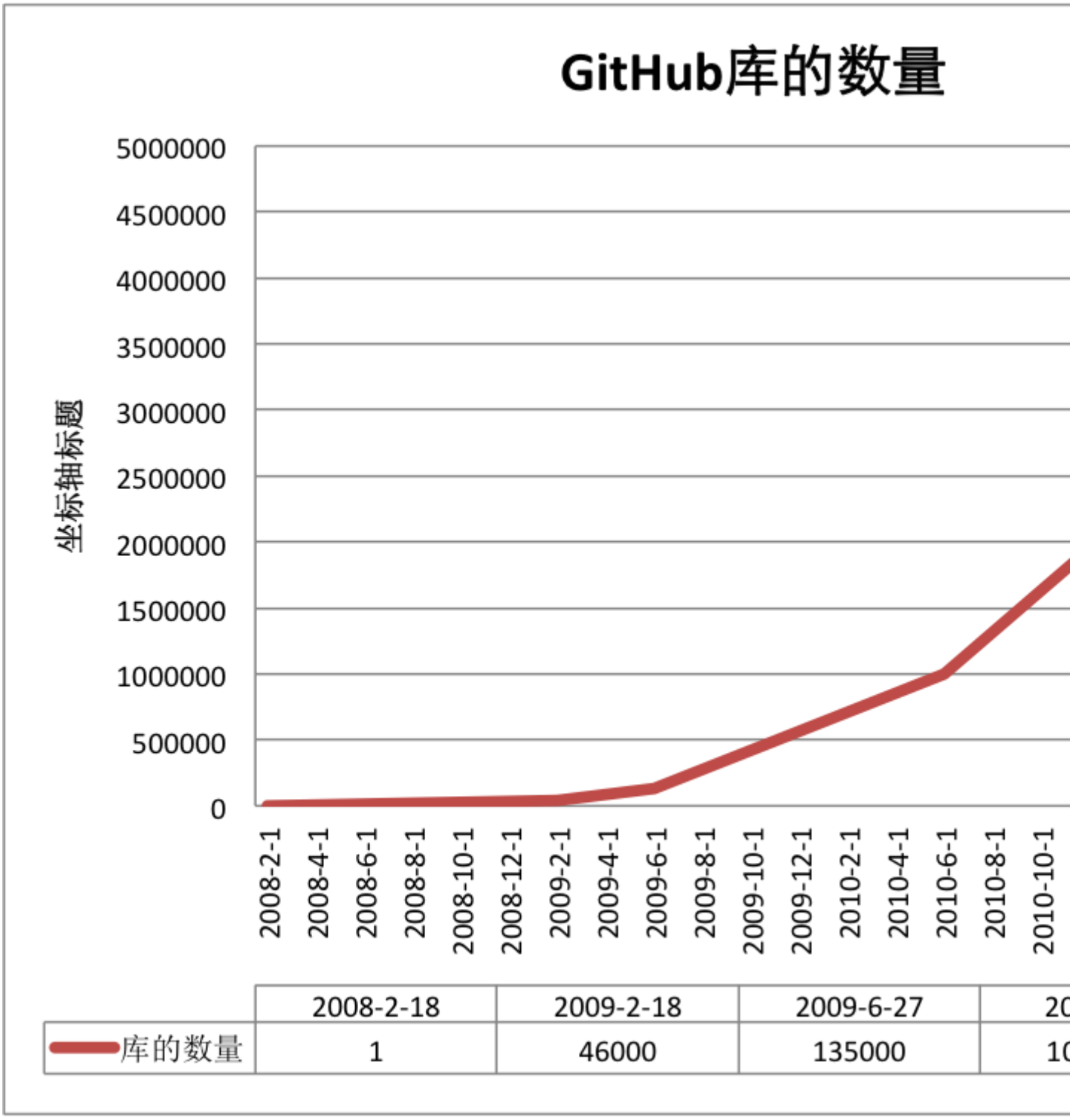
如果你是一枚 Coder，但是你不知道 Github，那么我觉的你就不是一个菜鸟级别的 Coder，因为你压根不是真正 Coder，你只是一个 Code 搬运工。
但是你如果已经在读这篇文章了，我觉的你已经知道 Github 了。
正是 Github，让社会化编程成为现实。

