ISLAND

随心随性

GitHub for windows使用教程（三）
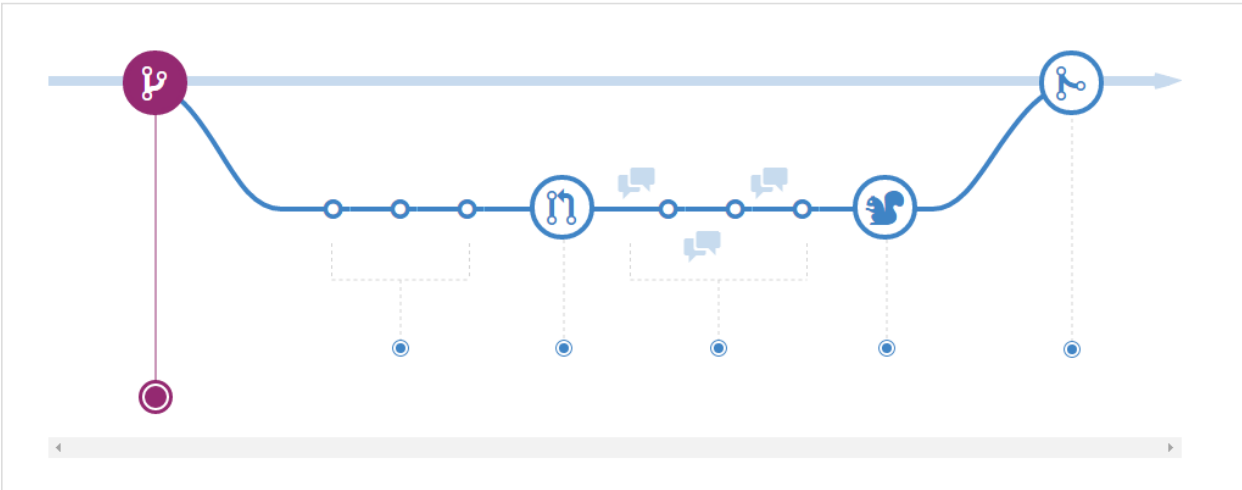
📅 2016-05-15　|　👁 5967
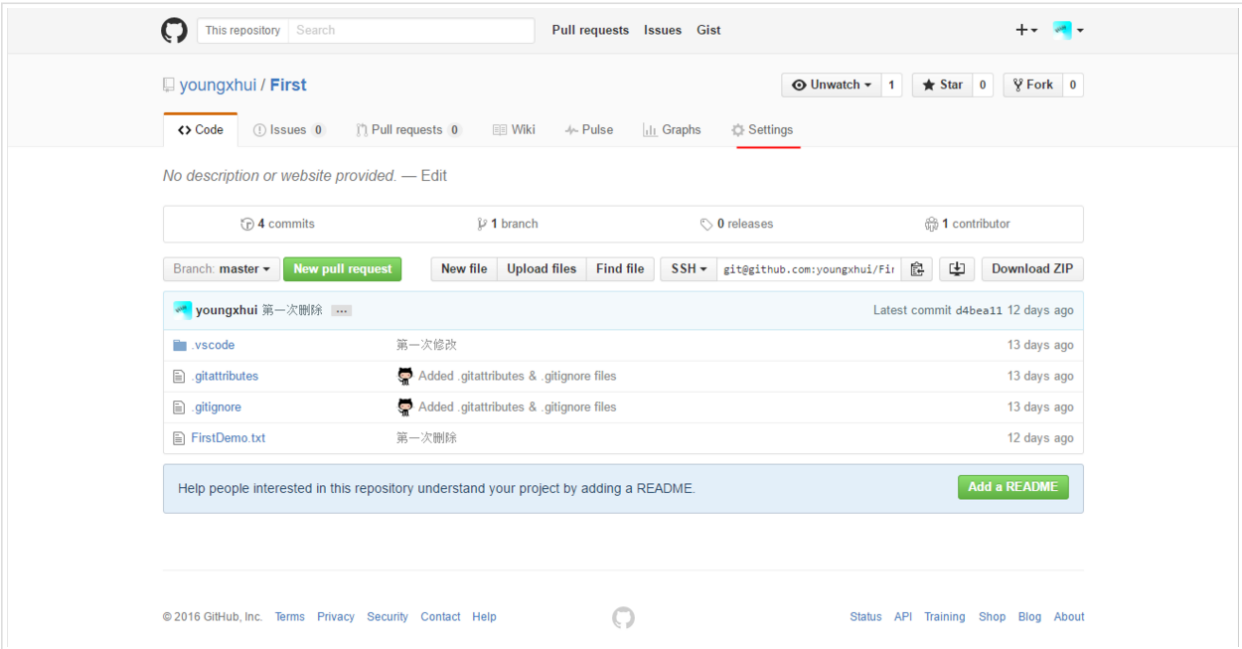
🗐 2,193

