### 一、在GitHub新建托管项目

#### 1、注册

1. 1

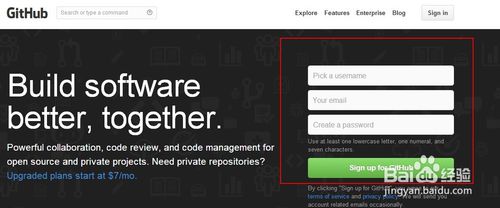
要托管到github，那你就应该要有一个属于你自己的github帐号，所以你应该先到github.com注册

打开浏览器

在地址栏输入地址：github.com

填写用户名、邮箱、密码

点击Sign up即可简单地注册



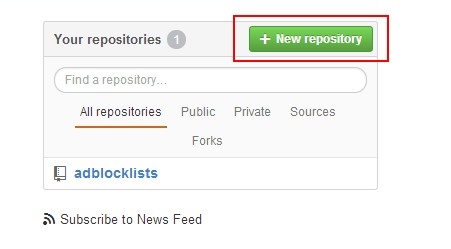
1. 2

完成注册，进入github平台，

#### 2、新建项目

点击new repositories

新建一个新项目（你也可以加入到一个已有的项目）



1. 3

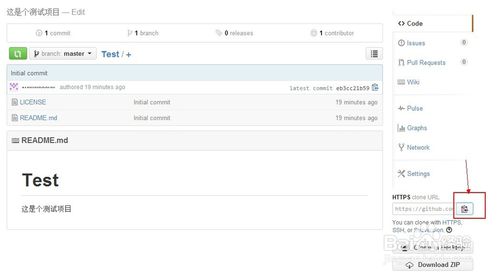
新建项目的操作，其实很简单，输入项目名就可以直接Create了

如图



1. 4

此时的界面，我想大家都应该很熟悉了。点击右边栏的剪切板图标，记录下你的项目地址。



### 二、上传项目至GitHub

#### 1、安装

https://git-for-windows.github.io/

下载并安装最新版本的msysgit

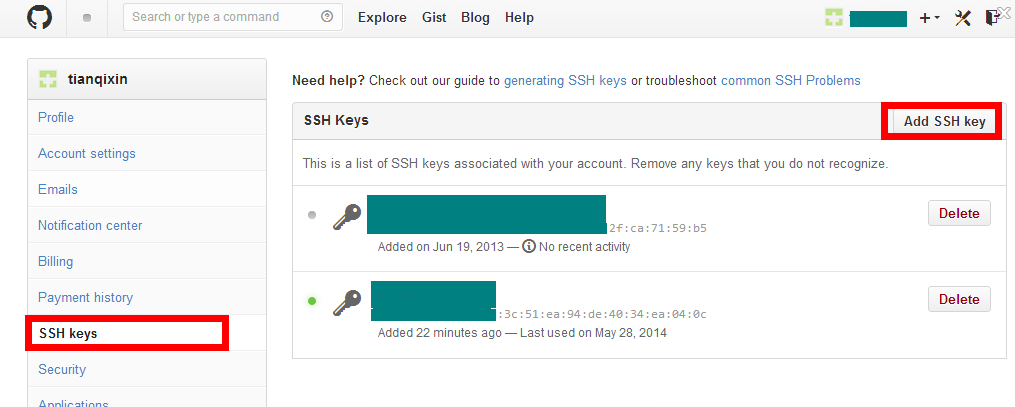
#### 2、设置用户

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"

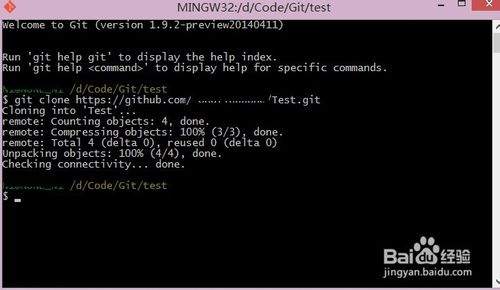
#### 3、下载仓库

启动Git Bash命令行

如下图所示



1. 输入git  clone  接着将先前记录下来的地址复制到后面，回车



1. 将下载下来的项目文件夹的所有文件及文件夹，包括.git文件夹在内，全部拷贝到你的托管项目的根目录（或者将你的托管项目拷贝到该目录）

cd test

进入到该托管项目的根目录。



1. 输入

git add .

