# 第二十一章 使用 git 完成本地代码版本管理

本节所讲内容：

21.1 版本控制介绍以及常用的版本控制工具

21.2 版本控制工具-GIT

神奇带闪电网站：https://learngitbranching.js.org/

## 21.1 版本控制介绍以及常用的版本控制工具

版本控制是指对软件开发过程中各种程序代码、[配置文件](https://baike.baidu.com/item/%E9%85%8D%E7%BD%AE%E6%96%87%E4%BB%B6/286550" \t "_blank)及说明文档等文件变更的管理，是[软件配置管理](https://baike.baidu.com/item/%E8%BD%AF%E4%BB%B6%E9%85%8D%E7%BD%AE%E7%AE%A1%E7%90%86/3765602)的核心思想之一。

编写一个成熟可用的程序是一个工作量很大的工程，并非我们一次性就可以搞定的工作，所以在开发过程当中需要：

1. 多人协作

随着对程序体验的需求的提高，一个程序需求的编程知识和模块也在增多，这种情况下让一个程序员同时掌握多门技术是不好实现的：

1、掌握的难度大，开发的成本高（比如：一个大牛的工资）

2、开发效率高，一个人开发的效率是不行的

所以，我们在工作当中大部分讲究的是协作开发，我们以项目需求的技术模块进行团队的组合。

比如，开发一个web项目：如果要招聘一个web大牛，前端、后端、运维服务器都很牛的大牛，薪资高先不说，人也不好找啊。并且一个大牛的开发效率和开发压力也很大。所以我们会形成一个开发的团队，找前端开发工程师，后端开发工程师，运维工程师，数据库工程师来完成这个光荣而又艰巨的任务。

1. 版本迭代

就好像一个美术家要完成一件作品，并不是一蹴而就的，好多时候是经历过多次修改的过程，我们编程也是一样的，当然这个修改要有原则，并不是推倒重来的过程（当然前期无药可救的不算），而是有简单的一个完整的框架开始，然后不断优化升级的过程，这个过程就是版本迭代。

那在这个过程当中，我们需要对代码进行管理，比如：提交、检出、回溯历史、冲突解决、多人协作。那这些需求也就衍生出了我们要学习使用的版本控制工具。

各个公司由于开发的需求和其他因素用到的版本控制工具不都相同，这里我们介绍几种使用较多的版本控制工具。

cvs： 是一个C/S系统,是一个常用的代码版本控制软件。主要在开源软件管理中使用。

多个开发人员通过一个中心版本控制系统来记录文件版本，从而达到保证文件同步的目的。是一种很古老的版本控制工具了，但是是很典型的集中式版本控制工具

SVN： 是一个开放源代码的版本控制系统，相较于RCS、CVS，它采用了分支管理系统，它的设计目标就是取代CVS。可以说是集中式版本控制的集大成者。

Git： 是一个开源的分布式版本控制系统，可以有效、高速的处理从很小到非常大的项目版本管理。是一种分布式的版本控制工具

GitHub：gitHub是一个面向开源及私有软件项目的托管平台,因为只支持git 作为唯一的版本库格式进行托管,故名gitHub。

上面介绍了我们常用的四种版本控制软件，但是也要给大家解释两个概念：

分布式版本控制：分布式的版本控制就是每个人都可以创建一个独立的代码仓库用于管理，各种版本控制的操作都可以在本地完成。每个人修改的代码都可以推送合并到另外一个代码仓库中。

集中式版本控制：只有一个中央控制，所有的开发人员都必须依赖于这个代码仓库。每次版本控的操作也必须链接到服务器才能完成。

所以很多公司喜欢用集中式的版本控制是为了更好的控制代码。如果个人开发，就可以选择Git这种分布式的。并不存在那个更加好或者其他的。

## 21.2 版本控制工具-GIT

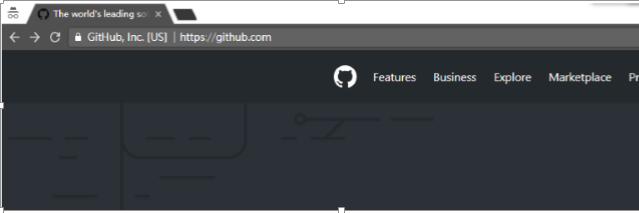
gitHub是一个面向开源及私有软件项目的托管平台,因为只支持git 作为唯一的版本库格式进行托管,故名gitHub。在这里我们以github作为案例给大家讲一下git在工作当中的使用