团队协作流程

认识Flow

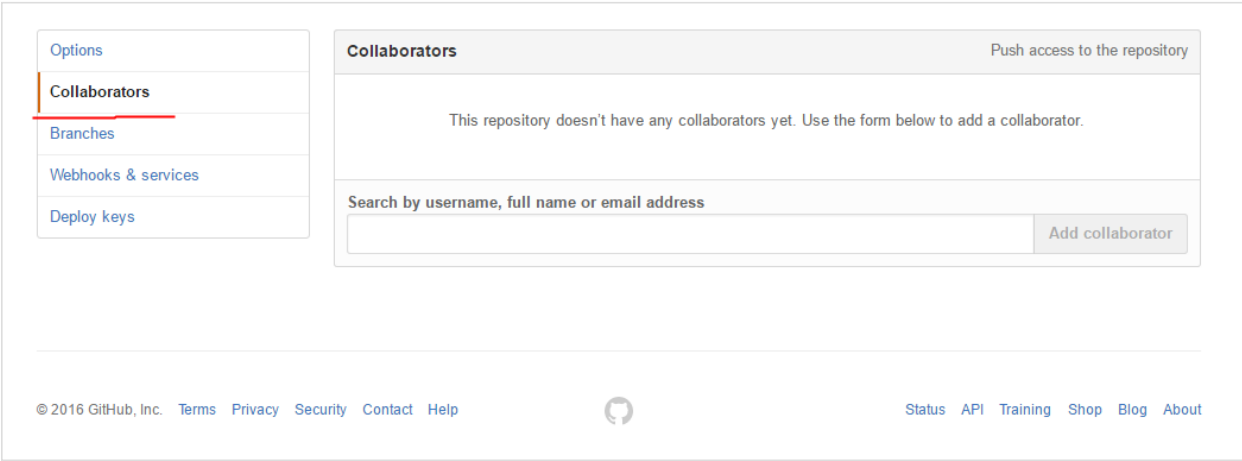**GitHub Flow**是一个轻量级的，基于分支的工作流程，支持团队和部署在那里的定期做项目。

