编程 Git Java GitHub C++

关注者 31871 被浏览 2172980

怎样使用 GitHub？

具体的使用方法和相关解释英语不好一点看不明白显示全部

22 条评论 分享 邀请回答

关注问题 写回答

120 个回答 默认排序

珊姗是个小太阳

答案被收藏是点赞三倍的娃儿：）你们敢赞吗！

12533 人赞同了该回答

作为一个文科妹子，我在看过几乎所有热门 github 教程之后依旧一头雾水，在近半年的摸索中终于明白啦~新年初，把自己纯小白的学习经验分享一下吧！

#什么是 Github ？

必须要放这张图了！！！

（图片来源GitHub 是怎样的一个存在？ - Deep Reader 的回答）

> Git 是由 Linux 之父 Linus Tovalds 为了更好地管理linux内核开发而创立的分布式版本控制／软件配置管理软件。

好吧，我相信看到这里你已经晕了，这也是我一开始看那些所谓经典教程的感受。写这些教程的人都是几年以上的程序员呀，他们往往直接就告诉你所有命令的含义或者整个体系。

> 专家盲点（expert blind spot）就是对一个事物知道的越多，就越发不记得“不知道这个事”的情形。

简单来说，**Git** 是一个管理你的「代码的历史记录」的工具。

我不是程序员为什么要学这个啊啊啊！又不要管理代码们！

别急，虽然 github 学习门槛高，一会你就知道为什么人人都应该会这个啦！

学习步骤

##注册安装
去官网注册一个账号（这个你应该会，恩就不放链接了）
然后，下载一个GitHub Desktop　mac客户端是最方便的啦！（命令行什么的真的是会越来越晕！先别管他们！）

假设33（珊姗就是我啦）、小四和你三个人一起写一本小说（澄清一下，并没有黑任何人的意思，恩！），暂且叫做...《梦里花落爱吃土时代》
--
图（脑补）
--
(⊙v⊙)嗯！终于可以正式开始了！

#step1:创建新项目

我们三个人在不同的城市要远程共同写一本书，要有一个漂亮的笔记本吧？

「repositories」就是你的笔记本们。你只需知道 Repository 是个放项目的地方就行。有时候会出现 Repositories，是多个 Repository 的意思。

fork
如果你不想新建一个笔记本，看到小四之前写过一个好到炸裂的文章，想把他的直接全部偷过来，修改修改就成你自己的文章了，这应该怎么办呢？
github 还提供了一个很赞的功能叫做 fork ，你只需要点击这个神奇的按钮，就可以把他的「笔记本」变成你自己的啦！任意修改都可以哦~

#step2：把「笔记本」克隆到本地

「笔记本」在云端，你要把它摘下来放到自己的电脑上写小说才方便呀，在这里我们叫「clone」是不是很形象？步骤如图：

或者是直接去我们的客户端

#step3：可以开始写作啦！

你的笔记本里已经自动有一个文档了，这个时候让我们回到网页版[微笑脸]

你只需要在 web 端点开这个README.md可以开始在里面写你的小说了。

或者直接点开刚刚 clone 到电脑上的文件夹直接在里面写。
ps:需要注意的是，文本支持 markdown 格式，可以先参考这个献给写作者的 Markdown 新手指南
。

#step4:上传你写的小说
在本地写完之后你要上传到云端让我和小四都能看见你写出什么幺蛾子了吧？
回到客户端，你发现有变化！！！

没错，在你头像旁边给你这次提交内容起一个名字，以后如果再次寻找的时候会很方便。然后点下面的 Commit to master，还有右上角的 Sync 就好啦！

#step5：回退到之前的版本
夜深人静的时候，我趁着你们都在睡觉把小说的结局偷偷地改成女主死掉了！
你醒来觉得我这结局改的也太悲伤了，完全不能接受！结局必须要和之前那样王子公主幸福的生活在一起的 happy ending！
问题又来了，怎么退回到我修改结局之前的 happy ending？

