目录

[Github 2](#_Toc436641904)

[Github简介 2](#_Toc436641905)

[Github优势 3](#_Toc436641906)

[Github注册 3](#_Toc436641907)

[安装Git 5](#_Toc436641908)

[安装Git 5](#_Toc436641909)

[验证Github 6](#_Toc436641910)

[创建仓库 8](#_Toc436641911)

[在GitHub创建新仓库 8](#_Toc436641912)

[提交修改 10](#_Toc436641913)

[Fork仓库 13](#_Toc436641914)

[Fork一个仓库 13](#_Toc436641915)

[让你的仓库保持同步 14](#_Toc436641916)

[Fork其他仓库 16](#_Toc436641917)

[社交 17](#_Toc436641918)

[Follow用户 17](#_Toc436641919)

[Watch项目 17](#_Toc436641920)

[还能做些什么 17](#_Toc436641921)

[客户端 19](#_Toc436641922)

[客户端安装 19](#_Toc436641923)

[Fork 20](#_Toc436641924)

[提交到本地 21](#_Toc436641925)

[同步远程仓库 21](#_Toc436641926)

[创建组织 22](#_Toc436641927)

[创建GitHub团队仓库 23](#_Toc436641928)

[创建Team 24](#_Toc436641929)

[签入签出代码 25](#_Toc436641930)

[Git常用命令 26](#_Toc436641931)

[更多： 29](#_Toc436641932)

# Github

### Github简介

Git是一个分布式的版本控制系统，最初由[Linus Torvalds](http://baike.baidu.com/view/518124.htm)编写，用作[Linux内核](http://baike.baidu.com/view/573460.htm)代码的管理。在推出后，Git在其它项目中也取得了很大成功，尤其是在[Ruby](http://baike.baidu.com/subview/45135/5977034.htm)社区中。目前，包括[Rubinius](http://baike.baidu.com/view/1772201.htm)、Merb和Bitcoin在内的很多知名项目都使用了Git。Git同样可以被诸如[Capistrano](http://baike.baidu.com/view/7283319.htm)和Vlad the Deployer这样的部署工具所使用。

作为开源代码库以及版本控制系统，Github拥有140多万开发者用户。随着越来越多的应用程序转移到了云上，Github已经成为了管理软件开发以及发现已有代码的首选方法。

如前所述，作为一个分布式的版本控制系统，在Git中并不存在主库这样的概念，每一份复制出的库都可以独立使用，任何两个库之间的不一致之处都可以进行合并。

GitHub可以托管各种git库，并提供一个web界面，但与其它像 [SourceForge](http://baike.baidu.com/view/1091461.htm)或[Google Code](http://baike.baidu.com/view/2252816.htm)这样的服务不同，GitHub的独特卖点在于从另外一个项目进行分支的简易性。为一个项目贡献代码非常简单：首先点击项目站点的“fork”的按钮，然后将 代码检出并将修改加入到刚才分出的代码库中，最后通过内建的“pull request”机制向项目负责人申请代码合并。已经有人将GitHub称为代码玩家的MySpace。

在GitHub进行分支就像在[Myspace](http://baike.baidu.com/view/137969.htm)（或[Facebook](http://baike.baidu.com/view/409608.htm)）进行交友一样，在社会关系图的节点中不断的连线。

GitHub项目本身自然而然的也在GitHub上进行托管，只不过在一个私[](http://baike.baidu.com/picture/3366456/3366456/0/5366d0160924ab181d75807d34fae6cd7b890b1c.html?fr=lemma&ct=single)有的，公共视图不可见的库中。开源项目可以免费托管，但私有库则并不如此。Chris Wanstrath，GitHub的开发者之一，肯定了通过付费的私有库来在财务上支持免费库的托管这一计划。

是的，我们正是这么计划的。通过与客户的接洽，开发FamSpam，甚至是开发GitHub本身，GitHub的私有库已经被证明了物有所值。任何希望节省时间并希望和团队其它成员一样远离页面频繁转换之苦的人士都会从GitHub中获得他们真正想要的价值。

在GitHub，用户可以十分轻易地找到海量的开源代码。

### Github优势

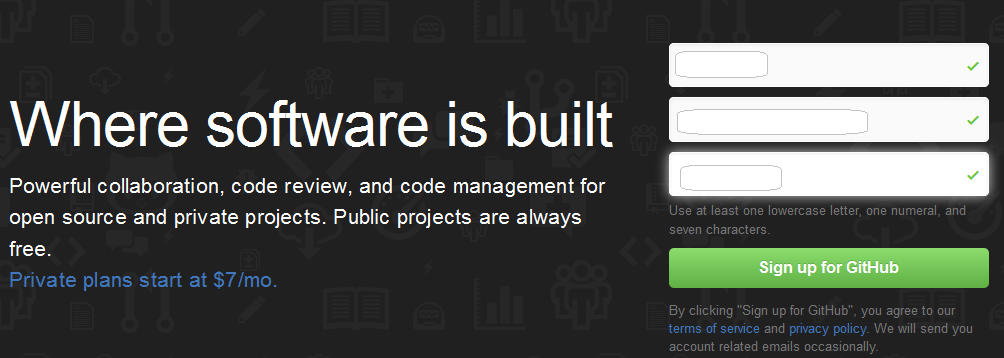
GitHub 之所以如此受欢迎，它的优势是不容忽视的：

1. GitHub 只支持 Git 格式的版本库托管，而不像其他开源项目托管平台还对CVS、SVN、Hg 等格式的版本库进行托管。GitHub 的哲学很简单，既然 Git 是最好的版本控制系统之一（对于很多喜欢 Git 和 GitHub 的人没有之一），没有必要为兼顾其他版本控制系统而牺牲 Git 某些独有特性。因此没有支持其他版本控制系统的历史负担，是 GitHub 成功的要素之一。
2. GitHub 对 Git 版本库提供了完整的协议支持，支持 HTTP 智能协议、Git-daemon、SSH 协议。
3. GitHub 提供在线编辑文件的功能，不熟悉 Git 的用户也可以直接通过浏览器修改版本库里的文件。
4. 将社交网络引入项目托管平台是 GitHub 的创举。用户可以关注项目、关注其他用户进而了解项目和开发者动态。
5. 项目的 Fork 和 Pull Request 构成 GitHub 最独具一格的工作模式。对提交代码的逐行评注及 Pull Request 构成 GitHub 特色的代码审核。
6. GitHub 通过私有版本库托管、面向企业的版本库托管和项目管理平台、人员招聘等付费服务获得了商业上的成功，这种成功使得 GitHub 不必以页面中嵌入广告的方式维持运营，最大的受益者还是用户。
7. GitHub 网站采用 Ruby on Rails 架构，在 Web 设计中运用了大量的 JavaScript、AJAX、HTML5 等技术，支持对使用 Markdown 等标记语言的内容进行渲染和显示等。关注细节使得 GitHub 成为了项目托管领域的后起之秀。

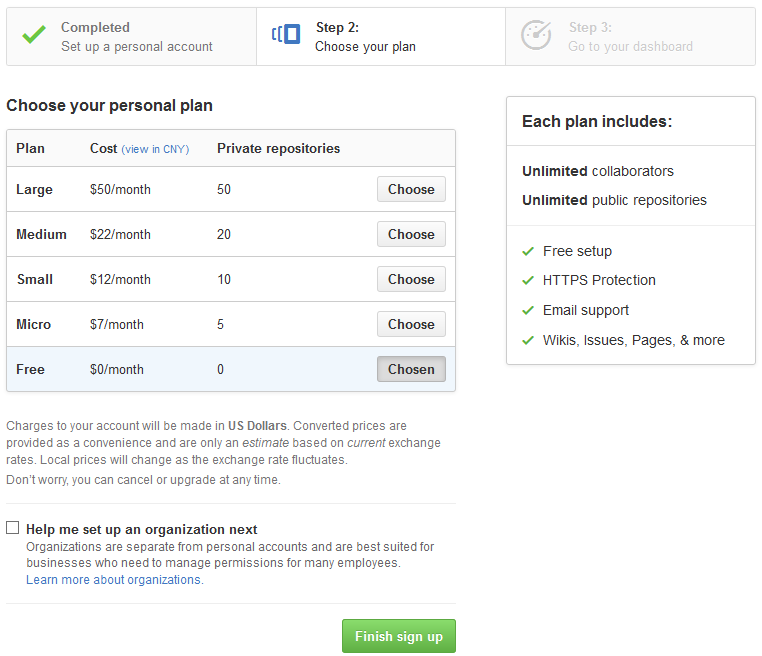
### Github注册

要想使用 GitHub 第一步当然是注册 GitHub 账号：

1. 首先打开 https://github.com 进行注册，输入正确的用户名、邮箱和密码。



1. 点击「Sign up now」出现如下页面，这里我们默认选择“Free”类型。



1. 点击「Finish sign up」按钮，注册邮箱会收到验证邮件，点击链接即可完成注册。

注：不同类型的选择根据我们的需要，如果存放开源项目，则免费托管；存放私有库，则需要付费。费用如下表：

|  |  |  |
| --- | --- | --- |
| 私有库空间 | 费用 | 私有库数量 |
| 大 | $50/月 | 50 |
| 中 | $20/月 | 20 |
| 小 | $10/月 | 10 |
| 微小 | $5/月 | 5 |
| 免费 | $0/月 | 0 |

