
Table of Contents

序言	1.1
初识 GitHub	1.2
加入 GitHub	1.3
Git 速成	1.4
向GitHub 提交代码	1.5
Git 进阶	1.6
团队合作利器：Git 分支詳解	1.7
GitHub 常见的几种操作	1.8
发现好用的开源项目	1.9

序言

我自己接触 GitHub 较早，可以说在 GitHub 在国内还没怎么普及、流行的时候就开始接触使用了，之后对我的工作以及思维方式产生了很大的影响，也大大提升了自己的开发效率与个人能力。从第一个使用的开源项目，到自己的第一篇博客，再到后面自己的第一个开源项目，都享受着 GitHub 给我带来的好处。

后面渐渐的自己也热衷于分享，拥抱开源，从博客，到公众号都在坚持写文章，分享自己过来人的技术积累、职场经验、人生总结等，甚至可以说是 GitHub 影响了我一生。

有一天我突然发现，关注我公众号的读者们，很多竟然没听说过 GitHub，或者部分听说过但是也没怎么使用过，这真的是巨大的一个损失啊，于是，应读者要求，我准备自己从 0 开始，写一篇针对初学者的 GitHub 教程，没想到，利用自己业余时间，持续了几个月，竟然形成了一个系列，评价也相当不错。

这个系列最初反响不错之后，甚至有出版社找我出书，还有部分平台找我合作，希望我出这个教程来进行销售，说实话，开的条件也都还不错的，可是我都一一拒绝了，原因很简单，既然答应了要出这个系列，就必须兑现承诺，最终我终于一篇篇在公众号上全部更新完了。

后面我的公号新增了不少读者，可能不少人不知道有这个系列，姑且马上过年了，趁这个机会我整理了一下，并把它做成一个电子书送给你们，以后需要的时候，拿出来看一下，说不定就对你有帮助。

这个教程包括如下内容：

1. 初识 GitHub
2. 加入 GitHub
3. Git 速成
4. 向 GitHub 提交代码
5. Git 进阶
6. 团队合作利器：Git 分支详解
7. GitHub 常见的几种操作
8. 发现好用的开源项目

希望通过这个教程，人人可以很方便的掌握 Git/GitHub 的使用。

最后，你还可以通过以下其他方式找到我：

GitHub：<https://github.com/stormzhang>

个人博客：<http://stormzhang.com>

微博：googdev

知乎：stormzhang

如果想获取其他更多原创分享，欢迎关注我的微信公众号 stormzhang。



版权声明：

本系列内容首发于我的微信公众号 **stormzhang**，原创作者 stormzhang，个人博客：<http://stormzhang.com>，可以随意转载，但必须保持署名，禁止商用。

初识 GitHub

1. 写在前面

我一直认为 GitHub 是程序员必备技能，程序员应该没有不知道 GitHub 的才对，没想到这两天留言里给我留言最多的就是想让我写关于 GitHub 的教程，说看了不少资料还是一头雾水，我转念一想，我当初接触 GitHub 也大概工作了一年多才开始学习使用，我读者里很多是初学者，而且还有很多是在校大学生，所以不会用 GitHub 也就不奇怪了，所以我觉得写一写关于 GitHub 的教程就非常有必要了！

2. 为什么还要造轮子

很多人难免要问这个问题，说网上关于 GitHub 的资料很多，为什么还要写呢？讲真，网上关于 Android 的资料更多，为什么你们还喜欢看我写的文章呢？是因为哪怕同样的内容，我写出来之后就有了我的风格，除了我的幽默以及我的帅，关键的是我有办法让你们看的轻松易懂，并且还有我个人的一些见解与指导，这大概是一种特殊的魅力吧！

我是从小白一路过来的，很能理解你们内心的感受与困惑，因为这些阶段都是我自己亲身经历过的，所以我写的文章都会从你们的角度去出发，并且我对文章高要求，除了排版、配图很用心外，文章的内容每次写完我都会亲自看三四遍，确保不会出现误导以及你们理解不了的情况，你们看的很轻松易懂的文章其实因为我背后做了很多的功课。

所以，为了你们，我觉得有必要用我的风格去教你们如何从0开始，跟着我一步步学习 GitHub！

3. 什么是 GitHub

确切的说 GitHub 是一家公司，位于旧金山，由 Chris Wanstrath, PJ Hyett 与 Tom Preston-Werner 三位开发者在2008年4月创办。这是它的 Logo：



2008年4月10日，GitHub正式成立，地址：[How people build software · GitHub](https://www.github.com)，主要提供基于git的版本托管服务。一经上线，它的发展速度惊为天人，截止目前，GitHub 已经发展成全球最大的开（同）源（性）社区。

4. GitHub 与 Git 的关系

这个我还专门在群里调查过，很多人以为 GitHub 就是 Git，其实这是一个理解误区。

Git 是一款免费、开源的分布式版本控制系统，他是著名的 Linux 发明者 Linus Torvalds 开发的。说到版本控制系统，估计很多人都用过 SVN，只不过 Git 是新时代的产物，如果你还在用 SVN 来管理你的代码，那就真的有些落伍了。不管是学习 GitHub ，还是以后想从事编程行业，Git 都可以算是必备技能了，所以从现在开始建议你先去学习熟悉下 Git ，后面我会有关文章推荐一些适合新手的 Git 学习资料给你们。

而 GitHub 上面说了，主要提供基于 git 的版本托管服务。也就是说现在 GitHub 上托管的所有项目代码都是基于 Git 来进行版本控制的，所以 Git 只是 GitHub 上用来管理项目的一个工具而已，GitHub 的功能可远不止于此！

5. GitHub 的影响力

上面我说了 GitHub 现在毫无疑问基本是全球最大的开源社区了，这样说你们可能认为未免有点浮夸，且听我一一举证：

全球顶级科技公司纷纷加入 **GitHub**，并贡献他们自己的项目代码

- Google: <https://github.com/google>
- 苹果: <https://github.com/apple>
- Facebook: <https://github.com/facebook>
- Twitter : <https://github.com/twitter>
- 微软 : <https://github.com/microsoft>
- Square : <https://github.com/square>
- 阿里 : <https://github.com/alibaba>
- ...

全球顶级开源项目都优先选择在 **GitHub** 上开源

- Linux : <https://github.com/torvalds/linux>
- Rails : <https://github.com/rails/rails>
- Nodejs : <https://github.com/nodejs/node>
- Swift : <https://github.com/apple/swift>
- CoffeeScript : <https://github.com/jashkenas/coffeescript>
- Ruby : <https://github.com/ruby/ruby>
- ...

全球顶级编程大牛加入 **GitHub**

- Linux 发明者 Linus Torvalds : <https://github.com/torvalds>



Linus Torvalds
torvalds

Linux Foundation
 Portland, OR
 Joined on Sep 3, 2011

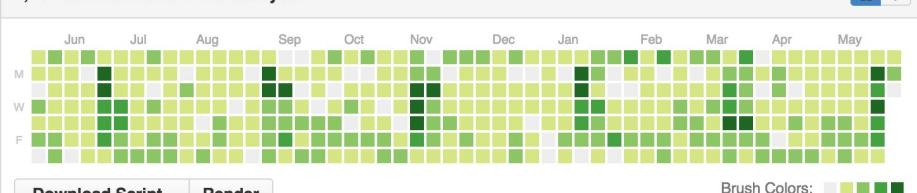
37.8k Followers **2** Starred **0** Following

[Contributions](#) [Repositories](#) [Public activity](#) [Follow](#) [Block or report](#)

Popular repositories

- linux** Linux kernel source tree 32,519 ★
- subsurface** Advanced multi-platform divelog based on Qt 965 ★

2,181 contributions in the last year



Download Script... Render Brush Colors: ■ ■ ■ ■

Contribution activity Period: 1 week

- Rails 创始人 DHH : <https://github.com/dhh>



David Heinemeier Hansson
dhh

Basecamp
 Chicago, USA
 david@basecamp.com
 <http://david.heinemeierhansson.org>
 Joined on Mar 11, 2008

8.2k Followers **28** Starred **0** Following

[Contributions](#) [Repositories](#) [Public activity](#) [Follow](#) [Block or report](#)

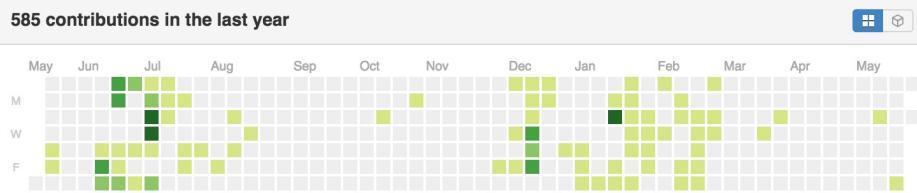
Popular repositories

- tolk** Tolk is a web interface for doing i18n translation... 514 ★
- custom_configuration** Custom configuration storage for Rails 204 ★
- asset-hosting-with-minimum-ssl** Rails plugin for picking a non-ssl asset host as... 78 ★
- conductor** 65 ★
- delayed_job** Database based asynchronously priority queue.. 32 ★

Repositories contributed to

- rails/rails** Ruby on Rails 31,271 ★
- rails/actionable** Framework for real-time communication over w... 1,120 ★
- rails/homepage** rubyonrails.org site anno 2015 16 ★
- rails/weblog** 18 ★
- rails/actionable-examples** Action Cable Examples 303 ★

585 contributions in the last year



Download Script... Render Brush Colors: ■ ■ ■ ■

Organizations

- 被称为「Android之神」的 JakeWharton : <https://github.com/JakeWharton> , 你们用的很多开源库如 ButterKnife、OkHttp、Retrofit、Picasso、ViewPagerIndicator 等都是出自他之手 !

Popular repositories

butterknife	9,271 ★
Bind Android views and callbacks to fields and...	
ViewPagerIndicator	7,482 ★
Paging indicator widgets compatible with the V...	
ActionBarSherlock	7,277 ★
[DEPRECATED] Action bar implementation wh...	
u2020	3,703 ★
A sample Android app which showcases adva...	
NineOldAndroids	3,504 ★
[DEPRECATED] Android library for using the H...	

Repositories contributed to

square/retrofit	12,641 ★
Type-safe HTTP client for Android and Java by...	
square/wire	1,482 ★
Clean, lightweight protocol buffers for Android ...	
square/sqlodelight	665 ★
Generates Java models from CREATE TABLE...	
square/okhttp	11,429 ★
An HTTP+HTTP/2 client for Android and Java ...	
square/sqlbrite	2,591 ★
A lightweight wrapper around SQLiteOpenHelper...	

3,527 contributions in the last year

Brush Colors: ■ ■ ■

Organizations

- Square, Inc.
- Robolectric
- GRPC
- Dagger

其他就不一一列举了，GitHub 上活跃的很多是 Google、Square、阿里等公司的员工，有些甚至还是 Google Android Team 组的，所以在这里你可以接触到全球顶级编程大牛！

6. GitHub 有什么用

- 学习优秀的开源项目

开源社区一直有一句流行的话叫「不要重复发明轮子」，某种意义上正是因为开源社区的贡献，我们的软件开发才能变得越来越容易，越来越快速。试想你在做项目时，如果每一模块都要自己去写，如网络库、图片加载库、ORM 库等等，自己写的好不好是一回事，时间与资源是很大的成本。对于大公司可能会有人力与资源去发明一套自己的轮子，但是对于大部分互联网创业公司来说时间就是一切。而且你在使用开源项目的过程也可以学习他们优秀的设计思想、实现方式，这是最好的学习资料，也是一份提升自己能力的绝佳方式！

- 多人协作

如果你想发起一个项目，比如翻译一份不错的英文文档，觉得一个人的精力不够，所以你需要更多的人参与进来，这时候 GitHub 是你的最佳选择，感兴趣的人可以参与进来，利用业余时间对这个项目做贡献，然后可以互相审核、合并，简直不要太棒！

- 搭建博客、个人网站或者公司官网

这个就不用多说了，现在越来越多的博客都是基于 GitHub Pages 来搭建的了，你可以随心所欲的定制自己的样式，可以给你博客买个逼格高的域名，再也不用忍受各大博客网站的约束与各式各样的广告了！

- 写作

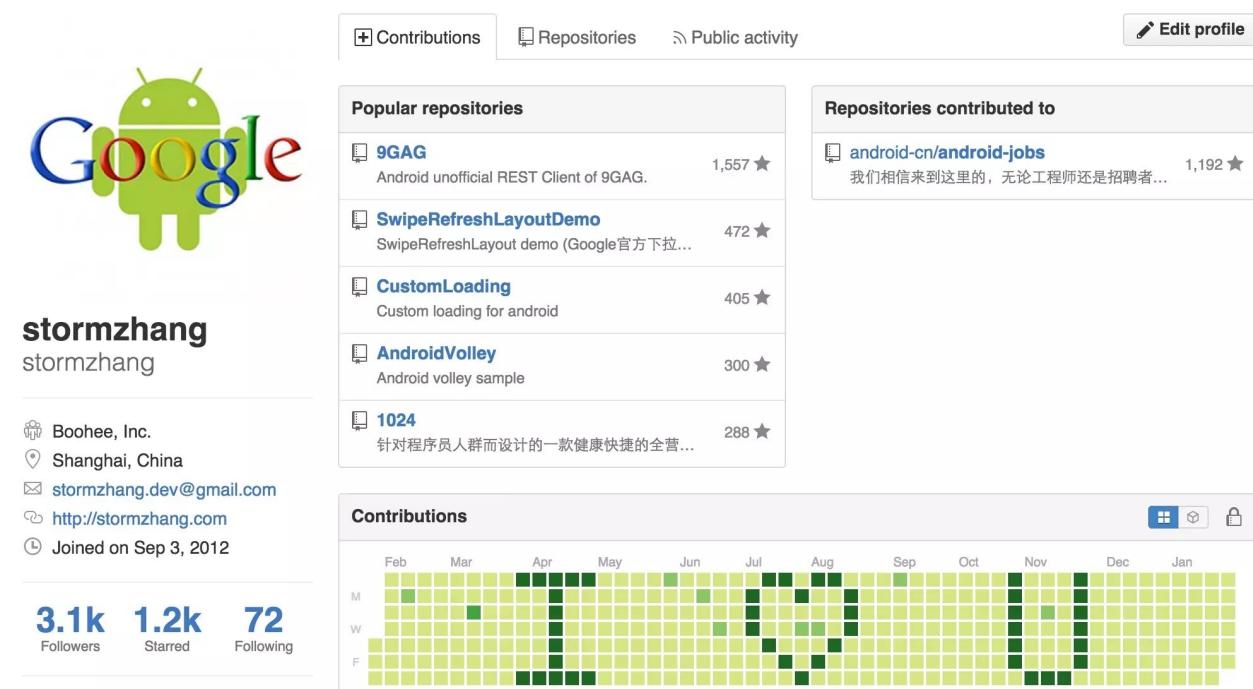
如果你喜欢写作，而且基于 Markdown，并准备出版书籍，那么推荐你用 Gitbook，技术写作人的最爱！