## 什么是 Github?

github 是一个基于 git 的代码托管平台，付费用户可以建私人仓库，我们一般的免费用户只能使用公共仓库，也就是代码要公开。
Github 由 Chris Wanstrath, PJ Hyett 与 Tom Preston-Werner 三位开发者在 2008 年 4 月创办。迄今拥有 59 名全职员工，主要提供基于 git 的版本托管服务。
目前看来，GitHub 这场冒险已经胜出。根据来自维基百科关于 GitHub 的描述，我们可以形象地看出 GitHub 的增长速度：




今天，GitHub 已是：

•一个拥有 143 万开发者的社区。其中不乏 Linux 发明者 Torvalds 这样的顶级黑客，以及 Rails 创始人 DHH 这样的年轻极客。

•这个星球上最流行的开源托管服务。目前已托管 431 万 git 项目，不仅越来越多知名开源项目迁入 GitHub，比如 Ruby on Rails、jQuery、Ruby、Erlang/OTP；近三年流行的开源库往往在 GitHub 首发，例如：BootStrap、Node.js、CoffeScript 等。

•alexa 全球排名 414 的网站。

## 注册账户以及创建仓库

要想使用 github 第一步当然是注册 github 账号了， github 官网地址：https://github.com/。 之后就可以创建仓库了（免费用户只能建公共仓库），Create a New Repository，填好名称后 Create，之后会出现一些仓库的配置信息，这也是一个 git 的简单教程。