# 安装Git

### 安装Git

**在Mac OS X上安装Git**

如果你正在使用Mac做开发，直接从AppStore安装Xcode，Xcode集成了Git，不过默认没有安装，你需要运行Xcode，选择菜单“Xcode”->“Preferences”，在弹出窗口中找到“Downloads”，选择“Command Line Tools”，点“Install”就可以完成安装了。



**在Windows上安装Git**

Windows下要使用很多Linux/Unix的工具时，需要Cygwin这样的模拟环境，Git也一样。Cygwin的安装和配置都比较复杂，不过，有人已经把模拟环境和Git都打包好了，名叫msysgit，只需要下载一个单独的exe安装程序，其他什么也不用装。

msysgit是Windows版的Git，从<http://msysgit.github.io/>下载，然后按默认选项安装即可。

安装完成后，在开始菜单里找到“Git”->“Git Bash”，出现如下图的命令行窗口，就说明Git安装成功！



安装完成后，还需要最后一步设置，在命令行输入：

$ git config --global user.name “Your Name”

$ git config --global user.email [email@example.com](mailto:email@example.com)

注意git config命令的--global参数，用了这个参数，表示你这台机器上所有的Git仓库都会使用这个配置。

### 验证Github

当你通过 Git 连接到一个 GitHub 仓库后，我们需要通过SSH方式验证 GitHub，我们需要在电脑中[生成 SSH keys](https://help.github.com/articles/generating-ssh-keys)，用来从 GitHub 中 push 或 pull。

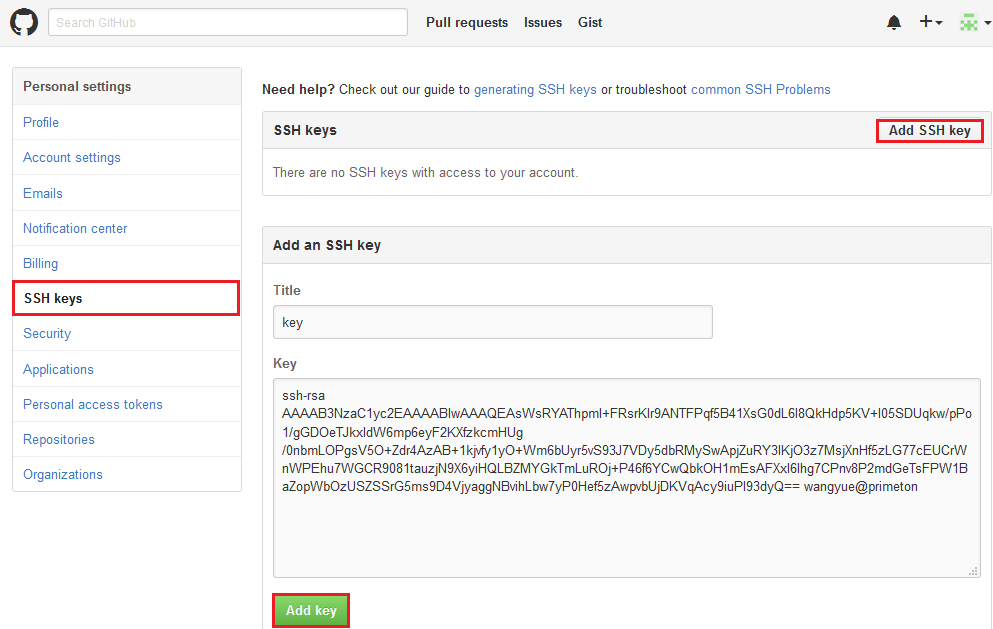
第1步：创建SSH Key。在用户主目录下，看看有没有.ssh目录，如果有，再看看这个目录下有没有id\_rsa和id\_rsa.pub这两个文件，如果已经有了，可直接跳到下一步。如果没有，打开Git Bash，创建SSH Key：

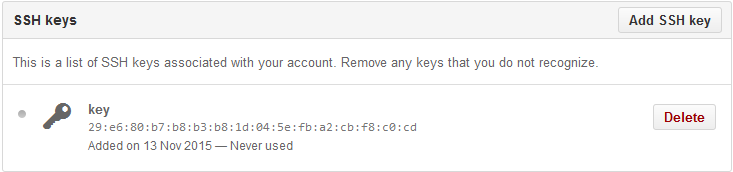
$ ssh-keygen -t rsa -C “youremail@example.com”

你需要把邮件地址换成你自己的邮件地址，然后回车，使用默认值即可，这个Key无需设置密码。

完成后在用户主目录里找到.ssh目录，里面有id\_rsa和id\_rsa.pub两个文件，这两个就是SSH Key的秘钥对，id\_rsa是私钥，不能泄露出去，id\_rsa.pub是公钥，可以放心地告诉任何人。

