Git使用说明

编写者：张经鑫

# 1 Git的下载与安装

## 1.1 Git下载

从Git官网上下载相应操作系统对应的Git版本，网址为：[https：//git-scm.com/downloads/点击相应版本即可下载，如图1-1所示。](https://git-scm.com/downloads/。点击相应版本即可下载，如图1-1所示。)

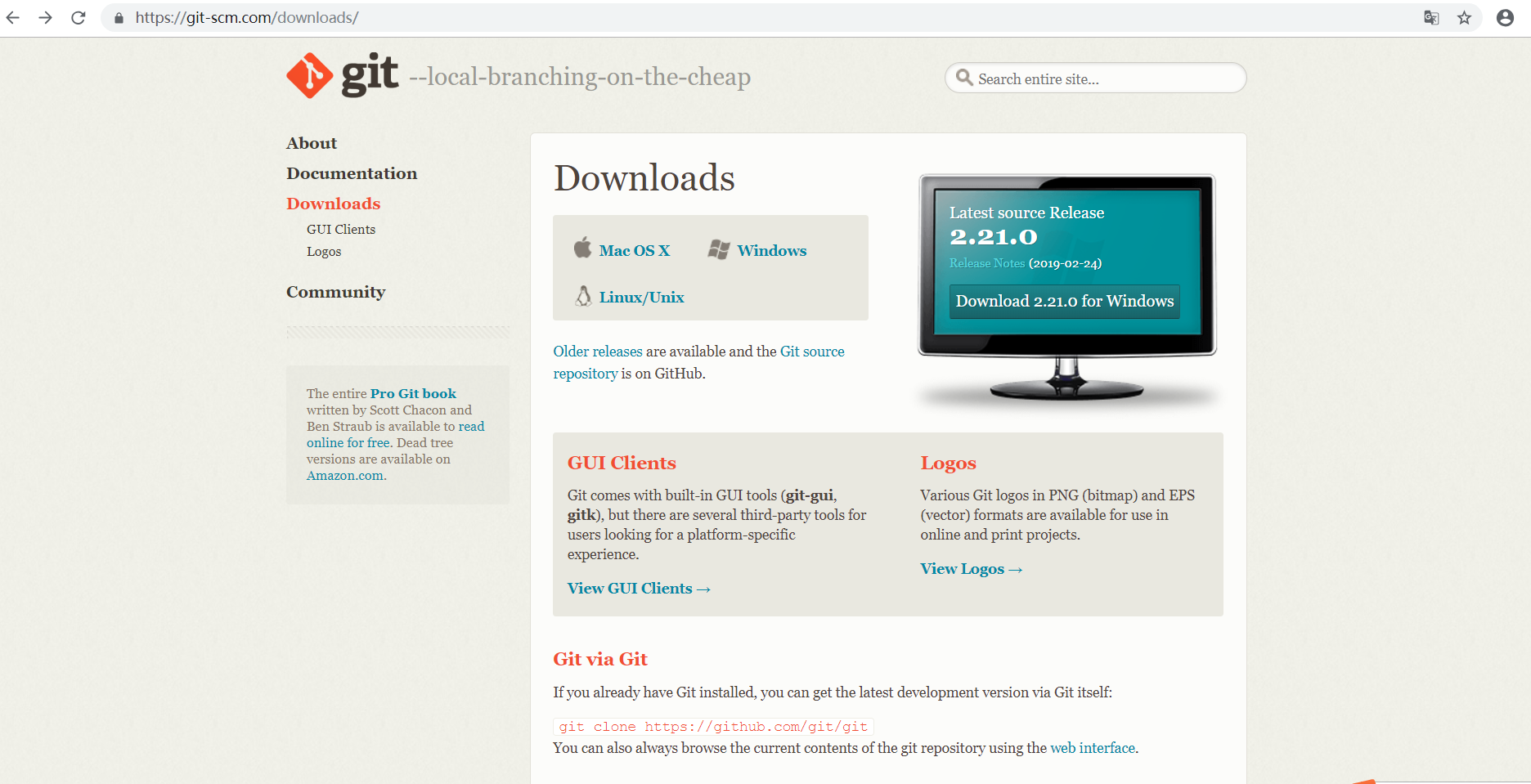


图1-1 Git下载页面

## 1.2 Git安装

①双击打开“Git-2.21.0-64-bit.exe”，会打开安装程序，安装程序初始页面如图1-2所示。

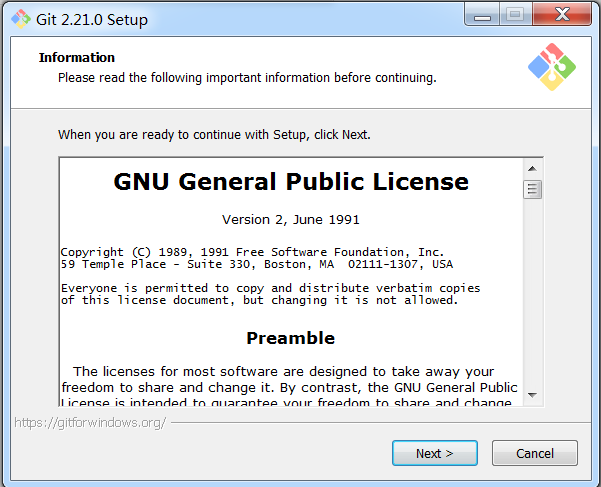


图1-2 Git安装程序初始界面

②点击“Next”按钮，跳转到安装路径界面，如图1-3所示。在下面可点击的按钮上方，有一行小提示，告诉用户至少需要多少硬盘容量来安装Git。

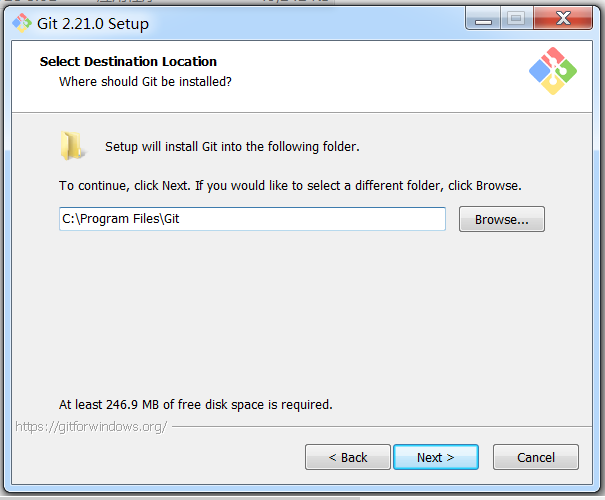


图1-3 选择安装路径

③继续点击“Next”按钮，跳转到选择安装组件的页面。如图1-4所示。

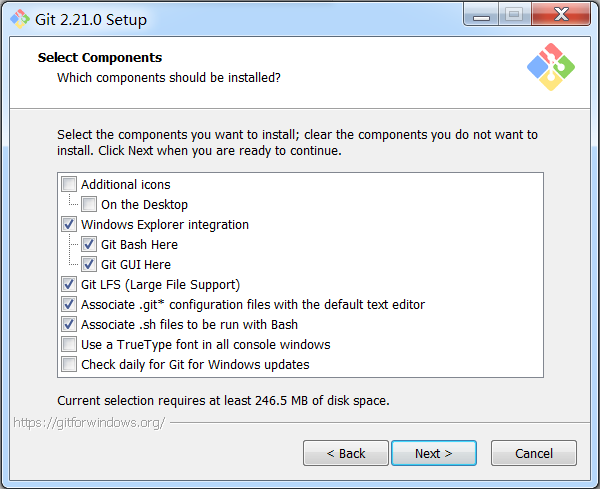


图1-4 选择安装组件

从上到下分别是：

1. 图标（Additional icons）：即选择是否创建桌面快捷方式。
2. Windows Explorer integration：如果选择，则在某个文件中单击鼠标右键，会在出现的一系列选项中出现“Git Bash Here”和“Git GUI Here”两个可选的选项。可快捷打开Git Bash或Git GUI。如图1-5所示。



图1-5 “Git Bash Here”和“Git GUI Here”

1. Git LFS(Lager File Support)：可以把音乐、图片、视频等指定的任意文件存在Git仓库之外，而在Git仓库中用一个占用空间1KB不到的文本指针来代替的小工具。通过把大文件存储在Git仓库之外，可以减小Git仓库本身的体积，使克隆Git仓库的速度加快，也使得Git不会因为仓库中充满大文件而损失性能。
2. Associate .git\* configuration files with the default text editor：将.git配置文件与默认文本编辑器相关联。
3. Associate .sh files to be run with Bash：将.sh文件关联到Bash运行。
4. Use a TrueType font in all console windows：在所有控制台窗口中使用TrueType字体。
5. Check daily for Git for Windows updates：每天检查Git是否有Windows更新。

④继续点击“Next”按钮，进入创建开始菜单目录页面，如图1-6所示。

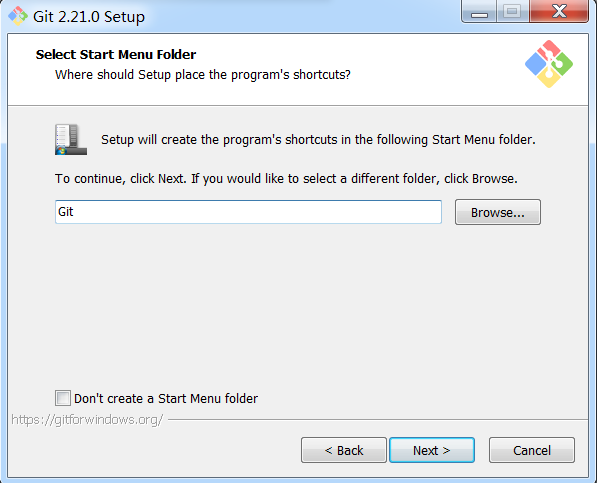


图1-6 创建开始菜单目录

⑤点击“Next”按钮，选择Git默认的编辑器。如图1-7所示。

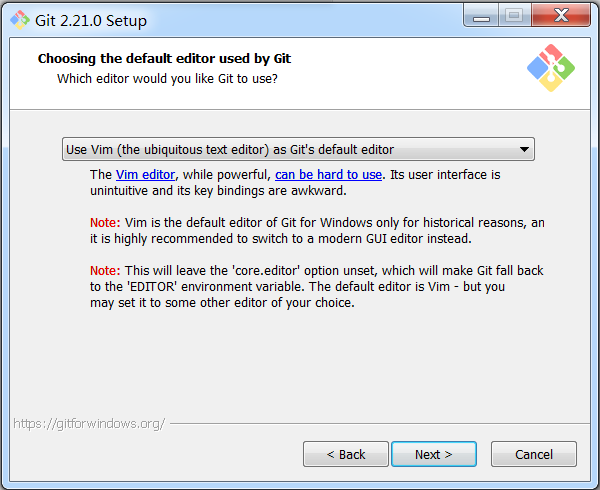


图1-7 选择Git默认的编辑器

其中不同选项的含义如图1-8所示。

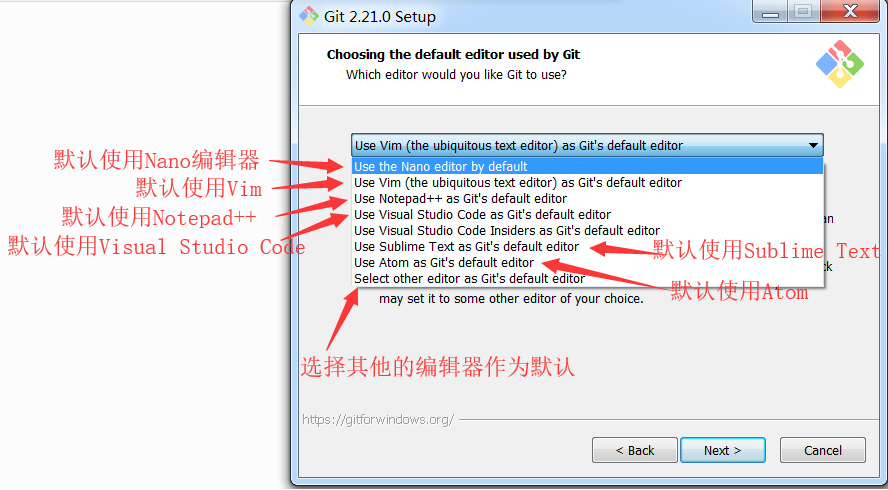


图1-8 选择Git默认的编辑器

⑥点击“Next”按钮，跳转到选择命令行工具。如图1-9所示。

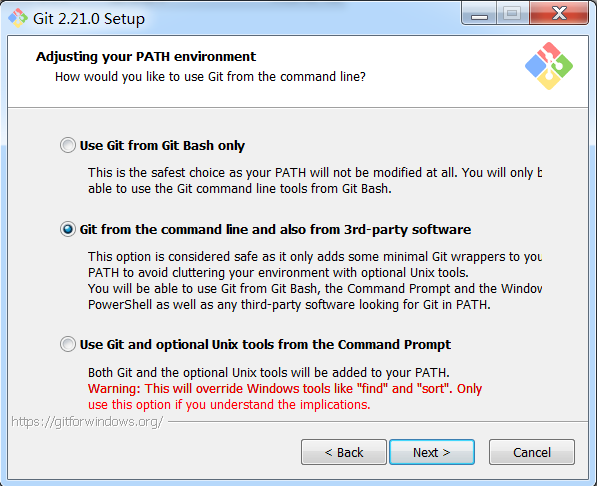


图1-9 选择命令行工具

其中，三个选项分别为：

1. Use Git from Bash only：这是最安全的选择，因为您的PATH根本不会被修改。您只能使用 Git Bash 的 Git 命令行工具。
2. Git from the command line and also from 3rd-party software：这个选项被认为是安全的，因为它只向PATH添加一些最小的 Git包，以避免使用可选的Unix工具混淆环境。 您将能够从 Git Bash 和 Windows 命令提示符中使用 Git。
3. Use Git and optional Unix tools from the Command Prompt：从Windows命令提示符使用Git和可选的Unix工具。需要注意的是，这个选项下有红色的警告语：这将覆盖Windows工具，如 “ find 和 sort ”。只有在了解其含义后才使用此选项。因此，一般不建议选择该选项。