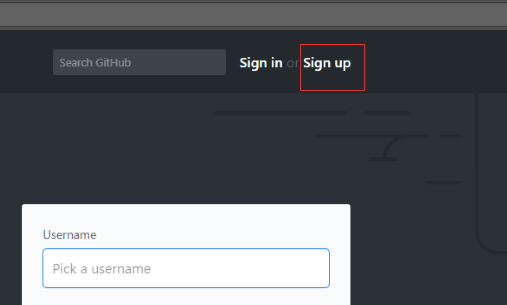
浏览器当中使用github

使用git

创建自己的GitHub账号：

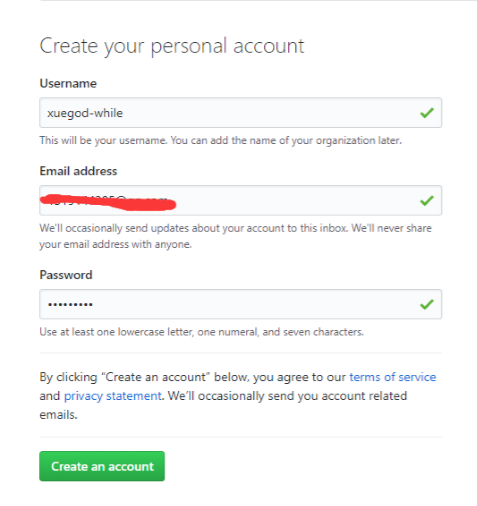
首先打开github登录界面，选择注册





然后填写自己的注册信息

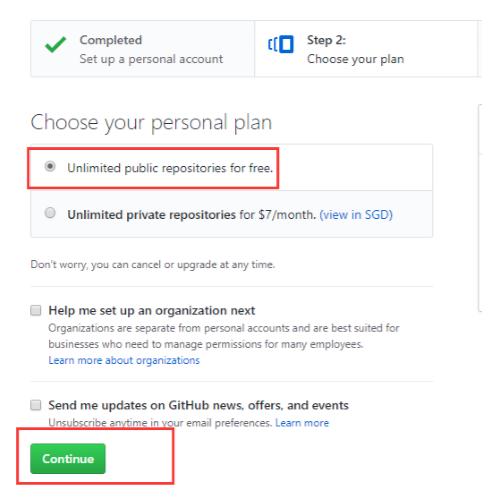
填写用户名，密码和绑定邮箱



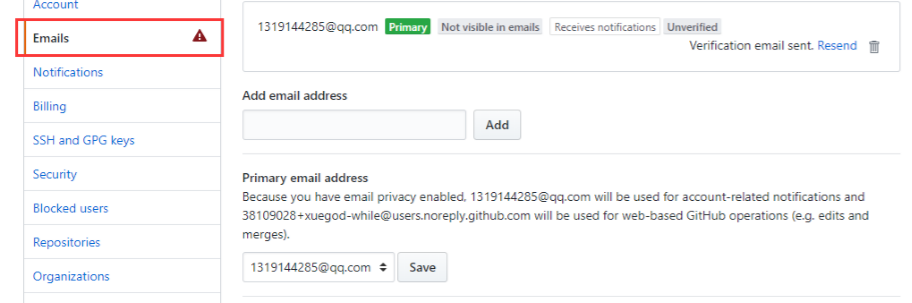
选择之后会进行仓库选项，github有免费的仓库和付费的仓库（每个月7$），

区别就在于免费的仓库当中的代码完全开源，付费仓库的代码就不一样了。

我们练习当然不必如此，所以在接下来要选择free：



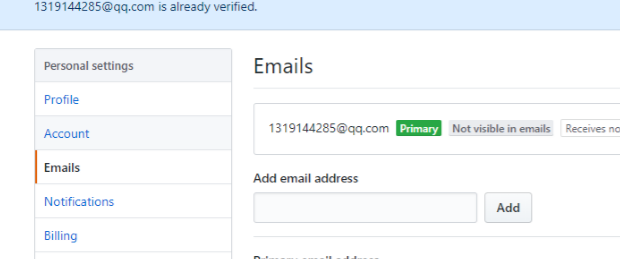
完成之后，会有邮件绑定警告，我们需要绑定一个自己的邮箱



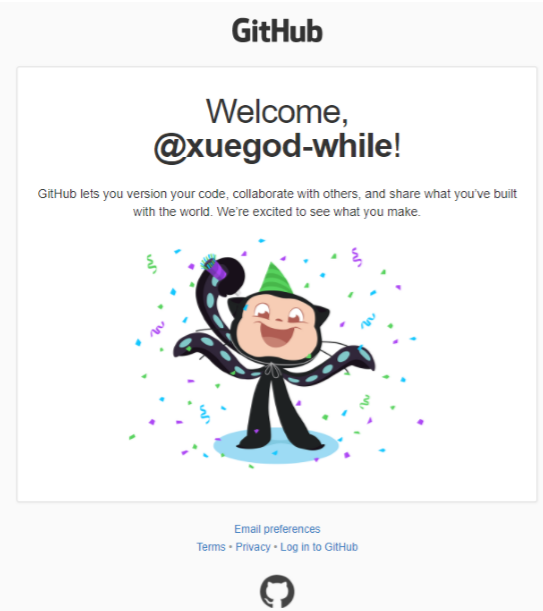
不用多说，我们到邮箱完成校验



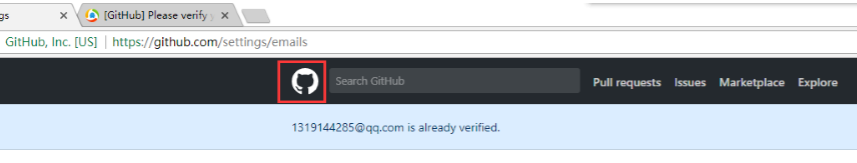
然后跳转回来，



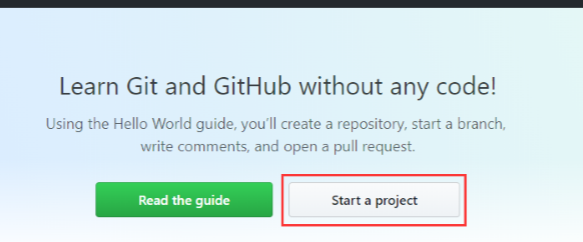
接着收到邮件

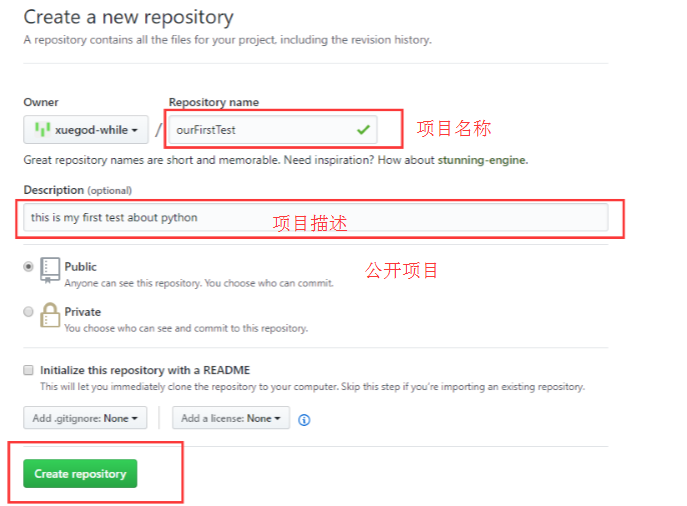


搞定收工，我们可以通过这里返回首页，创建项目



创建项目





然后出现git提醒

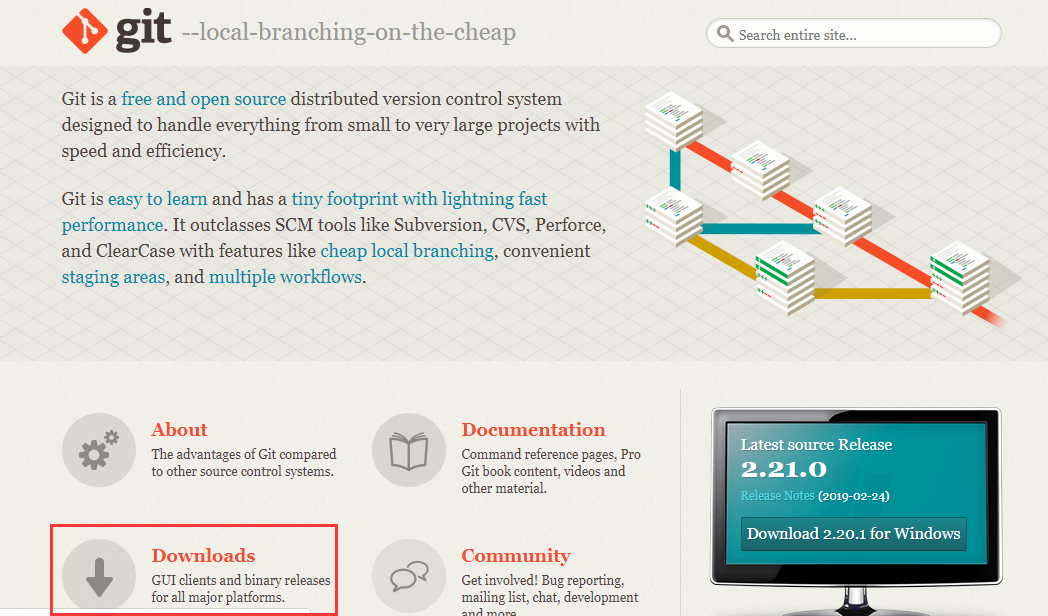


这样我们就完成了git账号的注册和项目的创建

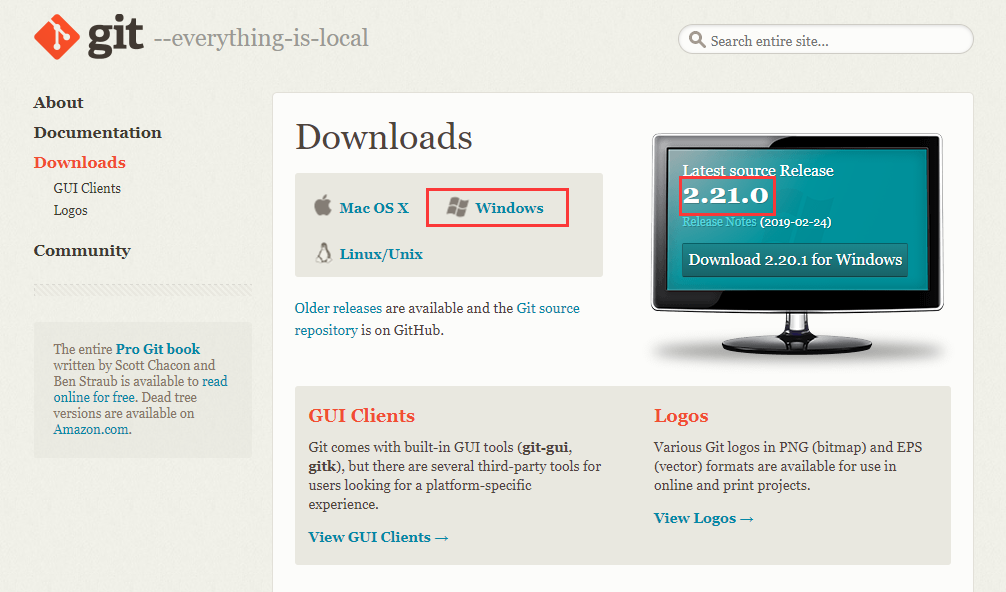
Github客户端的使用

1、下载、安装客户端

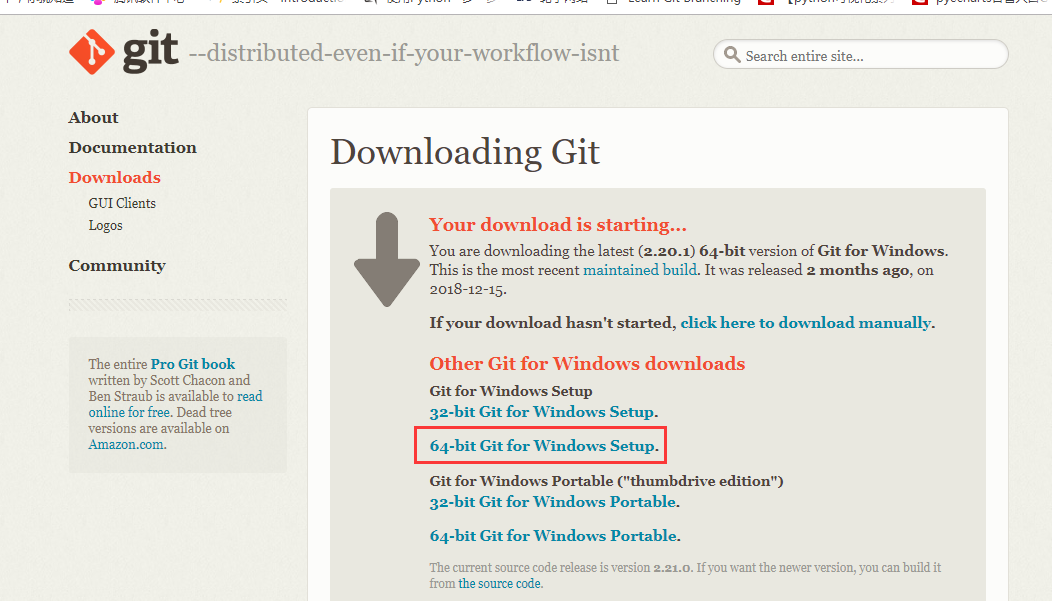
到官网选择下载：官网地址：https://git-scm.com/



选择windows版本



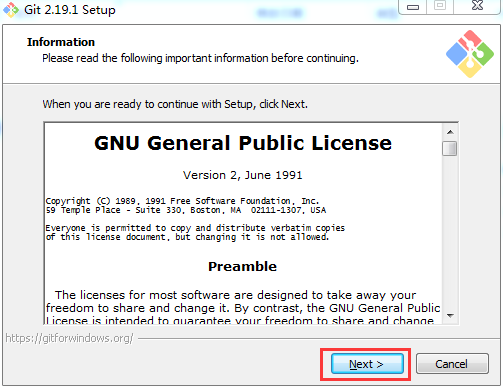
选择对应系统的版本

进行github客户端的安装

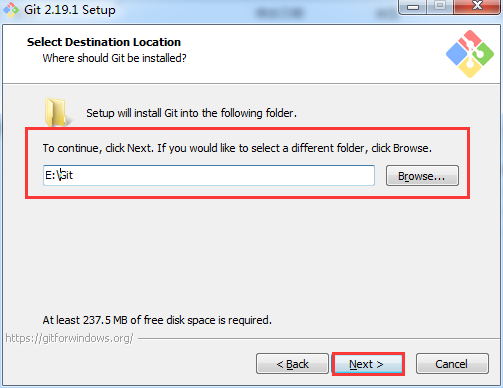
选择下载好的github客户端



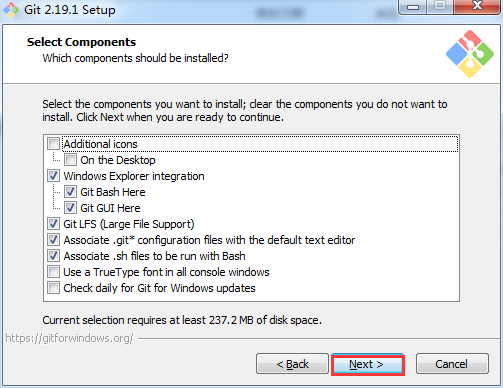