还是刚刚那个客户端，选择History 然后点击小齿轮，选择潇洒地点 roll back to this commit！
你又回到happy ending的状态啦！！

#step6：

小四写了一章华丽无比的番外，你要更新本地的小说和他写的保持一致怎么办？

git pull

好了，知道这些基本操作入门应该够了，我们来回顾一下（不要嫌弃我的画工啊喂！）

入门初期迅速得到一些正反馈对于学习一门新技能来说实在是太重要了！尤其是编程这么炫酷的事情！

所以先不要管什么复杂的 issue 呀 wiki 呀乱七八糟的操作，按照上面的一步一步来，如果遇到什么问题 google 之，一般都会解决的。

有一个段子不就是说，当你遇到问题去找最高级的工程师，他们一般都会直接 google 吗？而且自带的帮助手册也是解决问题的好办法，比如你要新建一个 branch=》Create a new branch with git and manage branches · Kunena/Kunena-Forum Wiki · GitHub

这种遇到问题先自己尝试解决的小技巧，也是我自从学编程以来最大的收获。

#除了写代码你还可以用 **github** 做什么？

回到文章开头，我又不是程序猿不用写代码玩这个干啥？

你有没有碰到过团队里几个人共同协作写一个文档的时候？或者说需要反复修改的东西？比如最简单的写论文，用 word 保存一个一个版本 e-mail 给 boss？下次再找上次修改了什么地方简直要死啊有木有！！！

相信你看了我的远程协作写小说的例子应该已经明白了， github 说白了就是一个「版本控制工具」。我们所谓的「回退」到历史记录，随时查看更改了什么地方，利用这个功能可以做的事情简直太多啦！

就像 github 其中一位创始人[Chris](defunkt (Chris Wanstrath) · GitHub)也详细描述了[GitHub初创的前因后果](Startup Riot 2009 Keynote 路 GitHub)，他说道：

> Do whatever you want.

所以不是程序猿可以用这个来做什么呢？

1、写书

和 33 一起写小说的例子，还记得吧？几个人你一章我一章共同修改一本书，或是几个出版社的编辑对新书进行校对，利用这个神器就可以随时看到哪里出现了问题和更改。如果想自己写书的话

gitbook 也是不错的选择（又是一个坑。。）

2、写文档神器

身为科研狗、产品狗、射鸡湿的你，是不是经常写文档？一个成熟的文档可能会有好几个版本，需要不断地迭代，然后不断提交给老板看哪里需要修改。在不同版本间自如切换就要用到git branch和git rebase了。

想想看，用 git 的分支管理不比拷贝粘贴更方便吗？

3、健身

有个哥们为了激励自己健身把每日计划都放上去了，还可以邀请其他人一起来相互监督！（我才不会说我自己也开了一个呢哈哈哈）

hoosin/EveryDaySport · GitHub

4、找男票

没错，看这个项目！利用众包的形式一起罗列男友条件的 list 然后试图自己开发出一个男票233333

YixuanFranco/YourBoyfriend · GitHub

有人评论问我用这个找到男票了吗？

统一回复：

并！没！有！

5、用**GitHub**搭建博客、个人网站或者公司官网

一个有自己域名的独立博客，是不是很帅？！

GitHub本身提供免费的托管服务，又提供了贴心的 Pages 功能，可以绑定你自己的域名，免费、高效、不限流量，做一个个人页面绰绰有余。

Jekyll 的教程和我自己的博客会稍后放出。。（先给自己挖个坑）

6、用**GitHub**协作翻译

苹果官方发布的各种官方手册，比如最近开源的 Swift numbbbbb/the-swift-programming-language-in-chinese · GitHub 就是国内一个自发组织起来的团队，30多个人用9天时间即将翻译和校对工作全部完成，他们每人都还有自己的事情，上班、上线、创业，这么大的工作量在以往简直是不可能完成的任务！