⑦点击“Next”按钮，跳转到选择HTTPS传输后端页面。如图1-10所示。

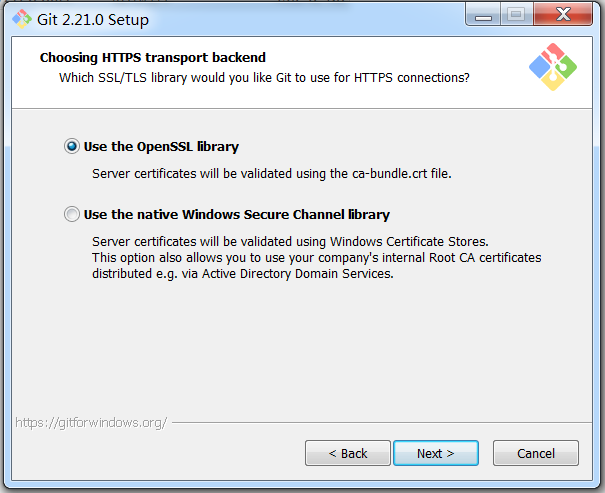


图1-10 选择HTTPS传输后端

两个选项分别为使用OpenSSL库和使用本地Windows安全通道库，默认选择OpenSSL库即可。

⑧点击“Next”按钮，进入配置行结束转换页面。如图1-11所示。

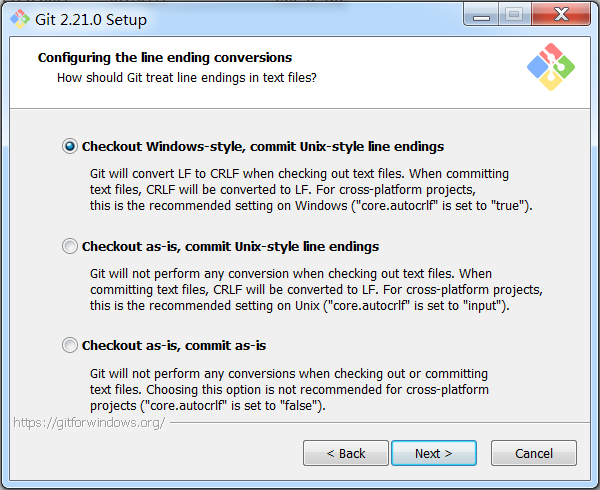


图1-11 配置行结束转换

一般选择默认的选项即可。

⑨点击“Next”按钮，进入配置[终端模拟器](https://www.baidu.com/s?wd=%E7%BB%88%E7%AB%AF%E6%A8%A1%E6%8B%9F%E5%99%A8&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd" \t "https://blog.csdn.net/sishen47k/article/details/_blank)以与 Git Bash 一起使用页面。如图1-12所示。

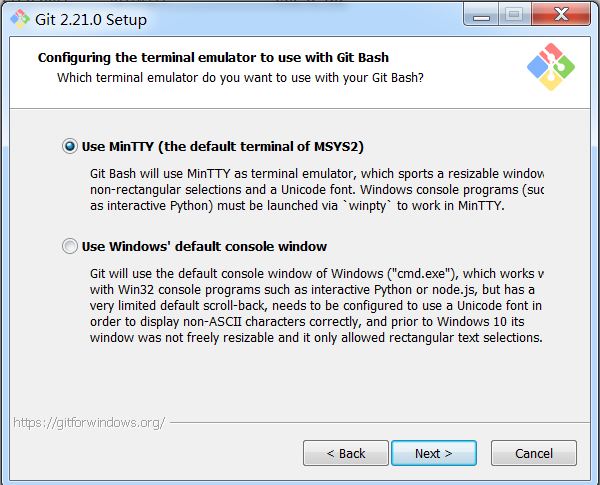


图1-12 配置[终端模拟器](https://www.baidu.com/s?wd=%E7%BB%88%E7%AB%AF%E6%A8%A1%E6%8B%9F%E5%99%A8&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd" \t "https://blog.csdn.net/sishen47k/article/details/_blank)以与 Git Bash 一起使用

同上，选择默认选项即可。

⑩点击“Next”按钮，进入配置额外的选项页面。如图1-13所示。

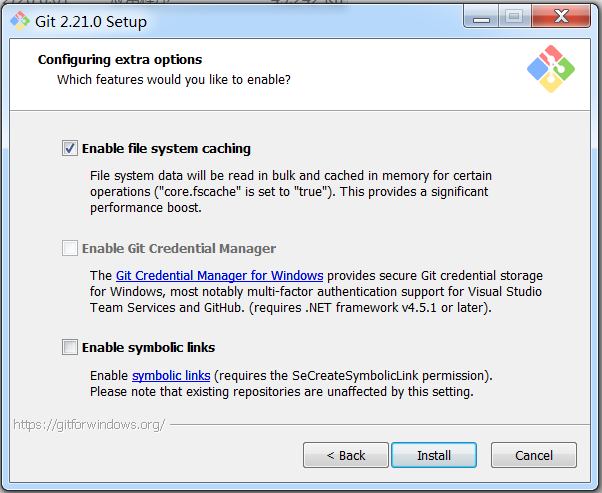


图1-13 配置额外的选项

选择默认的选项即可。

⑪点击“Install”按钮，进入安装页面，通过进度条显示安装进度。如图1-14所示。

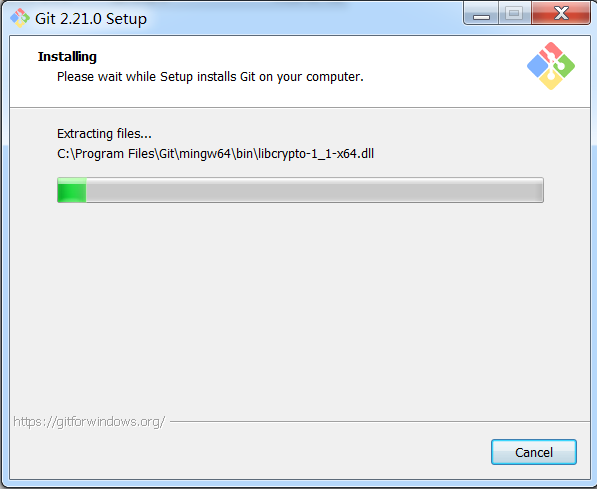


图1-14 安装进度

⑫安装完成，点击“Finish”按钮即可。如图1-15所示。其中两个选项分别为“运行Git Bash”和“查看发行注释文件”。

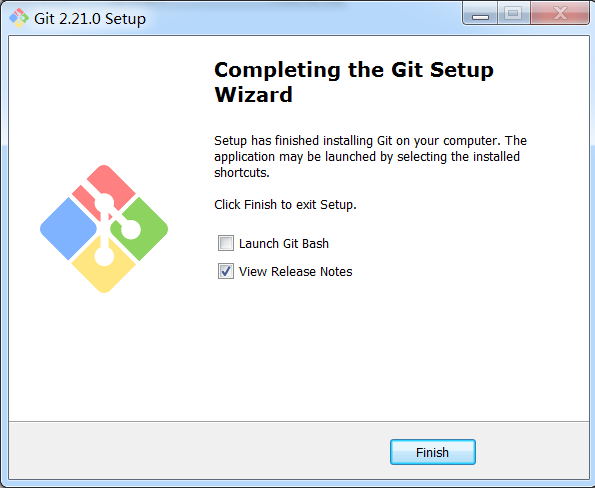


图1-15 安装完成页面

# 2 Git的使用

## 2.1配置用户名和邮箱

双击打开Git Bash，输入以下指令来配置用户名和邮箱：

git config --global user.name “你的用户名”

git config --global user.email “你的邮箱”

需要注意的是，这些指令中的空格以及“-”不能少输或输错，否则可能会失败。

配置用户名和邮箱如图2-1所示。

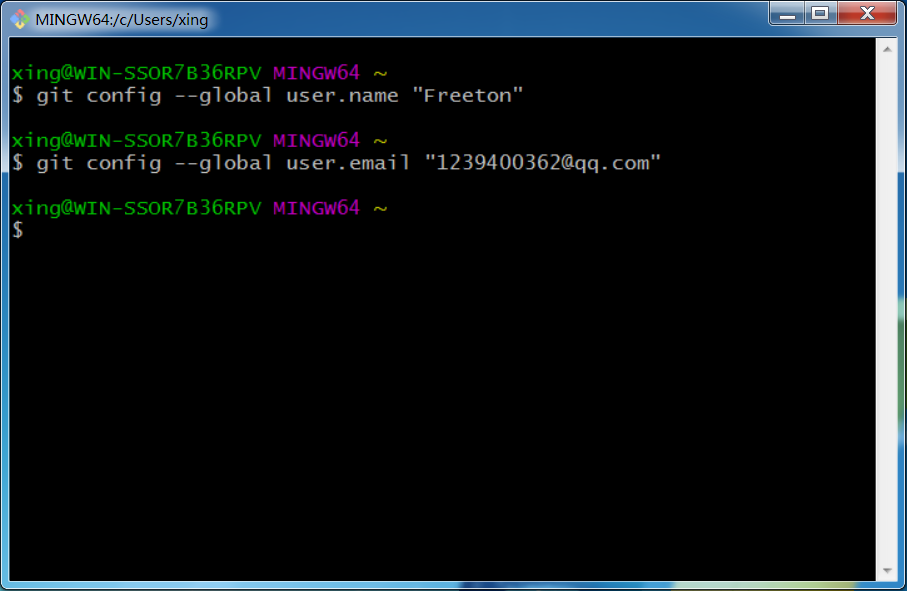


图2-1 配置用户名和邮箱

## 2.2 Git基本概念

工作区、暂存区和版本库的概念：

工作区：就是你在电脑里能看到的目录。通俗的说，就是存在硬盘里可以看见的目录。

暂存区：英文叫stage， 或index。一般存放在 ".git目录下" 下的index文件（.git/index）中，所以我们把暂存区有时也叫作索引（index）。

版本库：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。工作区、版本库中的暂存区和版本库之间的关系如图2-2所示。

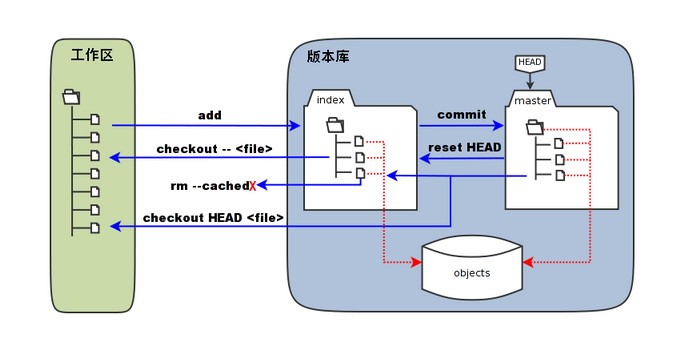


图2-2 工作区、暂存区和版本库

图中左侧为工作区，右侧为版本库。在版本库中标记为 "index" 的区域是暂存区（stage， index），标记为 "master" 的是 master 分支所代表的目录树。此时 "HEAD" 实际是指向 master 分支的一个"游标"。所以图示的命令中出现 HEAD 的地方可以用 master 来替换。

图中的 objects 标识的区域为 Git 的对象库，实际位于 ".git/objects" 目录下，里面包含了创建的各种对象及内容。

当对工作区修改（或新增）的文件执行 "git add" 命令时，暂存区的目录树被更新，同时工作区修改（或新增）的文件内容被写入到对象库中的一个新的对象中，而该对象的ID被记录在暂存区的文件索引中。

当执行提交操作（git commit）时，暂存区的目录树写到版本库（对象库）中，master 分支会做相应的更新。即 master 指向的目录树就是提交时暂存区的目录树。

当执行 "git reset HEAD" 命令时，暂存区的目录树会被重写，被 master 分支指向的目录树所替换，但是工作区不受影响。

当执行 "git rm --cached <file>" 命令时，会直接从暂存区删除文件，工作区则不做出改变。

当执行 "git checkout ." 或者 "git checkout -- <file>" 命令时，会用暂存区全部或指定的文件替换工作区的文件。这个操作很危险，会清除工作区中未添加到暂存区的改动。

当执行 "git checkout HEAD ." 或者 "git checkout HEAD <file>" 命令时，会用 HEAD 指向的 master 分支中的全部或者部分文件替换暂存区和以及工作区中的文件。这个命令也是极具危险性的，因为不但会清除工作区中未提交的改动，也会清除暂存区中未提交的改动。

## 2.3创建版本库

语法：git init

通过git init语句可以在某个目录中生成一个版本库，目录下会多了一个.git的文件夹，这个目录是Git来跟踪管理版本的，没事千万不要手动乱改这个目录里面的文件，否则，会把git仓库给破坏了。而选择路径的方式有两种，一种是通过打开桌面的Git Bash，通过“cd + 路径”和“mkdir + 文件夹”来到你输入的路径或者创建文件夹，如图2-3所示。