## Github 安装

- 下载 git OSX 版
- 下载 git Windows 版
- 下载 git Linux 版

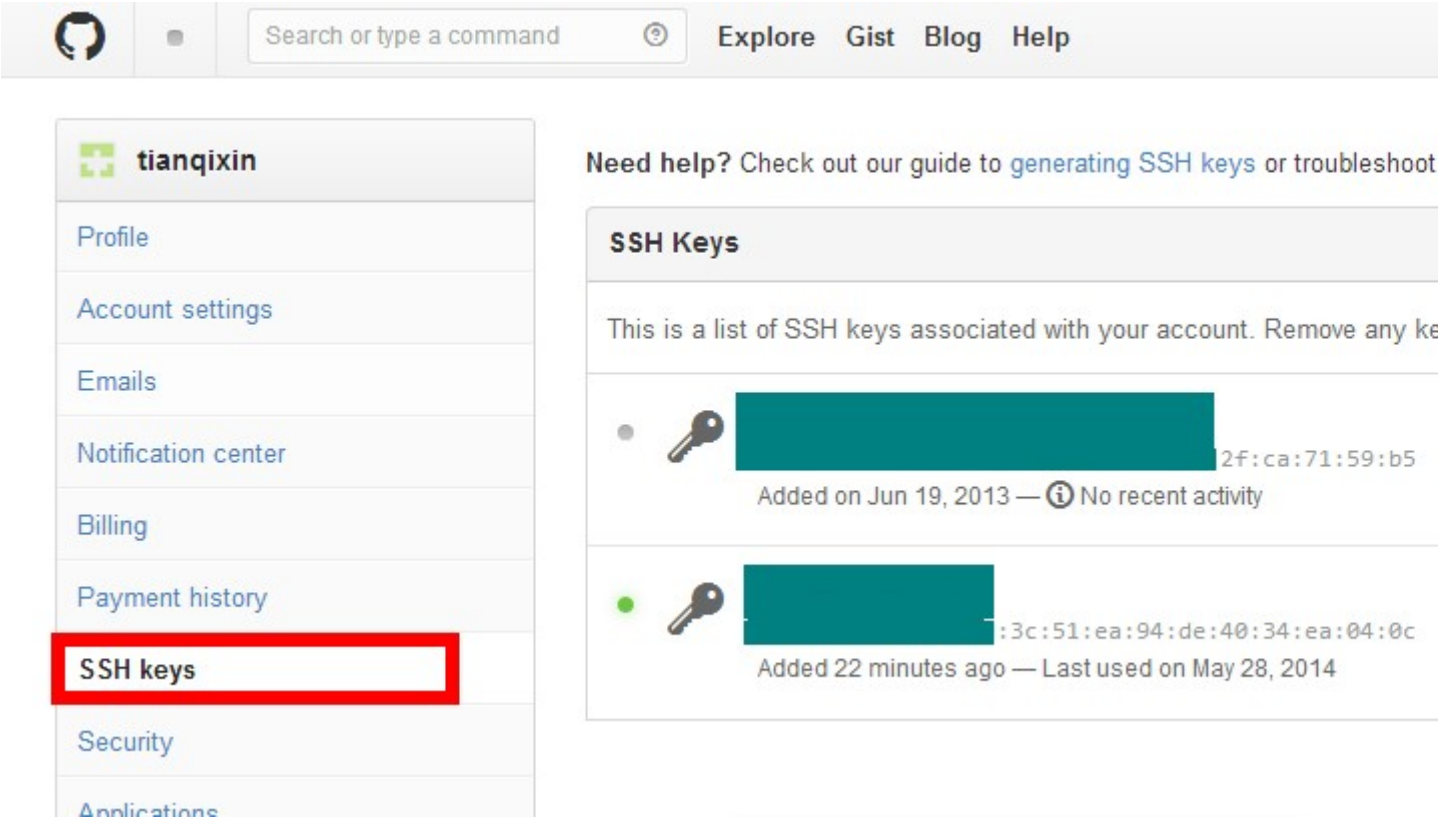
## 配置 Git

首先在本地创建 ssh key；

```
$ ssh-keygen -t rsa -C "your_email@youremail.com"
```

后面的 your_email@youremail.com 改为你在 github 上注册的邮箱，之后会要求确认路径和输入密码，我们这使用默认的一路回车就行。成功的话会在~/下生成.ssh 文件夹，进去，打开 id_rsa.pub，复制里面的 key。

回到 github 上，进入 Account Settings（账户配置），左边选择 SSH Keys，Add SSH Key,title 随便填，粘贴在你电脑上生成的 key。




为了验证是否成功，在 git bash 下输入：

```
$ ssh -T git@github.com
```

如果是第一次的会提示是否 continue，输入 yes 就会看到：You've successfully authenticated, but GitHub does not provide shell access 。这就表示已成功连上 github。

接下来我们要做的就是把本地仓库传到 github 上去，在此之前还需要设置 username 和 email，因为 github 每次 commit 都会记录他们。

```
$ git config --global user.name "your name"
$ git config --global user.email "your_email@youremail.com"
```

进入要上传的仓库，右键 git bash，添加远程地址：

```
$ git remote add origin git@github.com:yourName/yourRepo.git
```

后面的 yourName 和 yourRepo 表示你再 github 的用户名和刚才新建的仓库，加完之后进入.git，打开 config，这里会多出一个 remote "origin"内容，这就是刚才添加的远程地址，也可以直接修改 config 来配置远程地址。
创建新文件夹，打开，然后执行 git init 以创建新的 git 仓库。

