关键词： git简介、创建版本库、远程仓库、版本回退

目录

[# git简介 1](#_Toc6153099)

[## git 的诞生 3](#_Toc6153100)

[## 集中式 vs 分布式 3](#_Toc6153101)

[## 工作区与暂存区 5](#_Toc6153102)

[# git安装 5](#_Toc6153103)

[# 创建版本库 9](#_Toc6153104)

[# 远程仓库 13](#_Toc6153105)

[## 添加远程库 13](#_Toc6153106)

[小结 18](#_Toc6153107)

[## 从远程库克隆 18](#_Toc6153108)

[小结 20](#_Toc6153109)

[# 时光穿梭 20](#_Toc6153110)

[## 版本回退 21](#_Toc6153111)

[## 管理修改 21](#_Toc6153112)

[## 撤销修改 21](#_Toc6153113)

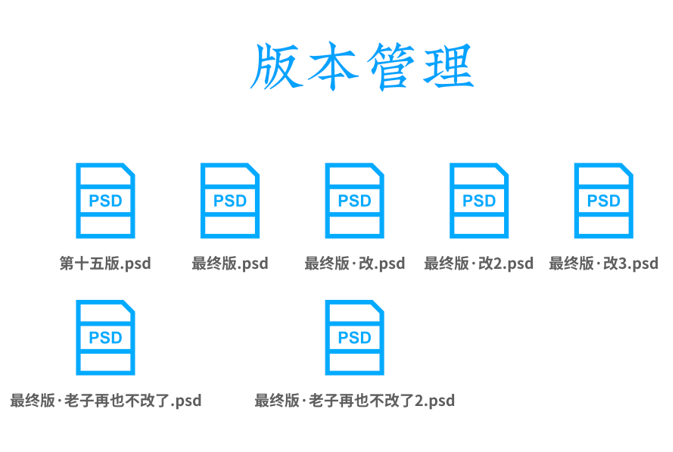
[## 删除文件 21](#_Toc6153114)



# # git简介

Git是目前世界上最先进的分布式版本控制系统！

据说每个设计师都会有一个文件夹，里边放满了各种版本的「最终版」设计稿。



想删除一个段落，又怕将来想恢复找不回来怎么办？

有办法！先把当前文件“另存为……”一个新的Word文件，再接着改，改到一定程度，再“另存为……”一个新文件，这样一直改下去，最后你的Word文档变成了上面这样。

过了一周，你想找回被删除的文字，但是已经记不清删除前保存在哪个文件里了，只好一个一个文件去找，真麻烦。看着一堆乱七八糟的文件，想保留最新的一个，然后把其他的删掉，又怕哪天会用上，还不敢删，真郁闷。

更要命的是，有些部分需要你的财务同事帮助填写，于是你把文件Copy到U盘里给她，然后，你继续修改Word文件。一天后，同事再把Word文件传给你，此时，你必须想想，发给她之后到你收到她的文件期间，你作了哪些改动，得把你的改动和她的部分合并，真困难。

