Git 版本控制软件和 GitHub 平台使用基础教程

有限元法基础助教：倪锐晨

2021 年 2 月 8 日

自从高级语言 Fortran 被广泛使用，人类所开发的软件规模爆炸式地增长起来。随之而来的还有大型程序开发与维护中的许多严重问题，这些问题可能导致软件产品的寿命缩短，甚至夭折。这就是所谓的软件危机。人类在应对软件危机的过程中逐渐总结出一套方法，形成了软件工程这一学科。其中，一个非常重要的方法就是要对程序进行严格的版本控制。

其中，Git 是一个分布式管理系统。这意味着每个开发者的计算机上都有一个完整的仓库，包括完整的源代码以及开发历史。每个开发者需要在自己的仓库内完成自己的工作，形成稳定版本后再上传至远程服务器。这大大减少了开发者之间的冲突，并且对网络的依赖也不如 SVN 那样严重。

因此，在本课程的学习过程中，我们将结合使用 Git 软件和 GitHub 平台对大作业的代码进行版本控制管理，并将使用 Git 协同工作的协同程度计入大作业的评分考量，希望大家熟悉并掌握基于版本控制的代码协同开发。

1 Git 的安装

Linux 系统

打开终端，运行命令
sudo get-apt install git
即可安装 Git。

Windows 系统

前往 Git 的官方下载网址（`https://git-scm.com/downloads`），选择 Windows 的版本下载软件安装包。下载完毕后双击运行软件安装包，一切都选择默认选项即可。安装完成后，在任意文件夹下单击鼠标右键，如果出现如图 1 所示的“Git Bash Here”和“Git GUI Here”则说明安装成功。



图 1: Git 软件成功安装

2 GitHub 平台的注册和学生私有库权限的申请

GitHub 用户注册

本学期的大作业代码将统一保存在 GitHub 平台上，为此需要先前往 GitHub 的官网（`https://github.com/`）进行账号的注册：

1. 在如图 2 所示的界面中填写自己的用户名、邮箱和密码。邮箱一定要使用清华大学的邮箱以申请学生私有库的权限，用户名推荐使用自己的名字全拼（例如助教倪锐晨的用户名可以设置为 Ruichen-Ni）以便老师和助教识别大家的身份。

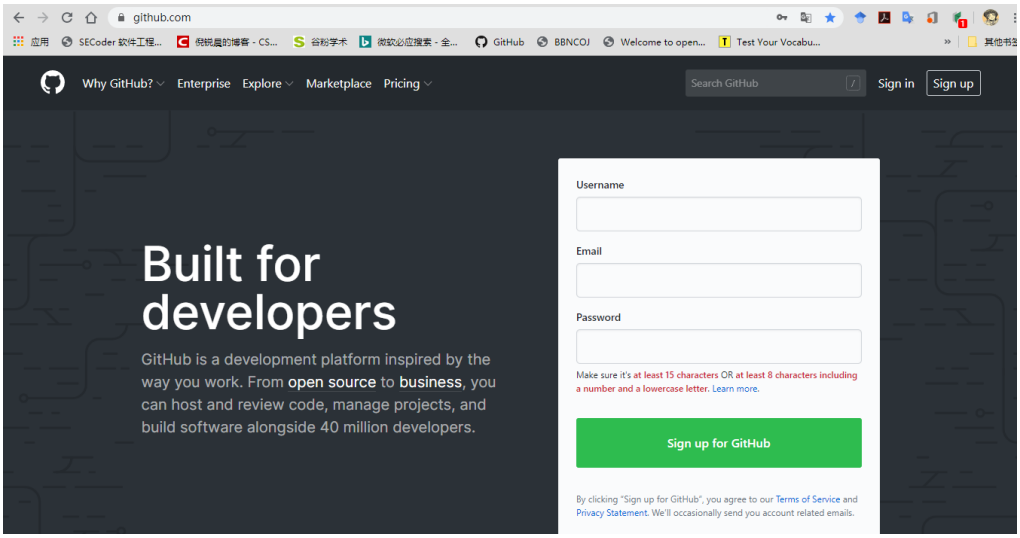


图 2: GitHub 平台注册界面

2. GitHub 会进入一个人机识别界面，正常操作即可，如图 3 所示

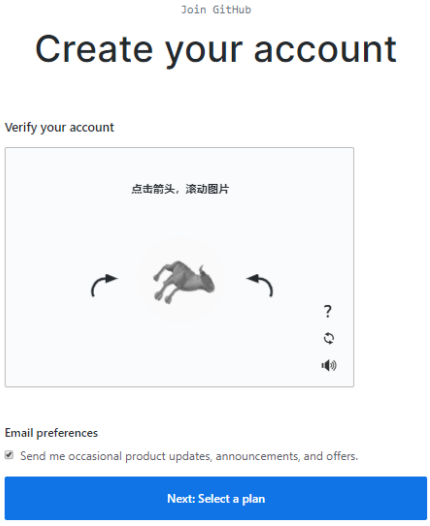


图 3: GitHub 平台人机识别

3. GitHub 会让大家选择一个账号计划（plan），我们选择左边的 Free Plan 即可，如图 4 所示

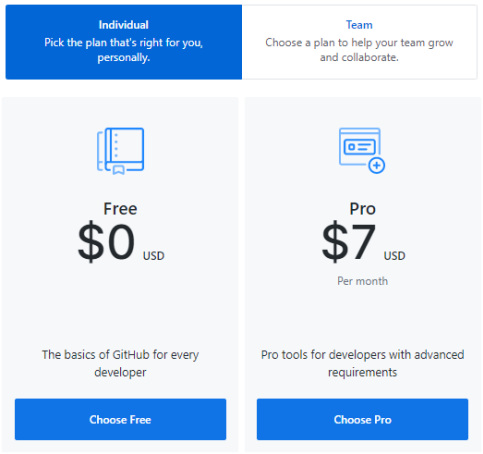


图 4: GitHub 平台账户计划选择

4. GitHub 平台会让大家填写一个用户问卷，大家可以按照自己的情况填写，也可以将网页拉到最下面选择跳过此步骤。大家打开邮箱后点击账户激活链接即可完成 GitHub 账号的申请，如图 5 所示

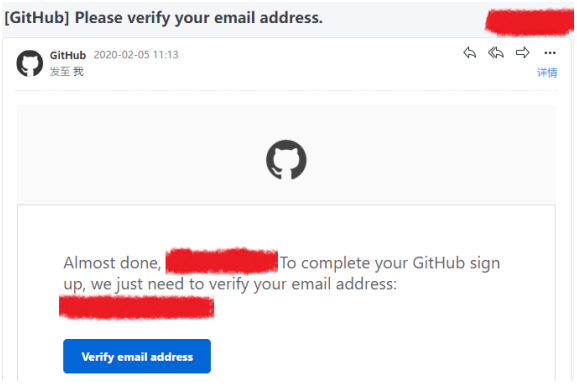


图 5: GitHub 平台账号激活邮件

5. 可以通过 GitHub 的登录界面（`https://github.com/login`）进行登录，如图 6 所示

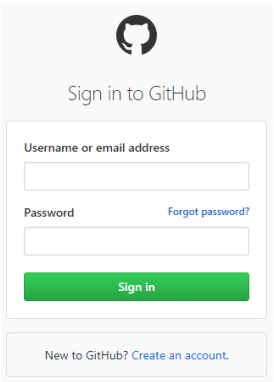


图 6: GitHub 平台登录界面

学生免费私有库权限申请

大作业会进行分组，为了保证各组独立完成该训练模块，我们要求大作业的代码仓库是私有的。同时每个组需要将老师和助教的账号也加入到自己的组内以便我们进行大作业进程的监督和最后的打分。GitHub 的 Free plan 对于 private 的代码仓库只允许三个合作者，不满足我们课程的需求。因此需要在 GitHub 平台上申请学生私有库权限，申请后 private 的代码仓库允许无限多的合作者（每个组的组长必须申请以邀请组内成员，其他同学可以按照自己的情况决定是否申请）。

1. 前往 GitHub 的教育申请界面（`https://education.github.com/students`），如图 7 所示，点击图中的“Get benefits for students”按钮。如果网页显示打不开，请通过 SSLVPN 挂回清华的 IP 地址（我家的网络 IP 地址就无法登陆该网站）。

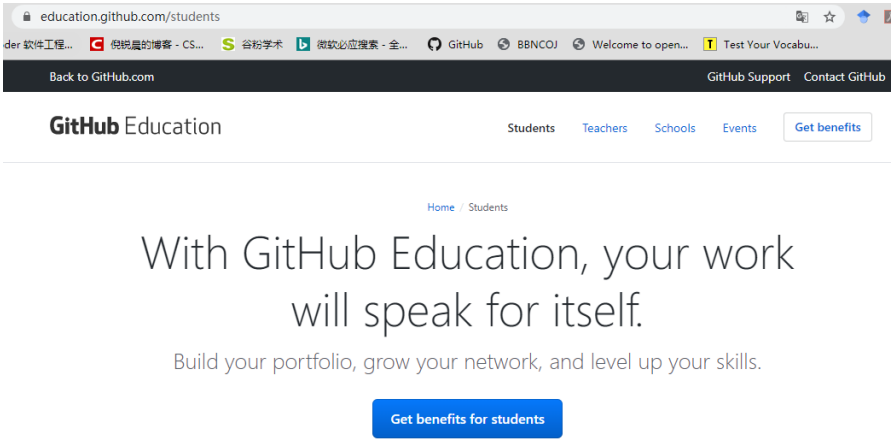


图 7: GitHub 平台学生权限申请

2. 进入到如图 8 所示的界面，如实填写个人信息。由于 GitHub 平台会检测申请发出的地理位置，不在学校附近的话申请会被拒绝。因此我们需要提供额外信息：将学生证照片和清华大学公众号的防疫通报截图合成在同一图片内上传（因为只允许上传一张照片），并在描述申请用途中写清楚“自己是清华大学学生，这次申请是为了进行有限元课程的学习，并且清华大学因为疫情决定在家通过远程 MOOC 在线上课”（由于是人工审核的，请大家自行翻译上面的这段话，意思相近即可，完全一致容易被误认为是机器申请而被拒绝）。然后点击“Submit your information”按钮即可