## 检出仓库

执行如下命令以创建一个本地仓库的克隆版本：

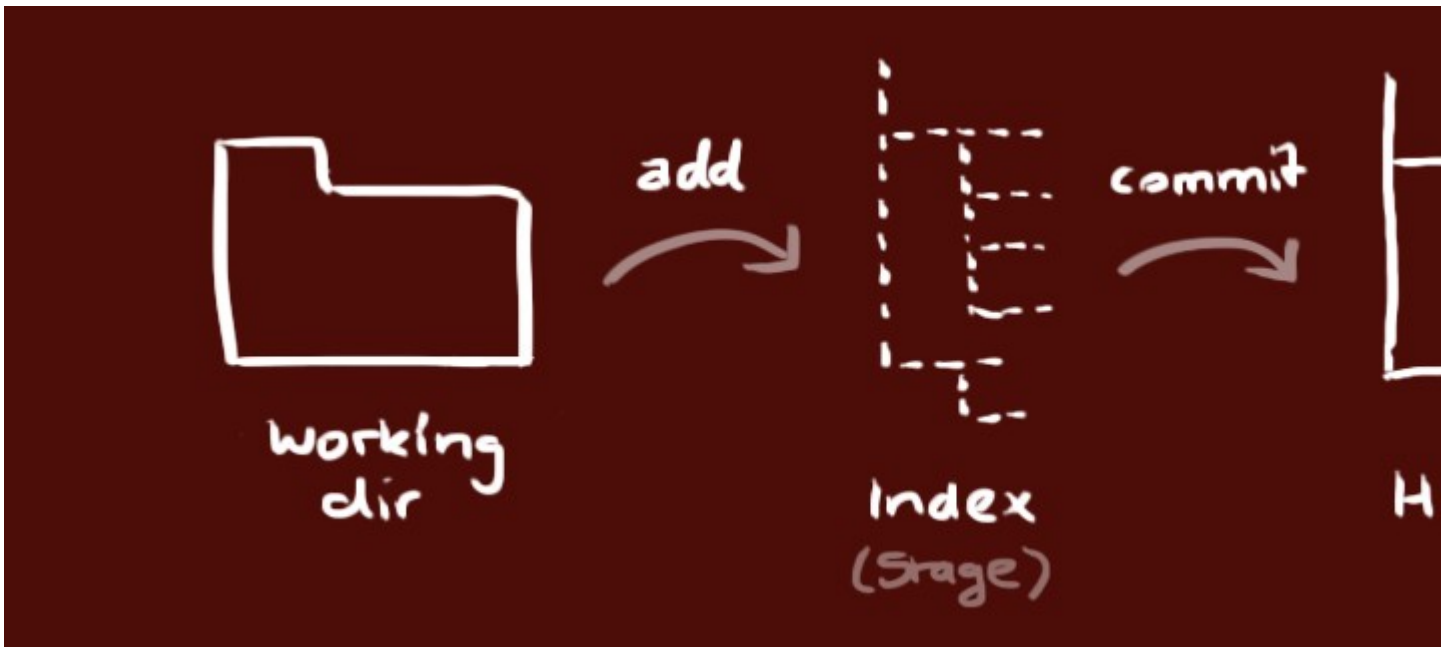
```
git clone /path/to/repository
```

如果是远端服务器上的仓库，你的命令会是这个样子：

```
git clone username@host:/path/to/repository
```

## 工作流

你的本地仓库由 git 维护的三棵"树"组成。第一个是你的 工作目录，它持有实际文件；第二个是 暂存区（Index），它像个缓存区域，临时保存你的改动；最后是 HEAD，它指向你最后一次提交的结果。
你可以提出更改（把它们添加到暂存区），使用如下命令：
git add <filename>
git add *
这是 git 基本工作流程的第一步；使用如下命令以实际提交改动：
git commit -m "代码提交信息"
现在，你的改动已经提交到了 **HEAD**，但是还没到你的远端仓库。