这其实就是「人肉版」的「版本管理」。

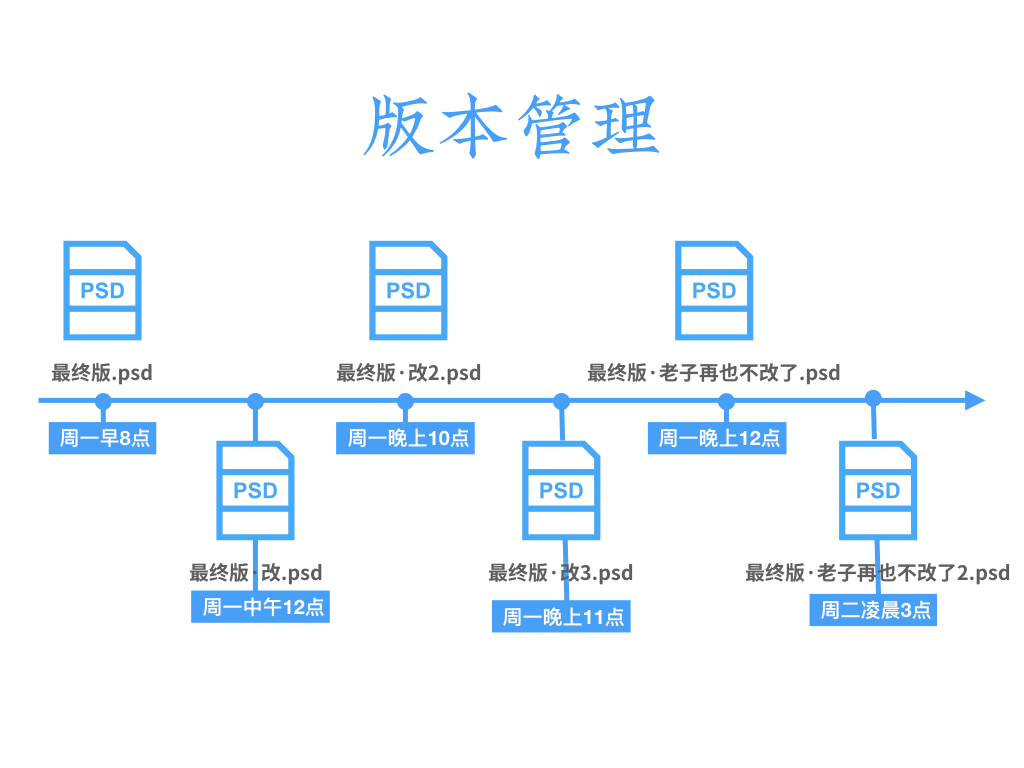
我们需要专门的软件来对代码做版本管理。

于是你想，如果有一个软件，不但能自动帮我记录每次文件的改动，还可以让同事协作编辑，这样就不用自己管理一堆类似的文件了，也不需要把文件传来传去。如果想查看某次改动，只需要在软件里瞄一眼就可以，岂不是很方便？



这样，你就结束了手动管理多个“版本”的史前时代，进入到版本控制的20世纪。

版本管理最大的意义在于，它可以让我们随时「回溯」到之前某个时刻已经被覆盖过的版本去。



## ## git 的诞生

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。

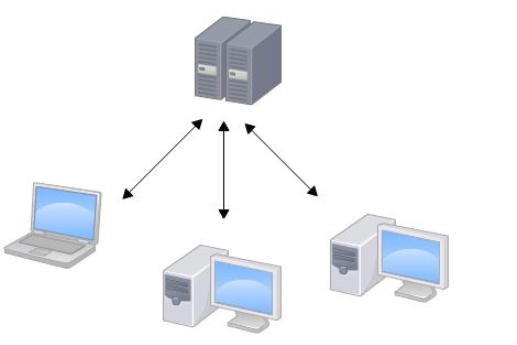
安定团结的大好局面在2005年就被打破了，原因是Linux社区牛人聚集，不免沾染了一些梁山好汉的江湖习气。开发Samba的Andrew试图破解BitKeeper的协议（这么干的其实也不只他一个），被BitMover公司发现了（监控工作做得不错！），于是BitMover公司怒了，要收回Linux社区的免费使用权。

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。

## ## 集中式 vs 分布式

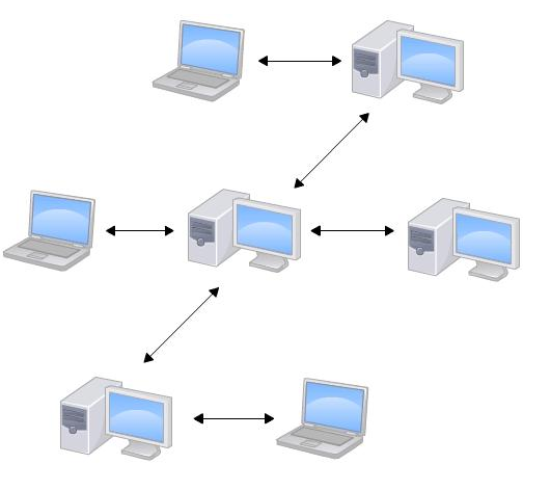
先说集中式版本控制系统，版本库是集中存放在中央服务器的，而干活的时候，用的都是自己的电脑，所以要先从中央服务器取得最新的版本，然后开始干活，干完活了，再把自己的活推送给中央服务器。中央服务器就好比是一个图书馆，你要改一本书，必须先从图书馆借出来，然后回到家自己改，改完了，再放回图书馆