将改动的地方添加到版本管理器

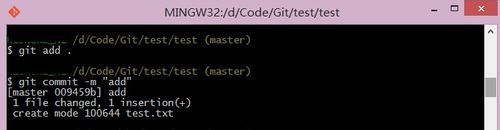


1. 输入

git  commit -m "changes log"

提交到本地的版本控制库里，

引号里面是你对本次提交的说明信息。

[](http://jingyan.baidu.com/album/f7ff0bfc7181492e27bb1360.html?picindex=9)

1. 最后输入

git push -u origin master

将你本地的仓库提交到你的github账号里，

此时会要求你输入你的github的账号和密码。

[](http://jingyan.baidu.com/album/f7ff0bfc7181492e27bb1360.html?picindex=10)

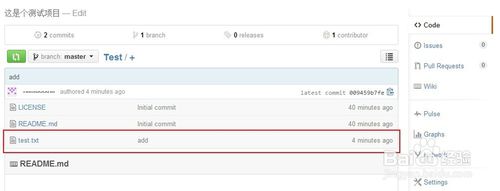
1. 回到你的github.com中的项目页。

里面的文件是不是已经发生改变了？

至此，

github项目已经成功更新。

更多的github的操作，你可以自己慢慢的体会。但有了这第一步认识，我相信你以后的摸索会更加容易得多。

[](http://jingyan.baidu.com/album/f7ff0bfc7181492e27bb1360.html?picindex=11)

END

### 注意事项

* 当你首次使用git的时候，会要求你输入你的用户信息，和你github的一致即可
* 将github中的项目同步更新至你本地目录你需要用到git fetch，这些都是以后进阶的事了
* 在下载或者上传的过程中，建议你不要断开你的网络，要耐心地等待，以免出现些不良的麻烦事
* 具体的git指令，可自行查看帮助（help）

### 分支

1. 下载新代码：git pull。

2. 从master出捡出一个新的分支：git checkout -b feature。feature是分支名称，你可以随意取名，但请用英文。

3. 在feature分支上随意修改，改完后你可以提交你的修改：git commit -m "some message"。

4. 此时要同步代码，请先切回主分支：git checkout master，然后更新git pull。

5. 如果想删除自己建立的分支，使用git branch -D feature，注意执行此命令后分支被强制删除，无法恢复。

## GIT详细教程

### 一、版本管理

#### 1、版本回退

$ git log

$ git log --pretty=oneline

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，也就是最新的提交3628164...882e1e0（注意我的提交ID和你的肯定不一样），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。

现在，我们要把当前版本“append GPL”回退到上一个版本“add distributed”，就可以使用git reset命令：

$ git re**set** --hard HEAD^

HEAD **is** now **at** ea34578 **add** distributed

--hard参数有啥意义？这个后面再讲，现在你先放心使用。

看看readme.txt的内容是不是版本add distributed：

$ cat readme.txt

Git **is** a distributed version control system.

Git **is** free software.

果然。

还可以继续回退到上一个版本wrote a readme file，不过且慢，然我们用git log再看看现在版本库的状态：

$ git log

commit ea34578d5496d7dd233c827ed32a8cd576c5ee85

Author: Michael Liao <askxuefeng@gmail.com>

Date: Tue Aug 20 14:53:12 2013 +0800

add distributed

commit cb926e7ea50ad11b8f9e909c05226233bf755030

Author: Michael Liao <askxuefeng@gmail.com>

Date: Mon Aug 19 17:51:55 2013 +0800

wrote a readme file

最新的那个版本append GPL已经看不到了！好比你从21世纪坐时光穿梭机来到了19世纪，想再回去已经回不去了，肿么办？

办法其实还是有的，只要上面的命令行窗口还没有被关掉，你就可以顺着往上找啊找啊，找到那个append GPL的commit id是3628164...，于是就可以指定回到未来的某个版本：

$ git re**set** --hard 3628164

HEAD **is** now **at** 3628164 append GPL

版本号没必要写全，前几位就可以了，Git会自动去找。当然也不能只写前一两位，因为Git可能会找到多个版本号，就无法确定是哪一个了。

再小心翼翼地看看readme.txt的内容：

$ cat readme.txt

Git **is** a distributed version control system.