第2步：登陆GitHub，打开页面右上角如图标志选择“Settings”。

第3步：点击“Add SSH Key”，填上任意Title，在Key文本框里粘贴id\_rsa.pub文件的内容后点击“Add key”：

添加成功后如图显示：

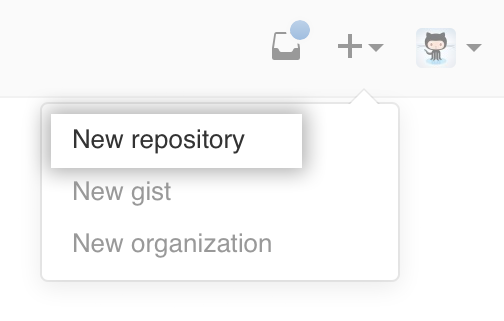
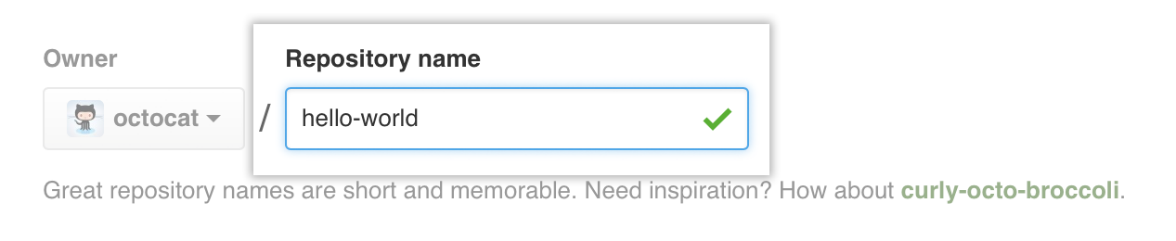
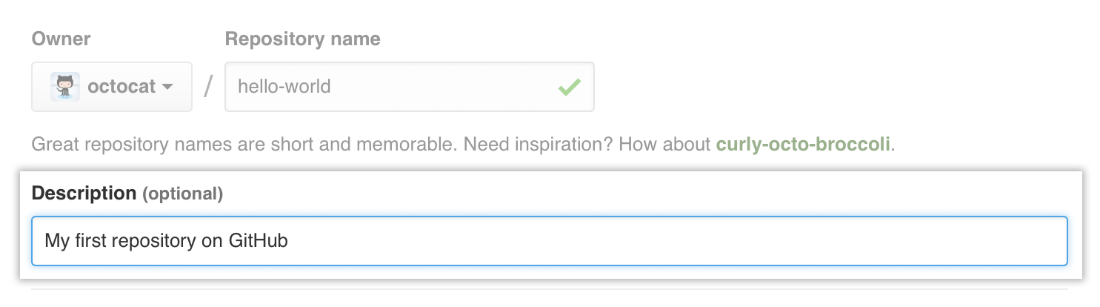
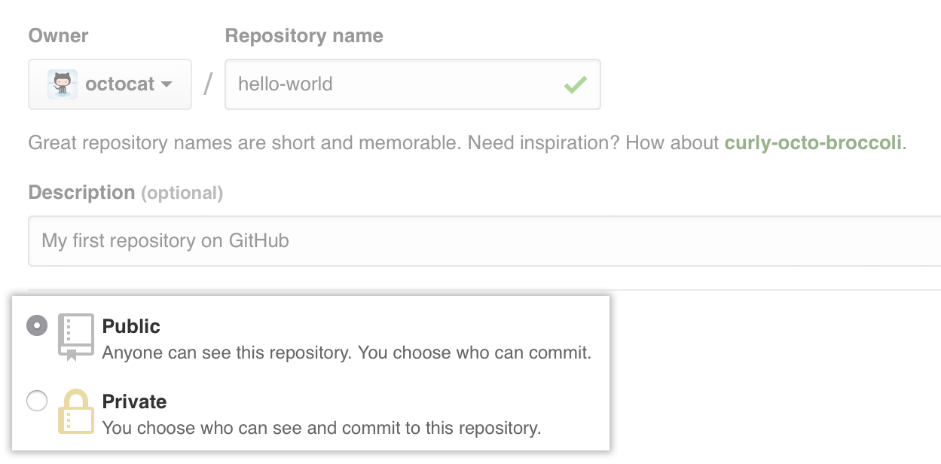
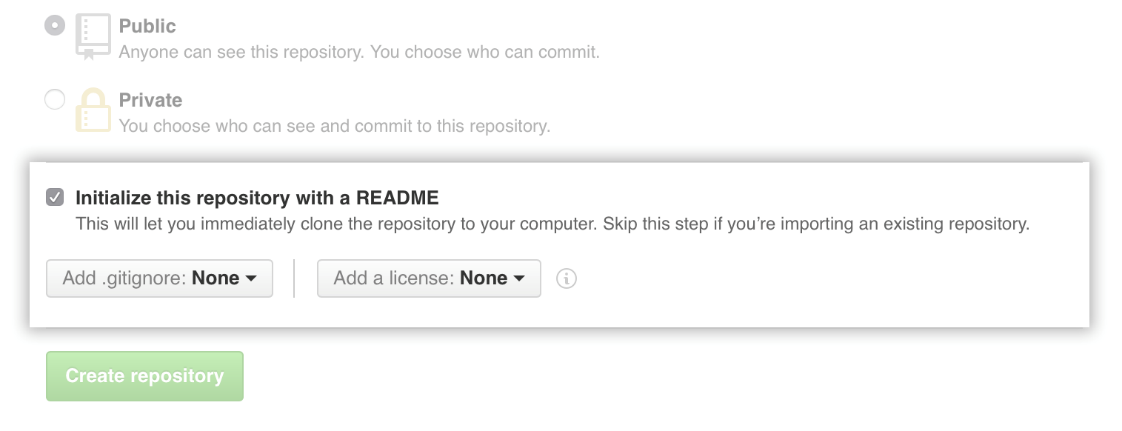
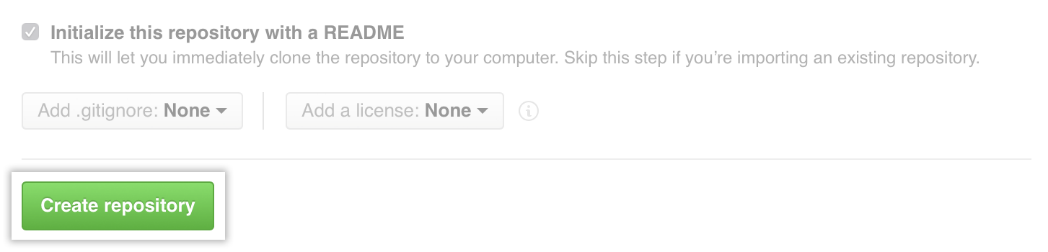
# 创建仓库

想把你的项目上传到GitHub，你需要一个仓库。让我们动手创建一个！

在GitHub， 你可以把各种各样的项目存储在仓库中。

每个用户都能拥有自己的仓库，如果你已经完成注册，就可以着手创建自己的仓库了！

### 在GitHub创建新仓库

1. 在任意页面的右上角，点击+，然后点击**New repository**新建仓库。 
2. 为你的仓库起一个简单易记的名称，例如“hello-world”。 
3. 接下来， 对你的仓库进行描述（非必须）。例如，“My first repository on GitHub”。
4. 选择开源或是私有。
   * **Public：**公有仓库是个不错的选择！这个仓库对GitHub所有用户可见，在这个互相协作的社区你会大大受益。
   * **Private：**私有仓库需要更多操作。这类仓库只对创建者和被赋予权限的用户可见。私有仓库仅对付费用户可用。 更多信息请参考“[What plan should I choose?](https://help.github.com/articles/what-plan-should-i-choose)。”
5. 选择**Initialize this repository with a README**。
6. 点击**Create repository**创建仓库。 

