

第一章 版本控制

本节所讲内容：

- 1.1 版本控制介绍以及常用的版本控制工具
- 1.2 版本控制工具-GIT 基础使用
- 1.3 git 使用实例

1.1 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、配置文件及说明文档等文件变更的管理，是软件配置管理的核心思想之一。

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。

2、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。每次版本控制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。

1.2 版本控制工具-GIT

Git 历史和原理

首先，我们聊聊 git 的历史,谈到 git 必须要谈到 linux 之父 Linus Torvalds 和 BitKeeper 的公司 BitMover



Linus Torvalds



BitKeeper

起初参与 Linux 开源项目的代码是由 Linus 本人通过“diff”和“patch”命令来手动为别人整合代码的，之后正如在《Pro Git》这本书中所讲到的，随着项目越做越大，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。直到 2005 年，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：

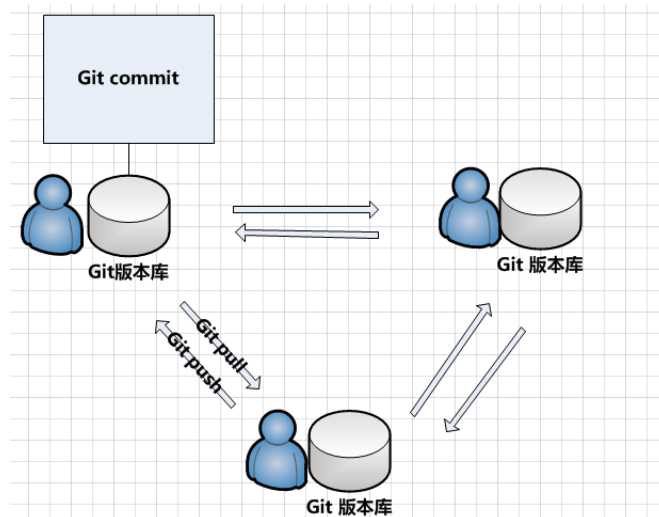


Linus 花了两周时间自己用 C 写了一个分布式版本控制系统，这就是 Git！一个月之内，Linux

系统的源码已经由 Git 管理了！Git 也由此诞生了。然而牛逼的人生从来不需要解释这句话被 Linus 展现的淋漓尽致.....

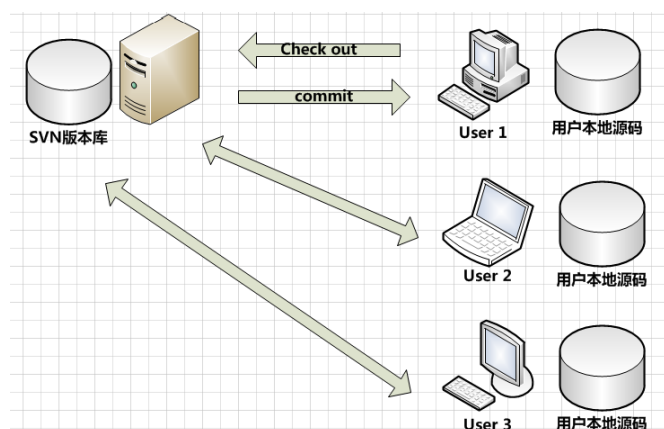
说完了 git 的历史,我们来看看 git 的原理,在 git 之前,我们使用的最多的版本控制工具是 SVN (当然现在也有好多人在用)。

Git 采用的是分布式的管理系统,



简单的说就是 Git 采用了分布式系统,每个用户都拥有独立的版本库,假如我们没有网也可以进行独立的版本控制,只是不能和其他人同步。

SVN 采用的是分布式的管理系统。



同样,SVN 采用集中式,客户端的代码集中在一台服务器上进行管理,如果失去网络,就失去效果了。

了解到这里,好多同学可能会认为 git 先进,git 牛,但是实际上我们回头看,svn 也许有点笨拙,但实际上在代码集中管理上还是有优势的,所以,任何技术没有先进和落后,只有对应自己工作的需求,才可以选择要用的技术。