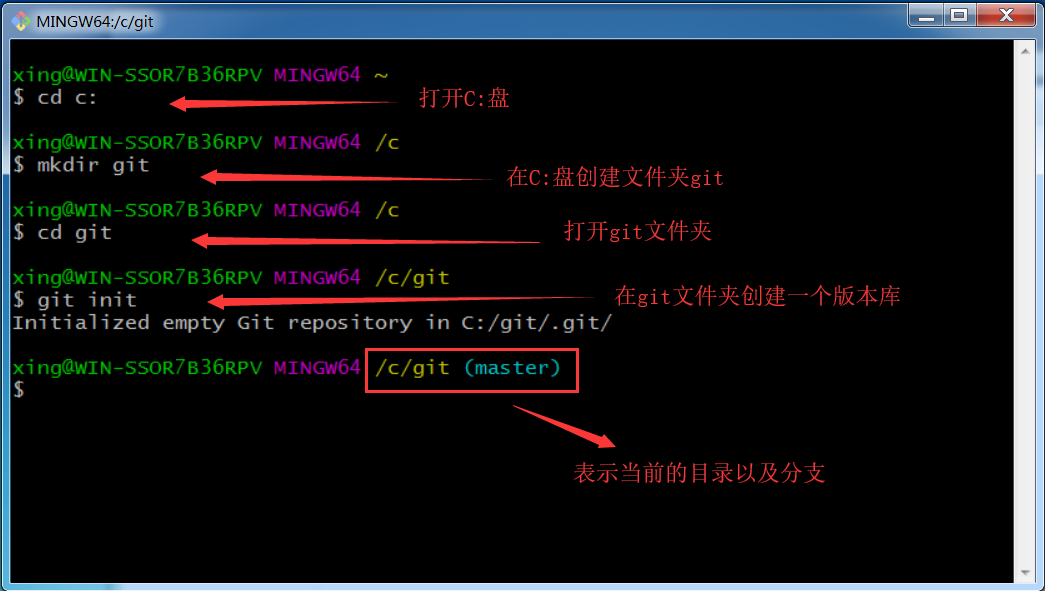


图2-3 通过方法一建立版本库

另一个方法是直接打开文件夹，在该文件夹单击鼠标右键，点击Git Bash Here就可以在当前目录打开git，如图2-4、图2-5所示。

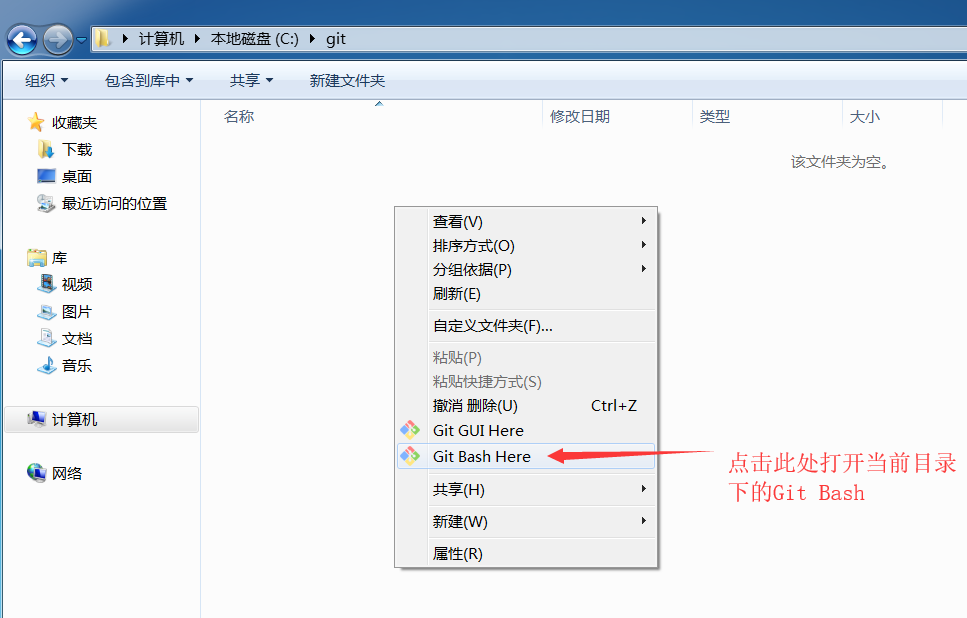


图2-4 打开当前目录的Git Bash

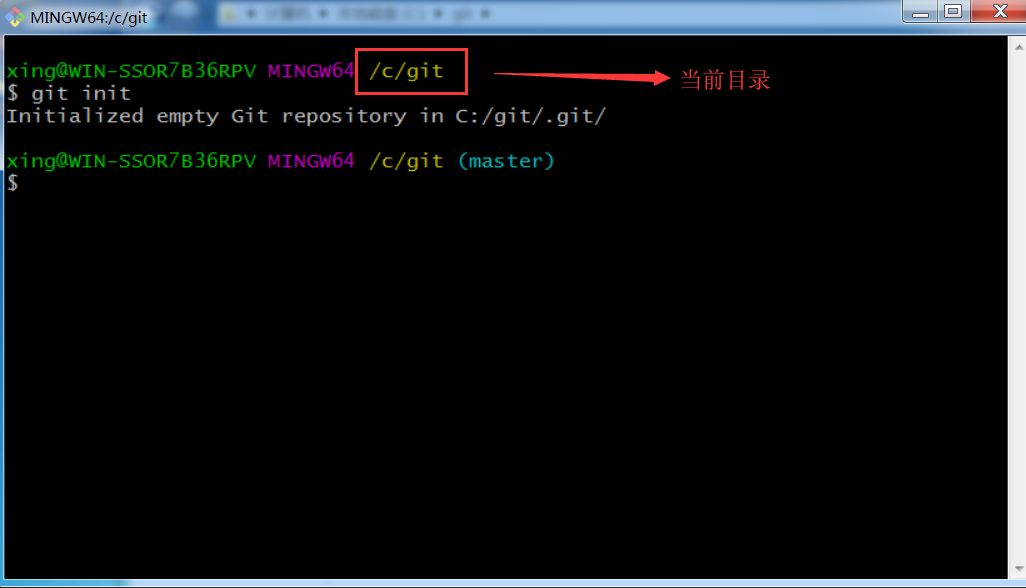


图2-5 方法二创建版本库

需要注意的是，.git文件夹是隐藏的，如果要看到，需要在文件夹选项中设置。在Windows 7操作系统中，通过“计算机-->组织-->文件夹和搜索选项-->查看-->显示隐藏的文件、文件夹和驱动器”可以显示隐藏文件夹。在Windows 10操作系统中，通过“此电脑-->查看-->在复选框中勾选隐藏的项目”即可显示隐藏文件夹。

## 2.4基本操作

为了更好地对命令的功能进行演示，在刚刚新建的版本库的路径（即.git文件夹所在的目录）下创建一个文本文档test.txt，文本文档的内容为一行“这里是第一行”。如图2-6所示。

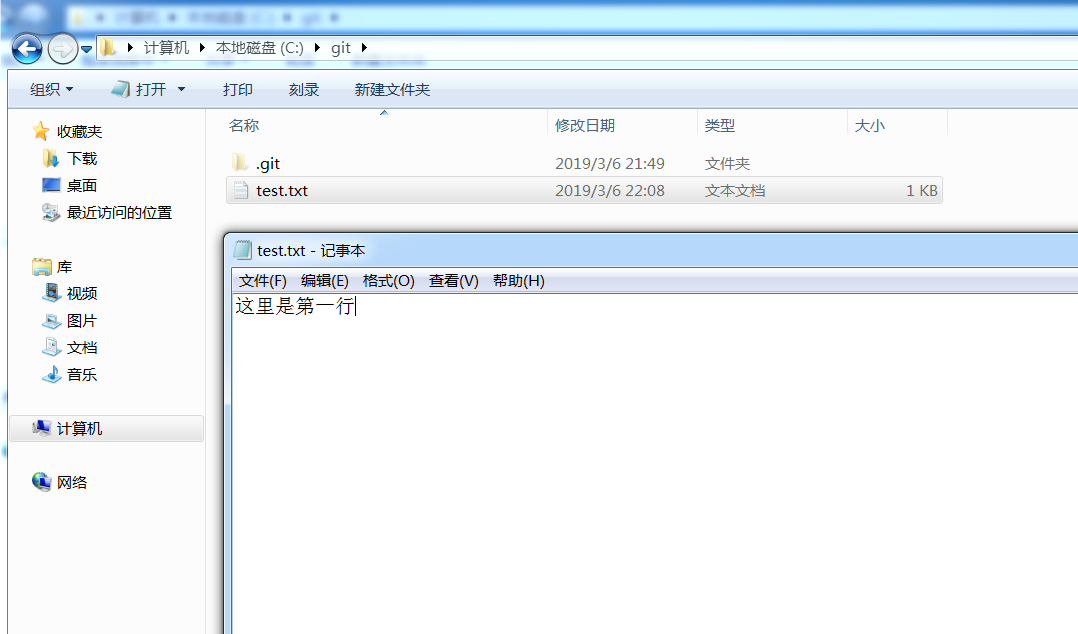


图2-6 新建文本文档

### 2.4.1将文件添加到暂存区

命令：git add test.txt

通过这个命令，可以将文件添加到暂存区，如果添加之后没有报错，则说明添加到暂存区成功。如图2-7所示。



图2-7 添加文件到暂存区

### 2.4.2将文件添加到仓库

命令：git commit -m “备注信息（即为这次提交这个动作取个名字）”

通过这个命令，可以将暂存区中的文件添加到仓库中。其中引号中的备注信息，表示对这次提交的备注信息，让你下次查看的时候，能够直观地知道这次提交的是什么文件。如图2-8所示。

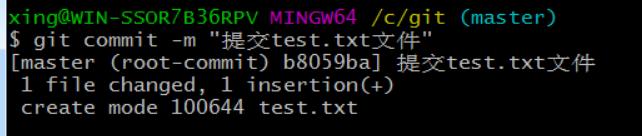


图2-8 将文件添加到仓库

### 2.4.3查看工作目录和暂存区的状态

命令：git status

通过这个命令，可以查看工作目录和暂存区的状态。现在使用该命令查看，会发现没有可改变的状态，如图2-9所示。

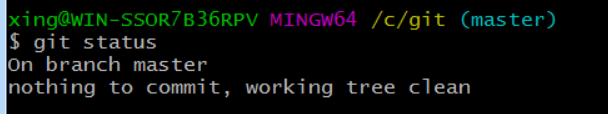


图2-9 无改动时查看状态

然后，将test.txt文件，在内容中添加一行数字“11111111”，再在Git Bash中通过该命令查看状态，会发现提示了工作目录有改动，如图2-10所示。该命令告诉我们 test.txt文件已被修改，但是是未被提交的修改。

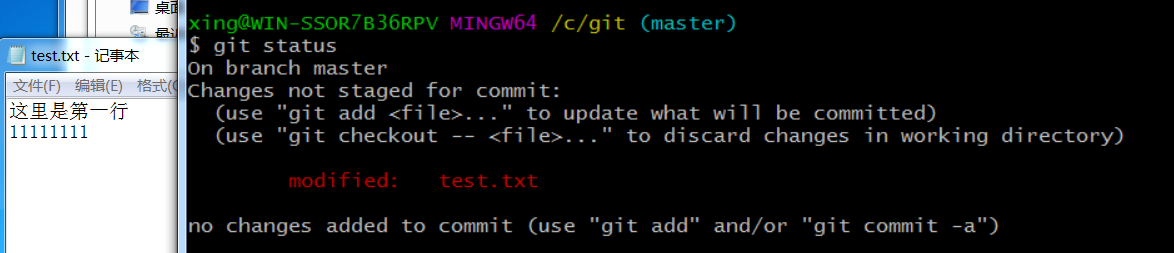


图2-10 工作目录中的内容有所改动后再查看状态

### 2.4.4查看文件修改的内容

命令：git diff test.txt

使用该命令可以查看未提交的修改后的文件和存在于仓库中的同一文件的差异，即查看文件修改的内容，如图2-11所示。



图2-11 查看文件修改的内容

了解了修改前后的内容之后，可以将修改后的文件也发送到仓库中，步骤见2.4.1和2.4.2，如图2-12所示。为了便于操作，再在test.txt中第三行添加一行字母aaaaaaaa，并提交到仓库。

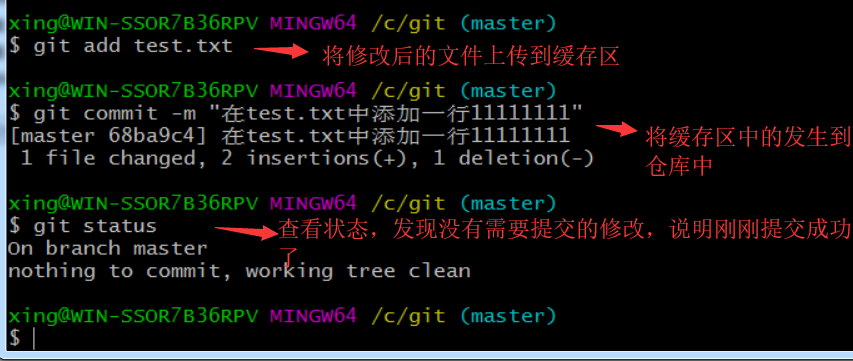


图2-12 将修改后的文件提交到仓库中

### 2.4.5查看历史记录和版本回退

查看历史记录：

命令：git log 或者 git log --pretty=oneline

其中使用git log可以查看历史记录的版本信息，信息显示比较全同时比较乱，而git log --pretty=oneline则是每个历史记录在显示的信息更简洁。我们可以看到最近三次提交。如图2-13所示。



图2-13 查看历史记录

版本回退：

命令：git reset --hard HEAD^ 或者 git reset --hard HEAD^^ 或者 git reset --hard HEAD~数字

通过这些命令，可以将版本回退到上个版本、上上个版本或者指定的某个版本。如图2-14所示。

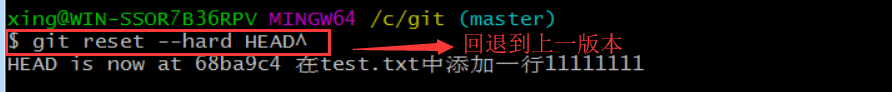


图2-14 回退到上一版本

通过 cat test.txt可以查看当前版本的test.txt中的内容。如图2-15所示。

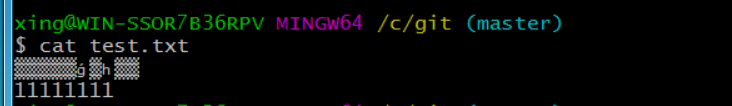


图2-15 查看当前版本的文件内容

通过git log可查看日志，会发现刚刚增加的第三行aaaaaaaa的日志不见了，如图2-16所示。

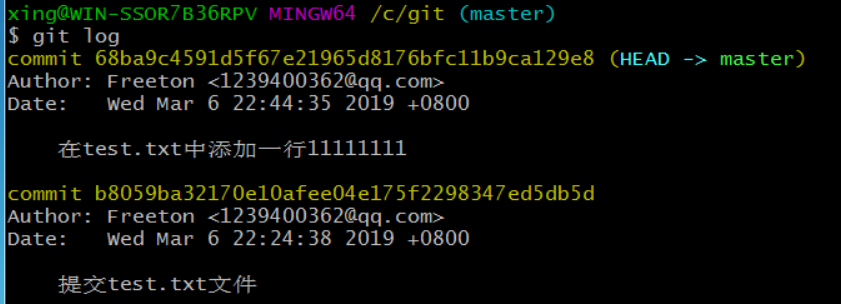


图2-16 查看日志

可以通过回退到某个版本号来恢复到增加第三行aaaaaaaa的内容。通过git reflog来获得版本号，并通过“git reset --hard 版本号”来回退到有aaaaaaaa的版本，如图2-17所示。需要注意的是，这里涉及的所有版本回退即修改，都是针对工作目录的，即你版本更改之后，可以看到在文件夹中的test.txt文件中的内容也会相应改变。

