# https://gss0.bdstatic.com/94o3dSag_xI4khGkpoWK1HF6hhy/baike/crop%3D19%2C0%2C607%2C400%3Bc0%3Dbaike80%2C5%2C5%2C80%2C26/sign=1cd6e3d3c4bf6c81e37876a8810f8819/241f95cad1c8a78649434db56f09c93d70cf501b.jpg

## Git 介绍

### Git的 诞生

很多人都知道，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。

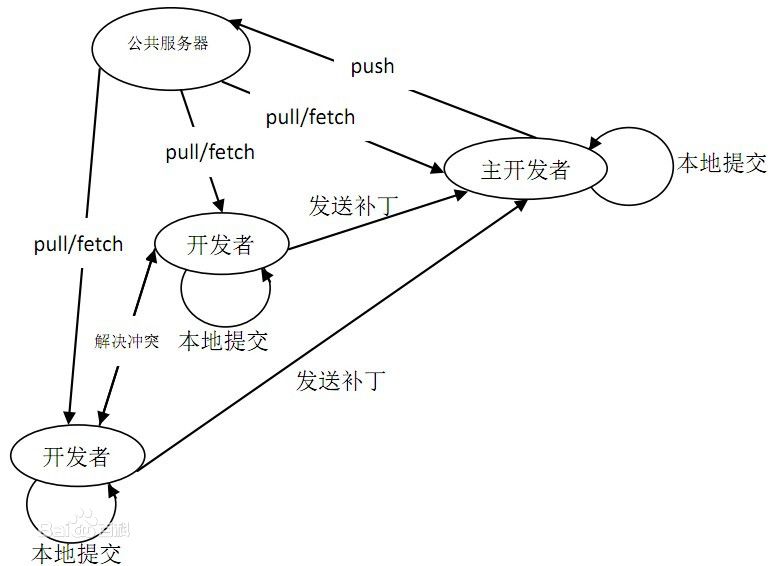
Git迅速成为最流行的分布式版本控制系统，尤其是2008年，GitHub网站上线了，它为开源项目免费提供Git存储，无数开源项目开始迁移至GitHub，包括jQuery，PHP，Ruby等等。

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。

### 集中式与分布式版本控制系统的区别

开发者之间的合作方式是**共用一个仓库**，无论这个仓库是在本地还是在远端，只要是所有成员都共同存取同一个仓库，那么这种方式就是集中式版本控制。分散式版本控制系统和集中式版本控制系统，其最大差别在于前者的仓库可以有多份。事实上，每个开发者都可以在自己的一台或者多台电脑上建立档案库

### Git控制图解



## 安装Git

最早Git是在Linux上开发的，很长一段时间内，Git也只能在Linux和Unix系统上跑。不过，慢慢地有人把它移植到了Windows上。现在，Git可以在Linux、Unix、Mac和Windows这几大平台上正常运行了。

### 在Linux上安装Git

直接命令安装：

sudo apt-get install git

sudo apt-get install git-core（针对老一些的系统）

或下载源码包安装：

先从Git官网下载源码，然后解压，依次输入：./config，make，sudo make install

### 在Windows上安装Git

从Git官网直接下载安装程序，（网速慢的移步国内镜像），然后按默认选项安装即可

## Git的使用

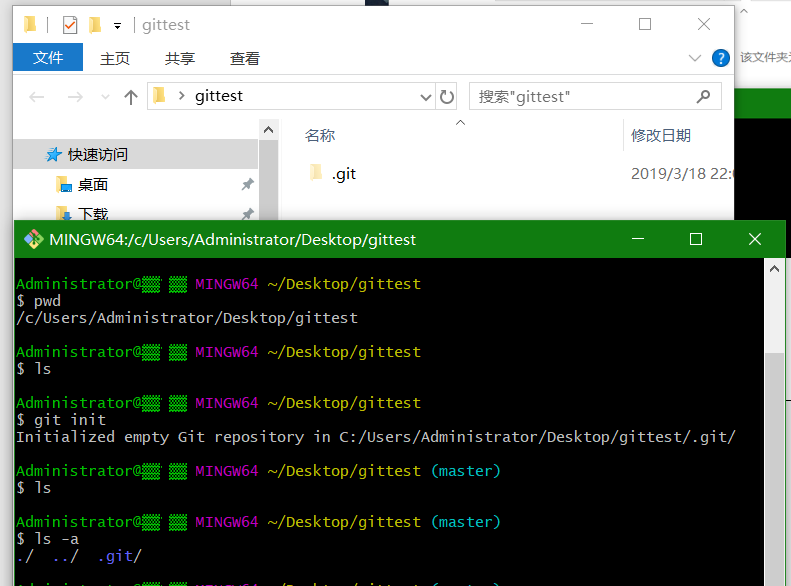
### 创建版本库

什么是版本库呢？版本库又名仓库，英文名repository，可以简单理解成项目目录.