Git **is** free software distributed under the GPL.

果然，我胡汉三又回来了。

Git的版本回退速度非常快，因为Git在内部有个指向当前版本的HEAD指针，当你回退版本的时候，Git仅仅是把HEAD从指向append GPL：



改为指向add distributed：



然后顺便把工作区的文件更新了。所以你让HEAD指向哪个版本号，你就把当前版本定位在哪。

现在，你回退到了某个版本，关掉了电脑，第二天早上就后悔了，想恢复到新版本怎么办？找不到新版本的commit id怎么办？

在Git中，总是有后悔药可以吃的。当你用$ git reset --hard HEAD^回退到add distributed版本时，再想恢复到append GPL，就必须找到append GPL的commit id。Git提供了一个命令git reflog用来记录你的每一次命令：

$ git reflog

ea34578 HEAD@{0}: reset: moving to HEAD^

3628164 HEAD@{1}: commit: append GPL

ea34578 HEAD@{2}: commit: add distributed

cb926e7 HEAD@{3}: commit (initial): wrote a readme file

终于舒了口气，第二行显示append GPL的commit id是3628164

小结

现在总结一下：

* HEAD指向的版本就是当前版本，因此，Git允许我们在版本的历史之间穿梭，使用命令git reset --hard commit\_id。
* 穿梭前，用git log可以查看提交历史，以便确定要回退到哪个版本。
* 要重返未来，用git reflog查看命令历史，以便确定要回到未来的哪个版本。

#### 2、工作区和暂存区

Git和其他版本控制系统如SVN的一个不同之处就是有暂存区的概念。

先来看名词解释。

##### 工作区（Working Directory）

就是你在电脑里能看到的目录，比如我的learngit文件夹就是一个工作区：



##### 版本库（Repository）

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。

Git的版本库里存了很多东西，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。



分支和HEAD的概念我们以后再讲。

前面讲了我们把文件往Git版本库里添加的时候，是分两步执行的：

第一步是用git add把文件添加进去，实际上就是把文件修改添加到暂存区；

第二步是用git commit提交更改，实际上就是把暂存区的所有内容提交到当前分支。

因为我们创建Git版本库时，Git自动为我们创建了唯一一个master分支，所以，现在，git commit就是往master分支上提交更改。

你可以简单理解为，需要提交的文件修改通通放到暂存区，然后，一次性提交暂存区的所有修改。

俗话说，实践出真知。现在，我们再练习一遍，先对readme.txt做个修改，比如加上一行内容：

Git **is** a distributed version control system.

Git **is** free software distributed under the GPL.

Git has a mutable index called stage.

然后，在工作区新增一个LICENSE文本文件（内容随便写）。

先用git status查看一下状态：

$ git status

*# On branch master*

*# Changes not staged for commit:*

*# (use "git add <file>..." to update what will be committed)*

*# (use "git checkout -- <file>..." to discard changes in working directory)*

*#*

*# modified: readme.txt*

*#*

*# Untracked files:*

*# (use "git add <file>..." to include in what will be committed)*

*#*

*# LICENSE*

no changes added to commit (use "git add" **and**/**or** "git commit -a")

Git非常清楚地告诉我们，readme.txt被修改了，而LICENSE还从来没有被添加过，所以它的状态是Untracked。

现在，使用两次命令git add，把readme.txt和LICENSE都添加后，用git status再查看一下：

$ git status

*# On branch master*

*# Changes to be committed:*

*# (use "git reset HEAD <file>..." to unstage)*

*#*

*# new file: LICENSE*

*# modified: readme.txt*

*#*

现在，暂存区的状态就变成这样了：



所以，git add命令实际上就是把要提交的所有修改放到暂存区（Stage），然后，执行git commit就可以一次性把暂存区的所有修改提交到分支。

$ git **commit** -m "understand how stage works"

[master 27c9860] understand how stage works

2 files changed, 675 insertions(+)

**create** mode 100644 LICENSE

一旦提交后，如果你又没有对工作区做任何修改，那么工作区就是“干净”的：

$ git status

*# On branch master*

nothing to commit (working directory clean)

现在版本库变成了这样，暂存区就没有任何内容了：



小结

暂存区是Git非常重要的概念，弄明白了暂存区，就弄明白了Git的很多操作到底干了什么。

没弄明白暂存区是怎么回事的童鞋，请向上滚动页面，再看一次。

#### 3、管理修改

你会问，什么是修改？比如你新增了一行，这就是一个修改，删除了一行，也是一个修改，更改了某些字符，也是一个修改，删了一些又加了一些，也是一个修改，甚至创建一个新文件，也算一个修改。

为什么说Git管理的是修改，而不是文件呢？我们还是做实验。第一步，对readme.txt做一个修改，比如加一行内容：

$ cat readme.txt

Git **is** a distributed version control system.