- 个人简历

如果你有一个活跃的 GitHub 账号，上面有自己不错的开源项目，还经常给别的开源项目提问题，push 代码，那么你找工作将是一个非常大的优势，现在程序员的招聘很多公司都很看中你 GitHub 账号，某种意义上 GitHub 就可以算是你的简历了。而且不仅国内，很多国外的科技公司都会通过 GitHub 来寻找优秀的人才，比如我甚至通过 GitHub 收到过 Facebook 的邀请邮件！

- 其他

当然 GitHub 能做的远不止这些，我见过很多在 GitHub 搞的一些有意思的项目，有找男朋友的，甚至还有利用 GitHub 的 commit 丧心病狂的秀恩爱的，没错，那个丧心病狂的人就是我，如果你前段时间关注了我的 GitHub，那么能看到这么一个壮观的景象：



7. 加入 GitHub

读完我的文章，我相信你已经蠢蠢欲动了，从现在开始，立刻、马上去注册个 GitHub 「<https://github.com/>」，去体验一番，不会用不要紧，接下来我会有一系列详细的文章，来教你学会使用 GitHub！

但是为了保证文章的质量，我要做很多准备工作，我没法保证每天都会连载，但是我会尽力尽快更新这个系列，让你们从0开始一步步一起来学习，如果周围有同学或者朋友想要学习的，那赶紧转发或者推荐他关注这个系列的文章，毕竟有个小伙伴一起学会更有氛围，后续除了理论我还会考虑结合实践，我不信你学不会！

版权声明：

本系列内容首发于我的个人微信公众号 **stormzhang**，原创作者 **stormzhang**，个人博客：<http://stormzhang.com>，可以随意转载，但必须保持署名，禁止商用。

加入 GitHub

看完上篇文章，估计不少人已经开始期待我继续更新了，这不赶紧马不停蹄，加班加点给你们更新了第二篇。在更新本篇文章之前先回答昨天大家留言的两个问题：

- GitHub 需要翻墙么？

印象中 GitHub 之前确实总是断断续续的访问不了，不过在13年初的时候有段时间最严重，一度被封了，当时李开复老师再也忍无可忍，公开发了一条抗议 GitHub 被封的微博，这事我印象很深，因为我是12年底加入的 GitHub，那时候简直像遇到世外桃源一般，但是也深受老是访问不了的困扰，很多人早就对这件事怨声载道了，加上李开复老师的声讨，这一下就炸开了锅，微博上纷纷转发谴责，算的上是整个IT界的大新闻，后来因为这事影响太大了，没过几天 GitHub 就可以正常访问了，这里真的要感谢李开复老师敢于站出来的勇气，可以这么说，如果没有 GitHub，中国的编程水平起码要倒退好多年！

因为 GitHub 的影响力太大，基本上是各种黑客攻击的对象，所以现在偶尔也会有宕机访问不了的情况，但是好在不会被封，所以大家不用担心，访问 GitHub 不用翻墙，只是可能访问速度稍慢些，另外为了维护一个和谐的环境，这里也呼吁大家不要在 GitHub 上发表任何关于政治的言论与文章，在 GitHub 上我们只是单纯的技术交流，无关政治，在复杂的大环境下，希望 GitHub 永远是我们程序员的一片净土！

- 英语差、0基础学得会么？

这个也是不少人问我的，GitHub 虽然都是英文，但是对英语水平的要求不是那么高，都是些简单的单词，遇到不会的查一下就行了，你觉得很难只是你对英文网站反射性的抵触而已，相信我，跟着我的详细教程，我的文章面向从没有接触过甚至没有听过 GitHub 的同学，一步步教你由浅入深。如果你学不会，那么来打我，不过我这么帅，你也不忍心！

好了，废话不多说，咱们进入正文！

1. 注册 GitHub

先去 GitHub 官网「How people build software · GitHub」注册「Sign Up」个账号，注册页面如下：

Join GitHub

The best way to design, build, and ship software.



Step 1:
Set up a personal account



Step 2:
Choose your plan



Step 3:
Go to your dashboard

Create your personal account

Username

This will be your username — you can enter your organization's username next.

Email Address

You will occasionally receive account related emails. We promise not to share your email with anyone.

Password

Use at least one lowercase letter, one numeral, and seven characters.

By clicking on "Create an account" below, you are agreeing to the [Terms of Service](#) and the [Privacy Policy](#).

Create an account

这个应该没啥说的，需要填用户名、邮箱、密码，值得一提的用户名请不要那么随便，最好取的这个名字就是你以后常用的用户名了，也强烈建议你各大社交账号都用一样的用户名，这样识别度较高，比如我的博客域名、GitHub、知乎等其他社交账号 ID 都是 stormzhang，微博是因为被占用了，无奈换了个id，而且这个用户名以后在 GitHub 搭建博客的时候默认给你生成的博客地址就是 <http://username.github.io>，所以给自己取个好点的用户名吧。

填好用户名、邮箱、密码紧接着到这一步：

You'll love GitHub

Unlimited collaborators

Unlimited public repositories

- ✓ Great communication
- ✓ Friction-less development
- ✓ Open source community

Welcome to GitHub

You've taken your first step into a larger world, @googdev.

✓ Completed Set up a personal account	□ Step 2: Choose your plan	⌚ Step 3: Go to your dashboard
---	---	---

Choose your personal plan

- Unlimited public repositories for free.
- Unlimited private repositories** for \$7/month. ([view in CNY](#))

Don't worry, you can cancel or upgrade at any time.

Help me set up an organization next

Organizations are separate from personal accounts and are best suited for businesses who need to manage permissions for many employees.
[Learn more about organizations.](#)

[Finish sign up](#)

这个是什么意思呢？GitHub 有两种，一种是公开，这种是免费的，就是你创建的项目是开放的，所有人都能看到；另一种是私有，这种是收费的，这种一般是一些企业在使用 GitHub 的私有仓库在托管自己的项目，这也是 GitHub 的一种盈利模式对于个人你就直接默认选择公开的就行了。

2. 认识 GitHub

注册成功之后你会到 GitHub 的主页面来：

The screenshot shows the GitHub homepage with several annotations:

- 1. 导航栏**: Points to the top navigation bar which includes the GitHub logo, a search bar, and links for Pull requests, Issues, and Gist.
- 2. 我的Timeline**: Points to the left sidebar where it lists recent starred repositories by other users.
- 3. 我的一些项目**: Points to the right sidebar titled "Your repositories" which lists the user's own repositories.

The main content area displays the user's timeline, showing recent activity from other users like "webaihai starred stormzhang/9GAG".

Your repositories 24	
Find a repository...	
All Public Private Sources Forks	
stormzhang.github.com	
slides	
9GAG	
awesome-java-cn	
magic_commit	
1024	
react-native	
AndroidVolley	
DroidPlugin	
BooheeToolbar	

你如果是新注册的可能看到的跟我不一样，因为你们新用户，没有自己的项目，没有关注的人，所以只有一个导航栏。

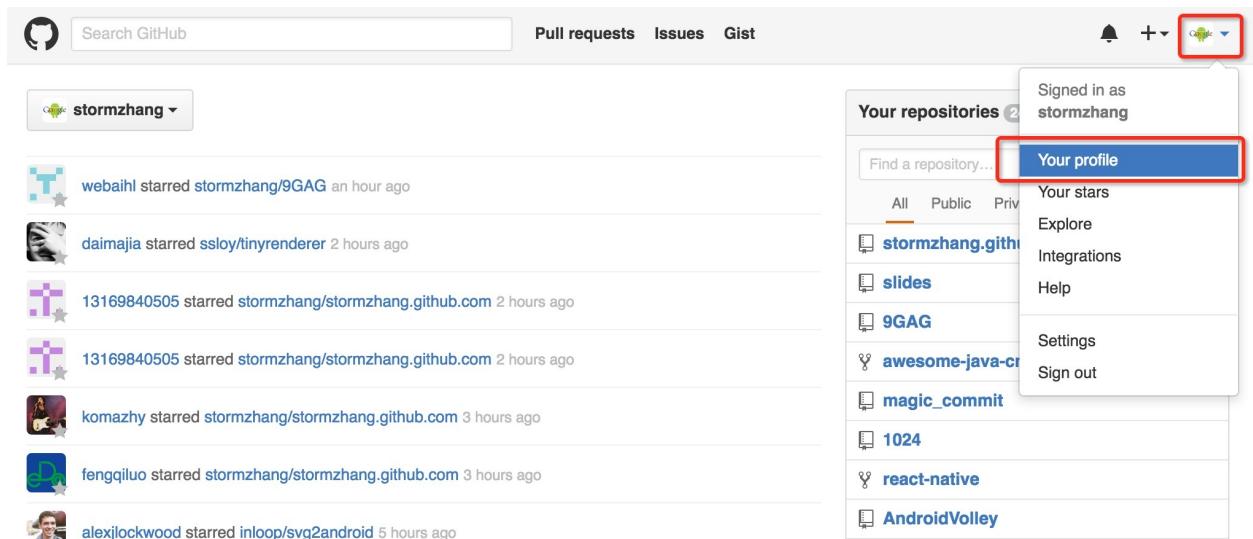
导航栏，从左到右依次是 GitHub 主页按钮、搜索框、PR、Issues、Gist（这些概念后面会讲的）、消息提醒、创建项目按钮、我的账号相关。

我的 Timeline，这部分你可以理解成微博，就是你关注的一些人的活动会出现在这里，比如如果你们关注我了，那么以后我 star、fork 了某些项目就会出现在你的时间线里。

我的项目，这部分就不用说了，如果你创建了项目，就里就可以快捷访问。

3. GitHub 主页

点击下图的 Your profile 菜单进入到你的个人 GitHub 主页。



还是以我的 GitHub 主页为例：

我的一些流行开源项目

Popular repositories

- 9GAG (1,557 stars)
- SwipeRefreshLayoutDemo (472 stars)
- CustomLoading
- AndroidVolley (300 stars)
- 1024

Repositories contributed to

- android-cn/android-jobs (1,192 stars)

我近期做过贡献的开源项目

别人给我项目的送的点赞数

我在GitHub上提交代码的记录，每一小格代表一天，颜色深浅按照当天提交次数的多少来算。

Contributions

关注我的人 我关注的人

stormzhang

stormzhang

我送出的star数

Boohee, Inc.
Shanghai, China
stormzhang.dev@gmail.com
<http://stormzhang.com>
Joined on Sep 3, 2012

3.1k Followers 1.2k Starred 72 Following

这么详细应该不会看不懂吧？只不过你的账号可能没有这么丰富，因为你可能啥也没做过，但是如果做全了基本上就会看到跟我一样的了。

4. 设置你的 GitHub

如果你是新注册的 GitHub 账号，是不是觉得很简陋？虽然你没有自己的项目，但是第一步起码要先完善自己的信息，点击如下的 Settings 菜单：

Search GitHub

Pull requests Issues Gist

stormzhang

Your repositories

- shadowliucs starred stormzhang/CustomLoading 18 minutes ago
- shadowliucs starred stormzhang/AndroidVolley 18 minutes ago
- shadowliucs starred stormzhang/SwipeRefreshLayoutDemo 18 minutes ago
- webaihl starred stormzhang/9GAG 2 hours ago
- daimajia starred ssloy/tinyrenderer 2 hours ago
- 13169840505 starred stormzhang/stormzhang.github.com 2 hours ago

Signed in as stormzhang

Find a repository... All Public Private

stormzhang.github.com

slides

9GAG

awesome-javascript

magic_commit

1024

react-native

Settings

Sign out

到设置页面来设置一些基本信息：

The screenshot shows the GitHub profile settings interface. On the left, there's a sidebar with 'Personal settings' and a 'Profile' tab selected. The main area is titled 'Public profile' and contains fields for 'Profile picture' (with a placeholder for Google), 'Name' (set to 'stormzhang'), 'Public email' (set to 'stormzhang.dev@gmail.com'), 'URL' (set to 'http://stormzhang.com'), 'Company' (set to 'Boohee, Inc.'), and 'Location' (set to 'Shanghai, China'). A green 'Update profile' button is at the bottom.

像头像、Name 建议要设置一个常用的，这两个很有识别性，公开的邮箱也要设置一个，这样那些企业啊、猎头啊就通过这个公开邮箱去联系你，友情提醒：别在 GitHub 把自己的 QQ 邮箱放上去，不显得太 low 了么？没有 gmail 邮箱，起码也得注册个 foxmail、163 邮箱之类的吧。

5. GitHub 基本概念

上面认识了 GitHub 的基本面貌之后，你需要了解一些 GitHub 的基本概念，这些概念是你经常会接触并遇到的。

- Repository

仓库的意思，即你的项目，你想在 GitHub 上开源一个项目，那就必须要新建一个 Repository，如果你开源的项目多了，你就拥有了多个 Repositories。

- Issue

问题的意思，举个例子，就是你开源了一个项目，别人发现你的项目中有bug，或者哪些地方做的不够好，他就可以给你提个 Issue，即问题，提的问题多了，也就是 Issues，然后你看到了这些问题就可以去逐个修复，修复ok了就可以一个个的 Close 掉。

- Star

这个好理解，就是给项目点赞，但是在 GitHub 上的点赞远比微博、知乎点赞难的多，如果你有一个项目获得100个star都算很不容易了！

- Fork

这个不好翻译，如果实在要翻译我把他翻译成分叉，什么意思呢？你开源了一个项目，别人想在你这个项目的基础上做些改进，然后应用到自己的项目中，这个时候他就可以 Fork 你的项目，这个时候他的 GitHub 主页上就多了一个项目，只不过这个项目是基于你的项目基础（本质上是在原有项目的基础上新建了一个分支，分支的概念后面会在讲解 Git 的时候说到），他就可以随心所欲的去改进，但是丝毫不会影响原有项目的代码与结构。

- Pull Request

发起请求，这个其实是基于 Fork 的，还是上面那个例子，如果别人在你基础上做了改进，后来觉得改进的很不错，应该要把这些改进让更多的人收益，于是就想把自己的改进合并到原有项目里，这个时候他就可以发起一个 Pull Request（简称 PR），原有项目创建人就可以收到这个请求，这个时候他会仔细 review 你的代码，并且测试觉得 OK 了，就会接受你的 PR，这个时候你做的改进原有项目就会拥有了。