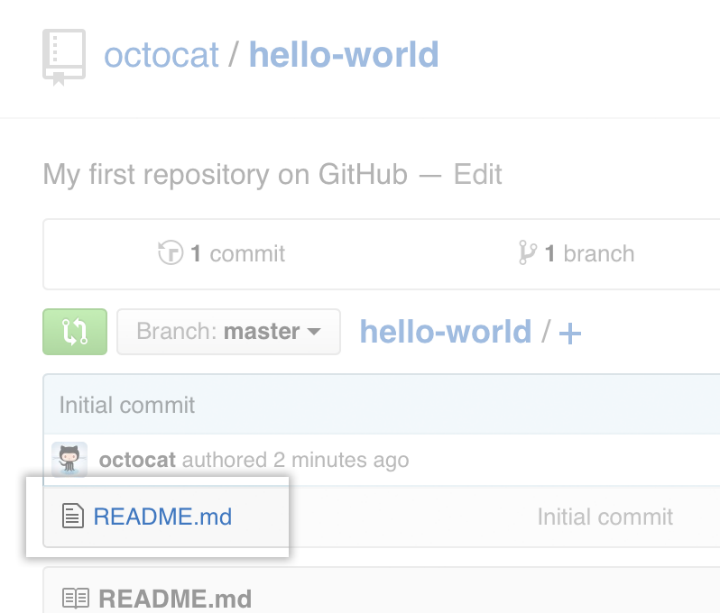
恭喜你！成功地创建了第一个仓库，并且用*README*文件对它进行了初始化了。

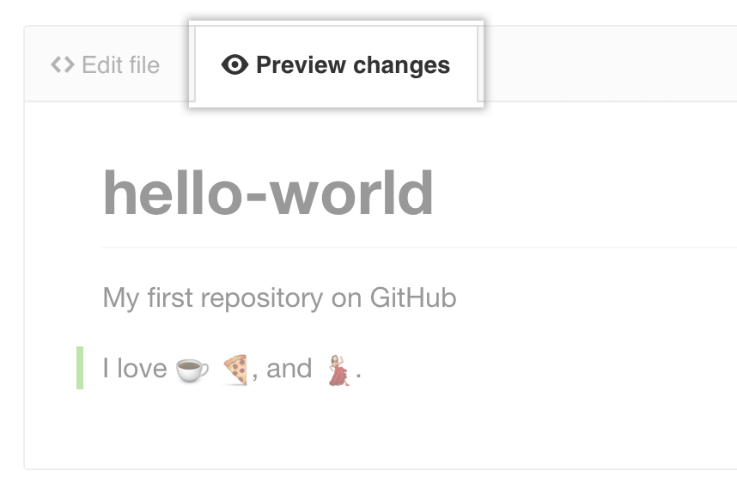
### 提交修改

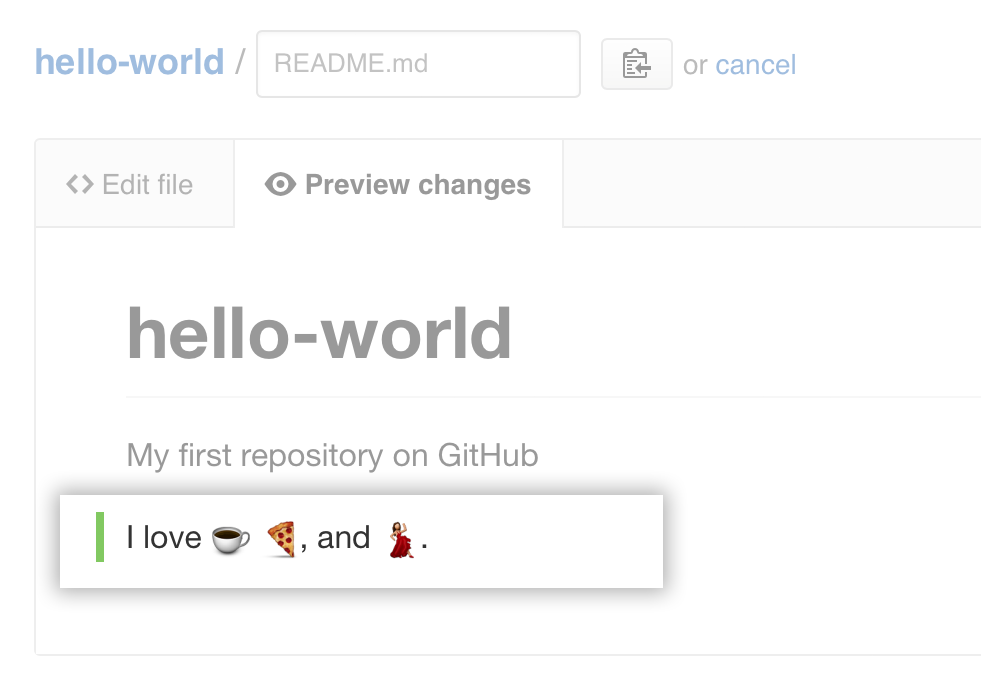
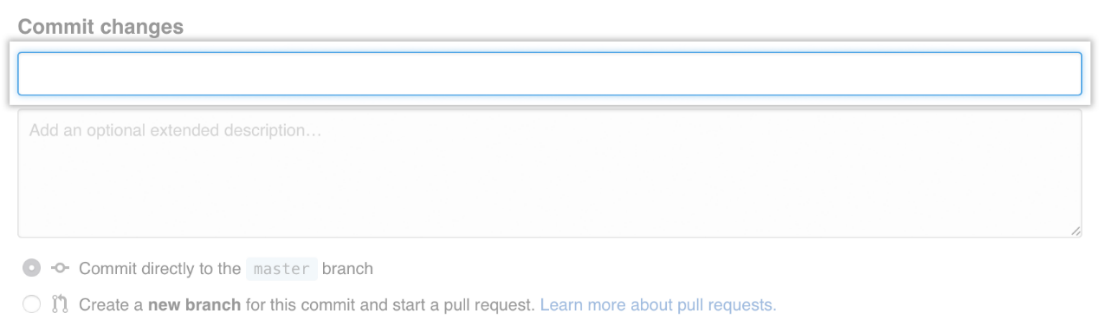
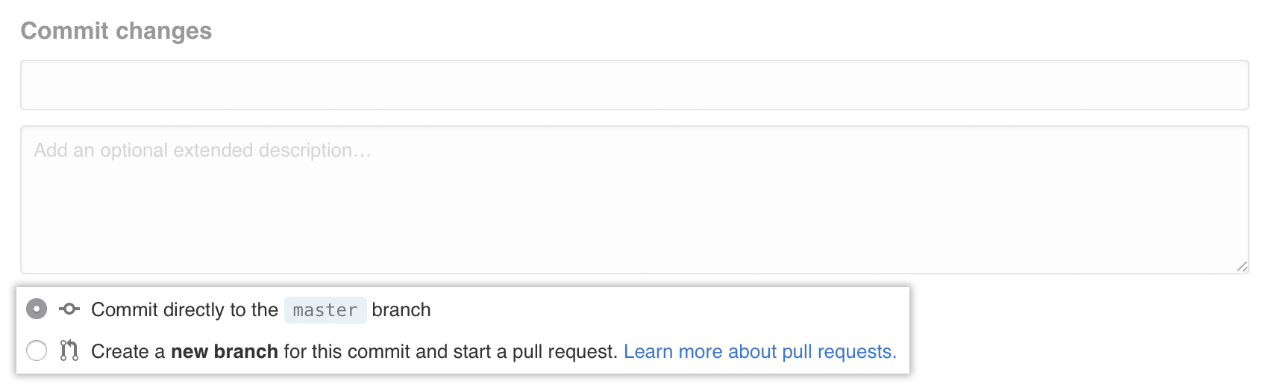
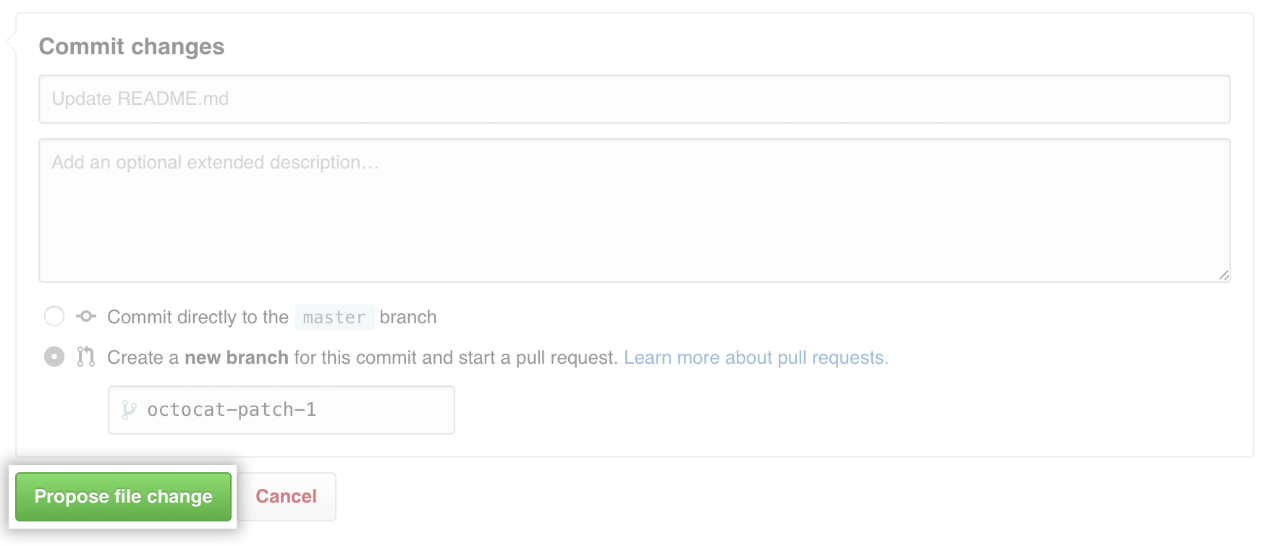
提交就相当于在特点的时间点对项目中的所有文件进行一次快照。

当仓库创建完成并通过*README*文件进行初始化后，*README*文件里有对你这个项目的详细描述，也可以添加一些如何安装或使用你的项目的文档。*README* 文件的内容会自动展示在仓库首页。

让我们提交一个对*README*文件的修改。

1. 在仓库文件列表里，点击***README.md***。
2. 在页面顶端，点击编辑按钮。
3. 在编辑页面，输入一些个人信息。
4. 点击**Preview changes**进行预览。



1. 查看一下你对文件进行的更改。你会发现新添内容被标记为绿色。
2. 在页面底端，输入简短有力的内容来概括你对文件进行的修改。 
3. 在提交更改页面底端有两个选项，你可以选择把更改提交到现有分支或新建分支。如果目前的分支是master，你可以为提交的修改新建一个分支然后发起变更请求。 
4. 点击**Propose file change**提交变更。

# Fork仓库

Fork就是对一个仓库进行克隆。 克隆后，你可以在不影响原始项目的前提下进行操作。

最常见的，克隆仓库可以用来对别人的项目进行修改或使用别人的项目实施自己的想法。

##### 对某个项目提出更改

使用fork的一个很好例子是进行Bug修复。比起简单地记录一下你发现的bug,你可以：

* 克隆仓库。
* 进行修复。
* 向项目拥有者提交pull request（更改申请）。

如果项目拥有者对你的修改感兴趣的话，他们就会把它更新到原始项目中！

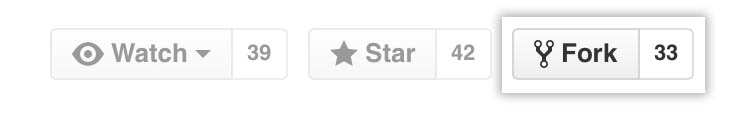
##### 用别人的项目实施自己的想法

开源的核心就在于通过分享代码，我们能够做出更好、更可靠的软件。

实际上, 当你在GitHub创建仓库时, 你可以自动选择制定一个开源协议, 这个协议决定了你是否要把项目分享给其他用户。

### Fork一个仓库

对一个仓库进行克隆仅需要两步操作。下面开始实践把！

1. 在GitHub, 搜索[octocat/Spoon-Knife](https://github.com/octocat/Spoon-Knife)库.
2. 在页面右上角, 点击**Fork按钮**. 