缺点：要求联网、服务器崩溃

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。

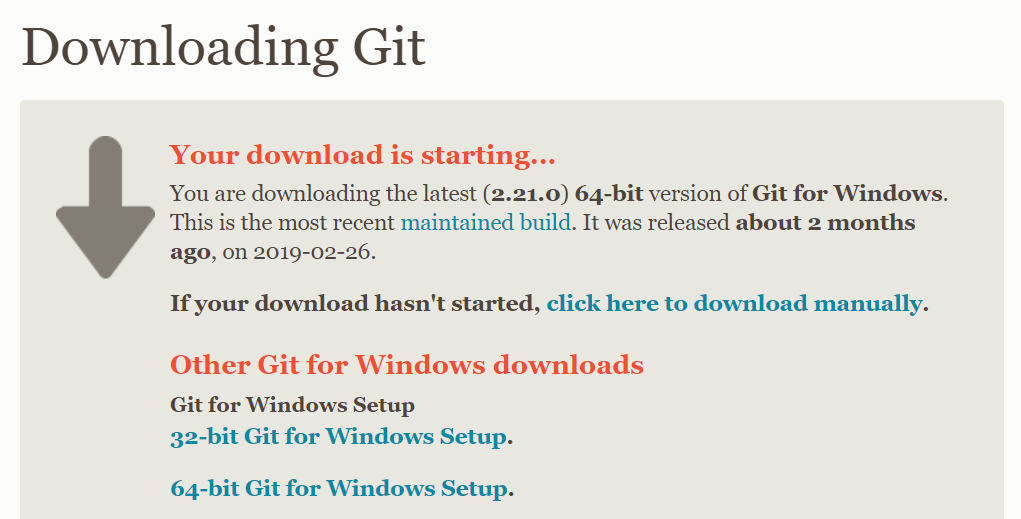


## ## 工作区与暂存区

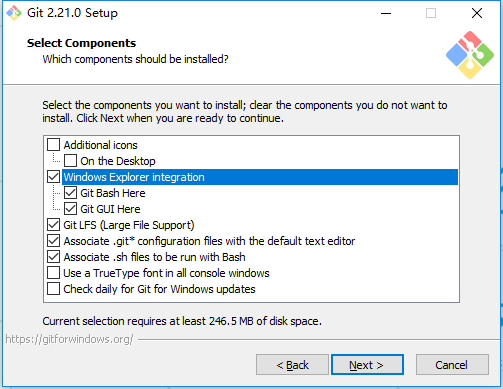
# # git安装

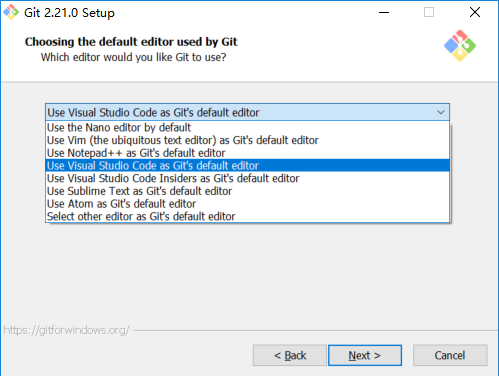
地址：

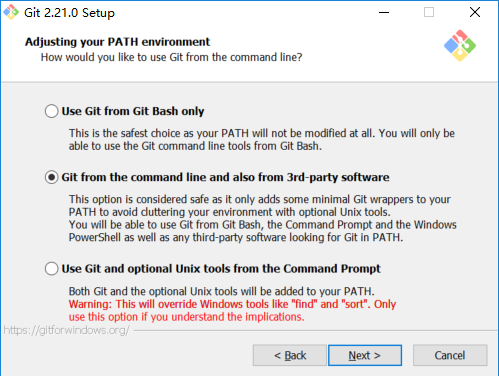
<https://git-scm.com/downloads>

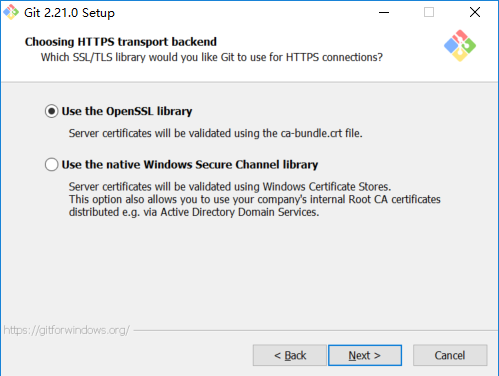


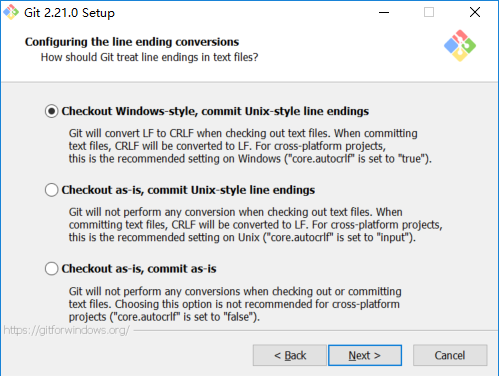
默认选项安装

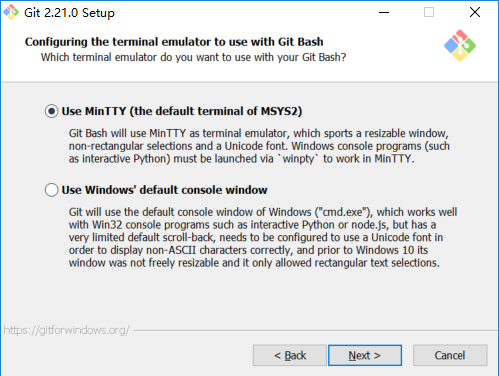


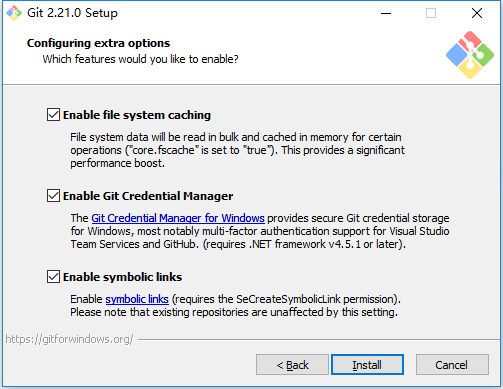


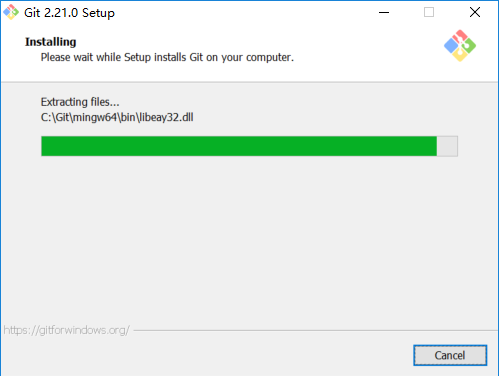








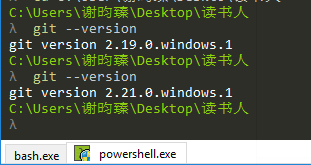




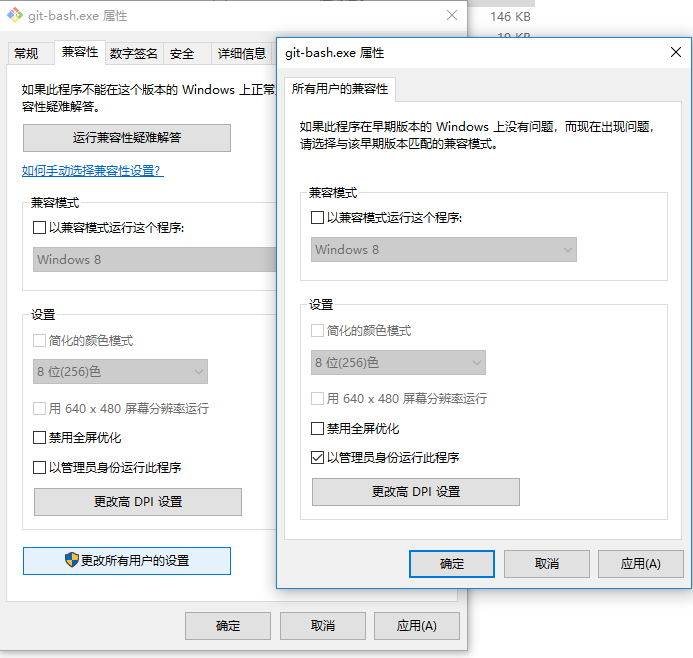
## 测试

右键，有git bash here 和 git GUI here选项则说明安装成功。

可以在命令行中输入git --version查看当前git版本



安装完成后，还需要一步设置：在git的安装路径将git-bash设置为以管理员身份允许。

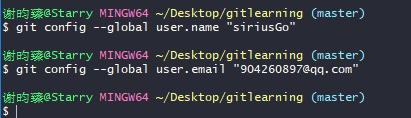


接下来要对本地的git的账号进行设置。

在命令行输入：

$ git config --global user.name “Your Name”

$ git config --global user.email “email@example.com”



注意git config命令的--global参数，用了这个参数，表示你这台机器上所有的Git仓库都会使用这个配置，当然也可以对某个仓库指定不同的用户名和Email地址。

# # 创建版本库

什么是版本库呢？版本库又名仓库，英文名**repository**，你可以简单理解成一个目录，这个目录里面的所有文件都可以被Git管理起来，每个文件的修改、删除，Git都能跟踪，以便任何时刻都可以追踪历史，或者在将来某个时刻可以“还原”。

所以，创建一个版本库非常简单，首先，选择一个合适的地方，右键，git bash here,在跳出的命令行中创建一个空目录（也可以手动创建）:

$ mkdir learngit

$ cd learngit

$ pwd



mkdir 创建文件夹

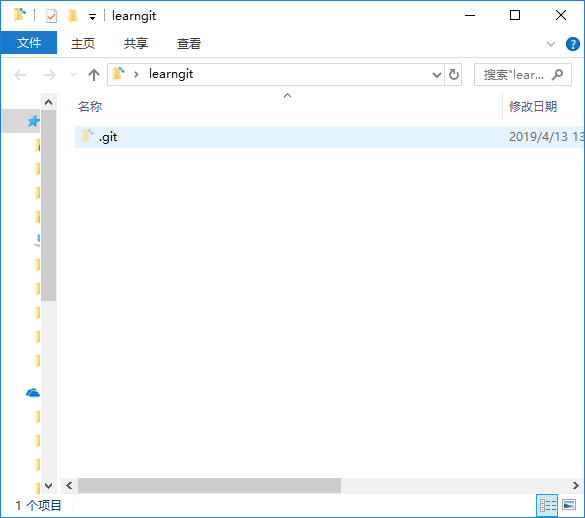
cd 转换目录

pwd命令用于显示当前目录。

Tab键可以自动补全文件名

第二步，通过git init命令把这个目录变成Git可以管理的仓库：





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。

如果没看到是因为生成的文件是隐藏文件夹，在文件属性里选择显示隐藏文件夹即可。

也不一定必须在空目录下创建Git仓库，选择一个已经有文件的目录也是可以的。

然后就可以把需要版本管理的文件放入这个文件夹，与.git同一个目录。

首先这里再明确一下，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。

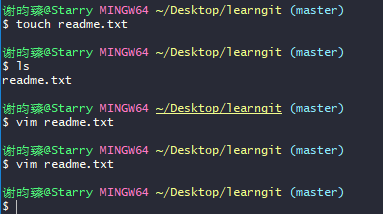
不幸的是，Microsoft的Word格式是二进制格式，因此，版本控制系统是没法跟踪Word文件的改动的，前面我们举的例子只是为了演示，如果要真正使用版本控制系统，就要以纯文本方式编写文件。

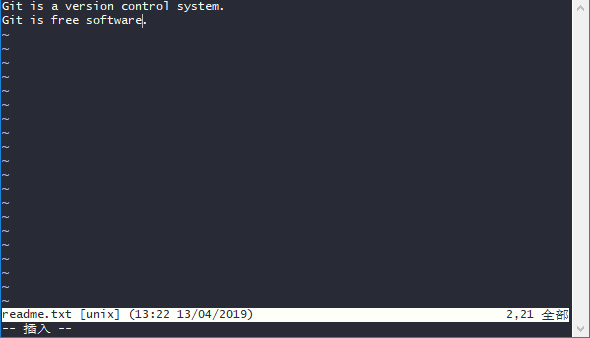
因为文本是有编码的，比如中文有常用的GBK编码，日文有Shift\_JIS编码，如果没有历史遗留问题，强烈建议使用标准的UTF-8编码，所有语言使用同一种编码，既没有冲突，又被所有平台所支持。

现在放了一个readme.txt文件进去 内容为：

Git is a version control system.

Git is free software.

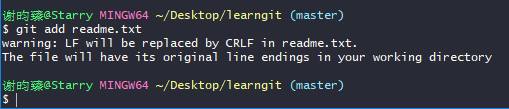




和把大象放到冰箱需要3步相比，把一个文件放到Git仓库只需要两步。

第一步，用命令git add告诉Git，把文件添加到git暂存区里：

$ git add readme.txt



（我们可以运行 git reset 命令来将暂存区里边的某个文件移出来。

$ git reset HEAD-- file 从暂存区移除某文件或某目录

$ git status 查看当前有.git文件的文件夹下的修改状态 ）

(.git文件夹下的内容

其中objects目录就是用来存放文件的全量历史数据的。

进入objects目录，我们可以发现下边有一个61目录，里边有一个以ac开头的文件。

这里将61和这个文件名拼接起来就可以得到我们刚才add的文件的SHA1。

把前两位拿出来作为目录名，这是为了避免在同一个目录里存放太多文件，导致文件索引过大而降低性能。

我们来确认下文件内容，通过git的cat-file命令，我们可以查看存储到objects目录下的所有对象。

而这个「index」文件就是我们之前提到的「暂存区」。

我们可以通过「git ls-files --stage」命令来查看它的内容。

可以看到，a.txt这个文件名和它当前对应的SHA1值都被记录在了暂存区中。

「HEAD」文件描述了当前分支上Head的位置。

通过「cat」命令我们可以查看到HEAD文件里边其实放的是一个引用地址。

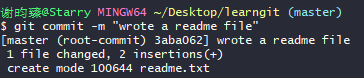
而这个引用指向的地方是不存在的。

这是因为，我们还没有进行过commit。

)

第二步，用命令git commit告诉Git，将暂存区的文件提交到本地仓库：

$ git commit -m "wrote a readme file"



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。

git commit命令执行成功后会告诉你，1 file changed：1个文件被改动（我们新添加的readme.txt文件）；2 insertions：插入了两行内容（readme.txt有两行内容）。

为什么Git添加文件需要add，commit一共两步呢？因为commit可以一次提交很多文件，所以你可以多次add不同的文件，比如：

$ git add file1.txt

$ git add file2.txt file3.txt

$ git commit -m "add 3 files."