【所有的版本控制系统，其实只能跟踪文本文件的改动，比如TXT文件，网页，所有的程序代码等等，Git也不例外。版本控制系统可以告诉你每次的改动，比如在第5行加了一个单词“Linux”，在第8行删了一个单词“Windows”。

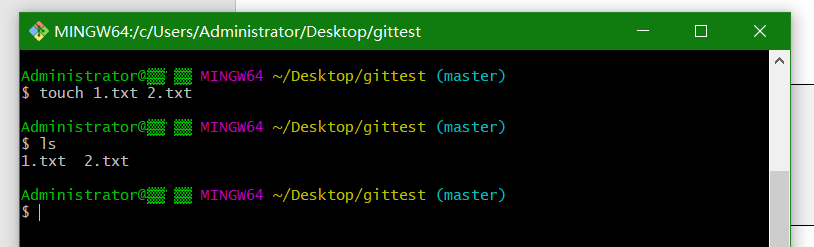
而图片、视频、word这些二进制文件，虽然也能由版本控制系统管理，但没法跟踪文件的变化，只能把二进制文件每次改动串起来，也就是只知道图片从100KB改成了120KB，但到底改了啥，版本控制系统不知道，也没法知道。】

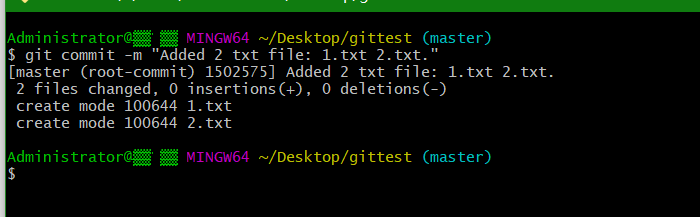
|  |
| --- |
| 1. 新建目录   mkdir dirname)  2.在你的项目文件目录下运行命令：  (cd dirname)  git init “把当前目录(pwd)变成Git可以管理的仓库 |



### 添加文件

|  |
| --- |
| 新建文件 1.txt 2.txt编辑完成后保存  把文件添加到版本控制仓库（[暂存区](#_工作区和暂存区)）  git add 1.txt 2.txt  把（[暂存区](#_工作区和暂存区)）文件提交到版本控制仓库  git commit -m “commit instructions(提交说明)”  成功后会返回更新信息 |

