

# 葱丝瓣酱

你真的需要呼吸吗?!

- [RSS](#)
- [Twitter](#)
- [GitHub](#)
- [blog](#)
- [archives](#)

## 使用GitHub进行团队合作

Mar 20th, 2013

原文: [Team Collaboration With GitHub](#)

---

[GitHub](#)已经成为的一切开放源码软件的基石。开发人员喜欢它，基于它进行协作，并不断通过它开发令人惊叹的项目。除了代码托管，[GitHub](#)的主要吸引力是使用它作为一个协作开发工具。在本教程中，让我们来看看一些最有用的GitHub的功能，特别是使团队工作更有效率，更高生产力，非常重要的，好玩的那些功能！

---

## GitHub和软件合作

有一件事我觉得非常有用的是，可以将GitHub的维基集成到项目的源代码主线上。

本教程假定您已经熟悉[Git](#) - 开放源码的分布式版本控制系统，由Linux的创世人[Linus Torvalds](#)在2005年创造的。如果您需要修改或查找有关[Git](#)，请访问我们以前的[截屏教程](#)，和一些[文章](#)。此外，你应该已经有一个[Github](#)上的帐户，并做了一些基本的功能，如创建一个存储库，并推送到[GitHub](#)上。如果没有，可以参照更多以前的[教程](#)。

在这个世界上的软件项目，不可避免的是，我们必须和一个团队一起工作来交付软件。在本教程中，我们将探索一些软件开发团队最常用的工具。这些工具包括：

- 添加团队成员 - 组织和合作者

- Pull请求 - 发送代码变更和合并
- 问题跟踪 - Github上的错误记录
- 分析 - 图形与网络
- 项目管理 - [Trello](#)与[Pivotal Tracker](#)
- 持续集成 - [Travis CI](#)
- 代码审查 - 代码行评论与URL查询
- 文档记录 - Wiki与Hubot

## 更喜欢截屏操作视频？

如果你倾向于观看截屏操作视频，可以观看下面的截屏操作视频，而将本教程作为旁注。



---

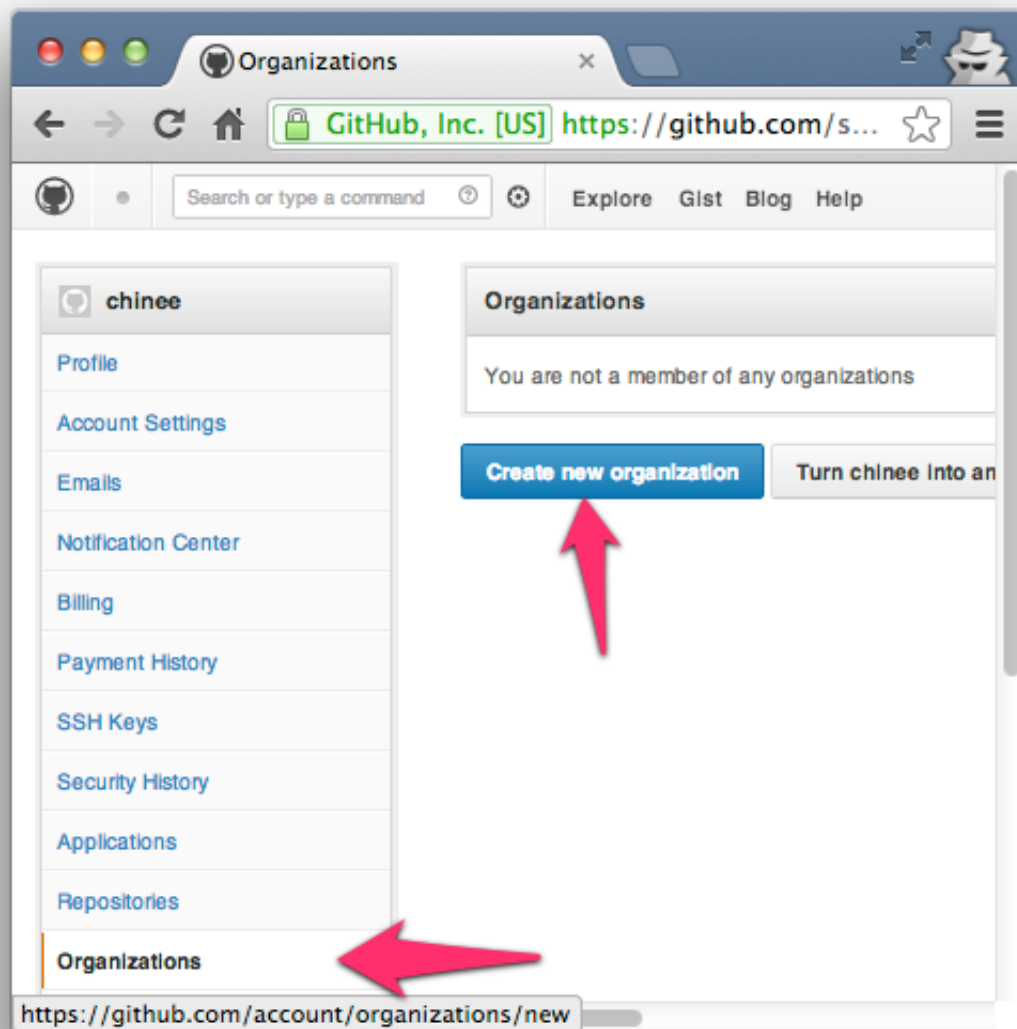
## 工具一：增加团队成员

有两种常用的方法在[GitHub](#)上建立团队合作：

- 组织 - 组织的所有者可以针对不同的代码仓库建立不同访问权限的团队。
- 合作者 - 代码仓库的所有者可以为单个仓库增加具备只读或者读写权限的协作者。

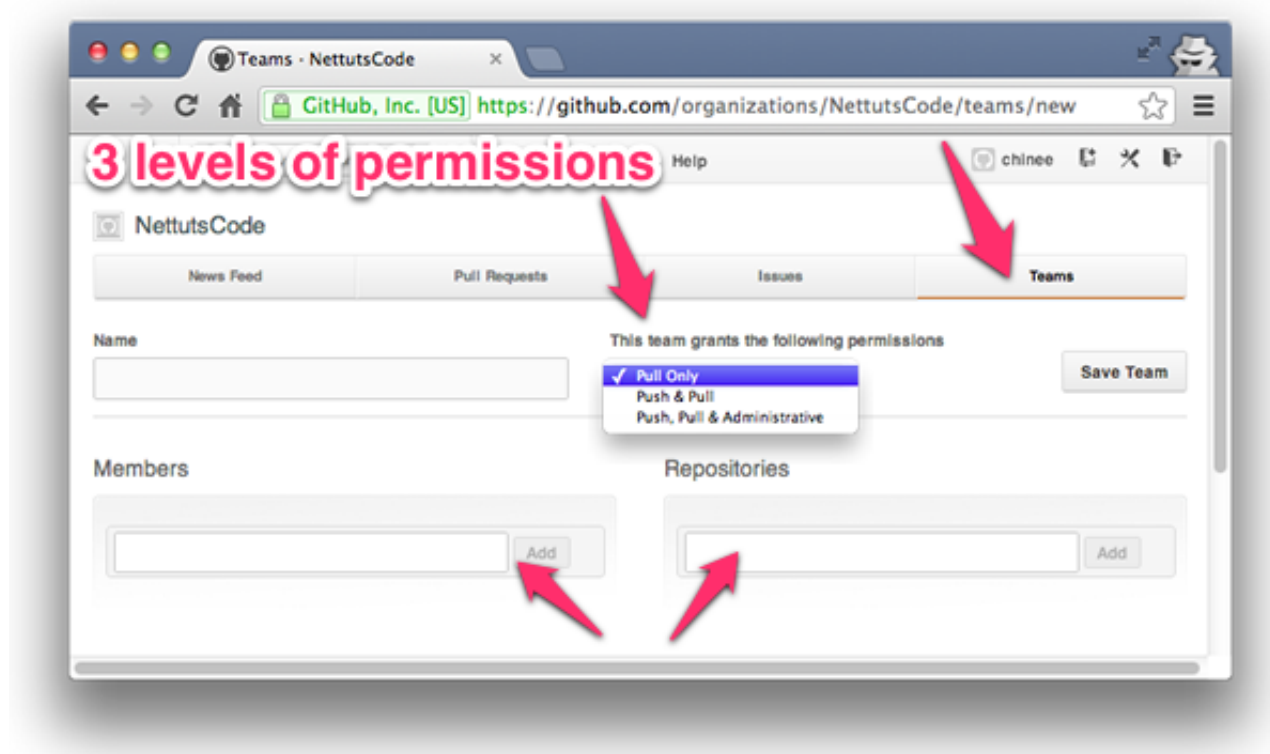
## 组织

如果您监管几个团队，想为每个团队设置不同的权限级别，或者为不同的代码仓库增加不同的成员组织(Organizations)将是最好的选择。任何GitHub用户帐户已经可以创建免费的开源代码库的组织。要创建一个组织，只需浏览您的[组织设置页面](#)：