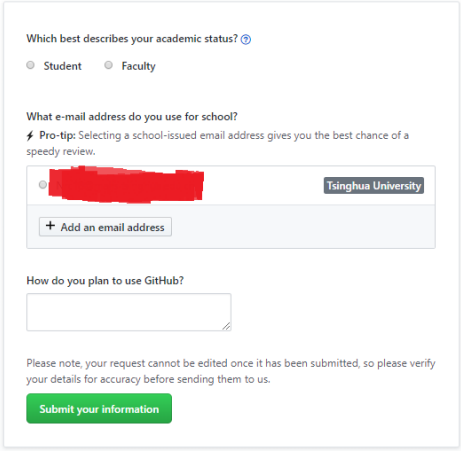


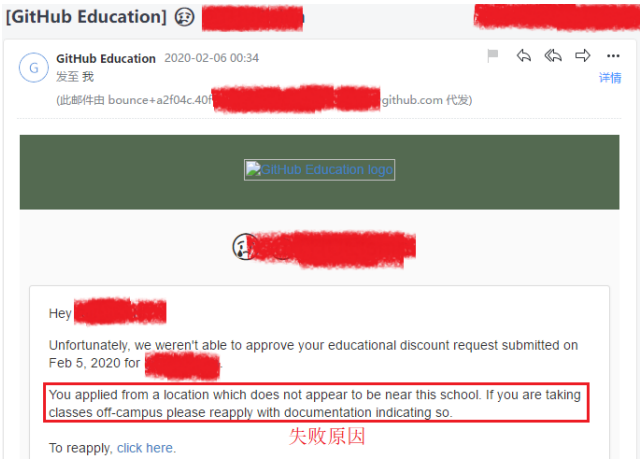
图 8: GitHub 平台学生权限申请问卷

3. 几个工作日内，GitHub 会向通过申请的账号邮箱发送如图 9 所示的申请成功邮件

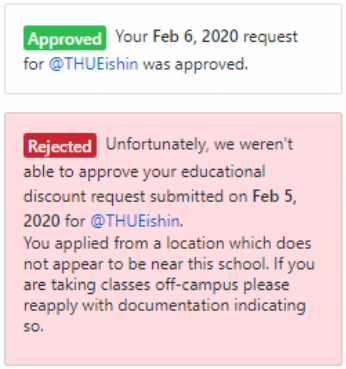


图 9: GitHub 平台学生权限申请成功邮件

4. 如果第一次申请被拒绝，GitHub 会发送申请失败邮件，查看失败原因补充相关材料后发起第二次申请



5. 如果一直没有收到邮件，可以重新进入申请界面，申请界面右侧会有你的申请记录，如下图所示



3 Git 和 GitHub 的 “Hello, World!”

任何语言的学习都会以一个 “Hello, World!” 程序进行入门学习，Git 也不例外。在这里我们会以一个简单的例子来讲解 GitHub 平台 SSH Key 的配置、代码仓库（repository）的创建、主分支（branch master）的合并保护权限的设置、本地仓库的创建、本地分支的创建、本地分支的提交、本地分支的合并、本地仓库向远端仓库推送、远端仓库的分支合并以及冲突处理。

SSH Key 的设置

如果不使用 SSH Key，每次向远端仓库提交时都会要求输入用户名和密码以确认用户的身份。配置了 SSH Key 之后，就免去了用户名和密码匹对的过程，在我们使用 git push 命令时会自动进行本地私钥和远端公钥的匹配，从而避免了输入用户名和密码的重复工作。

1. 在任意文件夹下点击鼠标右键打开 “Git Bash Here” 进入 Git 的命令行界面，如图 10 所示（我是在/e/temp 文件夹下进行的），输入如下命令

   ssh-keygen -t rsa -C ”your email”

   用你注册 GitHub 账号时使用的邮箱代替命令中的 your email，询问文件名称时推荐修改默认文件名（我使用的是./id_rsa_github）以便多账号的使用（没有多账户的同学也可以选择直接使用系统默认的文件名），询问私钥文件密码时可以选择不输入直接回车以方便以后推送，也可以选择设置以增加安全性。

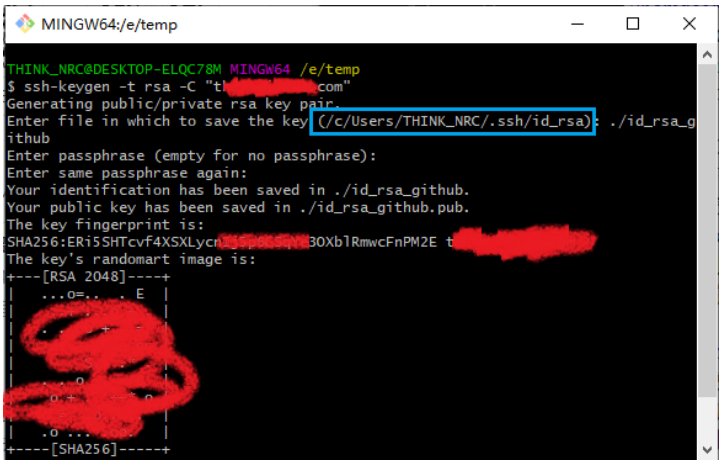


图 10: SSH Key 生成

将生成的私钥文件和公钥文件拷贝到 Git 可以访问的文件夹下，即询问文件名时系统提供的绝对路径（图中蓝框位置处，本例中为/c/Users/THINK_NRC/.ssh/）。如果没有该文件夹就创建一个。

2. 使用注册的账号登录 GitHub 平台，点击界面右上方的头像下拉菜单进入“settings”选项卡，在 settings 选项卡选择“SSH and GPG keys”选项，如图 11 所示。点击图中的“New SSH key”按钮。

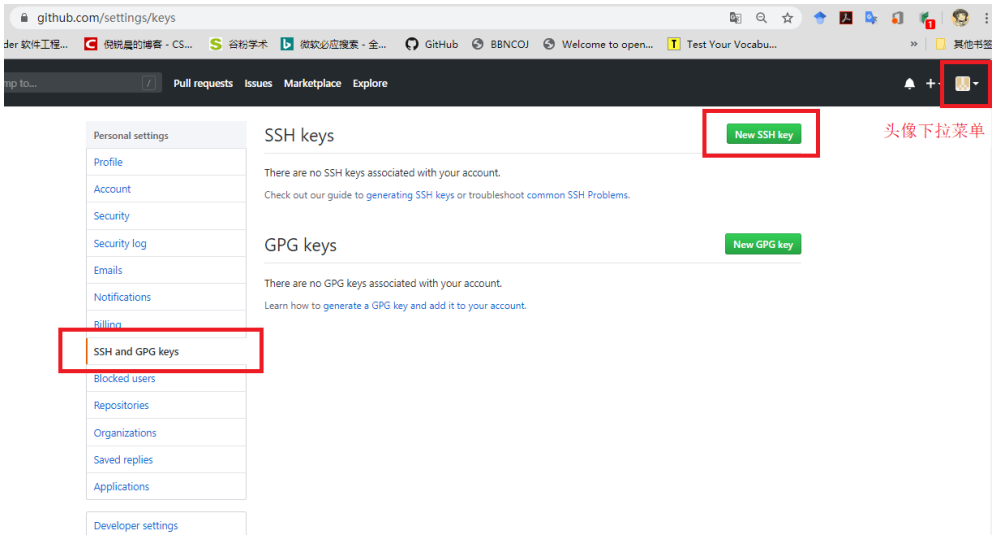


图 11: SSH Key 公钥设置

3. 打开公钥（.pub）文件，复制所有内容粘贴到图 12 所示的“Key”框内，在 Title 框内随便取一个易辨别的名字（例如 Windows 系统产生的公钥就命名为 Windows），点击“Add SSH key”按钮即可。

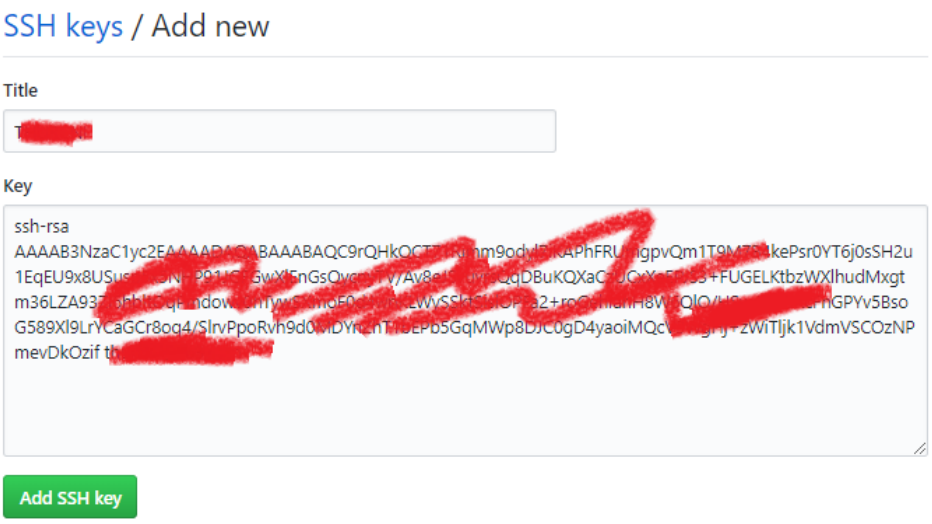


图 12: SSH Key 公钥设置 2

GitHub 私有仓库的创建和配置

1. 点击右上方头像左边的加号“+”的下拉菜单，选择其中的“New Repository”选项，出现如图 13 所示的新建仓库页面。“Repository name”是必须的，请选择一个能够反映仓库作用的名字。“Description”也尽可能描述清楚仓库的作用。仓库属性选择“Private”。勾选“README”文件创建的选项，该 README 文件可用于记录每个组员向大作业代码中添加的功能。根据大作业代码的语言选择.gitignore 文件以排除相关工程文件的提交。全部设置完毕后点击“Create repository”按钮。