安装首先是阅读声明，这一步我们直接next



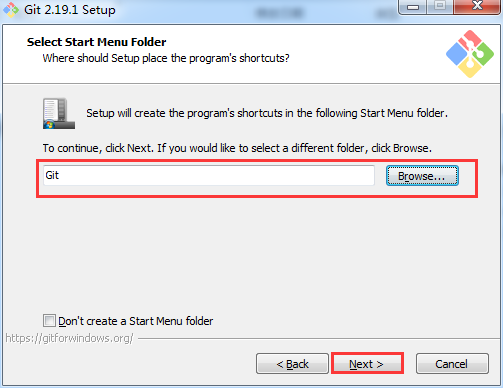
设置安装路径



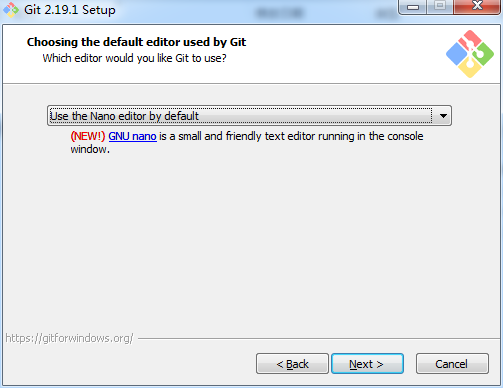
选择安装的组件



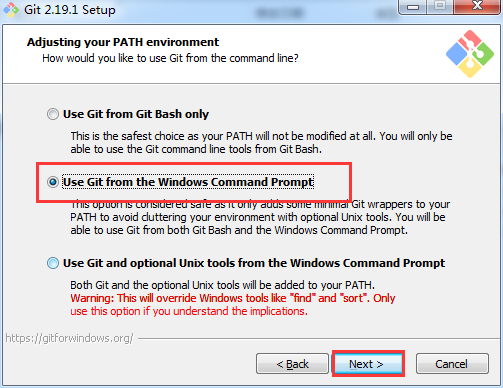
设置创建项目的目录



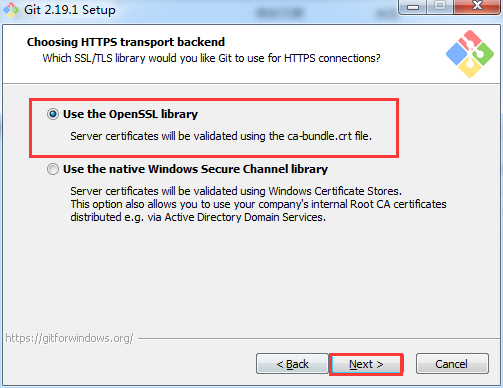
选择直接习惯的编辑器风格，这里我们选择默认



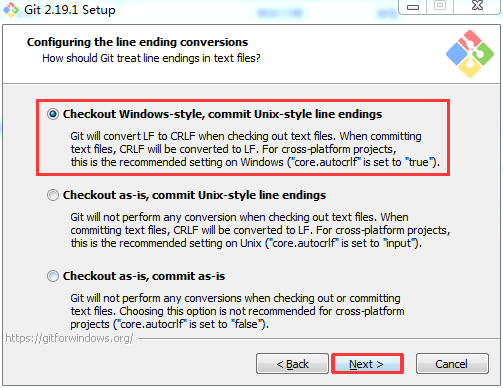
选择使用方式



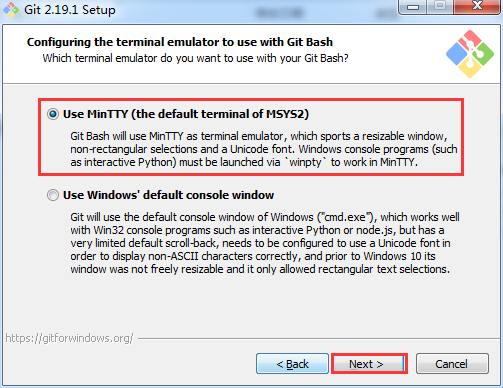
选择使用的库



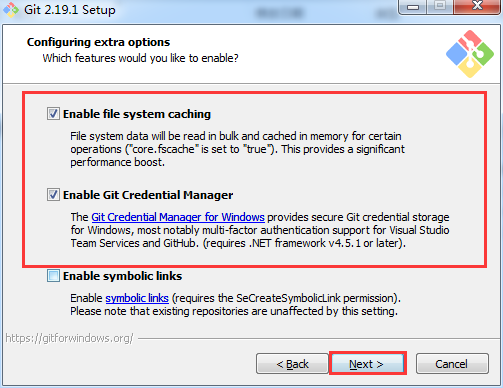
选择提交的风格



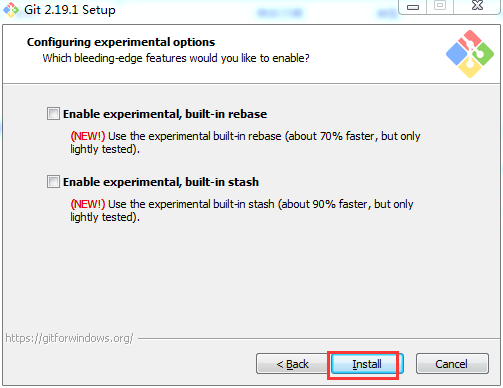
选择命令行操作的样式



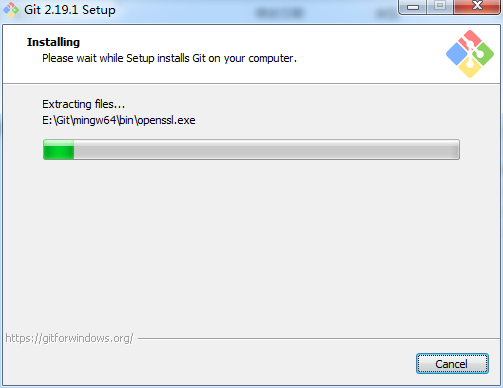
选择启用的git特性



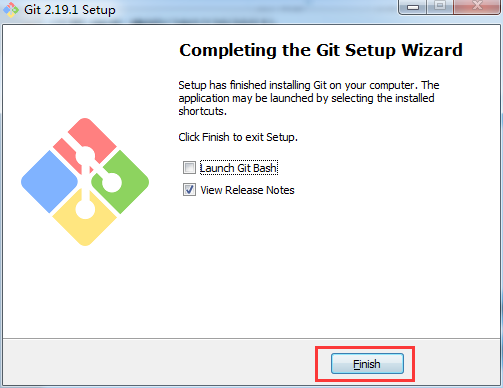
选择启用构建代码的特性，我们默认不选择



开始安装



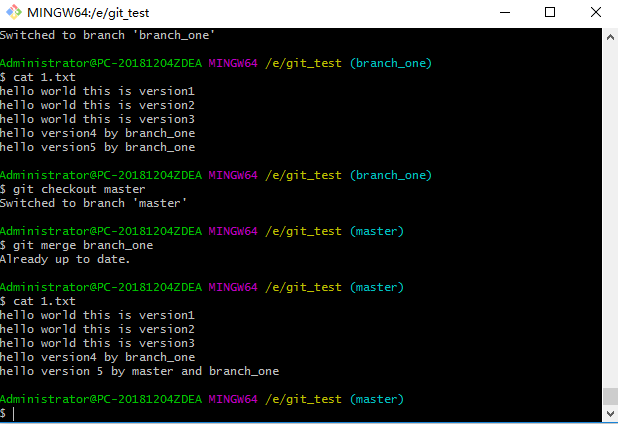
安装完成



完成之后，在任意目录下右键就会看到git的选项



我们大概Git Bash Here就可以在这里用命令行操作github了



安装好了之后就需要把我们的客户端和我们的github关联起来，关联的过程，类似大家了解ssh证书登录，首先要在客户端创建一个ssh key 这个的目的就是你现在需要在你电脑上获得一个密匙，就是咱们平时的验证码一样的东西，获取之后，在你的GitHub账号里边输入之后，你的电脑就和你的GitHub账号联系在一起了，这样以后就可以十分方便的通过Git bash 随时上传你的代码。

首先生产ssh-key:

1、设置全局用户名

bian@bian-PC MINGW64 /e/secondExample

$ git config --global user.name "While"

注意：这个全局name自己顺便起一个就好，这是代表本地的名字。

2、设置全局邮箱

bian@bian-PC MINGW64 /e/secondExample