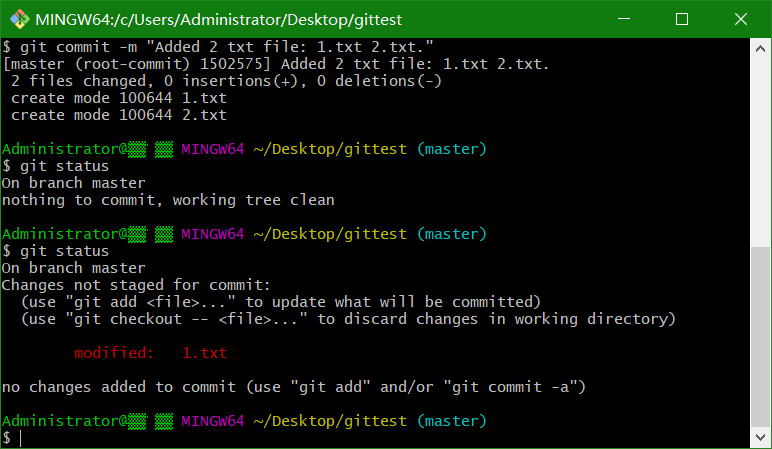




### 查看状态

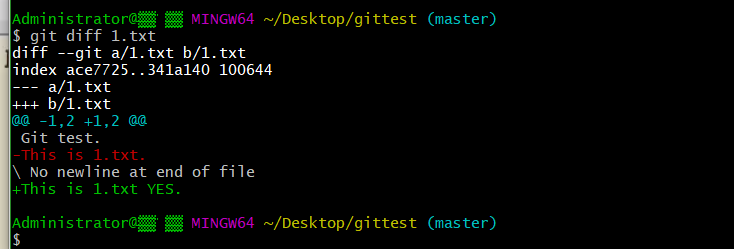
|  |
| --- |
| git status |

若存在文件被改动，但还未提交，则会提示：

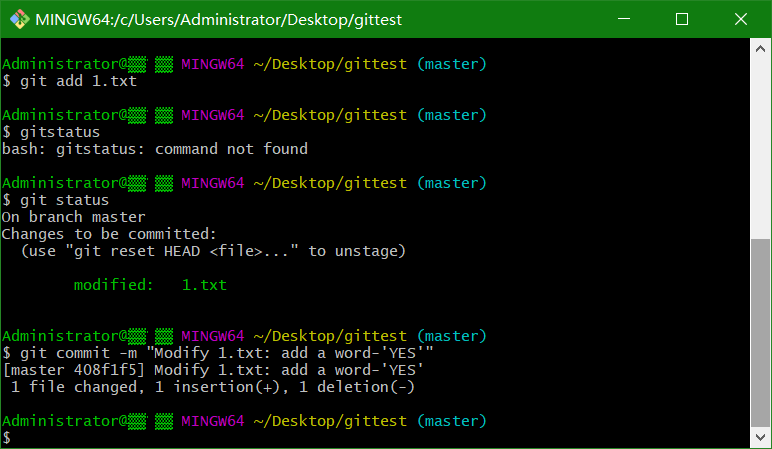


查看具体修改的内容(difference)：

|  |
| --- |
| git diff 1.txt |

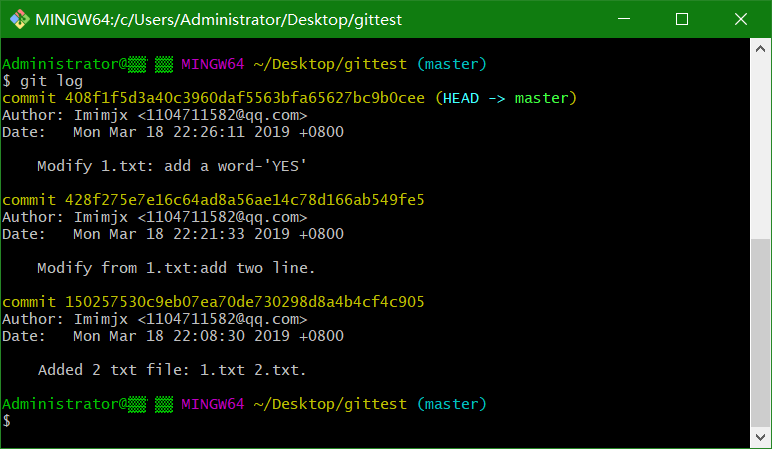


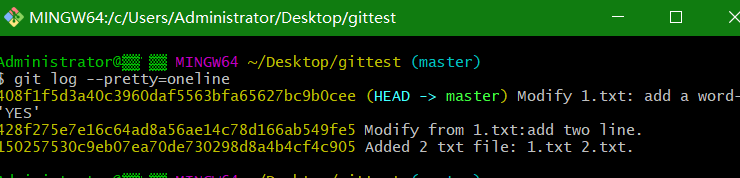
### 提交修改



### 查看日志

|  |
| --- |
| git log [--pretty=oneline] |





(前面一串为commit id（版本号）(由SHA1计算出来))

### 回退提交

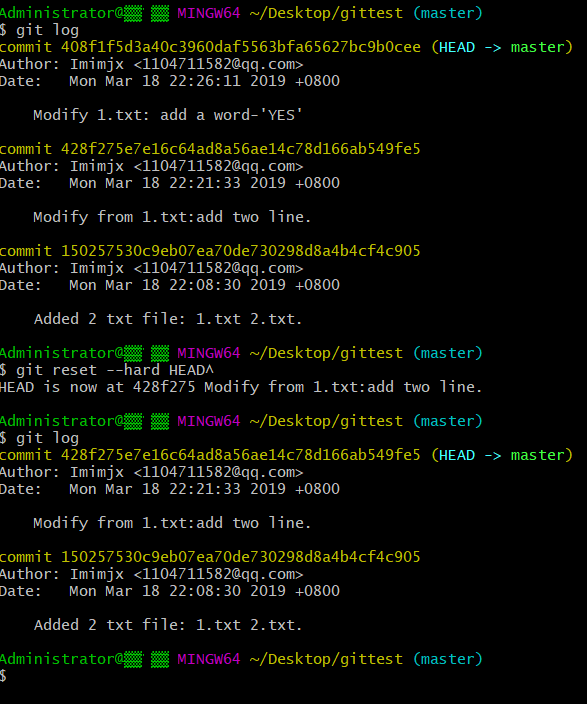
首先，Git必须知道当前版本是哪个版本，在Git中，用HEAD表示当前版本，上一个版本就是HEAD^，上上一个版本就是HEAD^^，当然往上100个版本写100个^比较容易数不过来，所以写成HEAD~100.