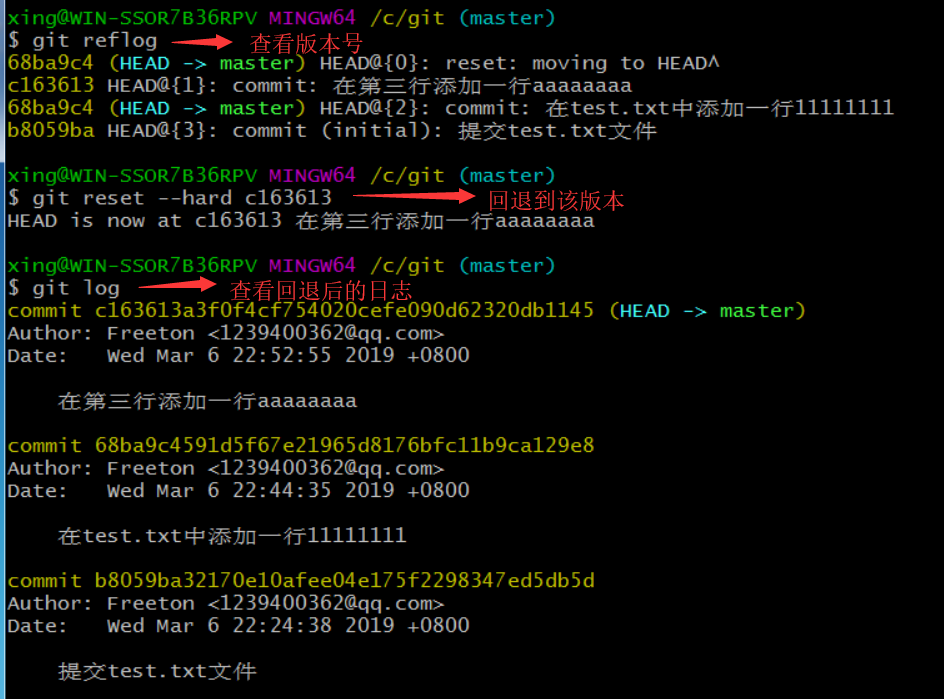


图2-17 通过版本号回退版本

### 2.4.6 Git撤销修改和删除文件操作

撤销修改：

命令：git checkout -- test.txt

把test.txt文件在工作区做的修改全部撤销，这里有2种情况，如下：

test.txt自动修改后，还没有放到暂存区，使用撤销修改就回到和版本库一模一样的状态。比如，在test.txt文件中，再加一行44444444并保存，然后在Git Bash中输入git checkout -- test.txt，再打开test.txt可以发现刚刚加的一行44444444没有了，说明撤销了刚刚的修改。如图2-17所示。

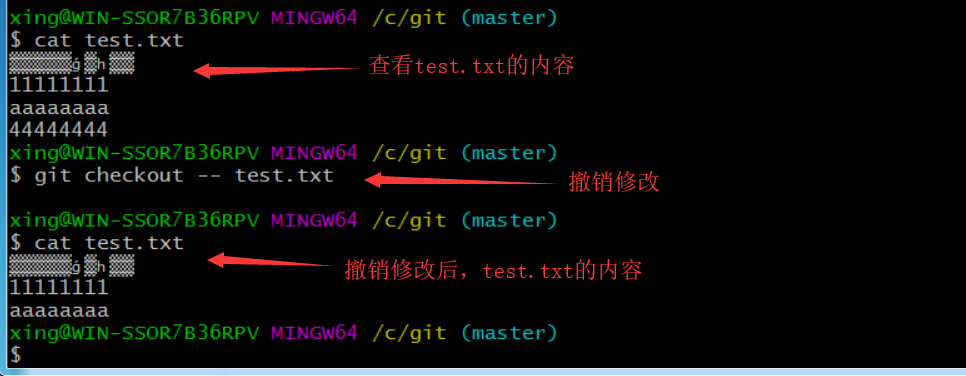


图2-17 撤销修改

另外一种是test.txt已经放入暂存区了，接着又作了修改，撤销修改就回到添加暂存区后的状态。如图2-18所示。

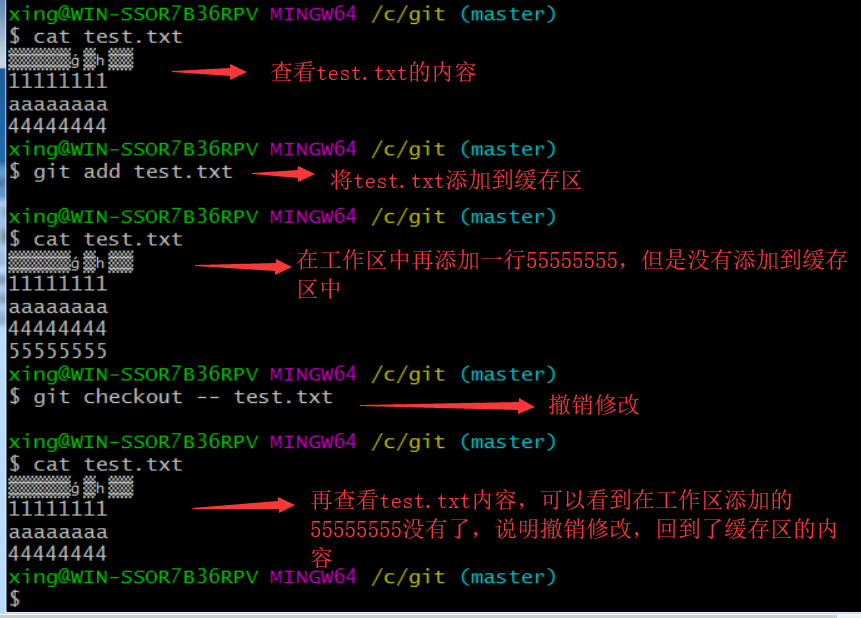


图2-18 在已经提交到暂存区后的撤销修改

删除文件：

命令：rm 文件名（要包括扩展名）

通过这个命令可以删除文件，如图2-19所示。



图2-19 删除文件

此时可以通过git commit -m指令提交掉，也可以通过git checkout a.txt撤回刚刚的删除指令。

## 2.5分支

每次提交，Git都把它们串成一条时间线，这条时间线就是一个分支。截止到目前，只有一条时间线，在Git里，这个分支叫主分支，即master分支。

### 2.5.1创建与合并分支

创建分支：

命令：“git branch 分支名（该命令表示创建一个分支） git checkout 分支名（该命令表示切换到该分支）”或者“git checkout -b 分支名（该语句表示创建一个分支并跳转到该分支）”

使用该命令可以创建一个分支，并且切换到这个分支上。使用git branch命令可以查看所有分支，并且在当前分支上会有一个星号（\*）标注，如图2-20所示。

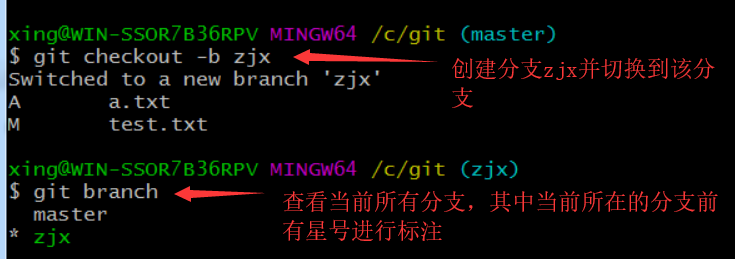


图2-20 创建并切换分支和查看所有分支

合并分支：

命令：git merge zjx（在某分支下使用，表示将zjx分支合并到该分支）

使用该命令，可以将zjx分支合并到master。先在zjx分支中，在test.txt文件中再加一行66666666，然后提交到仓库，如图2-21所示。

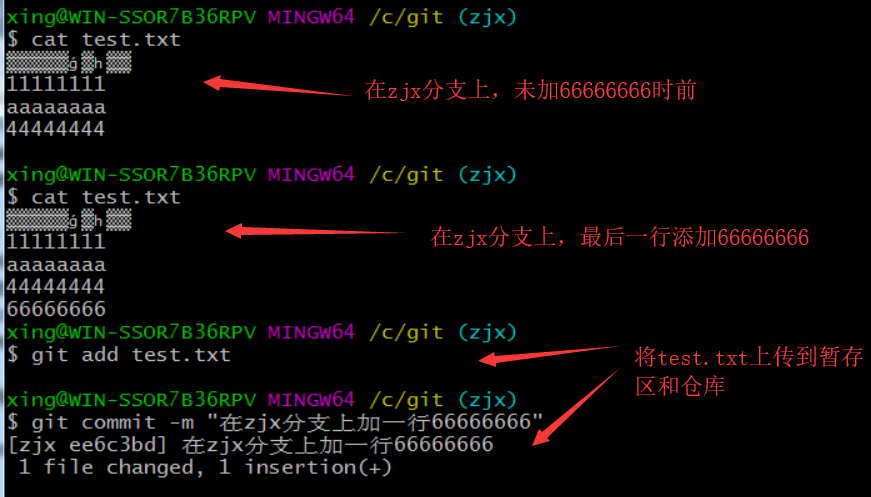


图2-21 在zjx分支上添加修改并上传

再切换到master分支，通过cat test.txt命令可以看到并没有刚刚添加的66666666，然后再将zjx分支的内容合并到master中，再查看内容，会发现66666666出现了，如图2-22所示。

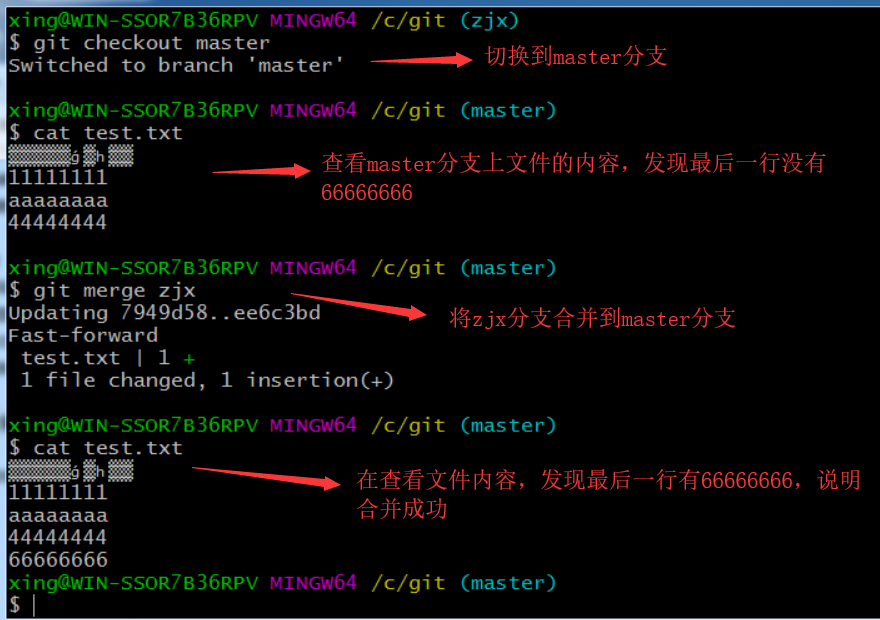


图2-22 合并分支

### 2.5.2删除分支

命令：git branch -d 分支名

使用该命令会将某分支删除，但是如果该分支没有和主分支合并，会出现要再输入“git branch -D 分支名”来确定删除。

### 2.5.3解决冲突

有时，在两个分支上都修改了该文件，在合并的时候会产生冲突，比如在master分支上，最后一行加上77777777并上传到仓库中，然后再zjx分支上，在最后一行加上88888888并上传到仓库中，则在合并分支时，就会产生冲突。如图2-23所示。

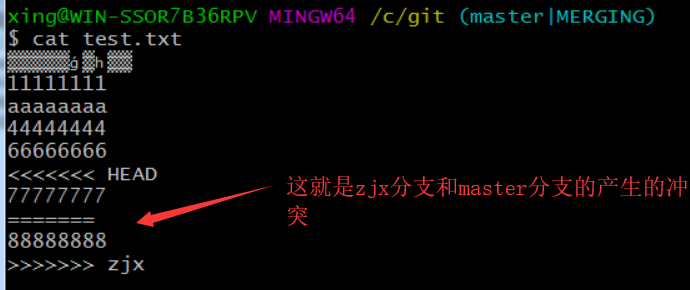


图2-23 产生冲突

在工作区将文件内容改为和master分支上的文件内容一样，即可解决冲突。如图2-24所示。

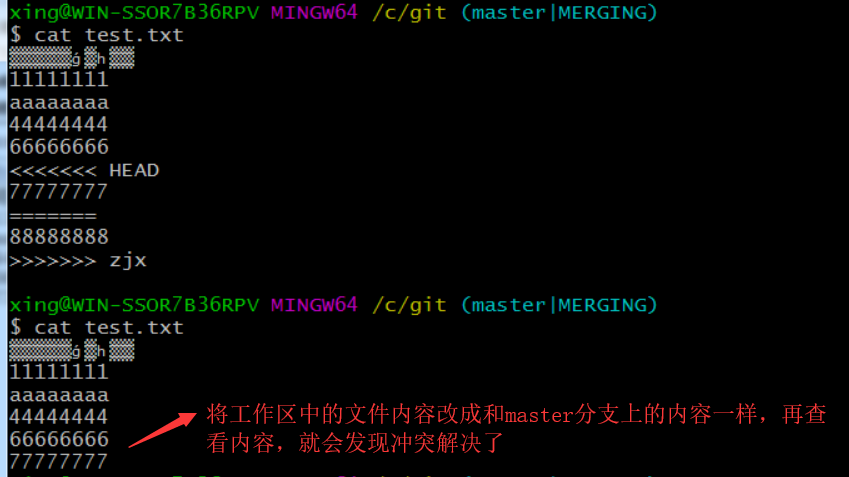


图2-24 解决冲突

### 2.5.4分支管理策略

　　通常合并分支时，git一般使用”Fast forward”模式，在这种模式下，删除分支后，会丢掉分支信息，现在我们来使用带参数 –no-ff来禁用”Fast forward”模式。首先我们来做demo演示下：