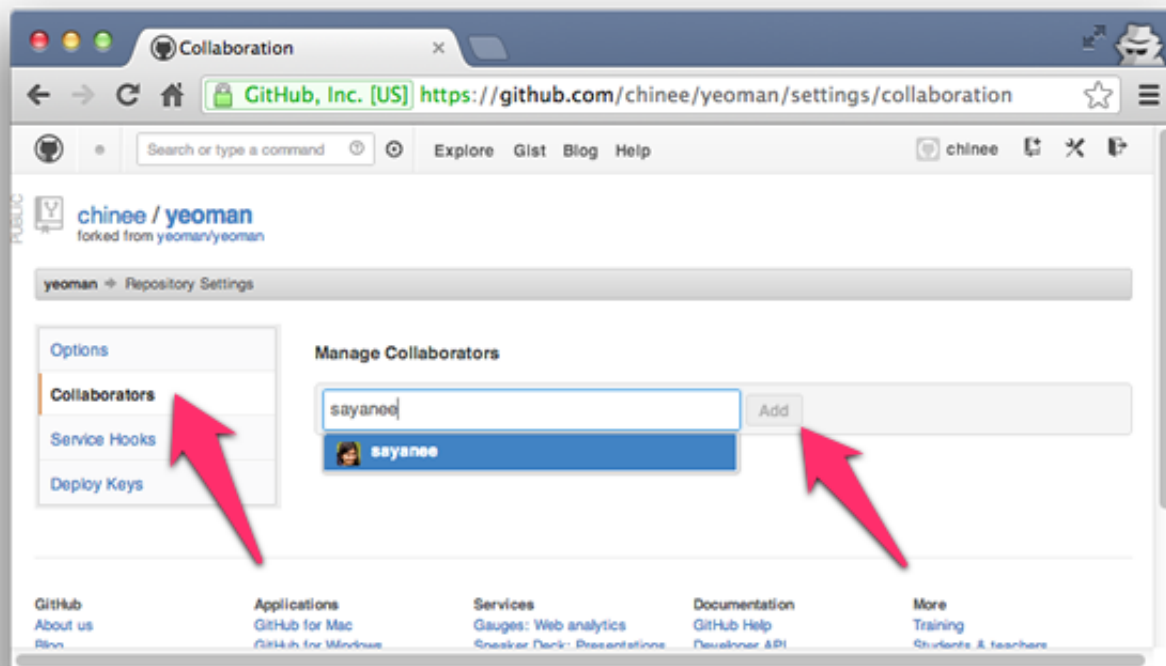
要访问组织的团队页面，你可以简单地去页面[http://github.com/organizations/\[组织名称\]/teams](http://github.com/organizations/[组织名称]/teams)来查看，或者访问页面[https://github.com/organizations/\[组织名称\]/teams/new](https://github.com/organizations/[组织名称]/teams/new)来创建新的具备3种不同的权限级别的团队成员，如：

1. Pull Only: [提取和合并](#)另一个库或本地副本。只读访问权限。
2. Push和Pull: (1)以及[更新](#)远程代码仓库。读+写访问权限。
3. Pull, Push和管理: (1), (2), 计费, 建立团队, 以及取消组织帐户。读+写+管理员权限



## 合作者

合作者主要用于读写访问个人账号所拥有的代码仓库。你可以通过[https://github.com/\[用户名\]/\[代码仓库名称\]/settings/collaboration](https://github.com/[用户名]/[代码仓库名称]/settings/collaboration)来增加[合作者](#)(其他github个人账号)。



一旦做到这一点，每个合作者将会看到代码库页面的访问状态的变化。在拥有对代码库的写访问权限后，我们可以做一个git克隆，进行代码变更，用git拉取和归并远程存储库中的任何变化，并最终将本地的变化git推送到远程代码库：



## 工具二：Pull请求(Pull request)

Pull请求是一个非常棒的方式，通过fork一个新的代码库用来独立开发，并将变更贡献回原始代码库。在一天结束的时候，如果我们愿意，我们可以发送一个pull请求给代码库所有者，来合并我们的代码更改。Pull请求本身可以引起合作者之间的评论，包括代码质量，功能，甚至总体战略等。

现在让我们浏览一个[pull请求](#)的基本步骤。

## 发起一个Pull请求

GitHub有两种Pull请求方式：

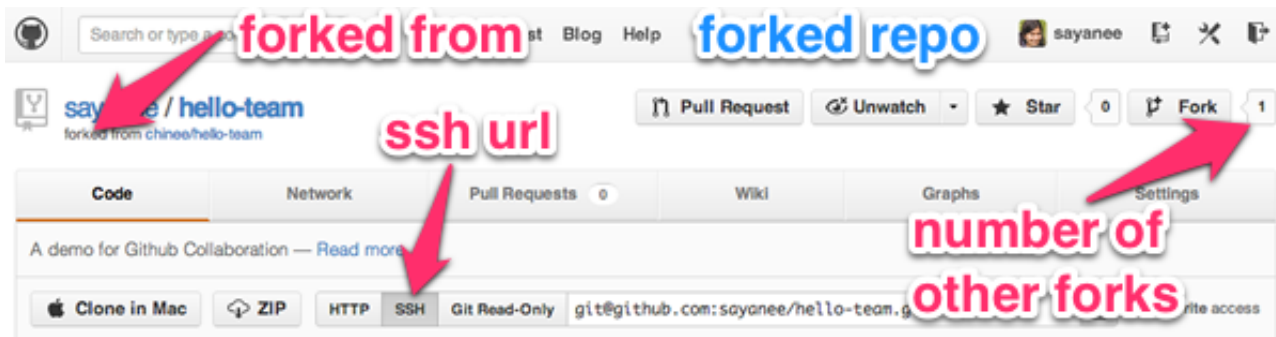
1. Fork & Pull 方式 - 用于在公共库中，我们没有推送(push)权限。
2. 共享库方式 - 用于私有代码仓库，我们有推送(push)权限。这种情况下没有必要进行fork。

下面的工作流程是在两个用户(原始代码库拥有者，和fork代码库拥有者)之间的fork-pull方式：

1. 进入你想贡献修改的GitHub代码库，单击“Fork”按钮来创建自己的Github帐户上的代码库克隆：



2. 这将在自己的帐户上创建一个该代码库的复制：



3. [选择 SSH URL](#)，那样它会自动使用你自己的SSH密钥，而不用每次在git pull或者push时询问你的用户名和密码。下一步，我们将克隆一份代码库到本地计算机：

```
$ git clone [ssh-url] [folder-name]
$ cd [folder-name]
```

4. 一般情况下，每一个新的功能，我们将创建一个新的Git分支。这是一个很好的做法，因为在未来，如果经过一番讨论后我们需要进一步更新分支，[Pull请求将被自动更新](#)。让我们创建一个新的分支做一个非常简单的变化修改的readme.md文件：

```
$ git checkout -b [new-feature]
```

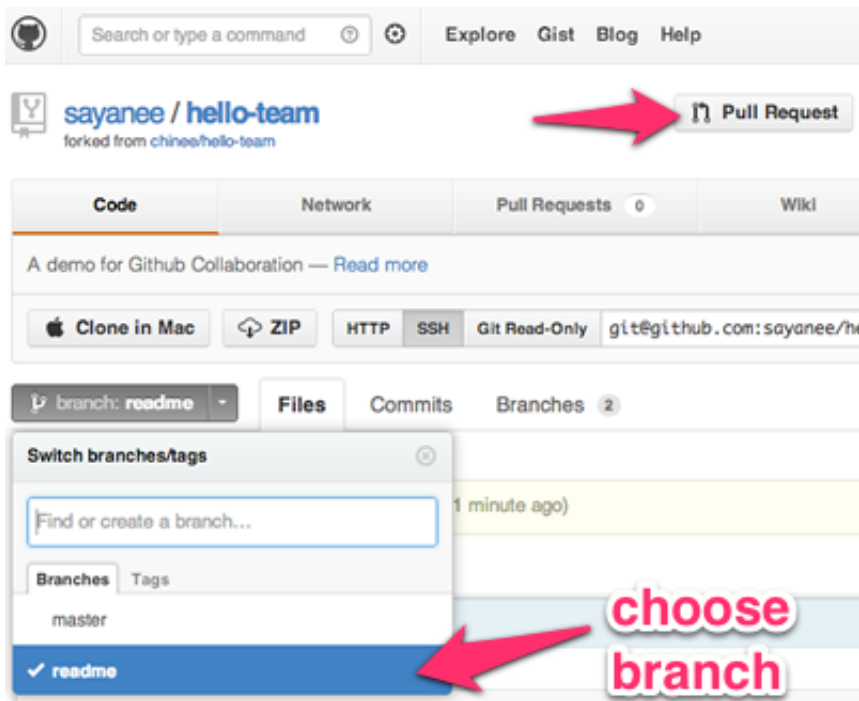
5. 在为这个新功能增加文件后，我们只需要将修改提交到这个新分支上，然后切换回master分支：

```
$ git add .  
$ git commit -m "information added in readme"  
$ git checkout master
```

6. 在这里，我们需要将新分支推送到远程代码仓库里。首先，我们需要检查这个新功能的分支名称以及其在远程仓库的别名，然后我们用git push [git-remote-alias] [branch-name]推送这个变更。

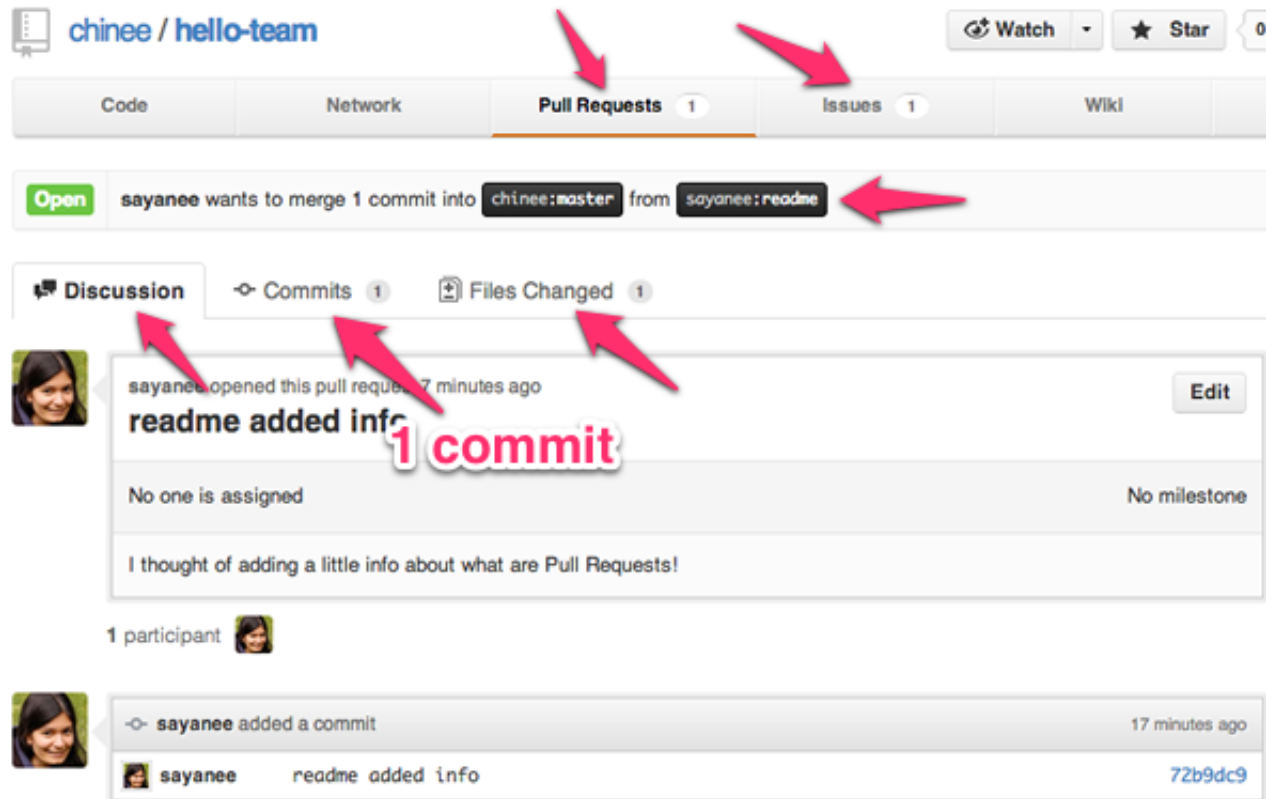
```
$ git branch  
* master  
readme  
$ git remote -v  
origin  git@github.com:[forked-repo-owner-username]/[repo-name].git (fetch)  
origin  git@github.com:[forked-repo-owner-username]/[repo-name].git (push)  
$ git push origin readme
```

