哈尔滨工业大学

**计算机科学与技术学院/国家示范性软件学院**

**2020年秋季学期**

**《软件过程与工具》课程**

**实验报告**

**Lab 2：配置管理工具Git实践**

|  |  |  |
| --- | --- | --- |
| **姓名** | **学号** | **联系方式** |
| 梅智敏 | 1183710118 | 13385658102 |

目 录

[1 实验要求 1](#_Toc55491473)

[2 安装Git 1](#_Toc55491474)

[2.1 本地机器上安装Git 1](#_Toc55491475)

[2.2 申请Gitee/GitLab/GitHub帐号 2](#_Toc55491476)

[3 Git操作过程 3](#_Toc55491477)

[3.1 实验场景(1)：仓库创建与提交 3](#_Toc55491478)

[3.2 实验场景(2)：分支管理 11](#_Toc55491479)

[3.3 实验场景(3)：远程分支管理 21](#_Toc55491480)

[4 小结 24](#_Toc55491481)

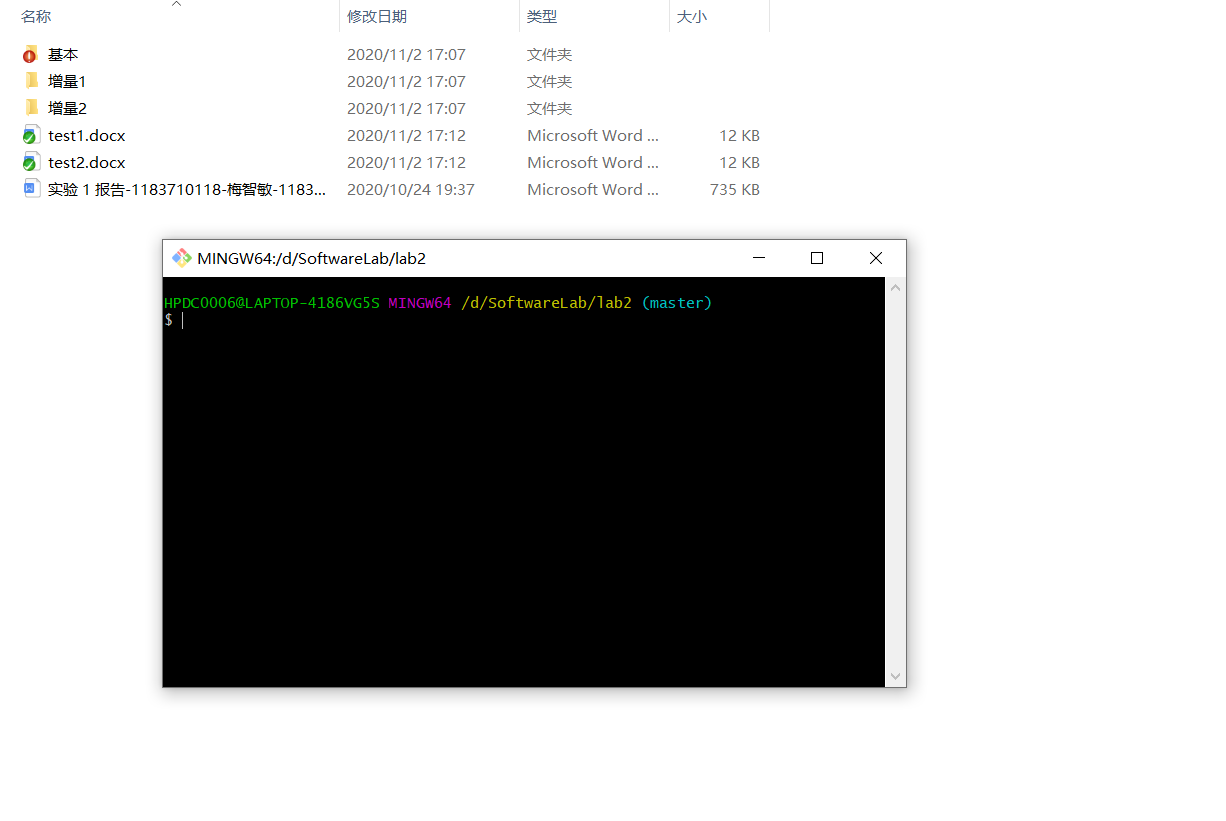
# 实验要求

* 了解配置管理工具Git及相应用环境；
* 熟练掌握Git的基本指令和分支管理指令；
* 掌握Git支持软件配置管理的核心机理；
* 在实践项目中使用Gitee/GitLab/GitHub管理自己的项目源代码；

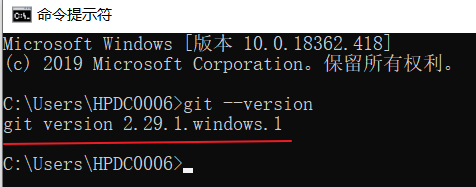
# 安装Git

## 本地机器上安装Git

运行界面：



版本号：



## 申请Gitee/GitLab/GitHub帐号

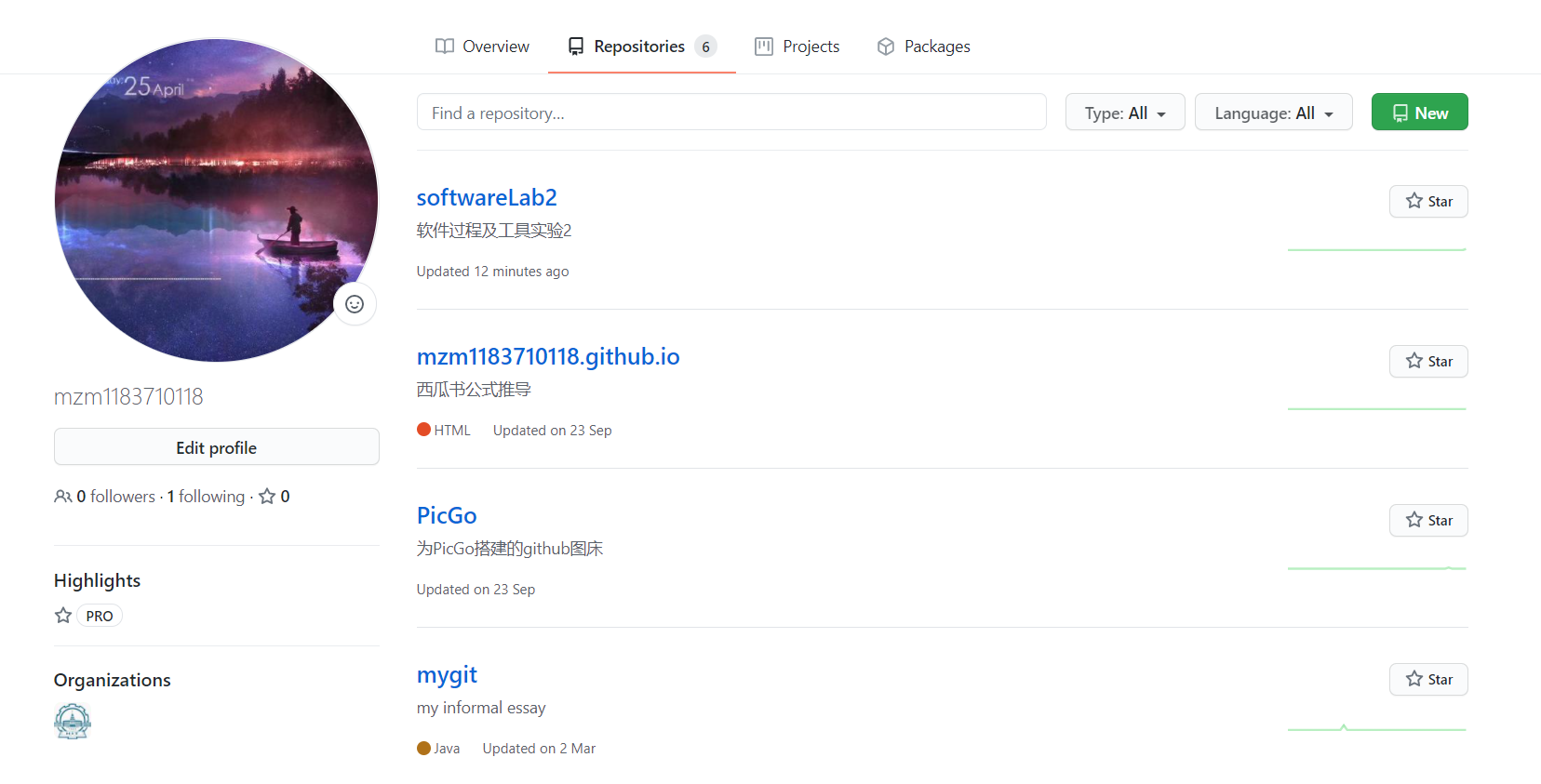
* 给出自己在Gitee/GitLab/GitHub上申请的帐号名称和本次实验中涉及的项目的URL地址。