Git **is** free software distributed under the GPL.

Git has a mutable index called stage.

Git tracks changes.

然后，添加：

$ git add readme.txt

$ git status

*# On branch master*

*# Changes to be committed:*

*# (use "git reset HEAD <file>..." to unstage)*

*#*

*# modified: readme.txt*

*#*

然后，再修改readme.txt：

$ cat readme.txt

Git **is** a distributed version control system.

Git **is** free software distributed under the GPL.

Git has a mutable index called stage.

Git tracks changes of files.

提交：

$ git **commit** -m "git tracks changes"

[master d4f25b6] git tracks changes

1 file changed, 1 insertion(+)

提交后，再看看状态：

$ git status

*# On branch master*

*# Changes not staged for commit:*

*# (use "git add <file>..." to update what will be committed)*

*# (use "git checkout -- <file>..." to discard changes in working directory)*

*#*

*# modified: readme.txt*

*#*

no changes added to commit (use "git add" **and**/**or** "git commit -a")

咦，怎么第二次的修改没有被提交？

别激动，我们回顾一下操作过程：

第一次修改 -> git add -> 第二次修改 -> git commit

你看，我们前面讲了，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。

提交后，用git diff HEAD -- readme.txt命令可以查看工作区和版本库里面最新版本的区别：

$ git diff HEAD -- readme.txt

diff --git a/readme.txt b/readme.txt

index 76d770f..a9c5755 100644

--- a/readme.txt

+++ b/readme.txt

@@ -1,4 +1,4 @@

Git **is** a distributed version control system.

Git **is** free software distributed under the GPL.

Git has a mutable index called stage.

-Git tracks changes.

+Git tracks changes of files.

可见，第二次修改确实没有被提交。

那怎么提交第二次修改呢？你可以继续git add再git commit，也可以别着急提交第一次修改，先git add第二次修改，再git commit，就相当于把两次修改合并后一块提交了：

第一次修改 -> git add -> 第二次修改 -> git add -> git commit

好，现在，把第二次修改提交了，然后开始小结。

小结

现在，你又理解了Git是如何跟踪修改的，每次修改，如果不add到暂存区，那就不会加入到commit中。

#### 4、撤销修改

自然，你是不会犯错的。不过现在是凌晨两点，你正在赶一份工作报告，你在readme.txt中添加了一行：

$ cat readme.txt

Git **is** a distributed version control system.

Git **is** free software distributed under the GPL.

Git has a mutable index called stage.

Git tracks changes of files.

My stupid boss still prefers SVN.

在你准备提交前，一杯咖啡起了作用，你猛然发现了“stupid boss”可能会让你丢掉这个月的奖金！

既然错误发现得很及时，就可以很容易地纠正它。你可以删掉最后一行，手动把文件恢复到上一个版本的状态。如果用git status查看一下：

$ git status

*# On branch master*

*# Changes not staged for commit:*

*# (use "git add <file>..." to update what will be committed)*

*# (use "git checkout -- <file>..." to discard changes in working directory)*

*#*

*# modified: readme.txt*

*#*

no changes added to commit (use "git add" **and**/**or** "git commit -a")

你可以发现，Git会告诉你，git checkout -- file可以丢弃工作区的修改：

$ git checkout -- readme.txt

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

一种是readme.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readme.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。

现在，看看readme.txt的文件内容：

$ cat readme.txt

Git **is** a distributed version control system.

Git **is** free software distributed under the GPL.

Git has a mutable index called stage.

Git tracks changes of files.

文件内容果然复原了。

git checkout -- file命令中的--很重要，没有--，就变成了“切换到另一个分支”的命令，我们在后面的分支管理中会再次遇到git checkout命令。

现在假定是凌晨3点，你不但写了一些胡话，还git add到暂存区了：

$ cat readme.txt

Git is a distributed version control system.