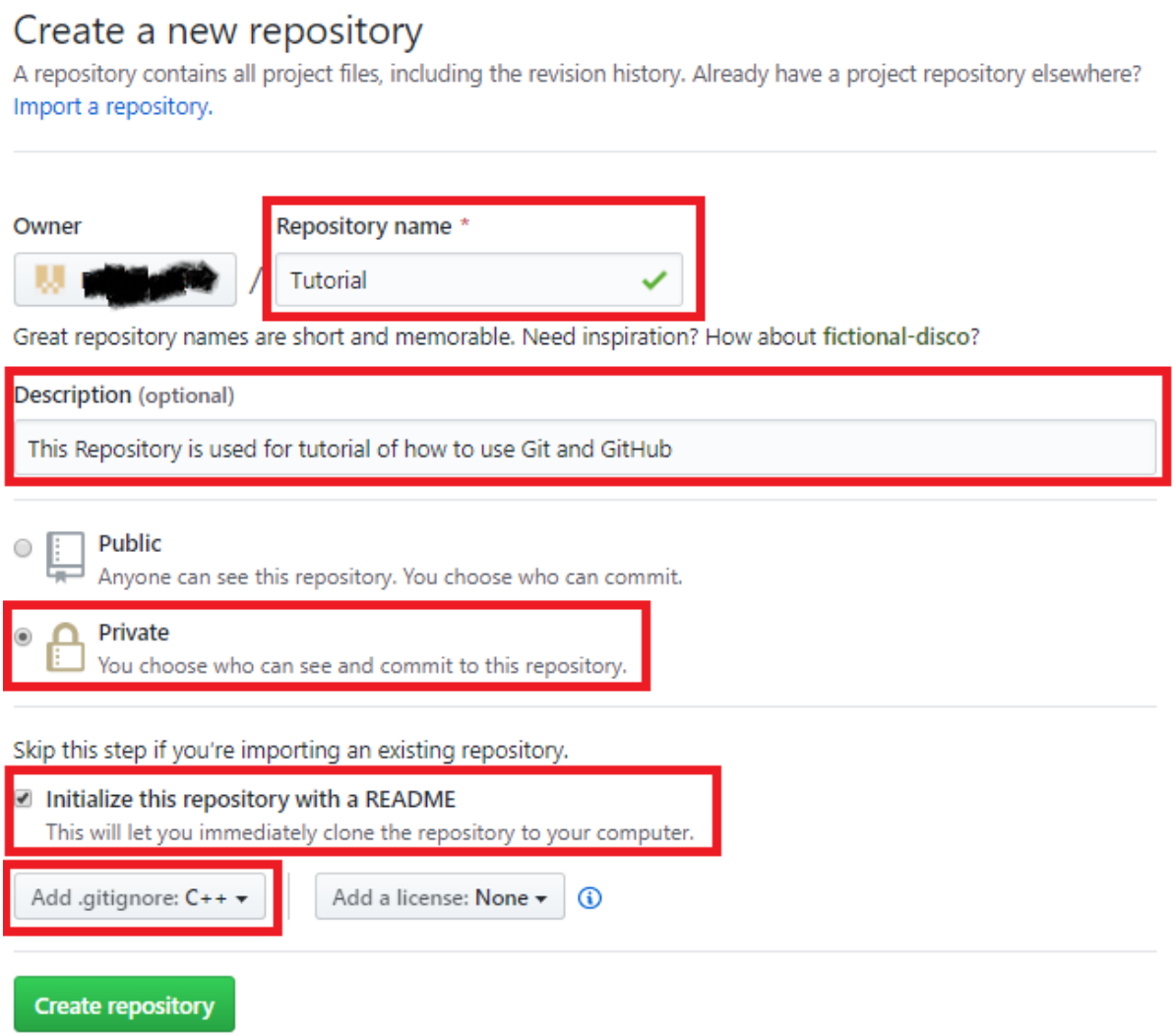


图 13: GitHub 私有仓库的创建

2. 进入新建仓库“settings”选项卡下的“Collaborators”选项，如图 14 所示。在下方的搜索框内搜索组员的 GitHub 用户名，搜到后可以点击“Add collaborator”按钮向组员发出加入仓库的邀请，组员会收到邀请邮件，选择接受邀请就可以对仓库代码进行协同开发。

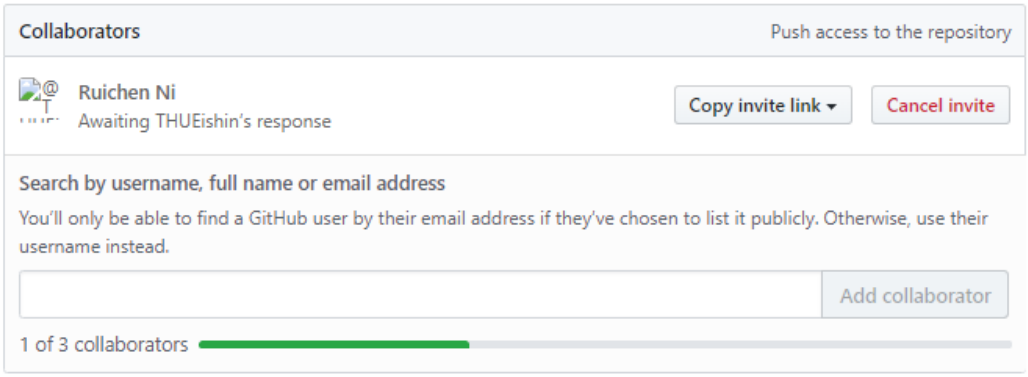


图 14: GitHub 私有仓库成员邀请

3. 进入新建仓库“settings”选项卡下的“Branches”选项，如图 15 所示。点击图中的“Add rule”按钮。（私有仓库必须通过学生权限申请后才能添加分支权限设置。）

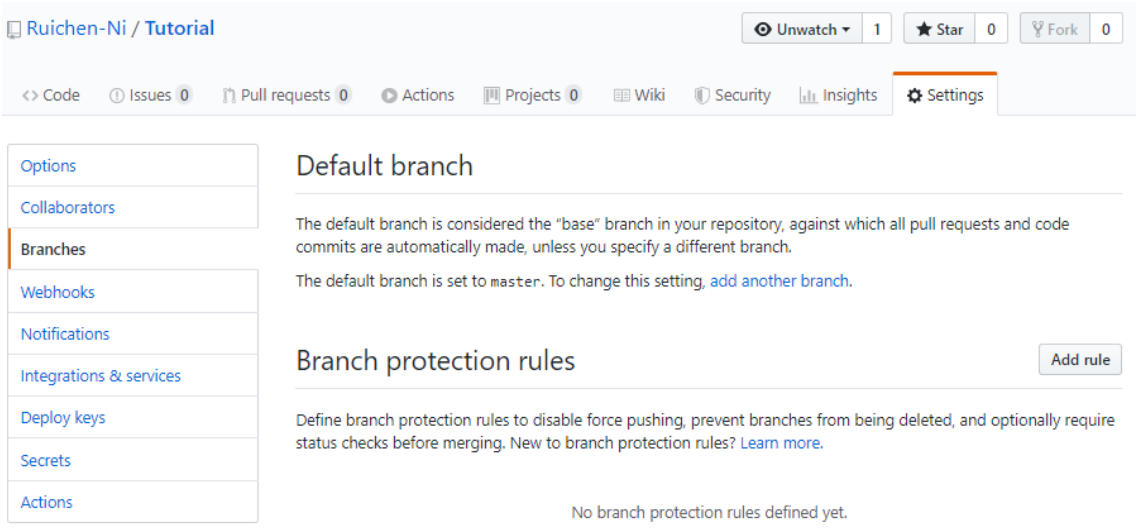


图 15: GitHub 仓库的分支权限设置

4. 在分支权限设置页面请按图 16 所示设置。其中“Required approving reviews”推荐设置为 2 或者 3，即至少组长和组内另外一位成员查看代码后认为没有问题才允许向 master 分支合并以增强 master 分支的稳定性。另外，“Rules applied to everyone including administration”中的两个选项绝对不能勾选以避免软件危机的出现。

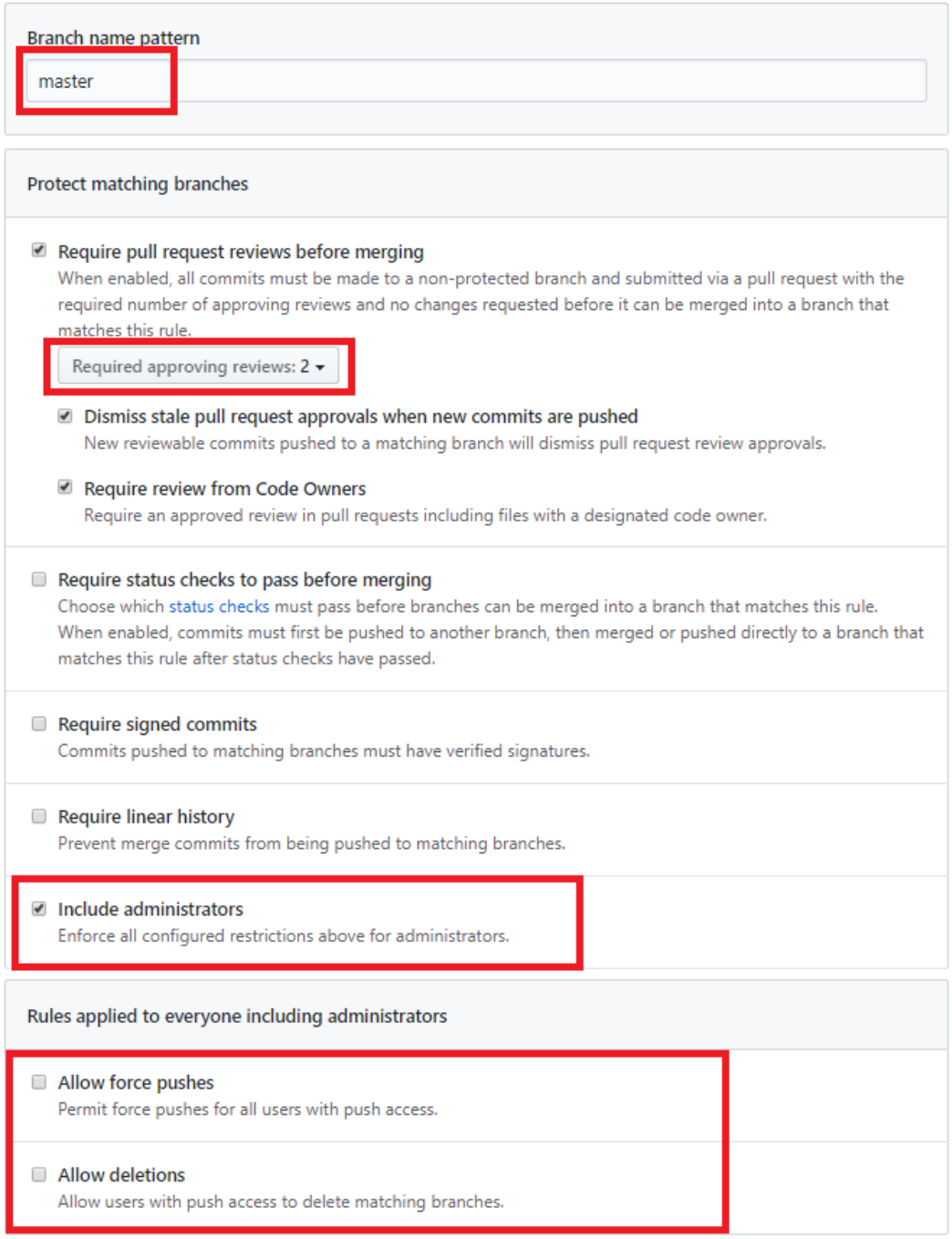


图 16: GitHub 仓库分支权限设置

一个简单的例子

假设现在来了一个任务：需要从零开发出软件 Tutorial 的 1.0 版本，该版本包含两个功能 feature1 和 feature2，这两个功能将分别由同学 A 和同学 B 完成（在本例中我会用两个账号来演示这个例子）。

先来看 A 同学完成 feature1 的开发：

1. 新建一个空的文件夹（本例中为/e/temp/），进入文件夹内单击鼠标右键选择 “Git Bash Here” 打开 Git 的命令行窗口。如图 17 所示，在命令行窗口中输入如下命令初始化本地仓库