1.创建一个dev分支。

2.修改readme.txt内容。

3.添加到暂存区。

4.切换回主分支(master)。

5.合并dev分支，使用命令 git merge --no-ff -m“注释”dev

6.查看历史记录

如图2-25所示。

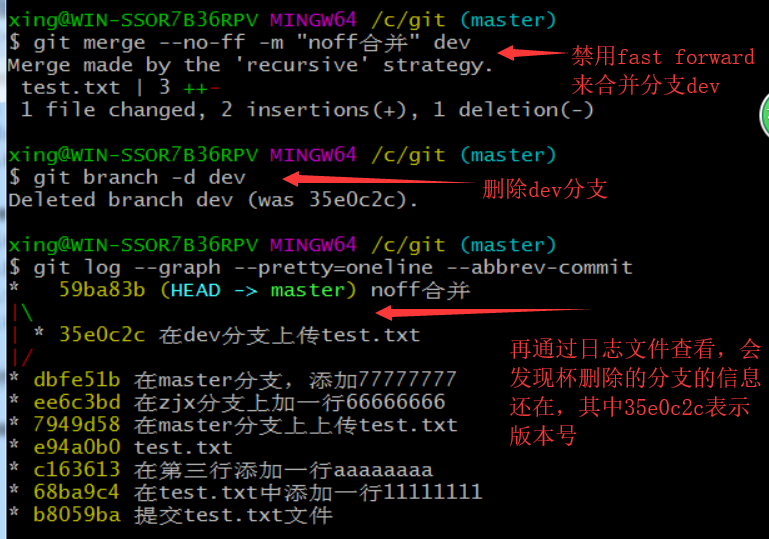


图2-25 禁用fast forward合并dev

分支策略：首先master主分支应该是非常稳定的，也就是用来发布新版本，一般情况下不允许在上面干活，干活一般情况下在新建的dev分支上干活，干完后，比如上要发布，或者说dev分支代码稳定后可以合并到主分支master上来。

### 2.5.5工作现场的隐藏及还原

命令：git stash（隐藏工作现场）、git stash list（工作现场列表）、git stash apply（恢复工作现场）

使用该命令，可以将工作现场保留下来，然后隐藏

开发过程中，遇到了bug，可以先将正在做的工作现场隐藏起来，然后通过建立临时分支来解决bug，然后将临时分支合并再删除即可。如图2-26、图2-27所示。

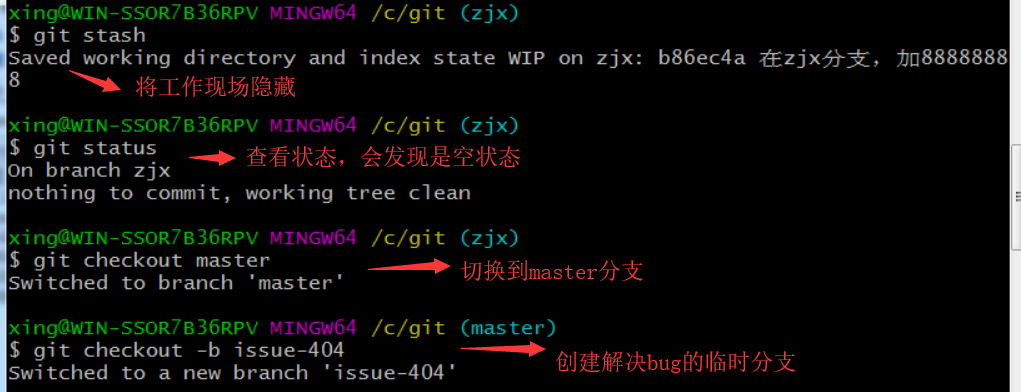


图2-26 隐藏工作现场



图2-27 恢复工作现场

## 2.6远程仓库（李建江）

第1步：创建SSH Key。在用户主目录（C：\Users\你的用户名）下，看看有没有.ssh目录，一般第一次安装使用Git的系统是没有此文件夹的。

如果没有则在Git安装目录中找到“ssh-keygen.exe”所在路径，并且添加到系统环境变量Path中。然后在Git Bash中运行“ssh-keygen –t rsa –C “youremail@example.com”，后一路回车，按照提示输入文件的完整路径，如果直接回车就是默认在用户主目录中.ssh文件夹中id\_rsa和id\_rsa.pub的SSH key的密钥对。如图2-28所示。