$ git config --global user.email [2312483892@qq.com](mailto:2312483892@qq.com)

注意：这个邮箱是自己随便写个就好，这是代表本地的email。

3、生成秘钥

命令：ssh-keygen -t rsa

bian@bian-PC MINGW64 /e/secondExample

$ ssh-keygen -t rsa #执行命令

Generating public/private rsa key pair.

Enter file in which to save the key (/c/Users/bian/.ssh/id\_rsa): #保存key的文件位置

/c/Users/bian/.ssh/id\_rsa already exists.

Overwrite (y/n)? y #重写key

Enter passphrase (empty for no passphrase): #添加密码

Enter same passphrase again:

Your identification has been saved in /c/Users/bian/.ssh/id\_rsa.

Your public key has been saved in /c/Users/bian/.ssh/id\_rsa.pub.

The key fingerprint is:

SHA256:9iP58TLbu3d2L+ANZR3oLHwWZtvxGh/yaoL7tesQHRc bian@bian-PC

The key's randomart image is:

+---[RSA 2048]----+

| .E |

| = o.|

| . =.+o+|

| o.B\*oo|

| S .++o+.|

| . o + ...|

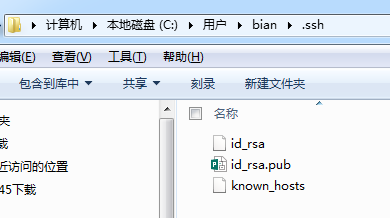
| o =o =. |

| +o\*+o= +|

| .==OB.++|

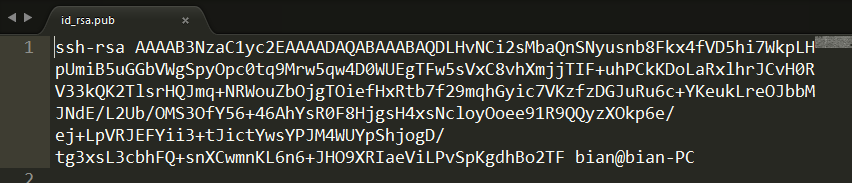
+----[SHA256]-----+

然后我们到默认的目录下，读这个key文件：/c/Users/bian/.ssh



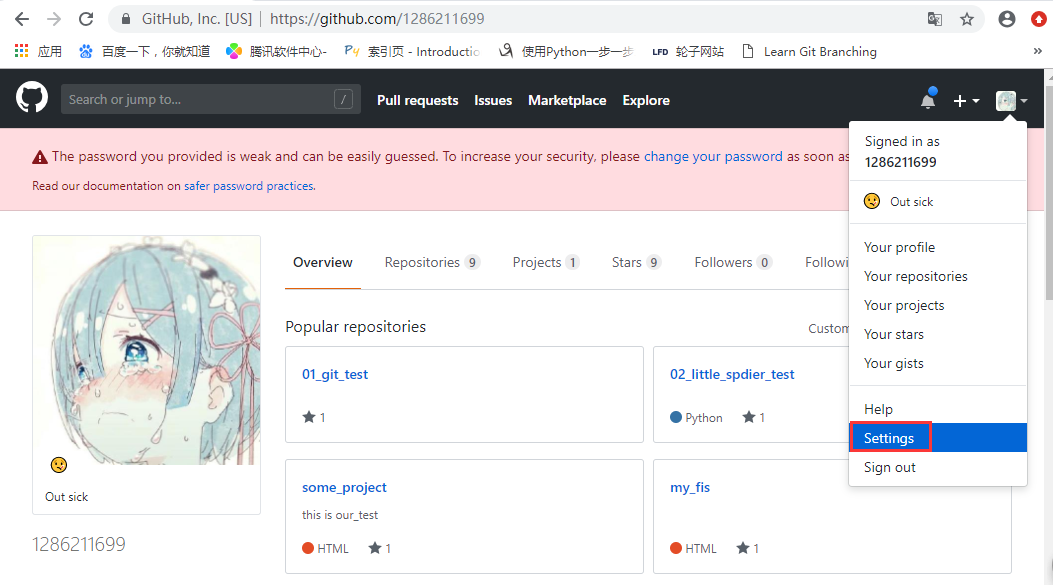
用pycharm或者sublime再或者其他的编译器打开id\_rsa.pub文件，