7. 进入我们fork的代码库的GitHub页面，选择为这个新功能建立的分支，然后点击Pull Request按钮：

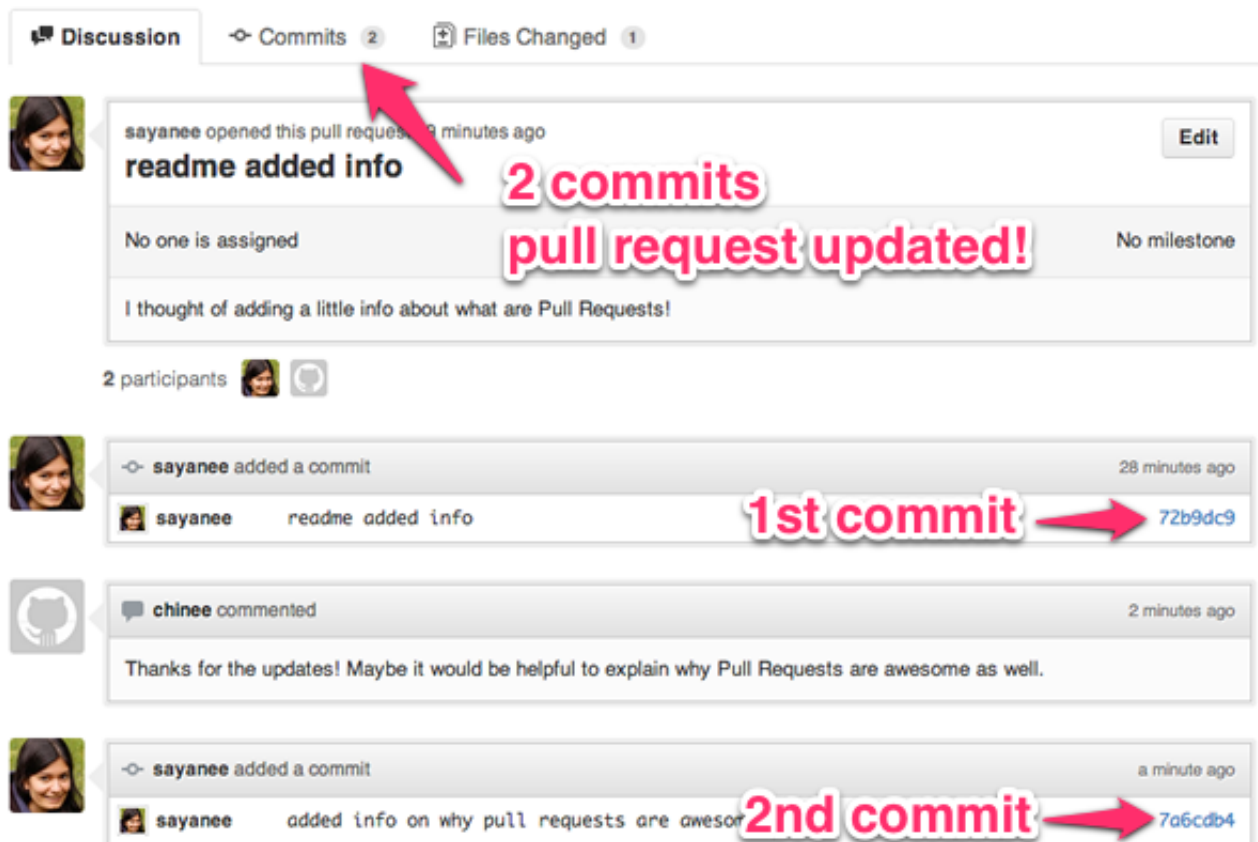


8. 提交Pull请求后，页面将直接跳转到原始库的Pull请求页面，我们将看到我们提交的Pull请求，作为一个新的问题，以及作为一个新的pull请求。





9. 在经过讨论后，fork的代码库的作者可能想为这个新功能增加一些新的改动。在这种场景下，我们需要在本地计算机上checkout这个同样的分支，修改，提交，并推送回GitHub。当我们再次访问原代码库的pull请求页面的时候，会发现上次提交的Pull请求已经自动更新了。



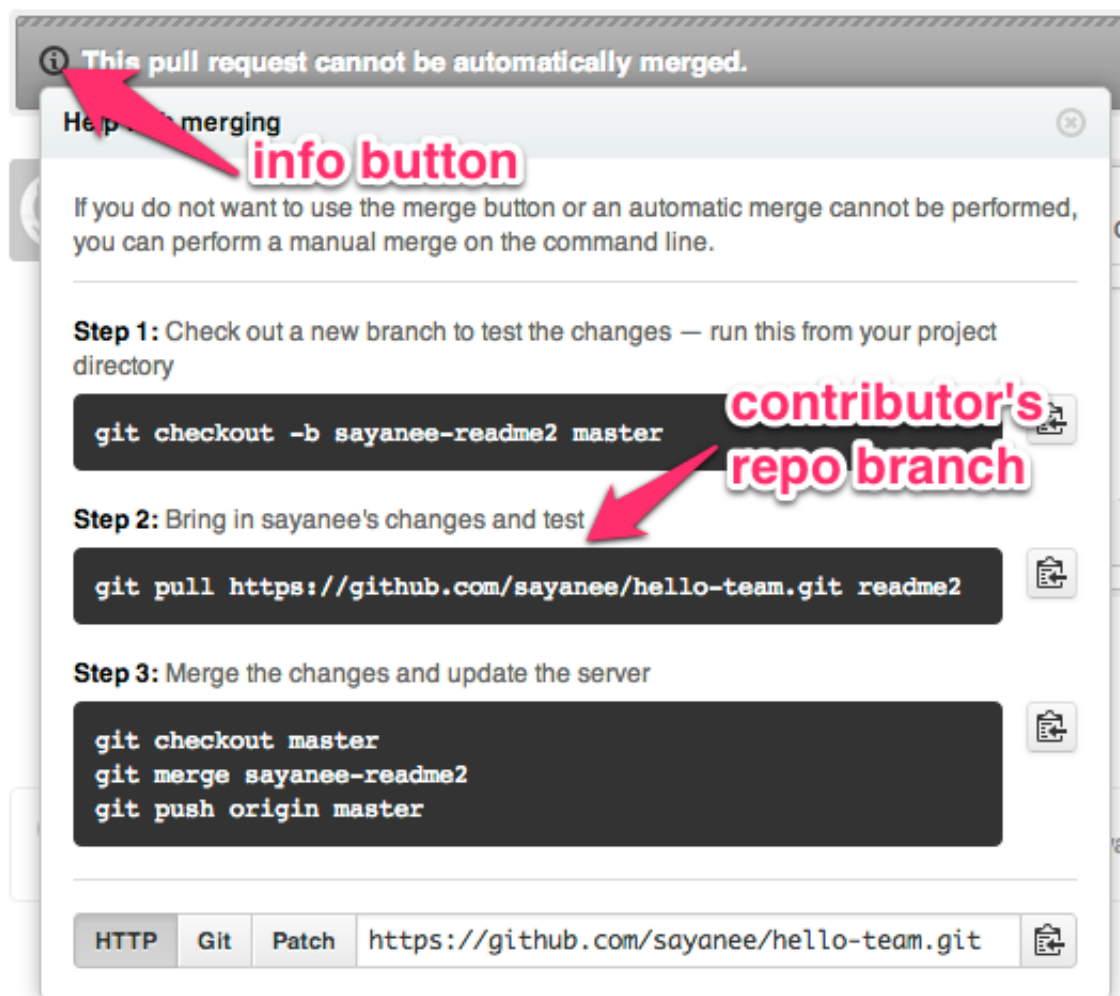
## 合并一个Pull请求

如果你是原始代码库的所有者，你将有[两种方式](#)来合并收到的Pull请求。

1. 直接在GitHub上合并：如果我们想直接在GitHub上进行合并，必须确保没有冲突。原始库的所有者可以通过简单地点击Merge Pull Request按钮来进行合并：



2. 在本地计算机上进行合并：另外一种情况，合并的时候可能会遇到冲突，点击上部的Info图标，GitHub有非常清晰的指导，怎么从贡献者的分支上下拉代码变更到本地，合并并解决冲突。