这个时候我们再来看一下HEAD里边的那个引用。

可以看到，现在这个引用就指向了一个提交。

而这个提交里边，正是我们刚才的内容。

有时候我们修改的文件比较多，可能会忘了哪些需要add。

这时候，我们可以通过git status命令来查看。

$ git status 查看本地文件的修改和暂存区情况

比如我们修改了b.txt，但是并没有add它，那么我们运行 git status 时就能看到上图的提示。▼

Untracked files:

(use “git ad <file>. . .” to include in what will be committed) . . .

千万不要使用Windows自带的**记事本**编辑任何文本文件。

# # 远程仓库

## ## 添加远程库

## 注册github账号

<https://github.com/>

## ssh key 权限

先在bash里看看自己的身份的名字和邮箱

1 git config user.name

2 git config user.email

如果不是之前设置的就重新设置一遍

找到这个文件夹

C:\Users\(计算机用户名)\.ssh

3 git 输入命令

ssh-keygen

接着出现：

Generating public/private rsa key pair.

Enter file in which to save the key (/Users/（计算机用户名）/.ssh/id\_rsa):

请直接按下回车，即将ssh文件下载在默认文件夹（/Users/（计算机用户名）/.ssh/）中

再回车 将密码设置成空白 以便之后登陆不需要再输入密码

再回车，确认密码

然后系统会自动在.ssh文件夹下生成两个文件，id\_rsa和id\_rsa.pub，用记事本打开id\_rsa.pub

（id\_rsa.pub 为公钥 另一个是私钥）

将全部的内容复制

打开https://github.com/，登陆你的账户，进入设置

进入ssh设置

New SSH key

把刚才复制的粘贴进去

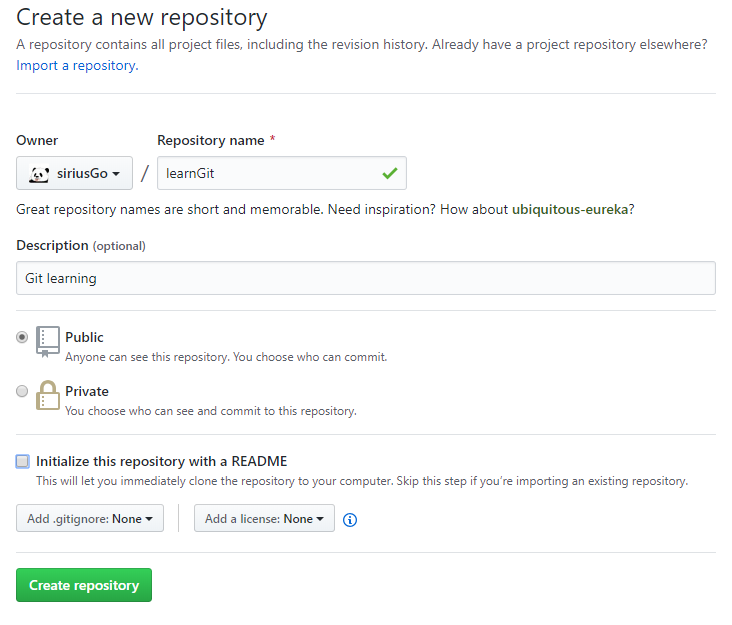
点击 add ssh key

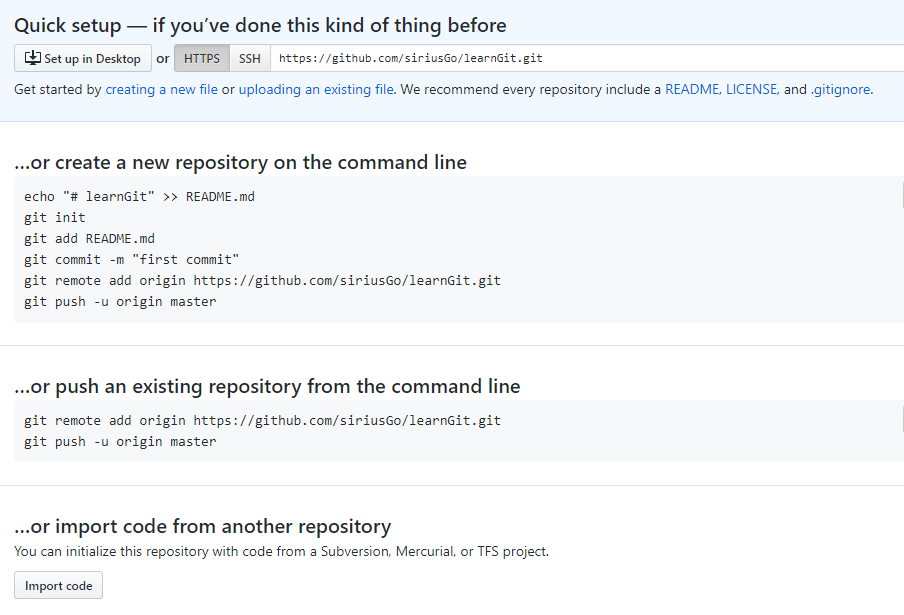
回车

提示成功

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关联、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：



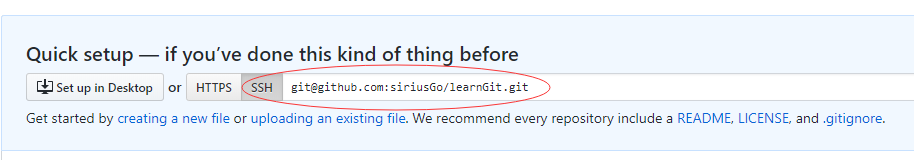


目前，在GitHub上的这个learngit仓库还是空的，GitHub告诉我们，可以从这个仓库克隆出新的仓库，也可以把一个已有的本地仓库与之关联，然后，把本地仓库的内容推送到GitHub仓库。

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:siriusGo/learnGit.git

这里 origin 后面的地址在这里



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

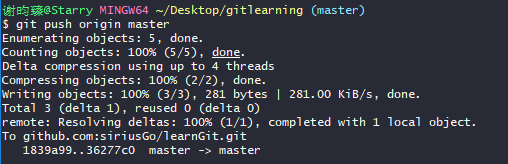
把本地库的内容推送到远程，用git push命令，实际上是把当前分支master推送到远程。