- Watch

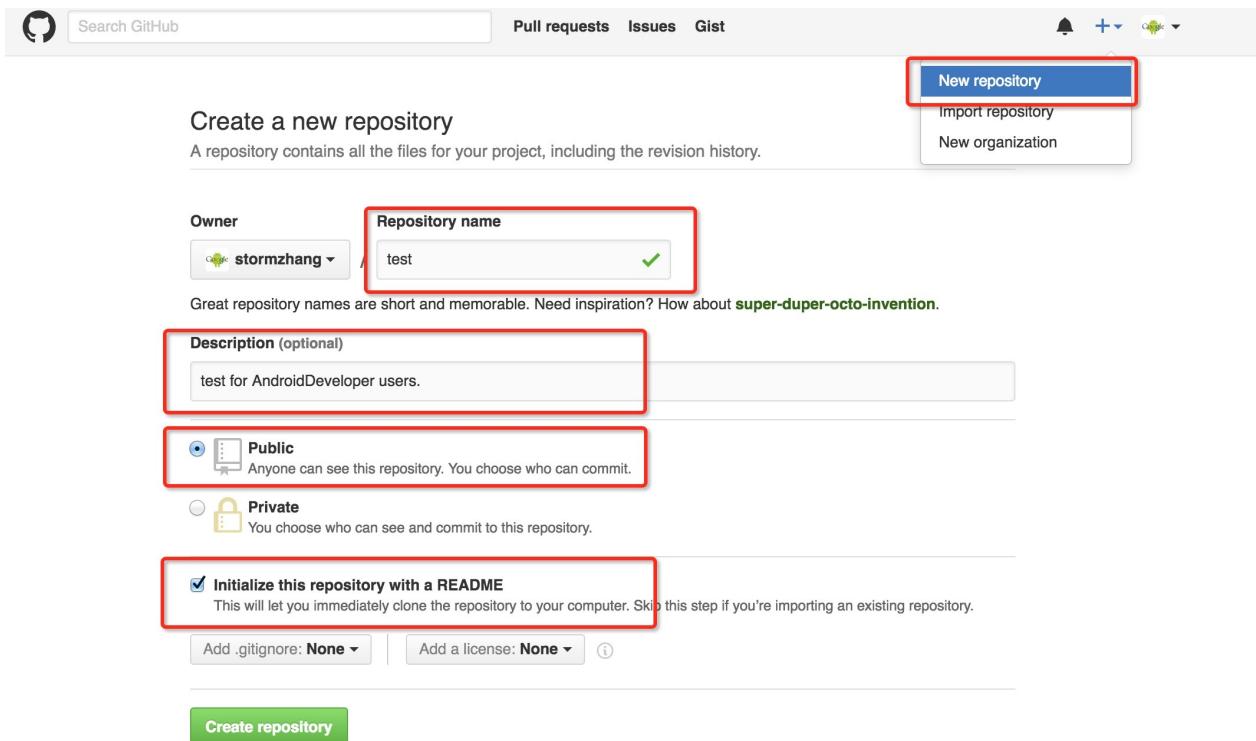
这个也好理解就是观察，如果你 Watch 了某个项目，那么以后只要这个项目有任何更新，你都会第一时间收到关于这个项目的通知提醒。

- Gist

有些时候你没有项目可以开源，只是单纯的想分享一些代码片段，那这个时候 Gist 就派上用场了！

6. 创建自己的项目

点击顶部导航栏的 + 可以快速创建一个项目，如下图：



创建一个项目需要填写如上的几部分：项目名、项目描述与简单的介绍，你不用付费也没法选择私有的，所以接着只能选择 public 的，之后勾选「Initialize this repository with a README」，这样你就拥有了你的第一个 GitHub 项目：

The screenshot shows the GitHub repository page for 'stormzhang / test'. At the top, there are buttons for 'Unwatch' (with 1 watch), 'Star' (0 stars), and 'Fork' (0 forks). Below that, there are buttons for 'Code', 'Issues 0', 'Pull requests 0', 'Wiki', 'Pulse', 'Graphs', and 'Settings'. The main area shows 'test for AndroidDeveloper users.' under the heading 'Edit'. Below this, there are stats: '1 commit', '1 branch', '0 releases', and '1 contributor'. There are buttons for 'Branch: master', 'New pull request', 'Create new file', 'Upload files', 'Find file', and 'Clone or download' (highlighted with a red box). The repository history shows a single commit from 'stormzhang' with the message 'Initial commit'. The README.md file content is displayed below, showing 'test for AndroidDeveloper users.'

可以看到这个项目只包含了一个 README.md 文件，但是它已经是一个完整的 Git 仓库了，你可以通过对它进行一些操作，如 watch、star、fork，还可以 clone 或者下载下来。

这里提一下 README.md ，GitHub 上所有关于项目的详细介绍以及 Wiki 都是基于 Markdown 的，甚至之后在 GitHub 上搭建博客，写博客也是如此，所以如果还不懂 Markdown 语法的，建议先去学习下。推荐一篇学习 Markdown 的文章给你们：

7. 总结

相信看完以上文章你已经基本算是了解 GitHub 的基本概念并且正式加入 GitHub 这个大家庭了，之后会有更深入的文章介绍 Git、介绍对项目的常用操作、介绍如何给开源项目提交代码、介绍如何协同合作甚至怎么搭建博客等，敬请期待我公众号 AndroidDeveloper 之后更新的内容吧！

版权声明：

本系列内容首发于我的个人微信公众号 **stormzhang**，原创作者 **stormzhang**，个人博客：<http://stormzhang.com>，可以随意转载，但必须保持署名，禁止商用。

Git 速成

前面的 GitHub 系列文章介绍过，GitHub 是基于 Git 的，所以也就意味着 Git 是基础，如果你不会 Git，那么接下来你完全继续不下去，所以今天的教程就来说说 Git，当然关于 Git 的知识单凭一篇文章肯定说不完的，我这篇文章先介绍一些最基本的、最常用的一些 Git 知识，争取让你们 Git 速成。

1. 什么是 Git？

Git 是 Linux 发明者 Linus 开发的一款新时代的版本控制系统，那什么是版本控制系统呢？怎么理解？网上一大堆详细的介绍，但是大多枯燥乏味，对于新手也很难理解，这里我只举几个例子来帮助你们理解。

熟悉编程的知道，我们在软件开发中源代码其实是最重要的，那么对源代码的管理变得异常重要：

比如为了防止代码的丢失，肯定本地机器与远程服务器都要存放一份，而且还需要有一套机制让本地可以跟远程同步；

又比如我们经常是好几个人做同一个项目，都要对一份代码做更改，这个时候需要大家互不影响，又需要各自可以同步别人的代码；

又比如我们开发的时候免不了有bug，有时候刚发布的功能就出现了严重的bug，这个时候需要紧急对代码进行还原；

又比如随着我们版本迭代的功能越来越多，但是我们需要清楚的知道历史每一个版本的代码更改记录，甚至知道每个人历史提交代码的情况；

等等等类似以上的情况，这些都是版本控制系统能解决的问题。所以说，版本控制是一种记录一个或若干文件内容变化，以便将来查阅特定版本修订情况的系统，对于软件开发领域来说版本控制是最重要的一环，而 Git 毫无疑问是当下最流行、最好用的版本控制系统。

2. Git 安装

上面说了，Git 是一个版本控制系统，你也可以理解成是一个工具，跟 Java 类似，使用之前必须得先下载安装，所以第一步必须要安装，我用的是 Mac，Mac 上其实系统自带 Git 的，不过这里统一提供一下各平台的安装方式，这部分就不过多介绍，相信大家这里搞的定。

- Mac : <https://sourceforge.net/projects/git-osx-installer/>
- Windows : <https://git-for-windows.github.io/>
- Linux : apt-get install git

3. 如何学习 Git ?

安装好 Git 之后，怎么学习是个问题，其实关于 Git 有很多图形化的软件可以操作，但是我强烈建议大家从命令行开始学习理解，我知道没接触过命令行的人可能会很抵触，但是我的亲身实践证明，只有一开始学习命令行，之后你对 Git 的每一步操作才能理解其意义，而等你熟练之后你想用任何的图形化的软件去操作完全没问题。

我一开始教我们团队成员全是基于命令行的，事后证明他们现在已经深深爱上命令行无法自拔，他们很理解 Git 每一步操作的具体含义，以致于在实际项目很少犯错，所以我这里也是基于命令行去教你们学习理解。

4. Git 命令列表

怎么判断你 Git 有没有安装成功？请在命令行里输入 git，如果出现以下提示证明你已经安装成功了。

```

1. storm@192: ~ /my_projects/stormzhang (zsh)
→ stormzhang git:(master) ✘ git
usage: git [--version] [--help] [-C <path>] [-c name=value] <command> [<args>]
   ★ Bookmarks [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
   [-p|--paginate|--no-pager] [--no-replace-objects] [--bare]
   [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
   <command> [<args>]

The most commonly used git commands are:
add           Add file contents to the index
bisect        Find by binary search the change that introduced a bug
branch       List, create, or delete branches
checkout     Checkout a branch 我们从头开始学习 Git，事后证明他们现在已经深深爱上了 Git。每一部分都深入浅出地讲解了 Git 每一步操作的具体含义，以至于在实际项目中遇到问题时，我们能够迅速找到解决方法。
clone        Clone a repository into a new directory
commit       Record changes to the repository
diff          Show changes between commits, commit and working tree, etc
fetch        Download objects and refs from another repository
grep          Print lines matching a pattern
init          Create an empty Git repository or reinitialize an existing one
log           Show commit logs
merge        Join two or more development histories together
mv            Move or rename a file, a directory, or a symlink
pull          Fetch from and integrate with another repository or a local branch
push          Update remote refs along with associated objects
rebase        Forward-port local commits to the updated upstream head
reset        Reset current HEAD to the specified state
rm            Remove files from the working tree and from the index
show          Show various types of objects
status        Show the working tree status
tag           Create, list, delete or verify a tag object signed with GPG

```

留言 每次勾选后，文章必须经过群发，读者才能在文章底部留言

Git 所有的操作命令开头都要以 `git` 开头，上面列举了最常用的一些 Git 命令，紧接着会有一句英文解释这个命令的意义，都不是很难的单词，不妨试着看一下，不过没有实际操作你仍然不好理解，下面我们来以一个实际的操作来介绍下一些常用命令的含义。

5. Git 具体命令

第一步，我们先新建一个文件夹，在文件夹里新建一个文件（我是用 Linux 命令去新建的，Windows 用户可以自己手动新建）

```

mkdir test      (创建文件夹test)
cd test       (切换到test目录)
touch a.md     (新建a.md文件)

```

这里提醒下：在进行任何 Git 操作之前，都要先切换到 Git 仓库目录，也就是先要先切换到项目的文件夹目录下。

这个时候我们先随便操作一个命令，比如 `git status`，可以看到如下提示（别纠结颜色之类的，配置与主题不一样而已）：

```
→ test git status
fatal: Not a git repository (or any of the parent directories): .git
→ test
```

意思就是当前目录还不是一个 Git 仓库。

git init

这个时候用到了第一个命令，代表初始化 git 仓库，输入 git init 之后会提示：

```
→ test git init
Initialized empty Git repository in /Users/storm/test/.git/
→ test git:(master)
```

可以看到初始化成了，至此 test 目录已经是一个 git 仓库了。

git status

紧接着我们输入 git status 命令，会有如下提示：

```
→ test git:(master) ✘ git status
On branch master
Initial commit
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    a.md
nothing added to commit but untracked files present (use "git add" to track)
```

默认就直接在 master 分支，关于分支的概念后面会提，这时最主要的是提示 a.md 文件 Untracked files，就是说 a.md 这个文件还没有被跟踪，还没有提交在 git 仓库里呢，而且提示你可以使用 git add 去操作你想要提交的文件。

git status 这个命令顾名思义就是查看状态，这个命令可以算是使用最频繁的一个命令了，建议大家没事就输入下这个命令，来查看你当前 git 仓库的一些状态。

git add

上面提示 a.md 文件还没有提交到 git 仓库里，这个时候我们可以随便编辑下 a.md 文件，然后输入 git add a.md，然后再输入 git status：

```

→ test git:(master) ✘ git add a.md
→ test git:(master) ✘ git status
On branch master
Initial commit
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:   a.md

```

此时提示以下文件 Changes to be committed ， 意思就是 a.md 文件等待被提交，当然你可以使用 git rm --cached 这个命令去移除这个缓存。

git commit

接着我们输入 git commit -m 'first commit' ，这个命令什么意思呢？ commit 是提交的意思， -m 代表是提交信息，执行了以上命令代表我们已经正式进行了第一次提交。

这个时候再输入 git status ，会提示 nothing to commit 。

git log

这个时候我们输入 git log 命令，会看到如下：

```

commit 7fa619d905fc9bba4938e1961b7a3056892e46b0
Author: stormzhang <zhangqi@boohee.com>
Date: Sun May 29 20:35:08 2016 +0800
Bing 薄荷
素材 / 新建图片信息
first commit

```

git log 命令可以查看所有产生的 commit 记录，所以可以看到已经产生了一条 commit 记录，而提交时候的附带信息叫 'first commit' 。

git add & git commit

看到这里估计很多人会有疑问，我想要提交直接进行 commit 不就行了么，为什么先要再 add 一次呢？首先 git add 是先把改动添加到一个「暂存区」，你可以理解成是一个缓存区域，临时保存你的改动，而 git commit 才是最后真正的提交。这样做的好处就是防止误提交，当然也有办法把这两步合并成一步，不过后面再介绍，建议新手先按部就班的一步步来。

git branch

`branch` 即分支的意思，分支的概念很重要，尤其是团队协作的时候，假设两个人都在做同一个项目，这个时候分支就是保证两人能协同合作的最大利器了。举个例子，A, B俩人都在做同一个项目，但是不同的模块，这个时候A新建了一个分支叫a，B新建了一个分支叫b，这样A、B做的所有代码改动都各自在各自的分支，互不影响，等到俩人都把各自的模块都做完了，最后再统一把分支合并起来。

执行 `git init` 初始化git仓库之后会默认生成一个主分支 `master`，也是你所在的默认分支，也基本是实际开发正式环境下的分支，一般情况下 `master` 分支不会轻易直接在上面操作的，你们可以输入 `git branch` 查看下当前分支情况：

```
→ test git:(master) git branch
* master https://mp.weixin.qq.com/cgi-bin/appn
```

如果我们想在此基础上新建一个分支呢，很简单，执行 `git branch a` 就新建了一个名字叫 `a` 的分支，这时候分支 `a` 跟分支 `master` 是一模一样的内容，我们再输入 `git branch` 查看的当前分支情况：

```
→ test git:(master) git branch
文列a表
* master
```

但是可以看到 `master` 分支前有个 * 号，即虽然新建了一个 `a` 的分支，但是当前所在的分支还是在 `master` 上，如果我们想在 `a` 分支上进行开发，首先要先切换到 `a` 分支上才行，所以下一步要切换分支

```
git checkout a
```

执行这个命令，然后再输入 `git branch` 查看下分支情况：

```
→ test git:(master) git checkout a
Switched to branch 'a'
→ test git:(a) git branch
文列a表
 master
```

