

1.什么是Git

Git 是一个开源的分布式版本控制系统，用于敏捷高效地处理任何或小或大的项目。

Git 是 Linus Torvalds 为了帮助管理 Linux 内核开发而开发的一个开放源码的版本控制软件。

Git 与常用的版本控制工具 CVS, Subversion 等不同，它采用了分布式版本库的方式，不必服务器端软件支持。

2. 为什么学习 Git

李纳斯憎恶分明，经常口不择言，比如他对 C++的评价是：C++是一门糟糕的语言。而且有一群不合格的程序员在使用C++，他们让它变得更糟糕了。他对自己的两个产品命名的解释是：我是个自大的混蛋，我所有的项目都以我的名字来命名。开始是Linux，然后是Git（英国俚语，饭桶的意思）。第一是生存，第二是社会秩序，第三是娱乐。**以上就是Linus所谓的人生意义，**Linux之父的自传《Just For Fun》

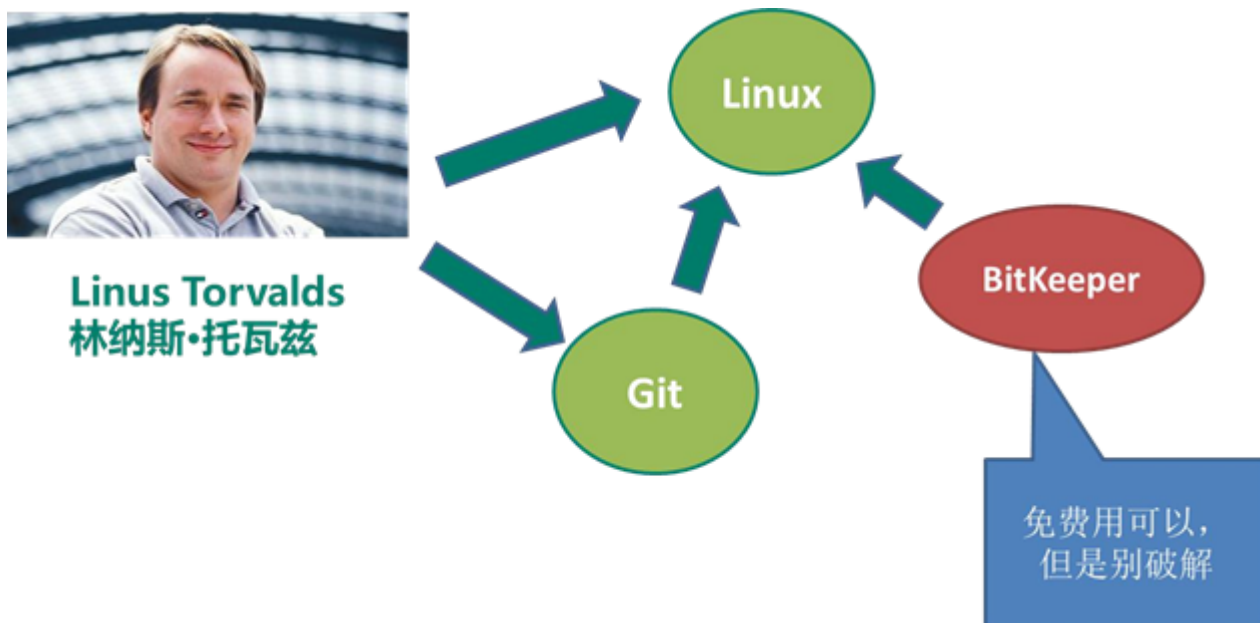
不过我最喜欢李纳斯说过的一句话是：Talk is cheap, Show me the code。他一直用自己的编程人生诠释着这句话。2006年的时候，Linux 内核代码的2%依然是李纳斯完成的，他是代码贡献最多的人之一（是年37岁）。到了2012年，他对内核的贡献主要是合并代码，编程变少了，但是他依然对是否将新代码并入到 Linux 内核具有最终决定权。