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。

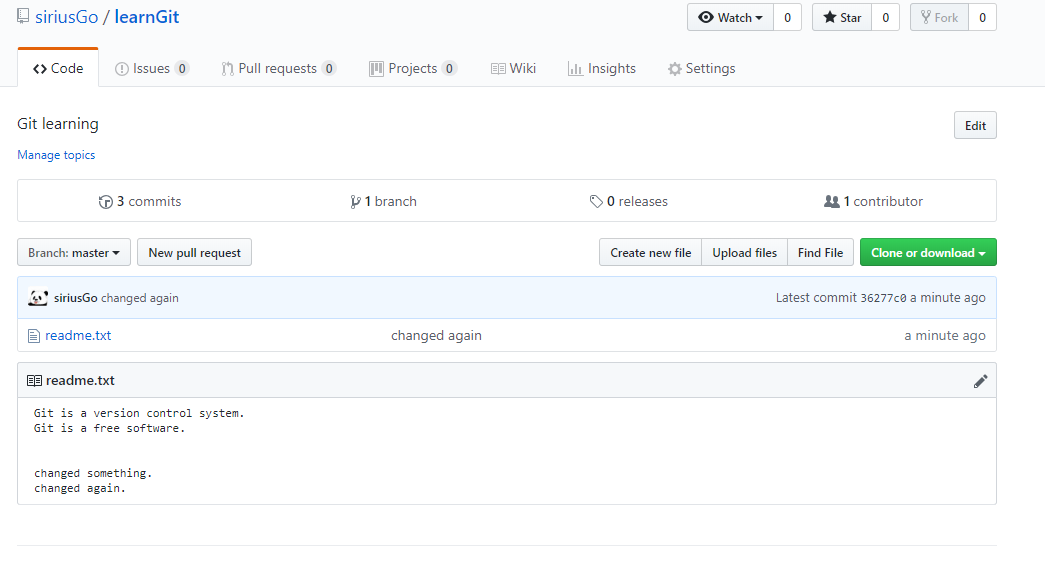
推送成功后，可以立刻在GitHub页面中看到远程库的内容已经和本地一模一样：

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！



那这个本地仓库是否和服务器端的仓库关联呢，我们来看看。

通过 git remote 命令，我们可以查看到所有关联到当前仓库的远程仓库。

$ git remote 远程仓库列表

orgin

可以看到，本地仓库和一个叫做「origin」的远程仓库关联。

而通过 git remote show 命令，我们可以查看某一个远程仓库的详细信息。

$ git remote show name 显示远程仓库信息

可以看到，这些信息包括了Fetch URL 和 Push URL。

通过 git push 和 git pull 命令，我们可以从服务器端推送和拉取代码。

### 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；

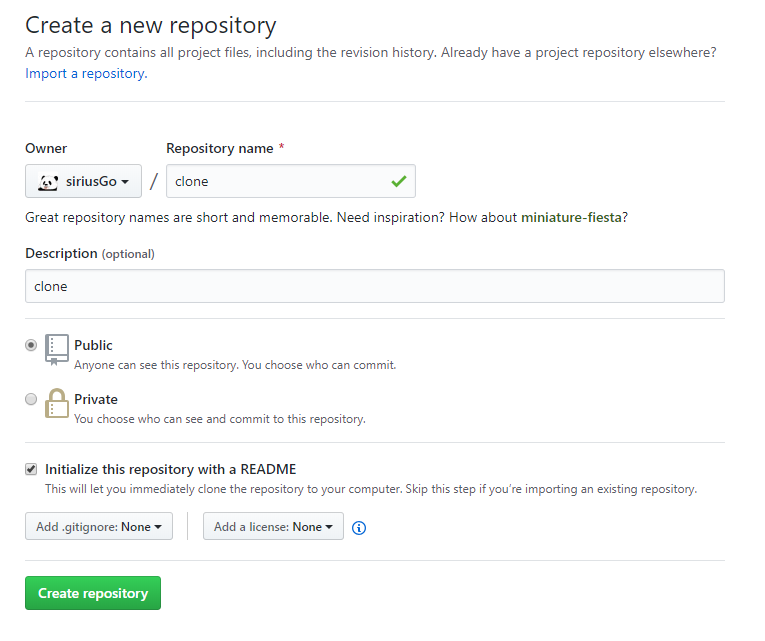
分布式版本系统的最大好处之一是在本地工作完全不需要考虑远程库的存在，也就是有没有联网都可以正常工作，而SVN在没有联网的时候是拒绝干活的！当有网络的时候，再把本地提交推送一下就完成了同步，真是太方便了！