可以看到当前我们在的分支已经是 `a` 了，这个时候 A 同学就可以尽情的在他新建的 `a` 分支去进行代码改动了。

那有人就说了，我要先新建再切换，未免有点麻烦，有没有一步到位的，聪明：

```
git checkout -b a
```

这个命令的意思就是新建一个a分支，并且自动切换到a分支。

git merge

A同学在a分支代码写的不亦乐乎，终于他的功能完工了，并且测试也都ok了，准备要上线了，这个时候就需要把他的代码合并到主分支master上来，然后发布。git merge 就是合并分支用到的命令，针对这个情况，需要先做两步，第一步是切换到 master 分支，如果你已经在了就不用切换了，第二步执行 git merge a ，意思就是把a分支的代码合并过来，不出意外，这个时候a分支的代码就顺利合并到 master 分支来了。为什么说不出意外呢？因为这个时候可能会有冲突而合并失败，留个包袱，这个到后面进阶的时候再讲。

git branch -d

有新建分支，那肯定有删除分支，假如这个分支新建错了，或者a分支的代码已经顺利合并到 master 分支来了，那么a分支没用了，需要删除，这个时候执行 git branch -d a 就可以把a分支删除了。

git branch -D

有些时候可能会删除失败，比如如果a分支的代码还没有合并到master，你执行 git branch -d a 是删除不了的，它会智能的提示你a分支还有未合并的代码，但是如果你非要删除，那就执行 git branch -D a 就可以强制删除a分支。

git tag

我们在客户端开发的时候经常有版本的概念，比如v1.0、v1.1之类的，不同的版本肯定对应不同的代码，所以我一般要给我们的代码加上标签，这样假设v1.1版本出了一个新bug，但是又不晓得v1.0是不是有这个bug，有了标签就可以顺利切换到v1.0的代码，重新打个包测试了。

所以如果想要新建一个标签很简单，比如 git tag v1.0 就代表我在当前代码状态下新建了一个v1.0的标签，输入 git tag 可以查看历史 tag 记录。

```
→ test git:(a) git tag
v1.0 / 新建图文消息
v1.1
```

可以看到我新建了两个标签 v1.0、v1.1。

想要切换到某个tag怎么办？也很简单，执行 git checkout v1.0 ，这样就顺利的切换到 v1.0 tag 的代码状态了。

OK，以上全是一些最基本的Git操作，而且全是在本地环境进行操作的，完全没有涉及到远程仓库，下一章节将以远程 GitHub 仓库为例，讲解下本地如何跟远程仓库一起同步协作，另外今天讲的全是最基础最简单的Git操作，一步步来，后续再继续讲解一下Git的高阶以及一些Git的酷炫操作。

另外，考虑到可能会有人嫌我讲解的太基础太慢，毕竟我是针对小白，所以得一步步来，迫不及待的想要提前自己学习的不妨在我公众号 `AndroidDeveloper` 回复「git」关键字，获取一份我推荐的还不错的 Git 学习资料，不谢，毕竟我这么帅！

版权声明：

本系列内容首发于我的个人微信公众号 **stormzhang**，原创作者 **stormzhang**，个人博客：<http://stormzhang.com>，可以随意转载，但必须保持署名，禁止商用。

向 GitHub 提交代码

之前的这篇文章「从0开始学习 GitHub 系列之「Git速成」」相信大家都已经对 Git 的基本操作熟悉了，但是这篇文章只介绍了对本地 Git 仓库的基本操作，今天我就来介绍下如何跟远程仓库一起协作，教你们向 GitHub 上提交你们的第一行代码！

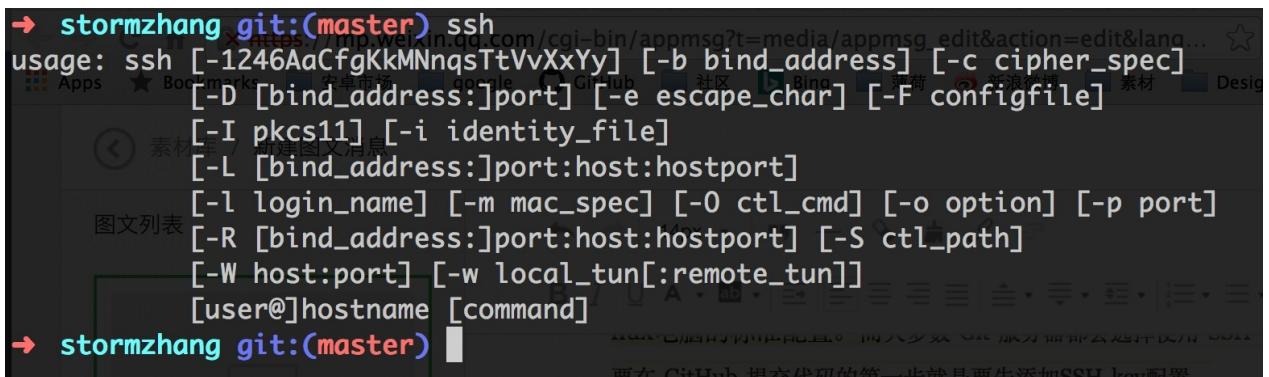
1. SSH

你拥有了一个 GitHub 账号之后，就可以自由的 `clone` 或者下载其他项目，也可以创建自己的项目，但是你没法提交代码。仔细想想也知道，肯定不可能随意就能提交代码的，如果随意可以提交代码，那么 GitHub 上的项目岂不乱了套了，所以提交代码之前一定是需要某种授权的，而 GitHub 上一般都是基于 SSH 授权的。

那么什么是 SSH 呢？简单点说，SSH 是一种网络协议，用于计算机之间的加密登录。目前是每一台 Linux 电脑的标准配置。而大多数 Git 服务器都会选择使用 SSH 公钥来进行授权，所以想要在 GitHub 提交代码的第一步就是要先添加 SSH key 配置。

2. 生成SSH key

Linux 与 Mac 都是默认安装了 SSH，而 Windows 系统安装了 Git Bash 应该也是带了 SSH 的。大家可以在终端（win下在 Git Bash 里）输入 `ssh` 如果出现以下提示证明你本机已经安装 SSH，否则请搜索自行安装下。



```
→ stormzhang git:(master) ssh
usage: ssh [-1246AaCfgKkMNnqsTtVvXxYy] [-b bind_address] [-c cipher_spec]
           [-D [bind_address:]port] [-e escape_char] [-F configfile]
           [-I pkcs11] [-i identity_file]
           [-L [bind_address:]port:host:hostport]
           [-l login_name] [-m mac_spec] [-O ctl_cmd] [-o option] [-p port]
           [-R [bind_address:]port:host:hostport] [-S ctl_path]
           [-W host:port] [-w local_tun[:remote_tun]]
           [user@]hostname [command]
```

紧接着输入 `ssh-keygen -t rsa`，什么意思呢？就是指定 `rsa` 算法生成密钥，接着连续三个回车键（不需要输入密码），然后就会生成两个文件 `id_rsa` 和 `id_rsa.pub`，而 `id_rsa` 是密钥，`id_rsa.pub` 就是公钥。这两文件默认分别在如下目录里生成：

Linux/Mac 系统 在 `~/.ssh` 下，win系统在 `/c/Documents and Settings/username/.ssh` 下，都是隐藏文件，相信你们有办法查看的。

接下来要做的是把 `id_rsa.pub` 的内容添加到 GitHub 上，这样你本地的 `id_rsa` 密钥跟 GitHub 上的 `id_rsa.pub` 公钥进行配对，授权成功才可以提交代码。

3. GitHub 上添加 SSH key

第一步先在 GitHub 上的设置页面，点击最左侧 SSH and GPG keys :

The screenshot shows the GitHub 'Personal settings' page. On the left sidebar, the 'SSH and GPG keys' option is highlighted with a red box. On the right, under the 'SSH keys' heading, there is a list of one key: 'storm@StormMBP.local'. A red box highlights the 'New SSH key' button in the top right corner of this section. Below the SSH keys section is a 'GPG keys' section which is currently empty.

然后点击右上角的 New SSH key 按钮：

SSH keys

This is a list of SSH keys associated with your account. Remove any keys that you do not recognize.

 storm@StormMBP.local	Delete
Fingerprint: a2:cc:81:70:ba:e4:7a:64:42:ab:e4:e0:6f:09:96:b9	
SSH	Added on May 1, 2013 — Last used within the last day
Title	
<input type="text"/>	
Key	
<pre>ssh-rsa NzaC1yc2EAAAQABAAQDLf0hHHFtW7xjLel/rwoYPC3+MHE4HT+ATSRuhKnr19k40RdpICgP4/uxcf5Rj3gnVv XjfNwqMOiOHhFs33tRztGeU/m1DZVV0v39rLpl+0Q8cC8M1EUReLnGSLvTGfMjsUr91YiANLkGSK0dXOrR63Vu7t26IF EY4HlnfkeKdvcUCxq/63lslu/v1svVHn8Ge6U9ZQdeK+O6Htb+CJFekJeYnq9036OxG0DHGHfI7zv7sftYg/yqlIQHtmLCKv MATg2cDOZ4BzPSa/SOQ2b9PUZoLcYjOq1GP4xWKV053mWTNeGnaKgKqVb8KwJ storm@StormMBP.local</pre>	
Add SSH key	
<small>② Check out our guide to generating SSH keys or troubleshoot common SSH Problems.</small>	

需要做的只是在 **Key** 那栏把 `id_rsa.pub` 公钥文件里的内容复制粘贴进去就可以了（上述示例为了安全粘贴的公钥是无效的），**Title** 那栏不需要填写，点击 **Add SSH key** 按钮就ok了。

这里提醒下，怎么查看 `id_rsa.pub` 文件的内容？

Linux/Mac 用户执行以下命令：

```
cd ~/.ssh
cat id_rsa.pub
```

Windows用户，设置显示隐藏文件，可以使用 EditPlus 或者 Sublime 打开复制就行了。

SSH key 添加成功之后，输入 `ssh -T git@github.com` 进行测试，如果出现以下提示证明添加成功了。

```
→ es ~ ssh -T git@github.com msg?t=media/appmsg_edit&action=edit&lang... ☆ 🌐 🌐 🌐
Hi stormzhang! You've successfully authenticated, but GitHub does not
provide shell access.
```

4. Push & Pull

在提交代码之前我们先要了解两个命令，也是上次的文章没有介绍的，因为这两个命令需要跟远程仓库配合。

Push：直译过来就是「推」的意思，什么意思呢？如果你本地代码有更新了，那么就需要把本地代码推到远程仓库，这样本地仓库跟远程仓库就可以保持同步了。

代码示例：

```
git push origin master
```

意思就是把本地代码推到远程 master 分支。

Pull：直译过来就是「拉」的意思，如果别人提交代码到远程仓库，这个时候你需要把远程仓库的最新代码拉下来，然后保证两端代码的同步。

代码示例：

```
git pull origin master
```

意思就是把远程最新的代码更新到本地。一般我们在 push 之前都会先 pull，这样不容易冲突。

5. 提交代码

添加 SSH key 成功之后，我们就有权限向 GitHub 上我们自己的项目提交代码了，而提交代码有两种方法：

Clone自己的项目 我们以我在 GitHub 上创建的 test 项目为例，执行如下命令：

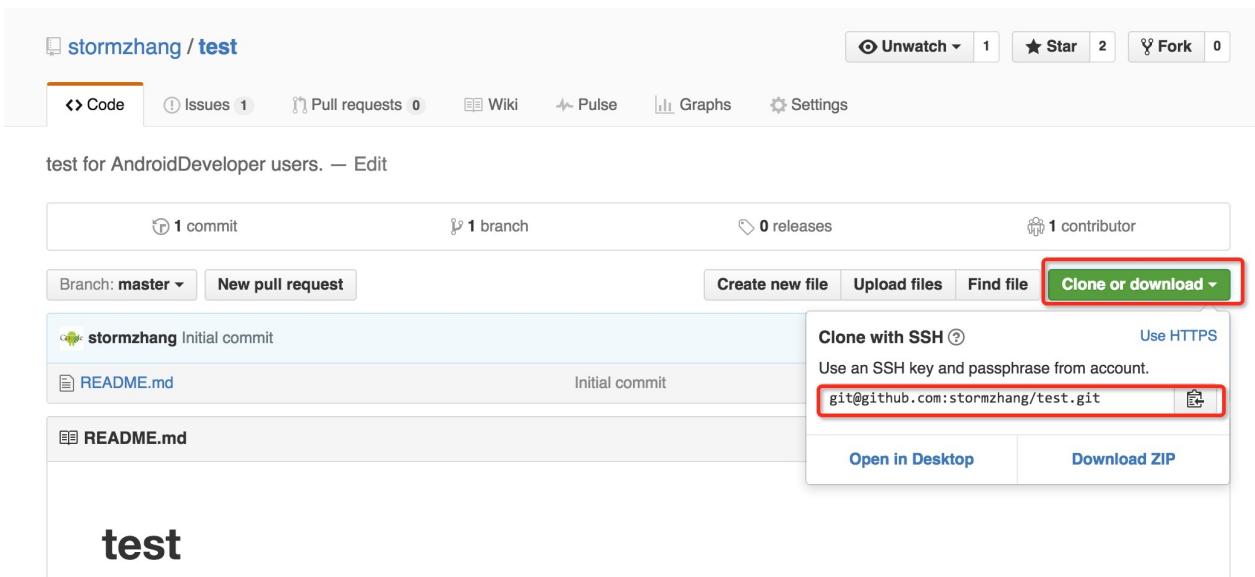
```
git clone git@github.com:stormzhang/test.git
```

这样就把 test 项目 clone 到了本地，你可以把 clone 命令理解为高级点的复制，这个时候该项目本身就已经是一个git 仓库了，不需要执行 **git init** 进行初始化，而且甚至都已经关联好了远程仓库，我们只需要在这个 test 目录下任意修改或者添加文件，然后进行 commit，之后就可以执行：

```
git push origin master
```

进行代码提交，这种是最简单方便的一种方式。

至于怎么获取项目的仓库地址呢？如下图：



关联本地已有项目 如果我们本地已经有一个完整的 git 仓库，并且已经进行了很多次 commit，这个时候第一种方法就不适合了。

假设我们本地有个 test2 的项目，我们需要的是在 GitHub 上建一个 test 的项目，然后把本地 test2 上的所有代码 commit 记录提交到 GitHub 上的 test 项目。

第一步就是在 GitHub 上建一个 test 项目，这个想必大家都会了，就不用多讲了。

第二步把本地 test2 项目与 GitHub 上的 test 项目进行关联，切换到 test2 目录，执行如下命令：

```
git remote add origin git@github.com:stormzhang/test.git
```