   ```
   git init
   ```

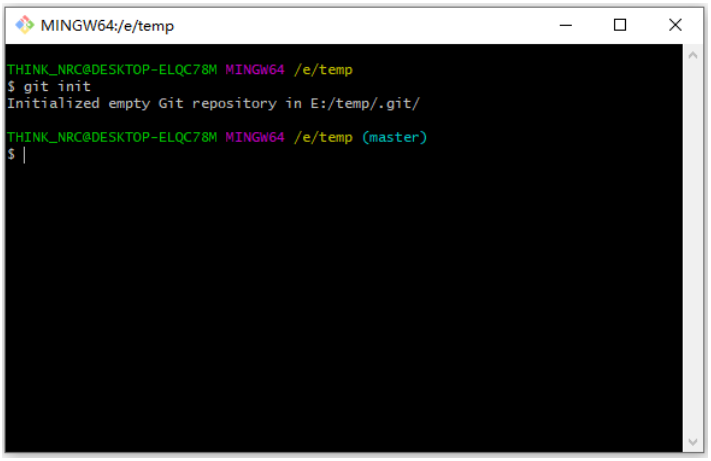


图 17: 本地仓库初始化

2. 为项目配置用户名和邮箱

   单独为本项目配置：

   ```
   git config user.email "GitHub注册邮箱"
   git config user.name "GitHub用户名"
   ```

   为电脑内所有项目配置用户名和邮箱

   ```
   git config --global user.email "GitHub注册邮箱"
   git config --global user.name "GitHub用户名"
   ```

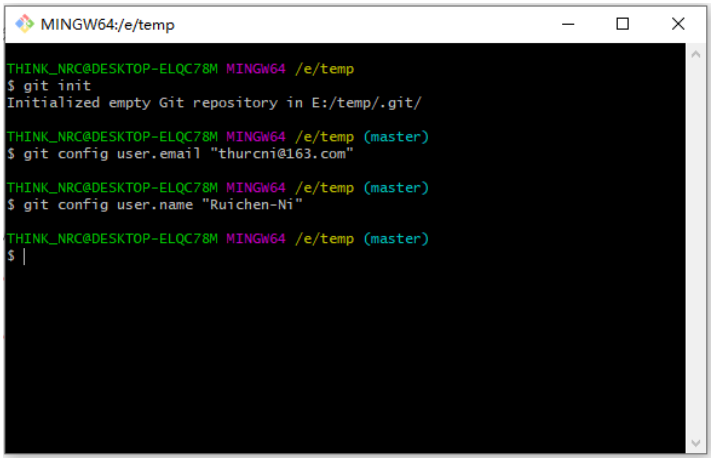


图 18: 配置用户名和邮箱

3. 可以通过如下命令来查看本项目的配置，可以看到我们成功设置了用户名和邮箱

```
git config --local --list
```

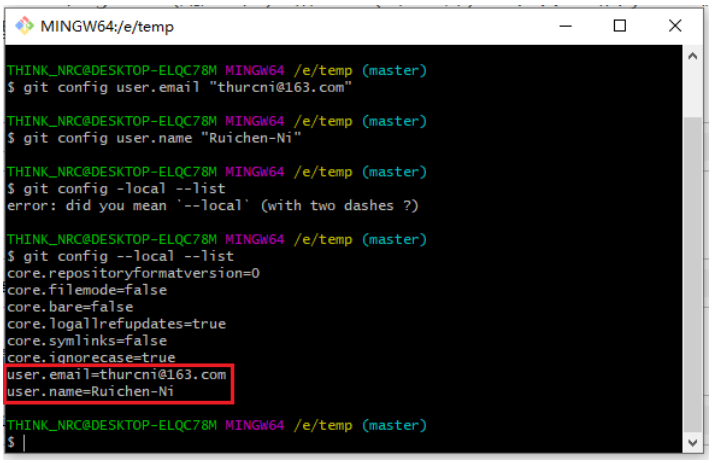


图 19: 查看项目配置

4. 点击图 20 中的 “Clone or download” 按钮，并点击 “Use SSH” 切换成如图所示的选项卡，复制其中的 SSH 仓库地址（本例中的仓库地址为 git@github.com:Ruichen-Ni/Tutorial.git）

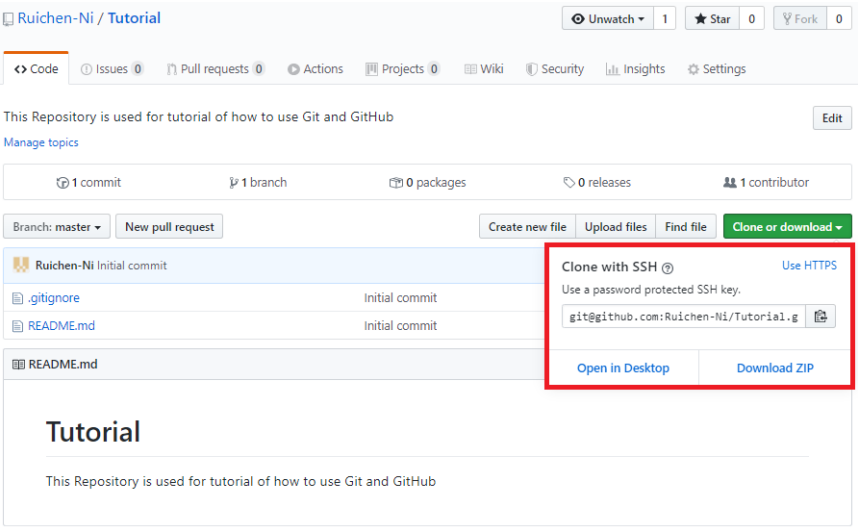


图 20: GitHub 仓库的 SSH 地址

5. 在 Git 的命令行窗口输入如下命令将远端仓库克隆到本地

   git clone 仓库地址

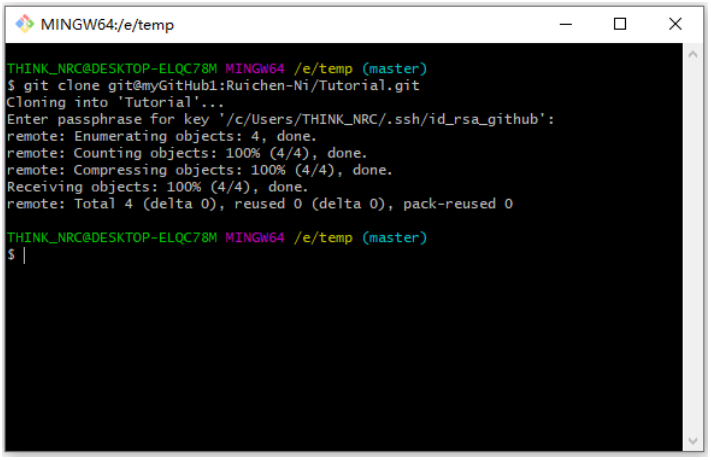


图 21: GitHub 仓库的克隆

细心的同学会发现我输入的是 git@myGitHub1:Ruichen-Ni/Tutorial.git 而不是 git@github.com:Ruichen-Ni/Tutorial.git。这是因为我有多个版本控制的账号，需要通过一个如图 22 的配置文件（该文件与私钥和公钥文件在同一文件夹下，并且该文件名为 config，没有后缀名）来定位到底使用哪一个私钥文件。

```
#github
Host myGitHub1
HostName github.com
IdentityFile ~/.ssh/id_rsa_github

#github2
Host myGitHub2
HostName github.com
IdentityFile ~/.ssh/id_rsa_github_tsinghua
```

图 22: 多个账户 SSH Key 的配置文件

6. 为了开发 Tutorial 软件的 1.0 版本，A 同学需要新建一个开发分支 dev。进入 Tutorial 文件夹（可以直接在命令行窗口输入命令cd Tutorial，也可以在文件管理器中进入 Tutorial 文件夹单击鼠标右键重新运行 Git 的命令行窗口），并输入如下命令（依次表示创建 dev 分支，切换到 dev 分支和查看分支状态）。前面带 * 号的分支表示目前所在的分支。

```
git branch dev
git checkout dev
git branch
```

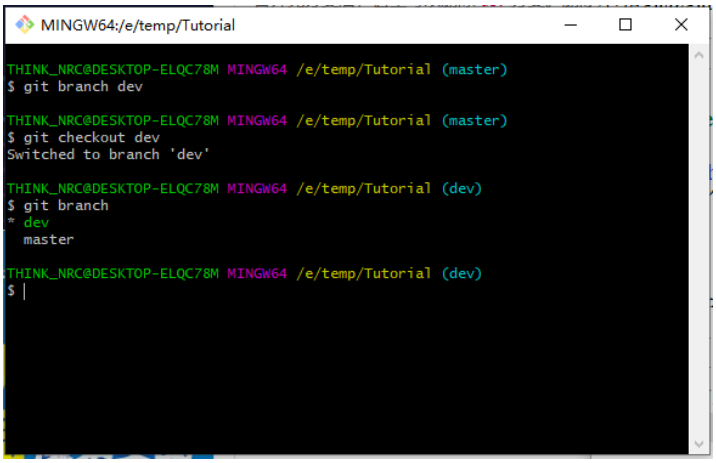


图 23: 创建分支、切换分支和查看分支状态

7. A 同学通过如下的命令在 GitHub 远端仓库同时创建了一个对应的 dev 分支用于软件开发

```
git push origin dev:dev
```

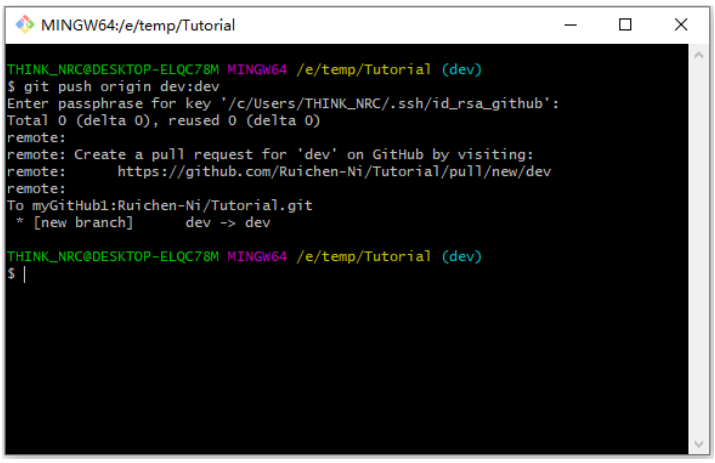


图 24: 创建 GitHub 远端仓库的开发分支 dev

8. A 同学现在开始着手开发功能 feature1，他需要从 dev 分支上创建一个分支 fea1 用于功能的开发，类似上面创建 dev 分支的操作，这里给出相应的命令但不再赘述。

```
git branch fea1
git checkout fea1
git push origin fea1:fea1
```

9. A 同学添加了一个文件名为 feature1.txt 的文件代表功能 feature1 的开发完成，可以通过如下命令进行工作区状态的检测和向本地仓库提交修改

```
touch feature1.txt
git status
git add -A
git commit -m "Finish Feature 1"
```

