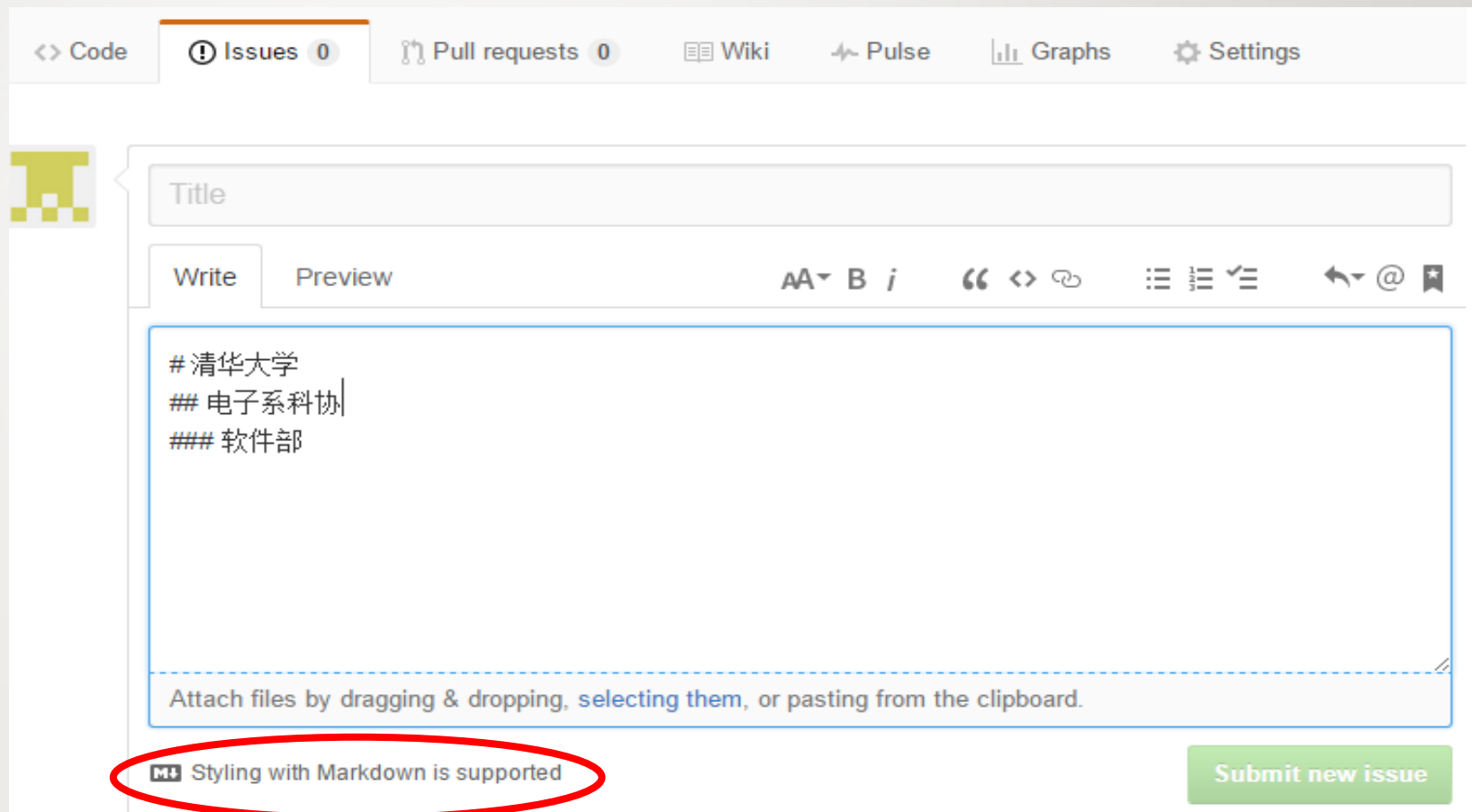
- 什么是Issues?
- 在软件开发过程中，开发者们为了跟踪BUG及进行软件相关讨论，进而方便管理，创建了**Issue**
- 遇到bug可以搜索Issue





# GitHub使用

- 可以自己添加Issue
- 支持Markdown语法进行描述



<> Code **! Issues 0** Pull requests 0 Wiki Pulse Graphs Settings

Title

Write Preview AA B i “ <> ↻ ⋮ ⋮ ✓ ↶ @ ★

# 清华大学  
## 电子系科协  
### 软件部

Attach files by dragging & dropping, selecting them, or pasting from the clipboard.

**M** Styling with Markdown is supported

Submit new issue

# GitHub使用


- 什么是**Milestone**?
- 顾名思义，用于记录项目的进度
- 可以自己创建Milestone
- 可以将Issue加入Milestone
- 进度由完成的Issue数量决定

The screenshot shows the 'New milestone' form in GitHub. At the top, there are tabs for 'Code', 'Issues 2', 'Pull requests 0', 'Wiki', 'Pulse', 'Graphs', and 'Settings'. Below the tabs, the title 'New milestone' is followed by a subtitle: 'Create a new milestone to help organize your issues and pull requests. Learn more about [milestones and issues](#).' The form has two main sections: 'Title' and 'Description'. The 'Title' section has a text input field with the placeholder text '[Title]'. The 'Description' section has a large text area. To the right of the 'Title' field, there is a 'Due Date (optional) clear' section with a calendar widget. The calendar shows the month of July 2016, with the 2nd day selected. At the bottom right of the form, there is a green button labeled 'Create milestone'.

The screenshot shows the 'Milestones' tab in a GitHub repository. At the top, there are tabs for 'Code', 'Issues 2', 'Pull requests 0', 'Wiki', 'Pulse', 'Graphs', and 'Settings'. Below the tabs, there are buttons for 'Labels' and 'Milestones', with 'Milestones' being the active tab. To the right of these buttons is a green button labeled 'New milestone'. Below the buttons, there is a section for the selected milestone. It shows '1 Open' and '0 Closed' issues. The milestone title is 'lecture'. Below the title, there is a progress bar that is 66% complete. To the right of the progress bar, it says '66% complete 1 open 2 closed'. Below the progress bar, there are links for 'Edit', 'Close', and 'Delete'. At the bottom, there is a date 'Due by July 7, 2016' and a clock icon with the text 'Last updated less than a minute ago'.

## 参考资料

- [Git教程 - 廖雪峰的官方网站]  
<https://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000>
- [官方git教程] <https://try.github.io/>
- [去年的ppt, 内容差不多]
- <https://github.com/eesast/Training/tree/master/git>



Git第二讲结束