## ## 从远程库克隆

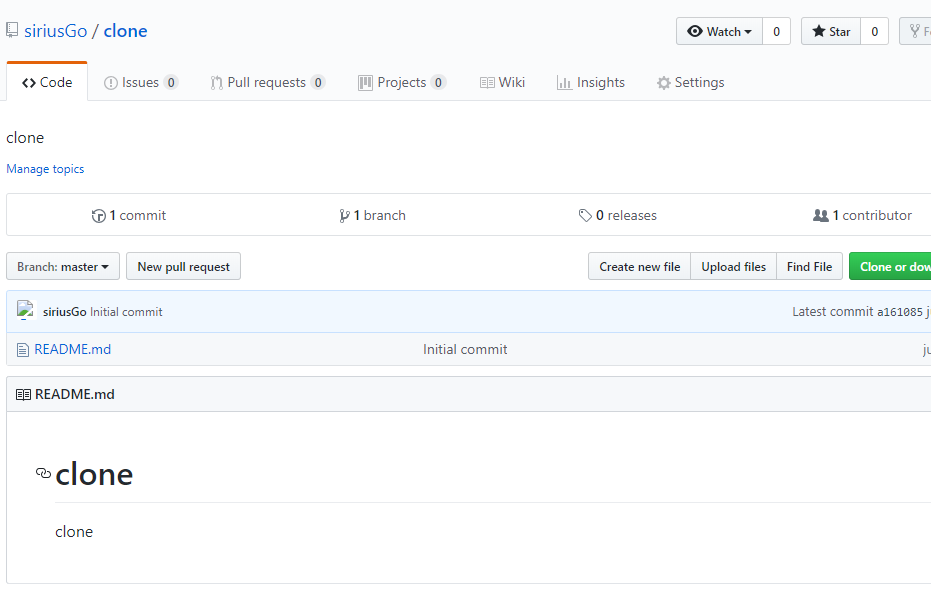
上次我们讲了先有本地库，后有远程库的时候，如何关联远程库。

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。

创建一个新的仓库

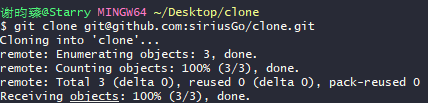


我们勾选Initialize this repository with a README，这样GitHub会自动为我们创建一个README.md文件。创建完毕后，可以看到README.md文件：



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，在你想要克隆的目录下创建文件夹并打开bash输入：

$ git clone git@github.com:siriusGo/clone.git



注意把Git库的地址换成你自己的，然后进入目录看看，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，还可以用仓库地址+.git <https://github.com/siriusGo/clone>.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。

# # 时光穿梭

## ## 版本回退

## ## 管理修改

## ## 撤销修改

## ## 删除文件