2.1 必点天赋

现在各个公司项目开发，团队协作已经从 svn 逐渐向 git 转化，git 是 开发人员的必备技术能力。

2.2 必然趋势

各大公司的开源项目均在 github 上发布，git 和 github 已经是项目产品版本控制的必然选择，市场占有率也会越来越大。



当大家使用 Google 搜索时，使用 Kindle 阅读时，使用淘宝购物时，使用 QQ 聊天时，很多人并不知道，支撑这些软件和服务的，是后台成千上万台 Linux 服务器，它们时时刻刻都在进行着忙碌的运算和数据处理，确保数据信息在人、软件和硬件之间安全的流动。可以这么说，世界上大部分软件和服务都运行在 Linux 操作系统之上，神马云计算、大数据、移动互联网，说起来风起云涌，其实没有 Linux 全得趴窝（微软除外）。Linux主要应用在企业端，由专业Linux运维工程师使用，我们所使用很便捷的各种互联网应用、娱乐、支付、聊天的背后，看似简单。越是简单

的应用，背后有着极其复杂的数据请求和响应。在腾讯、新浪、百度、苹果互联网公司的服务器机房里，至少千万台Linux服务器，去处理众多用户的请求。由于Linux的开源、稳定、安全性、开发灵活性，同时因为Windows系统的自身缺陷，也奠定了Linux操作系统在服务端的位置。

那么的作者呢是一个叫做的这样一个人啊，这个人呢是一个大神级的一个人。包括具有一些典型的一些特质。比如说非常厉害，非常聪明，是吧？

创作力非常强啊，然后呢性格呢是比较自我啊，不善于交际，喜欢独处啊这些啊。那么在这个tag中有一集关于他的一个访谈啊，大家一定要去看一下。

基本上可以了解他是一个怎样的一个人啊，以及呢了解一下他的利害之处。

那么他呢，是在一九九一年的时候，也就是说在他在上大学期间创作了一个操作系统的内核啊。

啊，那么他把它叫做linux kernel啊，大家一看看到啊他自己的本身的名字叫做linux。那么这个linux呢他就取了他的名字，后面改了一个字母。

它叫做linux可能可能就是内核的意思啊，内核呢就是操作系统的一个部分是应用程序跟硬件之间的一个抽象层，啊，可以认为是操作系统中最重要的一部分。

那么完成这个内核之后的话呢，出于一种成就感啊，或者说是一种炫耀啊，他就把它放在大学的服务器上，把代码的公开出来。

啊，让别的人去看啊，去浏览，去讨论。那么啊其实这种心理也很正常啊，我们现在呢每天都在做啊，发朋友圈求点赞。双击六六六。啊。那么确实呢由于他的技术很牛啊，代码开放之后呢，得到了很多大量的关注，很多人呢仔细阅读了他的代码啊，并且呢提出了一些更多的一些想法跟建议。然后呢，他就觉得这样也不错啊，在他的朋友的建议下啊，干脆呢就将linux内核作为一个开源项目来运作，采用协作的方式。继续开发下去。于是呢linux内核呢就成为了一个开源协作的社区项目。多人参与开发之后呢，他的发展呢就非常迅速啊，影响力呢也急剧的扩大啊，没有用多久他就遇到了一个非常重要的一个结合，也是他的一个转折点。啊，这个结合呢来自于自由软件运动的一个核心项目，叫做glue操作系统。啊，glue呢是在一九八三年启动的啊，经过了多年的发展呢。基本上已经万事俱备啊，只欠一个内核啊，只欠一个高性能的内核。那么于是呢linux内核呢作为一个非常杰出的一套内核系统啊。被他采纳了啊，变成了啊变成了一个完整的两者一结合呢就变成一个完整的一个操作系统啊，就被称为glue linux操作系统。大家可能不知道知道，Linus在1991年创建了开源的Linux，从此，Linux系统不断发展，已经成为最大的服务器系统软件了。Linus虽然创建了Linux，但Linux的壮大是靠全世界热心的志愿者参与的，这么多人在世界各地为Linux编写代码，那Linux的代码是如何管理的呢？事实是，在2002年以前，世界各地的志愿者把源代码文件通过diff的方式发给Linus，然后由Linus本人通过手工方式合并代码！你也许会想，为什么Linus不把Linux代码放到版本控制系统里呢？不是有CVS、SVN这些免费的版本控制系统吗？因为Linus坚定地反对CVS和SVN，这些集中式的版本控制系统不但速度慢，而且必须联网才能使用。有一些商用的版本控制系统，虽然比CVS、SVN好用，但那是付费的，和Linux的开源精神不符。不过，到了2002年，Linux系统已经发展了十年了，代码库之大让Linus很难继续通过手工方式管理了，社区的弟兄们也对这种方式表达了强烈不满，于是Linus选择了一个商业的版本控制系统BitKeeper，BitKeeper的东家BitMover公司出于人道主义精神，授权Linux社区免费使用这个版本控制系统。安定团结的大好局面在2005年就被打破了，原因是Linux社区牛人聚集，不免沾染了一些梁山好汉的江湖习气。开发Samba的Andrew试图破解BitKeeper的协议（这么干的其实也不只他一个），被BitMover公司发现了（监控工作做得不错！），于是BitMover公司怒了，要收回Linux社区的免费使用权。Linus可以向BitMover公司道个歉，保证以后严格管教弟兄们，嗯，这是不可能的。实际情况是这样的：Linus花了两周时间自己用C写了一个分布式版本控制系统，这就是Git！一个月之内，Linux系统的源码已经由Git管理了！牛是怎么定义的呢？大家可以体会一下。Git迅速成为最流行的分布式版本控制系统，尤其是2008年，GitHub网站上线了，它为开源项目免费提供Git存储，无数开源项目开始迁移至GitHub，包括jQuery，PHP，Ruby等等。历史就是这么偶然，如果不是当年BitMover公司威胁Linux社区，可能现在我们就没有免费而超级好用的Git了。Linux是一个全面、丰富多彩的生态圈，主流的IT技术都是各路大牛基于linux环境开发。