7、项目管理

GitHub最初是为了开发的管理而生，当然也就具备了项目管理的潜质，特别是与开发密切联系的项目中，它的优势尽显。比如这篇文章介绍了如何使用GitHub结合 Trello 等其它工具进行项目管理：使用GitHub进行团队合作 。当然，GitHub还是很偏重开发的管理，一般的项目管理还是适合使用 wortile 之类的产品。

7、政府文件？

之前看到一个知乎回答说：日本政府把宪法放上去了，德国政府也做过类似的事：German Federal Law Now on GitHub。除了德日之外，英美在 GitHub 上也有很多公众服务：英国政府多达 10 页的项目目录：Government Digital Service · GitHub 其中很多是政府项目的源代码或者设计原则之类。
芝加哥的公开地理信息：Forking your CityNew York Open City： City of New York 路
（原谅我找不到这个回答了，欢迎补充）

8、科研项目及数据

较早的arXiv 、PLoS 之外，较有气象的可以推荐mendeley 、开放期刊目录
教育方面：

- OpenStudy ：一个社会性学习网络，通过互助来更好地学习，主题涉及到计算机、数学、写作等。
- openhatch : 通过练习、任务等帮助新手更好地进入开源社区

9、个人简历

GitHub上的代码无法造假，也容易通过你关注的项目来了解你的知识面的宽度与深度。现在越来越

多知名公司活跃在GitHub，发布开源库并招募各类人才，例如：Facebook 、Twitter 、Yahoo ...

开始有了第三方网站提供基于GitHub的人才招聘服务，例如：

- GitHire :通过它，可以找出你所在地区的程序员。
- Gitalytics.com ：通过它，能评估某位程序员在GitHub、LinkedIn、StackOverflow、hackernews等多个网站的影响力。

甚至专门有一个项目就是自动根据你的 GtiHub 公开项目创建个人简历：
我们可以使用 Git 以及 GitHub 做哪些事情？ - Kane Blueriver 的回答

10、设计资源库（重点来了！！！）
做 ppt 不知道到哪里去找高质量美图？
最近半年初入设计圈，收集了不少 bookmark 想在年底来一个总结。 于是自己创建了这个Design-Resource List 项目，旨在让更多的设计师找资源变得有章可循。

先更新一部分，大概还有200多个还没放过来。。（吐血） 所以，欢迎大家也推荐自己收藏的资源，加入这个项目并一起持续更新么么哒 ：）
timmy3131/design-resource · GitHub

11、Explore · GitHub 更多好玩的内容等你自己发现哦
你在 GitHub 上看到过的最有意思的项目是什么？ - 调查类问题

#更多高阶教程：
如果你已经不满足于上面的基础知识了，欢迎探索更高级的玩法！
1、GitCafe / Help
2、[git简明指南](git - the simple guide)墙裂推荐！漫画的形式很形象（恩我承认比我画的好看多了）

3、在线交互学习 github 的网站Learn Git Branching 这个也很好玩~

4、[GitHub自身的官方博客](The GitHub Blog · GitHub)

5、git-flow 备忘清单

入门书籍推荐：

GitHub入门与实践 (豆瓣) 比较基础

Pro Git (豆瓣) 更高级的教程，很全面！

对了对了，还有阳志平老师的两篇非常全面的旧文（这么称呼好生疏啊2333）

如何高效利用GitHub

Git与Github入门资料

(⊙ o ⊙)啊！知乎居然还不支持 markdown 心好累。。

祝大家新年快乐。

ps：有朋友问我真的用 github 来写小说吗？

o(╯□╰)o只是举例子啊！方便大家理解而已...