在软件开发团队中有很多不同的代码分支模型。这里有两种非常常用的工作流程模型：

- [简单分支模型](#)以及Pull请求；
- [更加广泛的分支模型](#)。

至于采用何种分支模型，取决于团队，项目，以及当时的状态。

---

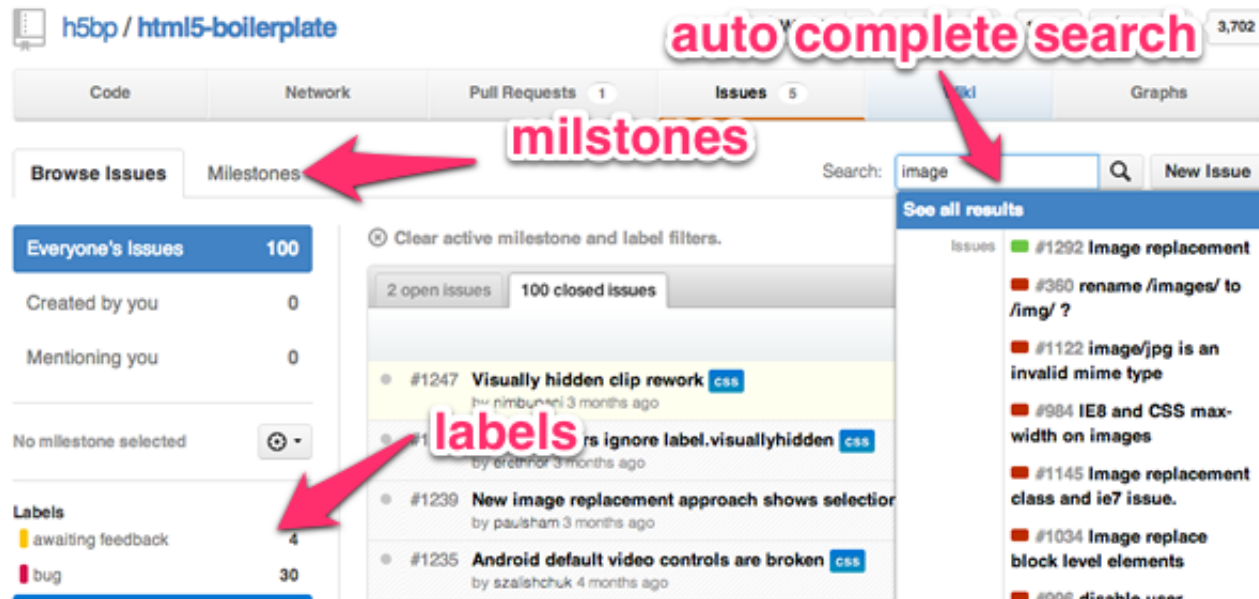
## 工具三：错误跟踪

在GitHub中，缺陷跟踪的中心是*问题列表(issues)*。虽然问题列表主要是为了跟踪缺陷，但我们经常会用以下面的方式：

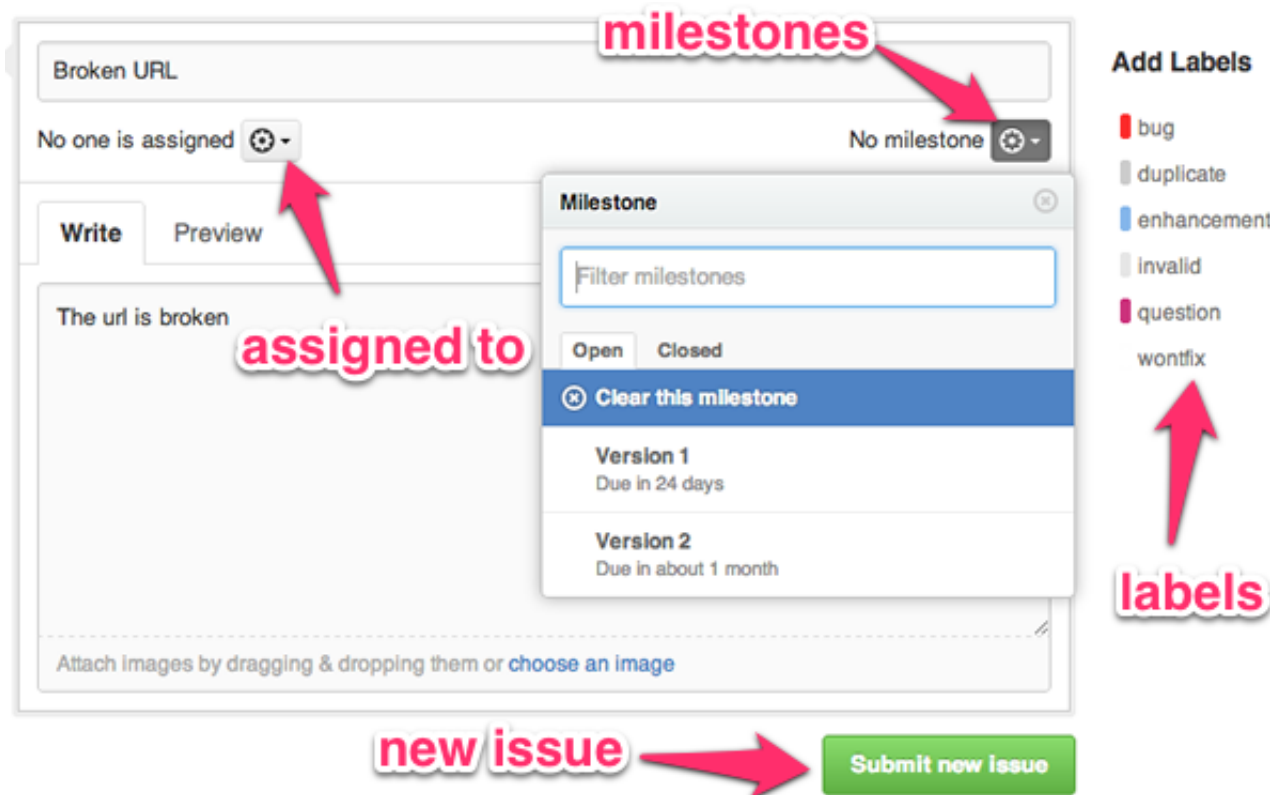
- **缺陷**：软件中显然坏了的行为，需要进行修正的地方。
- **功能**：需要实现的很酷很棒的新点子。
- **待完成清单**：待完成的检查清单。

让我们来探讨*问题列表*的以下特点：

1. **标签**：具有不同颜色的类别，用来帮助过滤问题。
2. **里程碑**：附加在每一个问题上的日期分类，可用于确定哪些问题需要在下一个版本解决。此外，由于每个问题都定有里程碑，每当一个问题解决，它会自动更新进度条。
3. **搜索**：搜索时能自动列出匹配的问题列表和里程碑。



4. 分配：每个问题都能分配一个人负责进行解决，同时这也能让我们知道目前我们需要工作在什么上面。



5. 自动关闭：包含Fixes/Fixed/Close/Closes/Closed #问题编号的提交记录，将自动关闭该问题。

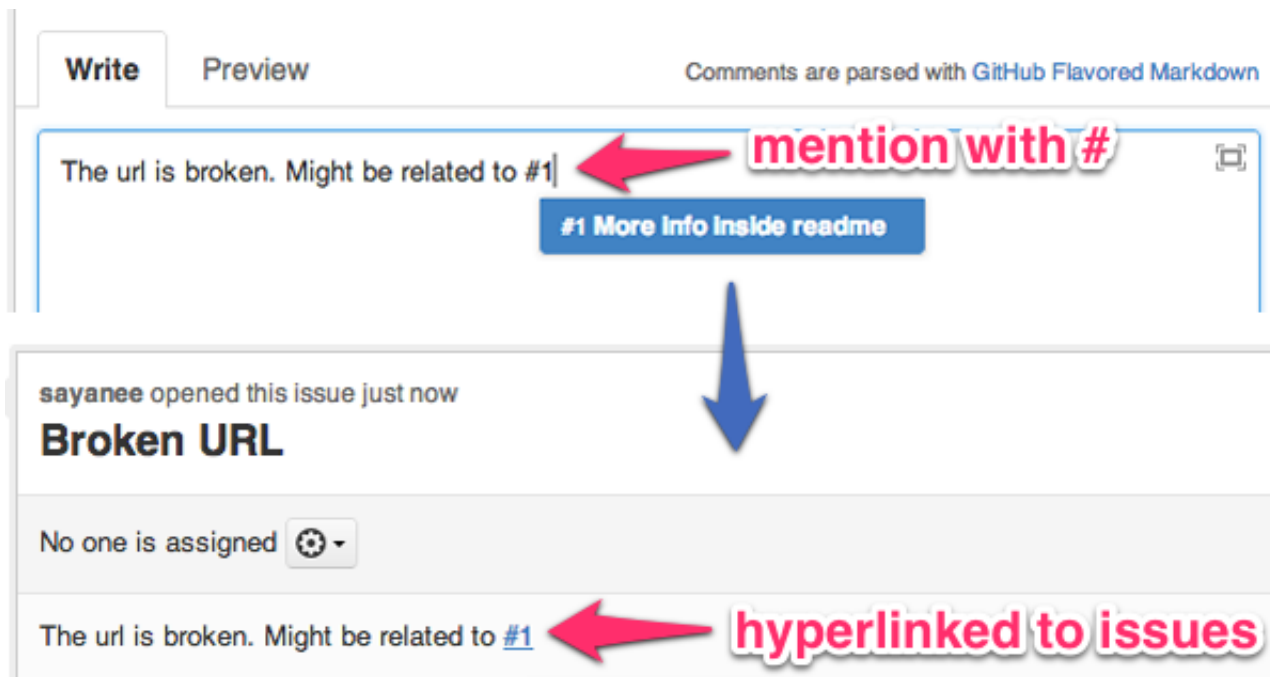
```
$ git add .
$ git commit -m "corrected url. fixes #2"
$ git push origin master
```



