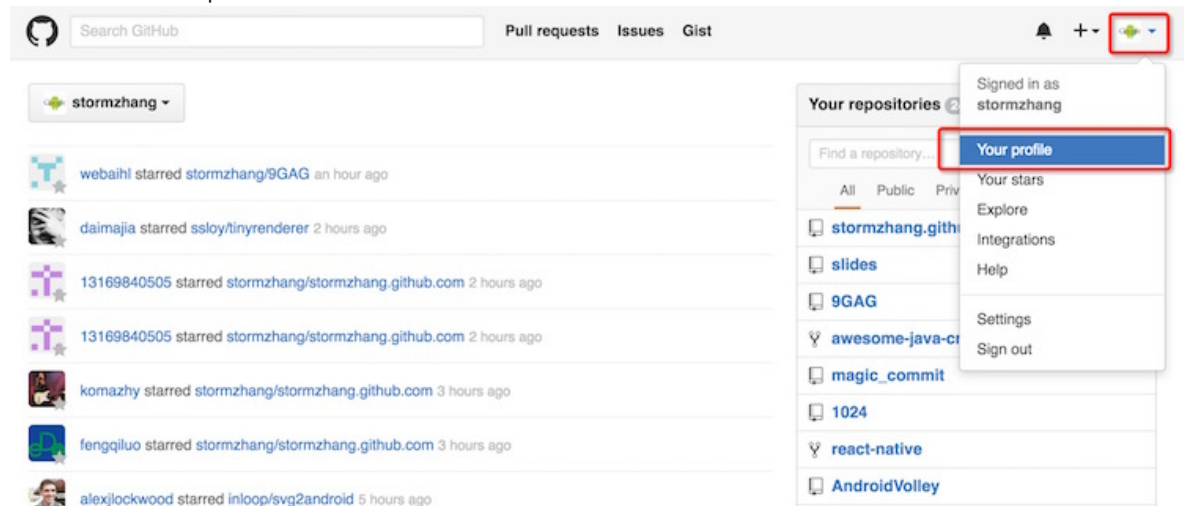


gitHub笔记

GitHub主页介绍

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



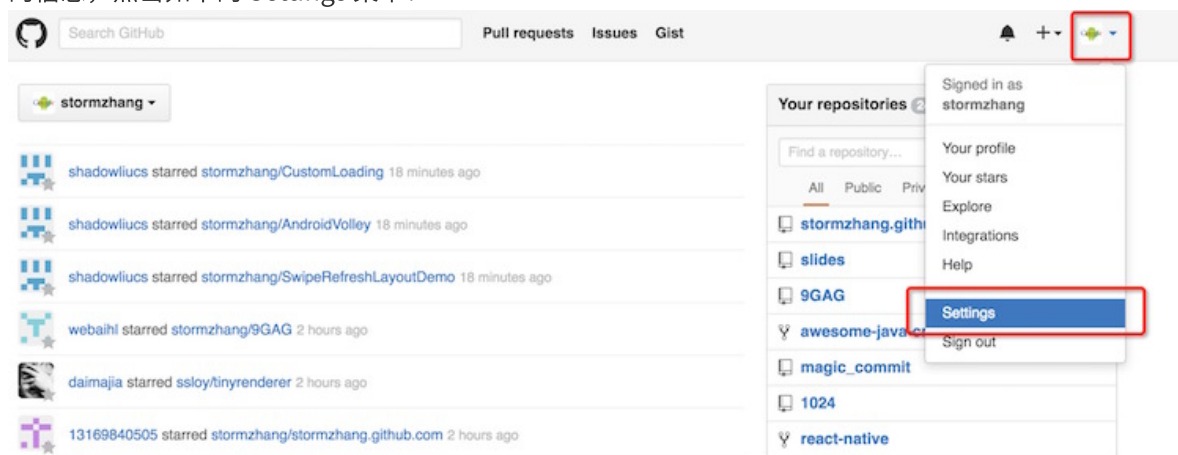
以我的 GitHub 主页为例：



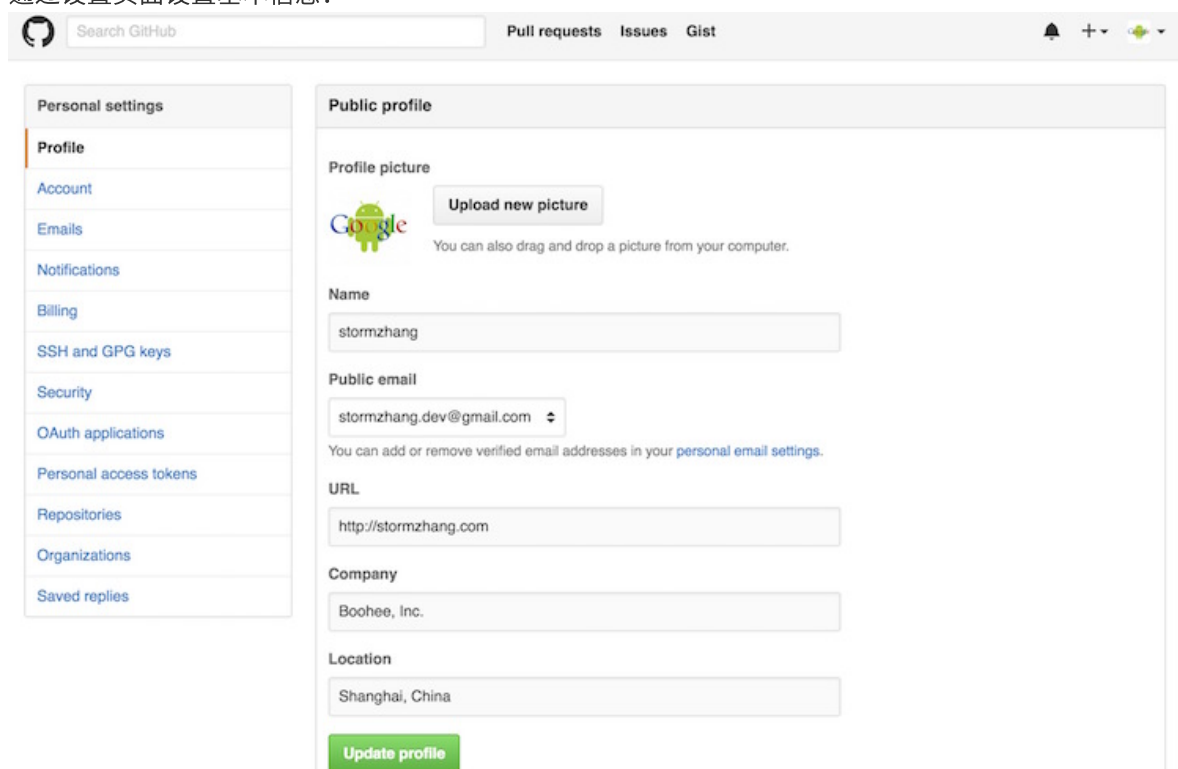
图片标注的已经很详细了，当然新的账号没有这么丰富，因为什么也没做过，但是如果做全了基本上就会看到跟我一样的效果。

设置你的GitHub

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



通过设置页面设置基本信息：



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、猎头就通过这个公开邮箱去联系你，友情提醒：绑定邮箱最好是 Gmail 邮箱，其次是 foxmail、163 的邮箱，不推荐使用 QQ 邮箱。

GitHub基本概念

认识了 GitHub 的基本面貌之后，接下来需要了解一些 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，如果你想在 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，你就会拥有多个 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，这个时候 Gist 就派上用场了！

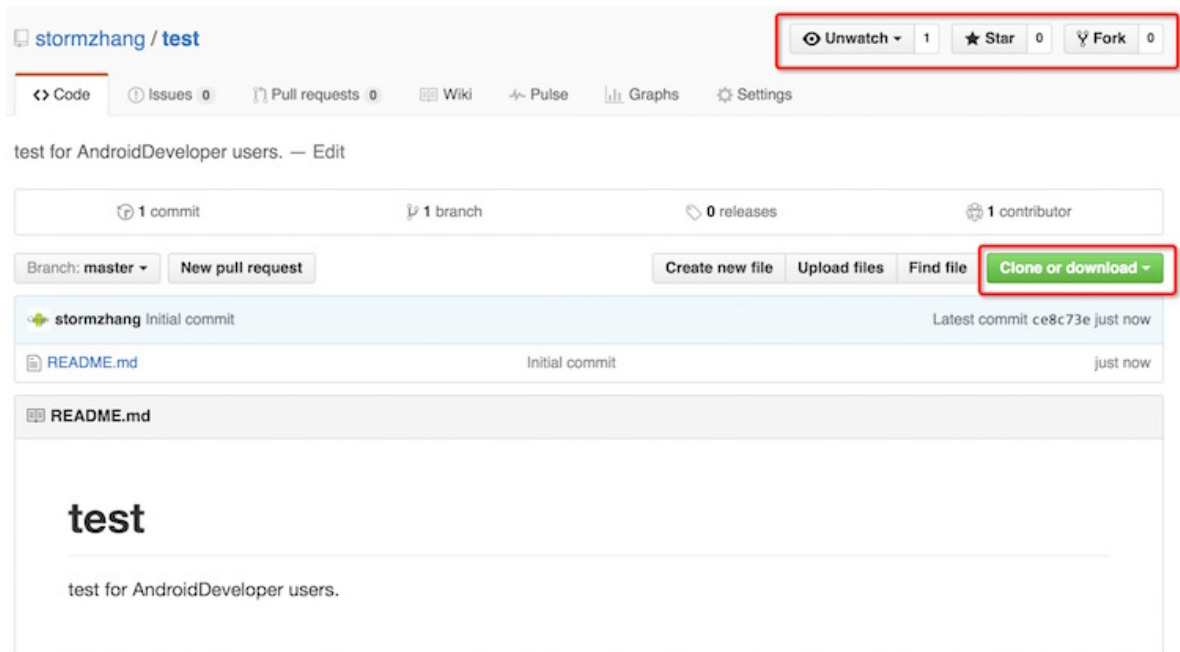
创建自己的项目

点击顶部导航栏的 + 可以快速创建一个项目，如下图：

The screenshot shows the GitHub 'Create a new repository' page. The form includes the following fields and options:

- Owner:** stormzhang
- Repository name:** test (with a green checkmark)
- Description (optional):** test for AndroidDeveloper users.
- Visibility:** Public (selected), Private (unselected)
- Initialize this repository with a README:** Checked
- Add .gitignore:** None
- Add a license:** None
- Create repository button:** Green button at the bottom

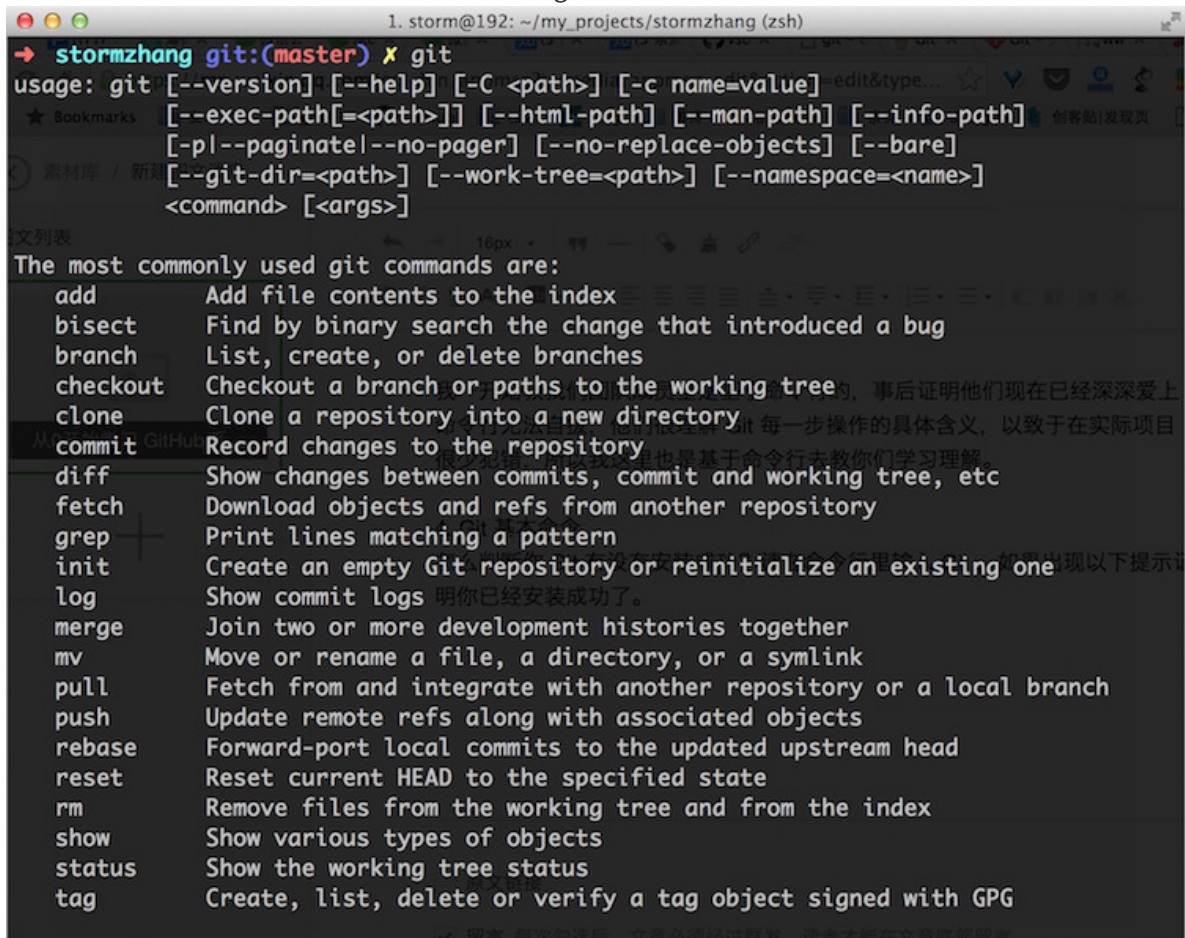
创建一个项目需要填写如上的几部分：项目名、项目描述与简单的介绍，你不付费没法选择私有的，所以接着只能选择 public 的，之后勾选「Initialize this repository with a README」，这样你就拥有了你的第一个 GitHub 项目：



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。

Git 命令列表

怎么判断 Git 有没有安装成功？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。



Git 所有的操作命令开头都要以 git 开头，上面列举了一些 Git 命令，后面有一句英文解释这个命令的意义，都是比较简单的词汇，不妨试着翻译一下，但是如果没有实际操作仍然不好理解，下面就以一个实际的操作来介绍一些常用命令的含义。

Git 常用命令举例

第一步，需要新建一个文件夹，在文件夹里新建一个文件（我是用 Linux 命令新建的，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）

注意：在进行任何 Git 操作之前，都要先切换到 Git 仓库目录，也就是要先切换到项目的文件夹目录下。

这个时候随便操作一个命令，比如 `git status`，可以看到如下提示（不要纠结颜色，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
```

意思是当前目录还不是一个 Git 仓库。

1.git init

这个时候就要用到第一个命令了，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) x
```

可以看到初始化成功了，至此 test 目录已经是一个 git 仓库了。

2.git status

接下来输入 `git status` 命令，会有如下提示：

```
→ test git:(master) x 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 文件没有被跟踪，还没有提交到 git 仓库里，提示可以使用 `git add` 去操作想要提交的文件。

`git status` 这个命令顾名思义就是查看状态，这是一个使用最频繁的命令，建议大家经常输入这个命令，来查看当前 git 仓库的一些状态。

3.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`：

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。

4.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。

5.git log

此时输入 git log 命令，会显示如下内容：

```
commit 7fa619d905fc9bba4938e1961b7a3056892e46b0
Author: stormzhang <zhangqi@boohee.com>
Date: Sun May 29 20:35:08 2016 +0800

    first commit
```

git log 命令可以查看所有产生的 commit 记录，从上图中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'。

6.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，当然也可以将这两步合并成一步，后面再介绍，建议新手先按部就班的一步步来。

7.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协同合作的最大利器。举个例子，A, B俩人在做同一个项目，但是不同模块，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对代码做的所有改动都在各自的分支下，互不影响，最终，各自的模块都完成后，可以把分支再合并起来。

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，可以输入 git branch 查看下当前分支情况：

```
→ test git:(master) git branch
* master
```

如果想在此基础上新建一个分支，只要执行 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，再输入 git branch 查看的当前分支情况：

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

从上图可以看到 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，如果想在 a 分支上进行开发，首先要切换到 a 分支上，执行 git checkout a 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
a
* a
```

可以看到当前所在分支已经是a了，这个时候 A 同学就可以尽情的在新建的a分支去进行代码改动了。

可能有些同学会觉得先新建再切换会有些麻烦，下面就提供给大家一个简便的命令：

```
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。

8.git merge

A同学在a分支下代码写的不亦乐乎，终于他的模块完成了，并且测试也都通过了，准备要上线，这个时候就需要把他的代码合并到主分支master上，然后发布。git merge 就是合并分支用到的命令，针对上述情况，需要完成两个步骤，第一步是切换到 master 分支，如果已经在就不需要切换了，第二步执行 git merge a，意思是把a分支的代码合并到master分支，不出意外，这个时候a分支的代码就顺利合并到 master 分支上了。为什么说不出意外呢？因为这个时候可能会有冲突而造成合并失败，留个包袱，后面进阶的时候再讲。

9.git branch -d

既然有新建分支，那么一定会有删除分支，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了 master 分支上，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。

10.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，执行 git branch -d a 是删除不了的，会提示a分支还有未合并的代码，但是如果一定要删除，那就执行 git branch -D a 就可以强制删除a分支。

11.git tag

客户端开发的时候经常有版本的概念，比如v1.0、v1.1等，不同的版本对应不同的代码，所以一般要给代码加上标签，这样假设v1.1版本出了一个新bug，但是又不清楚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，重新打包测试。

新建一个标签很简单，比如 git tag v1.0 就代表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。



从上图可以看出新建了两个标签 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 命令，就顺利切换到 v1.0 tag的代码状态了。

- 把tag推送到远程仓库
 - 列出标签 `git tag`
 - 搜索标签 `git tag -l 'v0.1.*'`
 - 推送标签到远程仓库，git push并不会把tag标签传送到远端服务器上，只有通过显式命令才能分享标签到远端仓库。
 - 1.push单个tag，命令格式为：git push origin [tagname]，例如：

```
git push origin v1.0 #将本地v1.0的tag推送到远端服务器
```

- 2.push所有tag，命令格式为：git push [origin] --tags，例如：

```
git push --tags
#或者
git push origin --tags
```

以上是一些最基本的Git操作，并且是在本地环境进行操作的，完全没有涉及到远程仓库，接下来的课程将以远程 GitHub 仓库为例，讲解本地如何与远程仓库一起同步协作，另外，本节课讲述的是最基础最简单的Git操作，后续会讲解一些Git的高阶以及一些Git的酷炫操作。

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]
→ stormzhang git:(master)
```