Github的账号名称：mzm1183710118

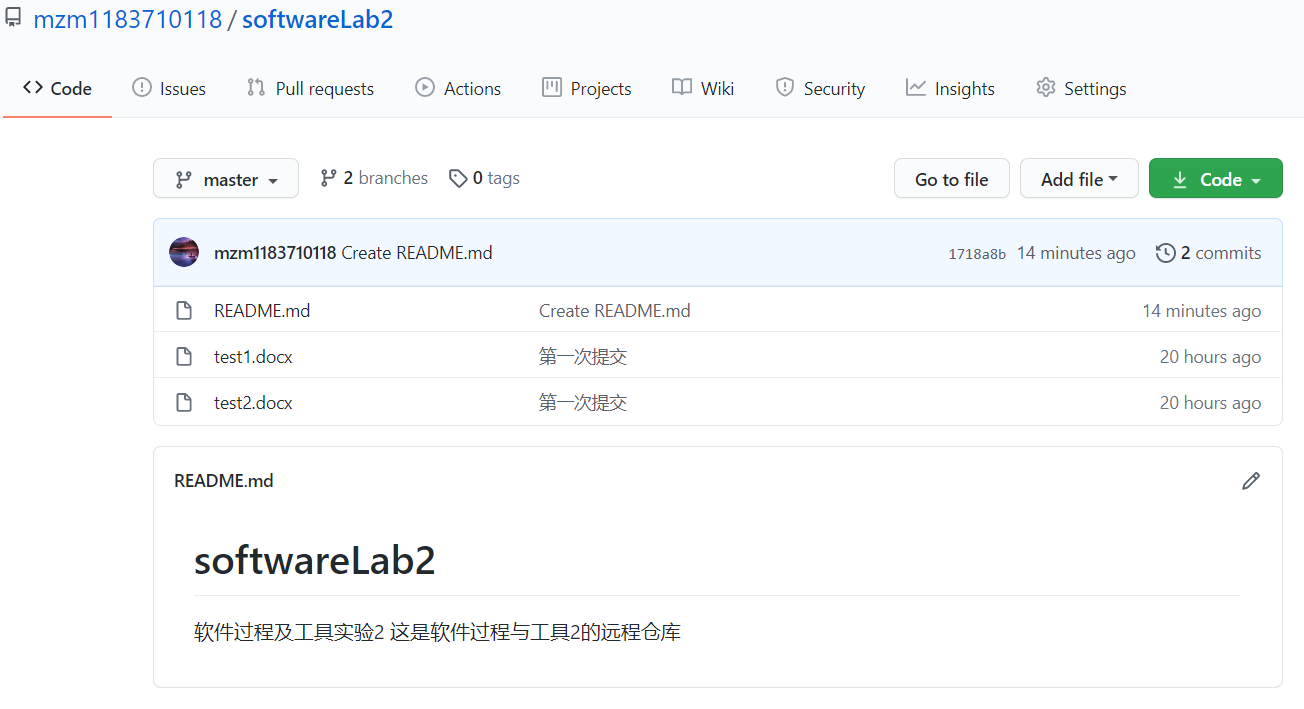
本实验项目的URL地址：<https://github.com/mzm1183710118/softwareLab2.git>