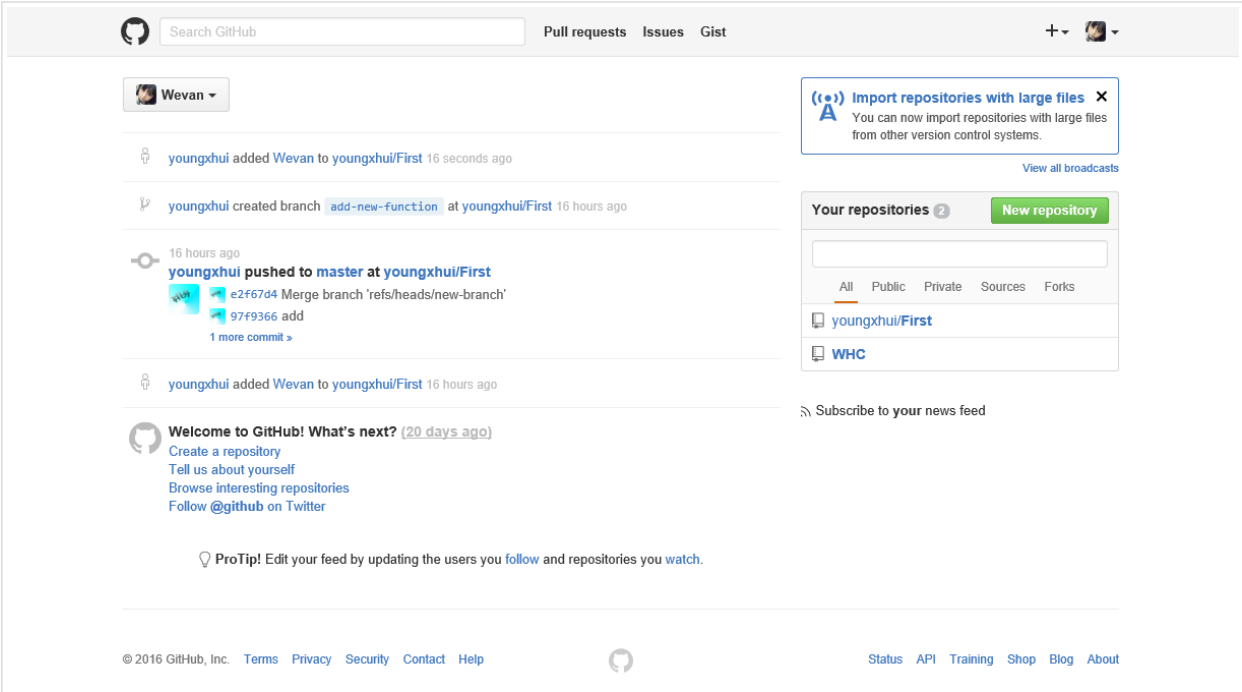
为团队成员写入权限

在我们的队友添加一个写的权限，这样我们的队友才能很好的修改代码。

我们打开网页上的GitHub,点击**settings**,

之后我们找到**collaborators**，这里会让我们验证密码，之后就有添加合作者的选项。这样我们就能添加我们的小伙伴了！

这样我们就添加了新的小伙伴，新的小伙伴有着同样的权限去修改和管理代码。
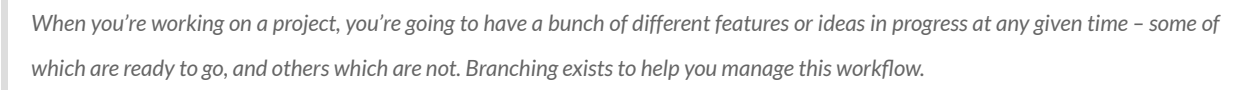
此时我们就会看到我的小伙伴wevan的github主页上就会出现关于我创建的First的各种通知。

创建分支

在我们创建一个叫**add new function**的分支。

创建一个分支

> *Create a branch*

当你工作的一个项目，你会在任何给定的时间有一堆不同的功能或正在进行的想法 - 其中一些是蓄势待发，而另一些则不是。分支的存在是为了帮助你管理这个工作流程。

> *When you're working on a project, you're going to have a bunch of different features or ideas in progress at any given time – some of which are ready to go, and others which are not. Branching exists to help you manage this workflow.*

当您在项目中创建一个分支，你创造一个环境，在那里你可以尝试新的想法。你让一个分支的更改不会影响主分支，让你可以自由进行实验，并提交更改，在你的分支将不会被合并，直到它准备好知识安全的人所正在与合作进行审查。

When you create a branch in your project, you're creating an environment where you can try out new ideas. Changes you make on a branch don't affect the master branch, so you're free to experiment and commit changes, safe in the knowledge that your branch won't be merged until it's ready to be reviewed by someone you're collaborating with.

ProTip

分支在Git中是一个核心概念，整个GitHub的流量是基于它。这里只有一个规则：在任何主分支总是部署。

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.