git reset -- hard HEAD^

git reset –soft 不会改变stage区，仅仅将commit回退到了指定的提交

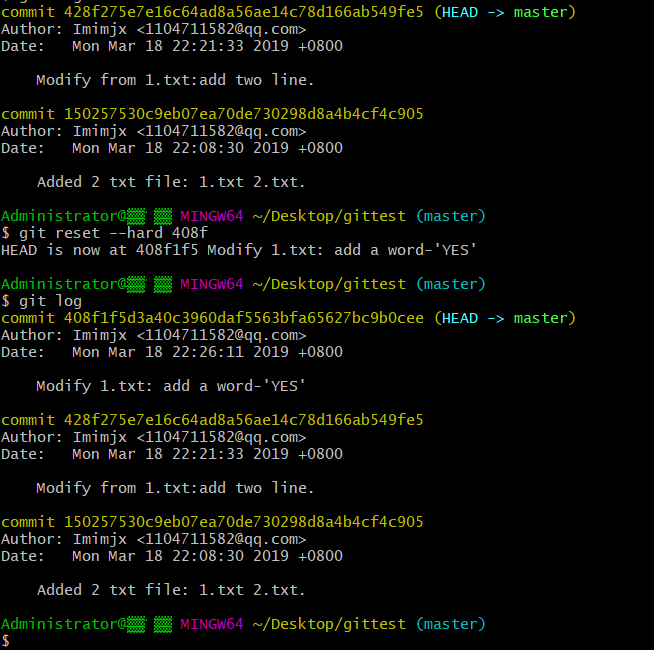
git reset –mixed 不回改变工作区，但是会用指定的commit覆盖stage 区，之前所有暂存的内容都变为为暂存的状态

git reset –hard 使用指定的commit的内容覆盖stage区和工作区。



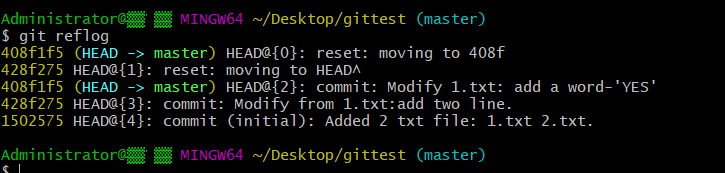
这时就回到了第二个状态，但还能回到第三个状态吗？ - 指定commit id

Git reset --hard 408f



忘记回退之前的commit id怎么办?

git reflog

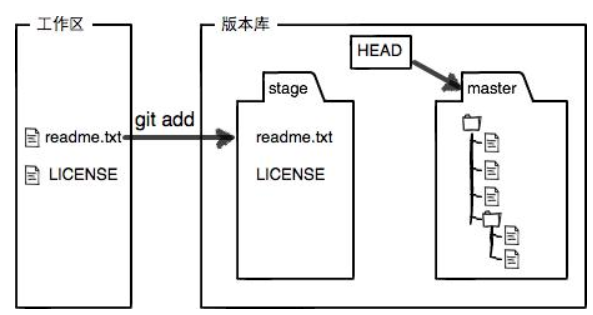


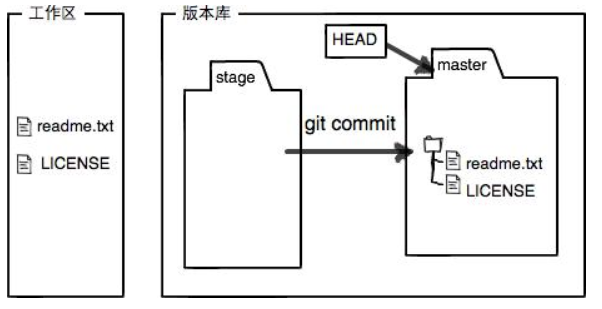
(版本控制的作用显示出来了，你可以得到多个时期的文件)