什么意思呢？就是添加一个远程仓库，他的地址是 `git@github.com:stormzhang/test.git`，而 `origin` 是给这个项目的远程仓库起的名字，是的，名字你可以随便取，只不过大家公认的只有一个远程仓库时名字就是 `origin`，为什么要给远程仓库取名字？因为我们可能一个项目有多个远程仓库？比如 GitHub 一个，比如公司一个，这样的话提交到不同的远程仓库就需要指定不同的仓库名字了。

查看我们当前项目有哪些远程仓库可以执行如下命令：

```
git remote -v
```

接下来，我们本地的仓库就可以向远程仓库进行代码提交了：

```
git push origin master
```

就是默认向 GitHub 上的 test 目录提交了代码，而这个代码是在 `master` 分支。当然你可以提交到指定的分支，这个之后的文章再详细讲解。

对了，友情提醒，在提交代码之前先要设置下自己的用户名与邮箱，这些信息会出现在所有的 commit 记录里，执行以下代码就可以设置：

```
git config --global user.name "stormzhang"  
git config --global user.email "stormzhang.dev@gmail.com"
```

6. 总结

通过本文的介绍，终于大家可以成功的向 GitHub 提交代码了，但是相信大家还有很多疑问，比如关于分支的理解与使用，比如 git 的其他一些有用的配置，比如怎么向一些开源项目贡献代码，发起 Pull Request 等，之后的系列文章会逐一进行介绍，敬请期待。

版权声明：

本系列内容首发于我的个人微信公众号 **stormzhang**，原创作者 **stormzhang**，个人博客：<http://stormzhang.com>，可以随意转载，但必须保持署名，禁止商用。

Git 进阶

关于 Git 相信大家看了之前一系列的文章已经初步会使用了，但是关于 Git 还有很多知识与技巧是你不知道的，今天就来给大家介绍下一些 Git 进阶的知识。

1. 用户名和邮箱

我们知道我们进行的每一次 commit 都会产生一条 log，这条 log 标记了提交人的姓名与邮箱，以便其他人方便的查看与联系提交人，所以我们在进行提交代码的第一步就是要设置自己的用户名与邮箱。执行以下代码：

```
git config --global user.name "stormzhang"  
git config --global user.email "stormzhang.dev@gmail.com"
```

以上进行了全局配置，当然有些时候我们的某一个项目想要用特定的邮箱，这个时候只需切换到你的项目，以上代码把 **--global** 参数去除，再重新执行一遍就ok了。

PS：我们在 GitHub 的每次提交理论上都会在主页的下面产生一条绿色小方块的记录，如果你确认你提交了，但是没有绿色方块显示，那肯定是你提交代码配置的邮箱跟你 GitHub 上的邮箱不一致，GitHub 上的邮箱可以到 **Setting -> Emails** 里查看。

2. alias

我们知道我们执行的一些 Git 命令其实操作很频繁的类似有：

```
git commit  
git checkout  
git branch  
git status  
...
```

这些操作非常频繁，每次都要输入完全是不是有点麻烦，有没有一种简单的缩写输入呢？比如我对应的直接输入以下：

```
git c
git co
git br
git s
...
```

是不是很简单快捷啊？这个时候就用到了alias配置了，翻译过来就是别名的意思，输入以下命令就可以直接满足了以上的需求。

```
git config --global alias.co checkout # 别名
git config --global alias.ci commit
git config --global alias.st status
git config --global alias.br branch
```

当然以上别名不是固定的，你完全可以根据自己的习惯去定制，除此之外还可以设置组合，比如：

```
git config --global alias.psm 'push origin master'
git config --global alias.plm 'pull origin master'
```

之后经常用到的**git push origin master** 和 **git pull origin master** 直接就用 **git psm** 和 **git plm** 代替了，是不是很方便？

另外这里给大家推荐一个很强大的 alias 命令，我们知道我们输入 **git log** 查看日志的时候是类似这样的：

```
commit 0b5e8caecc2c6ff8426079c005f3452288073463
Author: ttdevs <...>
Date:   Wed May 11 13:38:35 2016 +0800
        # 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 log # 只显示提交记录
commit 63ff87d6380135258964ddf689e7b23f438e576b
Merge: 6a5ed4c 0fd7789
Author: loody <...>
Date:   Thu May 19 11:04:11 2016 +0800
        git log
        Merge branch 'daily_build' into feature/ping++ # 将文件每次提交记录
        git log -p <file> # 查看每次详细修改内容的diff
commit 0fd7789e824f2ca8e67ae4f2f5a2ec81b0a15973
Merge: 48bbea3 9d10a3b
Author: loody <...>
Date:   Thu May 19 11:03:45 2016 +0800
        Mac上可以使用tig代替diff和log, brew install tig
        Merge branch 'daily_build' of git.boohee.cn:android/one into daily_build
        Git 本地分支管理
```

告诉大家一个比较屌的命令，输入`git log --graph --pretty=format:'%Cred%h%Creset - %C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset' --abbrev-commit --date=relative` 然后日志这样了：

```
* 0b5e8ca - (HEAD, feature/debugger) show cached ip (4 hours ago) <ttdevs>
* 63ff87d - (origin/s1, origin/feature/ping++, feature/ping++) Merge branch 'do
weeks ago) <loody>
| \
| * 0fd7789 - Merge branch 'daily_build' of git://github.com/loody/daily-build into dai
| | \
| | * 9d10a3b - 修改饮食工具 (4 weeks ago) <wanglinglong>
* | | 6a5ed4c - Merge branch 'daily_build' into feature/ping++ (4 weeks ago) <1
| \ \ \
| | | 48bbea3 - Merge branch 'daily_build' of git://github.com/loody/daily-build into do
| | | \
| | | * 2441d30 - 工具饮食改进 (4 weeks ago) <wanglinglong>
| | | * c7b8c38 - 修改评测 (4 weeks ago) <wanglinglong>
| | | * 6f4c304 - 评测适配小屏幕手机&优化 (4 weeks ago) <wanglinglong>
| | | * 8df50ef - 优化用户评测 (4 weeks ago) <wanglinglong>
| | | * 6b96942 - 处理BaseJsonRequest中判断是不是food逻辑错误 (4 weeks ago) <ttdevs>
| | | * 4a9649f - 用户在开始时设置的目标时间已到不需要再评测 (4 weeks ago) <wanglin
| | | * 6297df5 - 个人资料界面调整 (4 weeks ago) <wanglinglong>
| | | * f493984 - 刻度尺精度 (4 weeks ago) <wanglinglong>
| | | * c1a5489 - 修改评测相关问题 (5 weeks ago) <wanglinglong>
| | | * ca5320d - Merge branch 'daily_build' of git://github.com/loody/daily-build into do
ong>
| | | \
| | | * 34c17fe - (origin/feature/init) 完善评测 (5 weeks ago) <wanglinglong>
| | | * 1237856 - 修改用户评测 (5 weeks ago) <wanglinglong>前分支的分支
| * | | 6bd2e71 - Merge branch 'master' into daily_build (5 weeks ago) <loody>
```

是不是比较清晰，整个分支的走向也很明确，但是每次都要输这么一大串是不是也很烦？这时候你就该想到 alias 啊：

```
git config --global alias.lg "log --graph --pretty=format:'%Cred%h%Creset - %C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset' --abbrev-commit --date=relative"
```

这样以后直接输入 `git lg` 就行了。

3. 其他配置

当然还有一些其他有用的配置，默认情况下 git 用的编辑器是 vi，如果不喜欢可以改成其他编辑器，比如我习惯 vim。

```
git config --global core.editor "vim" # 设置Editor使用vim
```

你们如果喜欢其他编辑器可自行搜索配置，前提是本机有安装。

有些人纳闷我的终端怎么有各种颜色，自己却不是这样的，那是因为你们没有开启给 Git 着色，输入如下命令即可：

```
git config --global color.ui true
```

还有些其他的配置如：

```
git config --global core.quotepath false # 设置显示中文文件名
```

以上基本所有的配置就差不多了，默认这些配置都在 `~/.gitconfig` 文件下的，你可以找到这个文件查看自己的配置，也可以输入 `git config -l` 命令查看。

4. diff

`diff` 命令算是很常用的，使用场景是我们经常在做代码改动，但是有的时候2天前的代码了，做了哪些改动都忘记了，在提交之前需要确认下，这个时候就可以用 `diff` 来查看你到底做了哪些改动，举个例子，比如我有一个 `a.md` 的文件，我现在做了一些改动，然后输入 `git diff` 就会看到如下：

```
diff --git a/a.md b/a.md
index ea8f022..d2a5109 100644
--- a/a.md
+++ b/a.md
@@ -1 +1,3 @@
-aaaaaaaaaa
+bbbbbbbbbb
+ccccccccc
+ddddddddd
```

红色的部分前面有个 `-` 代表我删除的，绿色的部分前面有个 `+` 代表我增加的，所以从这里你们很一目了然的知道我到底对这个文件做了哪些改动。

值得一提的是直接输入 `git diff` 只能比较当前文件和暂存区文件差异，什么是暂存区？就是你还没有执行 `git add` 的文件。

当然跟暂存区做比较之外，他还可以有其他用法，如比较两次 `commit` 之间的差异，比较两个分支之间的差异，比较暂存区和版本库之间的差异等，具体用法如下：

```
git diff <$id1> <$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
```

5. checkout

我们知道 **checkout** 一般用作切换分支使用，比如切换到 **develop** 分支，可以执行：

```
git checkout develop
```

但是 **checkout** 不只用作切换分支，他可以用来切换**tag**，切换到某次**commit**，如：

```
git checkout v1.0
git checkout ffd9f2dd68f1eb21d36cee50dbdd504e95d9c8f7 # 后面的一长串是commit_id，是每次commit的SHA1值，可以根据 git log 看到。
```

除了有“切换”的意思，**checkout** 还有一个撤销的作用，举个例子，假设我们在一个分支开发一个小功能，刚写完一半，这时候需求变了，而且是大变化，之前写的代码完全用不了了，好在你刚写，甚至都没有 **git add** 进暂存区，这个时候很简单的一个操作就直接把原文件还原：

```
git checkout a.md
```

这里稍微提下，**checkout** 命令只能撤销还没有 **add** 进暂存区的文件。

6. stash

设想一个场景，假设我们正在一个新的分支做新的功能，这个时候突然有一个紧急的bug需要修复，而且修复完之后需要立即发布。当然你说我先把刚写的一点代码进行提交不就行了么？这样理论上当然是ok的，但是这会产品垃圾**commit**，原则上我们每次的**commit**都要有实际的意义，你的代码只是刚写了一半，还没有什么实际的意义是不建议就这样**commit**的，那么有没有一种比较好的办法，可以让我暂时切到别的分支，修复完bug再切回来，而且代码也能保留的呢？

这个时候 **stash** 命令就大有用处了，前提是我们的代码没有进行 **commit**，哪怕你执行了 **add** 也没关系，我们先执行

```
git stash
```

命令，什么意思呢？意思就是把当前分支所有没有 commit 的代码先暂存起来，这个时候你再执行 **git status** 你会发现当前分支很干净，几乎看不到任何改动，你的代码改动也看不见了，但其实是暂存起来了。执行

```
git stash list
```

你会发现此时暂存区已经有了一条记录。

这个时候你可以切换会其他分支，赶紧把bug修复好，然后发布。之后一切都解决了，你再切换回来继续做你之前没做完的功能，但是之前的代码怎么还原呢？

```
git stash apply
```

你会发现你之前的代码全部又回来了，就好像一切都没发生过一样，紧接着你最好需要把暂存区的这次 **stash** 记录删除，执行：

```
git stash drop
```

就把最近一条的 **stash** 记录删除了，是不是很方便？其实还有更方便的，你可以使用：

```
git stash pop
```

来代替 **apply** 命令，**pop** 跟 **apply** 的唯一区别就是 **pop** 不但会帮你把代码还原，还自动帮你把这条 **stash** 记录删除，省的自己再 **drop** 一次了，为了验证你可以紧接着执行 **git stash list** 命令来确认是不是已经没有记录了。

最后还有一个命令介绍下：

```
git stash clear
```

就是清空所有暂存区的记录，**drop** 是只删除一条，当然后面可以跟 **stash_id** 参数来删除指定的某条记录，不跟参数就是删除最近的，而 **clear** 是清空。

7. merge & rebase

我们知道 **merge** 分支是合并的意思，我们在一个 **featureA** 分支开发完了一个功能，这个时候需要合并到主分支 **master** 上去，我们只需要进行如下操作：

```
git checkout master  
git merge featureA
```

其实 `rebase` 命令也是合并的意思，上面的需求我们一样可以如下操作：

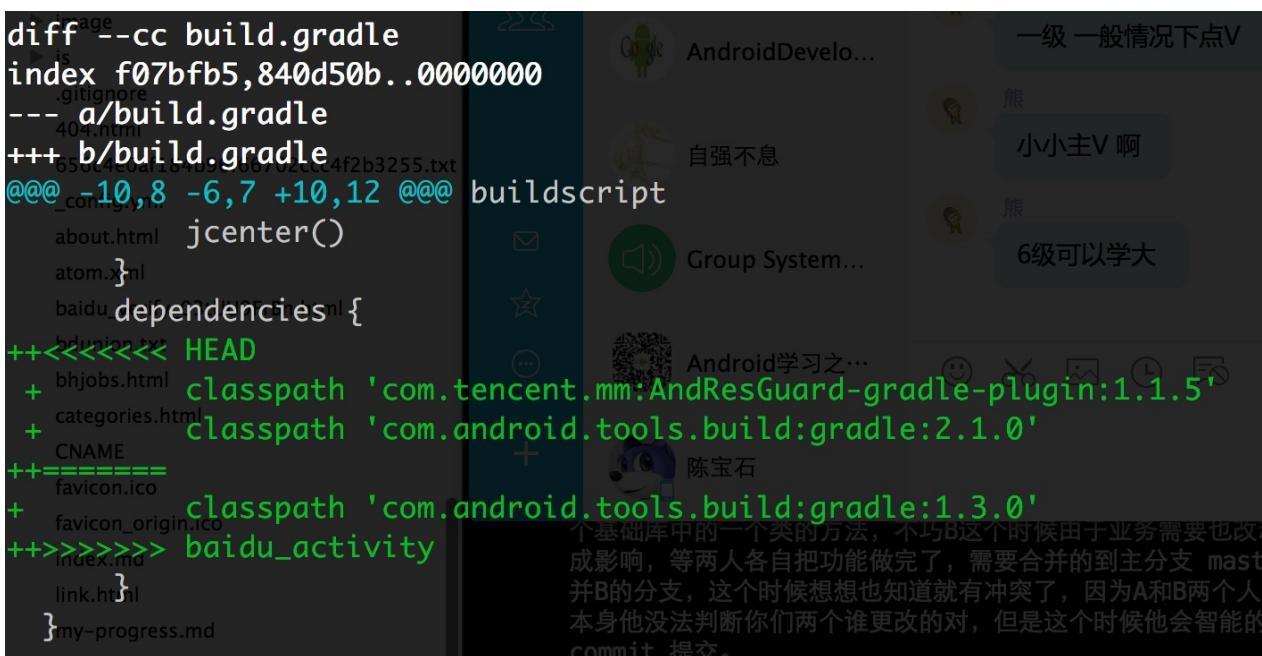
```
git checkout master  
git rebase featureA
```

rebase 跟 **merge** 的区别你们可以理解成有两个书架，你需要把两个书架的书整理到一起去，第一种做法是 **merge**，比较粗鲁暴力，就直接腾出一块地方把另一个书架的书全部放进去，虽然暴力，但是这种做法你可以知道哪些书是来自另一个书架的；第二种做法就是 **rebase**，他会把两个书架的书先进行比较，按照购书的时间来给他重新排序，然后重新放置好，这样做的好处就是合并之后的书架看起来很有逻辑，但是你很难清晰的知道哪些书来自哪个书架的。