很显然，我们能将任务清单与代码提交紧密地耦合在一起。

6. 提及或者引用：任何人在评论的时候在消息文本中包含#[问题编号]，将自动生成该问题的链接，使得在讨论的过程中能非常容易地提

及相关的问题。



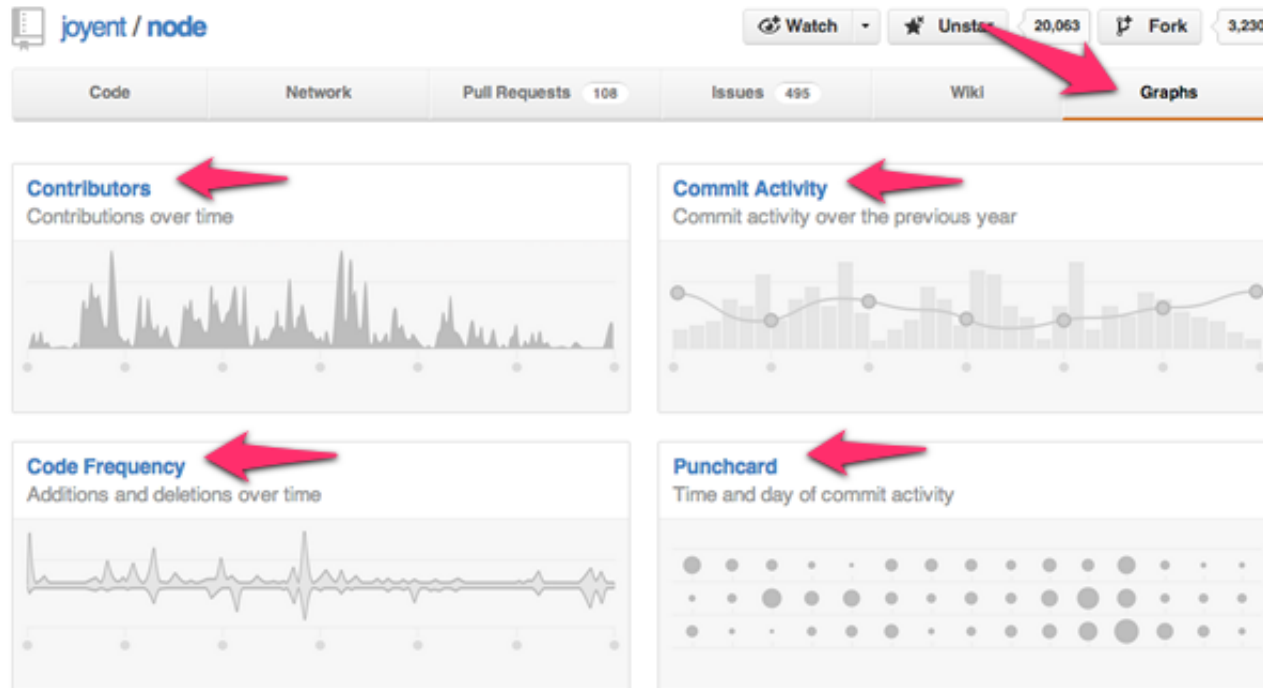
## 工具四：分析

有两个工具-图形和网络，让我们能洞察存储库的变化。[Github图](#) 提供了代码库的合作者，以及代码提交的直观展现，而[Github网络](#)可视化直观地展现了每一个贡献者和他们在所有分支上的代码提交。这些分析和图形非常强大，尤其是当在团队中工作。

### 图(Graphs)

图提供了详细的分析，包括：

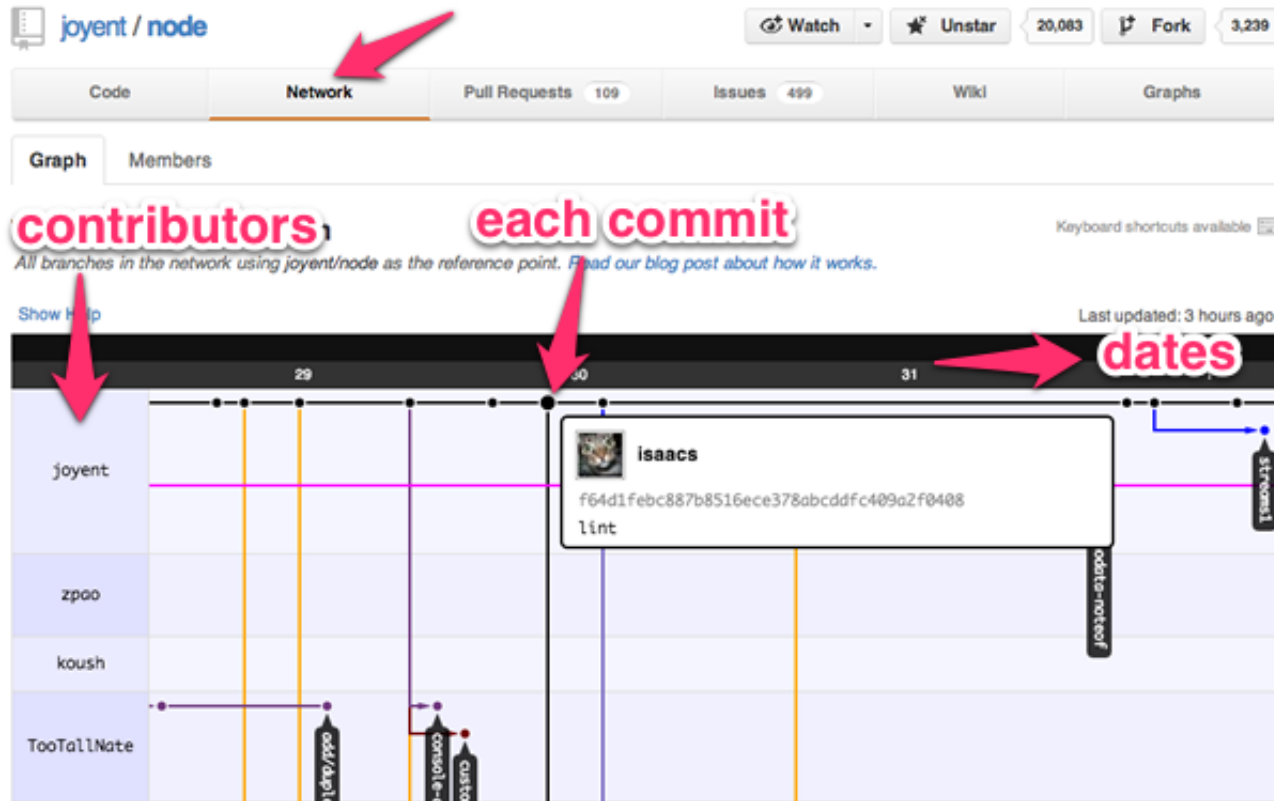
- 贡献者：有哪些代码提交者？他们增加或者删除了多少代码行？
- 代码提交活动：在过去的一年中，这些代码提交主要发生在哪些周？
- 代码频率：在整个项目的生命周期的不同阶段，提交了多少代码行？
- 记录卡：代码提交通常发生在每一天的什么时候？



## 网络(Network)

[GitHub网络\(Network\)](#)是一个非常强大的工具，让我们能看到每一个贡献者的代码提交，以及这些提交与其他的提交有什么关联。当我们作为一个整体观看这个网络的可视化展现时，我们能看到每一个库，每一个分支，和每一个提交，





## 工具五：项目管理

GitHub上的 [问题列表](#) 可以定义问题和里程碑，具有一定的项目管理能力。因为其他的某些功能或现有的工作流程，有些团队可能会更倾向于另外的工具。在本节中，我们将看到我们如何连接Github与其他流行的项目管理工具 - [Trello](#) 和 [Pivotal Tracker](#)。使用GitHub的服务钩子(hooks)，我们可以将代码提交，问题和许多其他活动自动更新到任务中。对于任何软件开发团队，这种自动化的帮助，不仅节省了时间，而且还可以提高更新的。

### GitHub和Trello

[Trello](#) 提供了一直简单而直观的方式管理任务。使用 [敏捷开发](#) 的方式，Trello任务卡能模拟简单，可视化的虚拟 [任务看板](#)。作为实例，当GitHub代码库收到一个Pull请求的时候，我们将利用GitHub的钩子(Hooks)服务，自动在Trello里生成一个任务卡。让我们来看看实现这个功能的具体步骤：

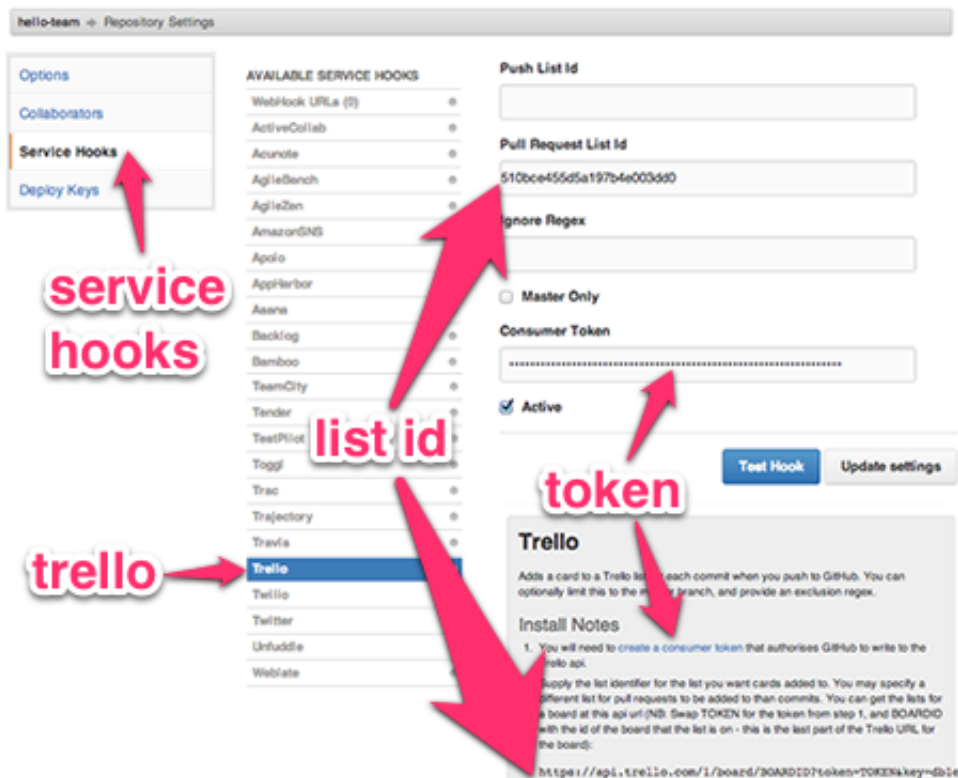
1. 首先注册一个Trello帐号，并建立一个新的Trello看版(Board)。



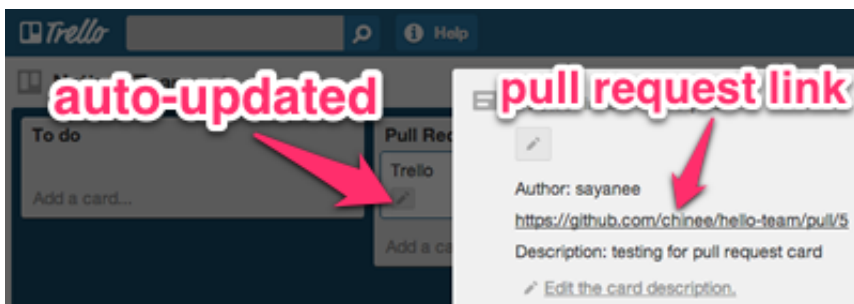
2. 访问GitHub repository > Settings > Service Hooks > Trello。
3. 通过Install Note #1描述的方法获得认证用的令牌(Token)。
4. 通过Install Note #2给的链接获得json格式的任务列表(list) id。BOARDID是我们访问网站看版的URL中的一部分https://trello.com/board/[BOARD-NAME]/[BOARDID]。