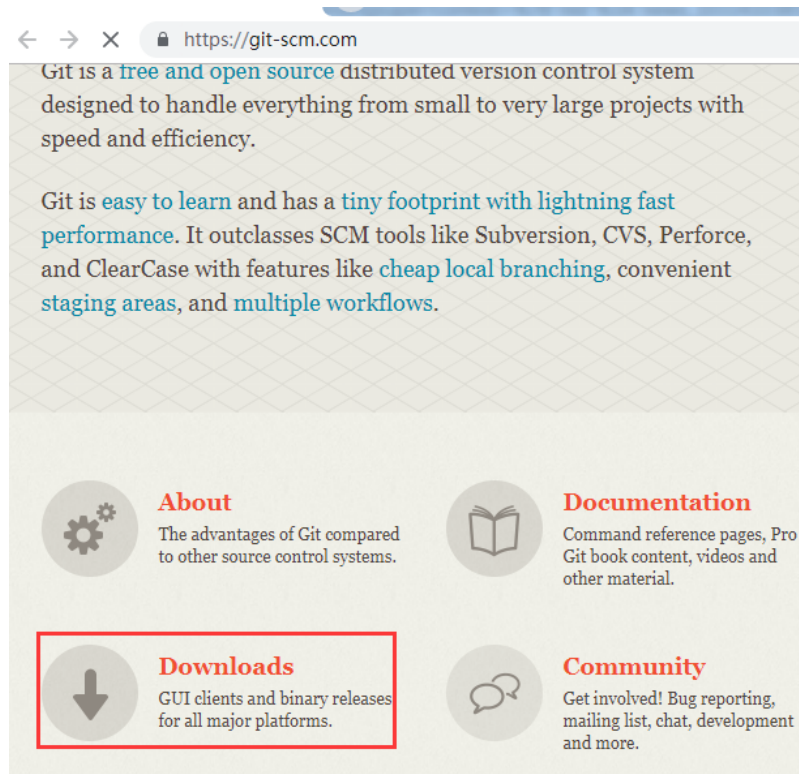
Git 的本地操作

Git 客户端的安装

1、 下载、安装客户端

到官网选择下载

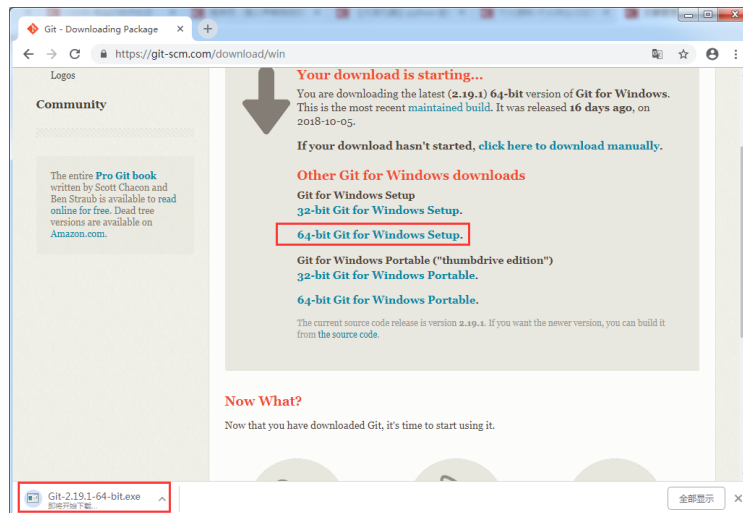
官网地址：<https://git-scm.com/>



选择 windows 版本



选择对应系统的版本

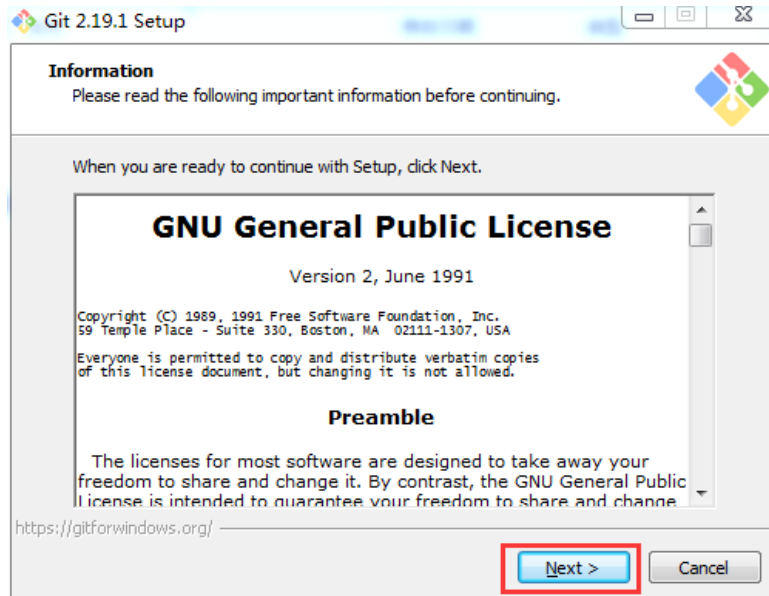


进行 github 客户端的安装

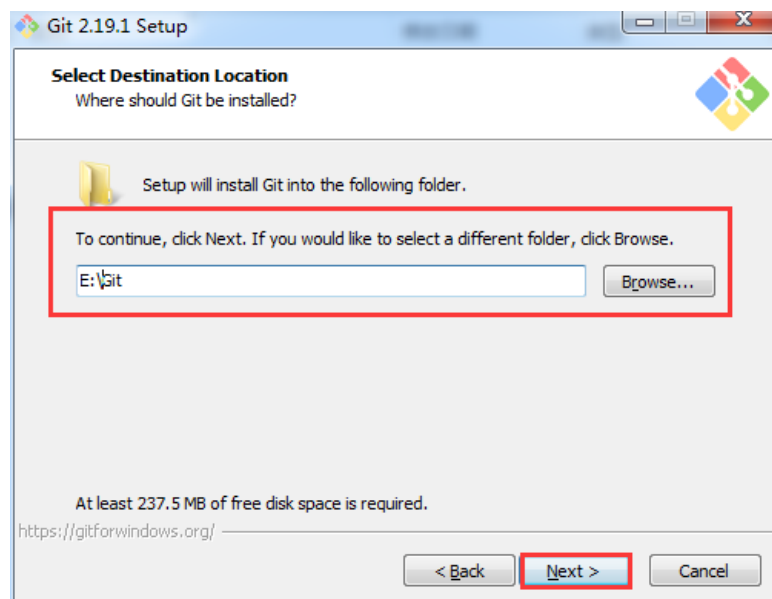
选择下载好的 github 客户端

Git-2.19.1-64-bit.exe 2018/10/21 15:54 应用程序 42,433 KB

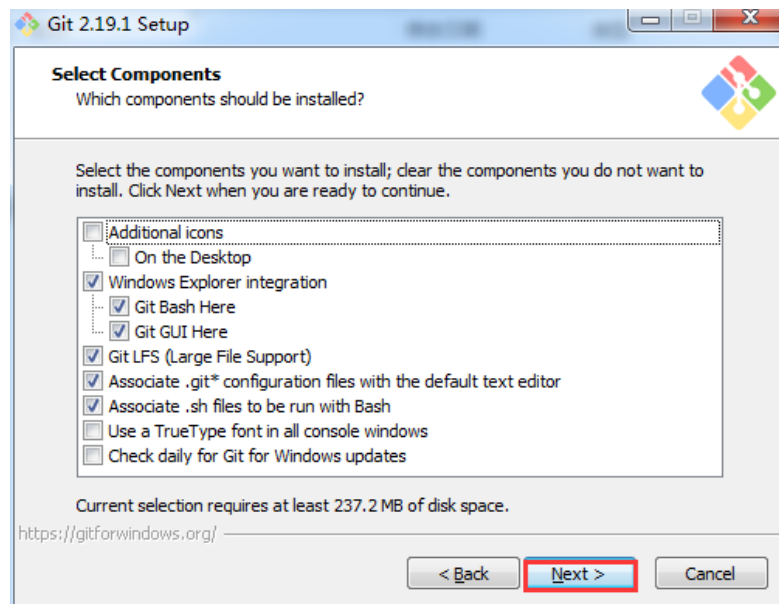
安装首先是阅读声明，这一步我们直接 next



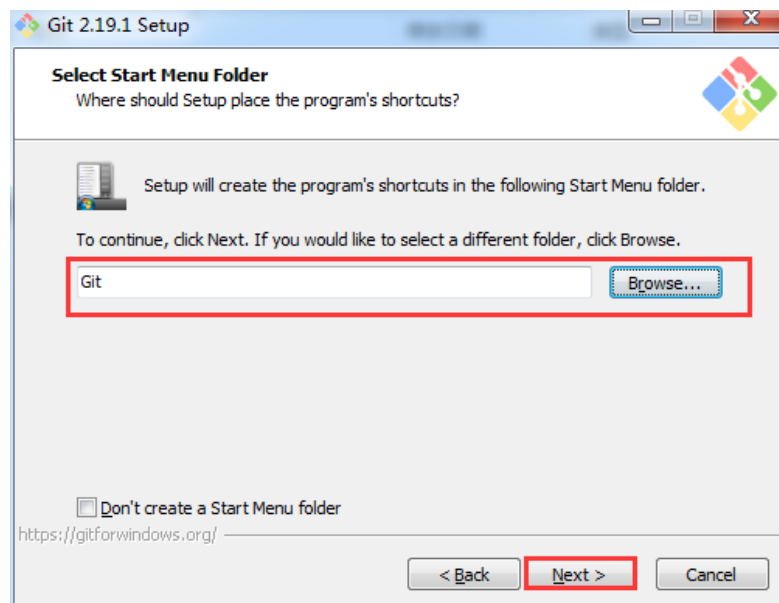
设置安装路径



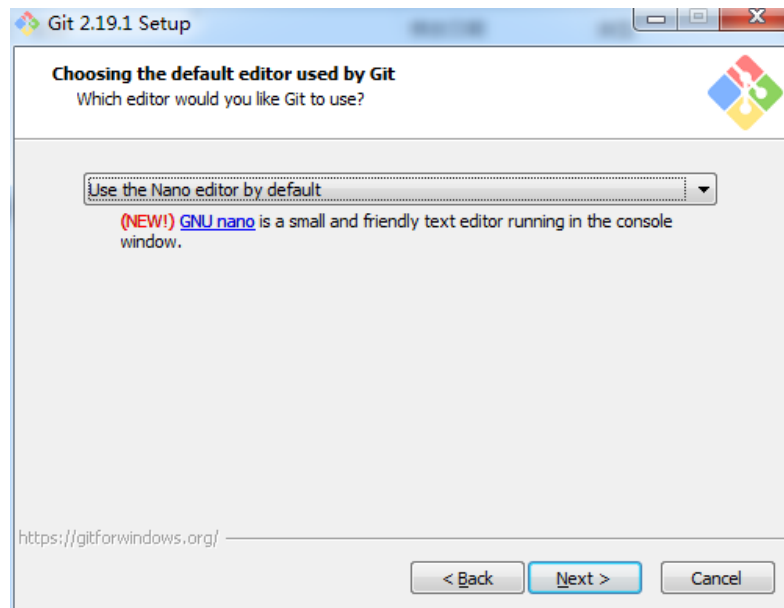
选择安装的组件



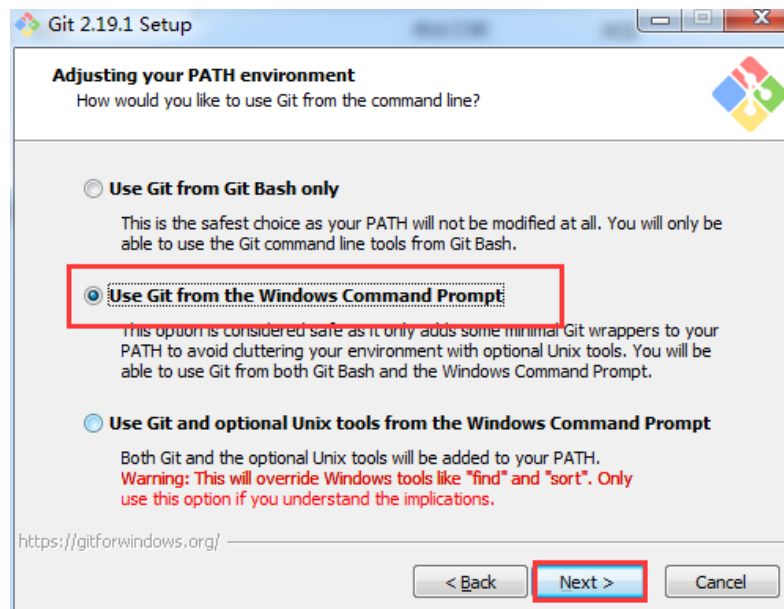
设置创建项目的目录



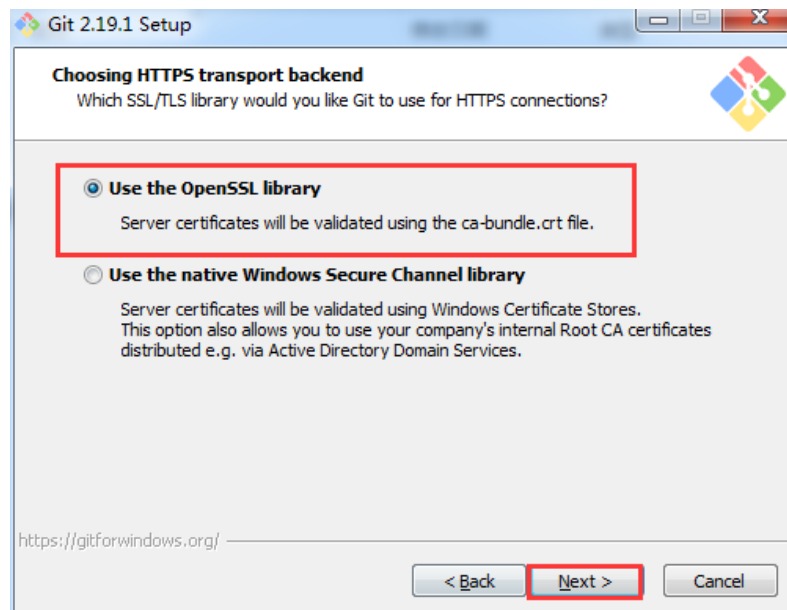
选择直接习惯的编辑器风格，这里我们选择默认



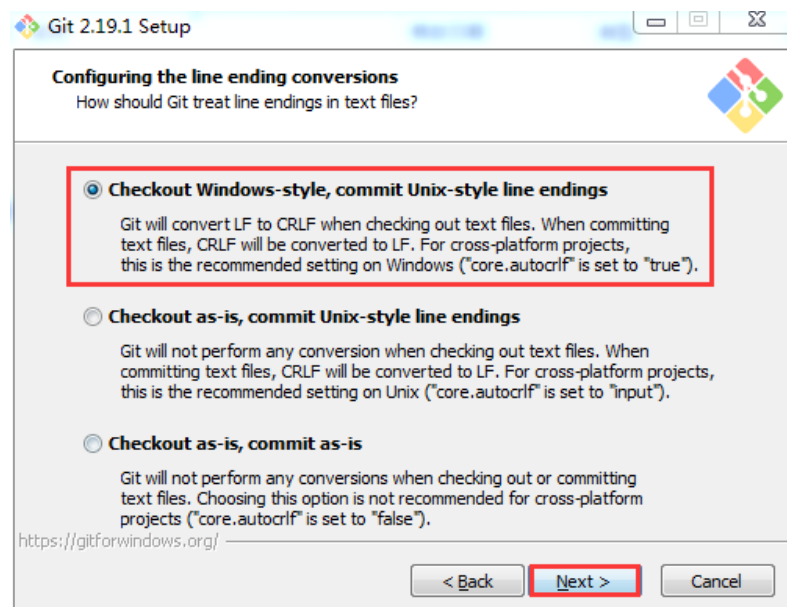
选择使用方式



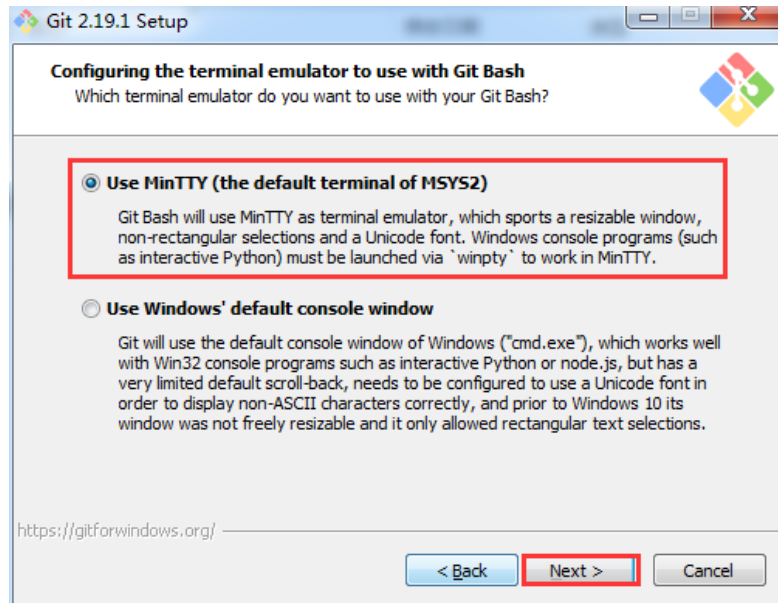
选择使用的库



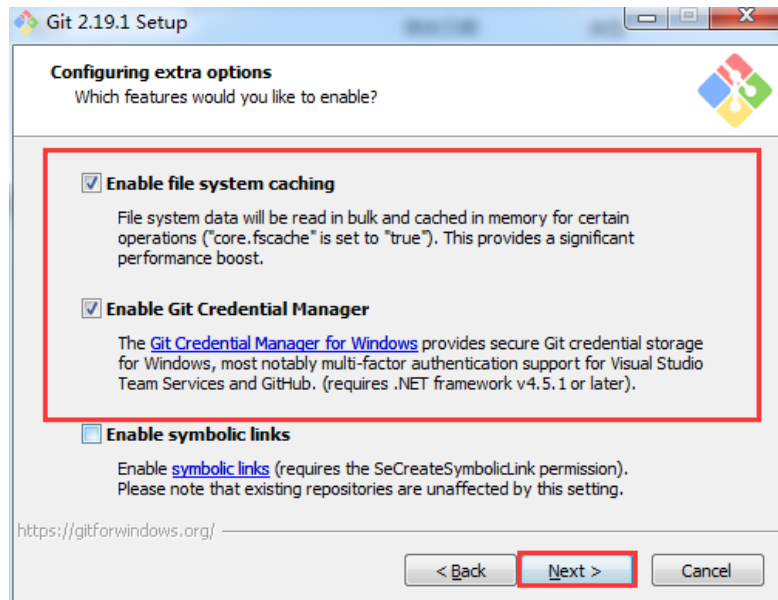
选择提交的风格



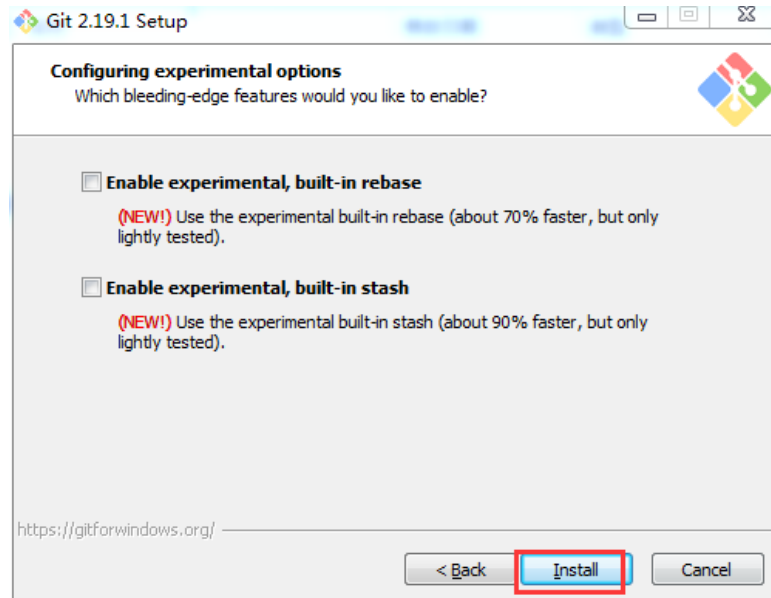
选择命令行操作的样式



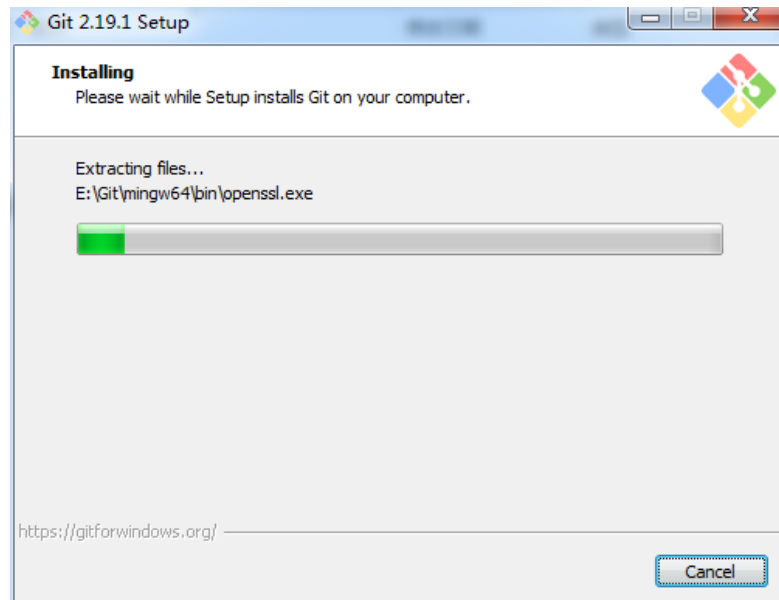
选择启用的 git 特性



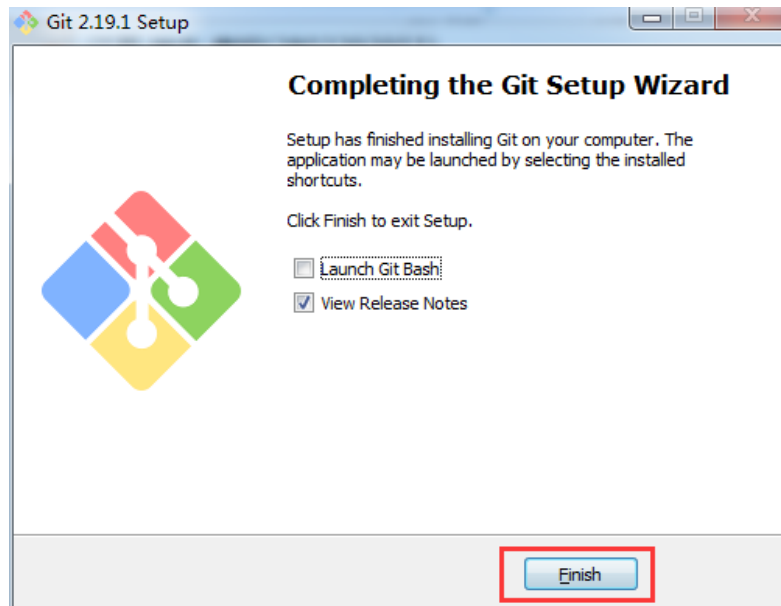
选择启用构建代码的特性，我们默认不选择



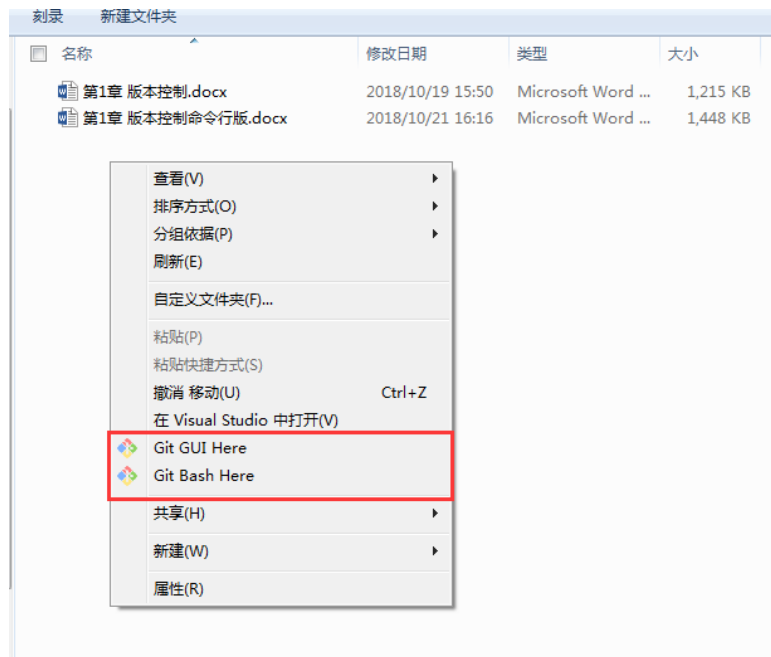
开始安装



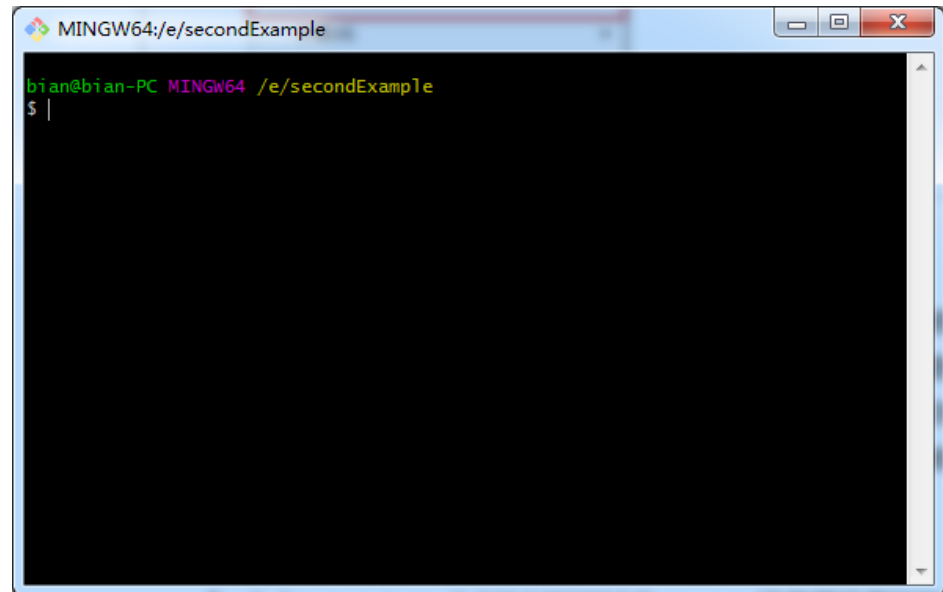
安装完成



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



完成了客户端的安装，好多同学认为会直接开始 github 的教学了，其实，不然，在上面 git 原理上说过，我们在本地也可以进行 git 版本控制。所以我们首先要开始的是 git 本地的使用。

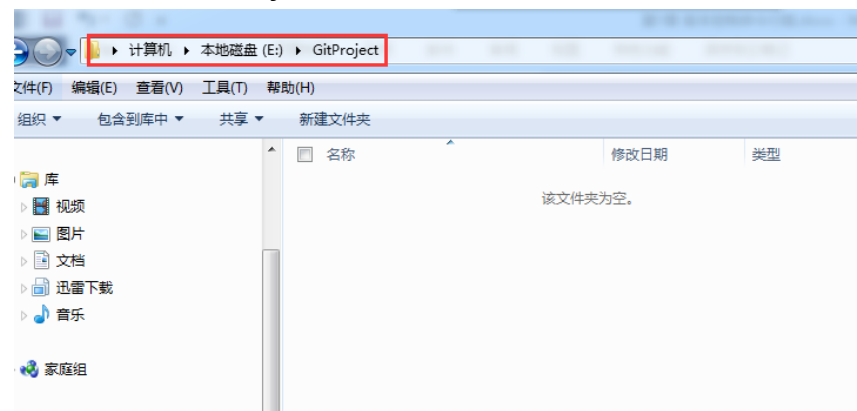
Git 本地库操作：

首先，我们需要在本地创建直接的 git 目录，我们要进行版本管理的的项目就要放在这个目录下。今天教学，我们从无到有。

1、创建目录，

我在 E 盘创建了 git 的项目目录，当然大家也可以创建在别的地方，但是在创建的路径上，不要有中文，这样容易导致乱码。

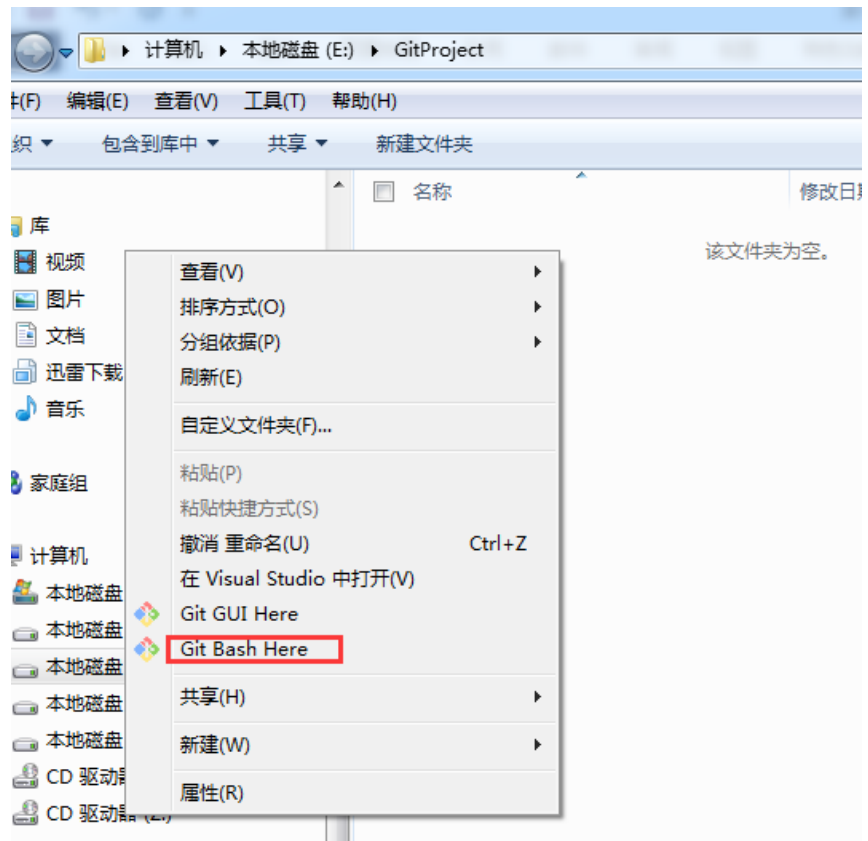
创建项目路径 E:\GitProject



2、设置版本库

在上面我们已经创建了目录，但是这个目录只是一个正常的 windows 目录，如果想要转换为 git 的目录还需要进行一次构建。

右键 → 选择 Git Bash Here



这样我们打开了一个当前目录的命令窗口，但是不要惊讶，这里面我们可以执行 linux 的基础命令。

```
MINGW64:/e/GitProject
bian@bian-PC MINGW64 /e/GitProject
$ ls

bian@bian-PC MINGW64 /e/GitProject
$ |
```

然后执行 git 库初始化的命令
命令：git init

```
MINGW64:/e/GitProject
bian@bian-PC MINGW64 /e/GitProject
$ git init
Initialized empty Git repository in E:/GitProject/.git/
bian@bian-PC MINGW64 /e/GitProject (master)
$
```

这样我们就创建了一个 git 本地仓库，下面的语句大概的意思就是安装了一个空的 git 仓库在当前的目录的.git 目录下，按照正常的 linux 习惯，.git 目录是隐藏目录，用 linux 的 ls -a 看一下：

```
MINGW64:/e/GitProject
bian@bian-PC MINGW64 /e/GitProject (master)