然后得到公钥：

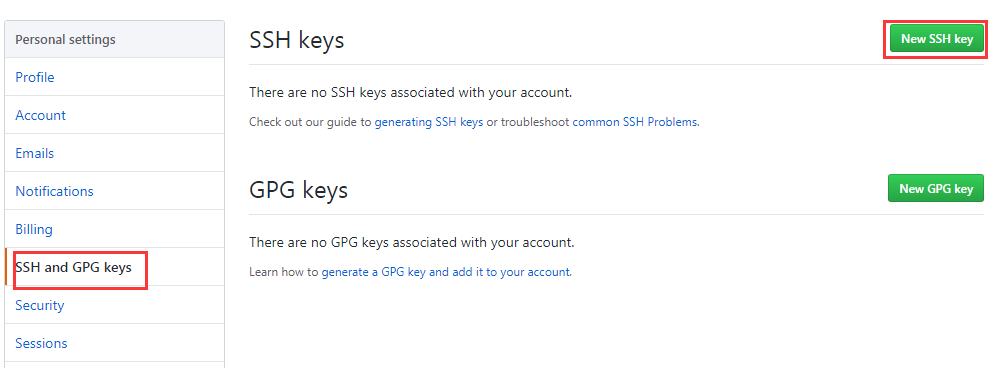


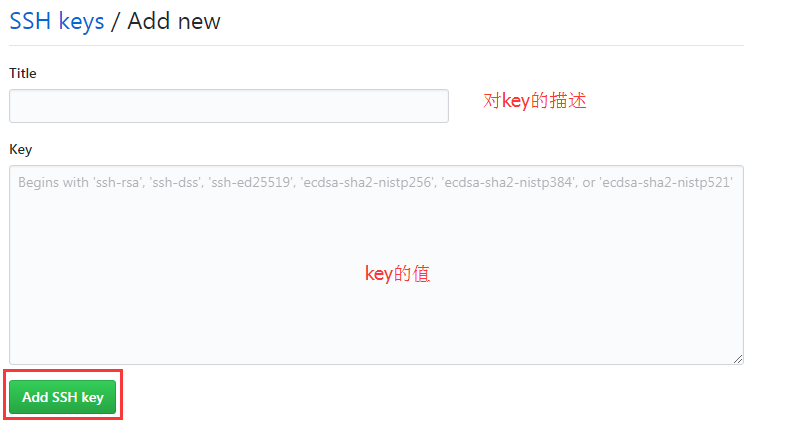
然后，将这个公钥，添加到github账号上

首先，进入github，然后在左上角的选项里面选择settings

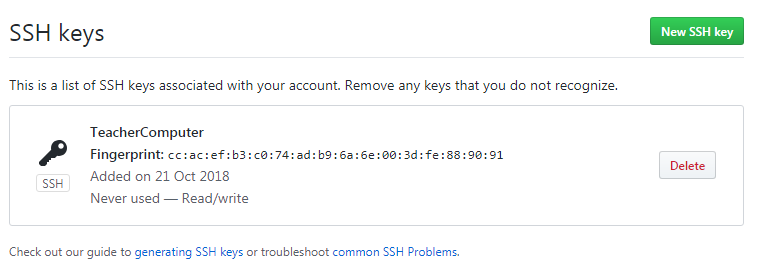


在settings当中接着选择绑定客户端



然后填入自己的key

选择添加之后会询问github的登录密码，输入，绑定成功



为什么要绑定呢？

为了去除每次推送项目，都要输入账号密码！

然后在客户端进行测试

命令：ssh -T git@github.com

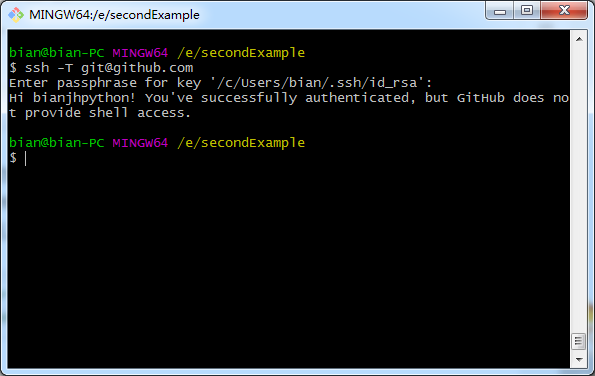
在这里需要你输入上面设置的密码然后有如下显示，证明成功了

bian@bian-PC MINGW64 /e/secondExample

$ ssh -T git@github.com

Enter passphrase for key '/c/Users/bian/.ssh/id\_rsa':

Hi bianjhpython! You've successfully authenticated, but GitHub does not provide shell access.



简单的分支操作

上面课程已经完成了github客户端和本地的关联，那么接下来我们以一个开发者的身份来参与到git的版本管理当中来。

1、上传项目

学习github，首先我们学会如何把自己当前的项目上传到git上面,在这里，我们要搞清楚一个道理，我们写的是项目，但是上传的是git 仓库，所以在上传项目之前需要先将目录转换为仓库，常用命令如下：

git init  #把这个目录变成Git可以管理的仓库

git add README.md  #文件添加到仓库

git add . #不但可以跟单一文件，还可以跟通配符，更可以跟目录。一个点就把当前目录下所有未追踪的文件全部add了

git commit -m "first commit"  #把文件提交到仓库

git remote add origin [git@github.com:wangjiax9/practice.git](mailto:git@github.com:wangjiax9/practice.git)  #关联远程仓库

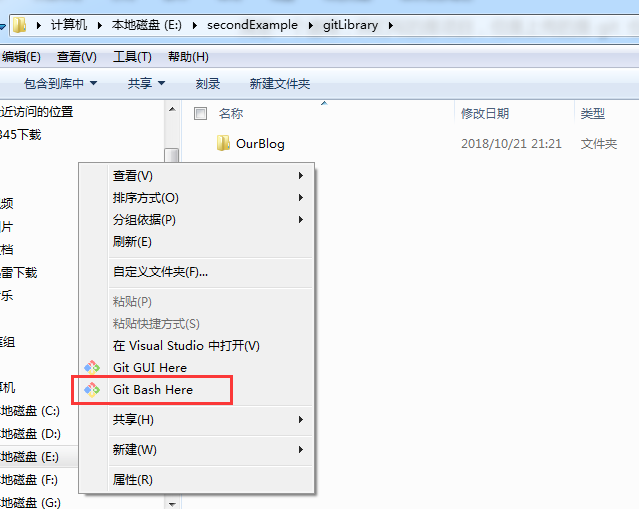
git push -u origin master #把本地库的所有内容推送到远程库上

我们以一个Django的博客项目为例子：

首先将博客项目放到我们要作为仓库的目录下



其次在目录下执行git bash命令行



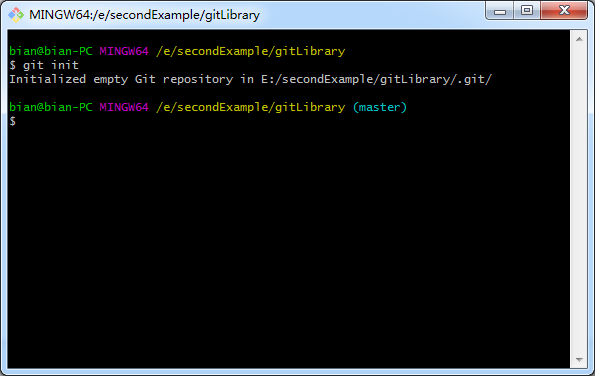
然后讲当前目录转换为一个git仓库

命令：git init

bian@bian-PC MINGW64 /e/secondExample/gitLibrary (master)

$ git init

Initialized empty Git repository in E:/secondExample/gitLibrary/.git/



这里会形成一个空的仓库，有一个隐性的目录.git,大家在看到之后，不要修改，会破坏仓库。

然后，添加内容到git库当中

命令：git add .

bian@bian-PC MINGW64 /e/secondExample/gitLibrary (master)

$ git add .

warning: LF will be replaced by CRLF in OurBlog/.idea/OurBlog.iml.

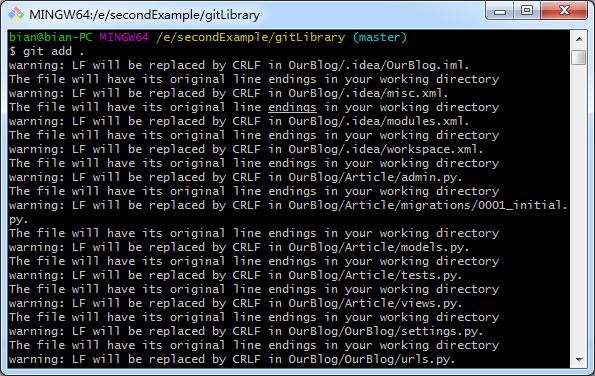
The file will have its original line endings in your working directory

warning: LF will be replaced by CRLF in OurBlog/.idea/misc.xml.