Git is free software distributed under the GPL.

Git has a mutable index called stage.

Git tracks changes of files.

My stupid boss still prefers SVN.

$ git add readme.txt

庆幸的是，在commit之前，你发现了这个问题。用git status查看一下，修改只是添加到了暂存区，还没有提交：

$ git status

*# On branch master*

*# Changes to be committed:*

*# (use "git reset HEAD <file>..." to unstage)*

*#*

*# modified: readme.txt*

*#*

Git同样告诉我们，用命令git reset HEAD file可以把暂存区的修改撤销掉（unstage），重新放回工作区：

$ git re**set** HEAD readme.txt

Unstaged changes after reset:

M readme.txt

git reset命令既可以回退版本，也可以把暂存区的修改回退到工作区。当我们用HEAD时，表示最新的版本。

再用git status查看一下，现在暂存区是干净的，工作区有修改：

$ git status

*# On branch master*

*# Changes not staged for commit:*

*# (use "git add <file>..." to update what will be committed)*

*# (use "git checkout -- <file>..." to discard changes in working directory)*

*#*

*# modified: readme.txt*

*#*

no changes added to commit (use "git add" **and**/**or** "git commit -a")

还记得如何丢弃工作区的修改吗？

$ git checkout -- readme.txt

$ git status

*# On branch master*

nothing to commit (working directory clean)

整个世界终于清静了！

现在，假设你不但改错了东西，还从暂存区提交到了版本库，怎么办呢？还记得[版本回退](https://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000/0013744142037508cf42e51debf49668810645e02887691000)一节吗？可以回退到上一个版本。不过，这是有条件的，就是你还没有把自己的本地版本库推送到远程。还记得Git是分布式版本控制系统吗？我们后面会讲到远程版本库，一旦你把“stupid boss”提交推送到远程版本库，你就真的惨了……

小结

又到了小结时间。

场景1：当你改乱了工作区某个文件的内容，想直接丢弃工作区的修改时，用命令git checkout -- file。

场景2：当你不但改乱了工作区某个文件的内容，还添加到了暂存区时，想丢弃修改，分两步，第一步用命令git reset HEAD file，就回到了场景1，第二步按场景1操作。

场景3：已经提交了不合适的修改到版本库时，想要撤销本次提交，参考[版本回退](https://www.liaoxuefeng.com/wiki/0013739516305929606dd18361248578c67b8067c8c017b000/0013744142037508cf42e51debf49668810645e02887691000)一节，不过前提是没有推送到远程库。

#### 5、删除文件

在Git中，删除也是一个修改操作，我们实战一下，先添加一个新文件test.txt到Git并且提交：

$ git add test.txt

$ git **commit** -m "add test.txt"

[master 94cdc44] **add** test.txt

1 file changed, 1 insertion(+)

**create** mode 100644 test.txt

一般情况下，你通常直接在文件管理器中把没用的文件删了，或者用rm命令删了：

$ rm test.txt

这个时候，Git知道你删除了文件，因此，工作区和版本库就不一致了，git status命令会立刻告诉你哪些文件被删除了：

$ git status

*# On branch master*

*# Changes not staged for commit:*

*# (use "git add/rm <file>..." to update what will be committed)*

*# (use "git checkout -- <file>..." to discard changes in working directory)*

*#*

*# deleted: test.txt*

*#*

no changes added to commit (use "git add" **and**/**or** "git commit -a")

现在你有两个选择，一是确实要从版本库中删除该文件，那就用命令git rm删掉，并且git commit：

$ git rm test.txt

rm 'test.txt'

$ git **commit** -m "remove test.txt"

[master d17efd8] remove test.txt

1 file changed, 1 deletion(-)

**delete** mode 100644 test.txt

现在，文件就从版本库中被删除了。

另一种情况是删错了，因为版本库里还有呢，所以可以很轻松地把误删的文件恢复到最新版本：

$ git checkout -- test.txt

git checkout其实是用版本库里的版本替换工作区的版本，无论工作区是修改还是删除，都可以“一键还原”。

小结

命令git rm用于删除一个文件。如果一个文件已经被提交到版本库，那么你永远不用担心误删，但是要小心，你只能恢复文件到最新版本，你会丢失**最近一次提交后你修改的内容**。