$ ls -a
./ ../ .git/
bian@bian-PC MINGW64 /e/GitProject (master)
$
```

果然，有货，这里就是我们整个 git 版本库的配置文件：

```
MINGW64:/e/GitProject
bian@bian-PC MINGW64 /e/GitProject (master)
$ ll -a .git
total 11
drwxr-xr-x 1 bian 197121 0 十一月 6 11:18 ./
drwxr-xr-x 1 bian 197121 0 十一月 6 11:18 ../
-rw-r--r-- 1 bian 197121 130 十一月 6 11:18 config
-rw-r--r-- 1 bian 197121 73 十一月 6 11:18 description
-rw-r--r-- 1 bian 197121 23 十一月 6 11:18 HEAD
drwxr-xr-x 1 bian 197121 0 十一月 6 11:18 hooks/
drwxr-xr-x 1 bian 197121 0 十一月 6 11:18 info/
drwxr-xr-x 1 bian 197121 0 十一月 6 11:18 objects/
drwxr-xr-x 1 bian 197121 0 十一月 6 11:18 refs/

bian@bian-PC MINGW64 /e/GitProject (master)
$ |
```

config:这个是 Git 仓库的配置文件

description:仓库的描述信息，主要给 gitweb 等 git 托管系统使用

HEAD:这个文件包含了一个当前分支（branch）的引用，通过这个文件 Git 可以得到下一次 commit 的 parent

hooks:这个目录存放一些 shell 脚本，可以设置特定的 git 命令后触发相应的脚本；在搭建 gitweb 系统或其他 git 托管系统会经常用到 hook script

info:包含仓库的一些信息

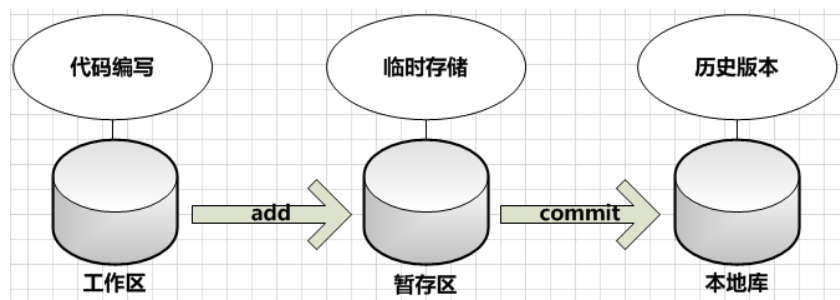
objects:所有的 Git 对象都会存放在这个目录中，对象的 SHA1 哈希值的前两位是文件夹名称，后 38 位作为对象文件名

refs:这个目录一般包括三个子文件夹，heads、remotes 和 tags，heads 中的文件标识了项目中的各个分支指向的当前 commit

哈哈哈哈哈，现在你就拥有了一个属于自己的 git 仓库，尽管你啥也不会操作

3、Git 本地的结构

现在我们拥有了一个完整的 git 仓库，那么记下来尝试了解一下他的结构。



在本地，我们的 git 实际上是图上的这样的结构的，我们编写完的代码先 add 到缓存区，然后通过 commit 提交，当然，在这里先简单的带着大家认识一下，下面开始详细的命令讲解。

4、创建 git 执行用户

上面完成了一个 git 本地库的构造，但是，我们提交代码需要一个身份，所以下下来我们需要构建我们的身份。

配置当前仓库当中的身份：

配置开发者姓名：git config user.name while

配置开发者邮箱（符合格式即可）：git config user.email while@qq.com

```
MINGW64:/e/GitProject
bian@bian-PC MINGW64 /e/GitProject (master)
$ git config user.name while

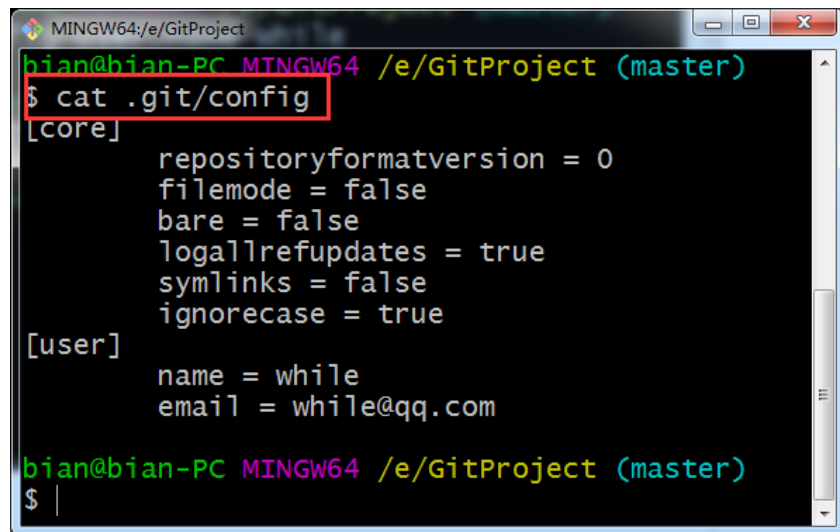
bian@bian-PC MINGW64 /e/GitProject (master)
$ git config user.email while@qq.com

bian@bian-PC MINGW64 /e/GitProject (master)
$ |
```

The screenshot shows a terminal window with the title 'MINGW64:/e/GitProject'. The user 'bian' is at the 'bian-PC' prompt. They are in the 'MINGW64' shell at the directory '/e/GitProject' on the '(master)' branch. They have entered two 'git config' commands: 'git config user.name while' and 'git config user.email while@qq.com'. The prompt '\$' is shown again, indicating the commands were executed successfully.