5. 回到GitHub的钩子服务，输入我们得到的list id和token，激活这个hook，可以通过Test Hook按钮来测试每次收到新的Pull请求时，是否自动更新看版内容。



6. 当下次收到新的Pull请求时，Trello看板就会自动生成一个Pull请求任务卡。



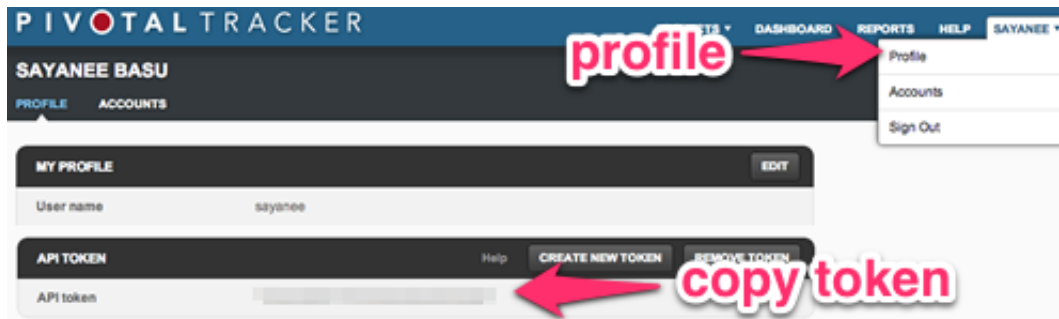
## GitHub和Pivotal Tracker

[Pivotal Tracker](#)是另外一个轻量级的项目管理工具，通过基于故事(story)的计划，让组员能对任何变化和进度进行回应，非常容易进行协作开发。基于项目当前的进度，能生成可视化的图表来分析团队的开发速度，迭代burn-up图，以及当前发布的burn-down图。在下面的例子中，我们通过关联一个GitHub代码提交(commit)到故事(story)，自动地交付一个故事。

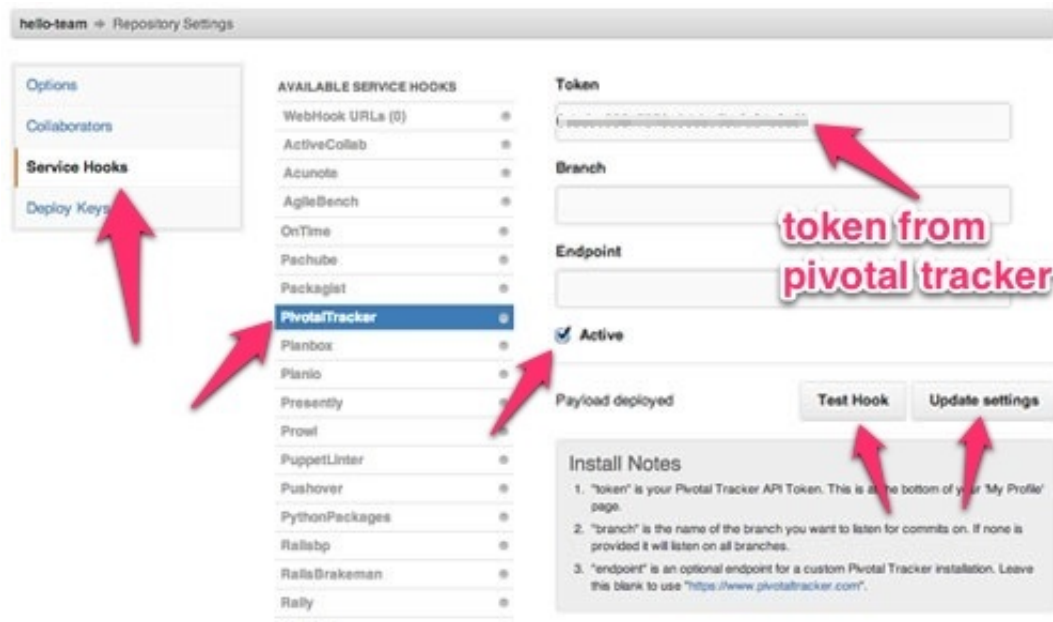
1. 在 [Pivotal Tracker](#) 上新建一个项目，并生成一个需要交付的故事(story)。



2. 访问 Profile > API Token, 复制给定的API令牌(token)。



3. 返回到GitHub页面访问 repository > Settings > Service Hooks > Pivotal Tracker, 粘贴刚才复制的token, 激活, 并保存设置。这样我们就能够在提交代码的时候自动交付对应的故事(story)。



4. 当我们最终提交代码修订的时候，按照格式`git commit -m "message [delivers #tracker_id]"` [将故事的id添加到提交记录里](#)。

```
$ git add .  
$ git commit -m "Github and Pivotal Tracker hooks implemented [delivers #43903595]"  
$ git push
```

5. 现在，返回到[Pivotal Tracker](#)页面，我们将发现这个指定的故事被自动的发布出去了，附着对应的GitHub上代码提交的链接。

通过[Trello](#)和[Pivotal Tracker](#)实例，非常清楚的一点是我们能紧密地将我们的任务和代码提交绑定在一起。当作为一个团队进行工作的时候，这将节省大量的时间，并且提高工作记录的精确度。非常好的消息是，如果你已经使用了其他项目管理工具，例如[Asana](#)，[Basecamp](#)，或者其他的，你能够用类似的方式添加hook。如果没有你目前正在使用的项目管理工具的hook，[自力更生自己做一个！](#)

## 工具六：持续集成

对于团队软件开发来说，持续集成(CI)是一个非常重要的部分。CI确保当开发人员提交代码改动的时候，将触发自动的构建(build)，包括测试，用来快速地检测软件集成的错误。这将毫无疑问地减少集成过程中的错误，提高快速开发迭代的效率。在下面的例子里，我们将看到如何与[GitHub](#)一起使用[Travis CI](#)，自动检测错误，并且在所有测试通过后进行代码合并。

# 设置 Travis CI

这里我们使用基于[node.js](http://nodejs.org/)服务器，基于[grunt.js][http://gruntjs.com/]作为构建工具的” Hello-World” 应用，来设置Travis CI项目。下面是项目中的文件：

1. hello.js文件是nodejs项目。我们有目的地漏写了一个分号，为了让这个文件不能通过grunt构建工具的lint(静态代码检测工具)：

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello World in Node!\n') // 这里没有分号，将不会通过linting
}).listen(1337, '127.0.0.1');
console.log('Server running at http://127.0.0.1:1337/');
```

2. package.json定义依赖的包：

```
{
  "name": "hello-team",
  "description": "A demo for github and travis ci for team collaboration",
  "author": "name <email@email.com>",
  "version": "0.0.1",
  "devDependencies": {
    "grunt": "~0.3.17"
  },
  "scripts": {
    "test": "grunt travis --verbose"
  }
}
```

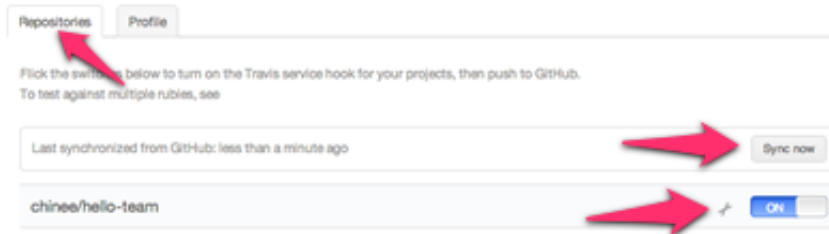
3. 为了简化起见，gruntjs构建工具的配置文件仅仅包含一个任务(linting)：

```
module.exports = function(grunt) {
  grunt.initConfig({
    lint: {
      files: ['hello.js']
    }
  });
  grunt.registerTask('default', 'lint');
  grunt.registerTask('travis', 'lint');
};
```

4. .travis.yml是Travis的配置文件，确保Travis运行我们的测试：