………

The file will have its original line endings in your working directory

注：这里出现一个 warning: LF will be replaced by CRLF in,这个错误无关大局，可以用git config core.autocrlf false命令关闭



添加完成之后，我们就可以将文件上传到我们的github上来了。

命令 git commit -m "Django学习案例博客" #引号里面的是描述

bian@bian-PC MINGW64 /e/secondExample/gitLibrary (master)

$ git commit -m "Django学习案例博客"

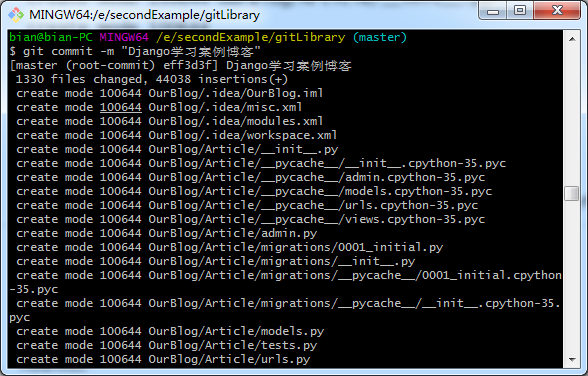
[master (root-commit) eff3d3f] Django学习案例博客

1330 files changed, 44038 insertions(+)

………

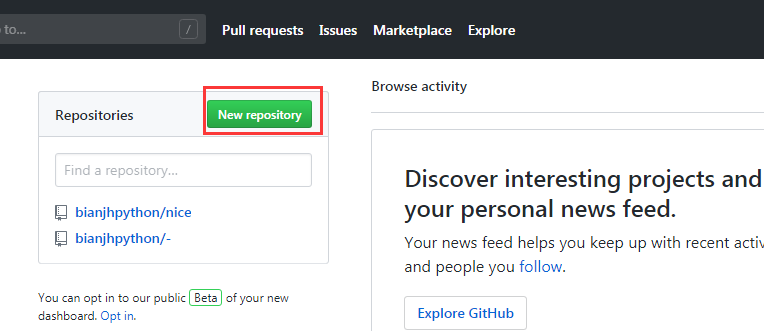
create mode 100644 OurBlog/template/myBlog.html

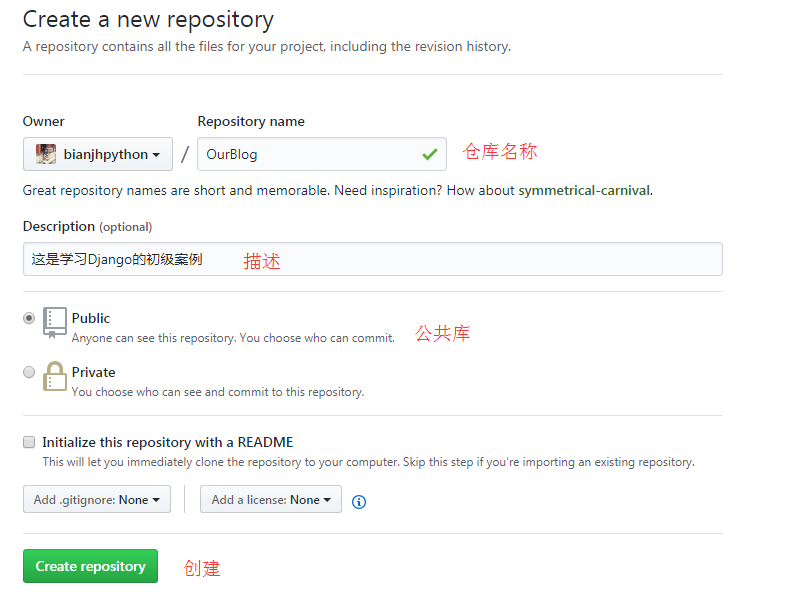
create mode 100644 OurBlog/template/myPicture.html



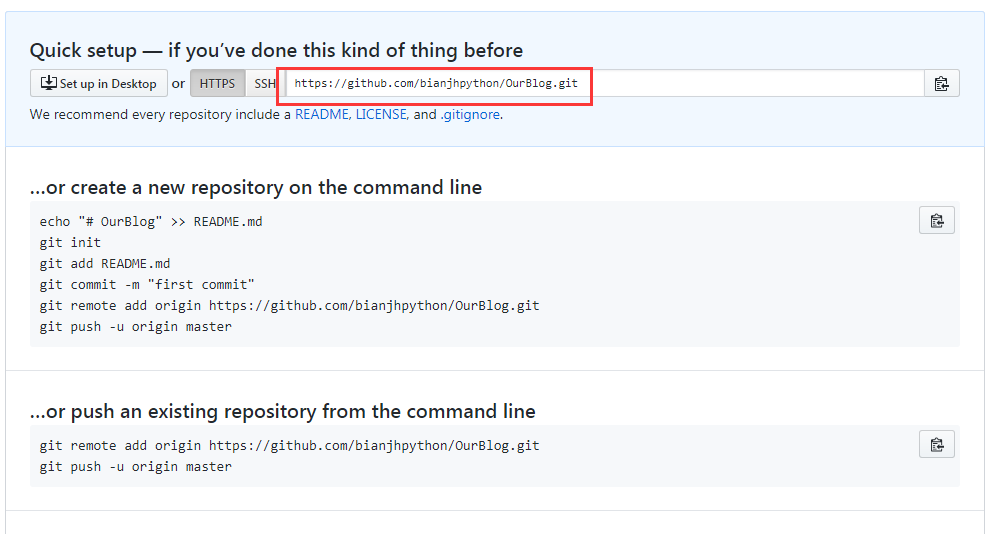
注意这个时候的代码还没有到我们的github上面，我们还需要下面几步

1、到github 创建仓库复制仓库地址





得到仓库的地址，下面也有基础的命令提示

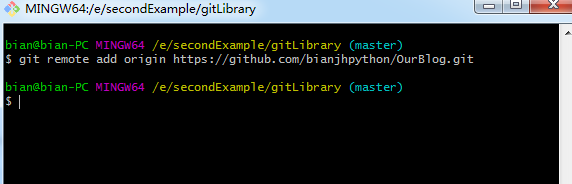


关联远程仓库

命令：git remote add origin https://github.com/bianjhpython/OurBlog.git

bian@bian-PC MINGW64 /e/secondExample/gitLibrary (master)

$ git remote add origin https://github.com/bianjhpython/OurBlog.git



然后上传代码

命令：git push -u origin master

bian@bian-PC MINGW64 /e/secondExample/gitLibrary (master)

$ git push -u origin master

Enumerating objects: 1475, done.

Counting objects: 100% (1475/1475), done.

Delta compression using up to 8 threads

Compressing objects: 100% (1392/1392), done.

Writing objects: 100% (1475/1475), 10.65 MiB | 2.05 MiB/s, done.

Total 1475 (delta 141), reused 0 (delta 0)

remote: Resolving deltas: 100% (141/141), done.

remote:

remote: Create a pull request for 'master' on GitHub by visiting:

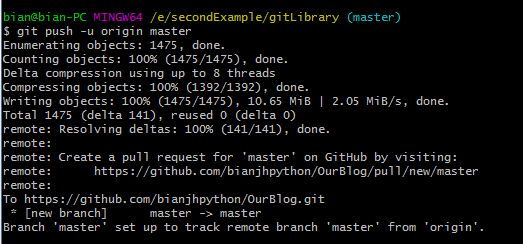
remote: https://github.com/bianjhpython/OurBlog/pull/new/master

remote:

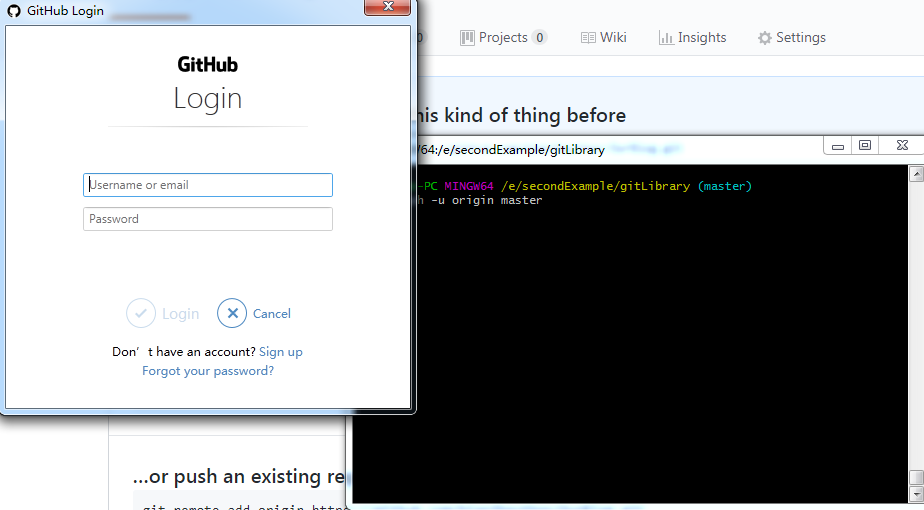
To https://github.com/bianjhpython/OurBlog.git