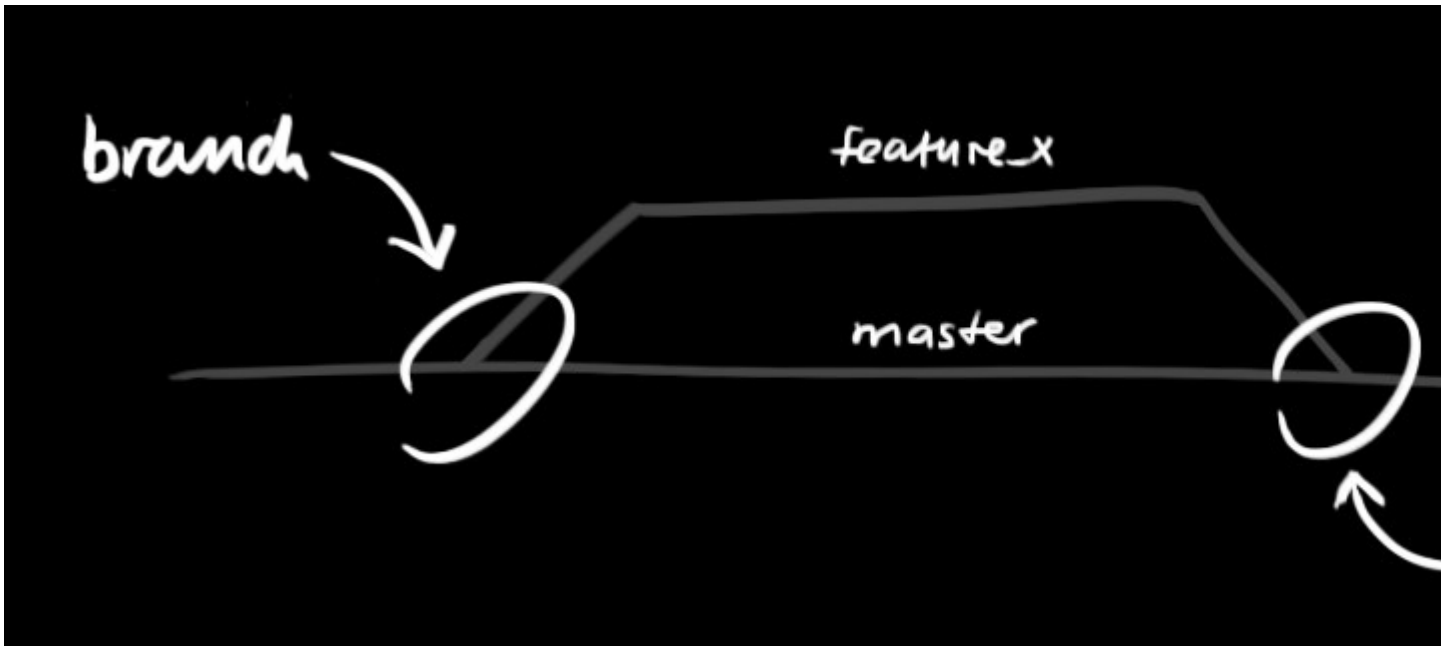

## 推送改动

你的改动现在已经在本地仓库的 **HEAD** 中了。执行如下命令以将这些改动提交到远端仓库：
git push origin master
可以把 *master* 换成你想要推送的任何分支。

如果你还没有克隆现有仓库，并欲将你的仓库连接到某个远程服务器，你可以使用如下命令添加：
git remote add origin <server>
如此你就能够将你的改动推送到所添加的服务器上去了。

## 分支

分支是用来将特性开发绝缘开来的。在你创建仓库的时候，*master* 是"默认的"分支。在其他分支上进行开发，完成后再将它们合并到主分支上。




创建一个叫做"feature_x"的分支，并切换过去：
git checkout -b feature_x

切换回主分支：
git checkout master
再把新建的分支删掉：
git branch -d feature_x
除非你将分支推送到远端仓库，不然该分支就是 *不为他人所见的*：
git push origin <branch>

## 更新与合并

要更新你的本地仓库至最新改动，执行：
git pull
以在你的工作目录中 *获取（fetch）* 并 *合并（merge）* 远端的改动。
要合并其他分支到你的当前分支（例如 master），执行：
git merge <branch>
在这两种情况下，git 都会尝试去自动合并改动。遗憾的是，这可能并非每次都成功，并可能出现*冲突（conflicts）*。 这时候就需要你修改这些文件来手动合并这些*冲突（conflicts）*。改完之后，你需要执行如下命令以将它们标记为合并成功：
git add <filename>
在合并改动之前，你可以使用如下命令预览差异：
git diff <source_branch> <target_branch>

## 标签

为软件发布创建标签是推荐的。这个概念早已存在，在 SVN 中也有。你可以执行如下命令创建一个叫做 *1.0.0* 的标签：
git tag 1.0.0 1b2e1d63ff
*1b2e1d63ff* 是你想要标记的提交 ID 的前 10 位字符。可以使用下列命令获取提交 ID：
git log
你也可以使用少一点的提交 ID 前几位，只要它的指向具有唯一性。

## 替换本地改动

假如你操作失误（当然，这最好永远不要发生），你可以使用如下命令替换掉本地改动：
git checkout -- <filename>
此命令会使用 HEAD 中的最新内容替换掉你的工作目录中的文件。已添加到暂存区的改动以及新文件都不会受到影响。
假如你想丢弃你在本地的所有改动与提交，可以到服务器上获取最新的版本历史，并将你本地主分支指向它：
git fetch origin
git reset --hard origin/master

## 实用小贴士

内建的图形化 git：
gitk
彩色的 git 输出：
git config color.ui true
显示历史记录时，每个提交的信息只显示一行：
git config format.pretty oneline

交互式添加文件到暂存区：
git add -i

## 链接与资源

### 图形化客户端

- GitX (L) (OSX, 开源软件)
- Tower (OSX)
- Source Tree (OSX, 免费)
- GitHub for Mac (OSX, 免费)
- GitBox (OSX, App Store)

### 指南和手册

- Git 社区参考书
- 专业 Git
- 像 git 那样思考
- GitHub 帮助
- 图解 Git

### 相关文章

- Github 简明指南：http://rogerdudler.github.io/git-guide/index.zh.html
- 如何高效利用 GitHub:http://www.yangzhiping.com/tech/github.html