### 二、远程仓库

在继续阅读后续内容前，请自行注册GitHub账号。由于你的本地Git仓库和GitHub仓库之间的传输是通过SSH加密的，所以，需要一点设置：

第1步：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：

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

你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。

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：



点“Add Key”，你就应该看到已经添加的Key：



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。

当然，GitHub允许你添加多个Key。假定你有若干电脑，你一会儿在公司提交，一会儿在家里提交，只要把每台电脑的Key都添加到GitHub，就可以在每台电脑上往GitHub推送了。

最后友情提示，在GitHub上免费托管的Git仓库，任何人都可以看到喔（但只有你自己才能改）。所以，不要把敏感信息放进去。

如果你不想让别人看到Git库，有两个办法，一个是交点保护费，让GitHub把公开的仓库变成私有的，这样别人就看不见了（不可读更不可写）。另一个办法是自己动手，搭一个Git服务器，因为是你自己的Git服务器，所以别人也是看不见的。这个方法我们后面会讲到的，相当简单，公司内部开发必备。

确保你拥有一个GitHub账号后，我们就即将开始远程仓库的学习。

小结

“有了远程仓库，妈妈再也不用担心我的硬盘了。”——Git点读机

#### 1、添加远程仓库

现在的情景是，你已经在本地创建了一个Git仓库后，又想在GitHub创建一个Git仓库，并且让这两个仓库进行远程同步，这样，GitHub上的仓库既可以作为备份，又可以让其他人通过该仓库来协作，真是一举多得。

首先，登陆GitHub，然后，在右上角找到“Create a new repo”按钮，创建一个新的仓库：



在Repository name填入learngit，其他保持默认设置，点击“Create repository”按钮，就成功地创建了一个新的Git仓库：



目前，在GitHub上的这个learngit仓库还是空的，GitHub告诉我们，可以从这个仓库克隆出新的仓库，也可以把一个已有的本地仓库与之关联，然后，把本地仓库的内容推送到GitHub仓库。

现在，我们根据GitHub的提示，在本地的learngit仓库下运行命令：

$ git remote add origin git@github.com:michaelliao/learngit.git

请千万注意，把上面的michaelliao替换成你自己的GitHub账户名，否则，你在本地关联的就是我的远程库，关联没有问题，但是你以后推送是推不上去的，因为你的SSH Key公钥不在我的账户列表中。

添加后，远程库的名字就是origin，这是Git默认的叫法，也可以改成别的，但是origin这个名字一看就知道是远程库。

下一步，就可以把本地库的所有内容推送到远程库上：

$ git push -u origin master

Counting objects: 19, done.

Delta compression using up to 4 threads.

Compressing objects: 100% (19/19), done.

Writing objects: 100% (19/19), 13.73 KiB, done.

Total 23 (delta 6), reused 0 (delta 0)

To git@github.com:michaelliao/learngit.git

\* [new branch] master -> master

Branch master **set** up **to** track remote branch master **from** origin.

把本地库的内容推送到远程，用git push命令，实际上是把当前分支master推送到远程。

由于远程库是空的，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。

推送成功后，可以立刻在GitHub页面中看到远程库的内容已经和本地一模一样：



从现在起，只要本地作了提交，就可以通过命令：

$ git push origin master

把本地master分支的最新修改推送至GitHub，现在，你就拥有了真正的分布式版本库！

SSH警告

当你第一次使用Git的clone或者push命令连接GitHub时，会得到一个警告：

The authenticity of host 'github.com (xx.xx.xx.xx)' can't be established.

RSA key fingerprint is xx.xx.xx.xx.xx.

Are you sure you want to continue connecting (yes/no)?

这是因为Git使用SSH连接，而SSH连接在第一次验证GitHub服务器的Key时，需要你确认GitHub的Key的指纹信息是否真的来自GitHub的服务器，输入yes回车即可。

Git会输出一个警告，告诉你已经把GitHub的Key添加到本机的一个信任列表里了：

Warning: Permanently added 'github.com' (RSA) to the **list** of known hosts.