```
language: node_js
node_js:
  - 0.8
```

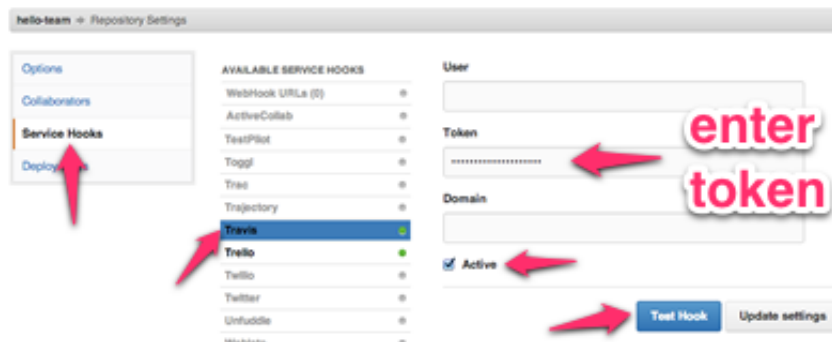
5. 接着，用GitHub帐号登录到Travis，在repository选项卡打开repository hook:



6. 如果上述步骤还不能触发构建，我们将不得不手工配置hook，在Travis的profile栏复制token。



7. 返回到GitHub代码库，使用复制的token设置Travis Hook:



8. 第一次，我们必须手工做一次git push来触发Travis构建，如果一切ok，我们可以访问[http://travis-ci.org/\[用户名\]/\[repo名\]](http://travis-ci.org/[用户名]/[repo名])查看

构建的结果。



## Travis CI 和 Pull 请求(Pull request)

以前没有持续集成的Pull请求流程，步骤大概是(1)提交pull请求(2)合并(3)测试来看是否通过或者失败。带有持续集成hook的Pull请求流程将反转(2)和(3)步骤，Travis CI将向我们汇报每一个Pull请求的持续集成结果，让我们能够知道这个Pull请求是否足够好，并快速作出判断是否合并进主线。下面我们来看这是怎样做到的：

1. 提交一个附带通过构建结果的Pull请求。Travis将做所有的一切，让我们在合并前就能知道这个合并是否足够好。



Discussion Commits 1 Files Changed 1


sayanee opened this pull request 2 minutes ago [Edit](#)

pass the build

No one is assigned [+](#) No milestone [+](#)

No description given.

✓ Good to merge — The Travis build passed ([Details](#))

1 participant 

sayanee added a commit 3 minutes ago

sayanee pass the build [70583f3](#)

You can add more commits to this pull request by pushing to the **master** branch on **sayanee/hello-travis**

Good to merge — The Travis build passed ([Details](#))

ⓘ This pull request can be automatically merged. [Merge pull request](#)

**good to merge!**

**more info on build**

**passed status**

2. 如果这个Pull请求使得构建失败，Travis同样会警告你：

The screenshot shows a GitHub pull request interface. At the top, there are tabs for 'Discussion', 'Commits 1', and 'Files Changed 1'. The main content area shows a pull request titled 'pass the build' by user 'sayanee', opened an hour ago. Below the title, it says 'No one is assigned' and 'No milestone'. A red arrow points to a 'Failed' status message: 'X Failed — The Travis build failed (Details)'. Below this, it says '1 participant'. The commit history shows two commits by 'sayanee': 'pass the build' (commit 70583f3) and 'failing the build with' (commit aa97ca3). A red arrow points to the second commit, which has a tooltip that says 'Failure: The Travis build failed'. At the bottom, there is a 'Merge pull request' button and a 'Merge with caution.' warning. A red arrow points to a 'Failed' status message: 'Failed — The Travis build failed (Details)'.

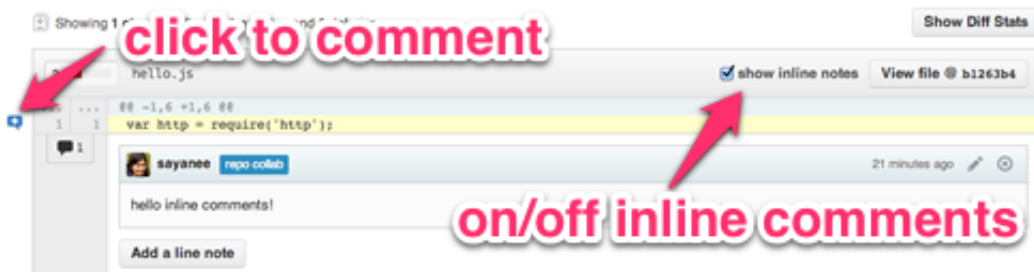
3. 如果我们点击红色的警告链接，浏览器将跳转到Travis页面，显示这次构建的详细信息。

因为自动的构建和及时地通知，[Travis CI](#)对团队来说非常有帮助，它能极大地缩短我们更正错误的周期。如果你使用另外一个非常有名的持续集成工具[Jenkins](#)，你能用相似的步骤设置hook服务。

## 工具七：代码评审

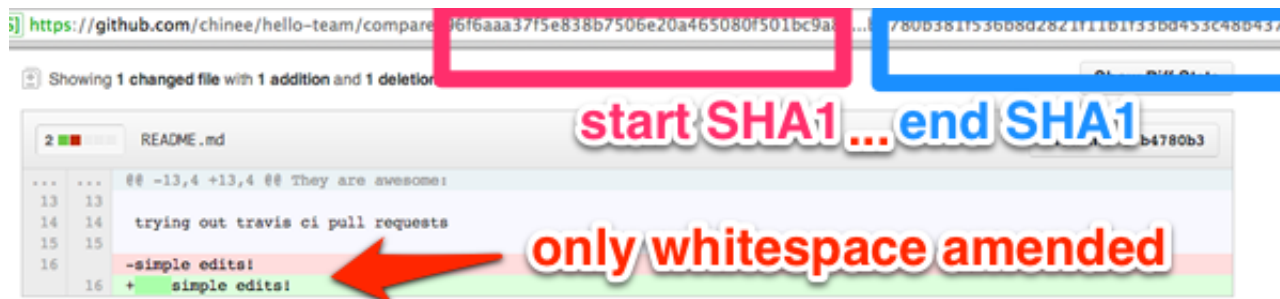
对于每个提交(commit)，GitHub有个干净的接口用来进行评论，甚至是对某行代码进行评论。在进行逐行代码评审的时候，针对单行代

码提出评论和问题的功能就显得非常重要了。打开提交(commit)界面的顶部的检查框，就能显示行内评论。

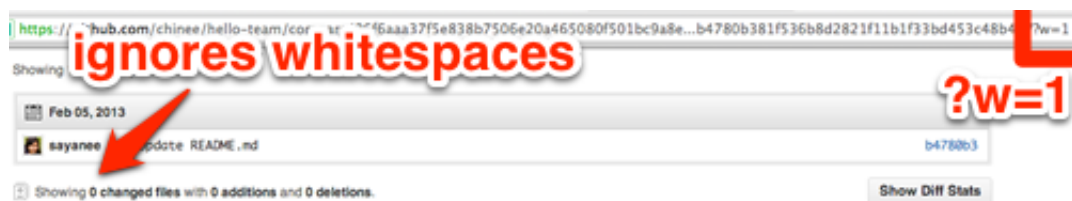


下面探讨一些帮助我们进行代码评审，能快速显示不同提交之间差异的URL模式：

1. 对比branch/tags/SHA1：使用URL模式[https://github.com/\[username\]/\[repo-name\]/compare/\[starting-SHA1\]...\[ending-SHA1\]](https://github.com/[username]/[repo-name]/compare/[starting-SHA1]...[ending-SHA1])。可以用分支或者标签名代替SHA1。



2. 去除空格进行对比：增加?w=1到对比的URL尾部。



3. Diff：增加.diff到URL的后面能得到git diff输出的纯文本信息。在写脚本的时候，这个功能非常有用。
4. Patch：增加.patch到URL的后面能得到[电子邮件补丁提交格式](#)的git diff输出的信息。
5. 行链接：在查看文件时，点击任何行号，GitHub将会在URL后面增加一个#行号，并且将该行的背景颜色置成黄色。这能干净利落地标识代码文件的某一行。我们同样能通过增加#开始行号-结束行号来指定一个范围。这里是[行链接](#)和[行范围链接](#)的例子。

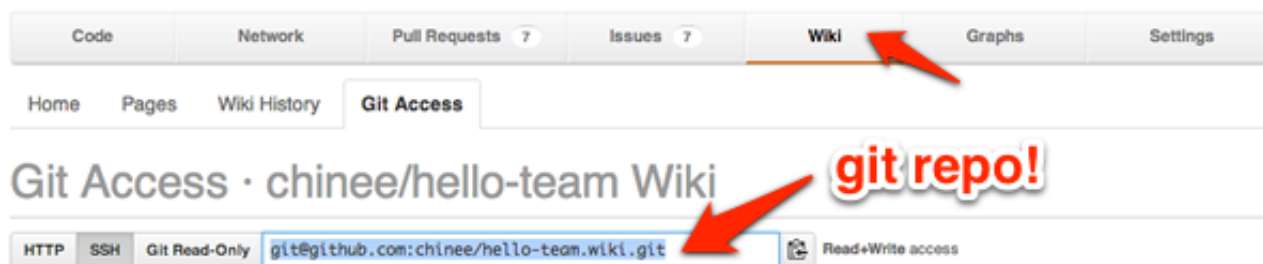
# 工具八：文档

在这段，我们将探讨两种文档方法：

1. 正式文档：使用GitHub Wiki生成正式的项目文档。
2. 非正式文档：使用GitHub [Hubot](#)来归档团队内部的讨论，以及与[Hubot](#)互动而自动获得的非常有趣的信息。
3. 提及，快捷键和表情符号

## GitHub维基(Wiki)

每个GitHub代码库都可以生成一个维基，这样非常方便地将代码和文档存放在同一个存储库中。要创建维基，访问主标题的维基选项卡，并设置创建页面的信息。其实维基也有自己的版本，并可以将数据复制到本地机器进行更新，甚至是离线访问。



有一件事我觉得非常有用的是可以将GitHub的维基整合到源代码中，这样我就不必维护两个独立的Git项目了。要做到这一点，我将Wiki作为[git子模块](#)增加到主分支上。如果您使用的是Travis CI或任何其他CI，必须确保构建工具会忽略wiki的子模块。在Travis的CI文件`.travis.yml`中，添加以下内容：

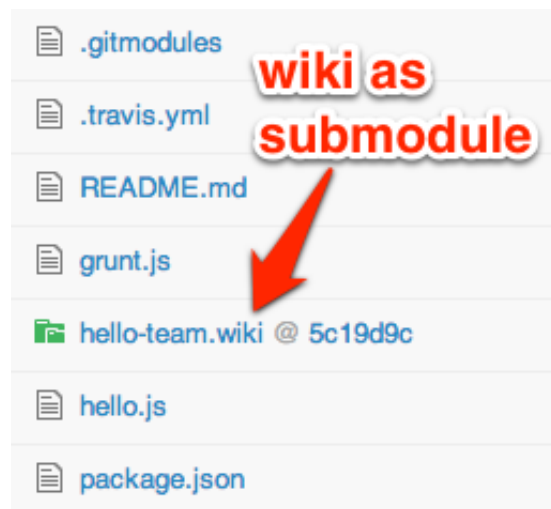
```
git:
  submodules: false
```

接着增加一个git子模块wiki到主代码库中：

```
$ git submodule add git@github.com:[username]/[repo-name].wiki.git
Cloning into 'hello-team.wiki'...
remote: Counting objects: 6, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 6 (delta 0), reused 0 (delta 0)
```