然后看一下

命令 `cat .git/config`

A terminal window titled 'MINGW64:/e/GitProject' showing the output of the command 'cat .git/config'. The output displays the configuration for the core and user sections. The 'core' section includes repositoryformatversion, filemode, bare, logallrefupdates, symlinks, and ignorecase. The 'user' section includes name and email.

```
bian@bian-PC MINGW64 /e/GitProject (master)
$ cat .git/config
[core]
    repositoryformatversion = 0
    filemode = false
    bare = false
    logallrefupdates = true
    symlinks = false
    ignorecase = true
[user]
    name = while
    email = while@qq.com

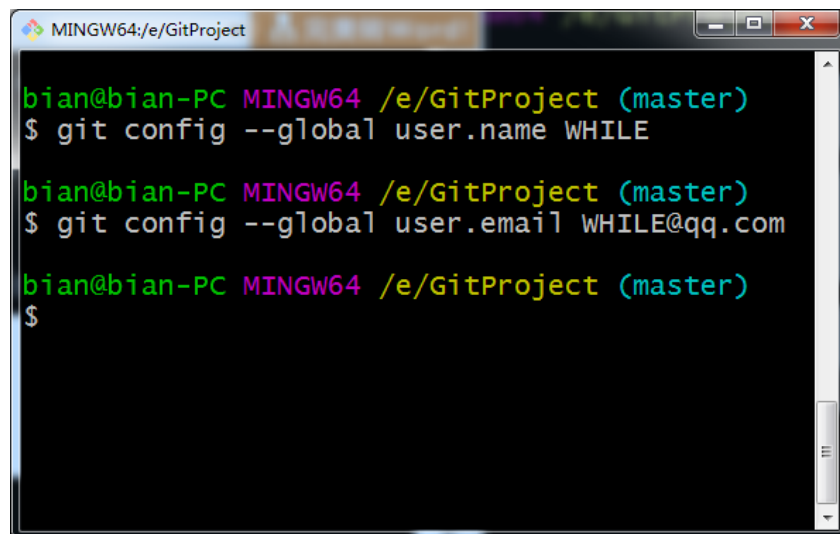
bian@bian-PC MINGW64 /e/GitProject (master)
$
```

配置全局的身份

当然，如果在当前计算机的所有项目当中都想要用统一的用户名和邮箱（支持）

配置开发者姓名：`git config --global user.name while`

配置开发者邮箱（符合格式即可）：`git config --global user.email while@qq.com`

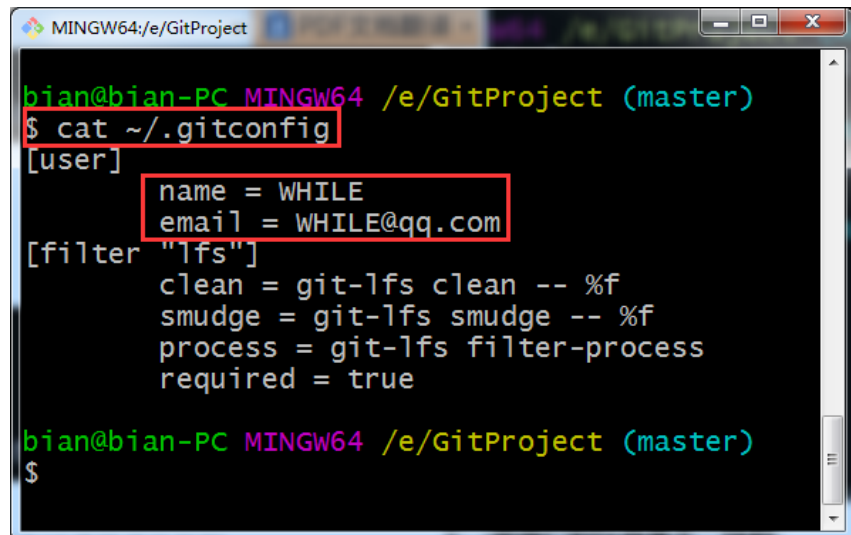
A terminal window titled 'MINGW64:/e/GitProject' showing three commands being executed to set global git configuration. The first command sets the user name to 'WHILE', the second sets the user email to 'WHILE@qq.com', and the third shows the prompt character.

```
bian@bian-PC MINGW64 /e/GitProject (master)
$ git config --global user.name WHILE

bian@bian-PC MINGW64 /e/GitProject (master)
$ git config --global user.email WHILE@qq.com

bian@bian-PC MINGW64 /e/GitProject (master)
$
```

接着查看，git 存放全局和本地用户的位置是不一样的

A terminal window titled 'MINGW64:/e/GitProject' showing the command 'cat ~/.gitconfig' and its output. The output displays the configuration for the user 'WHILE' with email 'WHILE@qq.com' and the 'lfs' filter settings. The user 'bian' is at the 'bian-PC' machine in the 'MINGW64' shell, currently in the '/e/GitProject' directory on the 'master' branch.

```
bian@bian-PC MINGW64 /e/GitProject (master)
$ cat ~/.gitconfig
[user]
  name = WHILE
  email = WHILE@qq.com
[filter "lfs"]
  clean = git-lfs clean -- %f
  smudge = git-lfs smudge -- %f
  process = git-lfs filter-process
  required = true

bian@bian-PC MINGW64 /e/GitProject (master)
$
```

优先级别

就近原则：项目级别优先于系统用户级别，二者都有时采用项目级别的签名

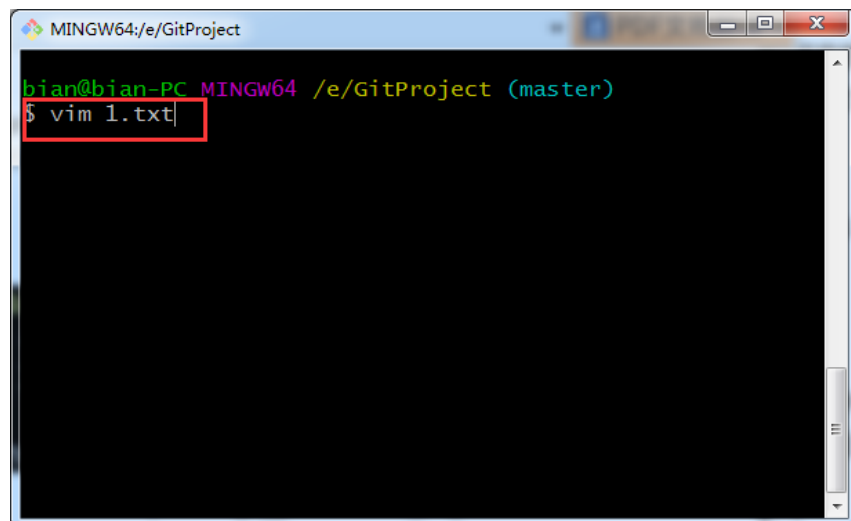
如果只有系统用户级别的签名，就以系统用户级别的签名为准

二者都没有不允许

5、执行基本的脚本提交

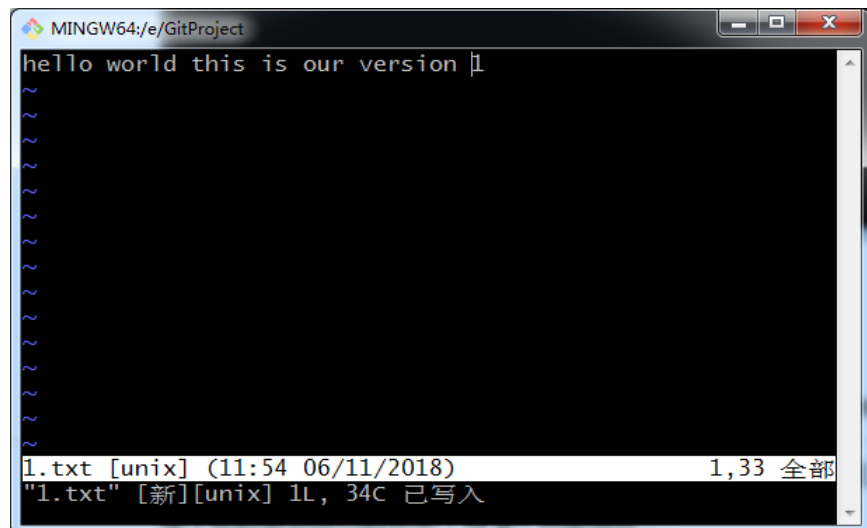
上面了解了本地 git 的结构，接下来我们来具体的提交一下脚本；

在我们的 git 项目目录下创建一个 1.txt，然后输入内容

A terminal window titled 'MINGW64:/e/GitProject' showing the command 'vim 1.txt' being entered. The user 'bian' is at the 'bian-PC' machine in the 'MINGW64' shell, currently in the '/e/GitProject' directory on the 'master' branch.

```
bian@bian-PC MINGW64 /e/GitProject (master)
$ vim 1.txt
```

然后编写：hello world this is our version 1

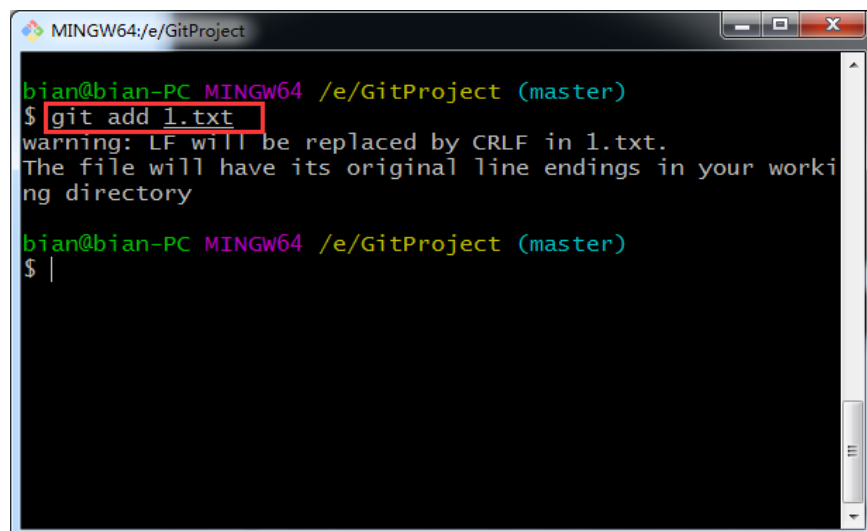


```
MINGW64:/e/GitProject
hello world this is our version 1
~
~
~
~
~
~
~
~
~
~
~
1.txt [unix] (11:54 06/11/2018) 1,33 全部
"1.txt" [新][unix] 1L, 34C 已写入
```

在 windows 上这么写是不是有种快感

首先，我们首先执行 `git add 1.txt`

`git add [filepath]` 将文件存入暂存区，如果不写会将当前目录下所有修改过的文件提交到暂存区。

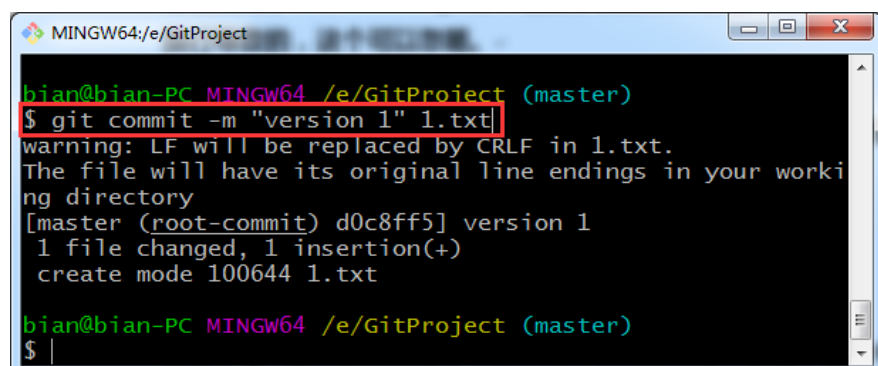


```
MINGW64:/e/GitProject
bian@bian-PC MINGW64 /e/GitProject (master)
$ git add 1.txt
warning: LF will be replaced by CRLF in 1.txt.
The file will have its original line endings in your working directory
bian@bian-PC MINGW64 /e/GitProject (master)
$ |
```

后面出现了一个 warning，这个是由于我们在安装 git 的时候配置的代码格式的换行导致的，这个可以忽略。

接着执行 `git commit -m "version 1" 1.txt`

`git commit -m "description" filepath` 提交git代码到本地库，-m后面加的是当前提交的注释。



```
MINGW64:/e/GitProject
bian@bian-PC MINGW64 /e/GitProject (master)
$ git commit -m "version 1" 1.txt
warning: LF will be replaced by CRLF in 1.txt.
The file will have its original line endings in your working directory
[master (root-commit) d0c8ff5] version 1
1 file changed, 1 insertion(+)
create mode 100644 1.txt
bian@bian-PC MINGW64 /e/GitProject (master)
$ |
```