- 数据库 MySQL、PostgreSQL
- Web Server Nginx
- 大数据 Hadoop、Spark
- 消息队列 kafka
- 虚拟化技术 kvm
- 容器 Docker、Kubernetes

运维领域涉及专业领域较宽、技能知识可以深度挖掘：

- 系统运维
- 网络运维
- 安全运维

- 数据库运维
- 大数据运维
- 运维开发

2.3能干吗？

各项功能与 svn 有重合。但因为 git 秉承开源原则，权限管理没有 svn 限制的灵活严格。Git 的分支管理、代码审查是其重要功能，也是其特色。



3.集中式版本管理

经典产品：CVS、VSS、SVN

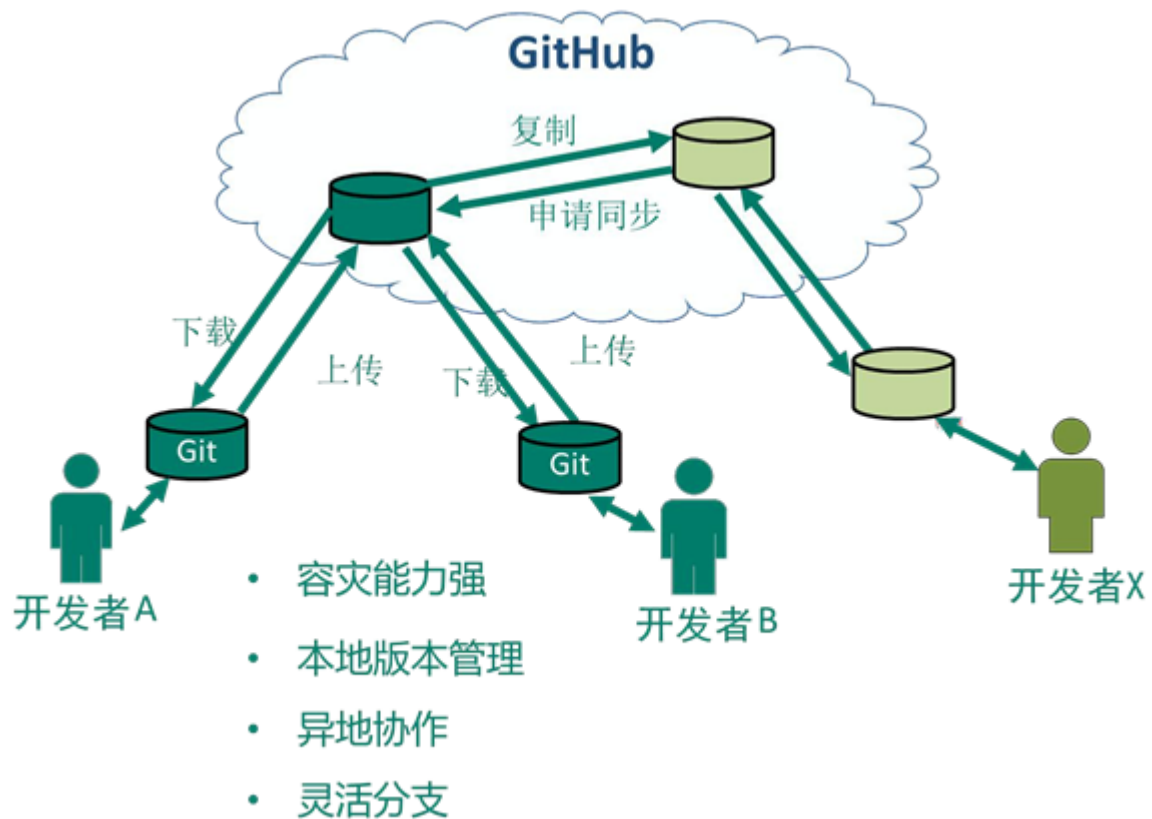
特点：由中央仓库统一管理，结构简单，上手容易。先说集中式版本控制系统，版本库是集中存放在中央服务器的，而干活的时候，用的都是自己的电脑，所以要先从中央服务器取得最新的版本，然后开始干活，干完活了，再把自己的活推送给中央服务器。中央服务器就好比是一个图书馆，你要改一本书，必须先从图书馆借出来，然后回到家自己改，改完了，再放回图书馆。集中式版本控制系统最大的毛病就是必须联网才能工作，如果在局域网内还好，带宽够大，速度够快，可如果在互联网上，遇到网速慢的话，可能提交一个10M的文件就需要5分钟，这还不得把人给憋死啊。