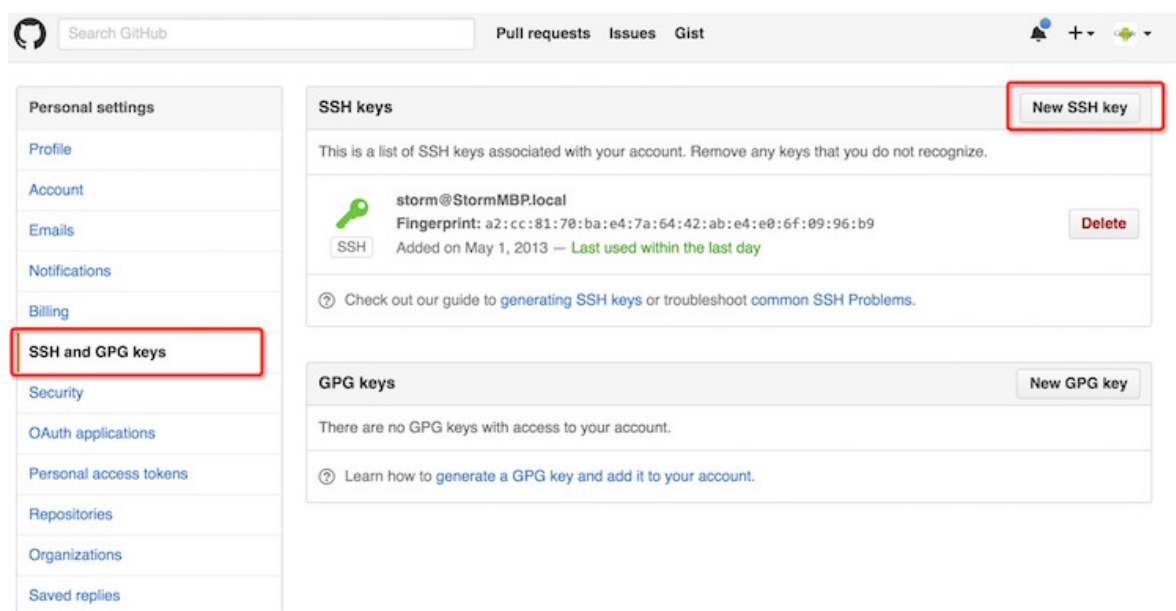
接下来输入 ssh-keygen -t rsa 命名，意思是指定 rsa 算法生成密钥，接着连续三个回车键（不需要输入密码）就会生成两个文件 id_rsa 和 id_rsa.pub，id_rsa 是密钥，id_rsa.pub 是公钥。这两个文件默认分别在如下目录里生成：

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。

GitHub上添加SSH Key

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



然后点击右上角的 New SSH key 按钮：

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 栏不需要填写，点击 Add SSH key 按钮就ok了。

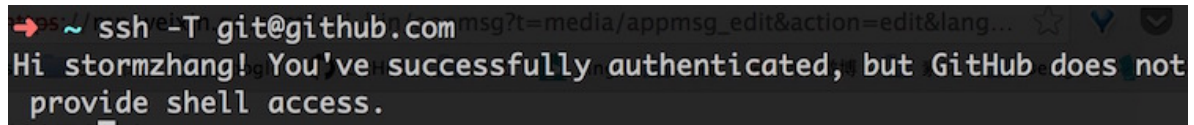
这里提醒下，怎么查看 id_rsa.pub 文件的内容？

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。

A terminal window showing the command `ssh -T git@github.com` and its output: `Hi stormzhang! You've successfully authenticated, but GitHub does not provide shell access.`

Push & Pull 命令介绍

在提交代码之前首先了解两个命令，这两个命令需要与远程仓库配合。

Push：直译过来就是「推」的意思，如果你本地代码有更新，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。一般在 push 之前都会先 pull，这样不容易冲突。

提交代码

添加 SSH key 成功后，就有权限向 GitHub 提交自己的项目代码了，提交代码有两种方法：

（1）Clone 自己的项目

此处以我在 GitHub 上创建的 test 项目为例，执行如下命令：

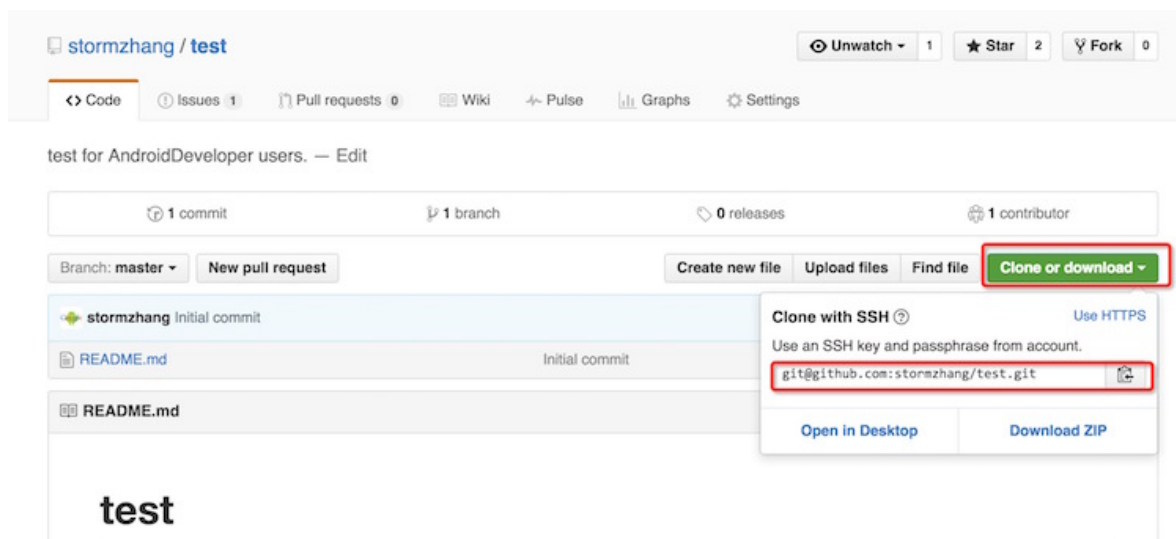
```
git clone git@github.com:stormzhang/test.git
```

这样就把 test 项目 clone 到了本地，可以把 clone 命令理解为高级的复制，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，并且已经关联好了远程仓库，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。

怎么获取项目的仓库地址呢？如下图：



(2) 关联本地已有项目

如果本地已经有一个完整的 git 仓库，并且已经进行了多次 commit，这个时候第一种方法就不适合了。

假设本地有个 test2 的项目，需要在 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
```

意思是添加一个远程仓库，地址是 [git@github.com:stormzhang/test.git](https://github.com/stormzhang/test.git)，而 origin 是给这个项目的远程仓库起的名字（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。

查看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，本地的仓库就可以向远程仓库进行代码提交了：

```
git push origin master
```

此命令就是默认向 GitHub 上的 test 目录提交了代码，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。

注意：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"
```

总结

通过本课时的介绍，终于可以成功的向 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。

用户名和邮箱

通过前面的学习，相信大家已经可以用git管理自己的代码了，但这些还远远不够，关于Git还有很多知识与技巧，接下来就给大家介绍一些 Git 进阶的知识。

在 GitHub 上的每一次commit都会产生一条log，这条log标记了提交人的姓名与邮箱，目的是便于其他人查看与联系提交人，因此，进行提交代码的第一步就是要设置自己的用户名与邮箱。执行以下命令：

```
git config --global user.name "stormzhang"
```

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

以上进行了全局配置，如果某一个项目想要使用特定的邮箱，那么只需切换到相应的项目，将上述代码中的 --global 参数去除，再重新执行一遍就可以了。

注意：在 GitHub 上的每次提交理论上都会在主页的下方产生一条绿色小方块的记录，如果确认提交后，没有绿色方块显示，那一定是所提交代码配置的邮箱与当前 GitHub 上的邮箱不一致，GitHub 上的邮箱可以到 Setting -> Emails里查看。

alias

一些Git命令操作很频繁，例如：

- git commit
- git checkout
- git branch
- git status
- ...

对于这些频繁的操作，每次都要输入完全确实有些麻烦，有没有一种简单的缩写输入呢？比如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？这个时候就用到alias配置，翻译过来就是别名的意思，输入以下命令就可以直接满足以上的需求。

- 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，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 ，直接使用 git psm 和 git plm 就可以代替。

这里给大家推荐一个很强大的 alias 命令，当输入 git log 查看日志的时候是类似这样的：

```
commit 0b5e8caecc2c6ff8426079c005f3452288073463
Author: ttdevs <[redacted]@boohoo.com>
Date:   Wed May 11 13:38:35 2016 +0800

    show cached ip

commit 63ff87d6380135258964ddf689e7b23f438e576b
Merge: 6a5ed4c 0fd7789
Author: loody <[redacted]@boohoo.com>
Date:   Thu May 19 11:04:11 2016 +0800

    Merge branch 'daily_build' into feature/ping++

commit 0fd7789e824f2ca8e67ae4f2f5a2ec81b0a15973
Merge: 48bbea3 9d10a3b
Author: loody <[redacted]@boohoo.com>
Date:   Thu May 19 11:03:45 2016 +0800

    Merge branch 'daily_build' of git.boohoo.cn:android/one into daily_build
```

如果输入git log --graph --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset' --abbrev-commit --date=relative ，日志就会变成这样：

```
* 0b5e8ca - (HEAD, feature/debugger) show cached ip (4 hours ago) <ttdevs>
* 63ff87d - (origin/s1, origin/feature/ping++, feature/ping++) Merge branch 'daily_build' of git.boohoo.cn:android/one into daily_build (4 weeks ago) <loody>
| \
| * 0fd7789 - Merge branch 'daily_build' of git.boohoo.cn:android/one into daily_build (4 weeks ago) <loody>
| \
| | * 9d10a3b - 修改饮食工具 (4 weeks ago) <wanglinglong>
| | * 6a5ed4c - Merge branch 'daily_build' into feature/ping++ (4 weeks ago) <loody>
| | \
| | \
| | * 48bbea3 - Merge branch 'daily_build' of git.boohoo.cn:android/one into daily_build (4 weeks ago) <loody>
| | \
| | \
| | * 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) <wanglinglong>
| | * 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.boohoo.cn:android/one into daily_build (5 weeks ago) <loody>
| | \
| | \
| | * 34c17fe - (origin/feature/init) 完善评测 (5 weeks ago) <wanglinglong>
| | * 1237856 - 修改用户评测 (5 weeks ago) <wanglinglong>
| | \
| | * 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，但是每次都要输入这么长是不是也很烦？这时候你就该想到 alias：


```
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"
```

配置完成后，直接输入 `git lg` 就可以了。

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 `vim`。

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

如果喜欢其他编辑器可自行搜索配置，前提是本机有安装。

有些人会疑问怎样才能让终端显示各种颜色，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。

diff

`diff` 命令是很常用的，在开发过程中，大家会经常做一些代码改动，但时间久了难免会忘记做过哪些改动，在提交之前需要确认下，这个时候就可以用 `diff` 来查看到底做了哪些改动，例如，有一个 `a.md` 的文件，做了一些改动，当输入 `git diff` 时，就会看到如下内容：