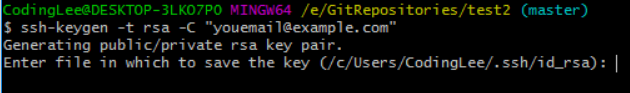


图2-28 创建密钥

然后按照提示输入密钥的密码，如图2-29所示。



图2-29 输入密钥的密码

id\_rsa是私钥，不能泄露出去id\_rsa.pub是公钥。如果想要在.ssh文件夹中生成其他名字的密钥，如“test”，可以执行如下命令：

ssh-keygen -t rsa -C "1805442532@qq.com" -f ~/.ssh/test

然后执行：

ssh-agent bash

ssh-add ~/.ssh/test

第2步：登录Github，点击Account settings，打开SSH and GPG Keys页面，新建SSH key，如图2-30所示。

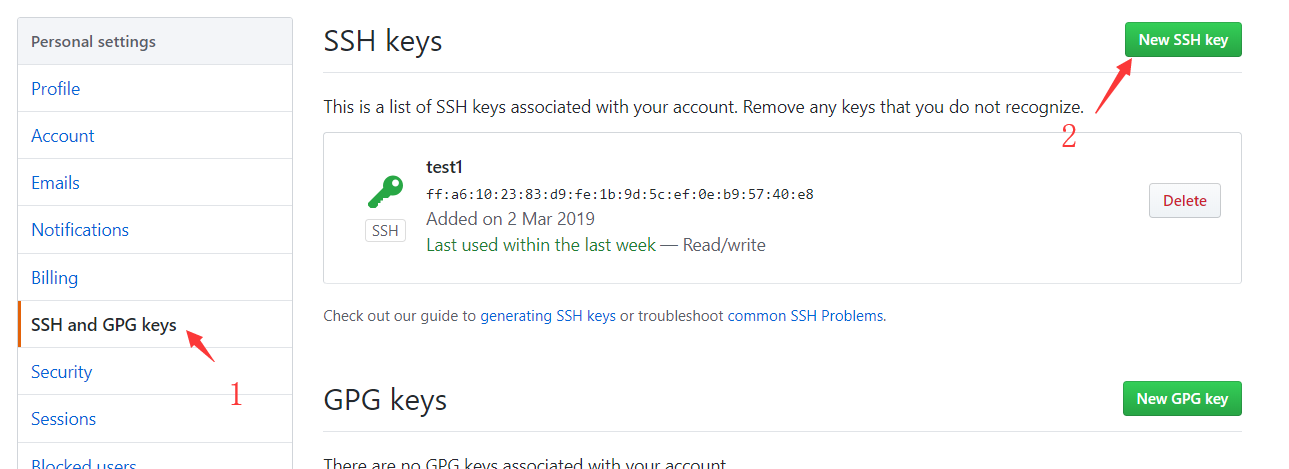


图2-30 新建SSH key

输入任意标题，在key文本框中填入id\_rsa.pub文件中的内容，如图2-31所示。

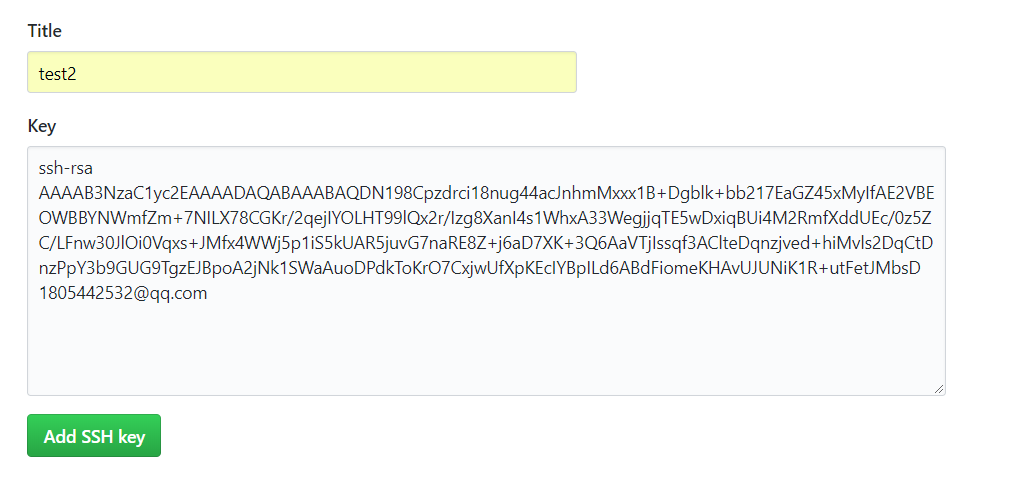


图2-31 设置标题和密钥

添加完成后如图2-32所示：

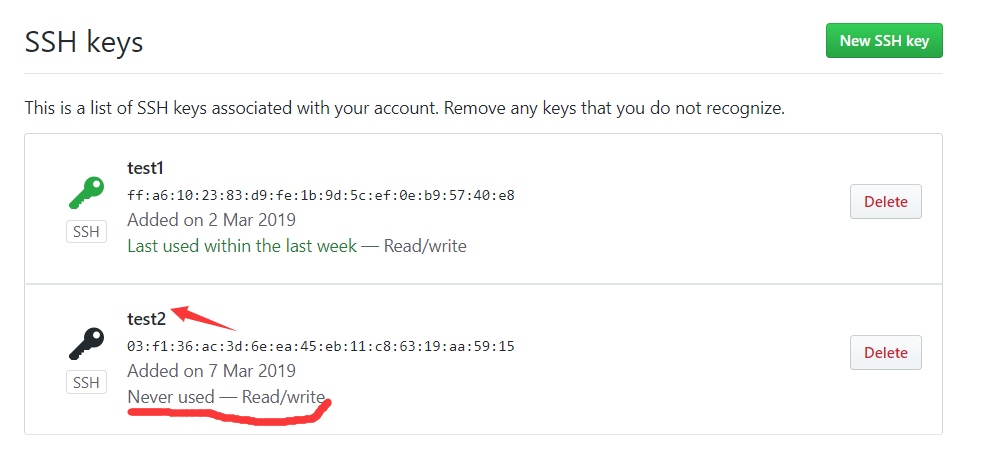


图2-32 新建SSH key完成

第3步：添加远程仓库并建立连接。

填写好仓库名字和其他选项后，点击创建仓库。如图2-33、图2-34所示。

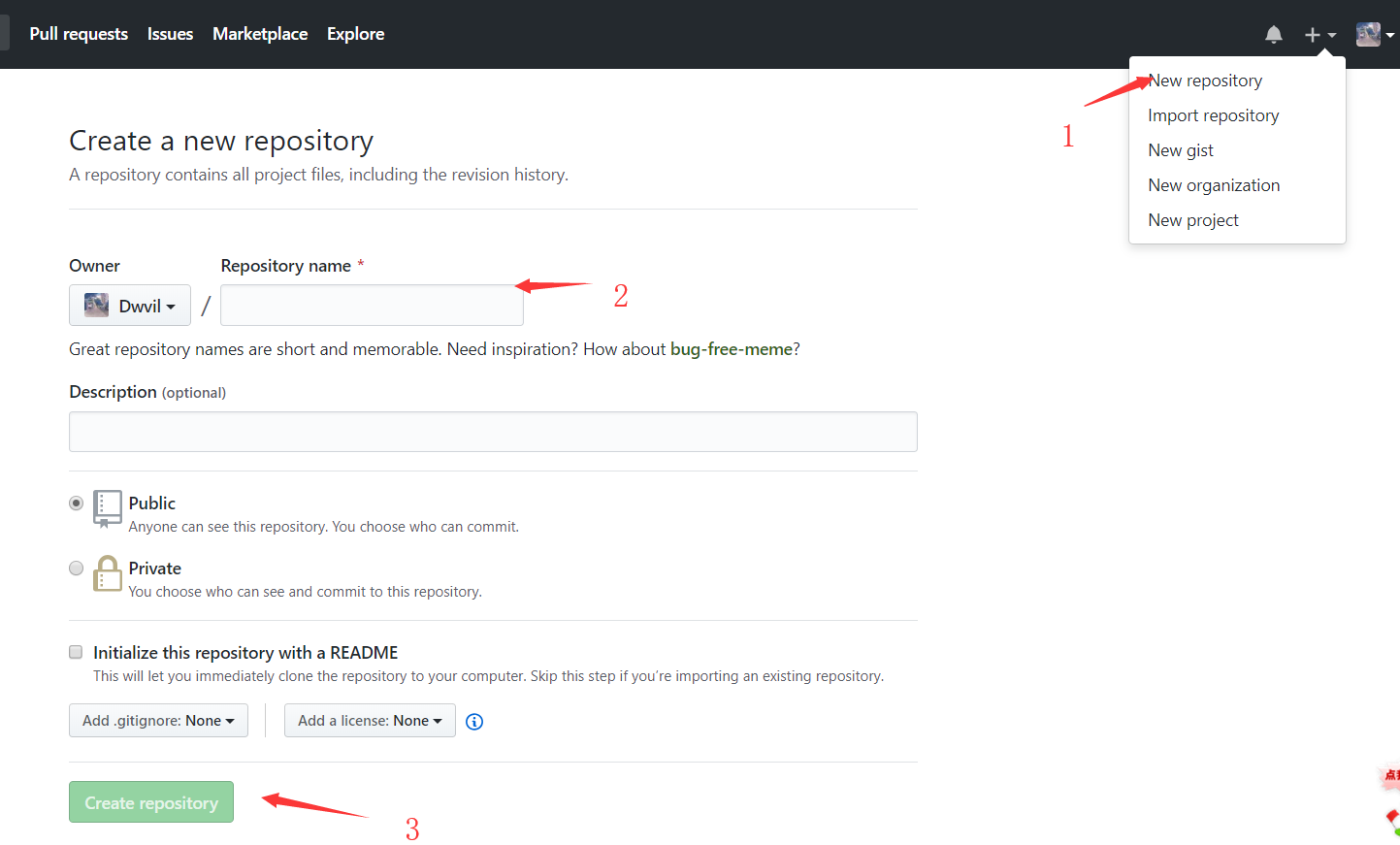


图2-33 新建仓库

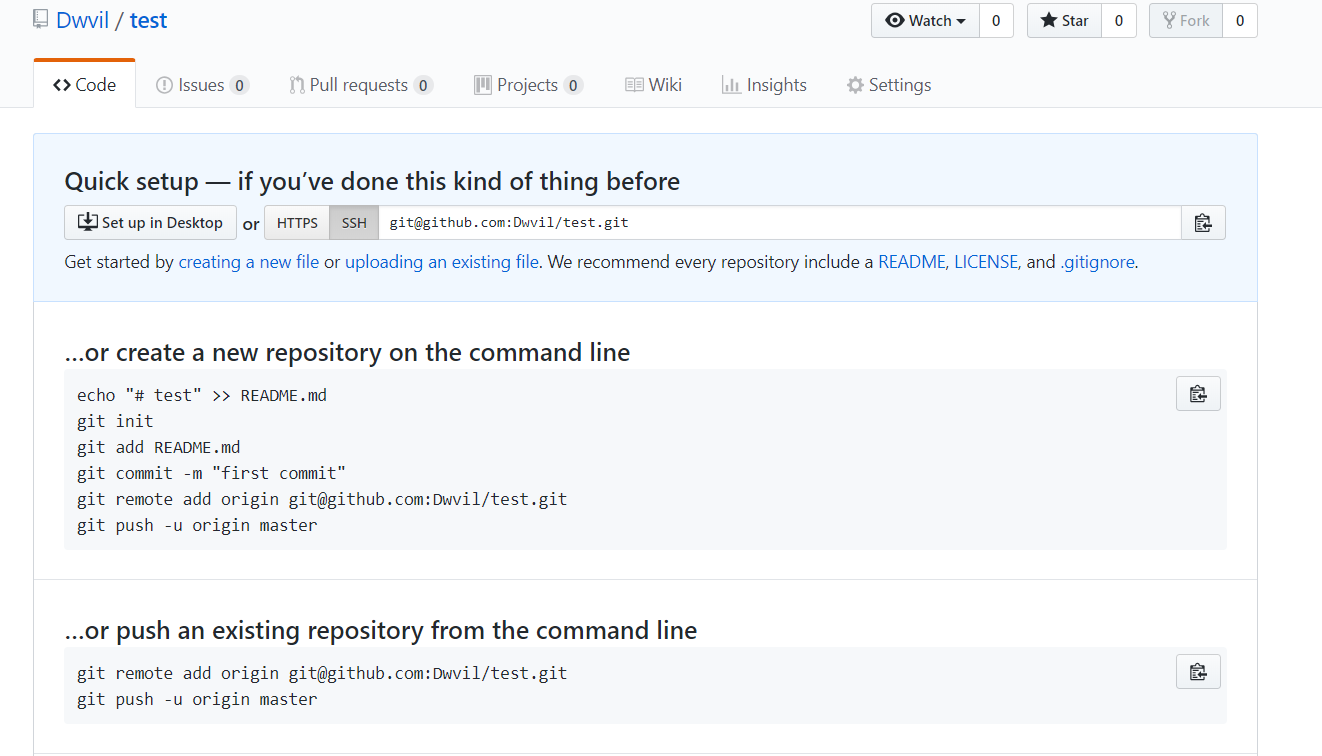


图2-34 仓库新建完成

目前，在GitHub上的这个test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Git Bash上运行下面的命令

git remote add origin [git@github.com：Dwvil/test.git](mailto:git@github.com:Dwvil/test.git)

其中origin为远程仓库名，一般默认使用 origin，可以改成自己想要的远程仓库名。

[git@github.com：Dwvil/test.git](mailto:git@github.com:Dwvil/test.git) 在这里，“github.com”为Hostname，代表改Hostname所对应的Host是github，类似的还有GItLab和Coding等代码托管平台。“Dwvil/test.git”为固定格式，“Dwvil”是GitHub的用户名，“test”对应的GitHub仓库名。

在运行完上面的命令后，可以使用“git remote”查看当前Git项目中的所有远程仓库名，使用“git remote rm 仓库名”可以移除指定远程仓库连接 。

最后使用“git push -u origin master”可以把master分支上git项目push到GitHub上的test仓库中。

## 2.7多人协作（斯文韬）

首先，在github中我们运用它的new organization（新建组织）功能来由项目组织者创建一个新的组织：[传送点](https://github.com/organizations/new)，如图2-35所示。

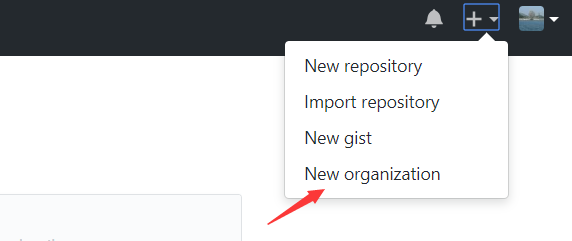


图2-35 选择创建新组织

接下来可以选择填写组织名以及Email，选择项目所需的计划，我们这里选择第一个免费的（私人项目需要付费），然后进入到下一步，如图2-36所示。

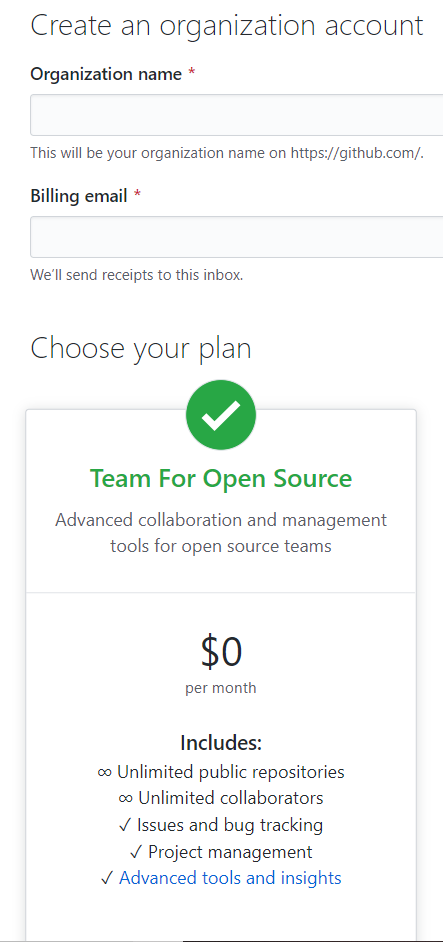


图2-36 填写组织信息

第二步邀请团队成员

这一步我们可以通过他们的github昵称邀请团队成员，也可以稍后在项目主页中进行邀请，如图2-37所示。

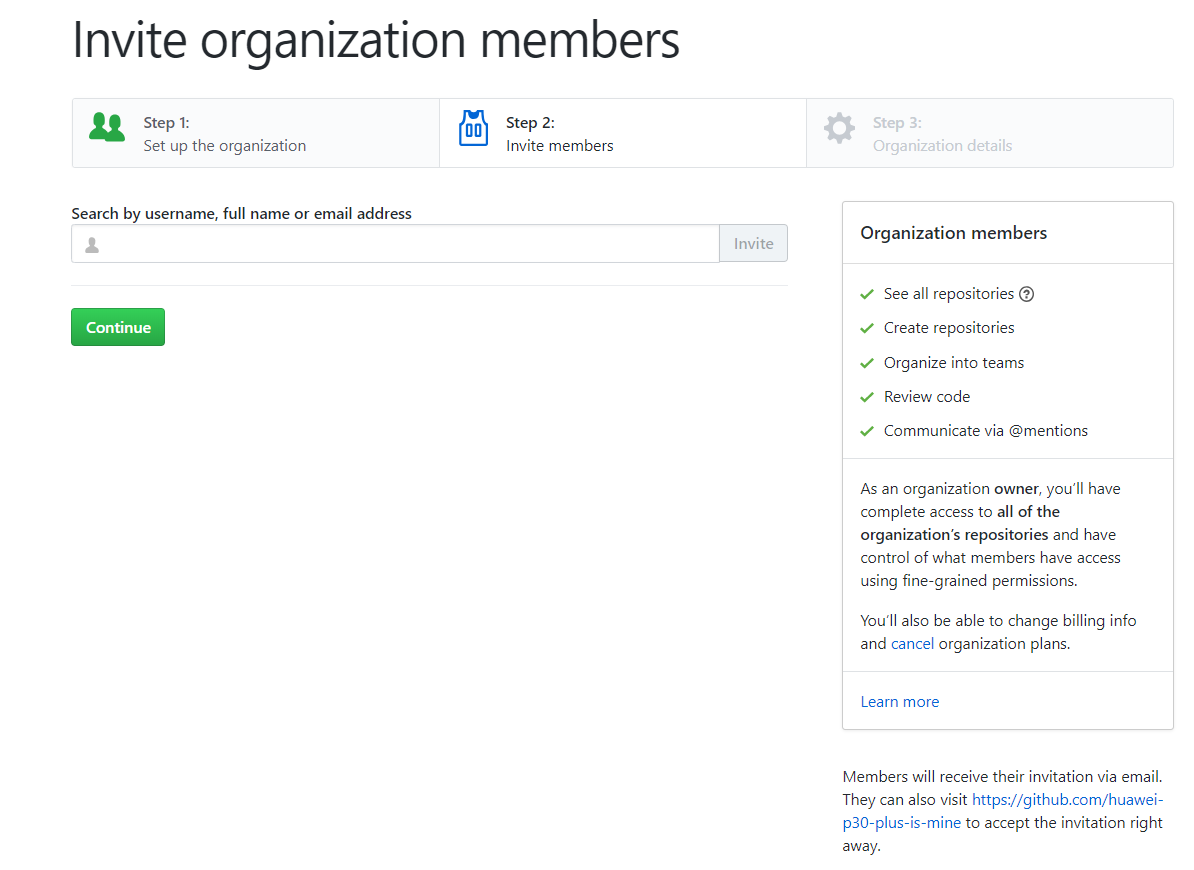


图2-37 邀请组织成员

接下来我们进入到组织的首页，如图2-38所示。

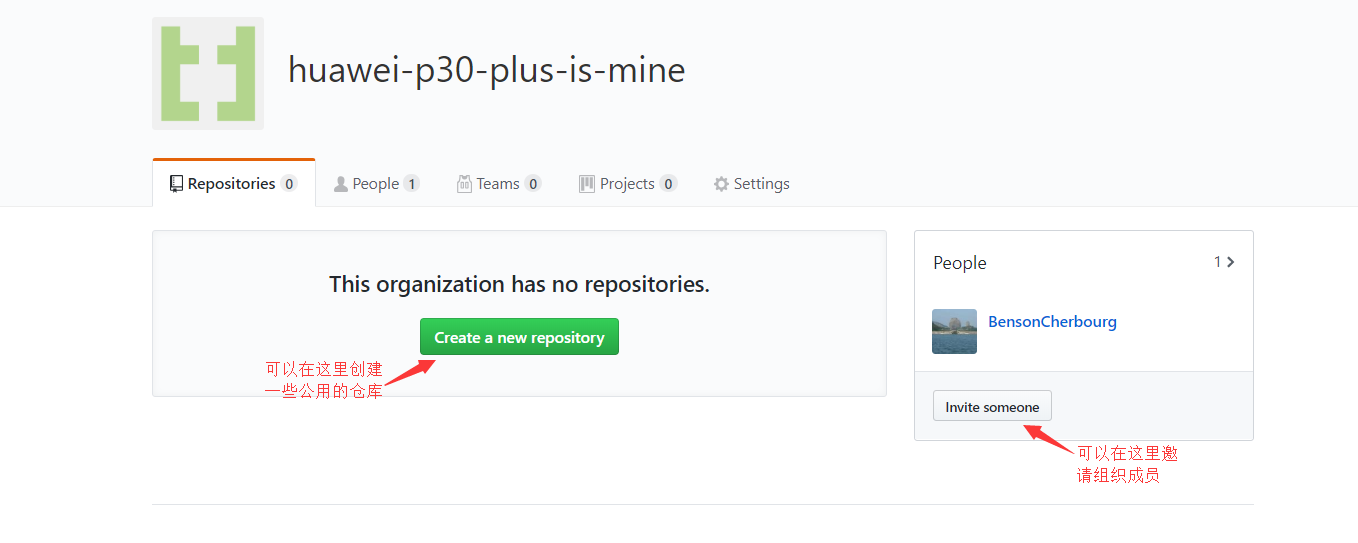


图2-38 进入组织首页

接下来我们首先需要在图4菜单栏中选择Settings，当每创建的一个新组织时处创建者外其余人员的默认权限是没有push权限（Read）的，需要手动改为Write或Admin权限，如图2-39所示。