* 给出Gitee/GitLab/GitHub网站上你的账号信息和项目信息的截图。

账号信息：



项目信息：



# Git操作过程

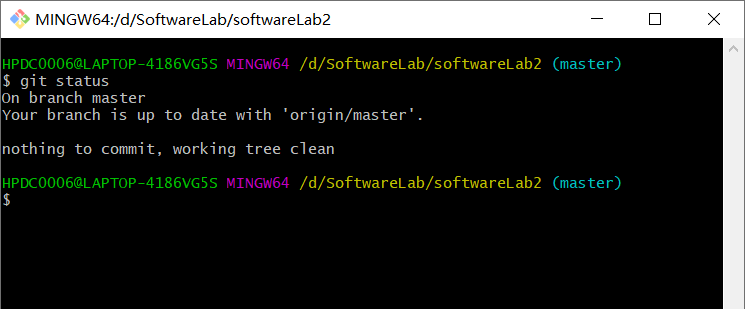
## 实验场景(1)：仓库创建与提交

给出R0~R7的操作命令，并给出执行界面的截图（命令输入界面和结果界面）。

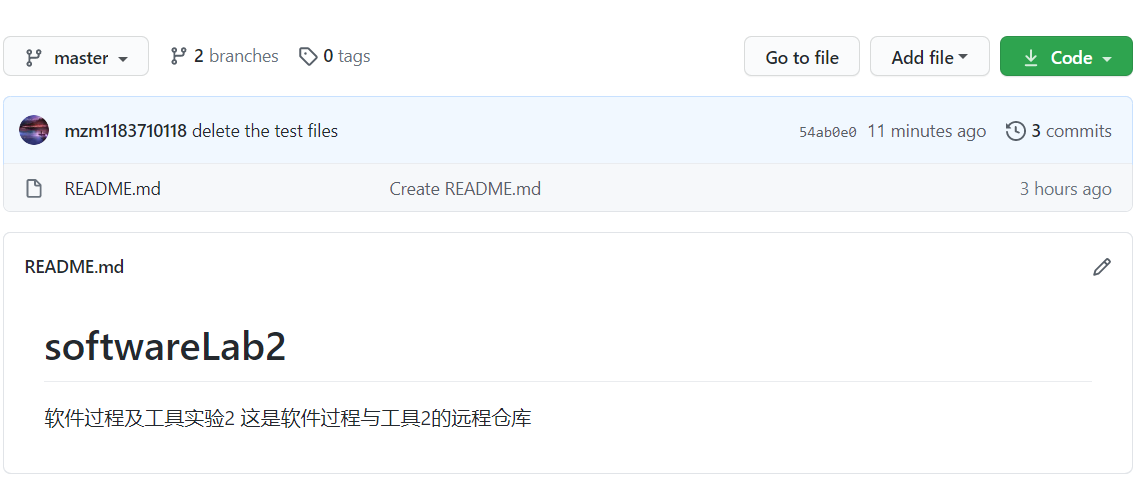
* R0指令：git status

起初，没有任何文件

输入界面：



结果界面：

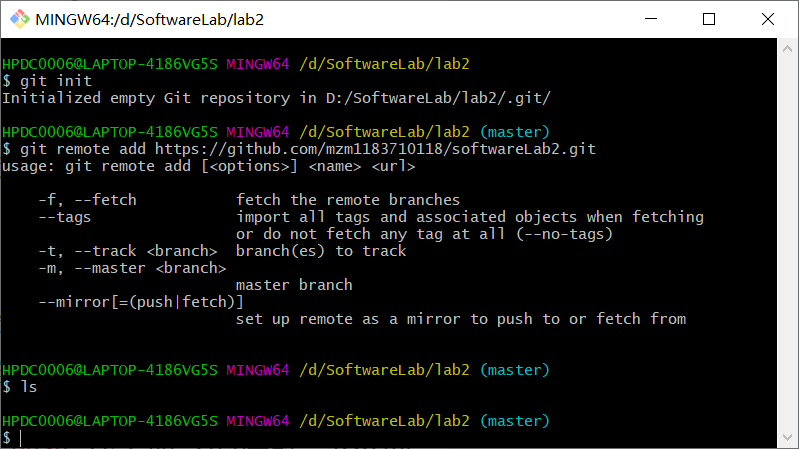


* R1指令：

初始化本地仓库：git init

本地仓库与远程仓库连接：git remote add URL

输入界面：

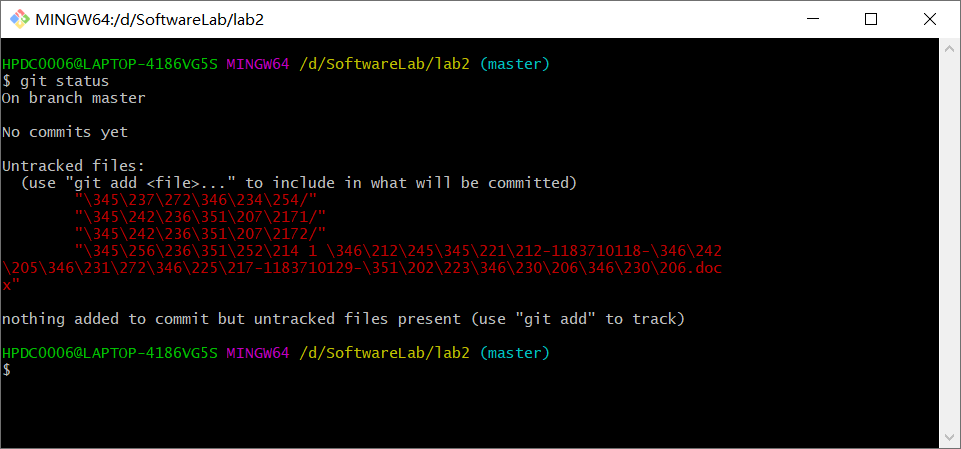


结果界面：

将lab1中的文件复制到本地目录。



此时的这些文件并未加入暂存区：



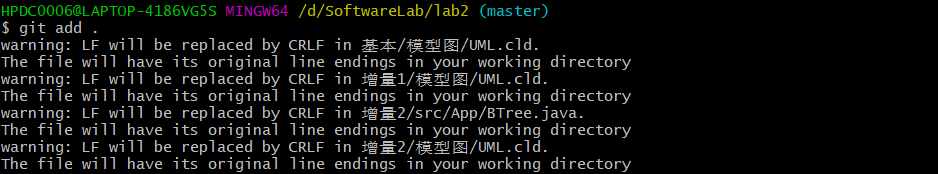
* R2指令

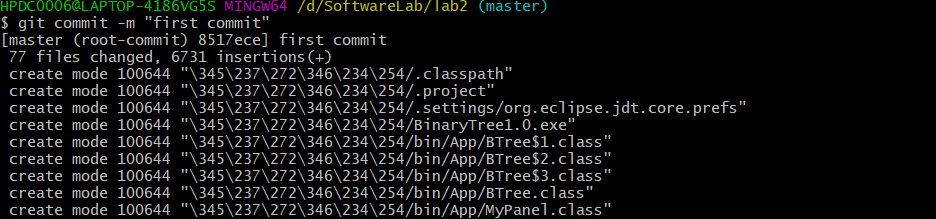
将文件加入暂存区：git add xxx

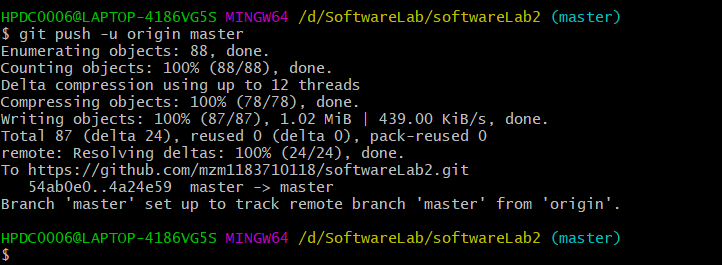
将暂存区的文件提交到本地仓库：git commit -m “message”