这样就完成了一次基本提交了。

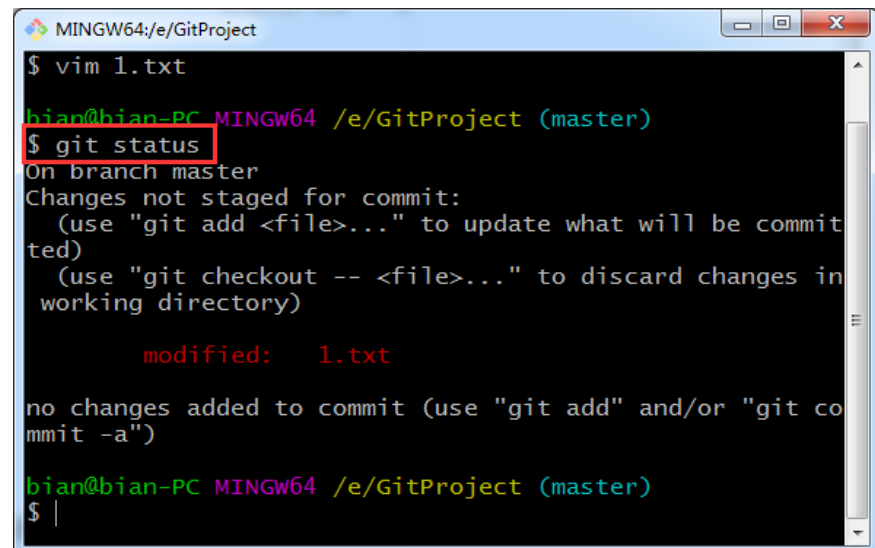
当然提交命令还有很多种，我们先学习最常用的！

6、查看状态

在整个 git 操作的过程当中，好多同学以为自己操作的最多的命令时候上传或者代码拉取，其实不然，就好像玩一款 RPG（角色扮演游戏）游戏，你最多的操作是不断的查看自己的状态，而 git 也是如此：

命令 `git status` 查看工作区、暂存区状态

有文件没有提交的状态

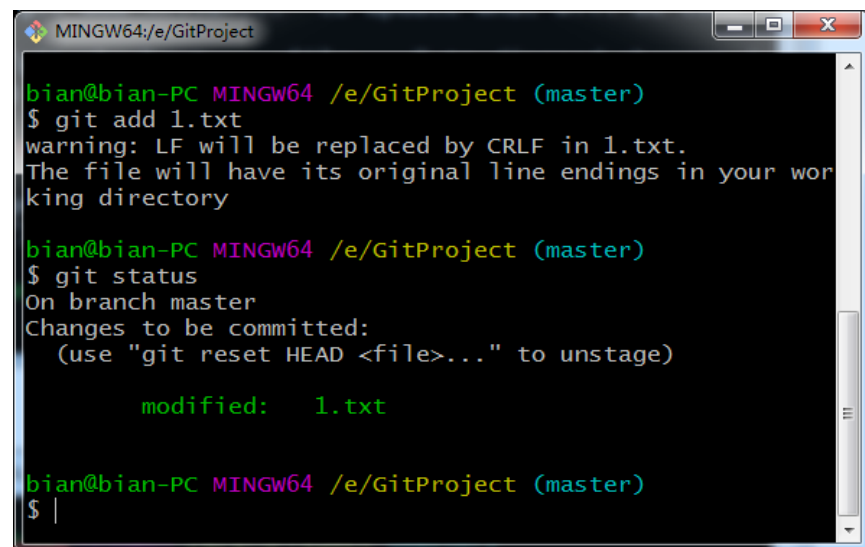


```
MINGW64:/e/GitProject
$ vim 1.txt
bian@bian-PC MINGW64 /e/GitProject (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   1.txt

no changes added to commit (use "git add" and/or "git commit -a")
bian@bian-PC MINGW64 /e/GitProject (master)
$ |
```

提交到缓存区，没有提交到本地库的状态



```
MINGW64:/e/GitProject
bian@bian-PC MINGW64 /e/GitProject (master)
$ git add 1.txt
warning: LF will be replaced by CRLF in 1.txt.
The file will have its original line endings in your working directory
bian@bian-PC MINGW64 /e/GitProject (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   1.txt

bian@bian-PC MINGW64 /e/GitProject (master)
$ |
```

提交完成的状态

```
MINGW64:/e/GitProject

bian@bian-PC MINGW64 /e/GitProject (master)
$ git commit -m "version 2" 1.txt
warning: LF will be replaced by CRLF in 1.txt.
The file will have its original line endings in your working directory
[master cdba861] version 2
1 file changed, 2 insertions(+)

bian@bian-PC MINGW64 /e/GitProject (master)
$ git status
On branch master
nothing to commit, working tree clean

bian@bian-PC MINGW64 /e/GitProject (master)
$ |
```

命令 `git log` 查看历史记录

```
MINGW64:/e/GitProject

bian@bian-PC MINGW64 /e/GitProject (master)
$ git log
commit d0c8ff5f1b5d6294fe1c234a8a03cd6876e1e3e1 (HEAD -> master)
Author: While <2312483892@qq.com>
Date: Tue Nov 6 12:01:40 2018 +0800

    version 1

bian@bian-PC MINGW64 /e/GitProject (master)
$
```

`git log --pretty=oneline`

```
MINGW64:/e/GitProject

bian@bian-PC MINGW64 /e/GitProject (master)
$ git log --pretty=oneline
d0c8ff5f1b5d6294fe1c234a8a03cd6876e1e3e1 (HEAD -> master) version 1

bian@bian-PC MINGW64 /e/GitProject (master)
$
```

`git log --oneline`

```
MINGW64:/e/GitProject
bian@bian-PC MINGW64 /e/GitProject (master)
$ git log --oneline
d0c8ff5 (HEAD -> master) version 1
bian@bian-PC MINGW64 /e/GitProject (master)
$ |
```

git relog

```
MINGW64:/e/GitProject
bian@bian-PC MINGW64 /e/GitProject (master)
$ git relog
d0c8ff5 (HEAD -> master) HEAD@{0}: commit (initial): version 1
bian@bian-PC MINGW64 /e/GitProject (master)
$ |
```

7、进行本地代码的版本控制

为了让版本控制的效果可以出来，我们多添加几个版本

```
MINGW64:/e/GitProject
bian@bian-PC MINGW64 /e/GitProject (master)
$ git relog
e8bb2a9 (HEAD -> master) HEAD@{0}: commit: version 5
f45a6ae HEAD@{1}: commit: version 4
d6bf482 HEAD@{2}: commit: version 3
747eea5 HEAD@{3}: commit: version 2
35f3e02 HEAD@{4}: commit (initial): version 1
bian@bian-PC MINGW64 /e/GitProject (master)
$ |
```

然后我们来查看一下 Git 版本控制的原理

1、hash 校验

git 本身提交文件发生修改依靠的是 hash 的 MD5 加密,主要用到了 MD5 的一致性验证:MD5 将整个文件当做一个大文本信息,通过不可逆的字符串

变换算法,产生一个唯一的 MD5 信息摘要.就像每个人都有自己独一无二的指纹,MD5 对任何文件产生一个独一无二的数字指纹.在这里要注意以下几个点 :

1、哈希 md5 具有压缩性 : 输入任意 , MD5 值长度固定文件

```
import hashlib

#md5 压缩性

example_1 = "abc".encode()

example_2 = """abcdefghijklmnopqrsjfljasdlkfjasdlkfj
lasdjflksadjklfjsadlkfjlkdsajflksdjafkjsads
fdlkajflsakjflksadjklfjlsadjflksjdalkfja""".encode()

md_1 = hashlib.md5()
md_1.update(example_1)
result_1 = md_1.digest()

md_2 = hashlib.md5()
md_2.update(example_2)
result_2 = md_2.digest()

print(len(result_1))
print(len(result_2))
```

结果如下

```
G:\Python35\python.exe C:/Users/bian/Desktop/hashTest.py
16
16

Process finished with exit code 0
```

2、哈希 md5 具有抗修改性 : 修改了一点点 , 所算出 MD5 值差别很大 当然 md5 也有其他的特性 , 我们在这里不再强调了。

```
import hashlib

#md5 压缩性

example_1 = "abc".encode()

example_2 = "aba".encode()

md_1 = hashlib.md5()
md_1.update(example_1)
result_1 = md_1.digest()

md_2 = hashlib.md5()
```

```
md_2.update(example_2)
result_2 = md_2.digest()
```

```
print(result_1)
print(result_2)
```

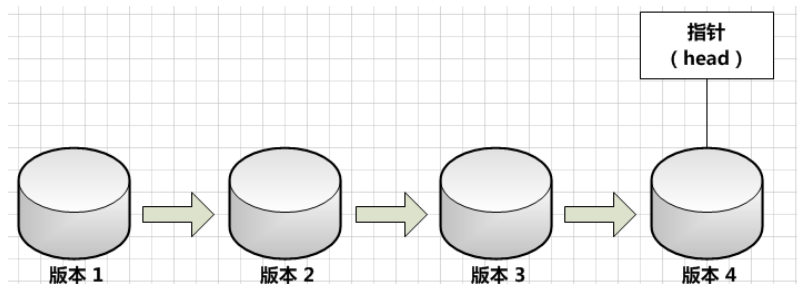
结果如下

```
G:\Python35\python.exe C:/Users/bian/Desktop/hashTest.py
b'\x90\x01P\x98<\xd20\xb0\xd6\x96?}(\xe1\x7fr'
b"y\xaf\x87r=\xc2\x95\xf9[\xdb'za\x18\x9a*"

Process finished with exit code 0
```

2、指针

Git 进行版本控制实际上用到的是一种指针机制，每次 git 提交都会形成一个独立的版本，我们每次进行 git 版本移动，实际上是在移动我们的指针，我们来看图：



那么我们现在开始具体的 git 执行版本移动操作

基于索引值操作[推荐]

`git reset --hard [局部索引值]`

首先我们查看一下当前的 1.txt

```
MINGW64:/e/GitProject
bian@bian-PC MINGW64 /e/GitProject (master)
$ vim 1.txt

bian@bian-PC MINGW64 /e/GitProject (master)
$ |
```

```
MINGW64:/e/GitProject
hello world this is our version 1
hello world this is our version 2
hello world this is our version 3
hello world this is our version 4
hello world this is our version 5
~
~
~
~
~
1.txt [unix] (11:07 07/11/2018) 5,33 全部
"1.txt" [unix] 5L, 170C
```

然后查看我们的版本记录，命令前面讲过

```
MINGW64:/e/GitProject
bian@bian-PC MINGW64 /e/GitProject (master)
$ git reflog
e8bb2a9 (HEAD -> master) HEAD@{0}: commit: vers
ion 5
f45a6ae HEAD@{1}: commit: version 4
d6bf482 HEAD@{2}: commit: version 3
747eea5 HEAD@{3}: commit: version 2
35f3e02 HEAD@{4}: commit (initial): version 1

bian@bian-PC MINGW64 /e/GitProject (master)
$ |
```

我们将版本回退到 version4

```
MINGW64:/e/GitProject
bian@bian-PC MINGW64 /e/GitProject (master)
$ git reflog
e8bb2a9 (HEAD -> master) HEAD@{0}: commit: vers
ion 5
f45a6ae HEAD@{1}: commit: version 4
d6bf482 HEAD@{2}: commit: version 3
747eea5 HEAD@{3}: commit: version 2
35f3e02 HEAD@{4}: commit (initial): version 1

bian@bian-PC MINGW64 /e/GitProject (master)
$ git reset --hard f45a6ae
HEAD is now at f45a6ae version 4

bian@bian-PC MINGW64 /e/GitProject (master)
$ |
```

然后查看一下文件，内容回退到了版本 4


```
MINGW64:/e/GitProject
bian@bian-PC MINGW64 /e/GitProject (master)
$ vim 1.txt
bian@bian-PC MINGW64 /e/GitProject (master)
$ |
```

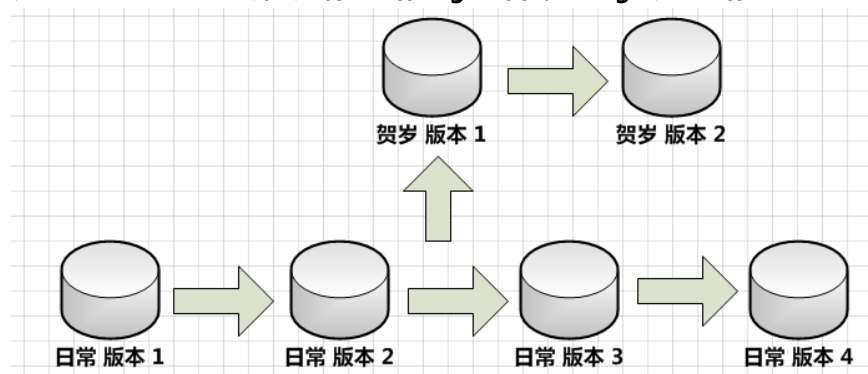
```
MINGW64:/e/GitProject
hello world this is our version 1
hello world this is our version 2
hello world this is our version 3
hello world this is our version 4
~
~
~
~
~
1.txt [dos] (11:25 07/11/2018) 1,1 全部
"1.txt" [dos] 4L, 140C
```

当然我们用这条命令还可以移动到各个版本，大家多多尝试

8、分支操作

分支介绍

在开发的过程当中，有时候我们需要进行版本的迭代，但是除了版本迭代还会有另外的一种情况，就是有另外的一种想法。比如我们开发一款游戏，在春节的时候需要开发贺岁版，那么这个就需要我们进行一个独立的开发环境了，和我们开发的主干线互相独立。那么这样的操作在 git 当中，叫做 git 分枝操作。