只能说各有好处的，不同的团队根据不同的需要以及不同的习惯来选择就好。

8. 解决冲突

假设这样一个场景，A和B两位同学各自开了两个分支来开发不同的功能，大部分情况下都会尽量互不干扰的，但是有一个需求A需要改动一个基础库中的一个类的方法，不巧B这个时候由于业务需要也改动了基础库的这个方法，因为这种情况比较特殊，A和B都认为不会对地方造成影响，等两人各自把功能做完了，需要合并的到主分支 master 的时候，我们假设先合并A的分支，这个时候没问题的，之后再继续合并B的分支，这个时候想想也知道就有冲突了，因为A和B两个人同时更改了同一个地方，Git 本身他没法判断你们两个谁更改的对，但是这个时候他会智能的提示有 **conflicts**，需要手动解决这个冲突之后再重新进行一次 commit 提交。我随便在项目搞了一个冲突做下示例：



以上截图里就是冲突的示例，冲突的地方由 **====** 分出了上下两个部分，上部分一个叫 **HEAD** 的字样代表是我当前所在分支的代码，下半部分是一个叫 **baidu_activity** 分支的代码，可以看到 **HEAD** 对 gradle 插件进行了升级，同时新增了一个插件，所以我们很容易判断哪些代码该保留，哪些代码该删除，我们只需要移除掉那些老旧代码，而且同时也要把那些 **<<< HEAD**、**====** 以及 **>>>>baidu_activity** 这些标记符号也一并删除，最后进行一次 commit 就ok了。

我们在开发的过程中一般都会约定尽量大家写的代码不要彼此影响，以减少出现冲突的可能，但是冲突总归无法避免的，我们需要了解并掌握解决冲突的方法。

版权声明：

本系列内容首发于我的个人微信公众号 **stormzhang**，原创作者 **stormzhang**，个人博客：<http://stormzhang.com>，可以随意转载，但必须保持署名，禁止商用。

团队合作利器：Git 分支管理

Git 相比于 SVN 最强大的一个地方就在于「分支」，Git 的分支操作简直不要太方便，而实际项目开发中团队合作最依赖的莫过于分支了，关于分支前面的系列也提到过，但是本篇会详细讲述什么是分支、分支的具体操作以及实际项目开发中到底是怎么依赖分支来进行团队合作的。

1. 什么是分支？

我知道读者中肯定有些人对分支这个概念比较模糊，其实你们可以这么理解，你们几个人一起去旅行，中间走到一个三岔口，每条路可能有不同的风景，你们约定 3 天之后在某地汇聚，然后各自出发了。而这三条分叉路就可以理解成你们各自的分支，而等你们汇聚的时候相当于把你们的分支进行了合并。

2. 分支的常用操作

通常我们默认都会有一个主分支叫 **master**，下面我们先来看下关于分支的一些基本操作：

- 新建一个叫 **develop** 的分支

```
git branch develop
```

这里稍微提醒下大家，新建分支的命令是基于当前所在分支的基础上进行的，即以上是基于 **master** 分支新建了一个叫做 **develop** 的分支，此时 **develop** 分支跟 **master** 分支的内容完全一样。如果你有 A、B、C 三个分支，三个分支是三位同学的，各分支内容不一样，如果你当前是在 B 分支，如果执行新建分支命令，则新建的分支内容跟 B 分支是一样的，同理如果当前所在是 C 分支，那就是基于 C 分支基础上新建的分支。

- 切换到 **develop** 分支

```
git checkout develop
```

- 如果把以上两步合并，即新建并且自动切换到 **develop** 分支：

```
git checkout -b develop
```

- 把 develop 分支推送到远程仓库

```
git push origin develop
```

- 如果你远程的分支想取名叫 develop2，那执行以下代码：

```
git push origin develop:develop2
```

但是强烈不建议这样，这会导致很混乱，很难管理，所以建议本地分支跟远程分支名要保持一致。

- 查看本地分支列表

```
git branch
```

- 查看远程分支列表

```
git branch -r
```

- 删除本地分支

```
git branch -d develop
```

```
git branch -D develop (强制删除)
```

- 删除远程分支

```
git push origin :develop
```

- 如果远程分支有个 develop，而本地没有，你想把远程的 develop 分支迁到本地：

```
git checkout develop origin/develop
```

- 同样的把远程分支迁到本地顺便切换到该分支：

```
git checkout -b develop origin/develop
```

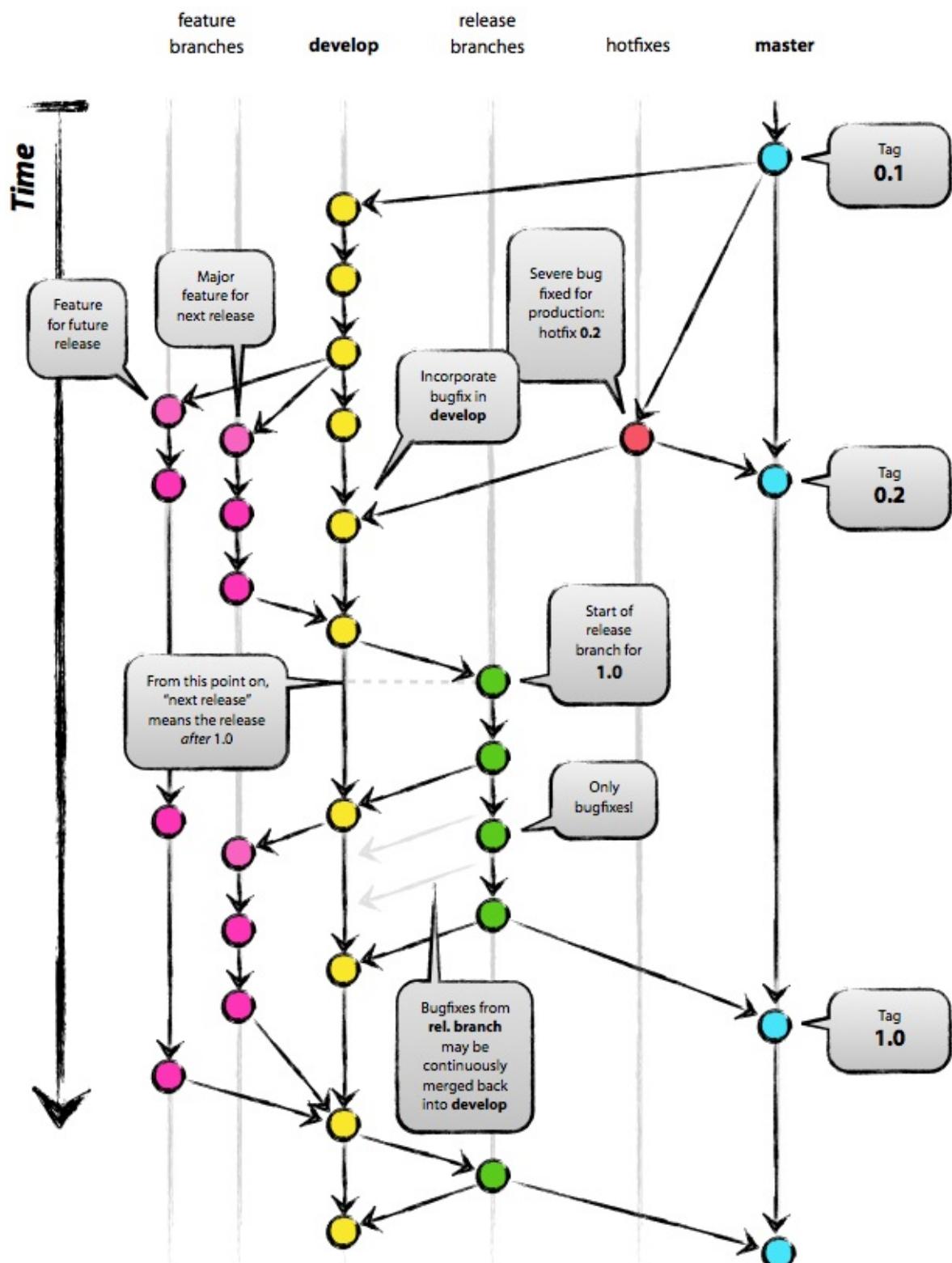
3. 基本的团队协作流程

一般来说，如果你是一个人开发，可能只需要 master、develop 两个分支就 ok 了，平时开发在 develop 分支进行，开发完成之后，发布之前合并到 master 分支，这个流程没啥大问题。

如果你是 3、5 个人，那就不一样了，有人说也没多大问题啊，直接可以新建 A、B、C 三个人的分支啊，每人各自开发各自的分支，然后开发完成之后再逐步合并到 master 分支。然而现实却是，你正在某个分支开发某个功能呢，这时候突然发现线上有一个很严重的 bug，不得不放下手头的工作优先处理 bug，而且很多时候多人协作下如果没有一个规范，很容易产生问题，所以多人协作下的分支管理规范很重要，就跟代码规范一样重要，以下就跟大家推荐一种我们内部在使用的一种分支管理流程 Git Flow。

4. Git Flow

准确的说 Git Flow 是一种比较成熟的分支管理流程，我们先看一张图能清晰的描述他整个的工作流程：



第一次看上面那个图是不是一脸懵逼？跟我当时一样，不急，我来用简单的话给你们解释下。

一般开发来说，大部分情况下都会拥有两个分支 master 和 develop，他们的职责分别是：

- master：永远处在即将发布(production-ready)状态
- develop：最新的开发状态

确切的说 master、develop 分支大部分情况下都会保持一致，只有在上线前的测试阶段 develop 比 master 的代码要多，一旦测试没问题，准备发布了，这时候会将 develop 合并到 master 上。

但是我们发布之后又会进行下一版本的功能开发，开发中间可能又会遇到需要紧急修复 bug，一个功能开发完成之后突然需求变动了等情况，所以 Git Flow 除了以上 master 和 develop 两个主要分支以外，还提出了以下三个辅助分支：

- feature: 开发新功能的分支，基于 develop, 完成后 merge 回 develop
- release: 准备要发布版本的分支，用来修复 bug，基于 develop，完成后 merge 回 develop 和 master
- hotfix: 修复 master 上的问题，等不及 release 版本就必须马上上线. 基于 master, 完成后 merge 回 master 和 develop

什么意思呢？

举个例子，假设我们已经有 master 和 develop 两个分支了，这个时候我们准备做一个功能 A，第一步我们要做的，就是基于 develop 分支新建个分支：

```
git branch feature/A
```

看到了吧，其实就是一个规范，规定了所有开发的功能分支都以 feature 为前缀。

但是这个时候做着做着发现线上有一个紧急的 bug 需要修复，那赶紧停下手头的工作，立刻切换到 master 分支，然后在此基础上新建一个分支：

```
git branch hotfix/B
```

代表新建了一个紧急修复分支，修复完成之后直接合并到 develop 和 master，然后发布。

然后再切回我们的 feature/A 分支继续着我们的开发，如果开发完了，那么合并回 develop 分支，然后在 develop 分支属于测试环境，跟后端对接并且测试的差不多了，感觉可以发布到正式环境了，这个时候再新建一个 release 分支：

```
git branch release/1.0
```

这个时候所有的 api、数据等都是正式环境，然后在这个分支上进行最后的测试，发现 bug 直接进行修改，直到测试 ok 达到了发布的标准，最后把该分支合并到 develop 和 master 然后进行发布。

以上就是 Git Flow 的概念与大概流程，看起来很复杂，但是对于人数比较多的团队协作现实开发中确实会遇到这么复杂的情况，是目前很流行的一套分支管理流程，但是有人会问每次都要各种操作，合并来合并去，有点麻烦，哈哈，这点 Git Flow 早就想到了，为此还专门推出了一个 Git Flow 的工具，并且是开源的：

GitHub 开源地址：<https://github.com/nvie/gitflow>

简单点来说，就是这个工具帮我们省下了很多步骤，比如我们当前处于 master 分支，如果想要开发一个新的功能，第一步切换到 develop 分支，第二步新建一个以 feature 开头的分支名，有了 Git Flow 直接如下操作完成了：

```
git flow feature start A
```

这个分支完成之后，需要合并到 develop 分支，然而直接进行如下操作就行：

```
git flow feature finish A
```

如果是 hotfix 或者 release 分支甚至会自动帮你合并到 develop、master 两个分支。

想必大家已经了解了这个工具的具体作用，具体安装与用法我就不多提了，感兴趣的可以看我下我之前写过的一篇博客：

<http://stormzhang.com/git/2014/01/29/git-flow/>

5. 总结

以上就是我分享给你们的关于分支的所有知识，一个人你也许感受不到什么，但是实际工作中大都以团队协作为主，而分支是团队协作必备技能，而 Git Flow 是我推荐给你们的一个很流行的分支管理流程，也是我们薄荷团队内部一直在实践的一套流程，希望对你们有借鉴意义。

版权声明：

本系列内容首发于我的个人微信公众号 **stormzhang**，原创作者 **stormzhang**，个人博客：<http://stormzhang.com>，可以随意转载，但必须保持署名，禁止商用。

GitHub 常见的几种操作

我们都说开源社区最大的魅力是人人多可以参与进去，发挥众人的力量，让一个项目更完善，更强壮。那么肯定有人疑问，我自己目前还没有能力开源一个项目，但是想一起参与到别的开源项目中，该怎么操作呢？那么今天，就来给大家一起介绍下 GitHub 上的一些常见的操作，看完之后你就知道方法了。