就这么简单!现在你成功地fork了octocat/Spoon-Knife仓库。

### 让你的仓库保持同步

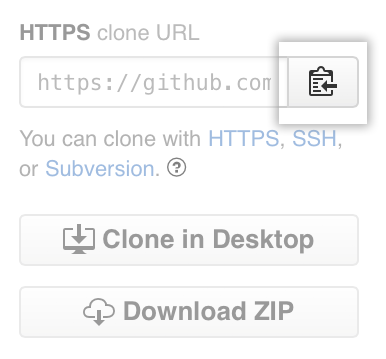
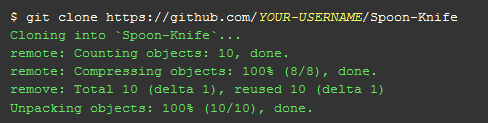
为了向upstream（上游）或者原始库提交修改，你需要时刻与原始库保持同步。这时就要用到Git命令来进行操作。你可以用克隆的[octocat/Spoon-Knife](https://github.com/octocat/Spoon-Knife)库来对upstream库进行操作！

##### 步骤 1: 安装Git

如果你还没有安装Git，请参考安装Git章节。别忘了对Github进行验证。

##### 步骤 2: 克隆远程库到本地

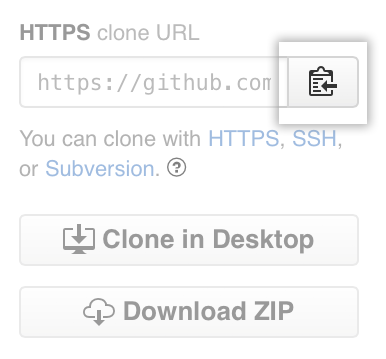
现在你以及成功fork了Spoon-Knife库,但本地电脑上还没有。让我们把fork库克隆到本地计算机吧！

1. 在GitHub, 找到你fork的Spoon-Knife库。
2. 在页面右侧边栏, 点击如图按钮复制fork库的链接。 
3. 打开Terminal(Mac或Linux用户)或命令提示符(Windows用户)。
4. 输入git clone,然后粘贴在第二步里复制的链接。如下图所示：
5. 回车。本地库成功创建。

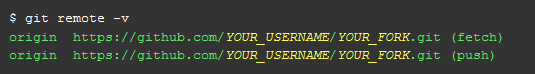
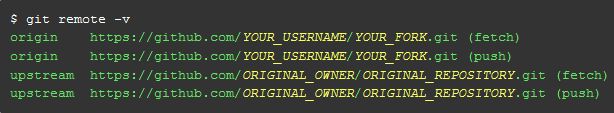
现在你拥有了针对你fork的Spoon-Knife库的本地副本库。

##### 步骤 3: 配置Git使fork库与原始Spoon-Knife库保持同步

当你想对原始库进行修改而创建fork库时，你可以通过配置Git把原始库或upstream的变动更新到本地。

1. 在GitHub, 导航到 [octocat/Spoon-Knife](https://github.com/octocat/Spoon-Knife)库。
2. 在页面右侧边栏，点击如图按钮复制这个仓库的链接：
3. 打开Terminal(Mac或Linux用户)或命令提示符 (Windows)。
4. 进入本地库目录。

* 输入cd回到根目录。
* 输入 ls查看当前目录所有文件。
* 输入cd your\_listed\_directory进入指定目录。
* 输入cd ...返回上一层目录。

1. 输入git remote –v后回车。你会看到远程仓库的详细信息。 
2. 输入git remote add upstream，粘贴你在第二步里复制的网址后回车。
3. 再次输入git remote –v，来验证你指定的上游仓库。 远程克隆仓库的URL标记为原始地址,而原始库的URL标记为upstream（上游，）。

现在，使用一些Git命令你就可以确保fork库和原始库同步更新了。更多信息请参见"[Syncing a fork](https://help.github.com/articles/syncing-a-fork)"。

##### 接下来

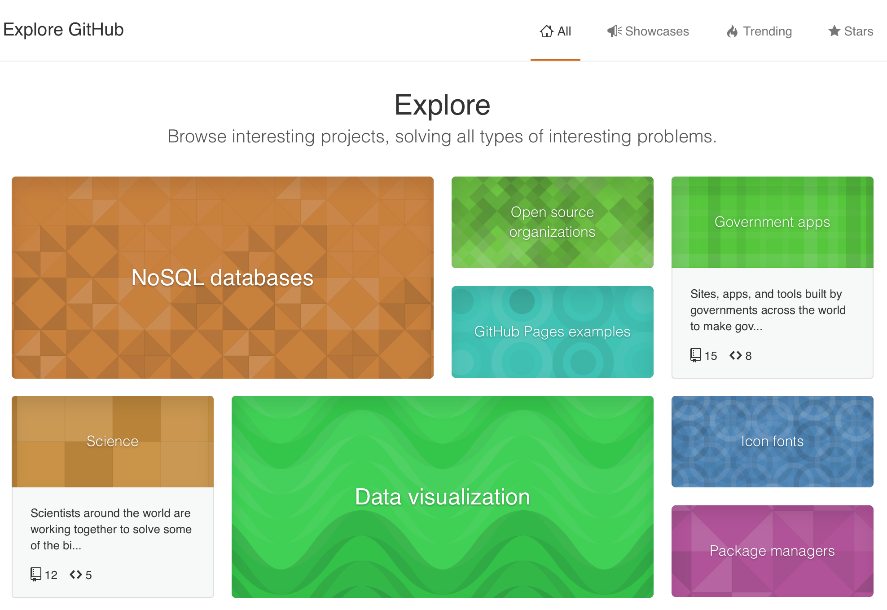
对fork库你能进行很多操作：

* **新建分支：**分支允许你在不对主干项目造成影响的前提下测试新的功能和想法。
* **提交变更请求：**如果你想对原始仓库做点贡献的话，你可以提出变更请求，让项目拥有者把你的克隆库和原始库进行合并更新。