正因为如此，这是非常重要的一个功能或修复工作时，你的新分支关老爷的创建。您的分支名应该是描述（例如，重构的身份验证，用户的内容缓存键，使视网膜-化身），以便其他人可以看到正在处理。
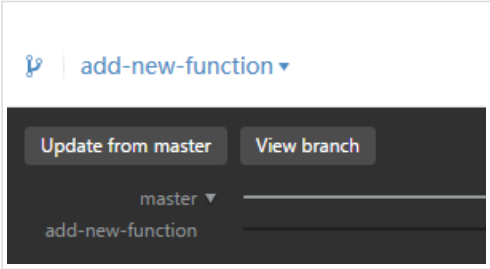
Because of this, it's extremely important that your new branch is created off of master when working on a feature or a fix. Your branch name should be descriptive (e.g., refactor-authentication, user-content-cache-key, make-retina-avatars), so that others can see what is being worked on.
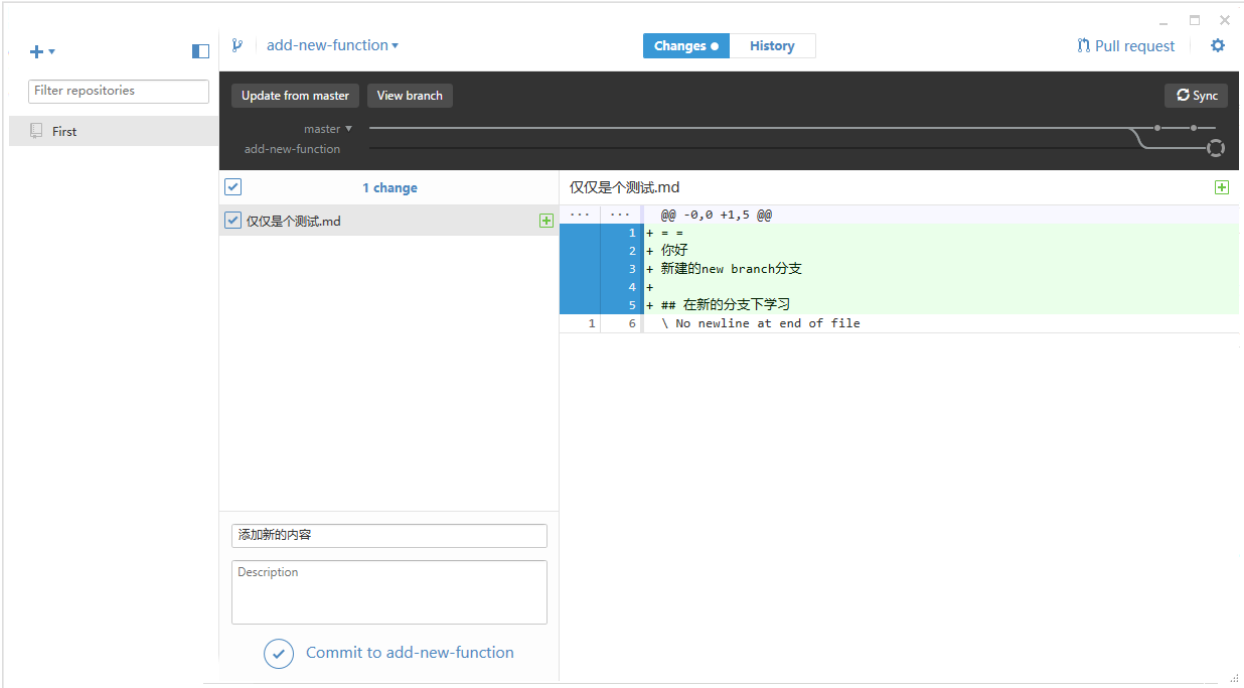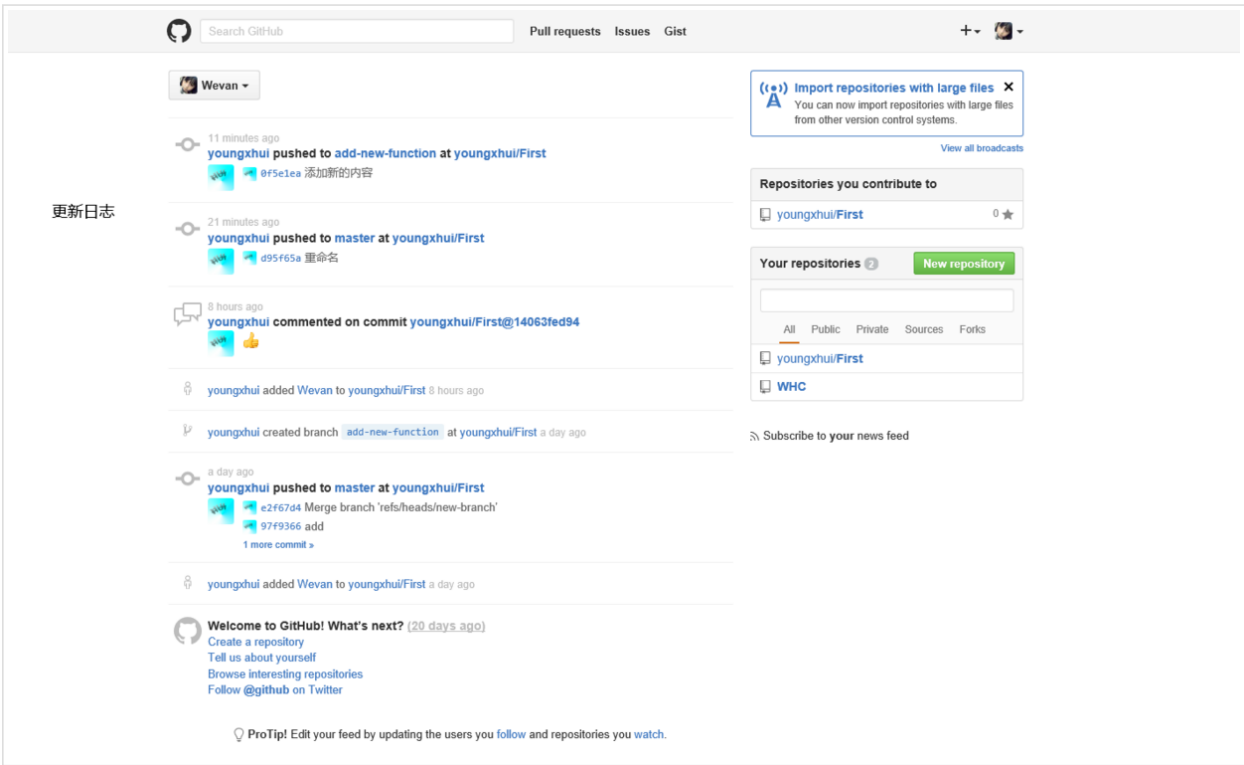
来自GitHub Flow