```
Receiving objects: 100% (6/6), done.  
$ git add .  
$ git commit -m "added wiki as submodule"  
$ git push origin master
```

现在，维基就作为一个子模块显示在代码库项目中。



## GitHub Hubot

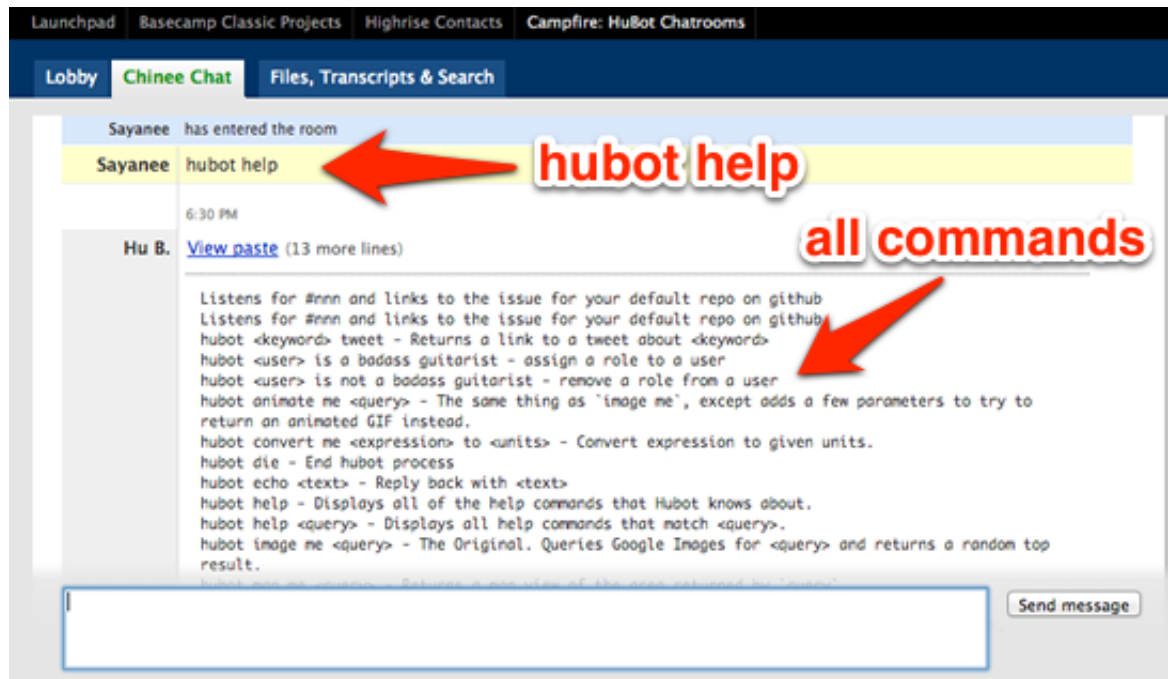
Hubot，总之，可以极大地增添了不少的乐趣记录，并通知小组讨论重要的提交。

[Hubot](#)是一个简单的聊天机器人，可以检索信息，或提供通知，每当GitHub有代码提交，问题，或活动时。在一个旨在减少，甚至完全消除会议的一个团队中，[Hubot](#)拥有所有团队成员的聊天接口，帮助您记录着每一个讨论。这当然促进灵活的工作时序，因为团队没有必要同时出席讨论。警告：[Hubot](#)是非常上瘾的！

有了这个，让我们开始在[Heroku](#)上设置[Hubot](#)，拥有[Campfire](#)聊天接口的聊天机器人！[\[Heroku\]](#)和[Campfire](#)，都有免费的版本供大家开始尝试。

1. 我们将使用[GitHub出品的支持Campfire的Hubot](#)。如果你愿意，也可以使用其他如Skype，IRC，GTalk等[聊天适配器](#)。
2. 建立一个仅为[Hubot](#)的[Campfire](#)账号，这个账号将新建一个房间，并邀请其他人加入。
3. 根据Hubot维基上给的[指示](#)，部署Hubot到Heroku上。如果Heroku的应用程序的URL返回了一个cannot GET /，别惊慌，因为默认情况下[不会得到任何返回](#)。

4. 从Hubot Campfire账号上，邀请你自己的账号，现在，登录你自己的Campfire账号，然后执行`Hubot help`，你将得到所有Hubot支持的命令。

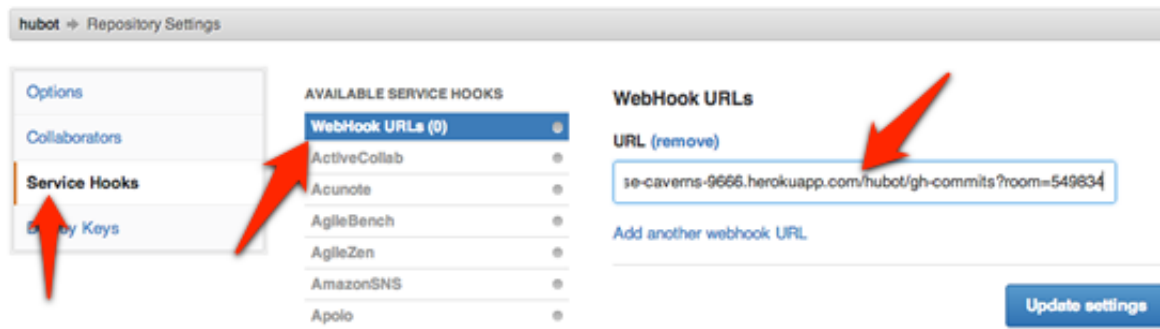


5. 尝试几次，例如`Hubot ship it`或者`Hubot map me CERN`。

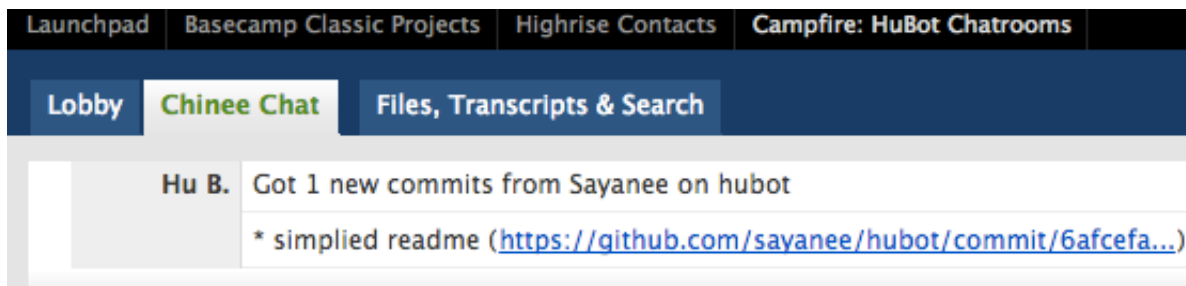


6. 下一步，我们将增加一个Hubot脚本，这里已经有[大量的脚本](#)，附带[命令说明](#)。
7. 作为实例，我们将增加一个github提交脚本，以至于每次有一个新的提交，Hubot将在聊天室通知大家。将文件github-commits.coffee放置到scripts目录。
8. 更新package.json文件，根据每个脚本文件头的指示，加入新的依赖包。
9. 使用下面命令再一次发布代码到Heroku: `git push heroku master`
10. 浏览GitHub代码库，我们希望代码提交通知能显示在聊天室，在代码库设置下增加一个web hook，对github-commits脚本，webhook将是[HUBOT\_URL]:[PORT]/hubot/gh-commits?room=[ROOM\_ID]





11. 下一次当代码库有新的代码提交，Hubot将在聊天室如此说：



检查其他[Github相关的Hubot的脚本](#)，或者如果您想自己写一个脚本，这里有[一个很酷的教程](#)！总之，Hubot可以极大地增添很多乐趣在文档记录、通知小组讨论代码库发生的重要提交，问题和活动。试试看吧！

关于和团队一起使用GitHub，最后要说明的是，这里有一些提高生产力的技巧：

1. 提及(Mentions) - 在任何文本区域中，我们可以通过@用户名提到另外一个GitHub用户，并且该用户将得到通知。
2. 快捷键 - 按SHIFT + ?可以查看Github上任何页面上的快捷键。
3. 表情符号 - 通过使用[表情符号](#)，Github上的文本区域还支持插入的图标。来吧，与队友一起工作时有点情趣！

## GitHub上非软件项目的合作

我们大多数人会认为使用Github只能为软件项目。毕竟，Github产生就是为了社交编程。但是，也有一些很酷的使用Github的库被用于非编码项目，和他们的合作和讨论同样非常棒。因为这些项目是开源的，任何人都可以作出贡献，这是快速修复错误，容易报告错误，与志同道合的人有效的合作。只是为了好玩，这里是其中的一些：



- 房屋修复: [房屋的问题跟踪](#)
- 书籍: [Little MongoDB Book](#), [Backbone Fundamentals](#)
- 歌词: [JSConfEU Lyrics](#)
- 找男朋友: [boyfriend\\_require](#)
- 教导: [Wiki](#)
- 基因组数据: [Ash Dieback epidemic](#)
- 博客: [CSS Wizardry](#)

你能想象[GitHub开发团队](#)怎么认为这些项目？

“我们挖掘像这样一样使用GitHub的乐趣！”

---

## 更多的资源

- [Social Coding in GitHub](#), a research paper by Carnegie Melon University
  - [How Github uses Github to build Github](#) by Zac Holman
  - [Git and Github Secrets](#) by Zac Holman
  - [New features in Github](#) from the Github Blog
  - Github Help: [pull requests](#), [Fork a Repo](#)
  - [Github features for collaboration](#)
  - Nettuts+ Tutorials: [Git](#) and [Github](#)
  - [Lord of the Files: How Github Tamed free Software \(and more\)](#) by Wired
- 

## 更多合作的乐趣！

那些都是在GitHub上积攒的协作化工具。大部分都是作为分析工具，或者用于和团队工作时节省时间的自动化工具。你有更多GitHub团队合作的技巧吗？让我们一起分享！

Categories: [github](#)  
[Tweet](#)

## Comments

0 条评论。



留言...

最好的 ▾ 社区

分享 ▾ ▾

还没有评论



订阅评论



通过邮件订阅

## License

如非特别声明，本 Blog 的文章由 Xiaocong He 创作，采用[知识共享署名-非商业性使用-相同方式共享 3.0 Unported 许可协议](#)进行许可。

## Recent Posts

- [在 Markdown 中嵌入 UML 文档](#)
- [使用 GitHub 进行团队合作](#)
- [介绍 Android DropBoxManager Service](#)
- [Yeoman: 一个新的 javascript 构建工具](#)
- [在 Backbone 项目中使用 backbone.layoutmanager 来组织页面布局](#)

## Reading

[xiaoconghe on douban.com](#)»

Powered by [Octopress](#) and [Compbits](#)