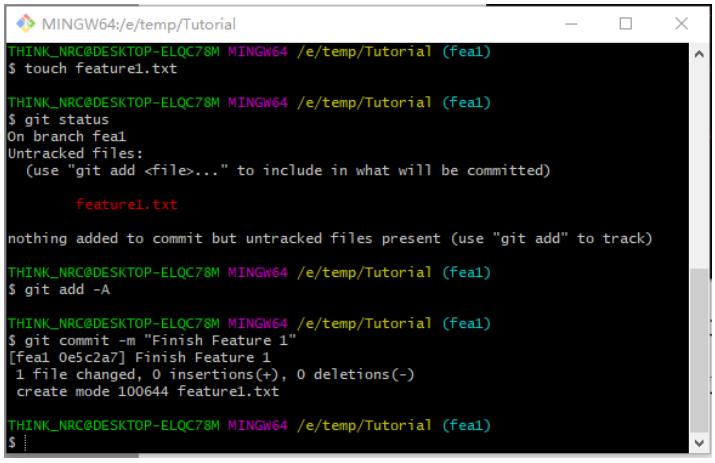


图 25: 文件修改、工作区状态检测、将修改提交至缓存区和将修改提交至本地仓库

10. A 同学将完成的功能 feature1 的相关代码通过如下的代码提交到 GitHub 远端库。登录 A 同学的 GitHub 账号会发现系统提醒我们有 fea1 分支上存在新的提交并提示我们进行比较合并。当 A 同学检查觉得功能 feature1 的代码正确无误就可以点击“Compare & pull request”按钮发起 Pull Request。

```
git push origin fea1:fea1
```

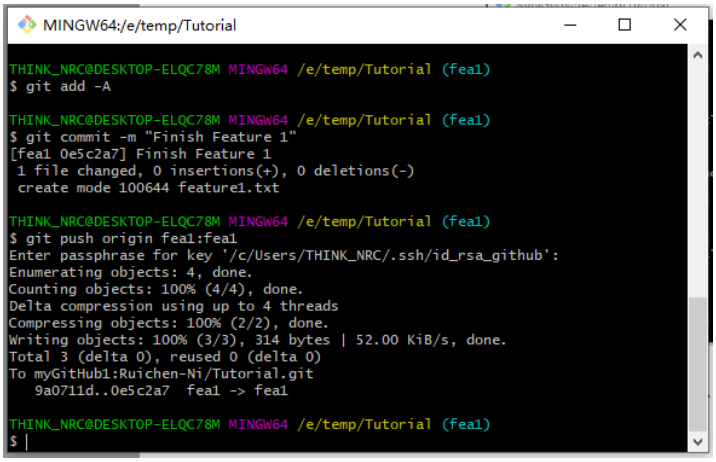


图 26: 将修改提交至远端仓库

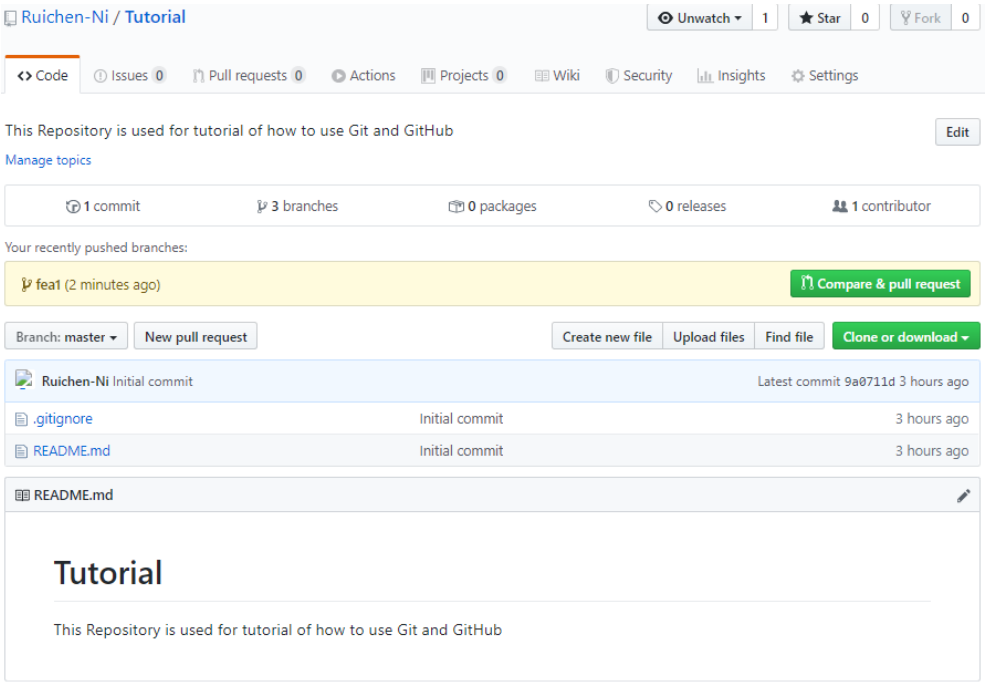


图 27: 发起 Pull Request

11. 如图 28 所示，A 同学发起分支合并请求，在 Tutorial 软件的 1.0 版本开发过程中，我们需要将所有功能提交到 dev 开发分支上。等所有功能都开发完毕，再将 dev 分支合并到 master 分支上。（为了更有效地避免软件危机，可以对 dev 分支也设置和 master 分支一样的合并权限）填写完所有信息后点击“Create pull request”

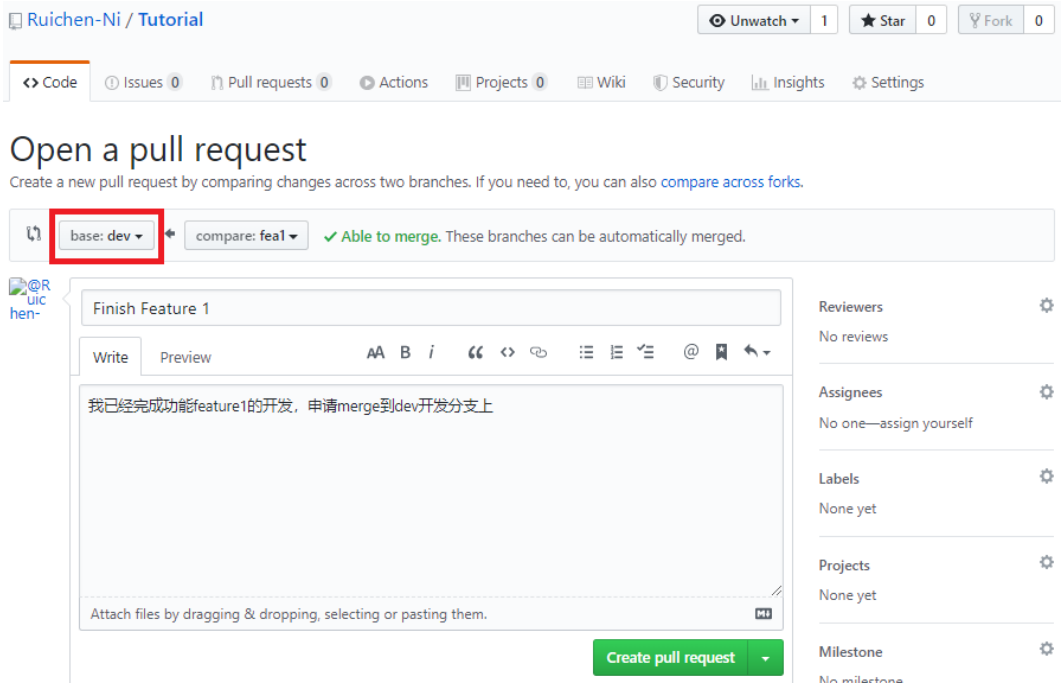


图 28: 发起 Pull Request 页面

12. 如图 29 所示，A 同学发起 pull request 后会进入一个类似投票的环节，如果有组员觉得添加的代码存在问题就可以在第三个红框上方的 comment 里写一段评论并关闭这次的 pull request。A 同学可以和那个组员进行在线 comment argue，如果那个组员觉得 A 说的是对的，可以再重新 reopen 这个 pull request，就像第一个红框操作的那样。如果在 review 界面（向设置了保护权限的分支合并时才会有进入该界面的链接，本例中的 dev 分支并没有设置保护权限。推荐给 dev 分支也设置权限保护）觉得添加的代码没问题就选择“Approve”，当认同的人数达到 branch rule 里设定的值时，第二个红框内的“merge pull request”按钮就会变得可以进行点击。

    所有在 GitHub 远端的分支合并操作都会被记录下来，包括所有的 comment 信息。在大作业评分中我们会根据这个来判断大家的小组分工合作情况，所以请好好地撰写 comment。