还是会写一点点代码的(*/ω＼*)

欢迎各位程序员哥哥们纠错呀，别忘了点赞赞赞！！！！！

编辑于 2016-01-02

12K 410 条评论 分享 收藏 感谢 收起

杨晓辉

在校大学生

1245 人赞同了该回答

这是我在学习**github**的时候顺便写的教程，简单明了。

2017.7.19更新

添加新旧对比图

2016.9.24更新

在使用技巧中添加一条查看代码比例。

2016.8.28更新

添加GitHub for Windows使用教程（四）

主要是一些github使用技巧。

博客地址

GitHub for Windows使用教程（一）

GitHub for Windows使用教程(二)

GitHub for windows使用教程（三）

Github for windows使用教程(四)

如若有错，还望指正。

如果你还不知道什么是git，只知道github，但是还不会用，我想这个教程会帮助你。

前言

鉴于网上目前的教材都太落后，github for windows已经更新了多个版本，好多界面都发生了变化，所以来写这个教程。目的是为了帮助和我一样初学github，但是苦于找不到教程的同学，为了写最详细的教程。配备了大量的图文介绍。该教程是基于**GitHub for windows (3.0.17.0)**

注：

由于教程为 3.0.17.0 ，之后github对客户端进行了新版的更新，这里的图为新版与旧版对比。希望可以给大家带来帮助。

什么是**Github**

说到什么事github，我们先看wikipedia的描述“GitHub是一个利用Git进行版本控制、专门用于存放软件代码与内容的共享虚拟主机服务。它由GitHub公司（曾称Logical Awesome）的开发者Chris Wanstrath、PJ Hyett和Tom Preston-Werner使用Ruby on Rails编写而成。 ”

准备工作

1. 下载github for windows ，安装这里不赘述。
2. 注册github账号

1. 登陆到github for windows。

准备工作都完了，我们开始正式学习。^_^

创建第一个代码库

认识界面

github for windows的界面非常清爽，的确符合geek的性质，个人表示非常喜欢。 我们来建立第一个仓库，点击左上角的+号 ，初次建立他会有一圈圈的涟漪，非常漂亮哦。 打开之后有三个选项，**Add**，**Create**，**Clone**。 我们分别来介绍一下这三个功能。

Add功能

如果本地有工程，就可以使用Add添加

Clone功能

这个功能其实最好理解了，克隆这名字通俗易懂好理解。 如何使用Clone功能呢？ 就是将在浏览器上已经创建好的项目导入到本地，换句话说就是下载到本地。

Create功能

创建一个代码库， Name填写你的仓库名字。Local path写你将要保存在本地路径。

我们主要从这个功能开始github之旅。 我们在这里填写First，来创建第一个我们自己的repoeitory。

开始使用第一个代码库

修改第一个代码库中内容

我们来找到刚刚创建的代码库在本地的位置。就是刚刚在local path的地址路径，当然如果你忘了，请右键点击First。

选择Open in Explorer。这样我们就可以转到刚刚的路径下。 我们新建一个文本文档。在里面编辑。如下

此时的github就会变成这个样子(Changs)：

你会发现此时github会出现刚刚编辑的内容。

1. 这个是测试文本
2. 你好

并且前面会有蓝色标识，那么这个蓝色标识是什么用呢？

其实这个蓝色标识是提示你会上改变的文本。比如我第一次只想改变 这个是测试文本并不想把你好上传。 这时我们点击一下你好的前面的蓝色标识。

你会发现你好前面的蓝色标识没有了。

我们填写好**Summer**和**Description** Summer就是这次改动的总结，我们也可以理解为标题*（必填），而*Description*可以理解为详细概况（选填）*

我们这里只选择第一个修改对象，也就是这个是测试文本就行修改。summer我们填写为第一次修改，Description我们填写为增加了这个是测试文本的内容，之后点击**Commit to master**。

切换到**History**目录下 我们会发现他改变了。 这次我们把你好进行添加。

在**History**目录下发生了这样的改变。会在**History**目录下形成一天时间线，来指出每一次的修改标题和内容，同时会把修改的内容用绿色标识标出。 我们打开本地的文本，删除刚刚添加的第一

行这个是测试文本。

此时你就会发现github发生了变化。 此时的红色标识标识删除。我们写好Summer和Description并点击Commit to master。 这样我们就删除了第一行。同时在**History**目录下又多了一条时间轴。

这样我们就完成了删除。

上传与同步

上传

此时，当我们打开github网页，就会发现此时你的修改的内容并没有出现在这里。这是因为你没有进行同步，仅仅是在本地就行了修改。此时我们仅仅需要点击右上角的**publish**