不足：

- 版本管理的服务器一旦崩溃，硬盘损坏，代码如何恢复？
- 程序员上传到服务器的代码要求是完整版本，但是程序员开发过程中想做小版本的管理，以便追溯查询，怎么破？
- 系统正在上线运行，时不时还要修改 bug，要增加好几个功能要几个月，如何管理几个版本？
- 如何管理一个分布在世界各地、互不相识的大型开发团队？

4. 分布式版本管理

分布式版本管理工具很好的解决了集中式管理的不足，也正是它的功能特色，使其成为主流的版本管理工具。



那分布式版本控制系统与集中式版本控制系统有何不同呢？首先，分布式版本控制系统根本没有“中央服务器”，每个人的电脑上都是一个完整的版本库，这样，你工作的时候，就不需要联网了，因为版本库就在你自己的电脑上。既然每个人电脑上都有一个完整的版本库，那多个人如何协作呢？比方说你在自己电脑上改了文件A，你的同事也在他的电脑上改了文件A，这时，你们俩之间只需把各自的修改推送给对方，就可以互相看到对方的修改了。

和集中式版本控制系统相比，分布式版本控制系统的安全性要高很多，因为每个人电脑里都有完整的版本库，某一个人的电脑坏掉了不要紧，随便从其他人那里复制一个就可以了。而集中式版本控制系统的中央服务器要是出了问题，所有人都没法干活了。

在实际使用分布式版本控制系统的时候，其实很少在两人之间的电脑上推送版本库的修改，因为可能你们俩不在一个局域网内，两台电脑互相访问不了，也可能今天你的同事病了，他的电脑压根没有开机。因此，分布式版本控制系统通常也有一台充当“中央服务器”的电脑，但这个服务器的作用仅仅是用来方便“交换”大家的修改，没有它大家也一样干活，只是交换修改不方便而已。

分支就是科幻电影里面的平行宇宙，当你正在电脑前努力学习Git的时候，另一个你正在另一个平行宇宙里努力学习SVN。

如果两个平行宇宙互不干扰，那对现在的你也没啥影响。不过，在某个时间点，两个平行宇宙合并了，结果，你既学会了Git又学会了SVN！



分支在实际中有什么用呢？假设你准备开发一个新功能，但是需要两周才能完成，第一周你写了50%的代码，如果立刻提交，由于代码还没写完，不完整的代码库会导致别人不能干活了。如果等代码全部写完再一次提交，又存在丢失每天进度的巨大风险。

现在有了分支，就不用怕了。你创建了一个属于你自己的分支，别人看不到，还继续在原来的分支上正常工作，而你在自己的分支上干活，想提交就提交，直到开发完毕后，再一次性合并到原来的分支上，这样，既安全，又不影响别人工作。