### 工作区和暂存区

工作区（Working Directory）就是你在电脑里能看到的目录，你工作的目录。

工作区有一个隐藏目录.git，这个不算工作区，而是Git的版本库，其中最重要的就是称为stage（或者叫index）的暂存区，还有Git为我们自动创建的第一个分支master，以及指向master的一个指针叫HEAD。





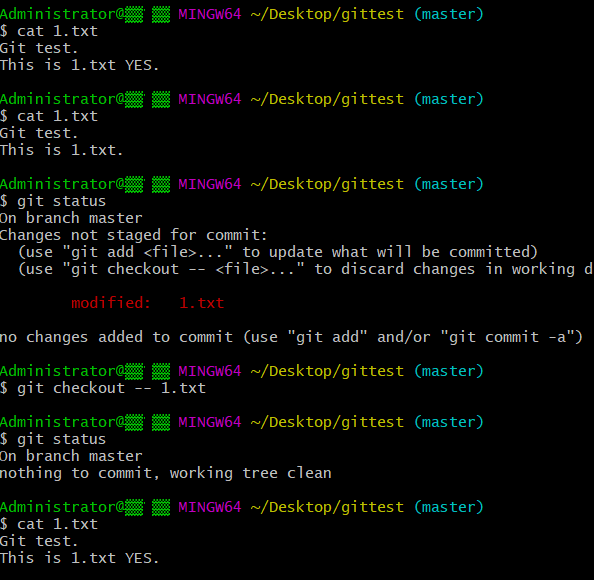
#### 修改/撤销修改

第一次修改 -> git add -> 第二次修改 -> git commit 🡺第二次修改不会被提交

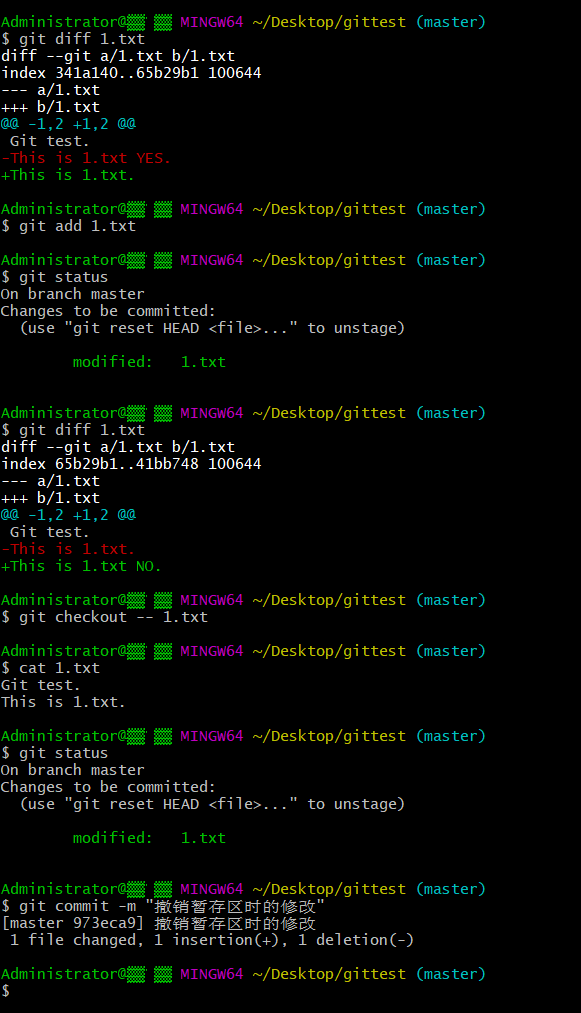
Git管理的是修改，当你用git add命令后，在工作区的第一次修改被放入暂存区，准备提交，但是，在工作区的第二次修改并没有放入暂存区，所以，git commit只负责把暂存区的修改提交了，也就是第一次的修改被提交了，第二次的修改不会被提交。

命令git checkout -- 1.txt意思就是，把1.txt文件在工作区的修改全部撤销，这里有两种情况：

一种是1.txt自修改后还没有被放到暂存区，现在，撤销修改就回到和版本库一模一样的状态：



一种是1.txt已经添加到暂存区后，又作了修改，现在，撤销修改就回到添加到暂存区后的状态：



总之，就是让这个文件回到最近一次git commit或git add时的状态。

#### 删除/撤销删除

删除文件：

rm 1.txt

git rm 1.txt

git commit -m “deleted 1.txt”