此时你就会本地内容已经上传到网页上。

同步

当你的代码库上传后就会发现，原来的**publish**以及变为了**Sync**。 点击**Sync**同步代码库！

分支的使用

创建分支

我们创建第一个分支取名为**new masterh,**点击**Create new branch**创建第一个分支。

我们发现此时的分支已经切换到了我们刚刚创建的分支**new masterch** 。

我们来修改**new masterch**分支上的内容。 我们仍旧打开**FirstDemo.txt**进行编辑。输入以下内容

> 创建的第一个分支。

打开github进行，填写**Summary**和**Description**

之后我们点击**Commit to new-master**在**History**目录下，我们可以看到会有两条主线，分别是**master**和**new-master**并且在**new-master**的分支下又一个蓝色的实线空心圈和一个虚线空心圈。

实线圈表示当前的节点，空心圈表示下一次修改时的节点。

红线标示的部分就是当前的分支

切换分支

点击红色划线部分就会出现分支的列表

我们点击**master**就会切换到**master**分支。

上传/同步分支

这个操作和同步仓库是一个操作，点击**Publish/Sync**上传或同步分支。

删除分支

首先要把分支切换到你要删除的分支下，如我们要删除**new master**，将分支切换到**new master**

点击右上角齿轮就会出现**Delete new master** 点击**Delete new master**就会弹出一个对话框，询问删除的内容。

第一个**yes ，Delete both**是将本地与网页全部删除；

第二个**Delete local only**仅仅是删除本地。

第三个是取消。

合并两个分支

将一个分支与**master**分支进行合并。 我们首先把分支切换到**master**下，点击**Update from new-branch**进行分支的合并。

此时我们查看**history**目录下就会

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.

基本的一些用法就完成了。看着这个操作一遍基础就差不多了。

这个知乎的编辑啊.....好累

在上述的几个教程里讲解了一些Github的基础使用，现在开始讲解一些使用技巧。

查找内容

在github页面上是没有搜索的按钮，如何搜索呢。 在网页上按**T**就会出现。 这样我们就能很方便的查找到我们需要的代码了。

评论小表情

常常在版本描述或者**pull request**时我们需要对伙伴的代码进行一下评论与说明，光是文字有点很死板，其实github给我有**emoji**，如何使用呢？其实很简单，只需要冒号就可以 ：

这样我们就可以看到emoji表情，当然默认会显示五个常用的，你也可以继续敲下emoji的名字，出现更多（这里有所有的表情 ）。

忽略不想上传的文件

有些在github中的文件我们是不想上传的，我们如何过滤掉它们呢？ 在github中对不想上传的文件点击右键。就会出现下面选项。 **Ignore file**忽略这个文件 **Ignore all.txt files** 忽略所有的以.txt结尾的文件 这样就可以过滤掉你不想上传的文件

搜索项目

如何高效的搜索一个你想要的库呢？我们常常评判一个项目的标准有star数目，fork数目和跟新时间。通过搜索命令

stars

```
stars:>1000
```

表示star数目大于1000。

fork

```
fork:>1000
```

表示fork数目大于1000。

语言搜索

java，html等等

综合一下就是，比如你要查找一个stars大于1000的，fork大于200的java代码。

```
stars:>1000 fork:>200 java
```

就是这样。

查看项目中的语言类型

一个项目中，可能使用了多种语言，我们如何一下子就能看到一个项目使用了什么语言？其实很简单，Github已经为我们统计好了。

也行你注意过，但是没有发现它有什么用。

点击下面的彩条

github已经为我们统计好这个项目所有的语言及其比例。

总结

码字不易，终于写完了，如果觉得对你有帮助，我的目的就达到了。

谢谢

如有错误，还望指正。

不要光收藏，点个赞

转载请通知作者！

谢谢

编辑于 2017-07-22

查看更多回答

13 个回答被折叠（为什么？）