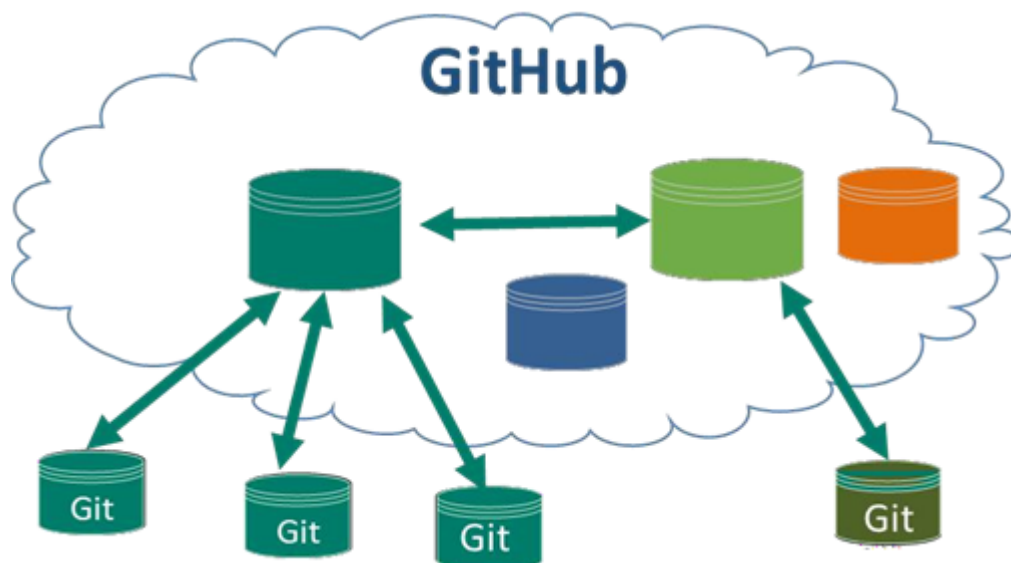
其他版本控制系统如SVN等都有分支管理，但是用过之后你会发现，这些版本控制系统创建和切换分支比蜗牛还慢，简直让人无法忍受，结果分支功能成了摆设，大家都不去用。