\* [new branch] master -> master

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



注：第一次上传需要要密码

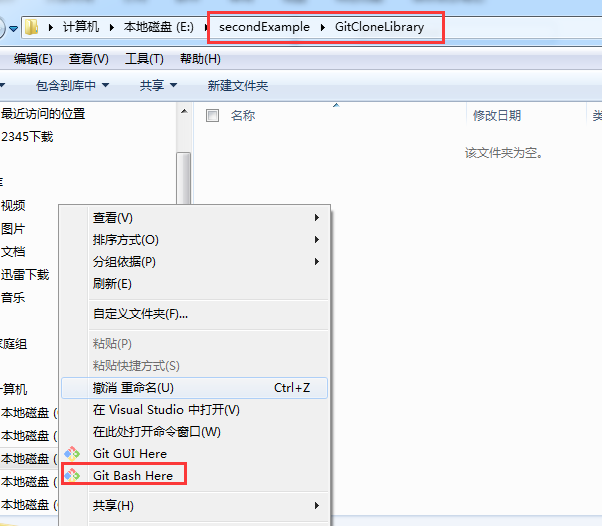


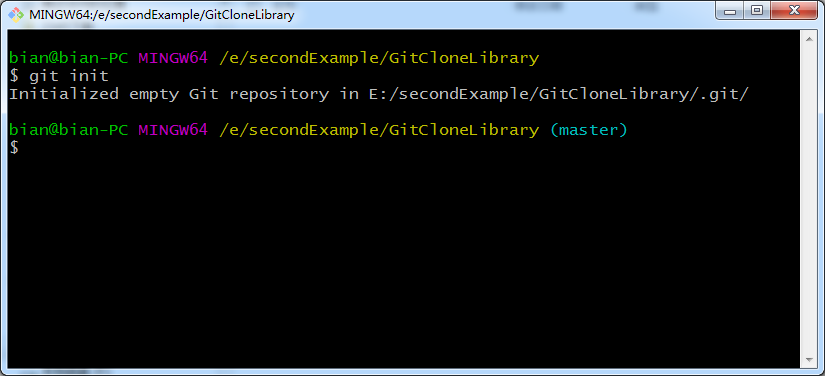
这样，我们就把项目上传到了github上面，但是这里要注意的是github无法管理空目录，所以上传空目录，github上面可能不显示。

2、克隆项目

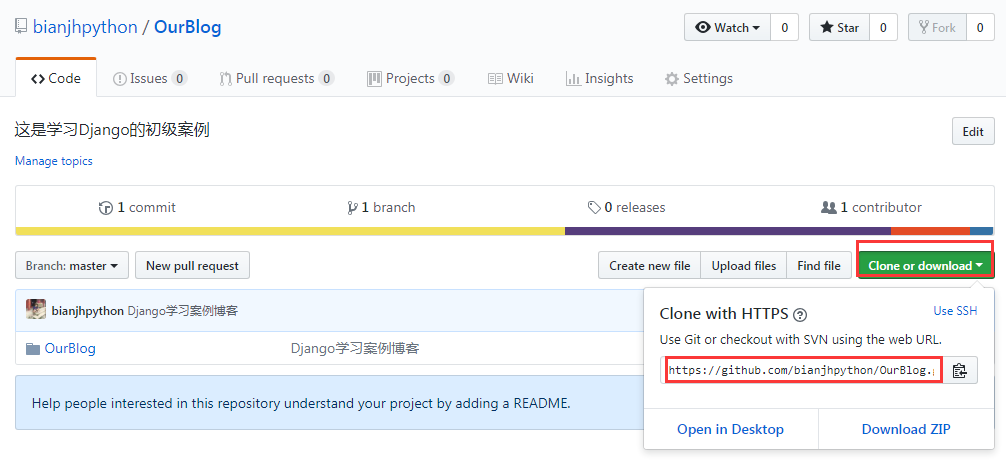
之前在讲github的时候，我们自己创建了一个项目，那么现在将这个项目克隆到本地。

我们首先创建一个目录，并且把他转换为git仓库





然后查看我们要克隆的仓库地址



然后执行命令进行克隆

命令：git clone <https://github.com/bianjhpython/OurBlog.git>

bian@bian-PC MINGW64 /e/secondExample/GitCloneLibrary (master)

$ git clone https://github.com/bianjhpython/OurBlog.git

Cloning into 'OurBlog'...

remote: Enumerating objects: 1475, done.

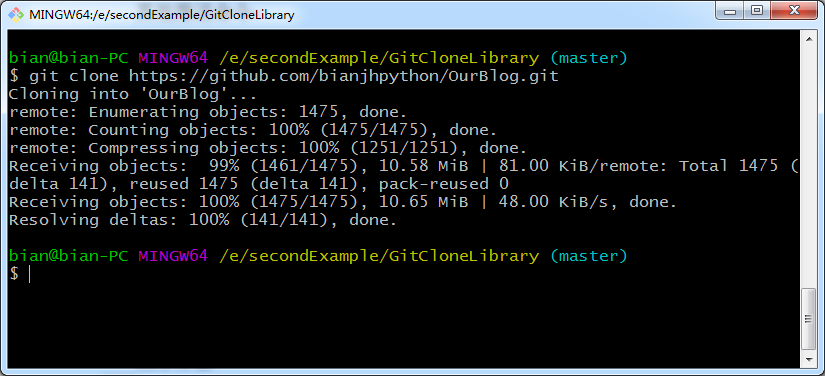
remote: Counting objects: 100% (1475/1475), done.

remote: Compressing objects: 100% (1251/1251), done.

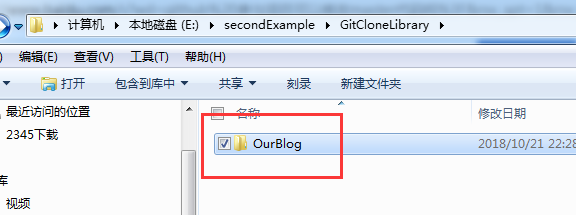
Receiving objects: 99% (1461/1475), 10.58 MiB | 81.00 KiB/remote: Total 1475 (delta 141), reused 1475 (delta 141), pack-reused 0

Receiving objects: 100% (1475/1475), 10.65 MiB | 48.00 KiB/s, done.

Resolving deltas: 100% (141/141), done.



效果如下：



1. 创建分支

几乎所有的版本控制系统都以某种形式支持分支。 使用分支意味着你可以把你的工作从开发主线上分离开来，以免影响开发主线。 在很多版本控制系统中，这是一个略微低效的过程——常常需要完全创建一个源代码目录的副本。对于大项目来说，这样的过程会耗费很多时间。

有人把 Git 的分支模型称为它的`‘必杀技特性’'，也正因为这一特性，使得 Git 从众多版本控制系统中脱颖而出。 为何 Git 的分支模型如此出众呢？ Git 处理分支的方式可谓是难以置信的轻量，创建新分支这一操作几乎能在瞬间完成，并且在不同分支之间的切换操作也是一样便捷。 与许多其它版本控制系统不同，Git 鼓励在工作流程中频繁地使用分支与合并，哪怕一天之内进行许多次。 理解和精通这一特性，你便会意识到 Git 是如此的强大而又独特，并且从此真正改变你的开发方式。

在使用之前，我们先来了解一个分支到底是干啥的

首先我们开发要有一个主干分支，叫做master，master分支主要做项目最终版本发布用，平时开发在分支内，开发完毕再和master分支合并。

当分支内文件被commit提交后，每个分支内只能看到各自分支内被commit提交的文件。比如在master里提交add1，在test分支提交add2，那么切换分支在master里只能看见add1，在test里只能看见add2，没提交的则都能看见。