将本地仓库的内容推送到远程仓库：git push -u origin master

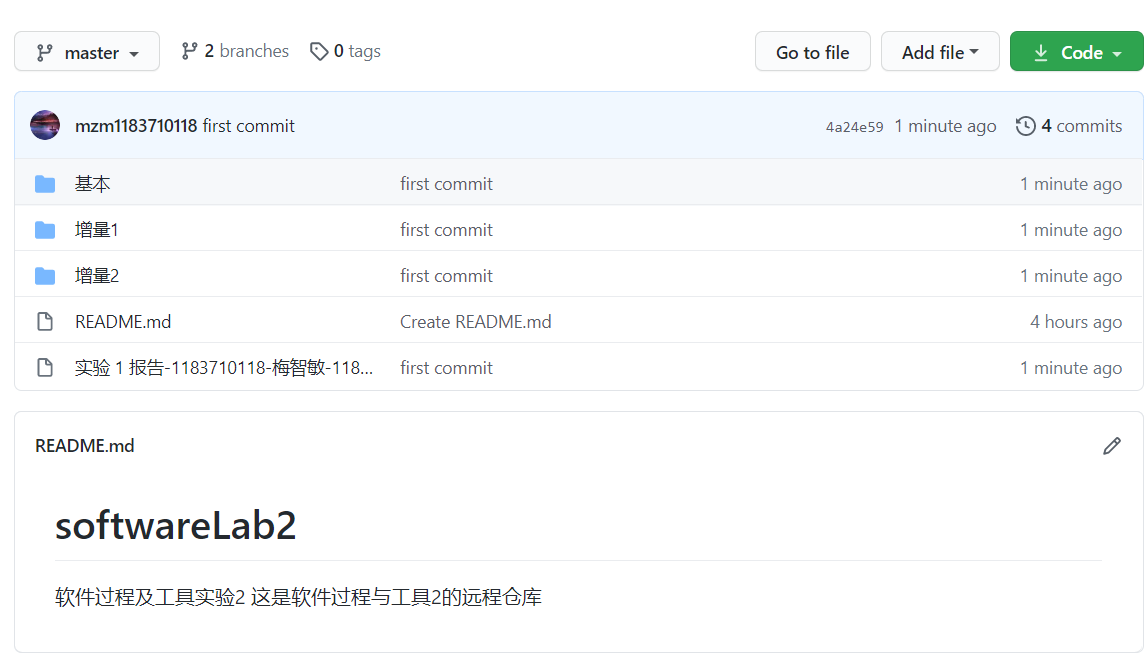
输入界面：





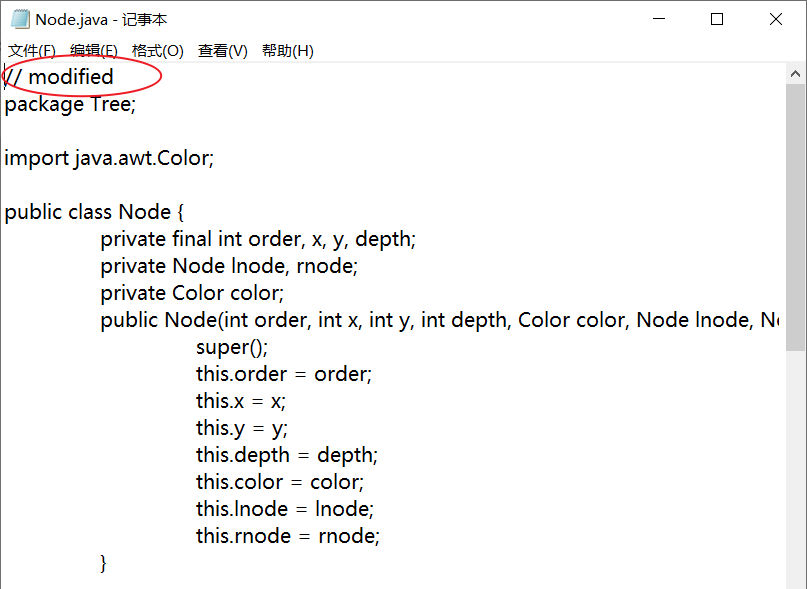


结果界面：



* R3指令

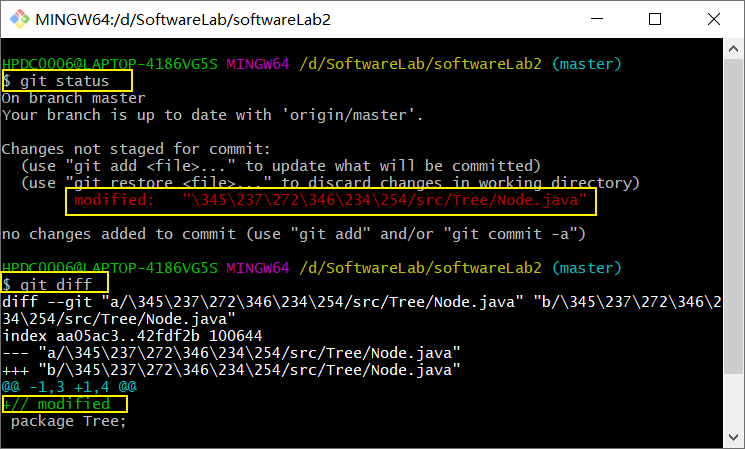
先将D:\SoftwareLab\softwareLab2\基本\src\Tree\Node.java进行手动修改：



查询哪些文件被修改：git status

查询具体修改内容：git diff

输入界面及结果界面：



* R4指令

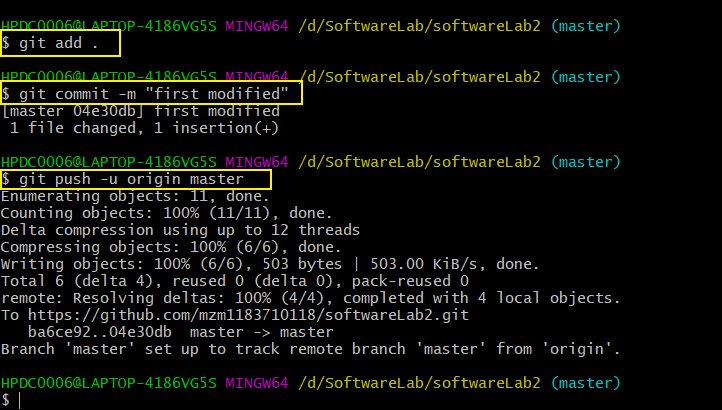
重新执行下列指令

将文件加入暂存区：git add xxx

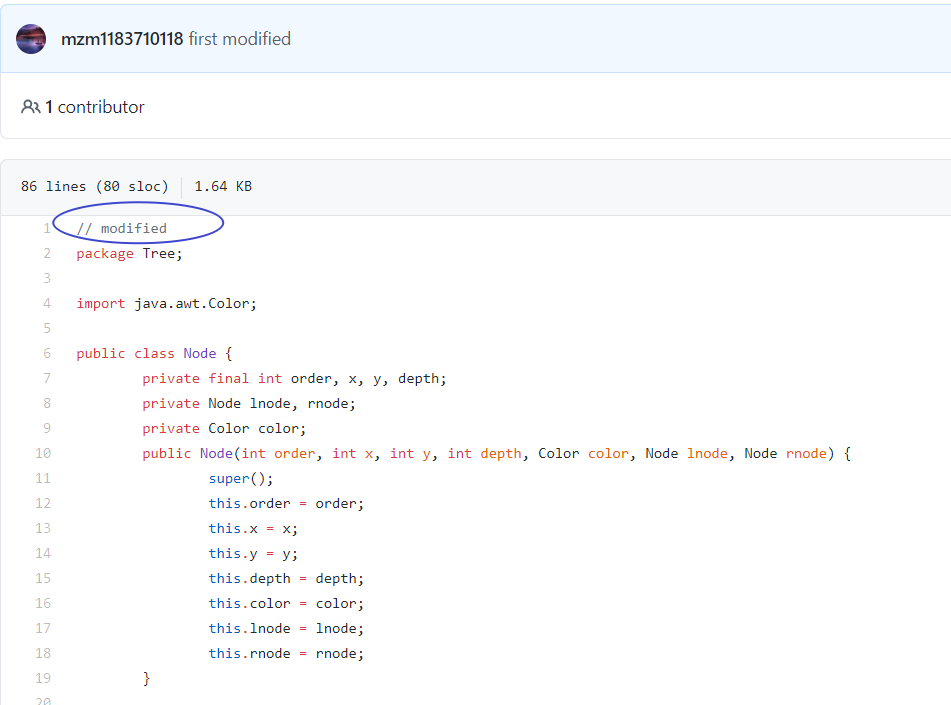
将暂存区的文件提交到本地仓库：git commit -m “message”

将本地仓库的内容推送到远程仓库：git push -u origin master

输入界面：

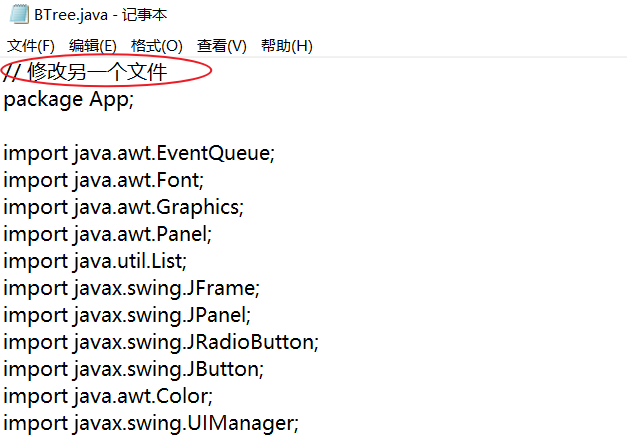


结果界面：



* R5指令

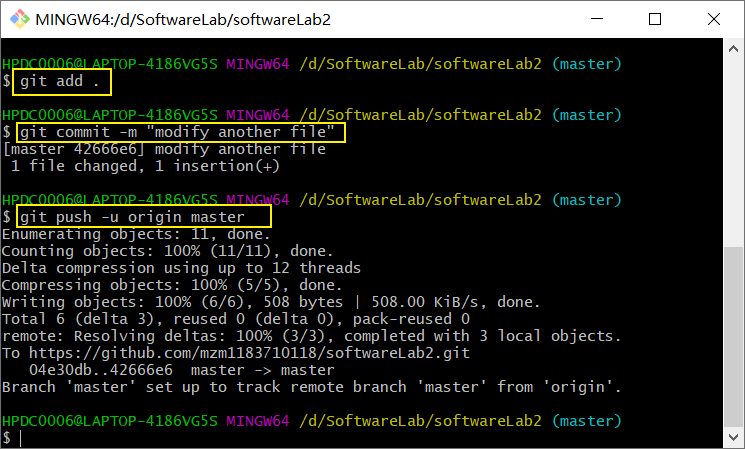
再次对另一个文件D:\SoftwareLab\softwareLab2\基本\src\App\BTree.java进行修改：