但Git的分支是与众不同的，无论创建、切换和删除分支，Git在1秒钟之内就能完成！无论你的版本库是1个文件还是1万个文件。

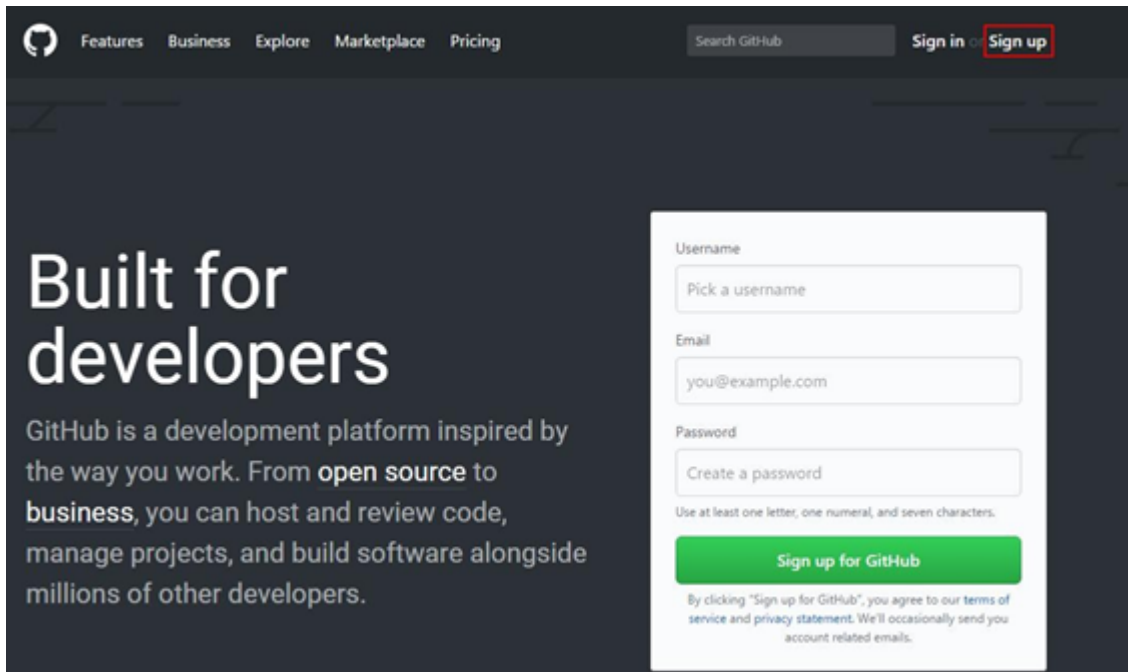
5. GitHub

1.简介

GitHub 是一个 Git 项目托管网站,主要提供基于 Git 的版本托管服务。





2. 注册




Join GitHub

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

 **Step 1:**
Create personal account

 **Step 2:**
Choose your plan

 **Step 3:**
Tailor your experience

Create your personal account

Username

This will be your username. You can add the name of your organization later.

Email address

We'll occasionally send updates about your account to this inbox. We'll never share your email address with anyone.

Password

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

By clicking "Create an account" below, you agree to our [terms of service](#) and [privacy statement](#). We'll occasionally send you account related emails.

[Create an account](#)

You'll love GitHub

- Unlimited collaborators
- Unlimited public repositories
- ✓ Great communication
- ✓ Frictionless development
- ✓ Open source community

3.简单操作

第一步：本地库联通 GitHub

①查看本地是否配置了密钥

只需要在第一次配置就可以了，如果存在就不需要额外生成密钥

②生成密钥

创建SSH Key。在用户主目录下，看看有没有.ssh目录，如果有，再看看这个目录下有没有 `id_rsa` 和 `id_rsa.pub` 这两个文件，如果已经有了，可直接跳到下一步。如果没有，打开Shell（Windows下打开Git Bash），创建SSH Key：

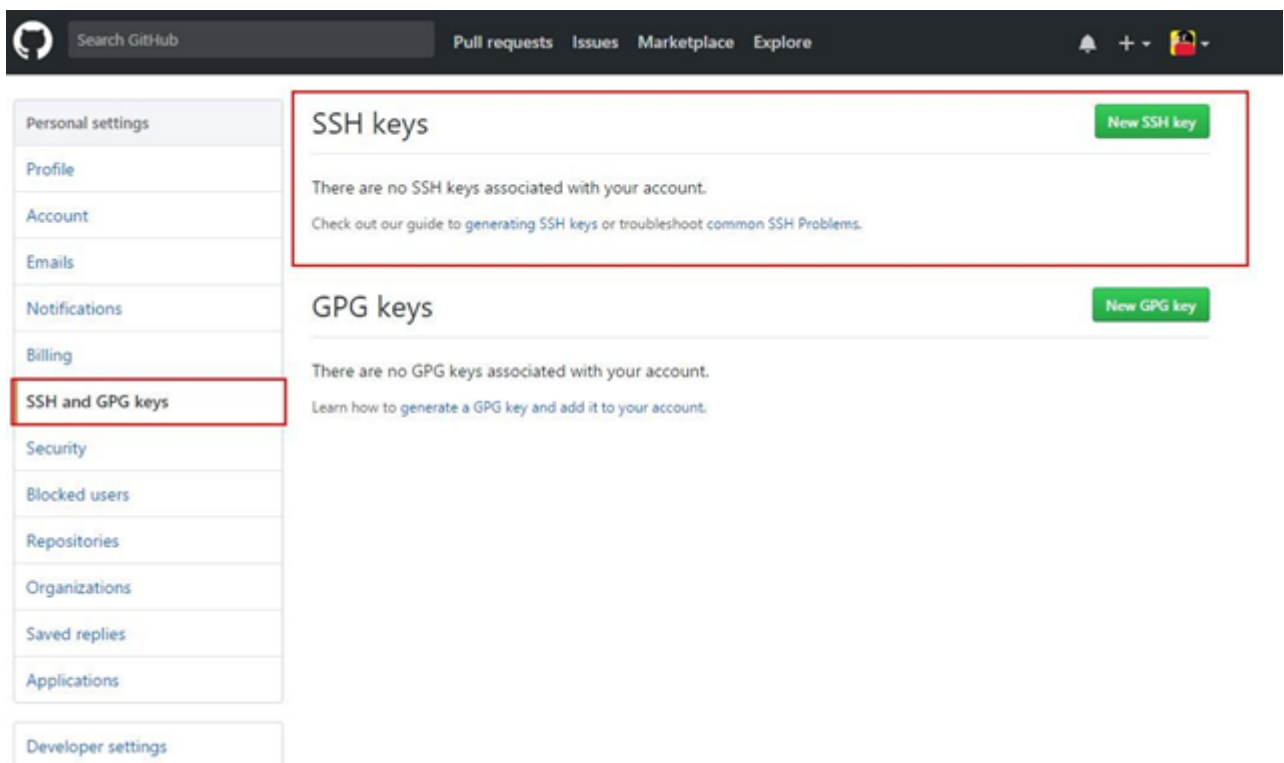
命令：`$ ssh-keygen -t rsa -C "youremail@example.com"`