### Fork其他仓库

所有开源库都能被克隆，去寻找一个你感兴趣的项目进行克隆吧！

在[Explore GitHub](https://github.com/explore)你能发现很多感兴趣的项目。经常访问这个页面才能保持与时俱进。

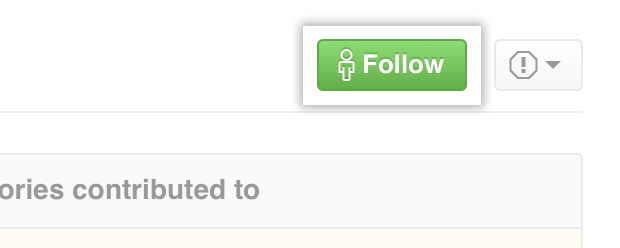


# 社交

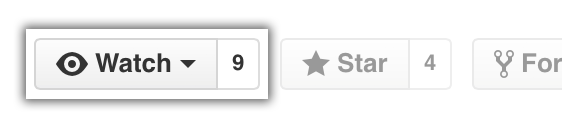
GitHub一个很强大的特性就是可以看到别人在忙些什么，他们在和哪些人接触。

### Follow用户

当你关注了某人，他们的最新动态会显示在你的控制面板。

你可以在他们的页面上点击"follow"按钮进行关注。 

### Watch项目

有时，你需要时时跟踪一个项目的动态。进入项目页面点击**Watch**按钮即可。 

如果作者更新了这个项目，你将会在你的控制面板收到改动的通知。

### 还能做些什么

你已经使用了GitHub提供的一些基础交互功能，但别仅限于此，来看看GitHub提供的其它交互功能吧：

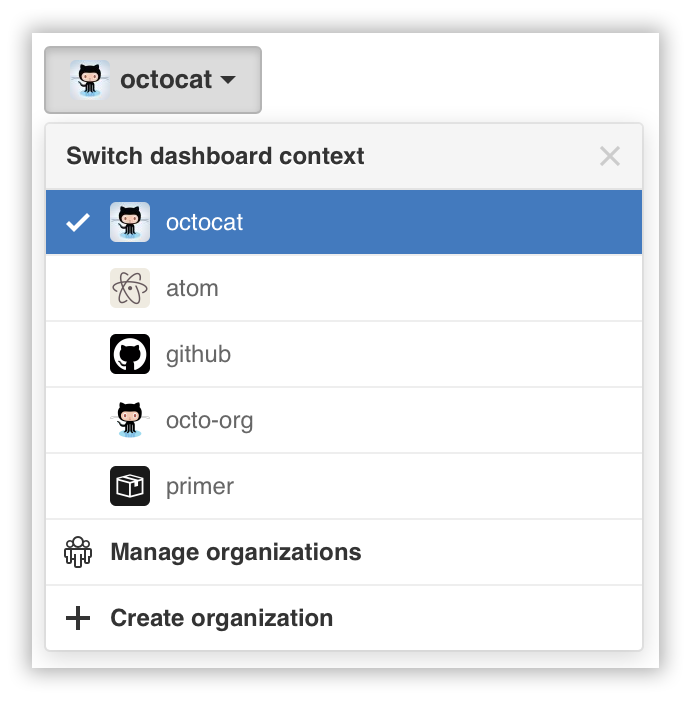
**变更申请**

不管是新添功能还是修复bug，你迫不及待地想为别人的项目做些贡献。 做出修改后，你可以通过向原作者提交变更请求来实现这个目的。

**Issues**Issues Button

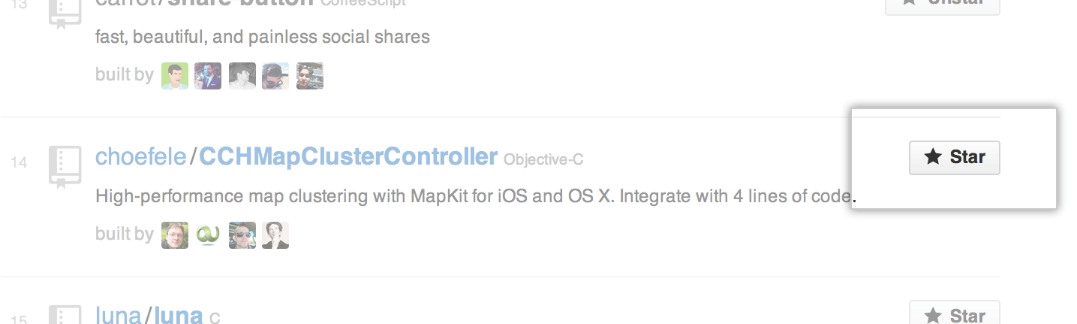
当你与别人合作一个项目的时候可能会遇到一些问题，每个仓库都有Issues功能来帮你追踪这些问题的进程。

**组织**



你是不是希望和一些志同道合的朋友合作开发一个项目？组织功能可以满足你的需求！ 在组织界面你可以设定特殊权限，了解团队概况并且追踪团队成员动态。

**探索**

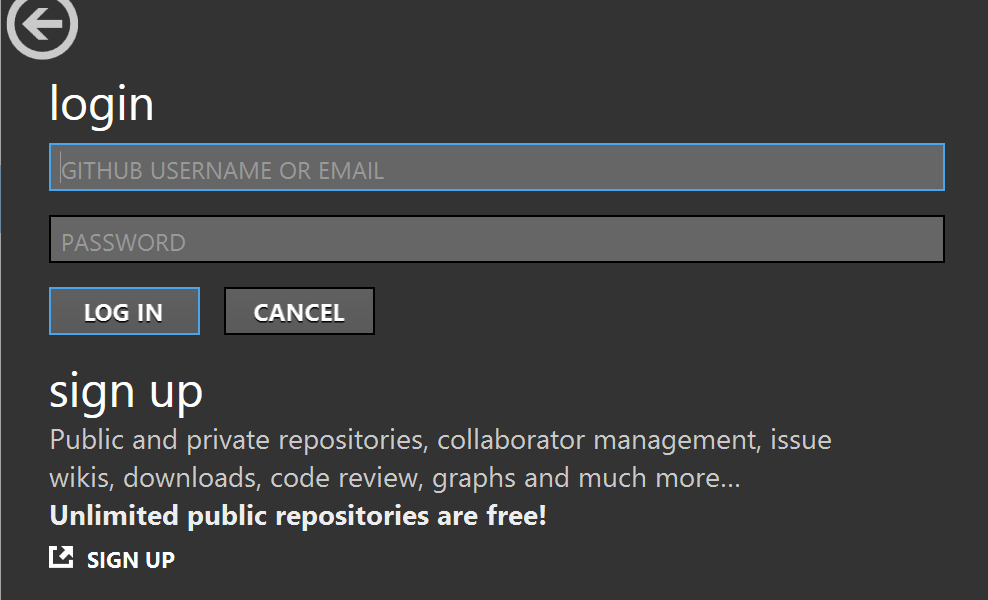


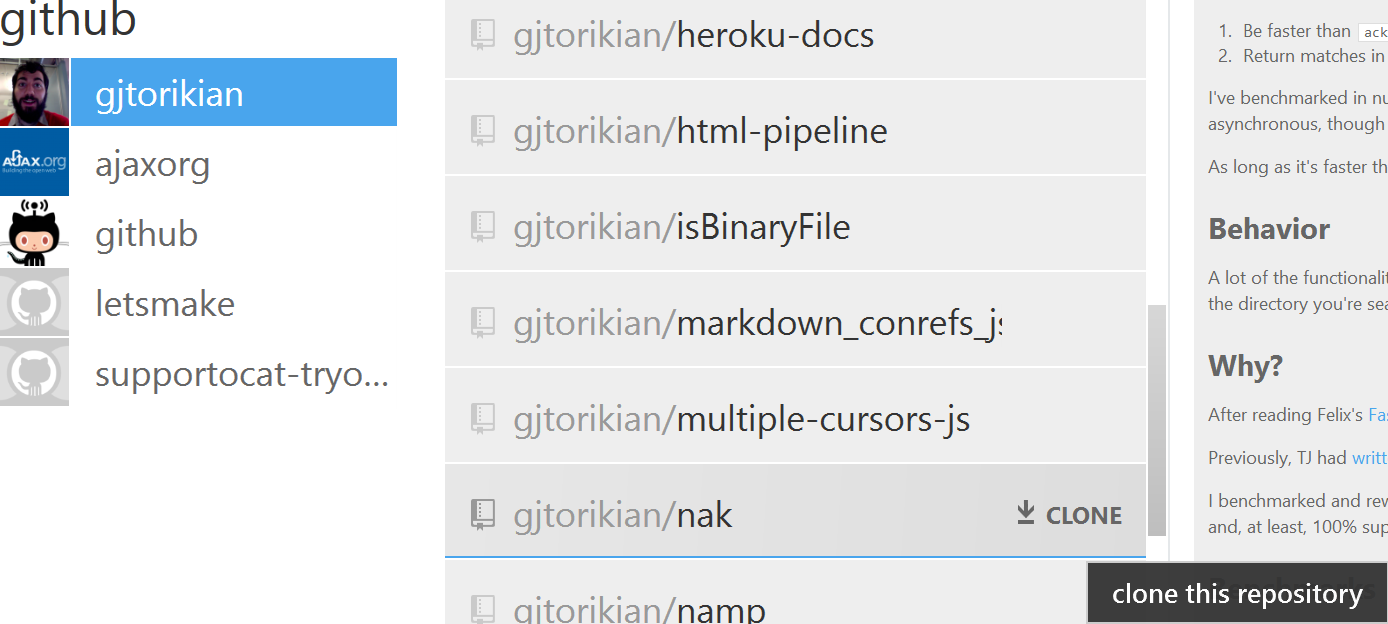
在[Explore GitHub](https://github.com/explore)和[Trending page](https://github.com/trending)页面你会发现很多有意思的项目，你可以收藏这些项目以便时时回顾-访问[stars page](https://github.com/stars)可以看到所有收藏项目。