再重新提交

仍然需要重新执行 git add xxx git commit -m “message” 和 git push -u origin master

输入界面：



结果界面：



* R6指令

先撤销掉最后一次提交：git reset --hard HEAD^

再将远程仓库进行覆盖：git push -u origin master

输入界面：

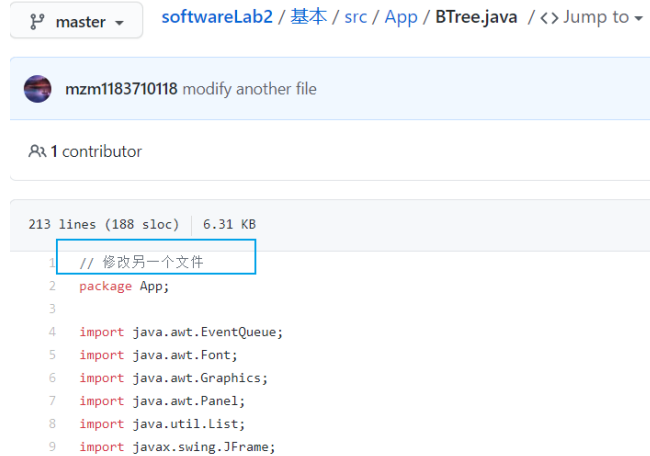




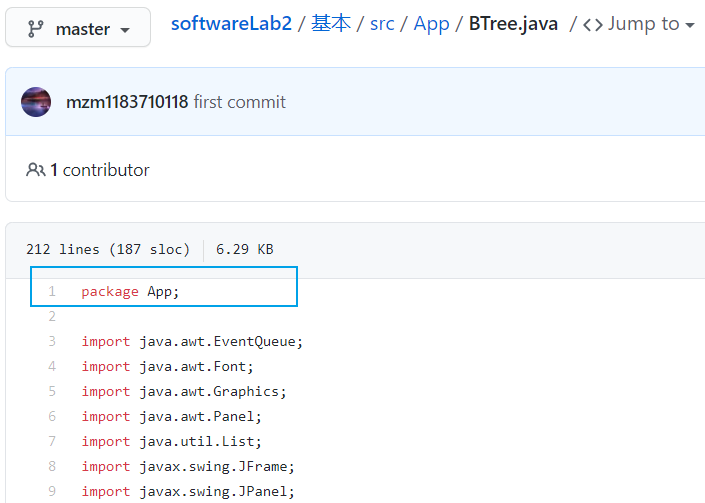
结果界面：

通过比较撤销前后的文件内容，可知我们前面的最后一次提交确实被撤销了。

撤销前：

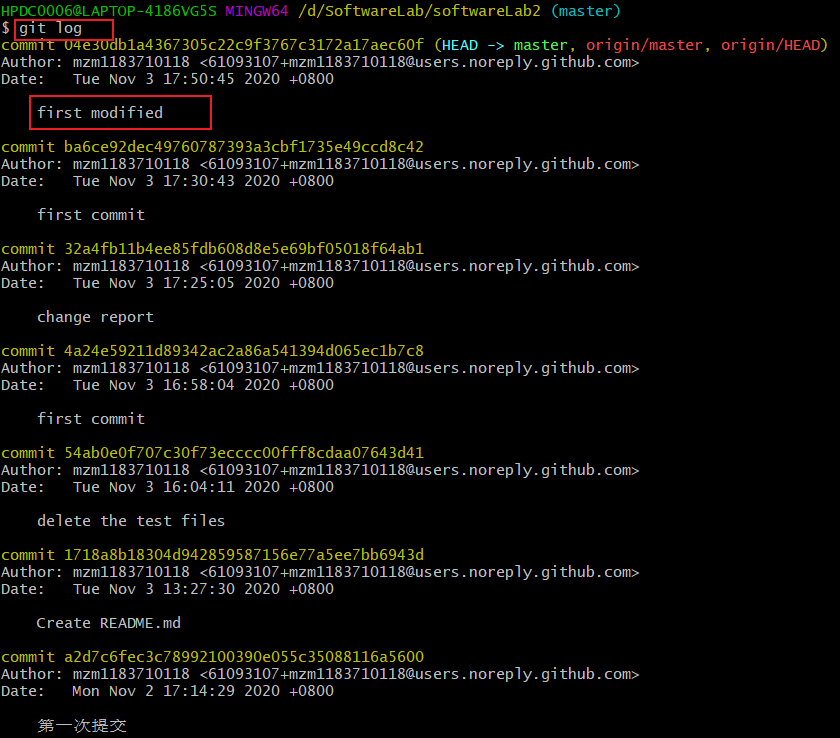


撤销后：



* R7指令

查询提交日志：git log



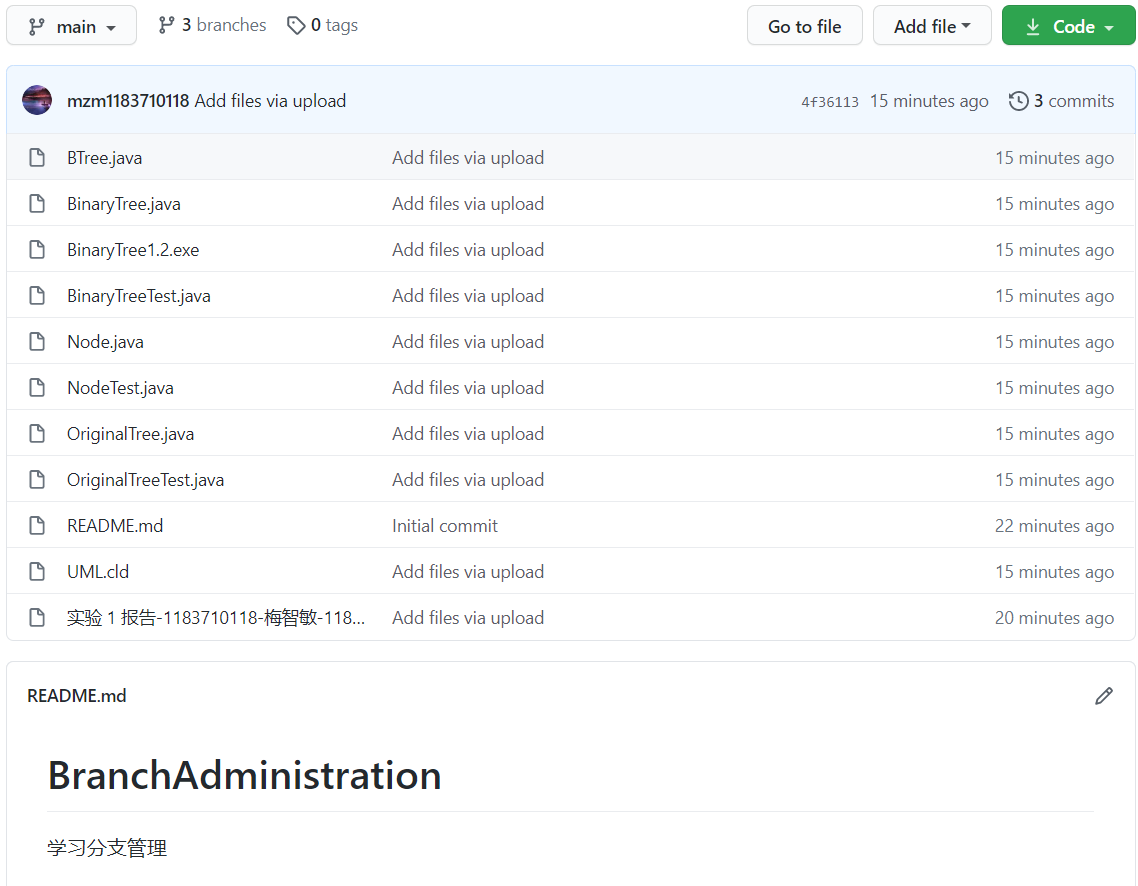
可见最近一次commit的message是“first modified”