```
diff --git a/a.md b/a.md
index ea8f022..d2a5109 100644
--- a/a.md
+++ b/a.md
@@ -1,3 @@
-aaaaaaaa
+bbbbbbbb
+cccccccc
+ddddddd
```

红色的部分前面有个 `-` 代表删除的内容，绿色的部分前面有个 `+` 代表增加的内容，从这里可以一目了然的知道自己到底对这个文件做了哪些改动。

需要注意的是，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？就是还没有执行 `git add` 的文件。

当然除了与暂存区做比较之外，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 ..` # 在两个分支之间比较
- `git diff --staged` # 比较暂存区和版本库差异

checkout

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，也可以用来切换tag，切换到某次commit，如：`git checkout v1.0`

`git checkout ffd9f2dd68f1eb21d36cee50dbdd504e95d9c8f7` # 后面的长串是commit_id，是每次commit的SHA1值，可以根据 `git log` 看到。

除了有“切换”的作用，checkout 还可以用作撤销，例如，假设在一个分支开发一个功能，写到一半时需求变化了，而且是大变化，之前写的代码完全不能用，目前还没有 `git add` 进暂存区，这个时候很简单的一个操作就可以直接把原文件还原：

```
git checkout a.md
```

注意，checkout 命令只能撤销还没有 `add` 进暂存区的文件。

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。同学可能会想先把刚写的一点代码进行提交就行了，这样做理论上是没有问题的，但是原则上每次的 `commit` 都要有实际的意义，目前代码只是写了一半，是不建议 `commit` 的，那么有没有一种比较好的办法，可以暂时切到其他分支，修复完bug再切换回来，并且代码也能保留的？

首先，暂存 `commit` 代码：

此时，`stash` 命令就大有用处了，前提是当前代码没有进行 `commit`，执行了 `add` 也没关系，首先执行命令：`git stash`

意思就是把当前分支所有没有 `commit` 的代码先暂存起来，这个时候再执行 `git status` 命令，会发现当前分支很干净，几乎看不到任何改动，自己代码的改动也看不见，但其实是暂存起来了。

执行命令 `git stash list` 会发现此时暂存区已经有了一条记录。

其次：切换分支，代码还原：

这个时候就可以切换到其他分支，把bug修复好，然后发布。一切都解决了，再切换回来继续之前没做完的功能，但之前的代码怎样还原呢？

执行命令 `git stash apply` 会发现之前的代码全部回来了，就好像一切都没发生过一样。

最后，删除暂存区记录：

接下来最好把暂存区的stash 记录删除，执行：`git stash drop` 就把最近一条的 stash 记录删除了。

代码还原其实还有更简便的，可以执行：`git stash pop` 来代替 `apply` 命令，`pop` 与 `apply` 的唯一区别就是 `pop` 不但会帮将代码还原，还会自动帮将本条 `stash` 记录删除，不需要手动 `drop` 了，为了验证你可以执行 `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，没有参数就是删除最近的，而 clear 是清空。

merge & rebase

大家都知道 merge 分支是合并的意思，当在一个 featureA 分支开发完一个功能需要合并到主分支 master 上时，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，上面的需求同样可以进行如下操作：

- git checkout master
- git rebase featureA

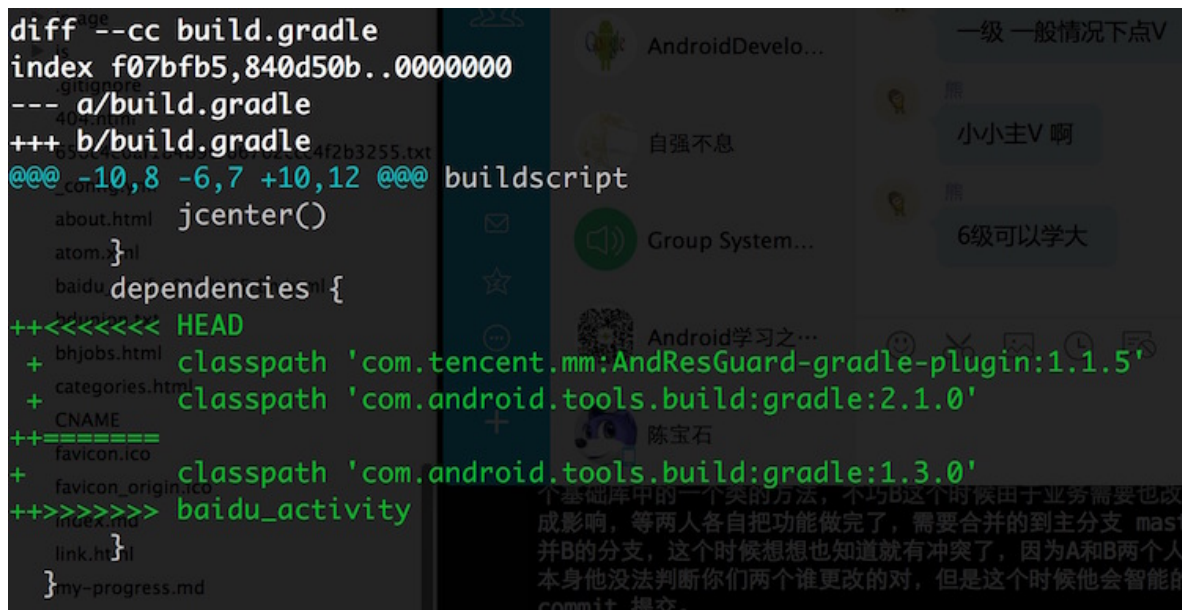
rebase 跟 merge 的区别？

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，第一种做法是 merge，简单粗暴，直接空出一块地方把另一个书架的书全部放进去，这种做法你可以知道哪些书是来自另一个书架；第二种做法就是 rebase，会对两个书架的书先进行比较，按照购书的时间重新排序，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逻辑清晰，但是很难清楚的知道哪些书来自哪个书架。

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。

解决冲突

假设这样一个场景，A和B两位同学各自建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，假设先合并A的分支，这个时候没问题，之后再继续合并B的分支，这个时候就会有冲突了，因为A和B两个人同时更改了同一个地方，Git 本身不能判断谁更改的对，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。接下来，随便在项目中制造一个冲突做演示：



```
diff --cc build.gradle
index f07bfb5,840d50b..0000000
--- a/build.gradle
+++ b/build.gradle
@@@ -10,8 -6,7 +10,12 @@@ buildscript
    jcenter()

    dependencies {
        classpath 'com.tencent.mm:AndResGuard-gradle-plugin:1.1.5'
        classpath 'com.android.tools.build:gradle:2.1.0'
        classpath 'com.android.tools.build:gradle:1.3.0'
        baidu_activity
    }
} my-progress.md
```

上图就是冲突的示例，冲突的地方由 ==== 分出了上下两个部分，上半部分 HEAD 是当前所在分支的代码，下半部分是 baidu_activity 分支的代码，从图中可以看出 HEAD 对 gradle 插件进行了升级，同时新增了一个插件，所以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，同时也要把那些 <<< HEAD、==== 以及 >>>>>baidu_activity 标记符号删除，最后进行一次 commit 就可以了。

在开发的过程中，一般都会约定大家写的代码尽量不要彼此影响，以减少出现冲突的可能，但是冲突终究无法避免，同学们需要了解并掌握解决冲突的方法。

Git分支常用操作

通常默认会有一个主分支叫 master，下面首先来介绍一下关于分支的一些基本操作：

(1) 新建一个叫 develop 的分支

```
git branch develop
```

注意，新建分支的命令是基于当前所在分支进行的，即以上是基于 mater 分支新建了一个叫做 develop 的分支，此时 develop 分支与 master 分支的内容完全一样。例如，有 A、B、C三个分支，各分支内容不同，如果当前是在 B 分支，执行新建分支命令，则新建的分支内容与 B 分支相同，同理如果当前在 C 分支，那就是基于 C 分支基础上新建的分支。

(2) 切换到 develop 分支

```
git checkout develop
```

如果把以上两步合并，即新建并自动切换到 develop 分支：

```
git checkout -b develop
```

(3) 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，执行以下代码：

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

注意：实际开发管理，建议不要这样做，这样会导致很混乱，难管理，建议本地分支与远程分支名称要保持一致。

(4) 查看本地分支列表