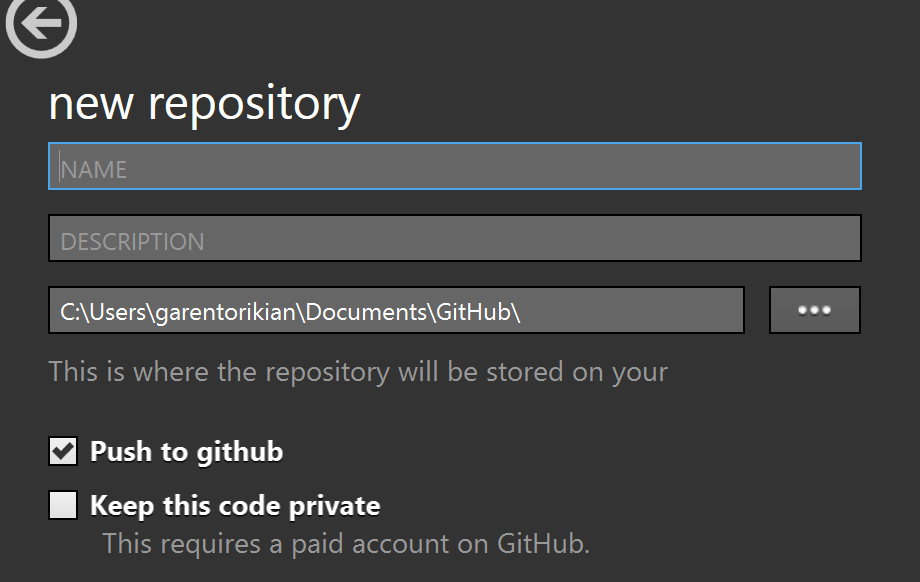
# 客户端

### 客户端安装

1、从 [windows.github.com](http://windows.github.com/) 下载最新版本的 GitHub。

2、当你开启软件时，你可以选择用你的 GitHub 账户登录，或者新建一个账户。

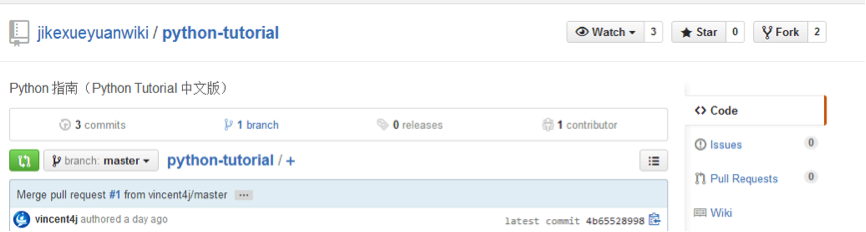
3、在左侧，你可以看到你的 GitHub 账户，同时你也能看到你所在组的其他（用户）。 点击一个用户名，你将看到哪些仓库是可用的，点击 **clone** 将把对应仓库克隆到你的电脑。

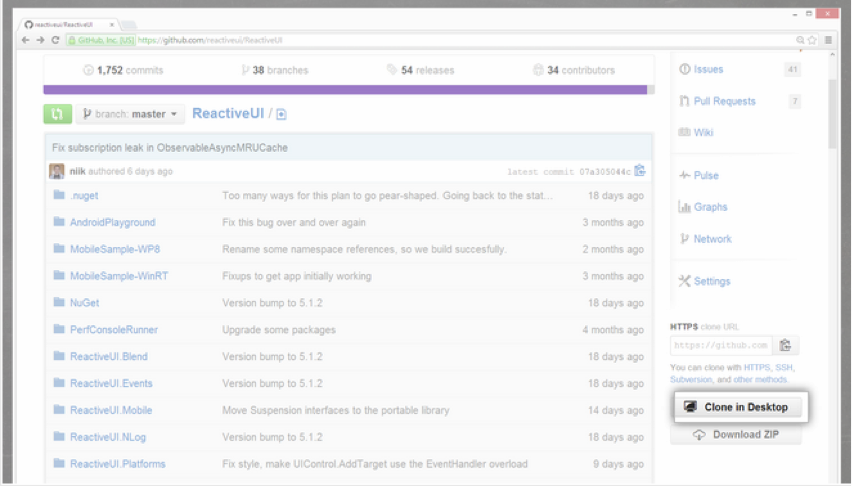
4、另外，你可以点击顶部的 **+ add** 来添加一个新的本地仓库。

你准备好通过 GitHub for Windows 开始编程和同步修改。

### Fork

直接从 GitHub 上克隆来添加库，也是一种不错的选择。

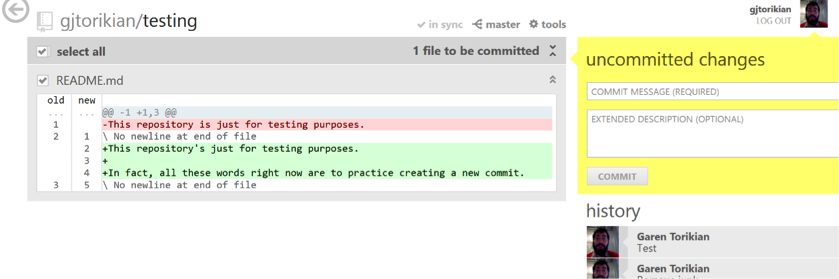
1、你可以直接用你的个人账户或者你所属的组织，通过 GitHub 来浏览相关项目。

2、你也可以直接在 GitHub 上通过 **Clone in Desktop** 按钮来进行一键克隆。

### 提交到本地

你在本地更新了数据，需要先提交到本地仓库：

1、点击你需要同步的库的名称。

2、你将看到一个表单，列举了你最新的变动。增添一个提交日志（另外可以选择增加一个描述），然后提交。

### 同步远程仓库

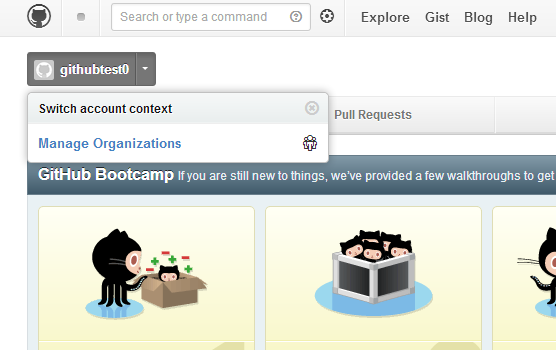
当有新的本地提交记录时，上图中的 **in sync** 按钮会被点亮，并且文字变成 **sync**，只需要点击 **sync** 按钮，就同步到了 GitHub 上的远程仓库。

# 创建组织

GitHub提供免费的团队环境，不过免费仓库容量是300MB，请大家注意。

组织的概念可以理解成公司，机构等。

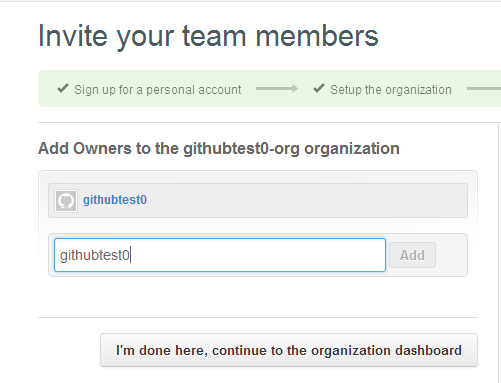
1. 登陆GitHub

2. 点击左上角，用户名旁的下拉按钮，点击【Manage Organizations】

3. 页面跳转至【组织】页面后，单击【Create new organization】按钮

4. 页面跳转至【Create an organization】页面后，填写组织名和邮箱。免费用户请无视下方的付费计划。滚动页面至页脚处，点击绿色【Create organization】按钮。

5. 跳转至【Invite your team member】页面后，为组织指定Owner(最高权限)。在输入框中填写GitHub已有的用户的用户名(系统会帮助自动提示用户名)，点击右侧的【Add】按钮。比如：填写自己的用户名。

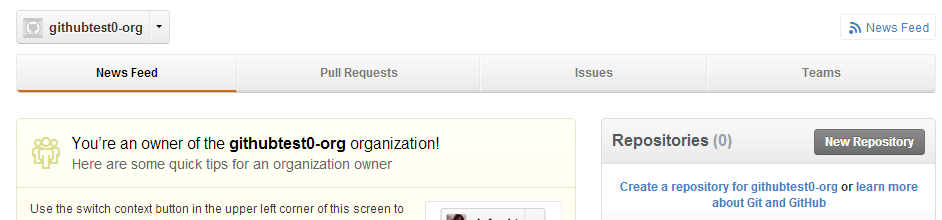


6. 完成后点击上图中的【I‘m done here, continue to the organization dashboard】

至此，组织创建完毕。一个公司下可能有多个团队，也会有多个项目的代码、文档、Bug需要维护。Github允许用户在组织下创建多个Team, 多个Repository。并提供Team和Repository的关联。