进一步验证了我们在前面的撤销操作是成功的！

## 实验场景(2)：分支管理

给出R8~R15的操作命令，并给出执行界面的截图（命令输入界面和结果界面）。

首先在github上创建一个新仓库，并将本地的10个文件上传，形成默认分支main



然后在main分支的基础上创建2个并行的分支B2和B3，并手工对某些文件进行修改并提交：

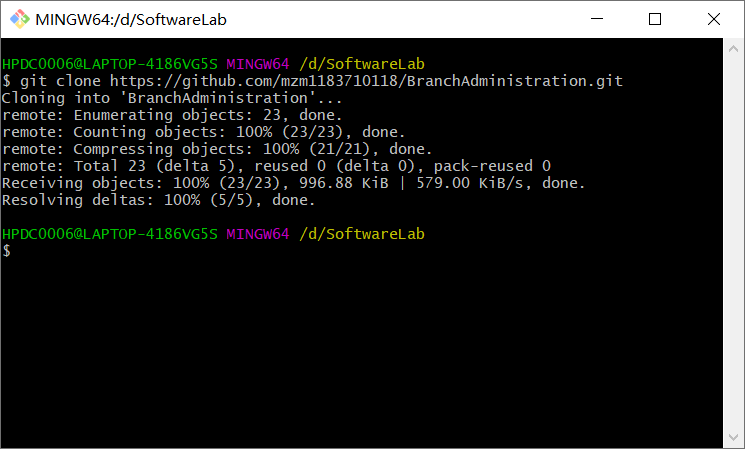




* R8指令：

克隆远程仓库： git clone URL

输入界面：



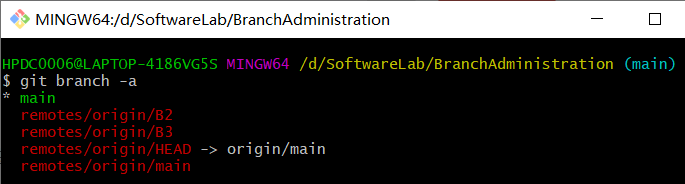
结果界面：



* R9指令：

查看远程仓库所有分支： git branch -a

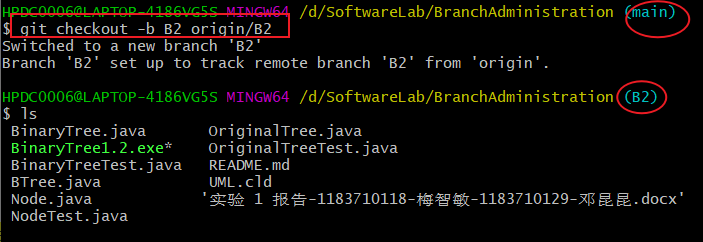
输入界面及结果界面：



获取远程仓库的某个分支：git checkout -b branch\_name origin/branch\_name

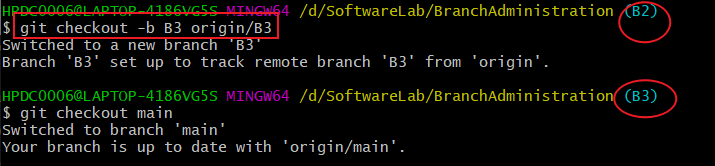
输入界面及结果界面：

获取B2分支





获取B3分支



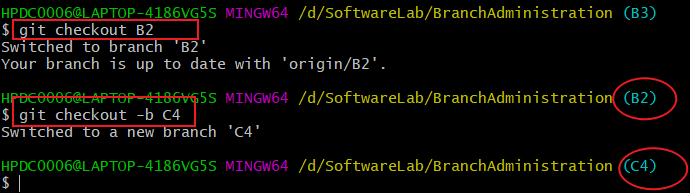


可见获取远程仓库所有分支成功！

* R10指令：

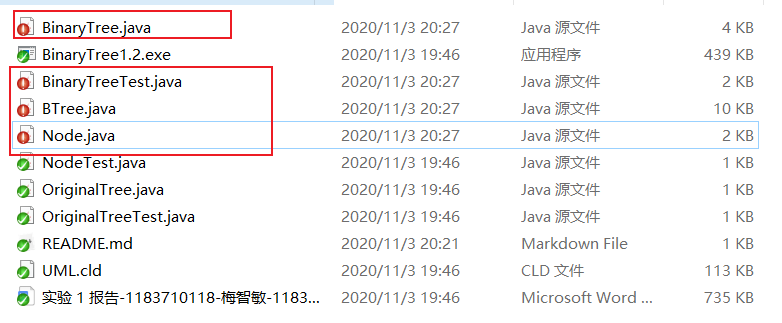
先切换到B2分支：git checkout B2

再在此分支基础上创建新分支C4：git checkout -b C4

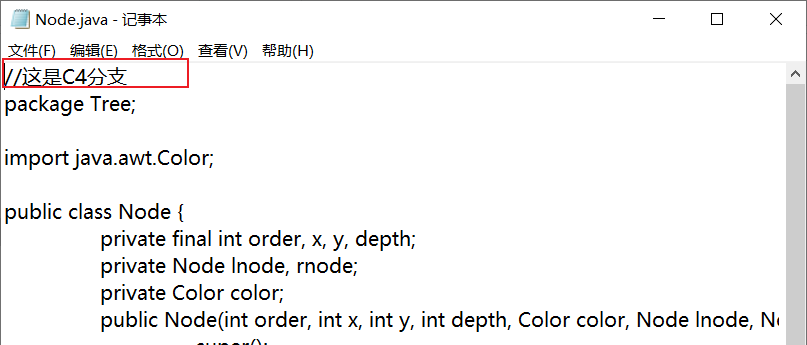


* R11指令：

在C4分支上对下面这4个文件进行修改：



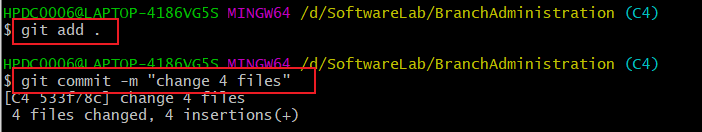
修改内容为：给每个java文件第一行加上注释//这是C4分支



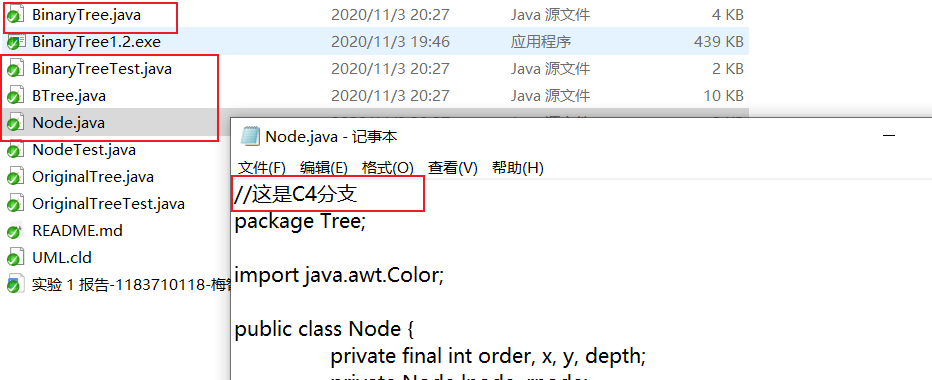
然后提交，需要依次执行下列指令

git add xxx 和 git commit -m “message”