撤销删除：

网上资料显示撤销删除与撤销修改一样，也是使用命令git checkout – filename

但是，我测试时该命令只能撤销尚未git rm filename以及git commit时的状况，不知道为何，我只能使用git reset --hard HEAD^ 进行版本回退了。。。

## 远程仓库的使用

### 配置Git用户信息

git config --global –list

配置你的ID(GitHub账户)

|  |
| --- |
| $ git config --global user.name "Your Name"  $ git config --global user.email "email@example.com" |

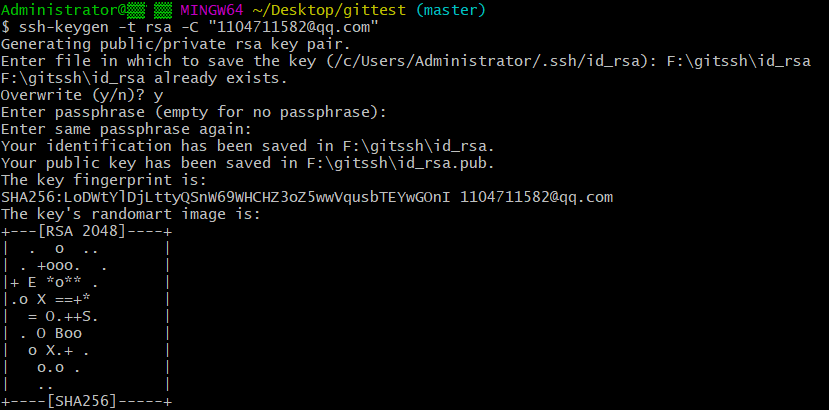
### SSH密匙

密匙生成：

ssh-keygen -t rsa -C "youremail"