查看用户名，需要把邮件地址换成你自己的邮件地址，然后一路回车，使用默认值即可，由于这个Key也不是用于军事目的，所以也无需设置密码。如果一切顺利的话，可以在用户主目录打开 `C:\Users\thinkpad.ssh`，可以看到 `id_rsa.pub` 这个文件里找到 `.ssh` 目录，里面有 `id_rsa` 和 `id_rsa.pub` 两个文件，这两个就是SSH Key的秘钥对，`id_rsa` 是私钥，不能泄露出去，`id_rsa.pub` 是公钥，可以放心地告诉任何人。

生成的密钥分为私钥和公钥，会保存在用户家目录的.ssh 文件夹中。

第2步：登陆GitHub，打开“Account settings”，“SSH Keys” 页面：

然后，点“Add SSH Key”，填上任意Title，在Key文本框里粘贴 `id_rsa.pub` 文件的内容：



为什么GitHub需要SSH Key呢？因为GitHub需要识别出你推送的提交确实是你推送的，而不是别人冒充的，而Git支持SSH协议，所以，GitHub只要知道了你的公钥，就可以确认只有你自己才能推送。

第三步：在 GitHub 上创建一个仓库

如果github上没有仓库，需要创建对应的仓库，然后把本地的文件上传到github该仓库下。

1. 登录github后，在已有仓库下，点击右上角“New”，如下图
2. 在新的窗口中，输入仓库名称，勾选新建的仓库中会有一个README，最后点击“Create repository”，如下图
3. 这样就创建好了一个仓库。

第四步：push

本地库推送到 GitHub

1) 准备本地库

本地文件

在本地已经创建一个test_git文件夹，里面有一个文件夹，需要上传picture

注意：如果当前目录下有多个文件夹，可以一起上传

2) 通过git上传文件

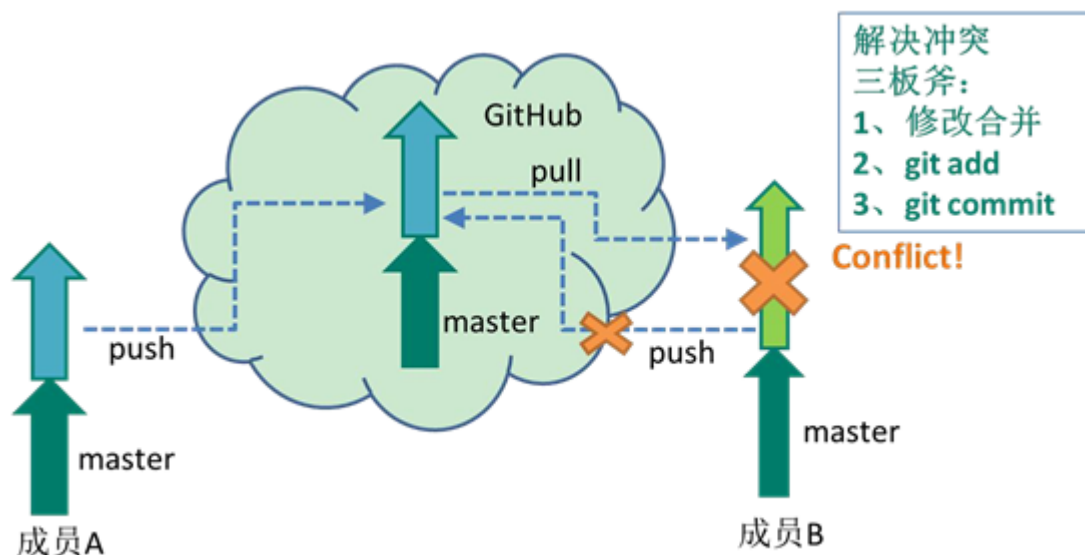
首先进入你的master文件夹下，右键打开“Git Bash Here”，弹出命令窗口，输入如下语句进行上传