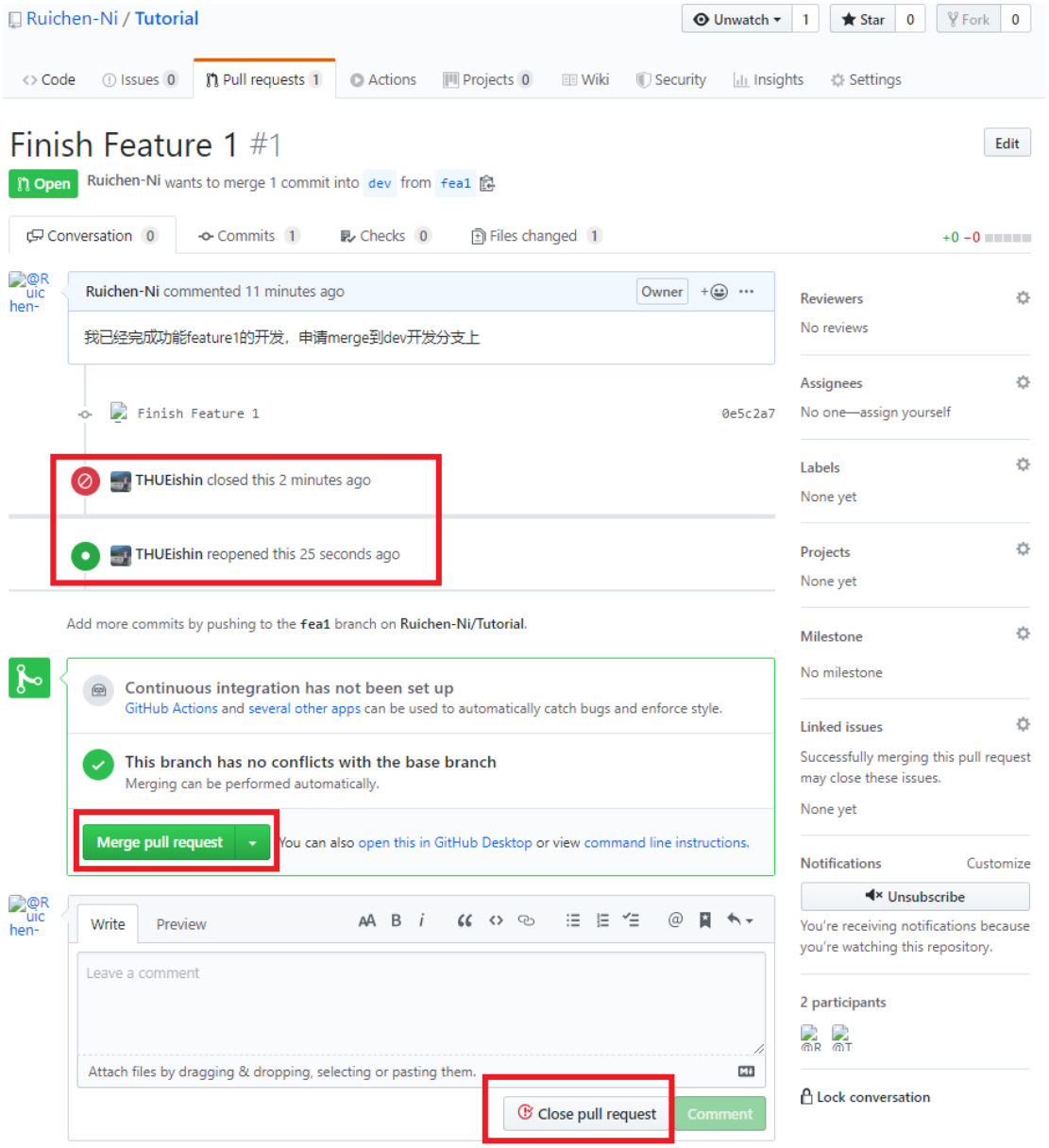


图 29: 组员决定是否进行 merge

13. 如图 30 所示，当 merge 成功后会出现该界面，由于 feature1 已经完成开发了，我们可以点击红框内的按钮删除 fea1 分支。

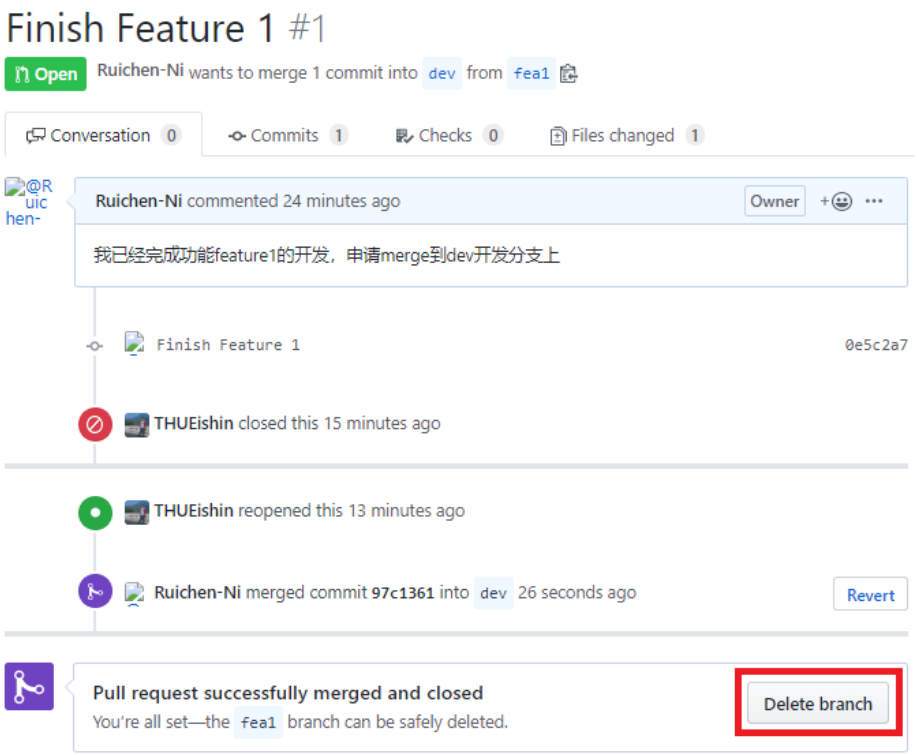


图 30: 成功 merge 并删除 fea1 分支

14. A 同学在 GitHub 远端仓库 merge 的时候没有遇到冲突，所以在本地仓库也可以将 fea1 分支简单地合并到 dev 分支中，保持本地与远端的同步。相关命令如下

```
git checkout dev        #切换回dev分支
git merge fea1          #将fea1分支merge到dev分支
git branch -d fea1      #删除fea1分支
git fetch origin dev    #从GitHub远端仓库获取dev的更新
git merge origin/dev    #将fetch到的远端分支克隆与本地的dev分支合并
```

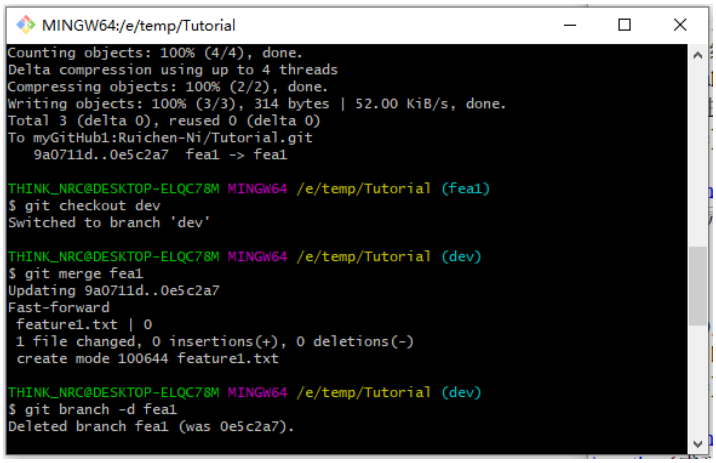


图 31: 切换分支、分支合并和删除分支

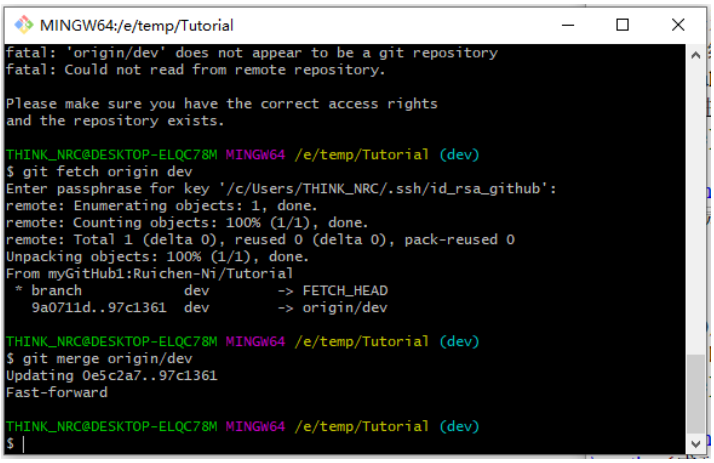


图 32: 保持本地分支和远端仓库分支同步

A 同学已经完成了自己的工作，但是 B 同学的工作由于比较难还没有完成，A 同学就开始帮助 B 同学检查 B 同学前面的代码是否有错误。而 B 同学就继续自己工作的代码撰写。我们希望通过这个案例来讲解分支合并冲突产生后的处理。

1. 在 A 同学完成远端仓库的开发分支 dev 的创建后，B 同学将远端仓库克隆到了本地准备开始 feature2 的开发。但是此时 B 同学本地只有 master 分支，并没有远端仓库的 dev 分支的克隆，因此 B 同学需要通过以下命令用远端仓库的 dev 分支创建本地 dev 分支。然后类似地创建 fea2 分支进行 feature2 的开发

```
git checkout -b dev origin/dev
git checkout -b fea2
touch feature2.txt
git push origin fea2:fea2
```

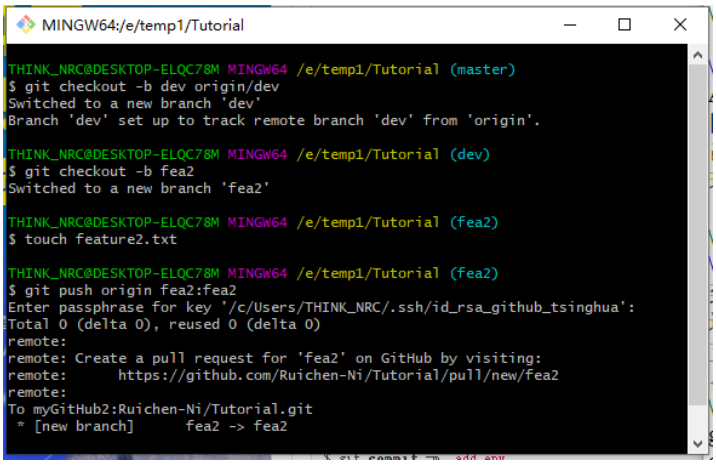


图 33: 本地 dev 分支创建、fea2 分支创建和远端 fea2 分支创建

2. B 同学的 feature2 代码开发到一半，如图 34 所示。并通过如下命令将开发到一半的代码推送到 GitHub 远端仓库的 fea2 分支中进行代码保存以防代码丢失。

```
git status
git add -A
git commit -m "Feature2 : Half"
git push origin fea2:fea2
```

```
正确代码段A
aaaaa
正确代码段B
bbbbb
错误代码段C
asdfikajnfvsjasfjmakf
正确代码段D
ddddd
```

图 34: B 同学开发到一半的 feature2 代码

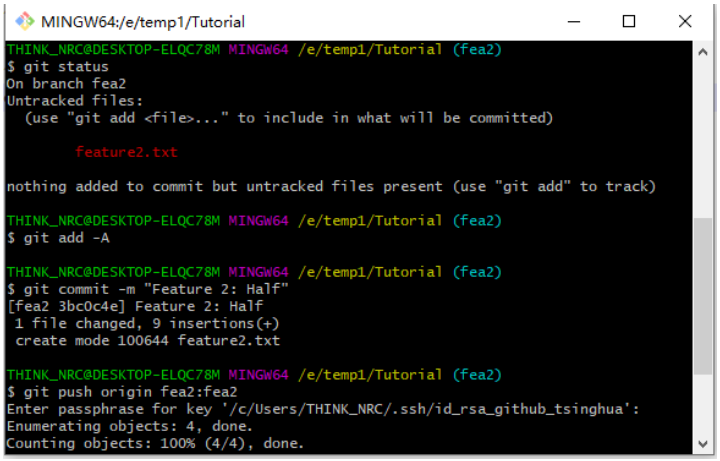


图 35: B 同学将开发到一半的 feature2 代码推送到远端仓库进行保存

3. A 同学在此时加入帮助 B 同学 debug 前面的代码段，通过以下代码获取远端仓库的 fea2 分支，并从 fea2 分支上创建本地 debug 分支。

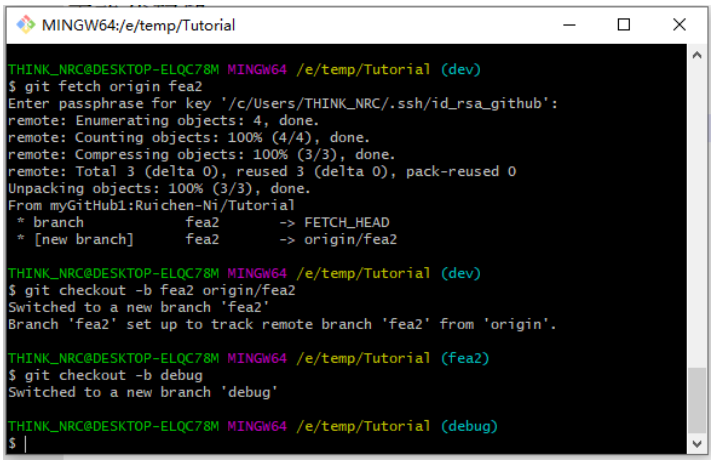


图 36: 获取远端仓库的 fea2 分支

```
git fetch origin fea2
git checkout -b fea2 origin/fea2
git checkout -b debug
```