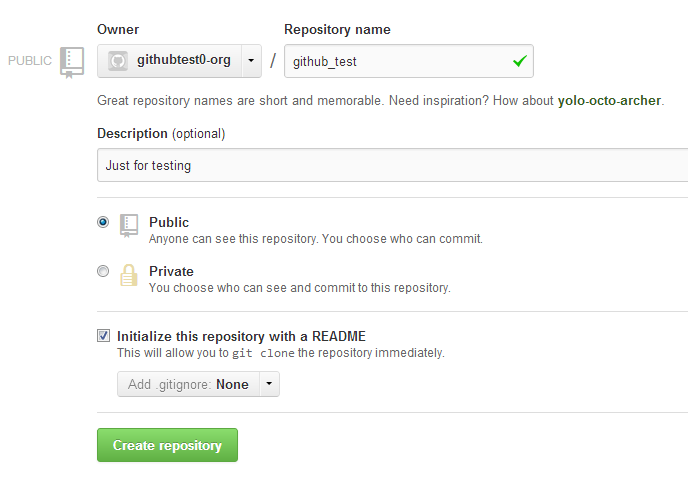
### 创建GitHub团队仓库

1. 创建完组织后，自动跳转到组织主页，也可以直接使用https://github.com/organizations/组织名 来访问。



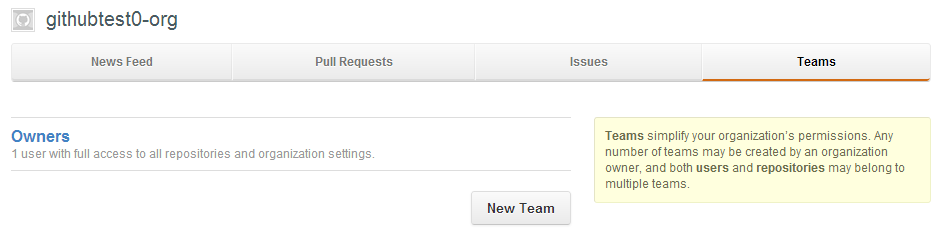
2. 点击上图中，右侧的深灰色【NewRepository】按钮。

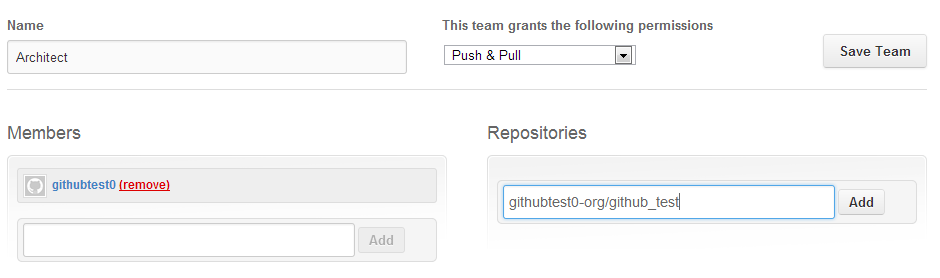
3. 如下图填写仓库信息后，点击绿色【Create repository】按钮。

  
 4. 成功创建仓库后，请回到组织主页。准备开始创建Team

### 创建Team

1. 点击组织主页上的名为Team的TAB，进入Team配置页，如下：

  
 2. 点击上图中的【New Team】按钮，开始创建团队。



填写团队名字，如：Architest。

选择团队的权限，GitHub提供3档权限。自上而下分别是：签出；签入签出；管理员

添加团队成员，请在Members的输入框内填入已有GitHub用户的用户名，并点击右侧的【Add】按钮。

添加团队的仓库，请在Repositories的输入框内填写仓库名，格式为：组织名/仓库名。点击右侧【Add】按钮添加。在此请注意：可选的仓库只是当前组织下的仓库，你的个人仓库不属于可选的仓库。

完成后，点击【Save Team】完成配置。至此GitHub的团队开发环境已经构建完毕。

### 签入签出代码

GitHub提供了多种方式。比较推荐的是GitHub客户端、GitHub的eclipse插件、TortoiseSVN(小乌龟)。

最佳选择是后两种

# [Git常用命令](http://www.cnblogs.com/vman/articles/Git_cmds.html)

**查看、添加、提交、删除、找回，重置修改文件**

git help <command> # 显示command的help

git show # 显示某次提交的内容 git show $id

git co -- <file> # 抛弃工作区修改

git co . # 抛弃工作区修改

git add <file> # 将工作文件修改提交到本地暂存区

git add . # 将所有修改过的工作文件提交暂存区

git rm <file> # 从版本库中删除文件

git rm <file> --cached # 从版本库中删除文件，但不删除文件

git reset <file> # 从暂存区恢复到工作文件

git reset -- . # 从暂存区恢复到工作文件

git reset --hard # 恢复最近一次提交过的状态，即放弃上次提交后的所有本次修改

git ci <file> git ci . git ci -a # 将git add, git rm和git ci等操作都合并在一起做　　　　　　　　　　　　　　　　　　　　　　　　　　　　git ci -am "some comments"

git ci --amend # 修改最后一次提交记录

git revert <$id> # 恢复某次提交的状态，恢复动作本身也创建次提交对象

git revert HEAD # 恢复最后一次提交的状态

**查看文件diff**

git diff <file> # 比较当前文件和暂存区文件差异 git diff

git diff <*id*1><id2> # 比较两次提交之间的差异

git diff <branch1>..<branch2> # 在两个分支之间比较

git diff --staged # 比较暂存区和版本库差异

git diff --cached # 比较暂存区和版本库差异

git diff --stat # 仅仅比较统计信息

**查看提交记录**

git log git log <file> # 查看该文件每次提交记录

git log -p <file> # 查看每次详细修改内容的diff

git log -p -2 # 查看最近两次详细修改内容的diff

git log --stat #查看提交统计信息

**tig**

Mac上可以使用tig代替diff和log，brew install tig

**Git 本地分支管理**

**查看、切换、创建和删除分支**

git br -r # 查看远程分支

git br <new\_branch> # 创建新的分支

git br -v # 查看各个分支最后提交信息

git br --merged # 查看已经被合并到当前分支的分支

git br --no-merged # 查看尚未被合并到当前分支的分支

git co <branch> # 切换到某个分支

git co -b <new\_branch> # 创建新的分支，并且切换过去

git co -b <new\_branch> <branch> # 基于branch创建新的new\_branch

git co $id # 把某次历史提交记录checkout出来，但无分支信息，切换到其他分支会自动删除

git co $id -b <new\_branch> # 把某次历史提交记录checkout出来，创建成一个分支

git br -d <branch> # 删除某个分支

git br -D <branch> # 强制删除某个分支 (未被合并的分支被删除的时候需要强制)

**分支合并和rebase**

git merge <branch> # 将branch分支合并到当前分支

git merge origin/master --no-ff # 不要Fast-Foward合并，这样可以生成merge提交

git rebase master <branch> # 将master rebase到branch，相当于： git co <branch> && git rebase master && git co master && git merge <branch>

**Git补丁管理(方便在多台机器上开发同步时用)**

git diff > ../sync.patch # 生成补丁

git apply ../sync.patch # 打补丁

git apply --check ../sync.patch #测试补丁能否成功

**Git暂存管理**

git stash # 暂存

git stash list # 列所有stash

git stash apply # 恢复暂存的内容

git stash drop # 删除暂存区

**Git远程分支管理**

git pull # 抓取远程仓库所有分支更新并合并到本地

git pull --no-ff # 抓取远程仓库所有分支更新并合并到本地，不要快进合并

git fetch origin # 抓取远程仓库更新

git merge origin/master # 将远程主分支合并到本地当前分支

git co --track origin/branch # 跟踪某个远程分支创建相应的本地分支

git co -b <local\_branch> origin/<remote\_branch> # 基于远程分支创建本地分支，功能同上

git push # push所有分支

git push origin master # 将本地主分支推到远程主分支

git push -u origin master # 将本地主分支推到远程(如无远程主分支则创建，用于初始化远程仓库)

git push origin <local\_branch> # 创建远程分支， origin是远程仓库名

git push origin <local\_branch>:<remote\_branch> # 创建远程分支

git push origin :<remote\_branch> #先删除本地分支(git br -d <branch>)，然后再push删除远程分支

**Git远程仓库管理**

Github

git remote -v # 查看远程服务器地址和仓库名称

git remote show origin # 查看远程服务器仓库状态

git remote add origin git@ github:robbin/robbin\_site.git # 添加远程仓库地址

git remote set-url origin git@ github.com:robbin/robbin\_site.git # 设置远程仓库地址(用于修改远程仓库地址) git remote rm <repository> # 删除远程仓库

**创建远程仓库**

git clone --bare robbin\_site robbin\_site.git # 用带版本的项目创建纯版本仓库

scp -r my\_project.git git@ git.csdn.net:~ # 将纯仓库上传到服务器上

mkdir robbin\_site.git && cd robbin\_site.git && git --bare init # 在服务器创建纯仓库

git remote add origin git@ github.com:robbin/robbin\_site.git # 设置远程仓库地址

git push -u origin master # 客户端首次提交

git push -u origin develop # 首次将本地develop分支提交到远程develop分支，并且track

git remote set-head origin master # 设置远程仓库的HEAD指向master分支

也可以命令设置跟踪远程库和本地库

git branch --set-upstream master origin/master

git branch --set-upstream develop origin/develop

# 更多：

下载GitHub客户端：<https://desktop.github.com/>

了解更多关于GitHub：<http://git-scm.com/>

了解更多关于Git：<http://git-scm.com/book/en/v2/Getting-Started-Git-Basics>

Pro Git：<http://git-scm.com/book/en/v2>