```
git branch
```

(5) 查看远程分支列表

```
git branch -r
```

(6) 删除本地分支


```
git branch -d develop  
git branch -D develop (强制删除)
```

(7) 删除远程分支

```
git push origin :develop
```

如果远程分支有 develop，而本地没有，想要将远程的 develop 分支迁到本地：

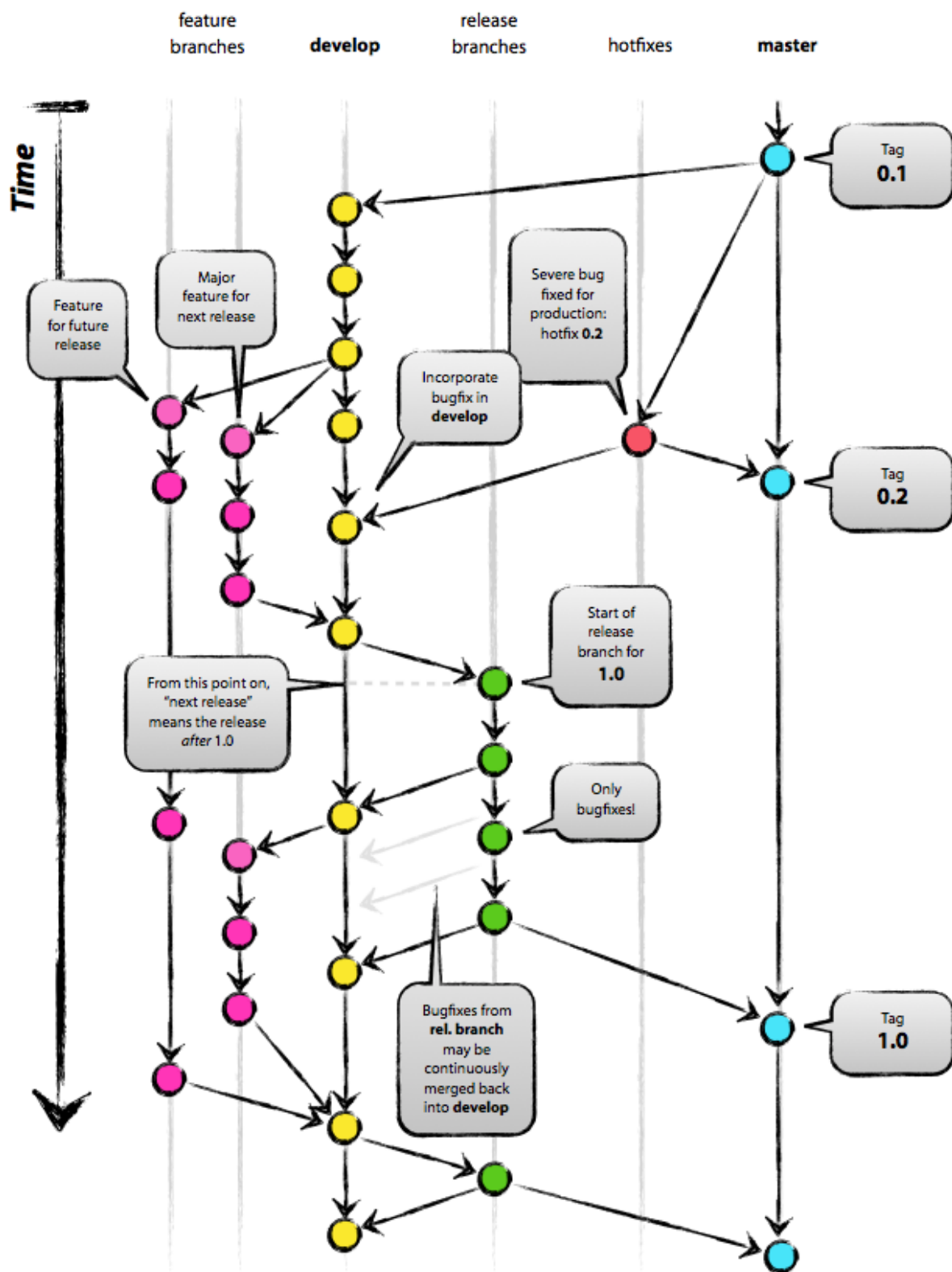
```
git checkout develop origin/develop
```

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
```

GitFlow详细操作

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：



第一次看上面的图是不是有些晕，接下来用简单的语言给大家解释一下。

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，准备发布时，会将 **develop** 合并到 **master** 上。

但是产品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的 **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

举个例子, 假设已经有 master 和 develop 两个分支了, 这个时候准备做一个功能 A, 第一步要做的就是基于 develop 分支新建个分支:

```
git branch feature/A
```

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。

但是这个时候发现线上有一个紧急的 bug 需要修复, 可以立刻切换到 master 分支, 然后再此基础上新建一个分支:

```
git branch hotfix/B
```

代表新建一个紧急修复分支, 修复完成之后直接合并到 develop 和 master, 然后发布。

然后再切换回 feature/A 分支继续开发, 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, 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
```

此时, 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, 达到发布的标准, 就可以把该分支合并到 develop 和 master 然后进行发布。

以上就是 Git Flow 的概念与大体流程, 看起来有些复杂, 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种复杂的情况, Git Flow 是目前很流行的一套分支管理流程, 但是有人会觉得每次都要进行各种合并操作有些麻烦, Git Flow 为此专门推出了一个工具, 并且是开源的: GitHub 开源地址: <https://github.com/nvie/gitflow>

简单来说, 这个工具会帮大家节省很多步骤, 例如, 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 直接如下操作就可以完成了:

```
git flow feature start A
```

这个分支完成之后, 需要合并到 develop 分支, 进行如下操作:

```
git flow feature finish A
```

如果是 hotfix 或者 release 分支, 会自动合并到 develop、master 两个分支。

到目前为止, 大家已经了解了这个工具的具体作用, 具体安装与用法就不多介绍了, 感兴趣的同学可以观看这篇博客: <http://stormzhang.com/git/2014/01/29/git-flow/>

总结

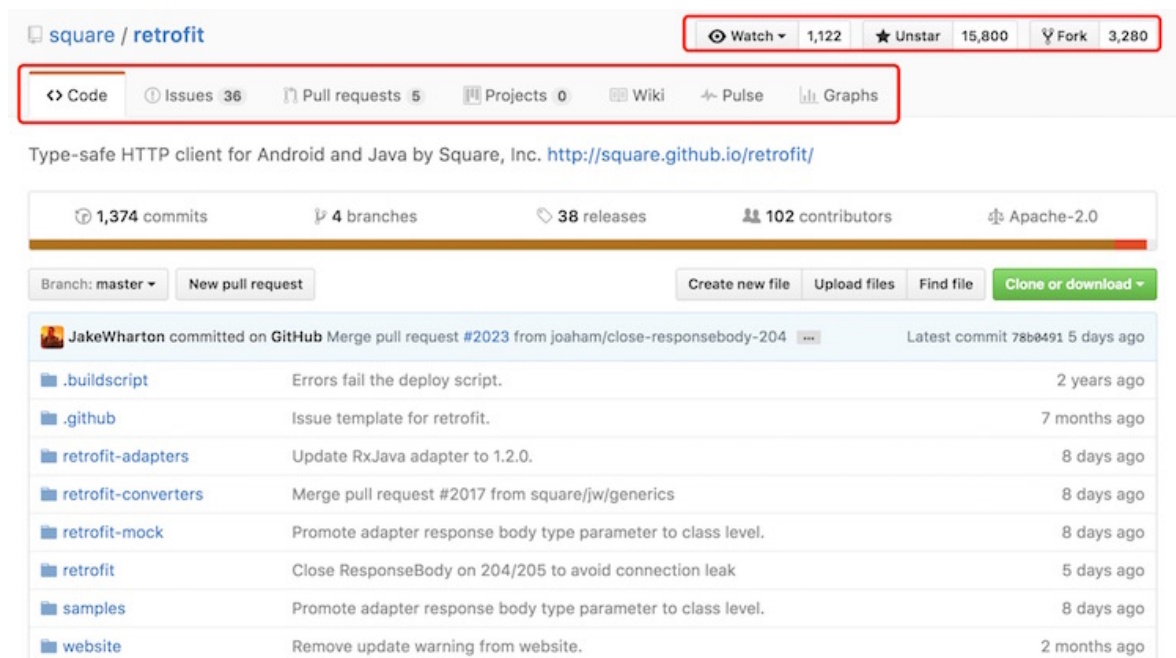
以上就是分享给大家的关于分支的所有知识, 一个人你也许感受不到什么, 但是实际工作中大都以团队协作为主, 而分支是团队协作必备技能, Git Flow 是一个很流行的分支管理流程, 也是公司团队内部经常使用的一套流程, 希望大家能够掌握。

Git常见的集中操作

开源社区最大的魅力是人人都是可以参与进去，发挥众人的力量，让一个项目更完善，更强壮。那么就会有人疑问，自己目前还没有能力开源一个项目，但是又想一起参与到别人的开源项目中，该怎样操作呢？本课时，就来给大家介绍 GitHub 上的一些常见的操作。

下面以 Square 公司开源的 Retrofit 为例来介绍。

打开链接：<https://github.com/square/retrofit>，可以看到如下的项目主页：



从上图可以看出，一个项目可以进行的操作主要有两部分，第一部分包括 Watch、Star、Fork，这三个操作前面的课时已经介绍过了，这里不做重复讲解。

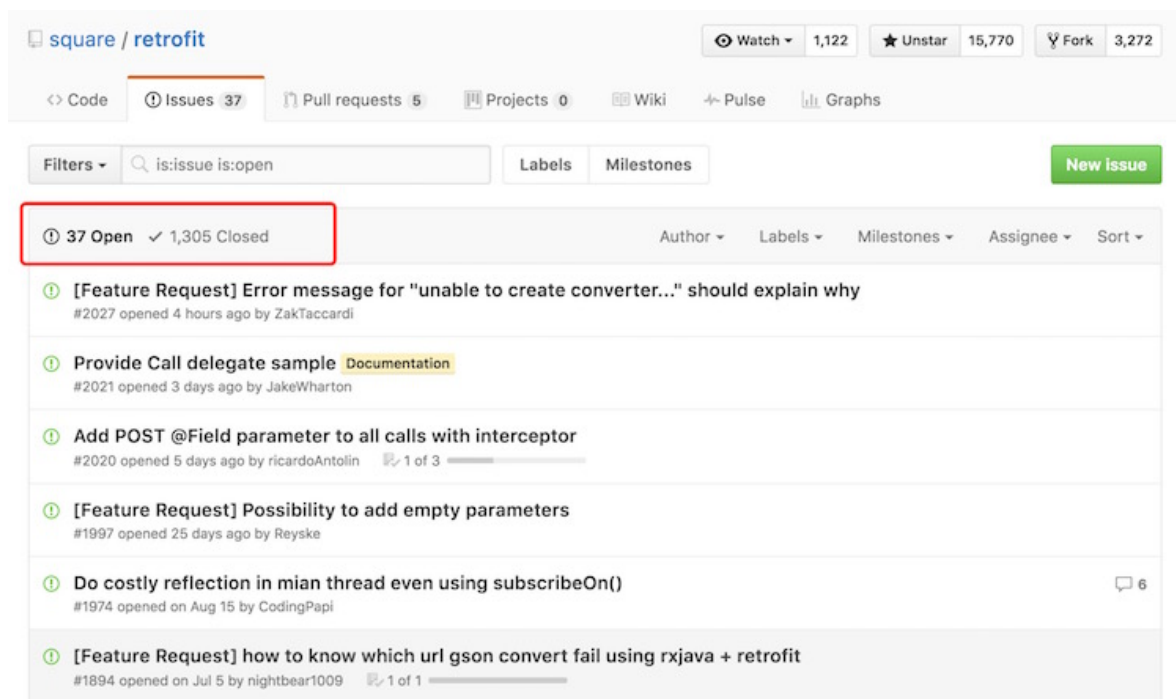
着重介绍第二部分，分别包括 Code、Issues、Pull requests、Projects、Wiki、Pulse、Graphs。接下来一一进行讲解。

(1) Code

代表项目的代码文件，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。

(2) Issues

Issues 代表该项目的一些问题或者 bug，并不是说 Issue 越少越好，Issue 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。观察 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，这种解决问题的比例与速度值得每位开源项目的作者学习。



同样的，大家在使用一些开源项目遇到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，添加一个标题与描述就可以了，这个操作很简单。

(3) Pull requests

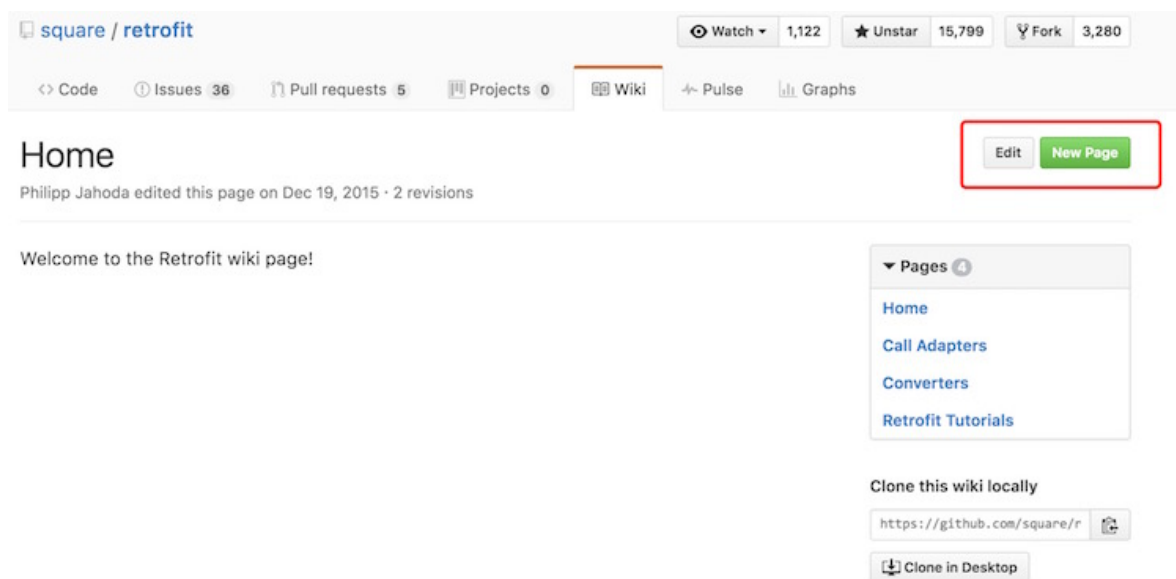
别人开源一个项目，如果我们想一起参与开发，一起来完善，这就要通过 Pull requests 来完成，简称 PR。

(4) Projects

这个是最新 GitHub 改版新增的一个项目，这个项目就是方便将一些 Issues、Pull requests 进行分类，目前为止该功能被使用的比较少，了解一下就好。

(5) Wiki

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

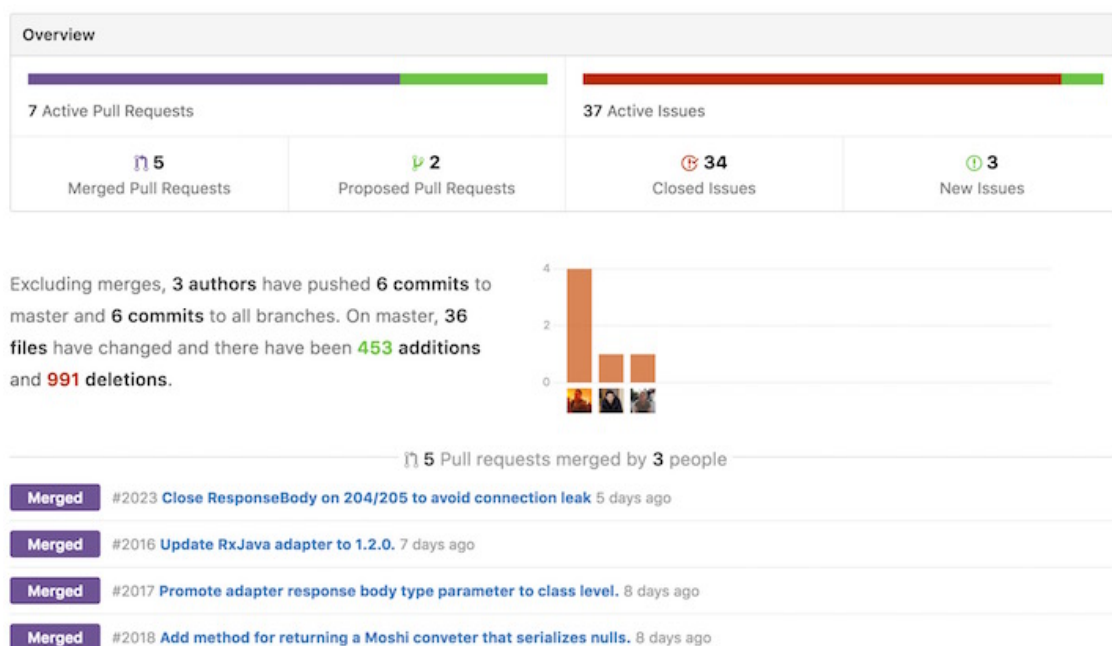


Wiki使用起来也很简单，直接 New Page ，然后使用 markdown 语法即可进行编写。

(6) Pulse

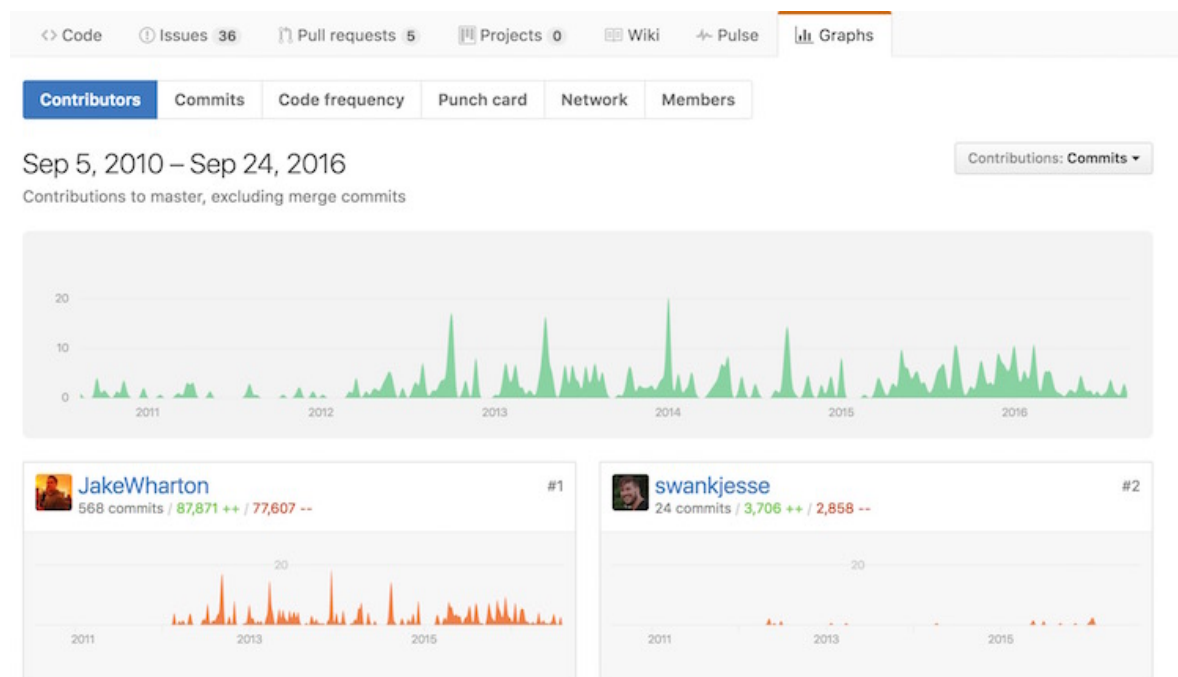
Pulse 可以理解成该项目的活跃汇总。包括近期该仓库创建了多少个 Pull Request 或 Issue，有多少人参与了这个项目的开发等，在这里都可以一目了然。

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。



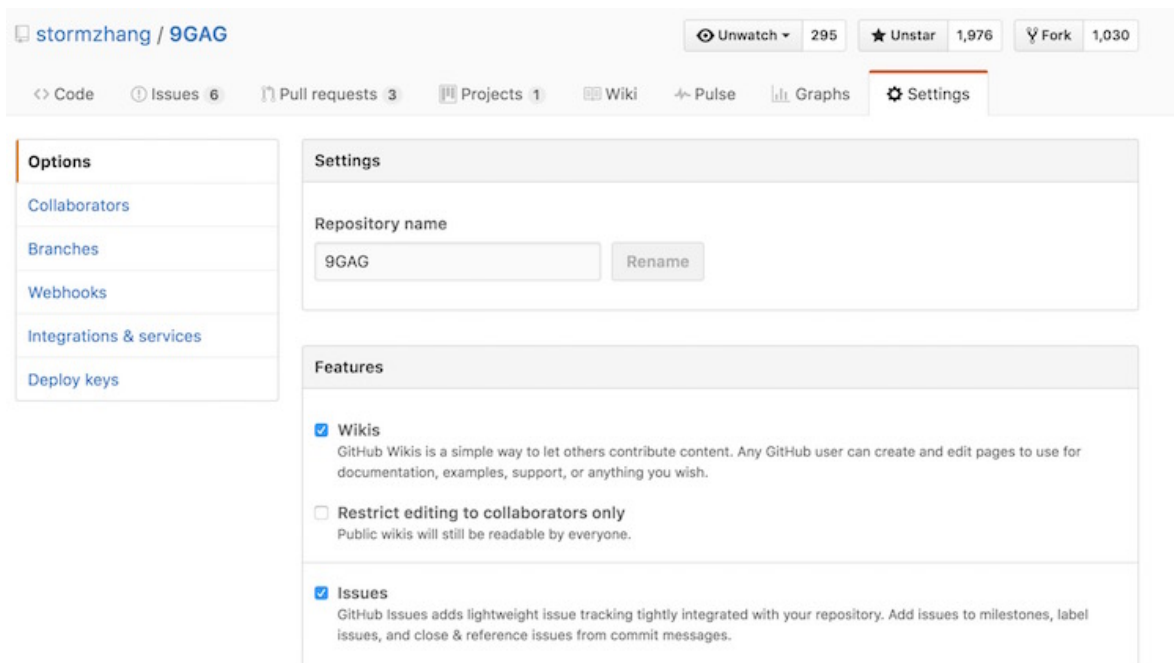
(7) Graphs

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



(8) Settings

如果一个项目是自己的，那么你会发现有一个菜单 Settings，这里包括了对整个项目的设置信息与功能，比如对项目重命名，删除该项目，关闭项目的 Wiki 和 Issues 功能等，不过大部分情况下都不需要对这些设置做更改。感兴趣的同学，可以自行了解这里的设置有哪些功能。



以上就包含了一个 GitHub 项目的操作，对初学者来说难度最主要的还是发起 Pull request，不过相信大家看完之后对 GitHub 上一些常用的操作应该已经熟悉了，从现在开始，请一起参与到开源社区中来吧，开源社区需要每个人都贡献一份力量，这样才能够越来越强大，也能够对更多的人有帮助！

提交自己的第一个PR

接下来我们向项目 9GAG 发起 PR 操作，说明，必须确保你可以正常向 GitHub 提交代码，如果不可以的话，请仔细观看前面课时讲解的内容。

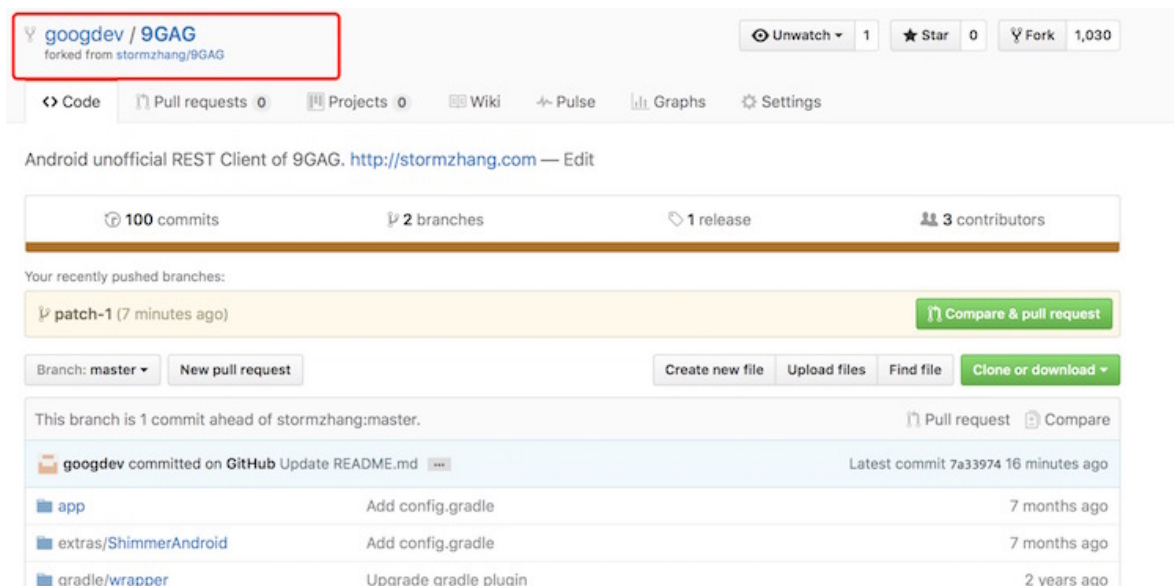
第一步，登录你的 GitHub 账号，然后找到想要发起 PR 的项目，这里以 [9GAG](https://github.com/stormzhang/9GAG) 为例，地址：

<https://github.com/stormzhang/9GAG>

点击右上角的 Fork 按钮，该项目就出现在你自己账号的 Repository 里。

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。