1. git init：在此文件夹生成一个.git隐藏文件；
2. git add .：将文件添加到缓存区(注意这个"."，是有空格的，"."代表这个test这个文件夹下的目录全部都提交，也可以通过git add 文件名 提交指定的文件)；
3. git status：查看现在的状态，也可以不看，随你啦，可以看到picture文件夹里面的内容都提交上去了；
4. git commit -m "这里是注释"：提交添加到缓存区的文件
5. git remote add origin <https://xxx@xxx/test.git>：添加新的git方式的origin, github上创建好的仓库和本地仓库进行关联
6. git push origin master：把本地库的所有内容推送到远程仓库（github）上，即上传本地文件，如果显示下图，则说明上传成功
7. 可能遇到的报错
8. 如果报上图的错误，则需要输入 git pull origin master命令；
9. 其实这个问题是因为 两个 根本不相干的 git 库，一个是本地库（只有picture），一个是远端库(test仓库里只有readme文件)，然后本地要去推送到远端，远端觉得这个本地库跟自己不相干，所以告知无法合并，解决方法：先把两部分内容合并以下，有两种方式
10.
 - 方式一：输入“`git pull --rebase origin master`”，然后输入git push origin master语句，即可
 - 方式二：输入“`git pull origin master --allow-unrelated-histories`”（会弹到文件里面，输入“:wq”退出该文件，如果没有遇到，请忽略），然后输入git push origin master语句，即可

刷新github，则可以发现，上传成功，如下图

我们第一次推送 master 分支时，加上了-u 参数，Git 不但会把本地的 master 分支内容推送的远程新的 master 分支，还会把本地的 master 分支和远程的 master 分支关联起来，在以后的推送或者拉取时就可以简化命令。

4 解决冲突



4.1 可能遇到问题

如果遇到报错fatal: repository 'xxx.git/' not found，即没找到'xxx.git'。

原因：repository地址被更改了。比如现在要更改repository名称：由原来的“https://XXX@XXX/a.git” 改为现在的“https://xxx@xxx/b.git”

解决方法：

方式1：

```
git remote set-url origin https://xxx@xxx/B.git # 设置远程url为修改后的地址
git remote -v # 查看remote链接
```

方式2：

```
git remote rm origin # 移出旧的http的origin
git remote add origin https://xxx@xxx/B.git # 添加新的git方式的origin
git remote -v # 查看remote链接
```

4.2更新仓库文件

如果对本地文件有了修改，则需要对仓库文件进行更新，比如本地文件中删除了一个文件。下面将显示如何对仓库更新。

首先进入你的master文件夹下，右键打开“Git Bash Here”，弹出命令窗口，输入如下内容

输入 以下文本即可更新仓库

git status

git add -A

git commit -a -m "update"：能提交修改过，但是没有添加到缓存区的文件（修改过的就能提交）

git push origin master -f

4.3删除github中的某个仓库

1. 登录github
2. 查看列表的仓库，找到需要删除的仓库，进入该仓库
3. 点击上图中的Settings
4. 进入Settings的选项之后，滑到最下面，点击“Delete this repository”

4.4删除github中的某个文件(夹)

github上只能删除仓库，但是，却无法删除文件或文件夹，所以只能通过命令来解决。

首先进入你的master文件夹下，右键打开“Git Bash Here”，弹出命令窗口，输入如下内容

```
git pull origin master # 将远程仓库里面的项目拉下来
dir # 查看有哪些文件夹
git rm -r --cached picture # 删除picture文件夹
git commit -m "删除了picture文件夹" # 提交,添加操作说明
git push origin master # 将本次更改更新到github项目上去
```

4.5提交代码报错

在使用git提交代码时，出现 fatal: Could not read from remote repository 这个错误，解决办法:

1. 打开C:\Users\thinkpad.ssh，将id_rsa以及id_rsa.pub这两个文件删除掉；
2. 使用语句 `ssh-keygen -t rsa -C "email"`，重新生成密钥，即上面第3部分（设置SSH key）第2步
3. 在github setting里添加SSH keys，具体见上面第3部分（设置SSH key）第4步；
4. SSH keys设置成功后，重新运行提交代码，即可成功