这个警告只会出现一次，后面的操作就不会有任何警告了。

如果你实在担心有人冒充GitHub服务器，输入yes前可以对照[GitHub的RSA Key的指纹信息](https://help.github.com/articles/what-are-github-s-ssh-key-fingerprints/)是否与SSH连接给出的一致。

小结

要关联一个远程库，使用命令git remote add origin git@server-name:path/repo-name.git；

关联后，使用命令git push -u origin master第一次推送master分支的所有内容；

此后，每次本地提交后，只要有必要，就可以使用命令git push origin master推送最新修改；

分布式版本系统的最大好处之一是在本地工作完全不需要考虑远程库的存在，也就是有没有联网都可以正常工作，而SVN在没有联网的时候是拒绝干活的！当有网络的时候，再把本地提交推送一下就完成了同步，真是太方便了！

#### 2、从远程仓库克隆

上次我们讲了先有本地库，后有远程库的时候，如何关联远程库。

现在，假设我们从零开发，那么最好的方式是先创建远程库，然后，从远程库克隆。

首先，登陆GitHub，创建一个新的仓库，名字叫gitskills：



我们勾选Initialize this repository with a README，这样GitHub会自动为我们创建一个README.md文件。创建完毕后，可以看到README.md文件：



现在，远程库已经准备好了，下一步是用命令git clone克隆一个本地库：

$ git clone git@github.com:michaelliao/gitskills.git

Cloning into 'gitskills'...

remote: Counting objects: 3, done.

remote: Total 3 (delta 0), reused 0 (delta 0)

Receiving objects: 100% (3/3), done.

$ cd gitskills

$ ls

README.md

注意把Git库的地址换成你自己的，然后进入gitskills目录看看，已经有README.md文件了。

如果有多个人协作开发，那么每个人各自从远程克隆一份就可以了。

你也许还注意到，GitHub给出的地址不止一个，还可以用https://github.com/michaelliao/gitskills.git这样的地址。实际上，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。

小结

要克隆一个仓库，首先必须知道仓库的地址，然后使用git clone命令克隆。

Git支持多种协议，包括https，但通过ssh支持的原生git协议速度最快。

### 创建一个版本库

创建一个版本库非常简单，

首先，选择一个合适的地方，创建一个空目录：

$ mkdir learngit

$ cd learngit

$ pwd

/Users/michael/learngit

如果你使用Windows系统，为了避免遇到各种莫名其妙的问题，请确保目录名（包括父目录）不包含中文。

第二步，通过git init命令把这个目录变成Git可以管理的仓库：

$ git init

Initialized empty Git repository **in** /Users/michael/learngit/.git/

瞬间Git就把仓库建好了，而且告诉你是一个空的仓库（empty Git repository），细心的读者可以发现当前目录下多了一个.git的目录，这个目录是Git来跟踪管理版本库的，没事千万不要手动修改这个目录里面的文件，不然改乱了，就把Git仓库给破坏了。

如果你没有看到.git目录，那是因为这个目录默认是隐藏的，用ls -ah命令就可以看见。

### 文件添加到版本库

现在我们编写一个readme.txt文件，内容如下：

Git **is** a version control system.

Git **is** free software.

一定要放到learngit目录下（子目录也行），因为这是一个Git仓库，放到其他地方Git再厉害也找不到这个文件。

和把大象放到冰箱需要3步相比，把一个文件放到Git仓库只需要两步。

第一步，用命令git add告诉Git，把文件添加到仓库：

$ git add readme.txt

执行上面的命令，没有任何显示，这就对了，Unix的哲学是“没有消息就是好消息”，说明添加成功。

第二步，用命令git commit告诉Git，把文件提交到仓库：

$ git **commit** -m "wrote a readme file"

[master (root-**commit**) cb926e7] wrote a readme file

1 file changed, 2 insertions(+)

**create** mode 100644 readme.txt

简单解释一下git commit命令，-m后面输入的是本次提交的说明，可以输入任意内容，当然最好是有意义的，这样你就能从历史记录里方便地找到改动记录。

嫌麻烦不想输入-m "xxx"行不行？确实有办法可以这么干，但是强烈不建议你这么干，因为输入说明对自己对别人阅读都很重要。实在不想输入说明的童鞋请自行Google，我不告诉你这个参数。

git commit命令执行成功后会告诉你，1个文件被改动（我们新添加的readme.txt文件），插入了两行内容（readme.txt有两行内容）。

为什么Git添加文件需要

$ git add file2.txt file3.txt

$ git commit -m "add 3 files."