Fork 之后，在账号 googdev 下多了一个 9GAG 的项目，如下图：

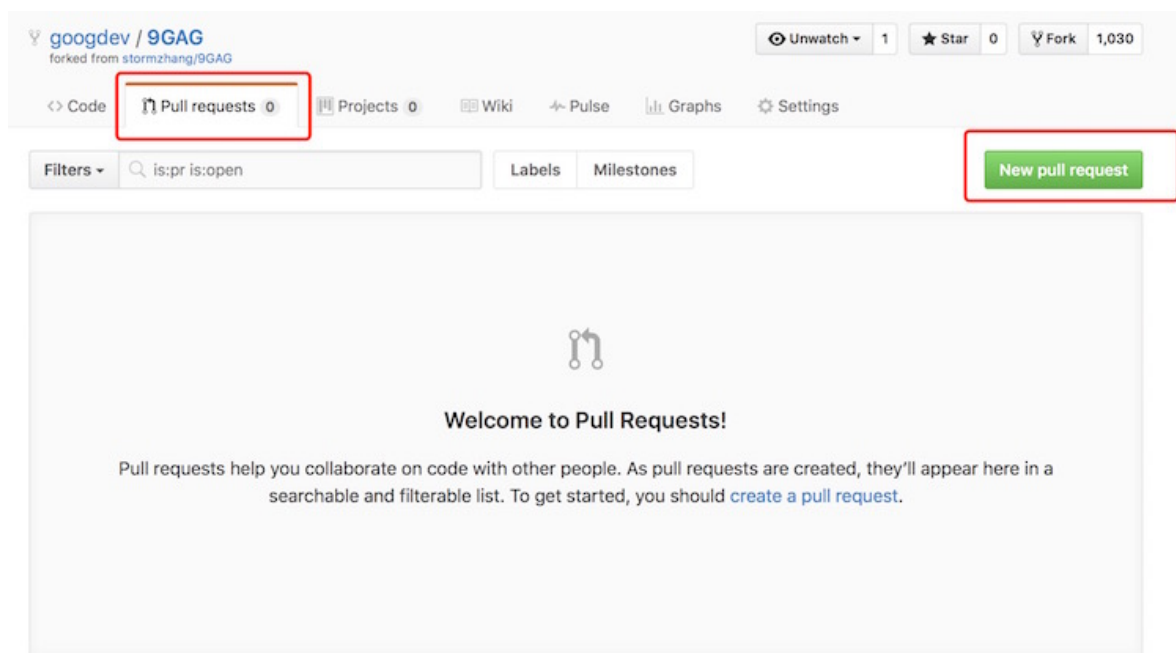


从上图可以看出 Fork 过来的项目标题底部会显示一行小字：fork from stormzhang/9GAG，除此之外，项目代码跟原项目一模一样，对于原项目来说，相当于别人新建了一个分支而已。

第二步，把该项目 clone 到本地，把自己新增或者改动的代码保存。

第三步，把自己做的代码改动 push 到你自己的 GitHub 项目上去。

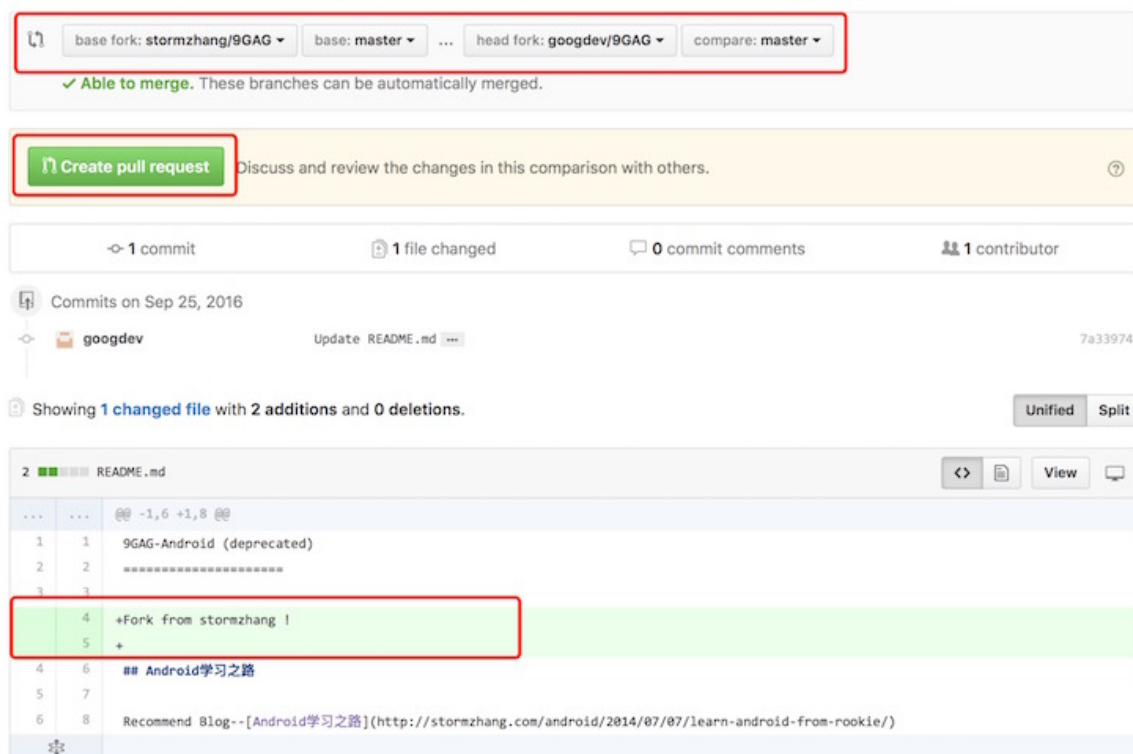
第四步，点击你 Fork 过来的项目主页的 Pull requests 页面。



点击 New pull request 按钮会看到如下页面：

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](#).



这个页面会自动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 项目的 master 分支所做的比较。

最底部可以方便直观的看到代码中做了哪些改动，这里可以看到我加了一句 Fork from stormzhang !

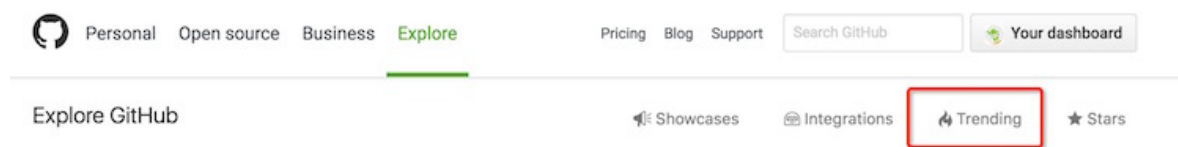
同样的，写好标题和描述，然后点击中间的 Create pull request 按钮，至此就成功给该项目提交了一个 PR。

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，那么你就已经是该项目的贡献者之一了。

Trending(趋势)

点击下图的 Explore 菜单到“发现”页面

点击下图的 Trending 按钮



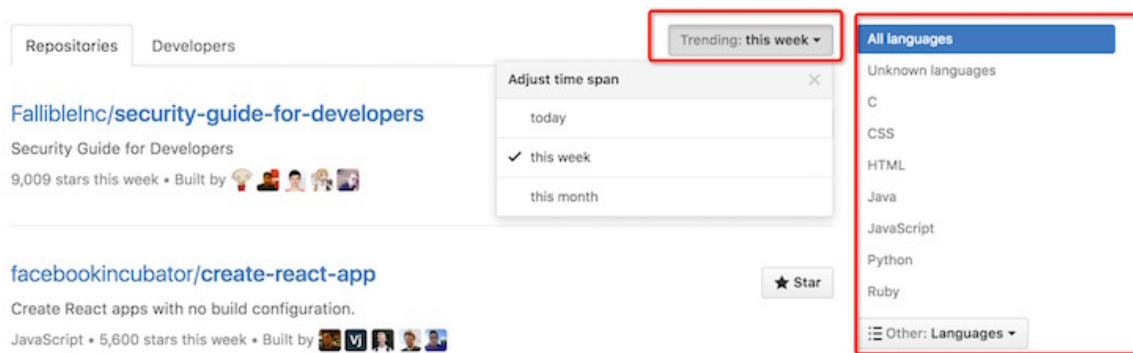
Project showcases

Browse interesting repositories, solving all types of interesting problems.

Trending 直译过来是趋势的意思，也就是说从这个页面可以看到最近一些热门的开源项目，这个页面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.



这个页面推荐大家每隔几天就去查看一下，主动发掘一些优秀的开源项目。

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。

例如，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：

Search

android

Search

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

[Advanced search](#) [Cheat sheet](#)

We've found 373,145 repository results

Sort: Most stars

Trinea/android-open-project

★ 17,353 🔗 9,458

Collect and classify **android** open source projects 微信公众号: codekk
Updated 13 days ago

Prinzhorn/skrollr

HTML ★ 15,926 🔗 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

wasabeef/awesome-android-ui

★ 15,691 🔗 4,362

A curated list of awesome **Android** UI/UX libraries
Updated 12 days ago

square/retrofit

Java ★ 14,286 🔗 2,933

Type-safe HTTP client for **Android** and Java by Square, Inc.
Updated 3 days ago

codepath/android_guides

★ 13,725 🔗 3,391

Extensive Open-Source Guides for **Android** Developers
Updated 7 days ago

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

ios

Search

 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

[Advanced search](#) [Cheat sheet](#)

We've found 135,181 repository results

Sort: Most stars

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 数排序，排名靠前的都是一些比较火的项目，一定是很有用，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。

对于实际项目中用到的一些库，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，因为我只是想找到关于 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，得到如下的结果：