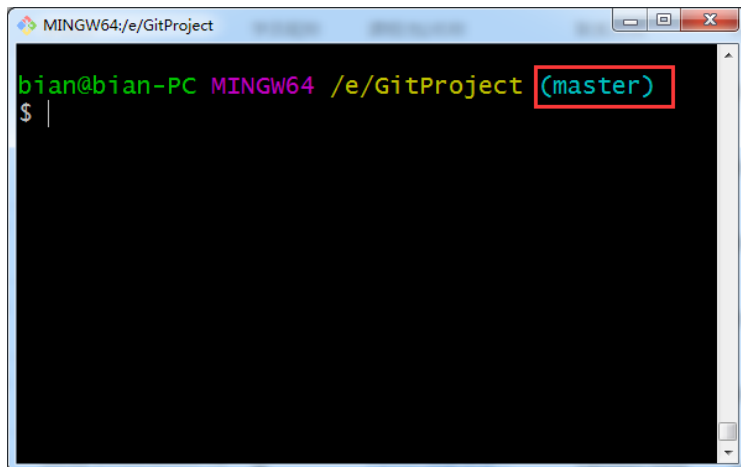
Git 的分枝操作，同时并行推进多个功能开发，提高开发效率

各个分支在开发过程中，如果某一个分支开发失败，不会对其他分支有任何影响。失败的分支删除重新开始即可。

分枝的操作

创建分支

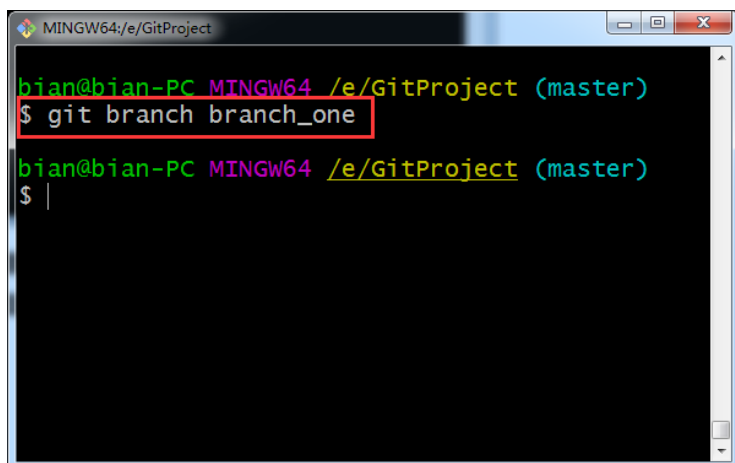
我们在创建 git 仓库之初就有一个主分支，系统命名为 master，在我们上面操作的时候，都是在 master 分支上操作的。



```
MINGW64:/e/GitProject
bian@bian-PC MINGW64 /e/GitProject (master)
$ |
```

当然，我们还可以创建新的分枝

命令：git branch [分支名]

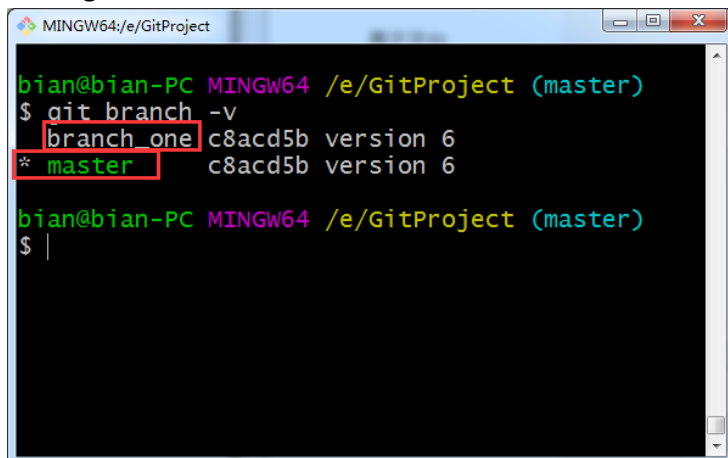


```
MINGW64:/e/GitProject
bian@bian-PC MINGW64 /e/GitProject (master)
$ git branch branch_one
bian@bian-PC MINGW64 /e/GitProject (master)
$ |
```

查看分支

创建完成分支之后，我们可以查看一下

命令：git branch -v



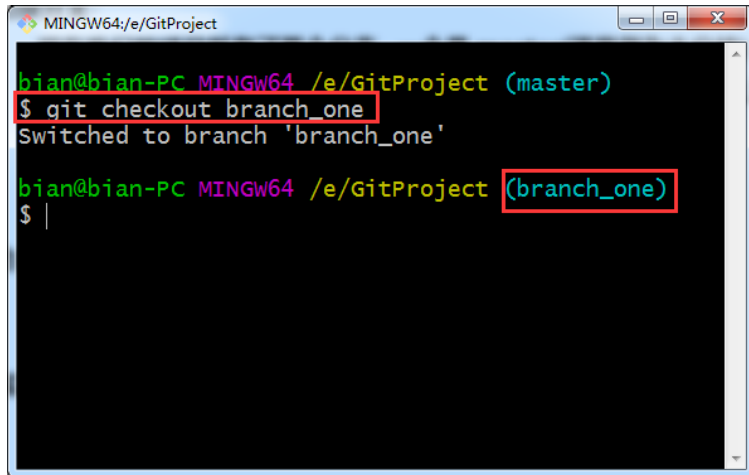
```
MINGW64:/e/GitProject
bian@bian-PC MINGW64 /e/GitProject (master)
$ git branch -v
  branch one c8acd5b version 6
* master    c8acd5b version 6
bian@bian-PC MINGW64 /e/GitProject (master)
$ |
```

绿色的部分是在使用的分支，上面是我们的分支

切换分支

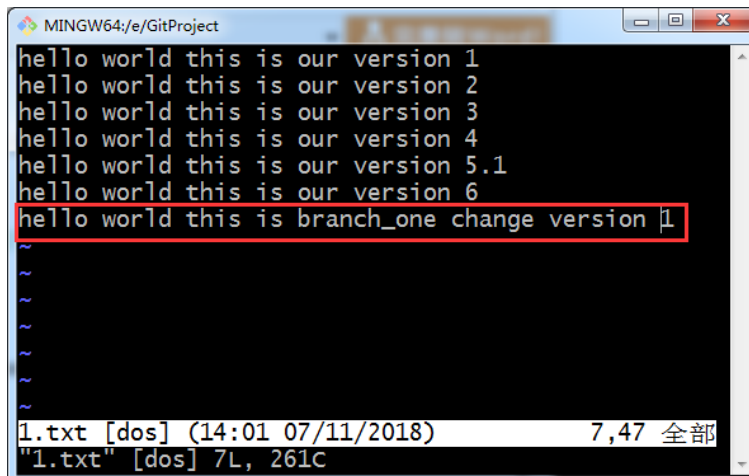
现在我们的项目就有了两个分支，一个是 master(通常做为主分支)，另一个就是我们刚刚创建的分支，我们切换到 branch_one 这个分支上进行操作。

命令：git checkout [branch_one]



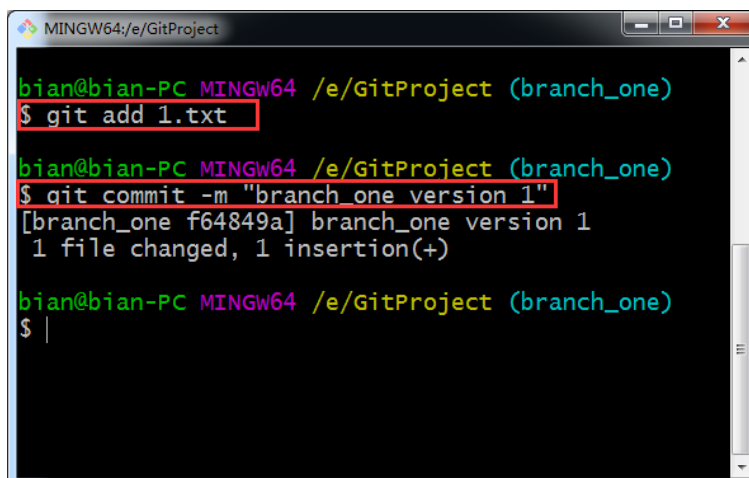
```
MINGW64:/e/GitProject
bian@bian-PC MINGW64 /e/GitProject (master)
$ git checkout branch_one
Switched to branch 'branch_one'
bian@bian-PC MINGW64 /e/GitProject (branch_one)
$ |
```

这样我们就切换到了刚刚创建的分支上，然后进行简单的文件修改



```
MINGW64:/e/GitProject
hello world this is our version 1
hello world this is our version 2
hello world this is our version 3
hello world this is our version 4
hello world this is our version 5.1
hello world this is our version 6
hello world this is branch_one change version 1
~
~
~
~
1.txt [dos] (14:01 07/11/2018) 7,47 全部
"1.txt" [dos] 7L, 261C
```

然后提交



```
MINGW64:/e/GitProject
bian@bian-PC MINGW64 /e/GitProject (branch_one)
$ git add 1.txt
bian@bian-PC MINGW64 /e/GitProject (branch_one)
$ git commit -m "branch_one version 1"
[branch_one f64849a] branch_one version 1
1 file changed, 1 insertion(+)
bian@bian-PC MINGW64 /e/GitProject (branch_one)
$ |
```

这个时候我们再次查看分支

```
MINGW64:/e/GitProject
bian@bian-PC MINGW64 /e/GitProject (branch_one)
$ git branch -v
* branch_one f64849a branch one version 1
  master    c8acd5b version 6

bian@bian-PC MINGW64 /e/GitProject (branch_one)
$ |
```

不同的分支已经有了各自的版本

合并分支

有了不同的分支，就有了不同的意见，但是最后的目的是为了提供一个统一的版本，所以，我们要学会合并分支。

切换分支

首先，我们要切换到我们的主分支下，要把那个分支作为合并后的主分支就切换到那个分支。比如我们当前是要将 branch 分支合并到 master 分支上，那么我们就先切换到 master。

命令：git checkout master

```
MINGW64:/e/GitProject
bian@bian-PC MINGW64 /e/GitProject (branch_one)
$ git checkout master
Switched to branch 'master'

bian@bian-PC MINGW64 /e/GitProject (master)
$ |
```

执行同步分支

命令：git merge branch_one

```
MINGW64:/e/GitProject
bian@bian-PC MINGW64 /e/GitProject (master)
$ git merge branch_one
Updating c8acd5b..f64849a
Fast-forward
 1.txt | 1 +
 1 file changed, 1 insertion(+)

bian@bian-PC MINGW64 /e/GitProject (master)
$ |
```

这样我们的版本就合并了

```
MINGW64:/e/GitProject
bian@bian-PC MINGW64 /e/GitProject (master)
$ git merge branch_one
Updating c8acd5b..f64849a
Fast-forward
 1.txt | 1 +
 1 file changed, 1 insertion(+)

bian@bian-PC MINGW64 /e/GitProject (master)
$ git branch -v
* branch_one f64849a branch_one version 1
  master     f64849a branch_one version 1

bian@bian-PC MINGW64 /e/GitProject (master)
$ |
```

查看一下，是不是整齐了好多

解决冲突

分支代表着项目有两个意见，合并就很可能出现冲突。

这里，在 master 分支上，有了不同的意见

```
MINGW64:/e/GitProject
hello world this is our version 1
hello world this is our version 2
hello world this is our version 3
hello world this is our version 4
hello world this is our version 5.1
hello world this is our version 6 breach_one has o
therider
hello world this is branch_one change version 1
hello I am master
~
~
~
~
1.txt [dos] (14:30 07/11/2018) 9,17 全部
"1.txt" [dos] 9L, 307C
```

然后进行提交

```
MINGW64:/e/GitProject
bian@bian-PC MINGW64 /e/GitProject (master)
$ git add 1.txt

bian@bian-PC MINGW64 /e/GitProject (master)
$ git commit -m "master's ider" 1.txt
[master 2a91003] master's ider
1 file changed, 1 insertion(+), 1 deletion(-)

bian@bian-PC MINGW64 /e/GitProject (master)
$ |
```

接着 branch 也有了不同的想法

```
MINGW64:/e/GitProject
hello world this is our version 1
hello world this is our version 2
hello world this is our version 3
hello world this is our version 4
hello world this is our version 5.1
hello world this is our version 6 breach_one has o
therider
hello world this is branch_one change version 1
hello I am breach_one
~
~
~
1.txt [dos] (14:32 07/11/2018) 9,21 全部
:
```

同样提交

```
MINGW64:/e/GitProject
$ git checkout branch_one
Switched to branch 'branch_one'

bian@bian-PC MINGW64 /e/GitProject (branch_one)
$ vim 1.txt

bian@bian-PC MINGW64 /e/GitProject (branch_one)
$ git add 1.txt

bian@bian-PC MINGW64 /e/GitProject (branch_one)
$ git commit -m "breach's ider" 1.txt
[branch_one eec2e21] breach's ider
1 file changed, 2 insertions(+)

bian@bian-PC MINGW64 /e/GitProject (branch_one)
$ |
```

接着进行合并

这次我们把 master 合并到 breach_one

```
MINGW64:/e/GitProject
bian@bian-PC MINGW64 /e/GitProject (branch_one)
$ git merge master
Auto-merging 1.txt
CONFLICT (content): Merge conflict in 1.txt
Automatic merge failed; fix conflicts and then commit the result.

bian@bian-PC MINGW64 /e/GitProject (branch_one|MERGING)
$ |
```

这个时候，二者的意见出现了不一致，我们查看一下 1.txt

```
MINGW64:/e/GitProject
hello world this is our version 1
hello world this is our version 2
hello world this is our version 3
hello world this is our version 4
hello world this is our version 5.1
hello world this is our version 6 breach_one has o
therider
hello world this is branch_one change version 1

<<<<<< HEAD
hello I am breach_one breach 的意见
=====
hello I am master master的意见
>>>>>> master
1.txt [dos] (14:34 07/11/2018) 12,12 全部
"1.txt" [dos] 13L, 369c
```

我们更具业务逻辑，对冲突的代码进行调整，（这里 master 最大）
所以，我们删除除了 master 的代码之外的代码

```
MINGW64:/e/GitProject
hello world this is our version 1
hello world this is our version 2
hello world this is our version 3
hello world this is our version 4
hello world this is our version 5.1
hello world this is our version 6 breach_one has o
therider
hello world this is branch_one change version 1

hello I am master
~
~
~
1.txt[+] [dos] (14:34 07/11/2018) 9,1 全部
:wq|
```