输入界面：

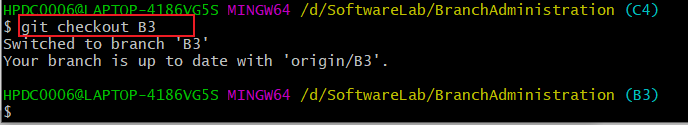


结果界面：

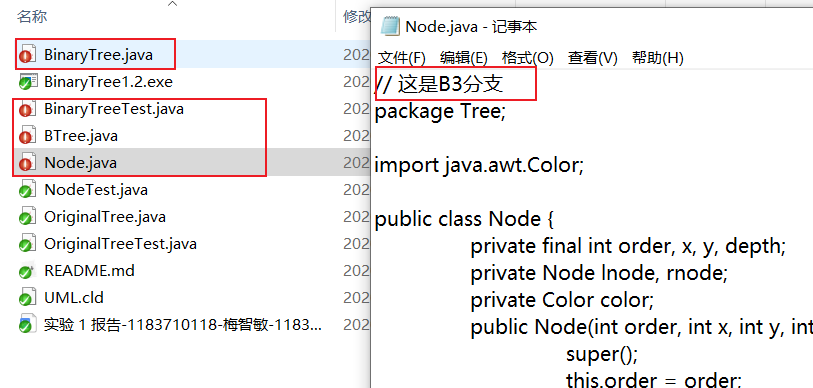


* R12指令：

首先切换到B3分支：git checkout B3



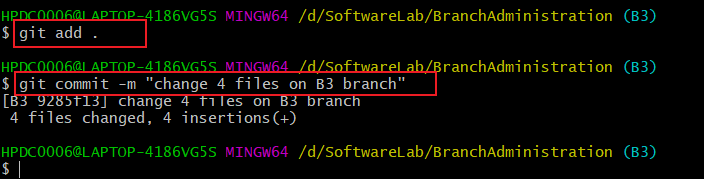
同样对4个文件进行修改



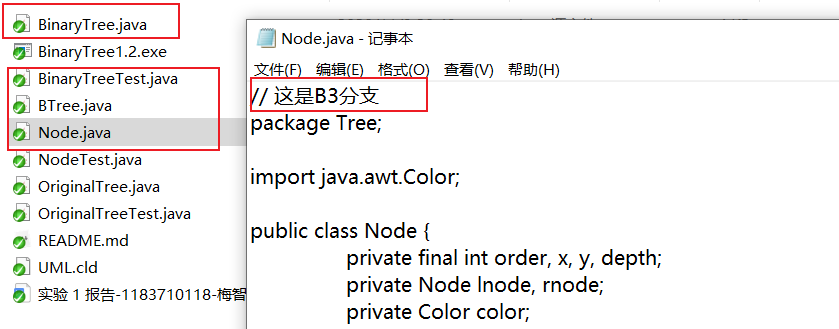
然后提交，需要依次执行下列指令

git add xxx 和 git commit -m “message”

输入界面：



结果界面：



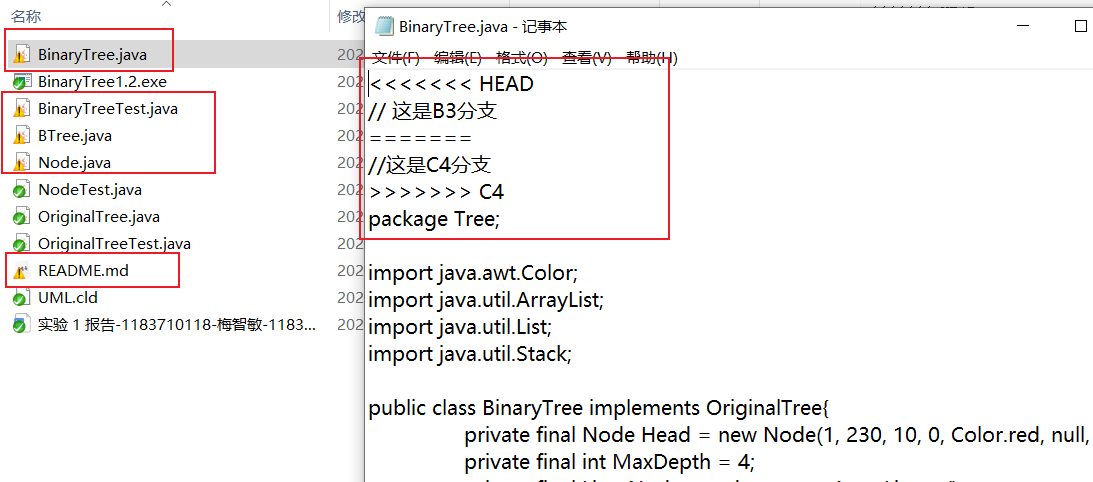
* R13指令：

合并分支指令：git merge C4

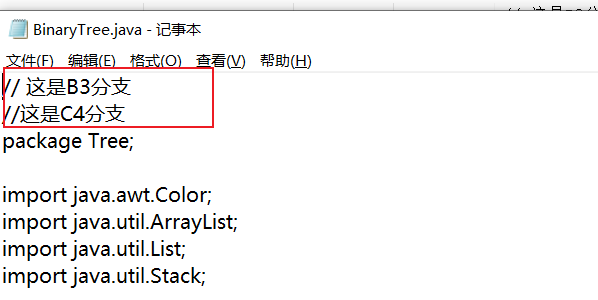


Git检查到了合并冲突。

因此需要自己手动解决：



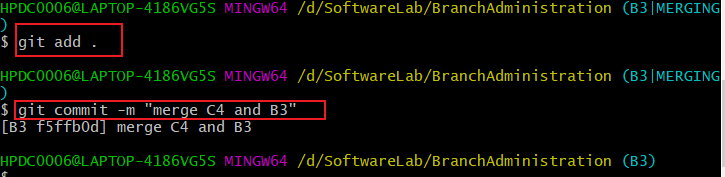
将各个冲突文件的内容手动修改为：



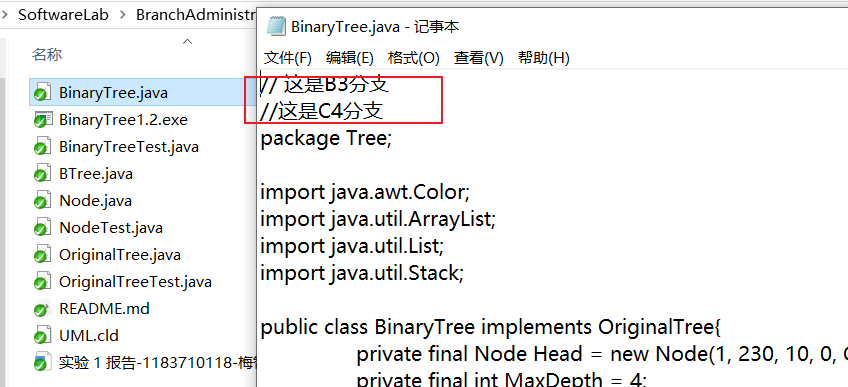
然后手动提交即可完成合并

同样依次执行：git add xxx 和 git commit -m “message”