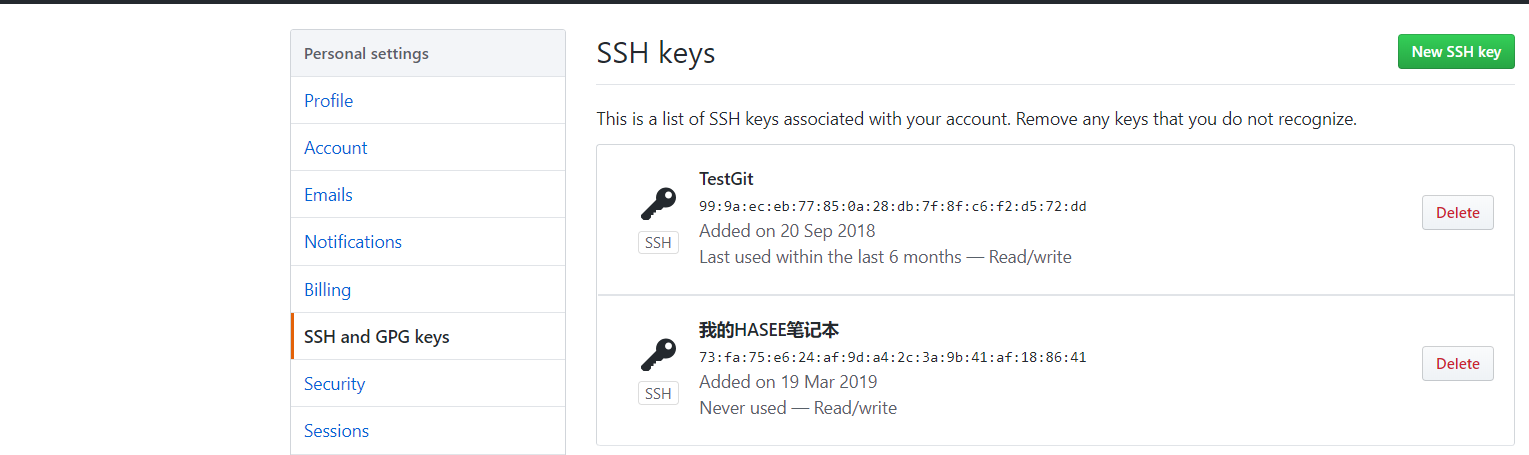
1 保存文件路径

2 密码

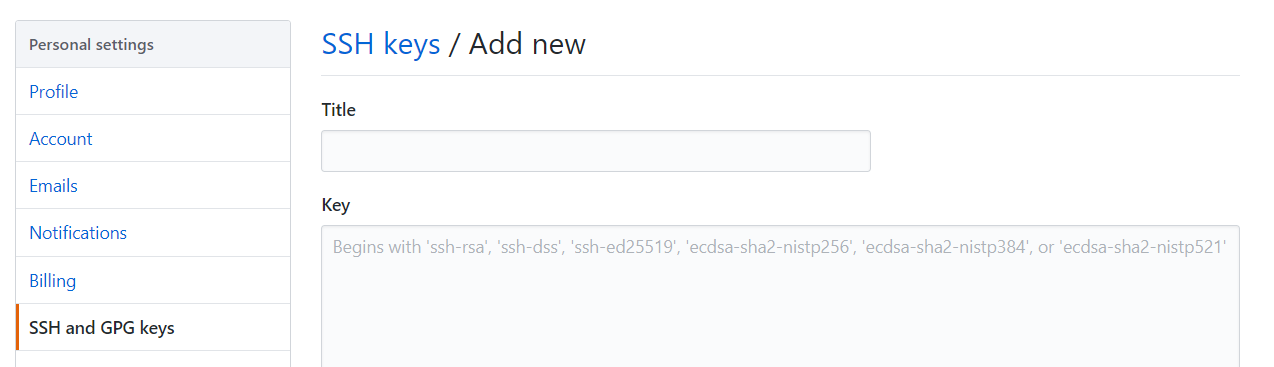




在GitHub上添加该SSH KEY



将id\_rsa.pub中的内容复制到Key下保存即可



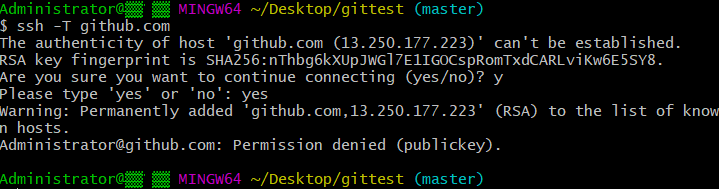
验证

ssh -T [git@github.com](mailto:git@github.com)