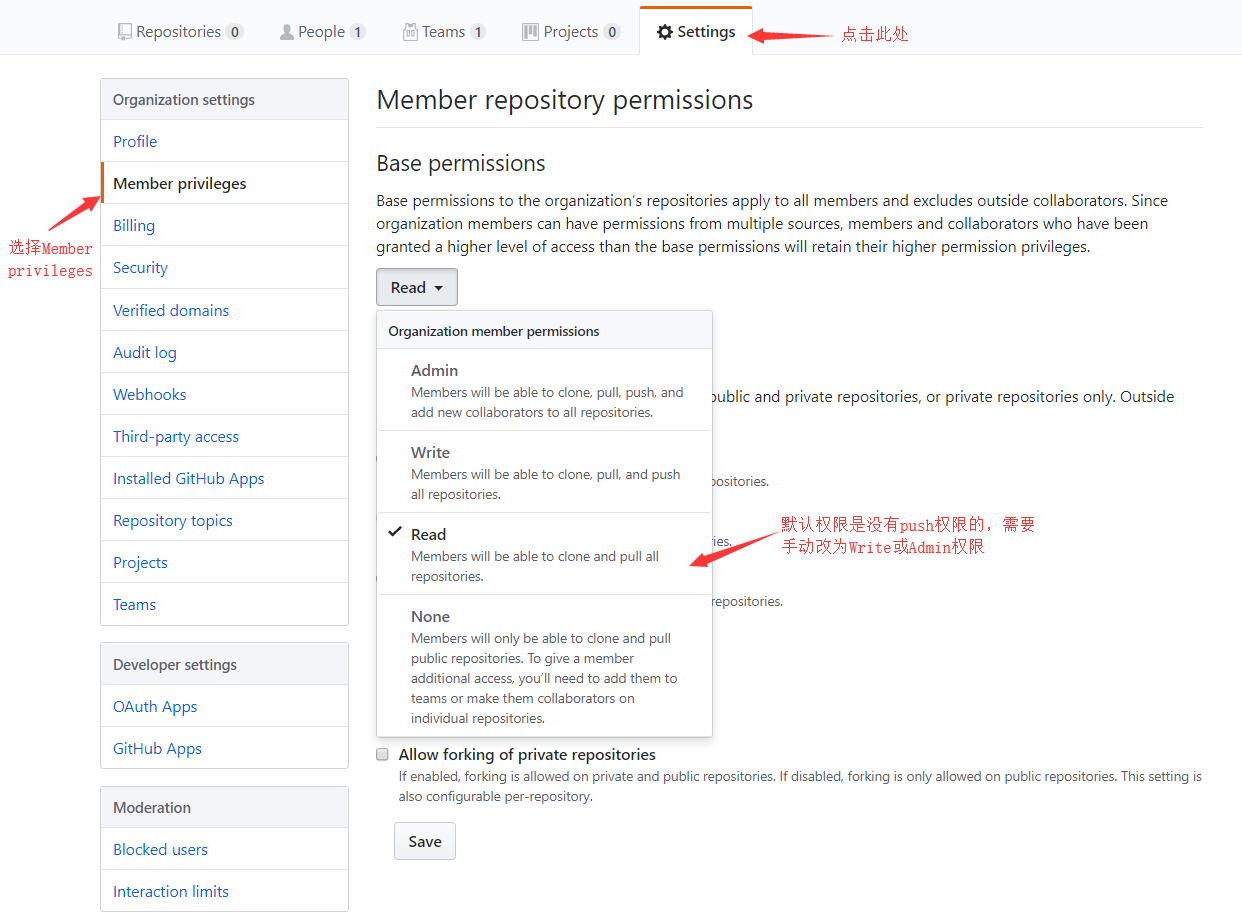


图2-39 设置权限

我们可以选择到team页面创建属于自己的团队，在GitHub中，一个组织可以分为多个小组，小组允许您与组内的其他人员共享设置权限，不同的组有不同的权限分配，可以极大的提高各分组的效率，下面我们可以来创建我们的开发团队，如图2-40、图2-41、图2-42所示。

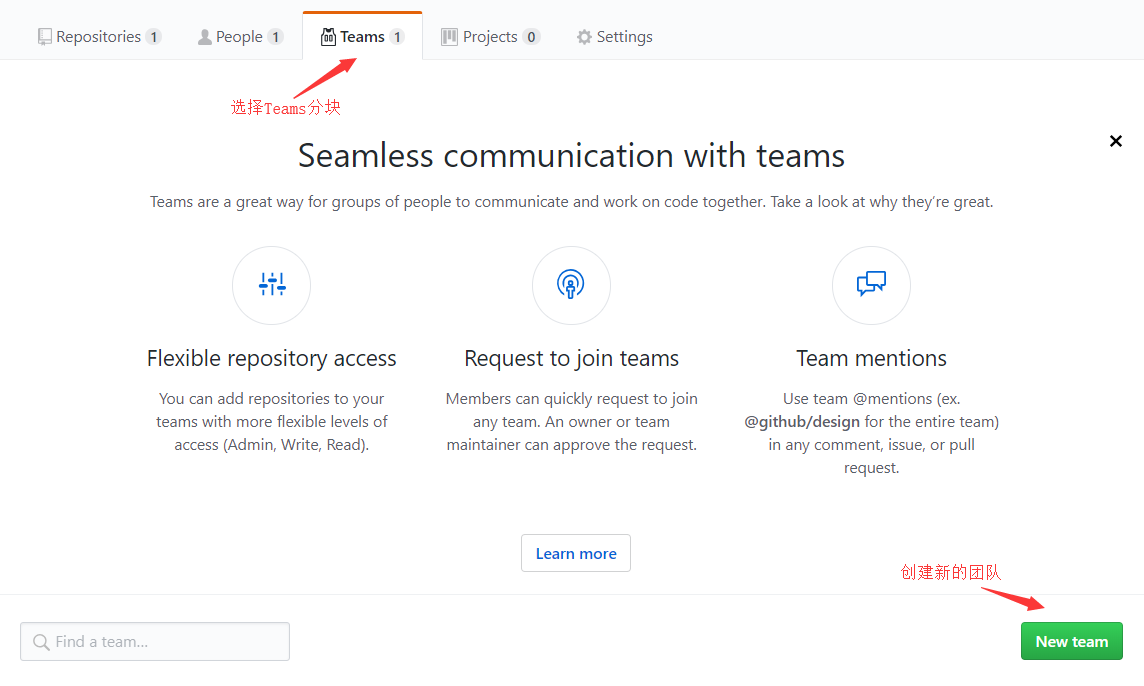


图2-40 团队页面视图

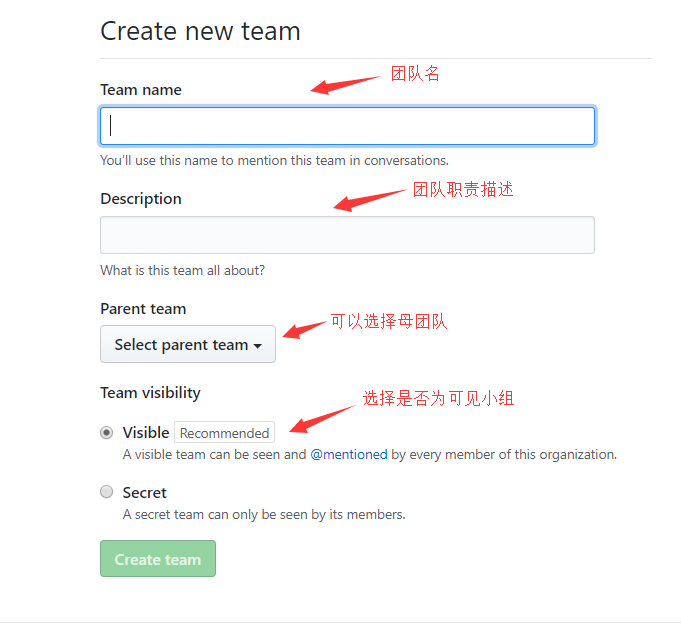


图2-41 创建团队

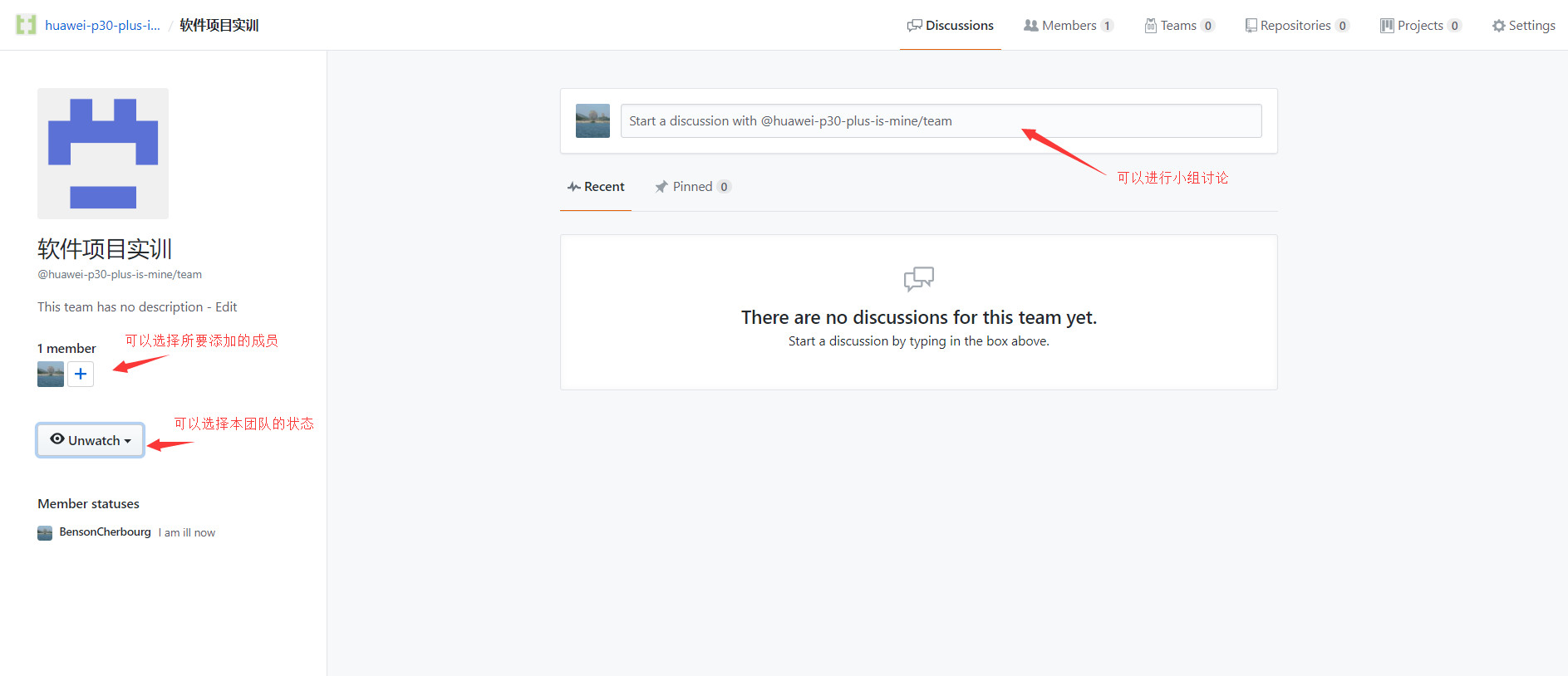


图2-42 团队界面

返回到首页，现在我们可以选择创建一个新的仓库来存储你本地仓的文件，如图2-43所示。

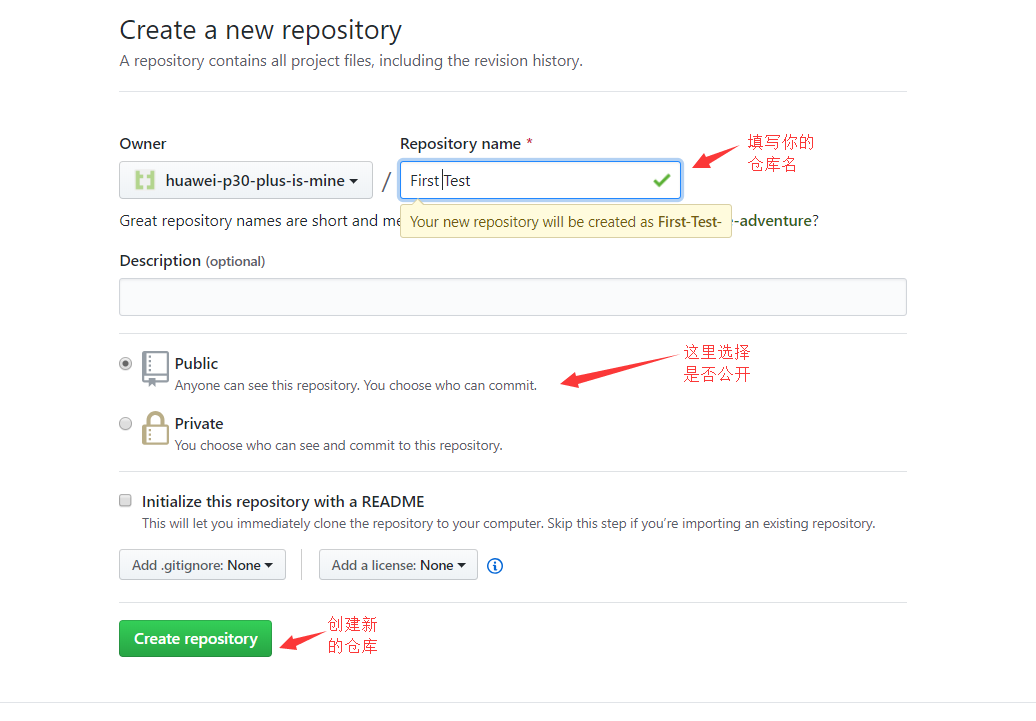


图2-43 创建新的仓库

建立好仓库后会进入到repository界面，如图2-44所示。

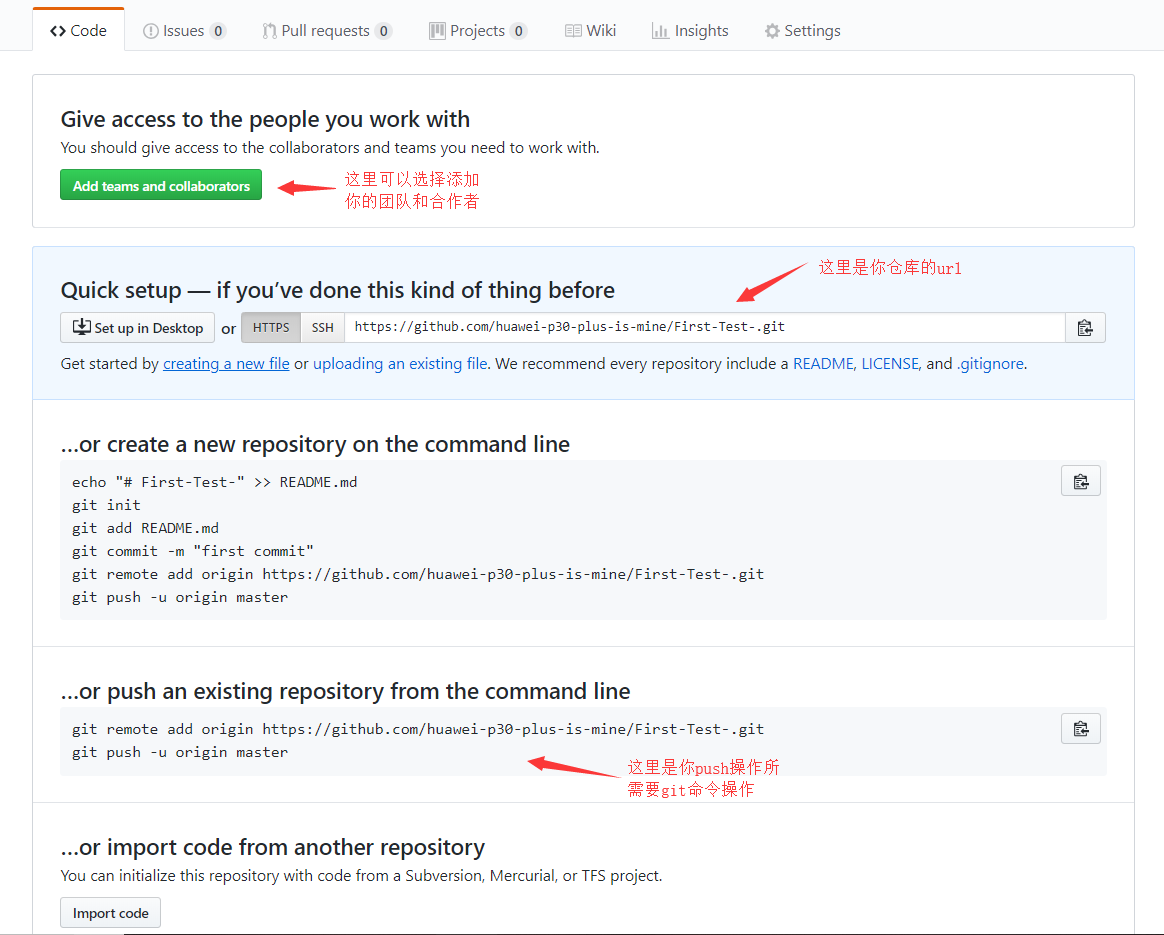


图2-44 repository初始界面

接下来我们可以点击图7 的Add teams and collaborators按钮，进入到选择添加成员页面，并将上面已经创建好的team加入到本仓库中，并可对不同的项目进行权限划分。如图2-45、图2-46所示。