4. A 同学在 debug 分支中找到了错误代码段 C，并将其改正如图 37 所示。并切换回 fea2 分支，将 debug 分支上的修改 merge 到 fea2 分支上，然后将本地 fea2 分支推送到 GitHub 远端仓库的 fea2 分支。

```
git status
git add -A
git commit -m "Debug: feature 2"
git checkout fea2
git merge debug
git branch -d debug
git pull origin fea2
git push fea2:fea2
```

```
正确代码段A
aaaaa
正确代码段B
bbbbb
正确代码段C
ccccc
正确代码段D
ddddd
```

图 37: 修改代码

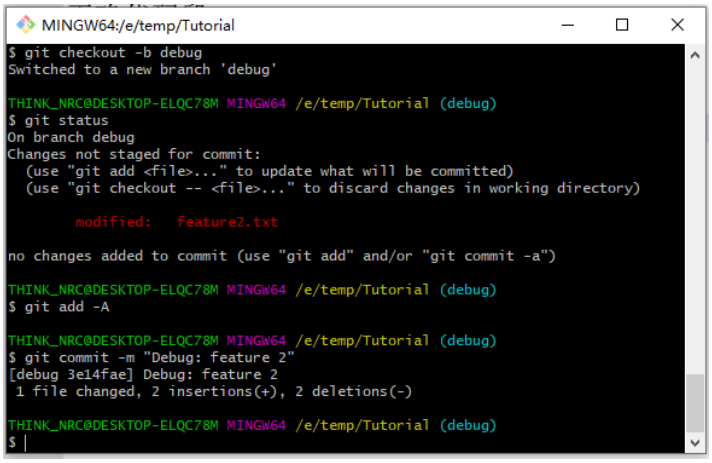


图 38: 修改 bug 并提交 debug 分支

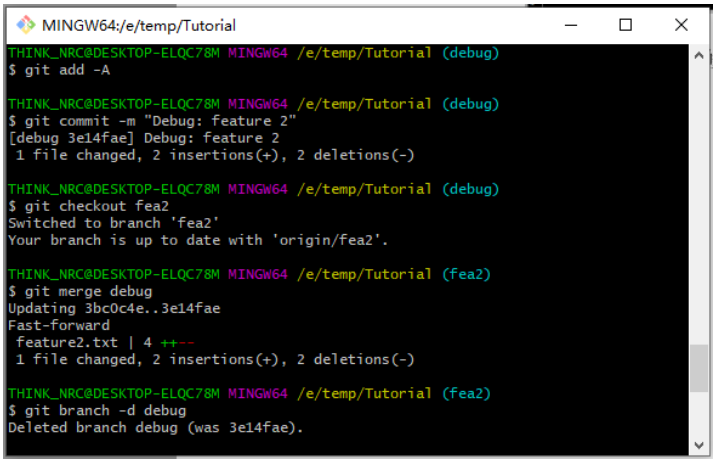


图 39: 将 debug 分支 merge 到 fea2 分支中

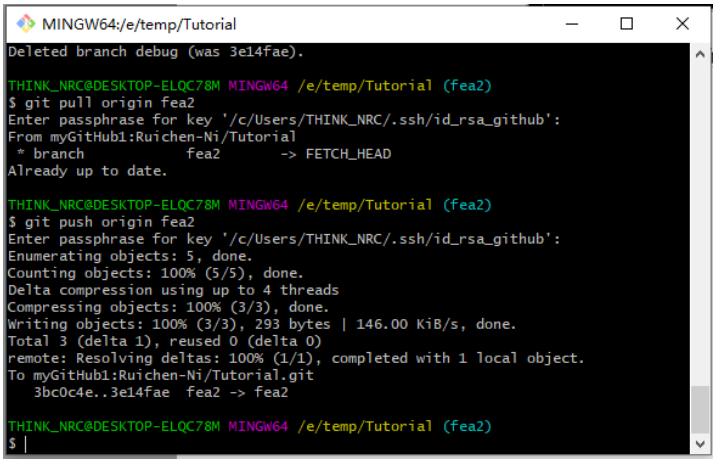


图 40: 将本地 fea2 分支推送到远端仓库的 fea2 分支中

5. 此时 B 同学发现他写的代码段 B 和代码段 C 对实现 feature2 没有作用，就将其删除，最后 feature2 的代码如图 41 所示

```
正确代码段A
aaaaa
正确代码段D
ddddd
正确代码段E
eeeee
正确代码段F
fffff
```

图 41: feature2 的最终代码

6. B 同学通过如下命令将 feature2 的代码提交至本地 fea2 分支

```
git status
git add -A
git commit -m "Finish Feature 2"
```

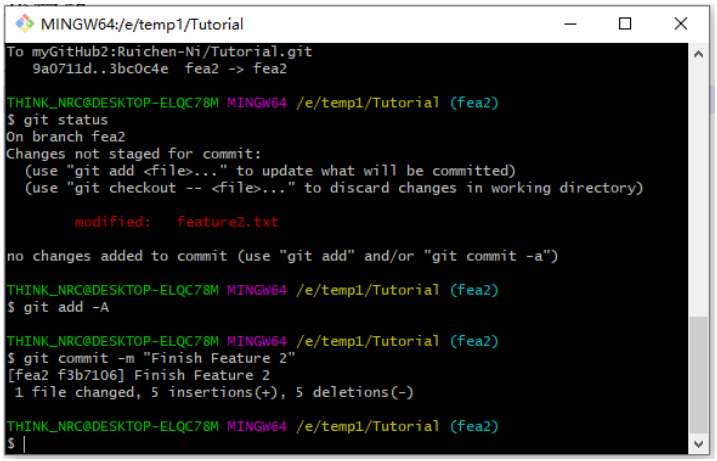


图 42: 将代码修改提交至本地 fea2 分支

7. B 同学的版本控制习惯较差，直接进行了 git push 命令，由于有其他用户在 B 同学之前向远端 fea2 分支进行过推送，因此被系统拒绝直接进行 push。

```
git push origin fea2
```

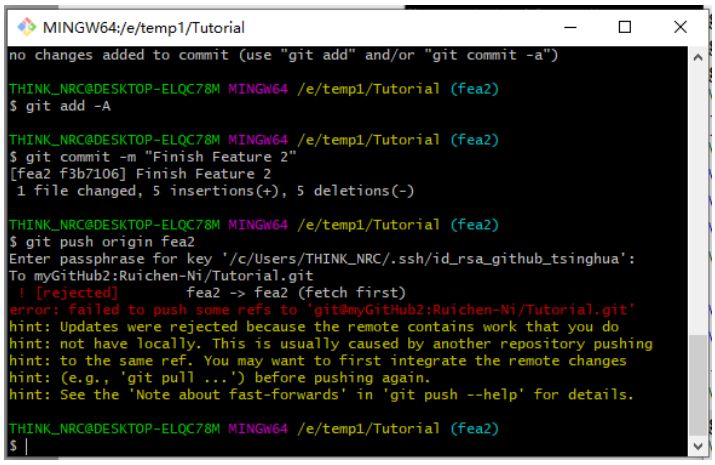


图 43: 向远端仓库的推送被拒绝

8. 因此 B 同学进行了 git pull 的操作，系统提醒存在内容冲突如图 44 所示。打开代码文件 feature.txt 可以看到相应的冲突

```
git pull origin fea2
```

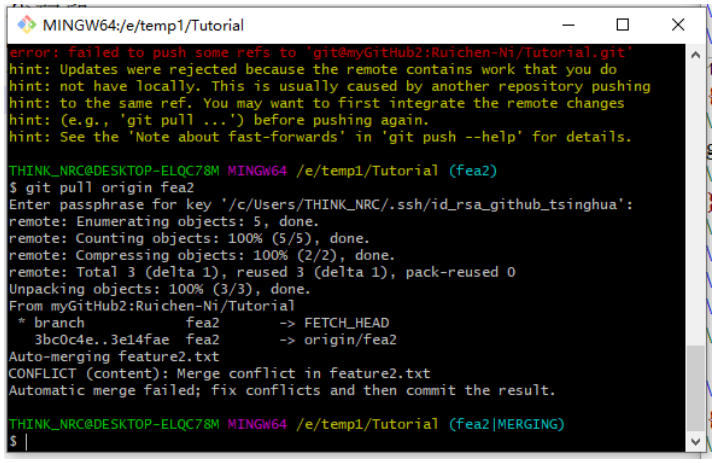


图 44: git pull 后系统提醒存在内容冲突

```
正确代码段A
aaaaa
<<<<<<< HEAD
=======
正确代码段B
bbbbb
正确代码段C
ccccc
>>>>>>> 3e14faea3f2f0886e448aeec6c4d832a557a0e49
正确代码段D
ddddd
正确代码段E
eeeee
正确代码段F
fffff
```

图 45: 出现冲突的内容

9. 手动将 feature.txt 的文件修改为如图 41 所示，就可以通过如下命令将本地 fea2 分支推送至远端仓库的 fea2 分支。

```
git status
git add -A
git commit -m "Fixed Conflict"
git push origin fea2
```

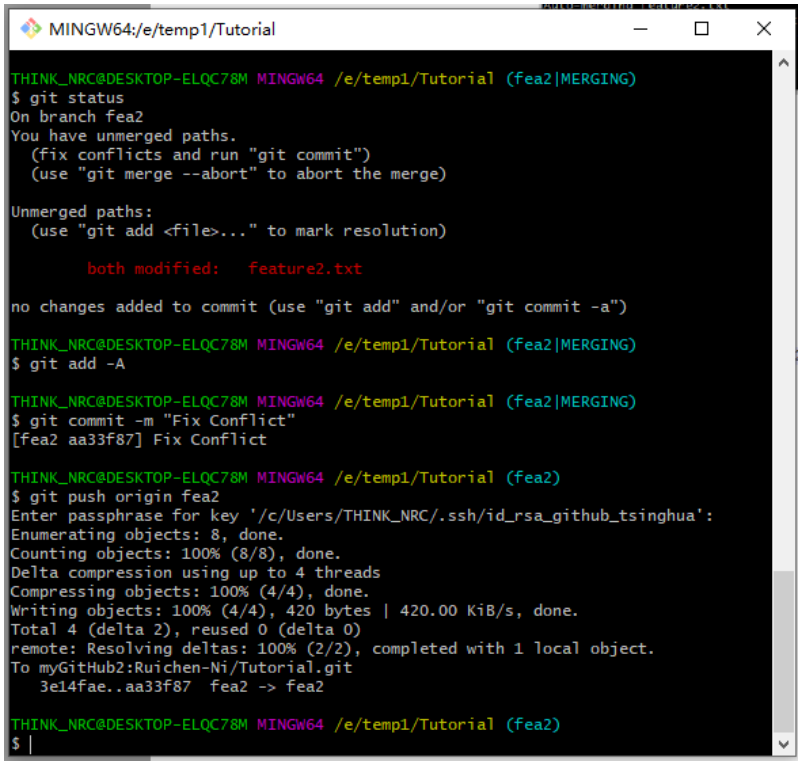


图 46: 解决冲突后重新推送本地的 fea2 分支

10. 类似 A 同学向 master 分支合并 fea1 分支一样，将 fea2 分支合并到 dev 分支中。由于所有功能都开发完毕了，所以我们可以将 dev 分支合并到 master 分支中完成 1.0 版本的 Tutorial 软件。如果再有任务要求添加新的功能更新软件的 2.0 版本，可以按这种方式进行软件迭代。