记着这个时候提交不要指定具体的文件名

```
MINGW64:/e/GitProject
bian@bian-PC MINGW64 /e/GitProject (branch_one|MER
GING)
$ git add 1.txt

bian@bian-PC MINGW64 /e/GitProject (branch_one|MER
GING)
$ git commit -m "deffirent ider"
[branch_one 1cacdca] deffirent ider

bian@bian-PC MINGW64 /e/GitProject (branch_one)
$ |
```

然后就 ok 啦

```
MINGW64:/e/GitProject
bian@bian-PC MINGW64 /e/GitProject (branch_one|MER
GING)
$ git add 1.txt

bian@bian-PC MINGW64 /e/GitProject (branch_one|MER
GING)
$ git commit -m "deffirent ider"
[branch_one 1cacdca] deffirent ider

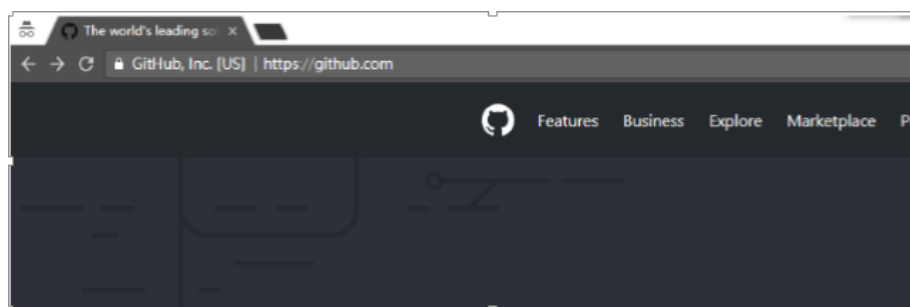
bian@bian-PC MINGW64 /e/GitProject (branch_one)
$ git merge master
Already up to date.

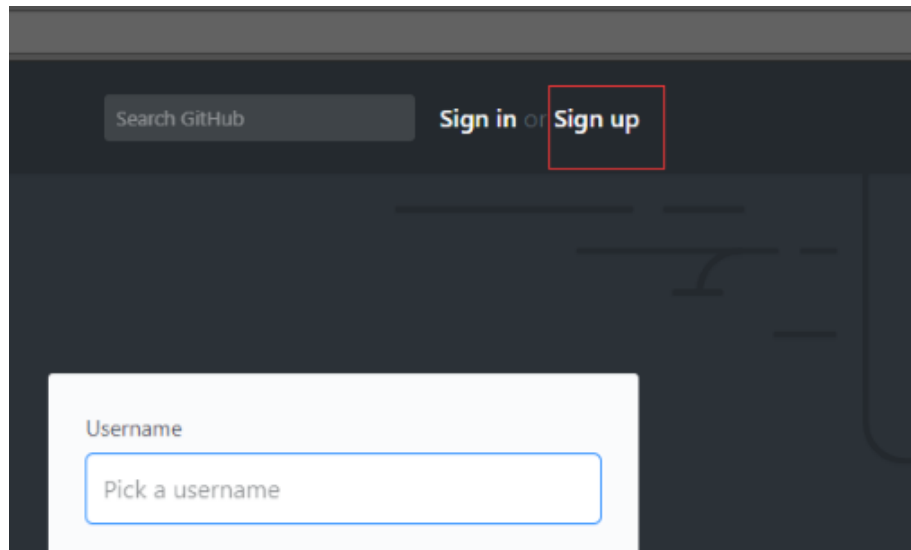
bian@bian-PC MINGW64 /e/GitProject (branch_one)
$ |
```

Git 的远程操作

Github 远程账号注册与库搭建

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

Create your personal account

Username

xuegod-while



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

Email address

1815741225@qq.com



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

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：

Completed

Set up a personal account

Step 2:

Choose your plan

Choose your personal plan

☒ Unlimited public repositories for free.

☐ Unlimited private repositories for \$7/month. (view in SGD)

Don't worry, you can cancel or upgrade at any time.

☐ Help me set up an organization next
Organizations are separate from personal accounts and are best suited for businesses who need to manage permissions for many employees.
[Learn more about organizations](#)

☐ Send me updates on GitHub news, offers, and events
Unsubscribe anytime in your email preferences. [Learn more](#)

Continue

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

Account

Emails

Notifications

Billing

SSH and GPG keys

Security

Blocked users

Repositories

Organizations

1319144285@qq.com

Primary

Not visible in emails

Receives notifications

Unverified

Verification email sent. [Resend](#)

Add email address

Add

Primary email address

Because you have email privacy enabled, 1319144285@qq.com will be used for account-related notifications and 38109028+xeugod-while@users.noreply.github.com will be used for web-based GitHub operations (e.g. edits and merges).

1319144285@qq.com

Save

不用多说，我们到邮箱完成校验

邮箱 腾讯&字神IT-讲师-while<1319144285@qq.com> 邮箱首页 | 设置 - 换肤

< 返回

回复

回复全部

转发

删除

彻底删除

举报

拒收

标记为...

移动到...

[GitHub] Please verify your email address. ☆

发件人: GitHub <noreply@github.com> [图]

时 间: 2018年4月5日(星期四) 上午8:27 (UTC-07:00 休斯顿, 波特兰时间)

收件人: 腾讯&字神IT-讲师-while <1319144285@qq.com>

为营造健康的邮箱环境，请确认该邮件是否由您订阅? [是我订阅的](#) [不是我订阅的](#) [忽略](#)

Hi @xeugod-while!

Help us secure your GitHub account by verifying your email address (1319144285@qq.com). This lets you access all of GitHub's feat

[Verify email address](#)

Button not working? Paste the following link into your browser:
https://github.com/users/xeugod-while/emails/49400530/confirm_verification/51c4a1ae31e5e8c4e98bd0220746d88486bea1b5

You're receiving this email because you recently created a new GitHub account or added a new email address. If this wasn't you, ple

然后跳转回来，

1319144285@qq.com is already verified.

Personal settings

Profile

Account

Emails

Notifications

Billing

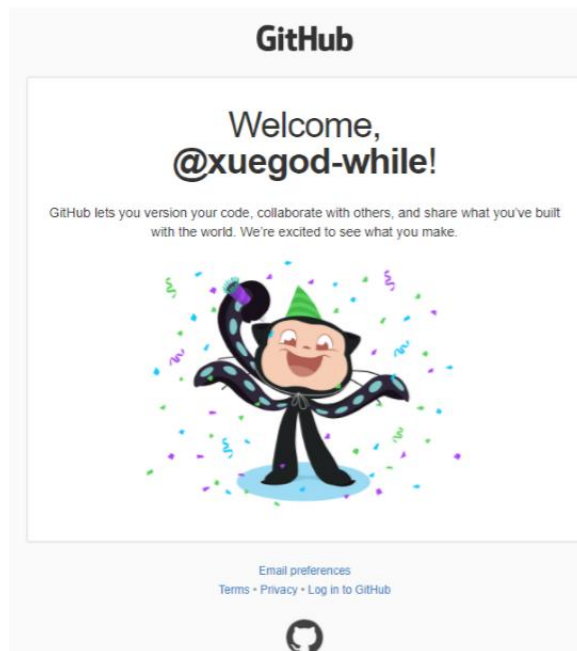
Emails

1319144285@qq.com	Primary	Not visible in emails	Receives no
-------------------	---------	-----------------------	-------------

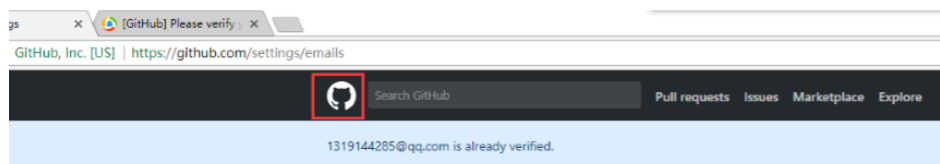
Add email address

Add

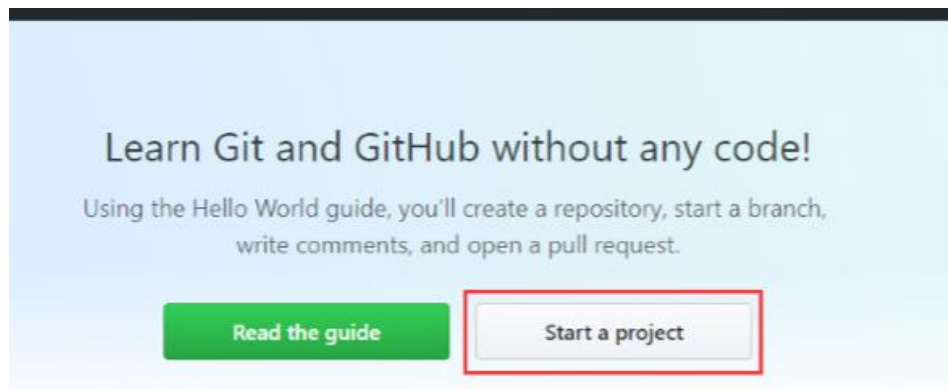
接着收到邮件



搞定收工，我们可以通过这里返回首页，创建项目



创建项目



Create a new repository

A repository contains all the files for your project, including the revision history.

Owner: xuegod-while / Repository name: ourFirstTest 项目名称

Great repository names are short and memorable. Need inspiration? How about [stunning-engine](#).

Description (optional): this is my first test about python 项目描述

☒ Public
Anyone can see this repository. You choose who can commit. 公开项目

☐ Private
You choose who can see and commit to this repository.

☐ Initialize this repository with a README
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: None Add a license: None ⓘ

Create repository

然后出现 git 提醒

Quick setup — if you've done this kind of thing before

☒ Set up in Desktop or ☐ HTTPS ☐ SSH https://github.com/xuegod-while/ourFirstTest.git

We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

...or create a new repository on the command line

```
echo "# ourFirstTest" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/xuegod-while/ourFirstTest.git
git push -u origin master
```

这样我们就完成了 git 账号的注册和项目的创建

Github 关联远程库

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钥，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。

通过 Git bash 随时上传你的代码。

首先生成 ssh-key:

1、设置用户名

```
bian@bian-PC MINGW64 /e/secondExample
$ git config --global user.name "while"
```

2、设置邮箱

```
bian@bian-PC MINGW64 /e/secondExample
$ git config --global user.email "2312483892@qq.com"
```

3、生成秘钥

命令如下：

```
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+ANZR3oLHWZtvxGh/yaoL7tesQHRC
```

```
bian@bian-PC
```