添加提交
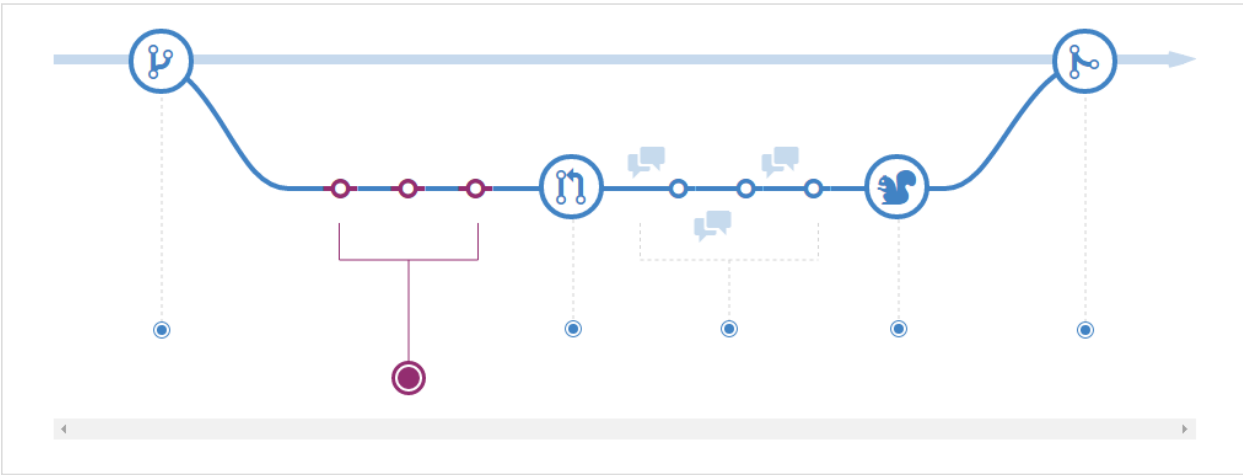
我们首先把分支切换到新的分支上**add new function**

修改新的版本

填写好新的**Summary**和**Description**，提交新的版本并同步。

这样小伙伴登陆到**GitHub**上就看到了就可以清楚的看到一切的修改。

添加提交

> *Add commits*

一旦你的分支已经建立，现在是时候开始进行更改。无论何时添加，编辑或删除一个文件，你作出承诺，并将其添加到您的分支。提交加入这一过程保持你的进步轨迹，你在一个特性分支工作。

> *Once your branch has been created, it's time to start making changes. Whenever you add, edit, or delete a file, you're making a*
>
> *commit, and adding them to your branch. This process of adding commits keeps track of your progress as you work on a feature*
>
> *branch.*

还承诺创建工作的透明历史，其他人可以按照理解你做了什么，以及为什么。每次提交都有一个关联的提交信息，这是解释为什么一个特定的变化作出了说明。此外，每次提交被认为是变革的一个独立单元。这使您可以回滚的变化，如果发现错误，或者如果你决定在一个不同的方向前进。

> *Commits also create a transparent history of your work that others can follow to understand what you've done and why. Each commit has an associated commit message, which is a description explaining why a particular change was made. Furthermore, each commit is considered a separate unit of change. This lets you roll back changes if a bug is found, or if you decide to head in a different direction.*