我们姑且以 Square 公司开源的 Retrofit 为例来介绍。

打开链接：

<https://github.com/square/retrofit>

然后看到如下的项目主页：

可以看到一个项目可以进行的操作主要就是两部分，第一部分包括 Watch、Star、Fork，这三个操作之前的系列介绍过了，这里就不啰嗦了。

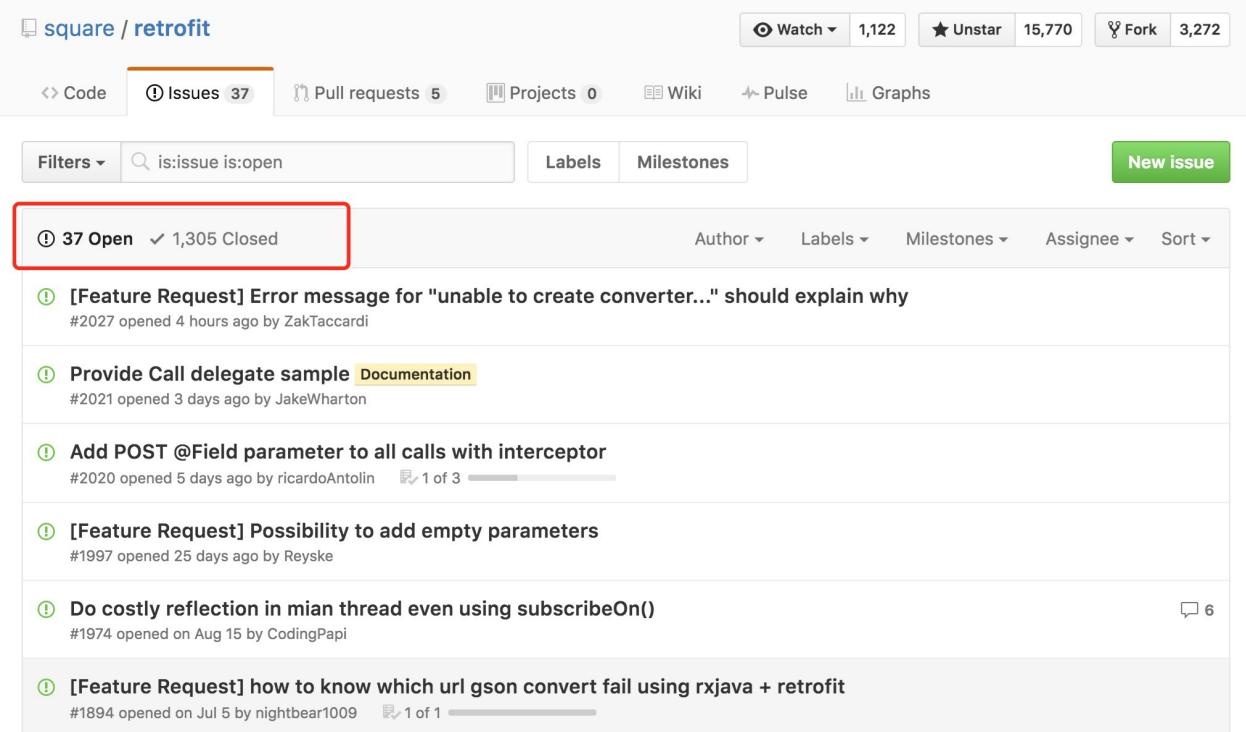
我们着重来介绍第二部分，分别包括 Code、Issues、Pull requests、Projects、Wiki、Pulse、Graphs。接下来我们来一个个解释下。

- Code

这个好理解，就是你项目的代码文件而已，这里说明一下，每个项目通常都会有对该项目的介绍，只需要在项目的根目录里添加一个 README.md 文件就可以，使用 markdown 语法，GitHub 自动会对该文件进行渲染。

- Issues

Issues 代表该项目的一些问题或者 bug，并不是说 Issues 越少越好，Issues 被解决的越多说明项目作者或者组织响应很积极，也说明该开源项目的作者很重视该项目。我们来看下 Retrofit 的 Issues 主页，截至目前 close（解决）了 1305 个 Issue，open（待解决）状态的有 37 个，这解决问题的比例与速度值得每位开源项目的作者学习。



The screenshot shows the GitHub Issues page for the `square / retrofit` repository. At the top, there are buttons for Code, Issues (37), Pull requests (5), Projects (0), Wiki, Pulse, and Graphs. Below these are filters for Labels and Milestones, and a prominent green "New issue" button. The main area displays a list of 37 open issues, each with a title, a link, and a timestamp. The first few issues are:

- ① [Feature Request] Error message for "unable to create converter..." should explain why
#2027 opened 4 hours ago by ZakTaccardi
- ① Provide Call delegate sample Documentation
#2021 opened 3 days ago by JakeWharton
- ① Add POST @Field parameter to all calls with interceptor
#2020 opened 5 days ago by ricardoAntolin 1 of 3
- ① [Feature Request] Possibility to add empty parameters
#1997 opened 25 days ago by Reyske
- ① Do costly reflection in main thread even using subscribeOn()
#1974 opened on Aug 15 by CodingPapi 6
- ① [Feature Request] how to know which url gson convert fail using rxjava + retrofit
#1894 opened on Jul 5 by nightbear1009 1 of 1

同样的，大家在使用一些开源项目有问题的时候都可以提 Issue，可以通过点击右上角的 New Issue 来新建 Issue，需要添加一个标题与描述就可以了，这个操作很简单。

- Pull requests

我们都知道 GitHub 的最大魅力在于人人都可参与，比如别人开源一个项目，我们每个人都可一起参与开发，一起来完善，而这都通过 Pull requests 来完成，简称 PR。这个没法在 Retrofit 演示，下面我就以我自己在 GitHub 上的一个项目 9GAG 来给大家详细演示下怎么给一个项目发起 PR：

提前说明下，你必须确保你可以正常向 GitHub 提交代码，如果不可以的话，请看我之前的系列文章。

第一步登录你的 GitHub 账号，然后找到你想发起 PR 的项目，这里以 9GAG 为例，点击右上角的 Fork 按钮，然后该项目就出现在了你自己账号的 Repository 里。

请注意，这个项目原本是属于 GitHub 账号 `stormzhang` 下的，为了演示，我自己又重新注册了另一个账号叫 `googdev` 单纯为了演示而用。

Fork 之后，在账号 `googdev` 下多了一个 `9GAG` 的项目，截图显示如下：

可以看到 Fork 过来的项目标题底部会显示一行小字：`fork from stormzhang/9GAG`，除此之外，项目代码跟原项目一模一样，对于原项目来说，相当于别人新建了一个分支而已。

第二步，把该项目 clone 到本地，然后修改的 bug 也好，想要新增的功能也好，总之把自己做的代码改动开发完，保存好。为了方便演示，我这里只在原项目的 `README.md` 文件添加了一行文字：`Fork from stormzhang !`

接着，把自己做的代码改动 push 到你自己的 GitHub 上去。

相信看过我前面教程的同学这一步应该都会，不会的可以滚回去看前面的教程了。

第三步，点击你 Fork 过来的项目主页的 Pull requests 页面，

The screenshot shows the GitHub repository 'googdev / 9GAG'. At the top, there are buttons for 'Unwatch', 'Star 0', 'Fork 1,030', and a 'New pull request' button. Below the header, there's a search bar with the query 'is:pr is:open' and tabs for 'Labels' and 'Milestones'. A red box highlights the 'Pull requests 0' button. Another red box highlights the 'New pull request' button.

点击 New pull request 按钮紧接着到如下页面：

Comparing changes

Choose two branches to see what's changed or to start a new pull request. If you need to, you can also compare across forks.

The screenshot shows the 'Comparing changes' page for the 'README.md' file. It displays a comparison between the 'base fork: stormzhang/9GAG' and 'head fork: googdev/9GAG' with 'base: master' and 'compare: master'. A green checkmark indicates that the branches are 'Able to merge'. A red box highlights the 'Create pull request' button. Below the comparison summary, it shows '1 commit', '1 file changed', '0 commit comments', and '1 contributor'. The commit details show a merge from 'Commits on Sep 25, 2016' in the 'googdev' branch. The diff view shows the addition of a line: '+Fork from stormzhang !'. A red box highlights this line. The bottom right has 'Unified' and 'Split' view options.

这个页面自动会比较该项目与原有项目的不同之处，最顶部声明了是 stormzhang/9GAG 项目的 master 分支与你 fork 过来的 googdev/9GAG 项目 master 分支所做的比较。

然后最顶部可以方便直观的看到到底代码中做了哪些改动，你们也看到我就是加了一句 Fork from stormzhang !

同样的我写好标题和描述，然后我们点击中间的 Create pull request 按钮，至此我们就成功给该项目提交了一个 PR。

然后就等着项目原作者 review 你的代码，并且决定会不会接受你的 PR，如果接受，那么恭喜你，你已经是该项目的贡献者之一了。

- Projects

这个是最新 GitHub 改版新增的一个项目，这个项目就是方便你把一些 Issues、Pull requests 进行分类，反正我觉得该功能很鸡肋，起码到目前为止基本没人用该功能，你们了解下就好。

- Wiki

一般来说，我们项目的主页有 README.me 基本就够了，但是有些时候我们项目的一些用法很复杂，需要有详细的使用说明文档给开源项目的使用者，这个时候就用到了 Wiki。

square / retrofit

Watch 1,122 Unstar 15,799 Fork 3,280

Code Issues 36 Pull requests 5 Projects 0 Wiki Pulse Graphs

Home

Philip Jahoda edited this page on Dec 19, 2015 · 2 revisions

Welcome to the Retrofit wiki page!

Pages 4

- Home
- Call Adapters
- Converters
- Retrofit Tutorials

Clone this wiki locally

<https://github.com/square/r>

Clone in Desktop

使用起来也很简单，直接 New Page，然后使用 markdown 语法即可进行编写。

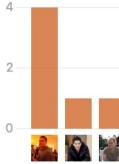
- Pulse

Pulse 可以理解成该项目的活跃汇总。包括近期该仓库创建了多少个 Pull Request 或 Issue，有多少人参与了这个仓库的开发等，都可以在这里一目了然。

根据这个页面，用户可以判断该项目受关注程度以及项目作者是否还在积极参与解决这些问题等。



Excluding merges, 3 authors have pushed 6 commits to master and 6 commits to all branches. On master, 36 files have changed and there have been 453 additions and 991 deletions.

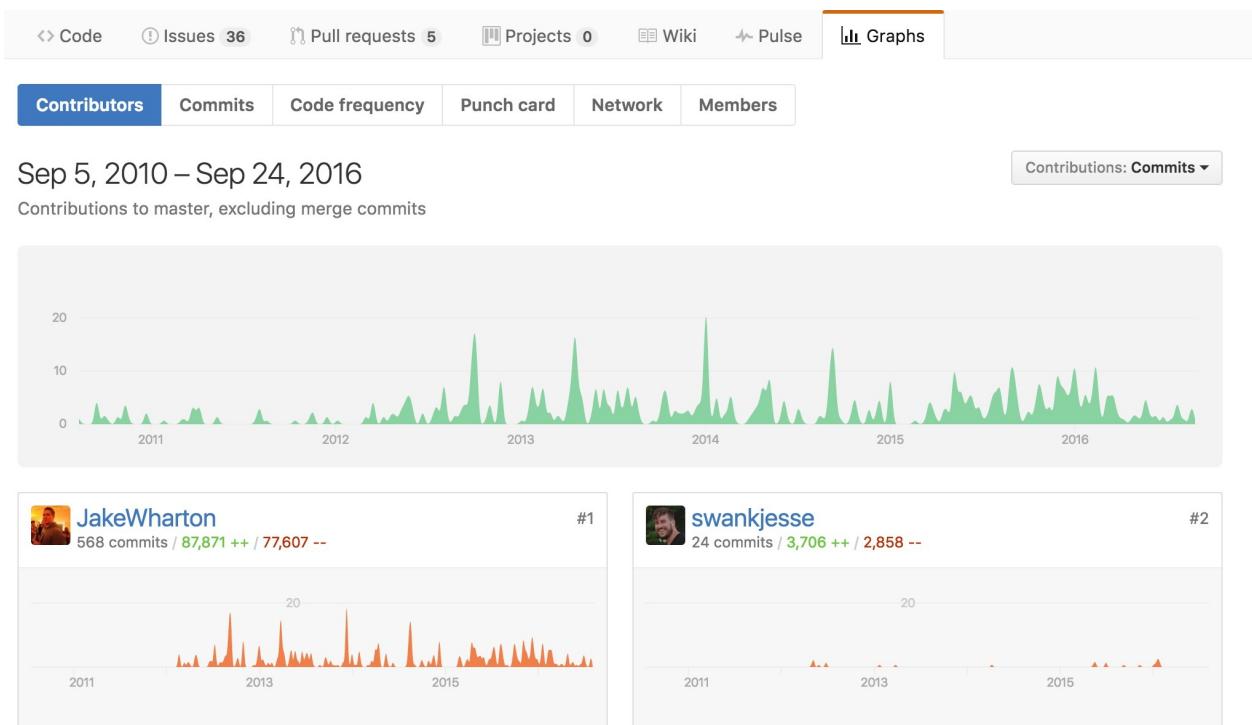


5 Pull requests merged by 3 people

- Merged #2023 Close ResponseBody on 204/205 to avoid connection leak 5 days ago
- Merged #2016 Update RxJava adapter to 1.2.0. 7 days ago
- Merged #2017 Promote adapter response body type parameter to class level. 8 days ago
- Merged #2018 Add method for returning a Moshi converter that serializes nulls. 8 days ago

• Graphs

Graphs 是以图表的形式来展示该项目的一个整体情况。比如项目的全部贡献人，比如 **commits** 的国度分析，比如某天代码提交的频繁程度等。



• Settings

如果一个项目是自己的，那么你会发现会多一个菜单 **Settings**，这里包括了你对整个项目的设置信息，比如对项目重命名，比如删除该项目，比如关闭项目的 **Wiki** 和 **Issues** 功能等，不过大部分情况下我们都不需要对这些设置做更改。感兴趣的，可以自行看下这里的设置有哪些功能。

The screenshot shows the GitHub repository settings for '9GAG'. The top navigation bar includes links for Code, Issues (6), Pull requests (3), Projects (1), Wiki, Pulse, Graphs, and Settings. The 'Settings' tab is selected. On the left, a sidebar titled 'Options' lists Collaborators, Branches, Webhooks, Integrations & services, and Deploy keys. The main area is titled 'Settings' and contains sections for 'Repository name' (set to '9GAG') and 'Rename'. Below this is a 'Features' section with two checked options: 'Wikis' (described as a simple way for others to contribute content) and 'Issues' (described as lightweight issue tracking integrated with the repository). There is also an unchecked option 'Restrict editing to collaborators only'.