4 Git 桌面版图形界面

在这里给大家推荐两种界面简洁好用的桌面版图形界面：Sourcetree 和 GitKraken。由于 Sourcetree 只能在 Windows 和 Mac OS 系统下进行安装而助教的 PC 是 Ubunutu 系统，所以这里给大家重点介绍跨平台的 GitKraken 软件。

软件安装

大家前往官网`https://www.gitkraken.com/` 即可下载安装。

1. 第一次运行软件会进入如图 47 所示的界面进行用户注册，我们选择第一项 “Sign in with GitHub”，然后会自动跳转到 GitHub 网页进行授权操作。按照提示正常操作即可。

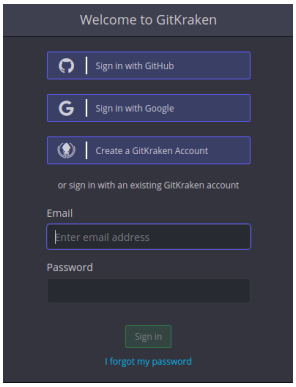


图 47: GitKraken 软件的用户注册

2. 授权完成后会进入到如图 48 所示的界面填写个人的相关信息，大家如实填写即可。

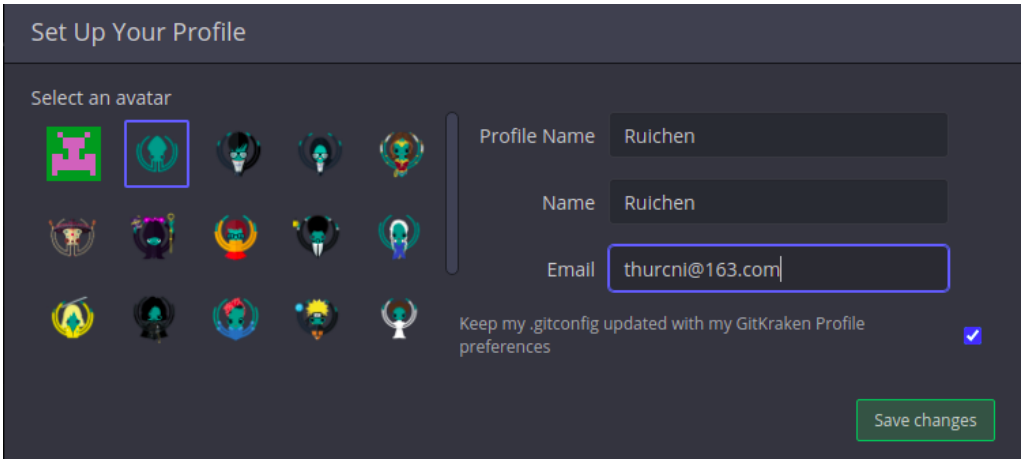


图 48: GitKraken 软件的用户信息

3. 填写完上述信息以及以后打开 GitKraken 软件就会进入如图 49 所示的欢迎界面。

4. 我们点击图 49 中的 “Clone a repo” 的按钮就会进入到如图 50 所示的仓库克隆界面。由于我们在第一步的时候已经使用 GitHub 平台对 GitKraken 软件进行了授权，因此当我们选择列表中的 “GitHub.com” 时可以直接选择账号中的远端仓库，非常方便。当然也可以使用 “Clone with URL”，将远端仓库的 URL 地址复制进去进行克隆。

5. 成功克隆之后会在软件上方弹出如图 51 所示的弹窗，我们直接点击就能进入仓库的管理界面如图 52 所示。

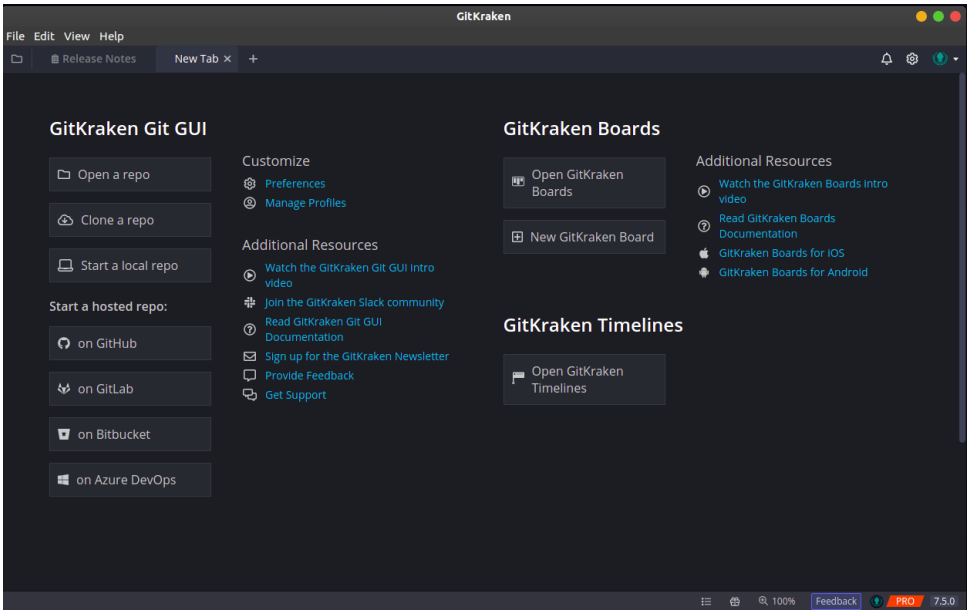


图 49: GitKraken 软件的欢迎界面

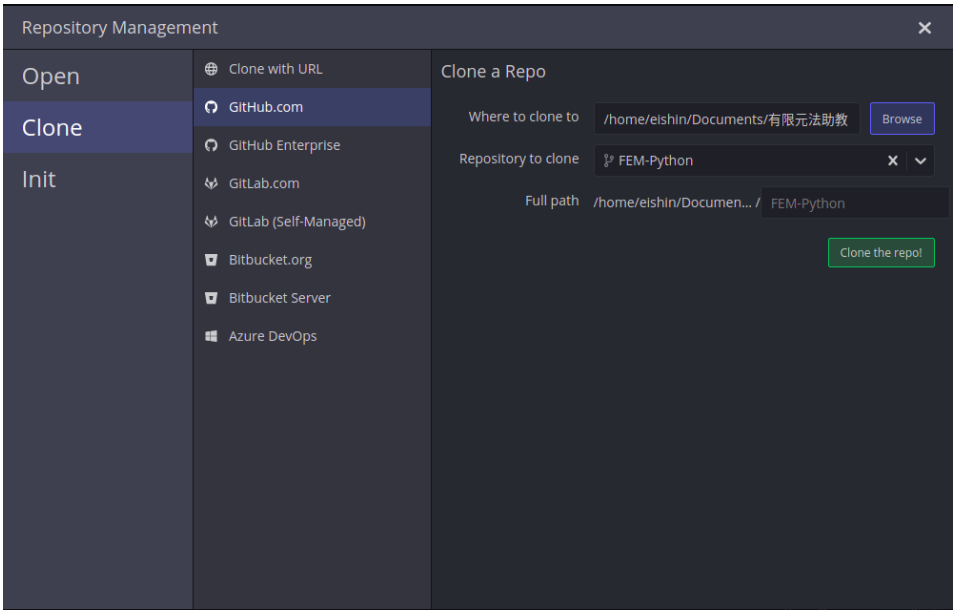


图 50: GitKraken 软件的仓库克隆界面

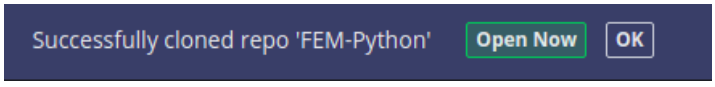


图 51: GitKraken 软件的仓库克隆界面

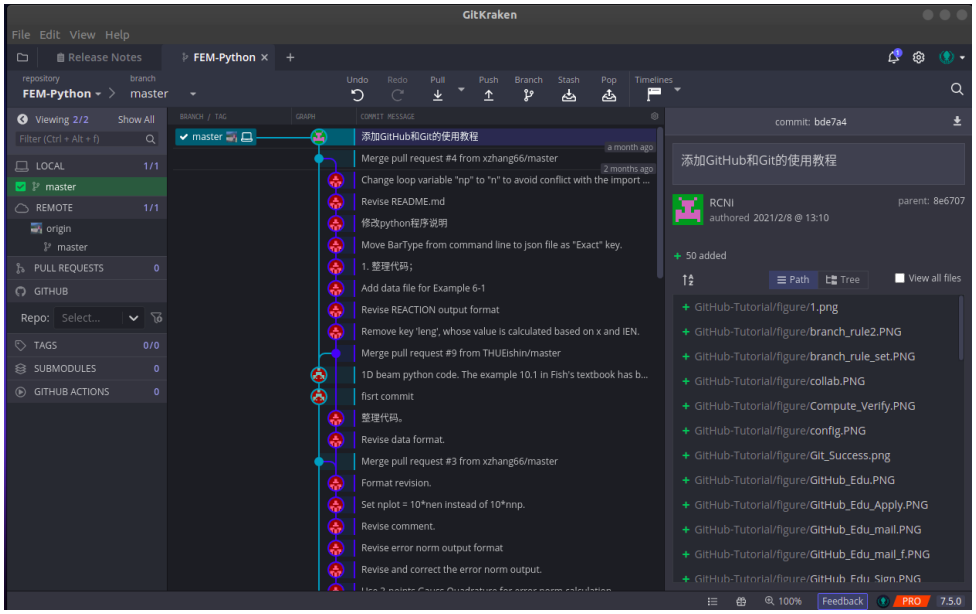


图 52: GitKraken 软件的仓库管理界面

5 小结&作业

1. 请大家两人一组进行上述简单例子的实验熟悉 Git 版本控制和 GitHub 平台；
2. 请大家百度搜索《廖雪峰的 Git 教程》进行深入学习掌握 Git 版本控制的相关概念；
3. 请大家自行熟悉 GitKraken 图形界面，多尝试一下各个按钮以及各个位置处鼠标右键的下拉菜单。可以尝试使用 GitKraken 软件来完成上述的简单例子。
4. 希望大家牢记常用的 Git 命令。