ProTip

提交信息是重要的，特别是因为Git跟踪更改，然后将它们显示为承诺一旦他们推到服务器。通过字迹清晰提交信息，你可以更容易为其他人跟着，并提供反馈。
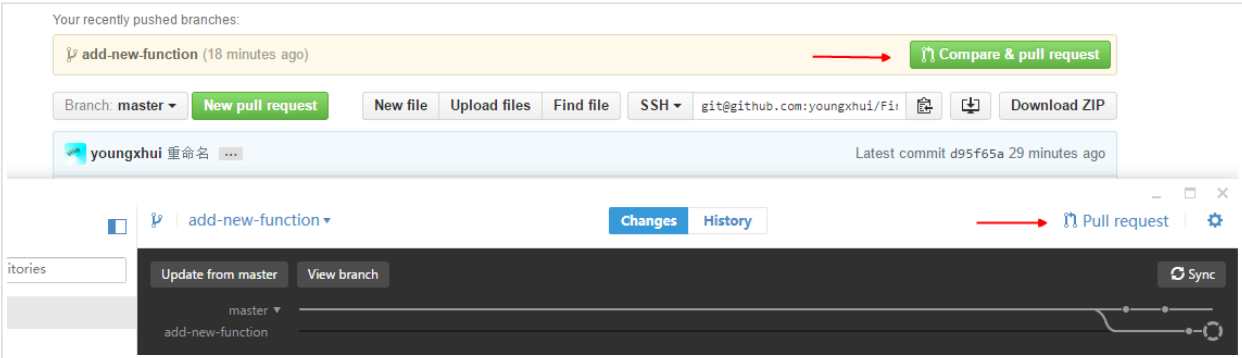
> *Commit messages are important, especially since Git tracks your changes and then displays them as commits once they're pushed to the server. By writing clear commit messages, you can make it easier for other people to follow along and provide feedback.*
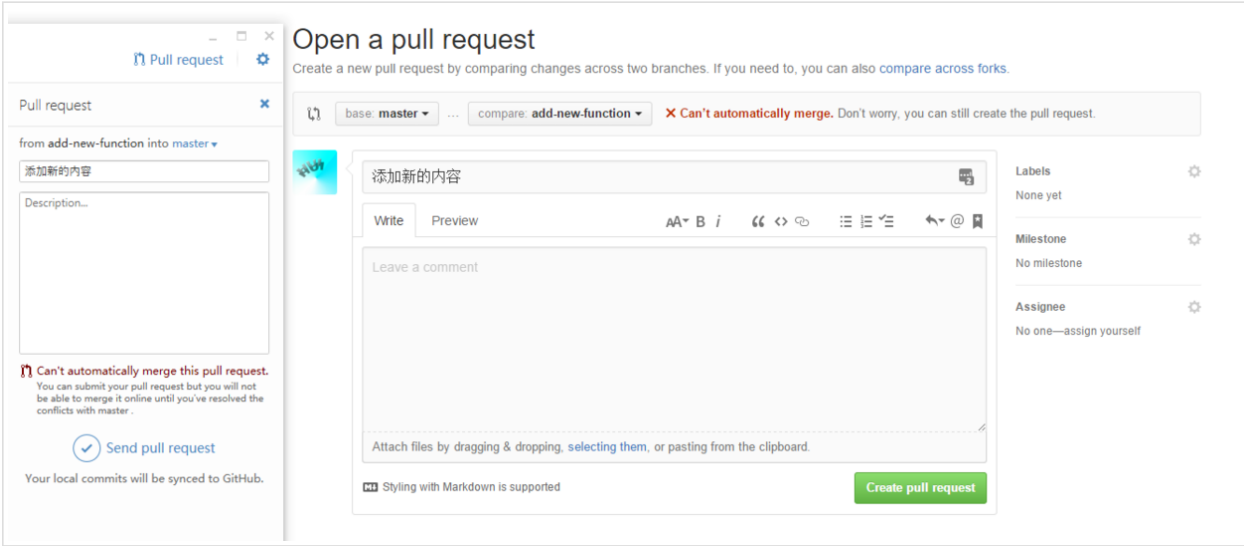>
> **来自GitHub Flow**

打开一个pull请求
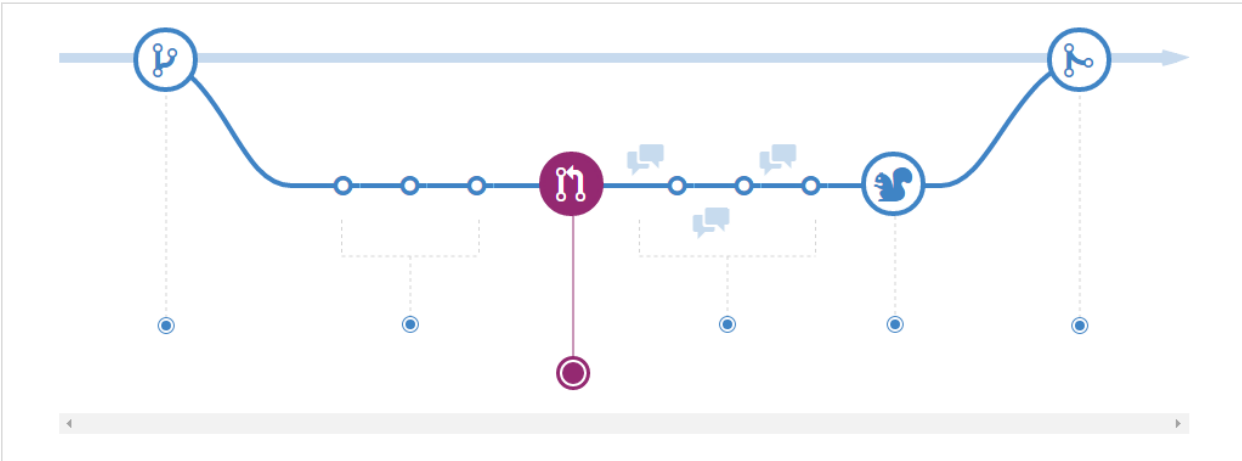
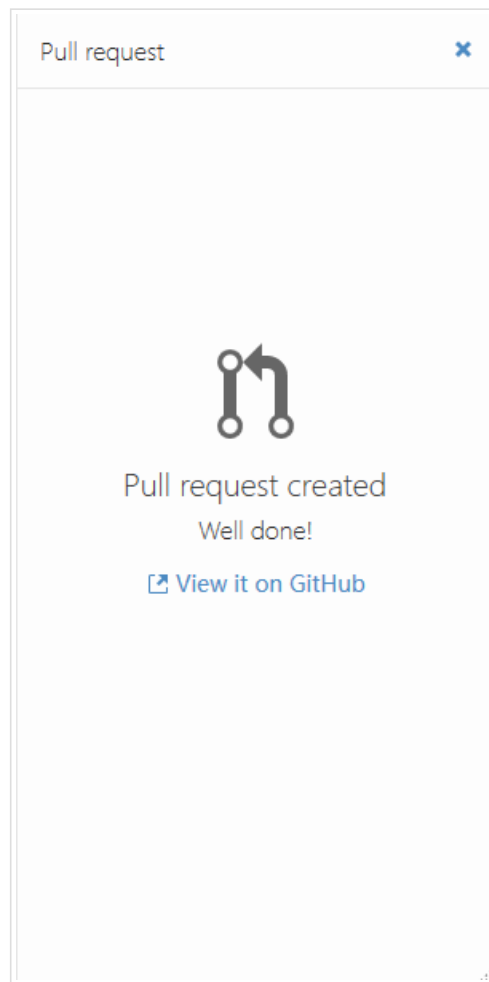这个是整个流程中比较关键的一步，发布**Pull Request**。