以上就包含了一个 GitHub 项目的所有操作，相信大家看完之后对 GitHub 上一些常用的操作都熟悉了，从现在开始，请一起参与到开源社区中来吧，开源社区需要我们每个人都贡献一份力，这样开源社区才能越来越强大，也才能对更多的人有帮助！

版权声明：

本系列内容首发于我的个人微信公众号 **stormzhang**，原创作者 **stormzhang**，个人博客：<http://stormzhang.com>，可以随意转载，但必须保持署名，禁止商用。

发现好用的开源项目

之前发过一系列有关 GitHub 的文章，有同学问了，GitHub 我大概了解了，Git 也差不多会使用了，但是还是搞不清 GitHub 如何帮助我的工作，怎么提升我的工作效率？

问到点子上了，GitHub 其中一个最重要的作用就是发现全世界最优秀的开源项目，你没事的时候刷刷微博、知乎，人家没事的时候刷刷 GitHub，看看最近有哪些流行的项目，久而久之，这差距就越来越大，那么如何发现优秀的开源项目呢？这篇文章我就来给大家介绍下。

1. 关注一些活跃的大牛

GitHub 主页有一个类似微博的时间线功能，所有你关注的人的动作，比如 star、fork 了某个项目都会出现在你的时间线上，这种方式适合我这种比较懒的人，不用主动去找项目，而这种基本是我每天获取信息的一个很重要的方式。不知道怎么关注这些人？那么很简单，关注我 **stormzhang**，以及我 GitHub 上关注的一些大牛，基本就差不多了。

stormzhang ▾

 asLody starred [p0sixspwn/p0sixspwn](#) 39 minutes ago

 baoyongzhang starred [a-voyager/AutoInstaller](#) 2 hours ago

 Skykai521 starred [reark/reark](#) 2 hours ago

 elezhangwen starred [stormzhang/free-programming-books](#) 2 hours ago

 green robot starred [markusl/GoogleTestRunner](#) 15 days ago

 chrisjenx starred [xetorthio/jedis](#) 15 days ago

 johnkil starred [reqquery/sqlite-android](#) 15 days ago

 Skykai521 starred [ldtk/SmallChart](#) 15 days ago

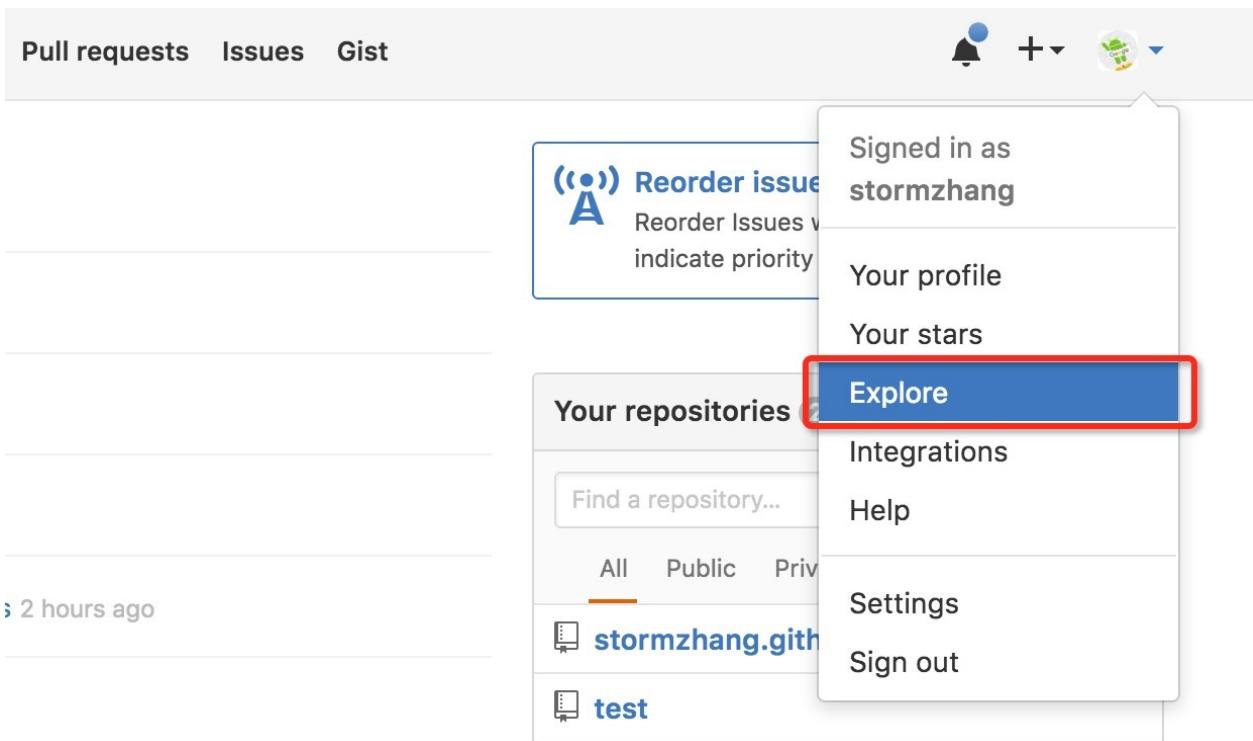
 ryancheung created repository [ryancheung/wowhub](#) 15 days ago

 ryancheung starred [AndorChen/rbenv-china-mirror](#) 15 days ago

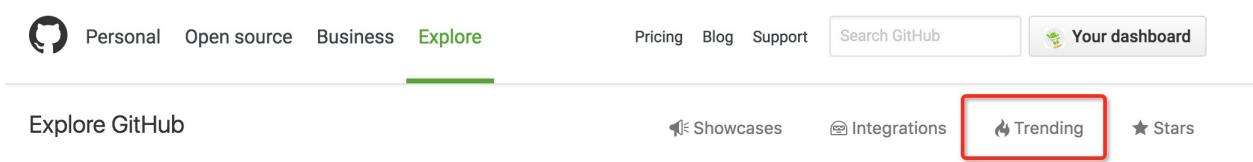
 mariotaku starred [kevin0571/STPopup](#) 15 days ago

2. Trending

点击下图的 Explore 菜单到“发现”页面



紧接着点击 Trending 按钮



Project showcases

Browse interesting repositories, solving all types of interesting problems.

这个 Trending 页面是干嘛的呢？直译过来就是趋势的意思，就是说这个页面你可以看到最近一些热门的开源项目，这个页面可以算是很多人主动获取一些开源项目最好的途径，可以选择「当天热门」、「一周之内热门」和「一月之内热门」来查看，并且还可以分语言类来查看，比如你想查看最近热门的 Android 项目，那么右边就可以选择 Java 语言。

Trending in open source

See what the GitHub community is most excited about this week.

A screenshot of the GitHub 'Trending in open source' page. It shows two repository cards: 'FallibleInc/security-guide-for-developers' and 'facebookincubator/create-react-app'. Below the cards are filtering options. A dropdown menu for 'Trending: this week' is open, with 'this week' selected (highlighted with a red box). To the right, there's a sidebar for 'All languages' with a list of popular languages: Unknown languages, C, CSS, HTML, Java, JavaScript, Python, and Ruby. A 'Languages' dropdown is also present. At the bottom, there's a 'Star' button.

这样页面推荐大家每隔几天就去看下，主动发掘一些优秀的开源项目。

3. Search

除了 Trending ，还有一种最主动的获取开源项目的方式，那就是 GitHub 的 Search 功能。

举个例子，你是做 Android 的，接触 GitHub 没多久，那么第一件事就应该输入 android 关键字进行搜索，然后右上角选择按照 star 来排序，结果如下图：

The screenshot shows the GitHub search interface with the query 'android' entered in the search bar. A red box highlights the 'Sort: Most stars' dropdown menu. The search results page displays 373,145 repository results. The first result is 'Trinea/android-open-project', which is a collection of Android open-source projects. Below it is 'Prinzhorn/skrollr', a parallax scrolling library. Further down is 'wasabeef/awesome-android-ui', a curated list of awesome Android UI/UX libraries. At the bottom is 'square/retrofit', a type-safe HTTP client. Each result includes a thumbnail, the repository name, a brief description, the programming language, the number of stars, and the number of forks.

Repositories	Code	Issues	Users
373,146	114,026,983	2,269,602	4,964

Languages	
Java	231,785
C	18,136
C++	7,858
JavaScript	6,914
Shell	5,184
Makefile	3,840
C#	3,057
Python	2,894
HTML	2,737
CSS	1,427

如果你是学习 iOS 的，那么不妨同样的方法输入 iOS 关键字看看结果：

Search Search

We've found 135,181 repository results Sort: Most stars ▾

Repositories	135,181
Code	25,987,056
Issues	475,924
Users	2,811

Languages

Objective-C	60,515
Swift	29,764
JavaScript	4,478
C	2,068
Java	1,759
C#	1,664
Ruby	1,559
C++	1,501
Python	1,082
HTML	1,042

AFNetworking/AFNetworking Objective-C ★ 26,401 ⚡ 8,376
A delightful networking framework for iOS, OS X, watchOS, and tvOS.
Updated 9 days ago

facebook/pop Objective-C++ ★ 16,019 ⚡ 2,561
An extensible iOS and OS X animation library, useful for physics-based interactions.
Updated on May 13

Prinzhorn/skrollr HTML ★ 15,927 ⚡ 3,235
Stand-alone parallax scrolling library for mobile (Android + iOS) and desktop. No jQuery. Just plain JavaScript (and some love).
Updated on May 13

vsouza/awesome-ios Swift ★ 13,810 ⚡ 2,372
A curated list of awesome iOS ecosystem, including Objective-C and Swift Projects
Updated 5 hours ago

BradLarson/GPUImage Objective-C ★ 13,332 ⚡ 3,312
An open source iOS framework for GPU-based image and video processing
Updated 24 days ago

可以看到按照 star 数，排名靠前基本是一些比较火的项目，一定是很实用，才会这么火。值得一提的是左侧依然可以选择语言进行过滤。

而对于实际项目中用到一些库，基本上都会第一时间去 GitHub 搜索下有没有类似的库，比如项目中想采用一个网络库，那么不妨输入 android http 关键字进行搜索，因为我只想找到关于 Android 的项目，所以搜索的时候都会加上 android 关键字，按照 star 数进行排序，我们来看下结果：

Search

Repositories	6,705
Code	71,686,340
Issues	896,804
Users	

We've found 6,705 repository results Sort: Most stars ▾

square/retrofit Java ★ 14,286 ⚡ 2,933
 Type-safe **HTTP** client for **Android** and Java by Square, Inc.
 Updated 3 days ago

Languages

Language	Count
Java	4,002
C	512
C++	209
JavaScript	125
C#	99
Shell	81
Python	49
PHP	45
Makefile	32
HTML	25

square/okhttp Java ★ 13,091 ⚡ 3,411
 An **HTTP + HTTP/2** client for **Android** and Java applications.
 Updated 3 hours ago

loopj/android-async-http Java ★ 9,078 ⚡ 4,228
 An Asynchronous **HTTP** Library for **Android**
 Updated on Jun 27

可以看到 Retrofit、OkHttp、android-async-http 是最流行的网络库，只不过 android-async-http 的作者不维护了，之前很多人问我网络库用哪个比较好？哪怕你对每个网络库都不是很了解，那么单纯的按照这种方式你都该优先选择 Retrofit 或者 OkHttp，而目前绝大部分 Android 开发者确实也都是在用这两个网络库，当然还有部分在用 Volley 的，因为 google 没有选择在 GitHub 开源 volley，所以搜不到 volley 的上榜。

除此之外，GitHub 的 Search 还有一些小技巧，比如你想搜索的结果中 star 数大于1000的，那么可以这样搜索：

android http stars:>1000

当然还有其他小技巧，但是我觉得不是很重要，就不多说了。

有些人如果习惯用 Google 进行搜索，那么想搜索 GitHub 上的结果，不妨前面加 GitHub 关键字就ok了，比如我在 google 里输入 GitHub android http ，每个关键字用空格隔开，然后搜索结果如下：

A screenshot of a Google search results page. The search query is "github android http". The results are filtered by "All" and show approximately 3,800,000 results in 0.41 seconds. The top result is a link to the GitHub repository for "loopj/android-async-http", which is described as an asynchronous, callback-based HTTP client for Android built on top of Apache's HttpComponents. Other results include "square/okhttp", "kevinsawicki/http-request", "afollestad/bridge", and "square/retrofit". Each result includes a link to the GitHub page, a brief description, and some statistics like star count or release information.

可以看到，基本也是我们想要的结果，只不过排序就不是单纯的按照 star 来排序了。

福利大放送

相信以上三种方法够大家遨游在 GitHub 的海洋了，最后给大家献上一些福利，这些项目是 GitHub 上影响力很大，同时又对你们很有用的项目：

- [free-programming-books](#)

这个项目目前 star 数排名 GitHub 第三，总 star 数超过6w，这个项目整理了所有跟编程相关的免费书籍，而且全球多国语言版的都有，中文版的在这里：[free-programming-books-zh](#)，有了这个项目，理论上你可以获取任何编程相关的学习资料，强烈推荐给你们！

- [oh-my-zsh](#)

俗话说，不会用 shell 的程序员不是真正的程序员，所以建议每个程序员都懂点 shell，有用不说，装逼利器啊！而 oh-my-zsh 毫无疑问就是目前最流行，最酷炫的 shell，不多说了，懂得自然懂，不懂的以后你们会懂的！

- [awesome](#)

GitHub 上有各种 awesome 系列，简单来说就是这个系列搜罗整理了 GitHub 上各领域的资源大汇总，比如有 awesome-android, awesome-ios, awesome-java, awesome-python 等等等等，就不截图了，你们自行去感受。

- [github-cheat-sheet](#)

GitHub 的使用有各种技巧，只不过基本的就够我们用了，但是如果你对 GitHub 超级感兴趣，想更多的了解 GitHub 的使用技巧，那么这个项目就刚好是你需要的，每个 GitHub 粉都应该知道这个项目。

- [android-open-project](#)

这个项目是我一个好朋友 Trinea 整理的一个开源项目，基本囊括了所有 GitHub 上的 Android 优秀开源项目，但是缺点就是太多了不适合快速搜索定位，但是身为 Android 开发无论如何你们应该知道这个项目。

- [awesome-android-ui](#)

这个项目跟上面的区别是，这个项目只整理了所有跟 Android UI 相关的优秀开源项目，基本你在实际开发终于到的各种效果上面都几乎能找到类似的项目，简直是开发必备。

- [Android_Data](#)

这个项目是我的邪教群的一位管理员整理的，几乎包括了国内各种学习 Android 的资料，简直太全了，我为这个项目也稍微做了点力，强烈推荐你们收藏起来。

- [AndroidInterview-Q-A](#)

这个就不多说了，之前给大家推荐过的，国内一线互联网公司内部面试题库。

- [LearningNotes](#)

这是一份非常详细的面试资料，涉及 Android、Java、设计模式、算法等等等，你能想到的，你不能想到的基本都包含了，可以说是适应于任何准备面试的 Android 开发者，看完这个之后别说你还不知道怎么面试！

总结

GitHub 上优秀开源项目真的是一大堆，就不一一推荐了，授人以鱼不如授人以渔，请大家自行主动发掘自己需要的开源项目吧，不管是应用在实际项目上，还是对源码的学习，都是提升自己工作效率与技能的很重要的一个渠道，总有一天，你会突然意识到，原来不知不觉你已经走了这么远！