The screenshot shows the GitHub search interface. At the top, there's a search bar with 'android http' entered and a 'Search' button. Below the search bar, on the left, is a sidebar with filters: 'Repositories' (6,705), 'Code' (71,686,340), 'Issues' (896,804), and 'Users'. Below this is a 'Languages' section with a list of languages and their counts: Java (4,002), C (512), C++ (209), JavaScript (125), C# (99), Shell (81), Python (49), PHP (45), Makefile (32), and HTML (25). The main area displays 'We've found 6,705 repository results' with a 'Sort: Most stars' dropdown. Three repositories are listed: 1. 'square/retrofit' (Java, 14,286 stars, 2,933 forks) described as a 'Type-safe HTTP client for Android and Java by Square, Inc.' updated 3 days ago. 2. 'square/okhttp' (Java, 13,091 stars, 3,411 forks) described as an 'HTTP + HTTP/2 client for Android and Java applications.' updated 3 hours ago. 3. 'loopj/android-async-http' (Java, 9,078 stars, 4,228 forks) described as an 'Asynchronous HTTP Library for Android' updated on Jun 27.

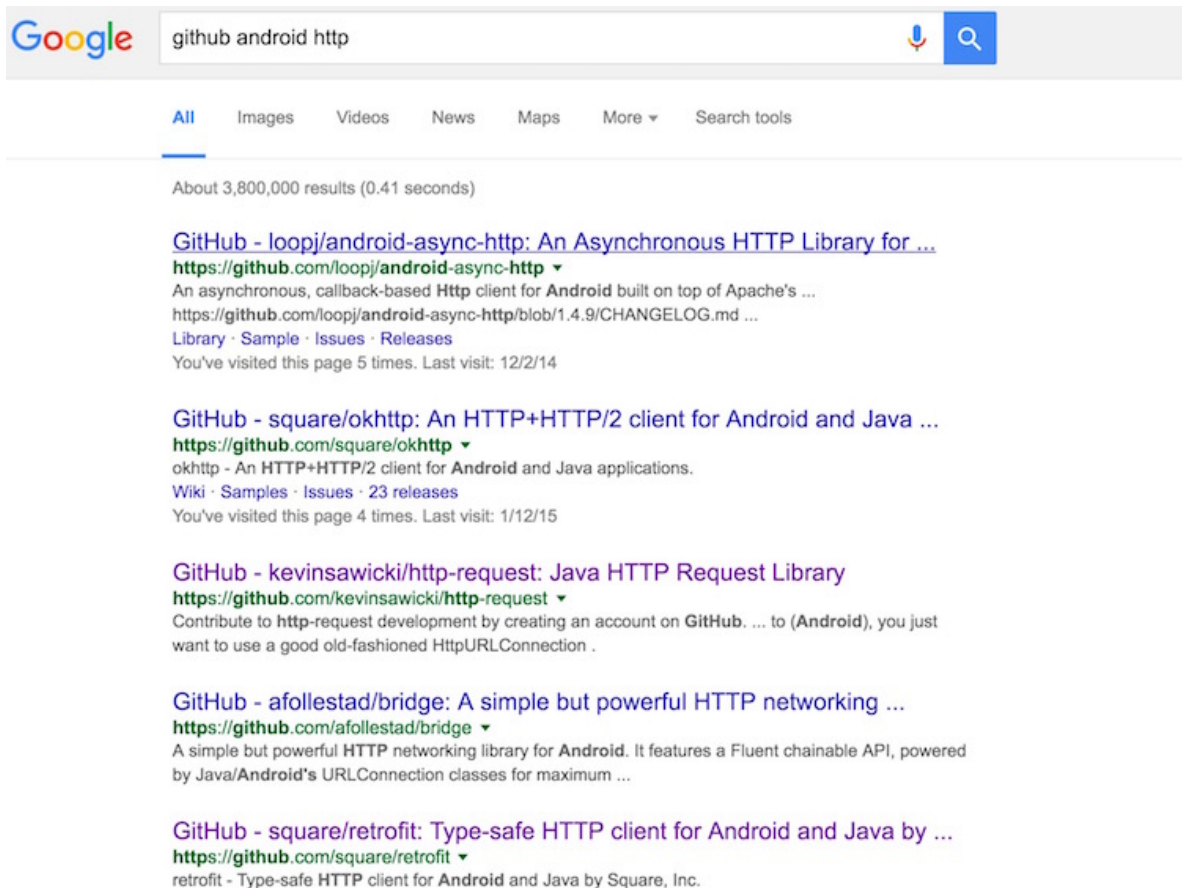
可以看到 Retrofit、OkHttp、android-async-http 是最流行的网络库，只不过 android-async-http 的作者不维护了，如果你不了解哪个网络库好用，那么单纯的按照这种方式就应该优先选择 Retrofit 或者 OkHttp，而目前绝大部分 Android 开发者确实也都是在用这两个网络库，当然还有部分在用 Volley 的，因为 google 没有选择在 GitHub 开源 volley，所以排行上看不到 volley。

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

当然还有些其他小技巧，这里就不多介绍了。

如果使用 Google 浏览器，想要搜索 GitHub 上的结果，可以在前面加 GitHub 关键字，比如 输入 GitHub android http，每个关键字用空格隔开，搜索结果如下：



从上图可以看到，基本也是想要的结果，只是不是单纯的按照 star 数来排序的。

资源项目

相信以上三种方法足够大家遨游在 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，同时又对大家很有帮助的项目：

- [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，包含全球多国语言版本，中文版的在这里：[free-programming-books-zh](#)，从这个项目中，你几乎可以获取任何编程相关的学习资料，强烈推荐给大家！

- [oh-my-zsh](#)

俗话说，不会用 shell 的程序员不是真正的程序员，建议每个程序员都要懂一些 shell，oh-my-zsh 就是目前最流行，最酷炫的 shell，不过多介绍了，大家自己了解。

- [awesome](#)

GitHub 上有各种 awesome 系列，这个系列将 GitHub 上各领域的资源进行了大汇总，比如 awesome-android, awesome-ios, awesome-java, awesome-python 等等，就不截图了，大家自行去感受。

- [github-cheat-sheet](#)

GitHub 的使用有各种技巧，只不过基本的就够大家用了，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 粉丝都应该知道这个项目。

- [android-open-project](#)

这个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，本项目只整理了所有跟 Android UI 相关的优秀开源项目，基本在实际开发中用到的各种效果上面都能找到类似的项目，简直是开发必备。

- [Android Data](#)

这个项目几乎包括了国内各种学习 Android 的资料，强烈推荐大家收藏起来。

- [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，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很多，就不一一推荐了，授人以鱼不如授人以渔，请大家自行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。

kk