当分支被删除后里面的文件也会一起被删除

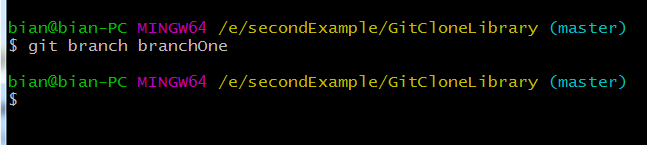
也就是说master分支是一个线上版本，大家都在基于这个版本开发，完成之后，进行版本的对比，合并，然后升级master，master分支一直处于一种稳定的状态，其他分支不断的升级，整合，然后同步到master。

明白分支的作用之后来学习常用的分支命令 。

在本地新建一个分支： git branch branchOne

bian@bian-PC MINGW64 /e/secondExample/GitCloneLibrary (master)

$ git branch branchOne



这里要注意：

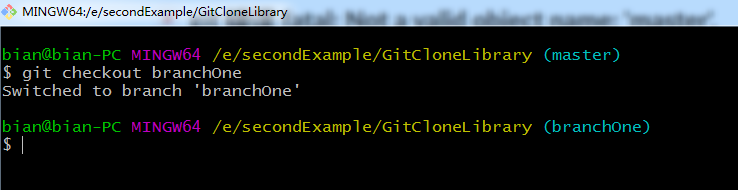
新建git项目，想用git branch串讲的时候可能会包出 fatal: Not a valid object name: master,这是因为刚刚创建的git仓库默认的master分支要在第一次commit之后才会建立，所以，我们需要先git add . 然后进行 git commit –m “first commit”

切换到你的新分支: git checkout branchOne

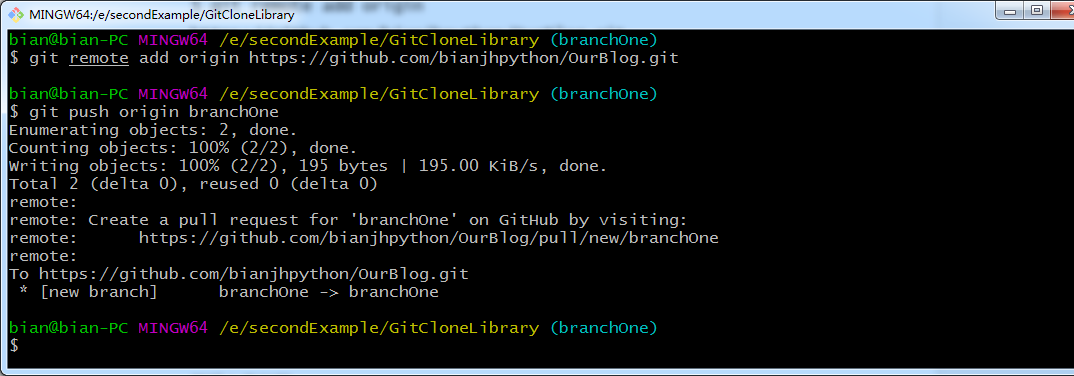
bian@bian-PC MINGW64 /e/secondExample/GitCloneLibrary (master)

$ git checkout branchOne

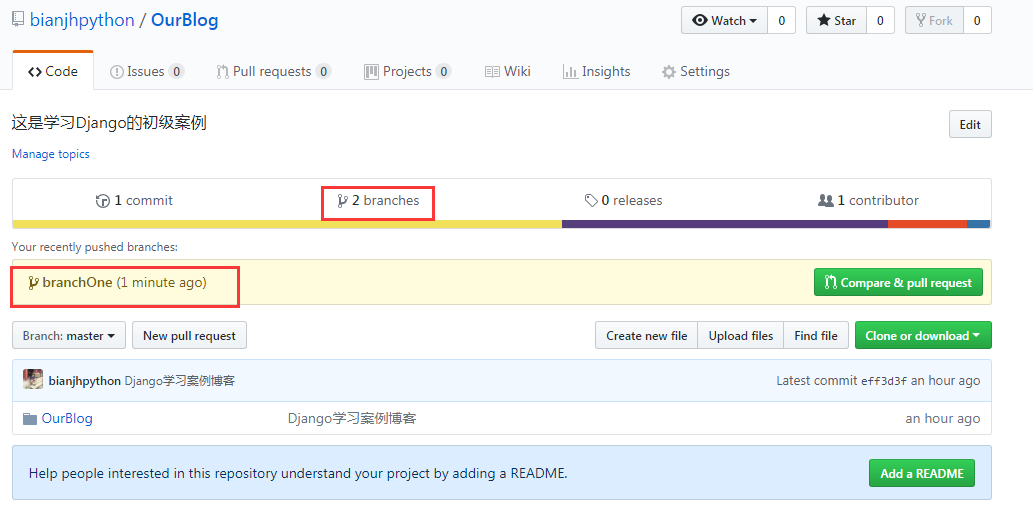
Switched to branch 'branchOne'



将新分支发布在github上： git push origin branchOne



效果已经出来了



但是在这里要注意，要想在github上发布分支，先确保和仓库进行了链接

然后我们尝试进行分支提交代码

bian@bian-PC MINGW64 /e/secondExample/GitCloneLibrary (branchOne)

$ git add 2.txt

bian@bian-PC MINGW64 /e/secondExample/GitCloneLibrary (branchOne)

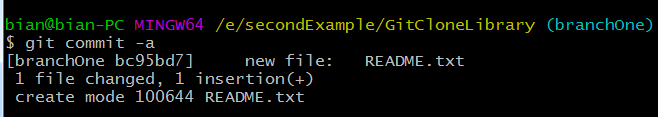
$ git commit -a

On branch branchOne

nothing to commit, working tree clean

bian@bian-PC MINGW64 /e/secondExample/GitCloneLibrary (branchOne)





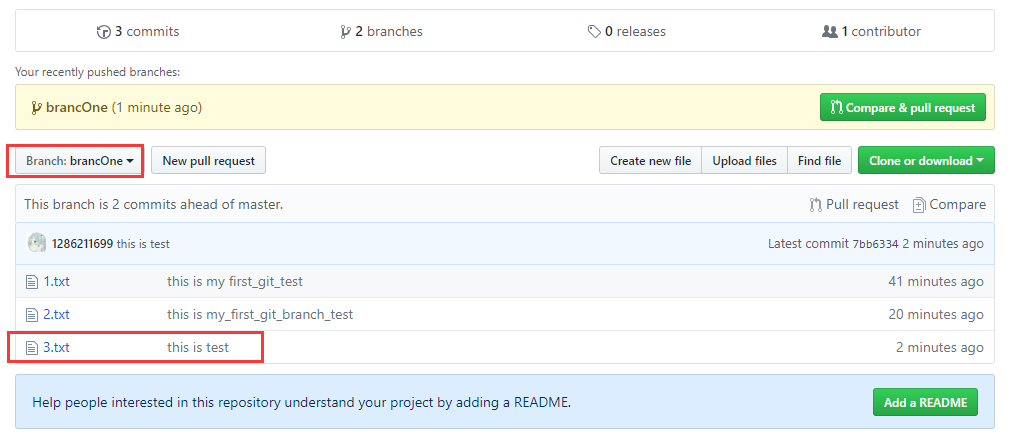
下面的命令同样的道理，大家可以多看看

在本地删除一个分支： git branch -d branchOne,

远程删除git服务器上的分支：git push origin -d BranchName

把项目推送到分支：

1. git add 2.txt
2. git commit –m ‘项目描述’2.txt
3. git push origin branchOne:branchOne



注意：可能出现以下错误，

error: failed to push some refs to 'https://github.com/1286211699/2019\_git\_test.

git'

如果出现这个错误，我们删除origin：git remote rm origin再进行普通的推送就好了

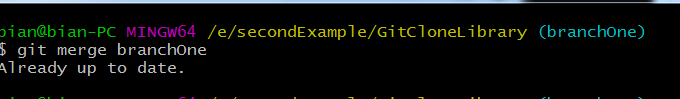
2、合并分支

在上面已经构建了一个master没有的readme文件出来，接下来进行合并

bian@bian-PC MINGW64 /e/secondExample/GitCloneLibrary (branchOne)

$ git merge branchOne

Already up to date.



总结：

21.1 版本控制介绍以及常用的版本控制工具

21.2 版本控制工具-GIT