点击客户端或者网页上的**Pull Request**发布。

我们这里点击**Pull Request**

我们填写好必要的说明性文字

点击**Send Pull Request**

他既然让我们到GitHub上看，我们就听他的，点击，进入。

我们发现小伙伴已经在下面留言了！

讨论和审核你的代码

你的小伙伴开始对你的代码讨论，修改，迭代。

讨论和审查你的代码

Discuss and review your code

一旦拉入请求已被打开，人或团队审查您的变化可能有疑问或意见。也许编码风格不匹配项目的指导方针，改变缺少单元测试，或者也许一切看起来不错，道具都是为了。引入请求旨在鼓励并捕获这种类型的对话。

> *Once a Pull Request has been opened, the person or team reviewing your changes may have questions or comments. Perhaps the coding style doesn't match project guidelines, the change is missing unit tests, or maybe everything looks great and props are in order. Pull Requests are designed to encourage and capture this type of conversation.*

您还可以继续推送到你的分支在你提交的讨论和反馈光。如果有人评论说，你忘了做某件事，或者如果在代码中的错误，你可以在你的分支修复它，推高的变化。GitHub上会显示新的提交和其他任何意见，你可能会收到统一拉请求视图。

> *You can also continue to push to your branch in light of discussion and feedback about your commits. If someone comments that you forgot to do something or if there is a bug in the code, you can fix it in your branch and push up the change. GitHub will show your new commits and any additional feedback you may receive in the unified Pull Request view.*