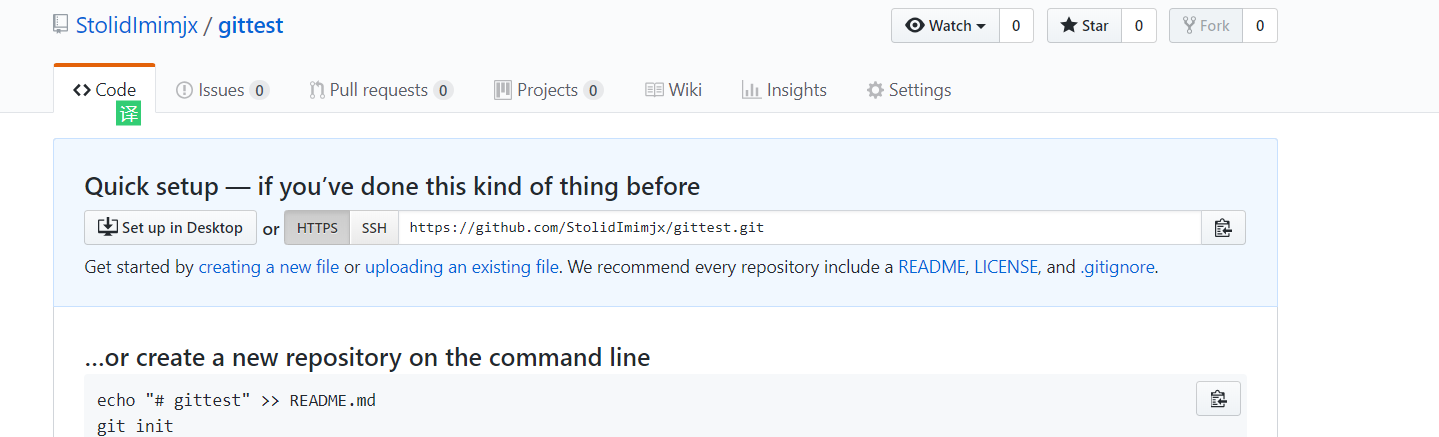
（使用公司代理网络时里出现连接超时错误：



使用自己的热点重试了一下

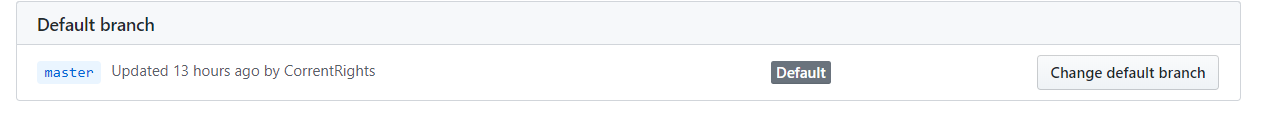
）

### 在GitHub远程仓库中创建分支



git remote add origin git@github.com:(github账户名)/(仓库名).git





### 查看远程仓库中的分支

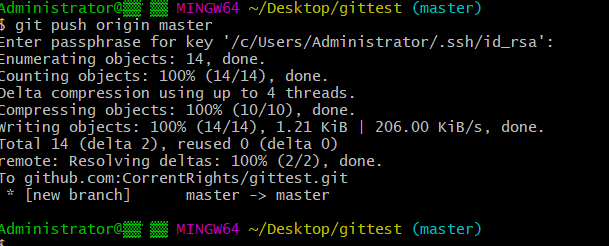
git remote [-v]

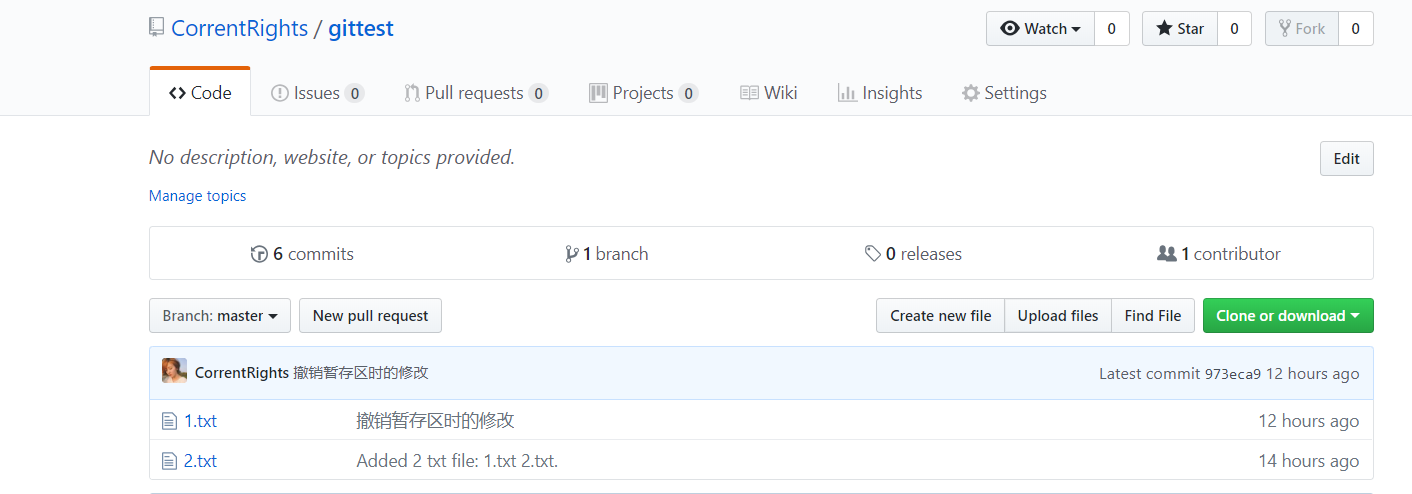
### 删除远程仓库中的分支

git remote rm fetchname

### 推送本地仓库到远程仓库

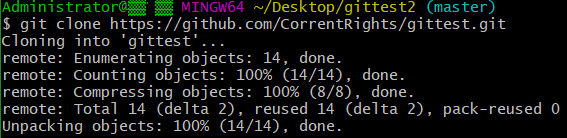
git push origin master



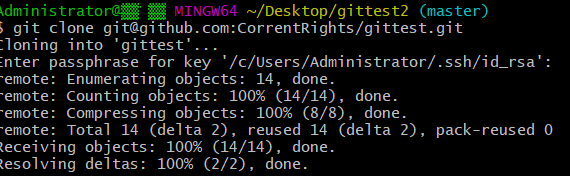


从远程库克隆到本地

git clone 远程库路径



或



GitHub给出的地址不止一个， Git支持多种协议，默认的git://使用ssh，但也可以使用https等其他协议。使用https除了速度慢以外，还有个最大的麻烦是每次推送都必须输入口令，但是在某些只开放http端口的公司内部就无法使用ssh协议而只能用https。

参考资料

1. [廖雪峰Git教程](https://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000)
2. [SSH设置](https://blog.csdn.net/lqlqlq007/article/details/78983879)

访问Git：<https://git-scm.com/>

访问GitHub: <https://github.com>/