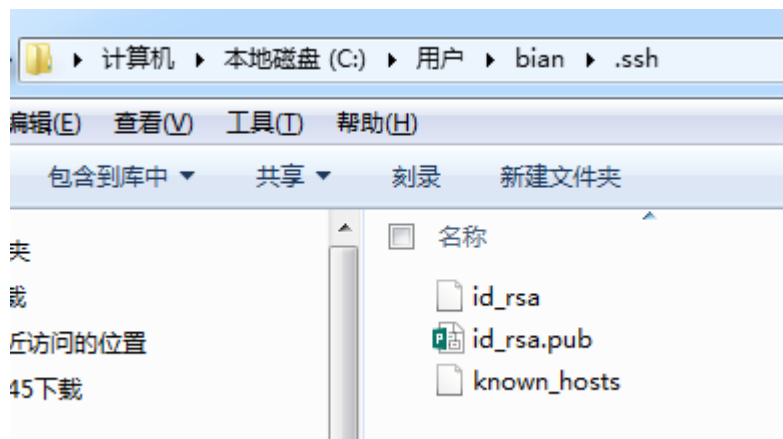
```
The key's randomart image is:
```

```
+---[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



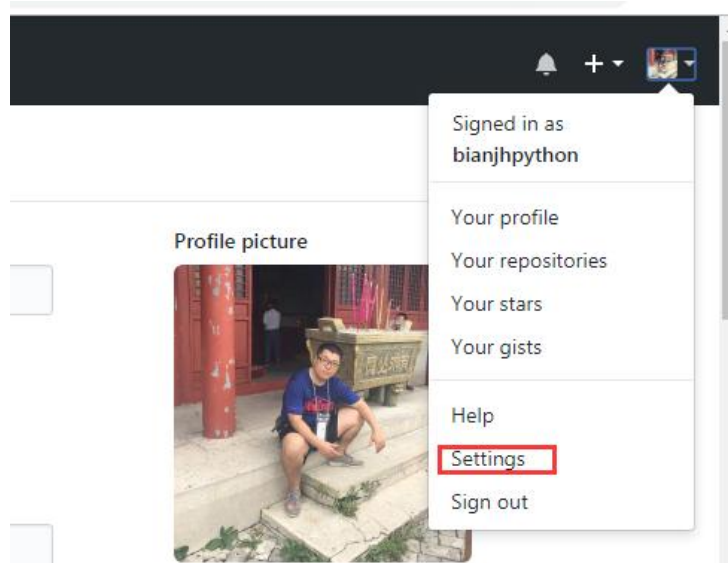
用 pycharm 或者 sublime 再或者其他的编译器打开 id_rsa.pub 文件 ,

然后得到公钥 :

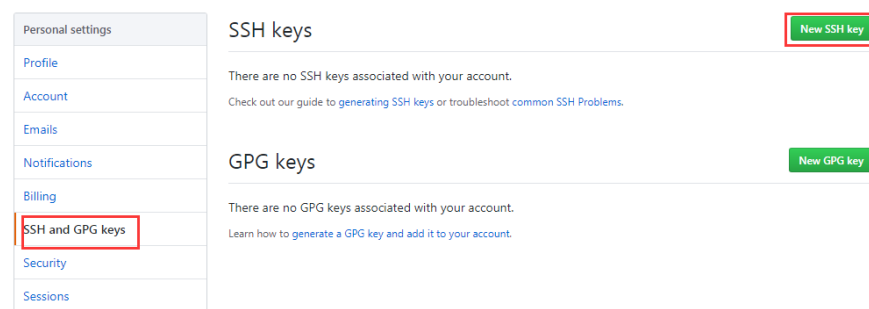
```
1 |ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQDLHvNCi2sMbaQnSNyusnb8Fkx4fVD5hi7WkpLH  
pUmIB5uGGbVWgSpy0pc0tq9Mrw5qw4D0WUEgTFw5sVxC8vhXmjTIF+uhPCkKDoLaRx1hrJCvH0R  
V33kQK2T1srHQJmq+NRWouZb0jgT0iefHxRtb7f29mqhGyic7VKzfzDGJuRu6c+YKeukLre0JbbM  
JNdE/L2Ub/0MS30fY56+46AhYsR0F8HjgsH4xsNcloy0oe91R9QQyzX0kp6e/  
ej+LpVRJEfYi3+tJictYwsYPJM4wUYpShjogD/  
tg3xsL3cbhFQ+snXCwmnKL6n6+JH09XRIaeViLPvSpKgdhBo2TF bian@bian-PC  
2
```

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

[SSH keys](#) / Add new

Title

对key的描述

Key

Begins with 'ssh-rsa', 'ssh-dss', 'ssh-ed25519', 'ecdsa-sha2-nistp256', 'ecdsa-sha2-nistp384', or 'ecdsa-sha2-nistp521'

key的值

Add SSH key

选择添加之后会询问 github 的登录密码，输入，绑定成功

SSH keys

New SSH key

This is a list of SSH keys associated with your account. Remove any keys that you do not recognize.



SSH

TeacherComputer

Fingerprint: cc:ac:ef:b3:c0:74:ad:b9:6a:6e:00:3d:fe:88:90:91

Added on 21 Oct 2018

Never used — Read/write

Delete

Check out our guide to [generating SSH keys](#) or troubleshoot [common SSH Problems](#).

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`

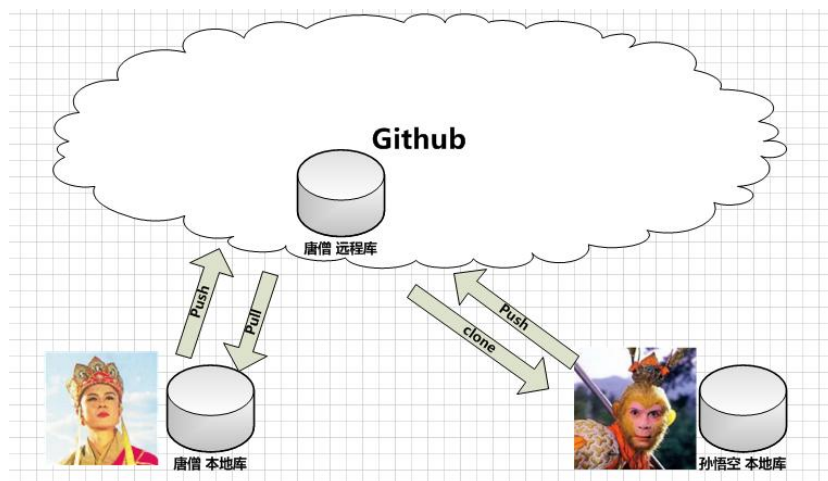
Enter passphrase for key '/c/Users/bian/.ssh/id_rsa':

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

```
MINGW64:/e/secondExample
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.
bian@bian-PC MINGW64 /e/secondExample
$ |
```

1.3 git 使用实例

上面的课程我们完成了本地库和网络库的设置，现在我们假设一个项目：
我们创建一个西游记的项目组，项目的创始人是唐僧，发起了西游的项目，然后邀请技术大牛孙悟空加入团队，一起 happy 的过程。



- 1、首先唐僧同志创建了自己的远程和本地库，叫做西游
创建本地库


```
MINGW64:/e/XiYou

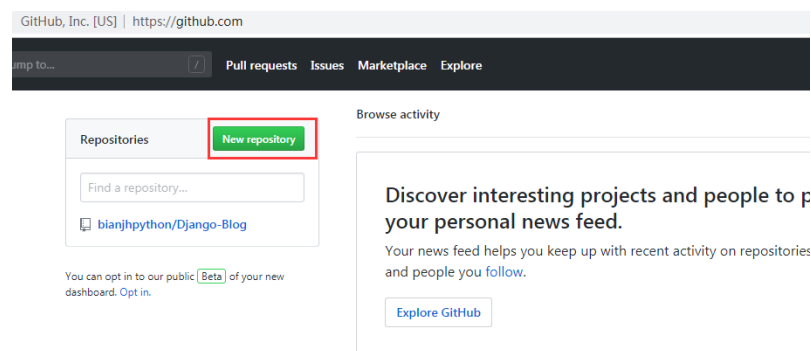
bian@bian-PC MINGW64 /e/XiYou
$ git init
Initialized empty Git repository in E:/XiYou/.git/

bian@bian-PC MINGW64 /e/XiYou (master)
$ git config user.name tang

bian@bian-PC MINGW64 /e/XiYou (master)
$ git config user.email tang@datan.com

bian@bian-PC MINGW64 /e/XiYou (master)
$ |
```

创建远程库



创建网络库

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner: bianjpython / Repository name:

Great repository names are short and memorable. Need inspiration? How about bookish-parakeet.

Description (optional):

☒ Public: Anyone can see this repository. You choose who can commit.

☐ Private: You choose who can see and commit to this repository.

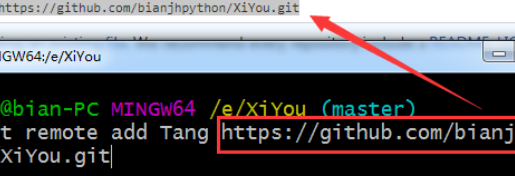
☐ Initialize this repository with a README
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: Add a license: ⓘ

然后进行本地和网络库关联

创建远程库地址别名

done this kind of thing before



The screenshot shows a terminal window titled 'MINGW64/e/XiYou'. The prompt is 'bian@bian-PC MINGW64 /e/XiYou (master)'. The command entered is '\$ git remote add Tang https://github.com/bianjhpython/XiYou.git'. A red box highlights the URL 'https://github.com/bianjhpython/XiYou.git' in the command. A red arrow points from this box to the address bar of a web browser above the terminal, which shows the same URL.

```
SSH https://github.com/bianjhpython/XiYou.git
```

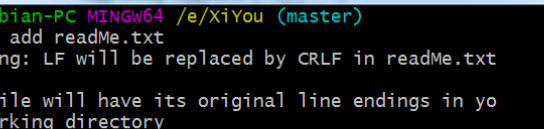
```
bian@bian-PC MINGW64 /e/XiYou (master)  
$ git remote add Tang https://github.com/bianjhpython/XiYou.git
```

尝试上传第一个文件 readme.txt

[illegible]

尝试上传

先提交到本地库



```
MINGW64:/e/XiYou 通过Git管理代码
bian@bian-PC MINGW64 /e/XiYou (master)
$ git add readMe.txt
warning: LF will be replaced by CRLF in readMe.txt
.
The file will have its original line endings in your working directory
bian@bian-PC MINGW64 /e/XiYou (master)
$ git commit -m "readme" readMe.txt
warning: LF will be replaced by CRLF in readMe.txt
.
The file will have its original line endings in your working directory
[master (root-commit) 84a6e30] readme
1 file changed, 1 insertion(+)
create mode 100644 readMe.txt
```

然后提交到远端库

命令 git push Tang master

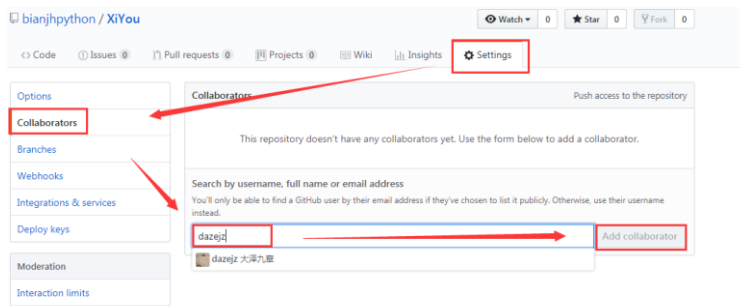
git push 远端用户名 提交的分支

```
MINGW64:/e/XiYou
bian@bian-PC MINGW64 /e/XiYou (master)
$ git push Tang master
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 220 bytes | 220.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
remote:
remote: Create a pull request for 'master' on GitHub by visit
remote:      https://github.com/bianjhpython/XiYou/pull/new/
remote:      master
remote:
To https://github.com/bianjhpython/XiYou.git
 * [new branch]      master -> master

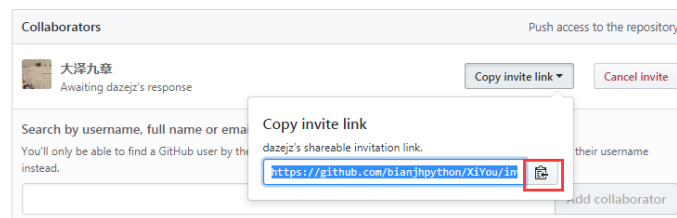
bian@bian-PC MINGW64 /e/XiYou (master)
$ |
```

这个时候，就完成了第一步。

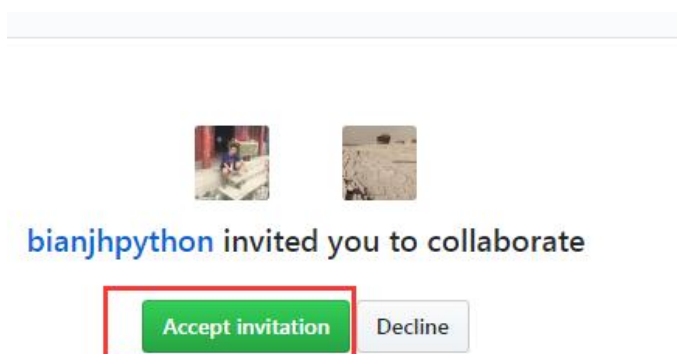
- 2、 然后唐僧邀请悟空进入项目
查找到孙悟空的账号



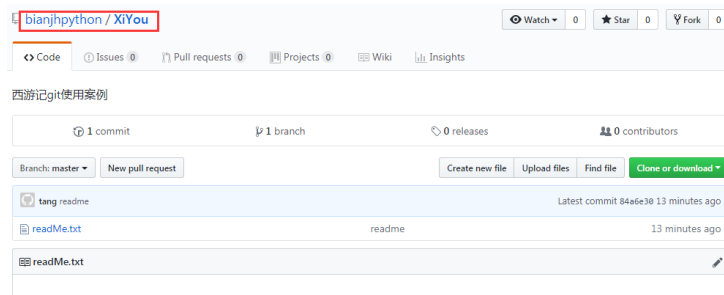
然后生成邀请链接



将链接交给孙悟空，然后打开，选择同意加入



这样就加入了项目组



3、最后悟空克隆代码进行编写提交

然后就可以克隆项目进行编写和提交了

命令：git clone https://github.com/bianjhpython/XiYou.git

```
MINGW64:/e/XiYou
bian@bian-PC MINGW64 /e/XiYou
$ git clone https://github.com/bianjhpython/XiYou.git
Cloning into 'XiYou'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.

bian@bian-PC MINGW64 /e/XiYou
$ |
```

克隆命令会自动实例化 git 库的不用先 init

当然 push 的命令也是一样的

修改项目

```
MINGW64:/e/XiYou/XiYou
来啊，快活啊
悟空来了
~
~
~
~
~
~
~
readMe.txt [dos] (15:53 07/11/2018) 2,10-7 全部
"readMe.txt" [dos] 2L, 34C
```

提交项目到本地库，然后添加用户名和邮箱

```
MINGW64:/e/XiYou/XiYou
bian@bian-PC MINGW64 /e/XiYou/XiYou (master)
$ git add readMe.txt

bian@bian-PC MINGW64 /e/XiYou/XiYou (master)
$ git commit -m "sun" readMe.txt
[master 320d068] sun
1 file changed, 1 insertion(+)

bian@bian-PC MINGW64 /e/XiYou/XiYou (master)
$ git config user.name Sun

bian@bian-PC MINGW64 /e/XiYou/XiYou (master)
$ git config user.name Sun@huanguo.com

bian@bian-PC MINGW64 /e/XiYou/XiYou (master)
$
```

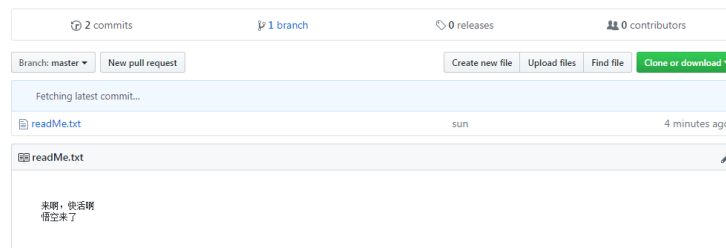
进行提交

```
MINGW64:/e/XiYou/XiYou
bian@bian-PC MINGW64 /e/XiYou/XiYou (master)
$ git remote add Sun https://github.com/bianjhpython/XiYou.git

bian@bian-PC MINGW64 /e/XiYou/XiYou (master)
$ git push Sun master
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Writing objects: 100% (3/3), 266 bytes | 133.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/bianjhpython/XiYou.git
84a6e30..320d068 master -> master

bian@bian-PC MINGW64 /e/XiYou/XiYou (master)
$
```

效果如下：



总结：

- 1.1 版本控制介绍以及常用的版本控制工具
- 1.2 版本控制工具-GIT 基础使用
- 1.3 git 使用实例