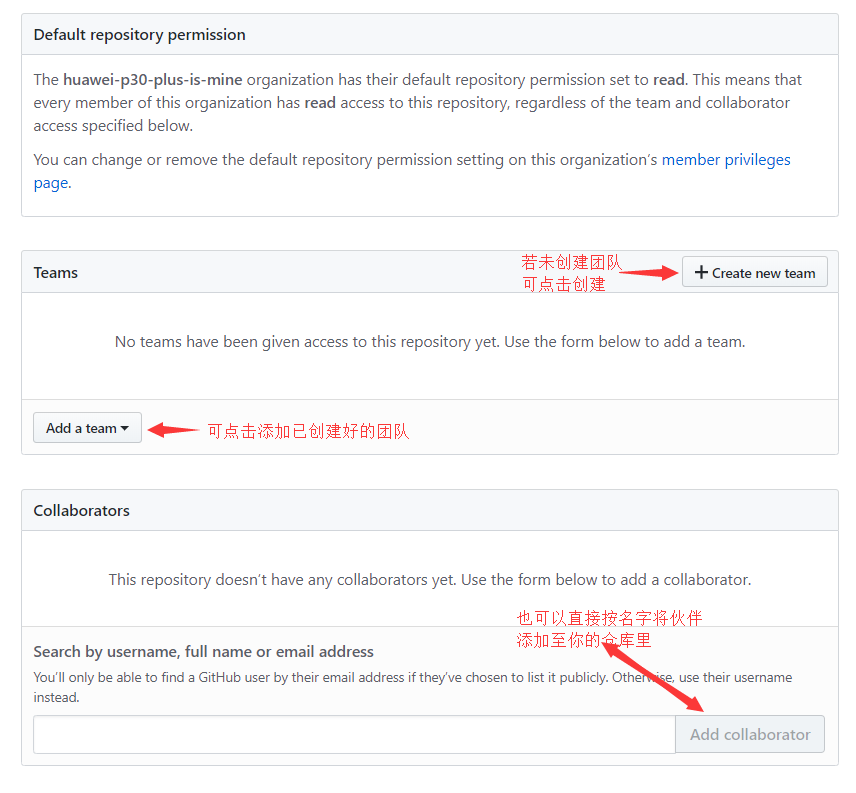


图2-45 添加team

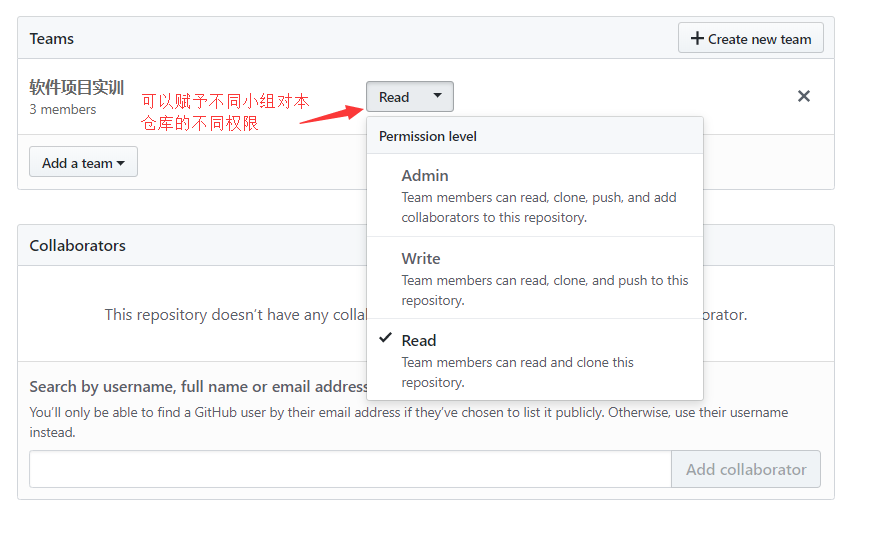


图2-46 添加team

# 3组员操作记录

在多人协作中，每个组员都将已建立好的一个远程仓库克隆到自己的文件夹中，并自己再创建一个文件，在上传到远程仓库中。

张经鑫：

克隆远程仓库如图3-1所示。

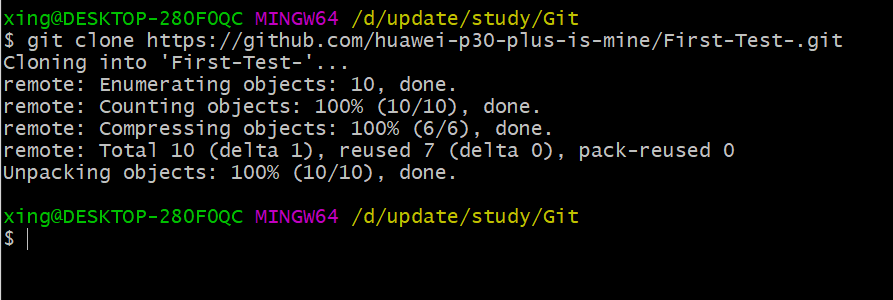


图3-1 克隆课程仓库

上传自己的文件，并上传到远程仓库。如图3-2所示。

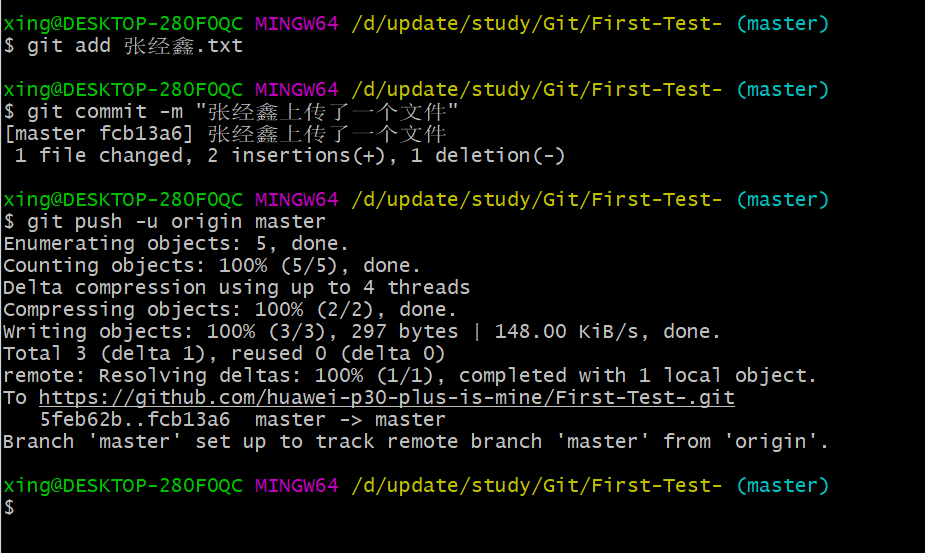


图3-2 将本地仓库上传到远程仓库

斯文韬：

上传自己的文件，并上传到远程仓库。如图3-3所示。

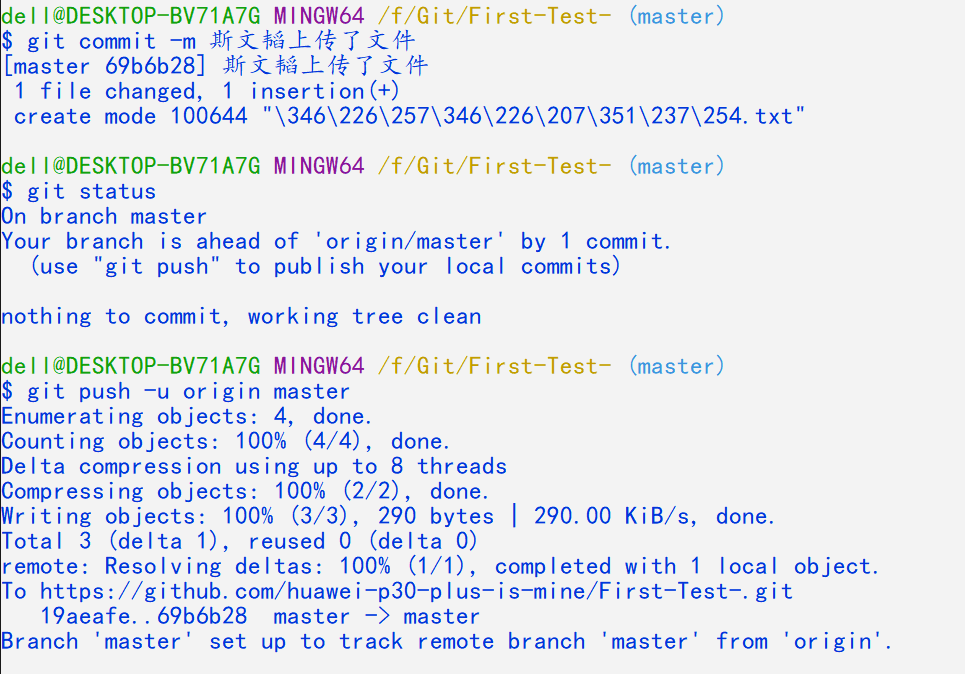


图3-3 将本地仓库上传到远程仓库

李建江：

上传自己的文件，并上传到远程仓库。如图3-4所示。

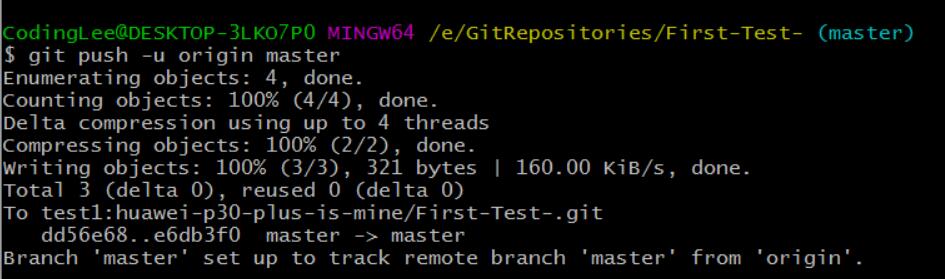


图3-4 将本地仓库上传到远程仓库

谢太政：

上传自己的文件，并上传到远程仓库。如图3-5所示。

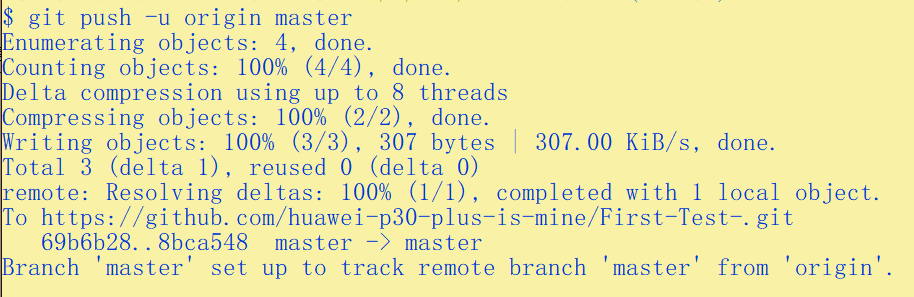


图3-5 将本地仓库上传到远程仓库

王良贤：

上传自己的文件，并上传到远程仓库。如图3-6所示。

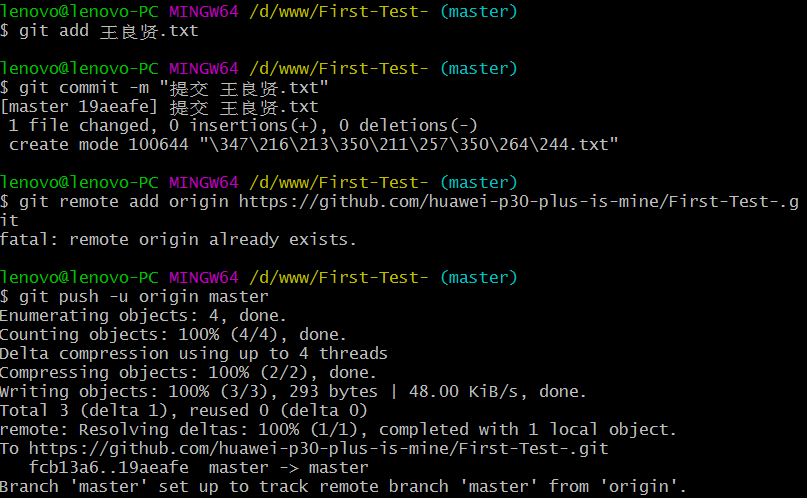


图3-6 将本地仓库上传到远程仓库

在GitHub查看团队文件夹，如图3-7所示。



图3-7 GitHub团队文件夹