ProTip

拉请求的意见都写在降价，所以你可以插入图片和表情符，使用预先格式化的文本块，等轻质格式。

> *Pull Request comments are written in Markdown, so you can embed images and emoji, use pre-formatted text blocks, and other lightweight formatting.*
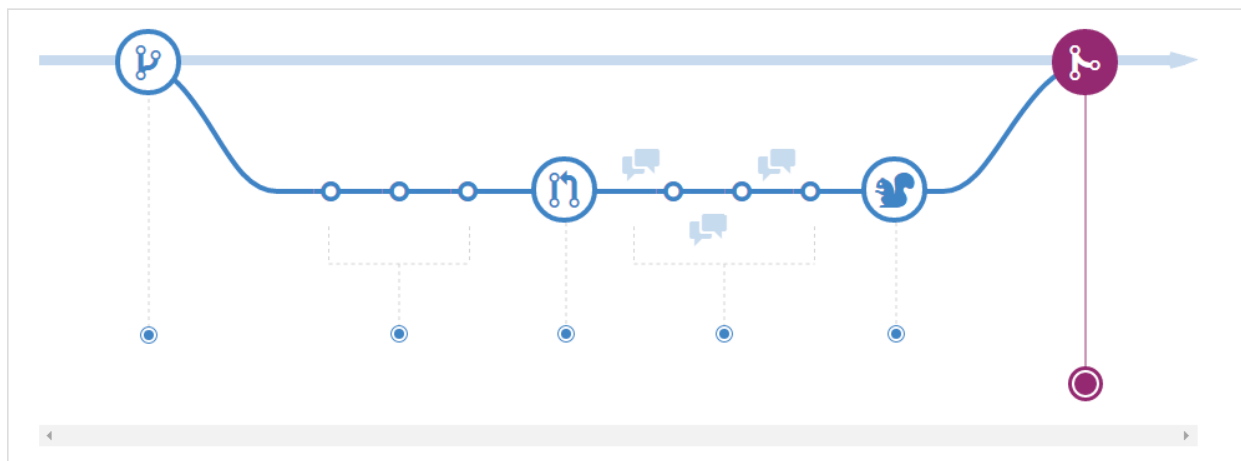
部署
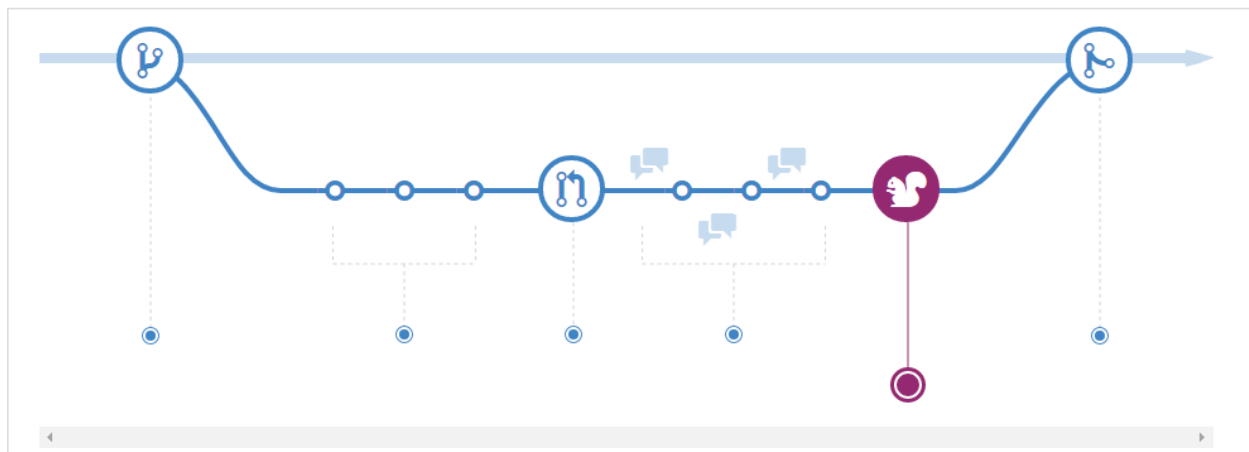
部署

> *Deploy*

一旦你拉的请求进行了审查和部门通过你的测试，您可以部署您的更改，以验证他们的生产。如果你的分支造成的问题，您可以通过部署现有的主投产回滚

> *Once your pull request has been reviewed and the branch passes your tests, you can deploy your changes to verify them in production. If your branch causes issues, you can roll it back by deploying the existing master into production.*

合并

合并分支我们之前已经说过，这里就不再赘述。

合并

> *Merge*

现在，您的更改在生产中得到了验证，现在是时候你的代码合并到主分支。

> *Now that your changes have been verified in production, it is time to merge your code into the master branch.*

合并后，引入请求保护的历史变迁到您的代码记录。因为他们是搜索的，他们不让任何人回去的时间理解为什么以及如何决定了。

> *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.*

ProTip

通过将某些关键字到您的拉请求的文本，你可以用代码相关联的问题。当你拉入请求合并，相关问题也将被关闭。例如，输入短语关闭＃32将关闭在仓库中发行数量32。欲了解更多信息，请查看我们的帮助文章。

> *By incorporating certain keywords into the text of your Pull Request, you can associate issues with code. When your Pull Request is merged, the related issues are also closed. For example, entering the phrase Closes #32 would close issue number 32 in the repository. For more information, check out our help article.*

注意：英文翻译为机器翻译，可能有翻译错误的地方，建议大家尽可能看英文

总结
基本的GitHub教程就算写完了，已有如果在有就是一些GitHub上的一些使用小技巧了。

博主都要穷的吃土了☺

打赏

本文作者： 杨晓辉
本文链接： http://youngxhui.github.io/2016/05/15/GitHub-for-windows使用教程（三）/
版权声明： 本博客所有文章除特别声明外，均采用 CC BY-NC-SA 3.0 许可协议。转载请注明出处！

🏷GitHub

❮ GitHub for Windows使用教程(二)　　　　　　　　　　　Java集合遍历 ❯