输入界面：

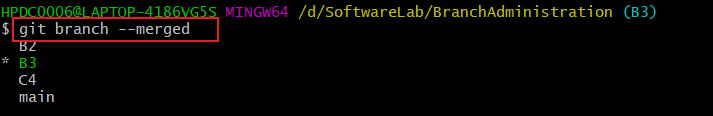


结果界面：



* R14指令：

查看哪些分支已经被合并：git branch --merged

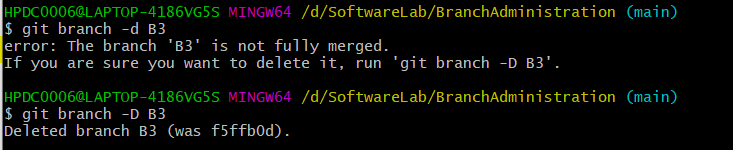


查看哪些分支没有被合并：git branch --unmerged

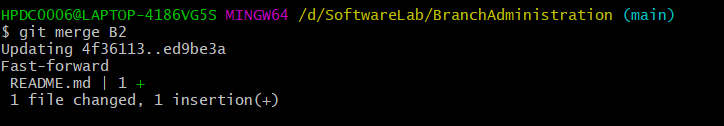


* R15指令：

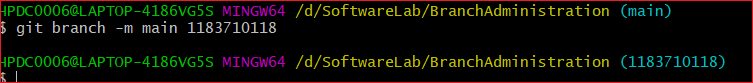
删除合并后的B3分支：git branch -d B3



将main分支和B2分支合并：git merge B2



修改分支name为我的学号1183710118：git branch -m oldname newname



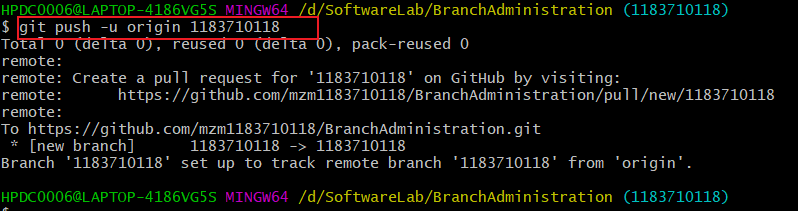
## 实验场景(3)：远程分支管理

给出R16~R18的操作命令，并给出执行界面的截图（命令输入界面和结果界面）。

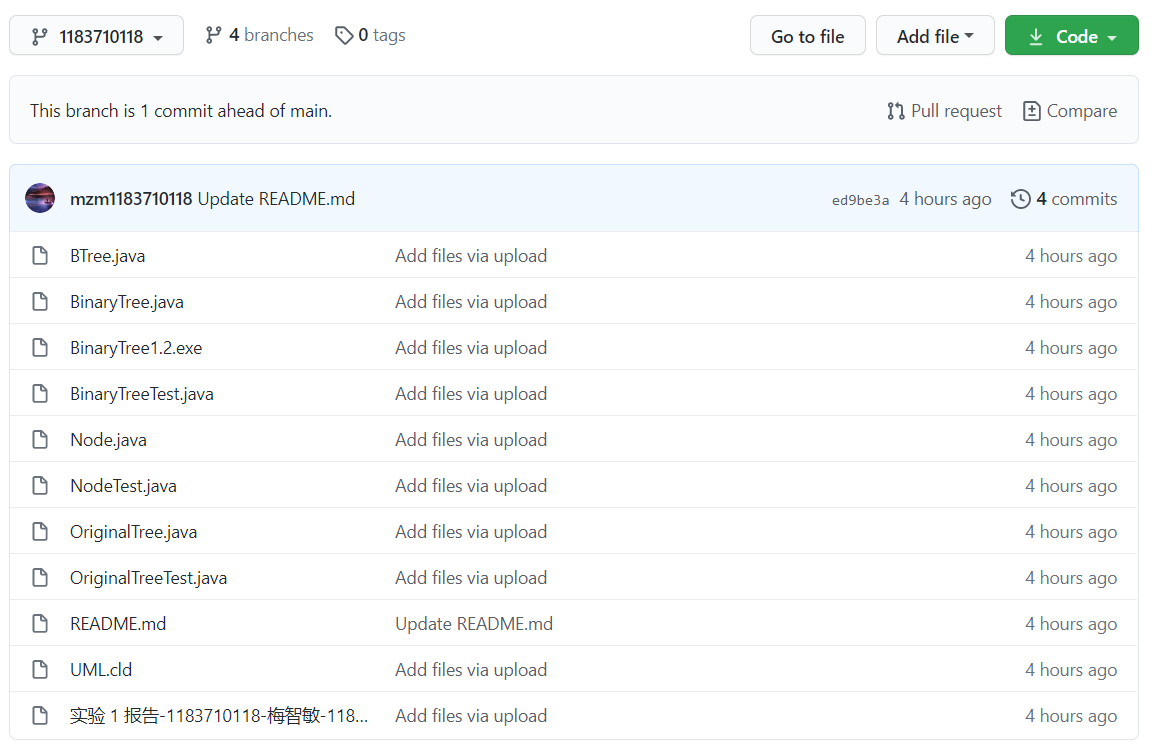
* R16指令

将本地分支推送到远程仓库：git push -u origin branchname

输入界面：



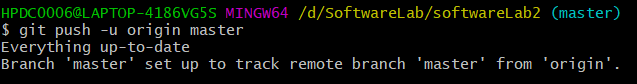
结果界面：

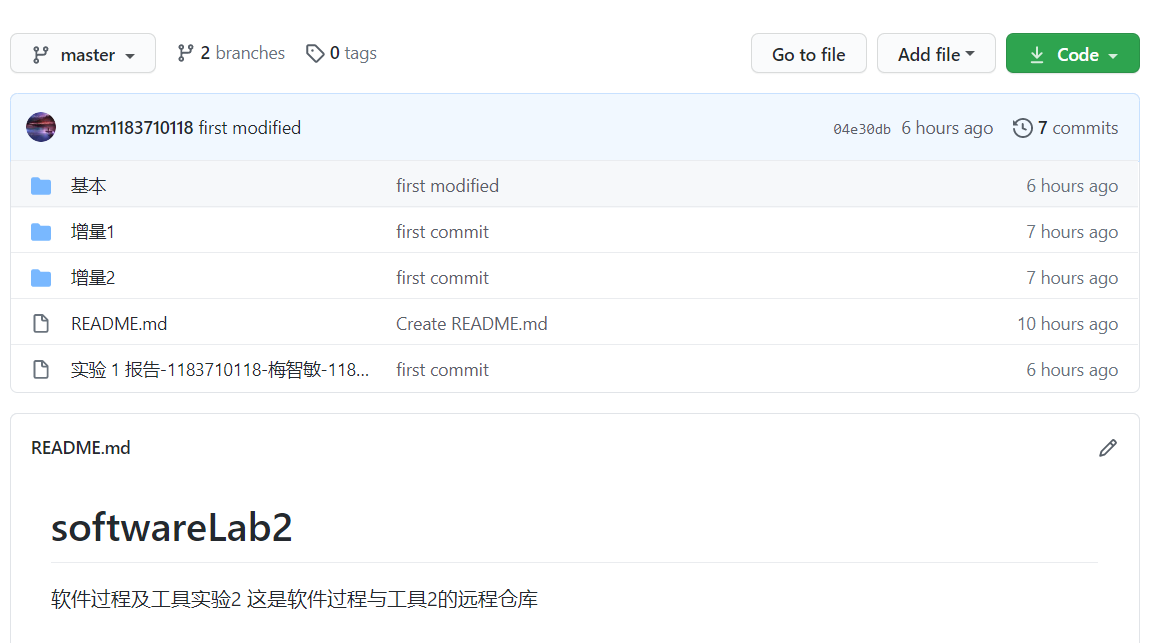


* R17指令：

将本地分支内容推送到远程仓库：git push -u origin branchname

此处的分支名即为master

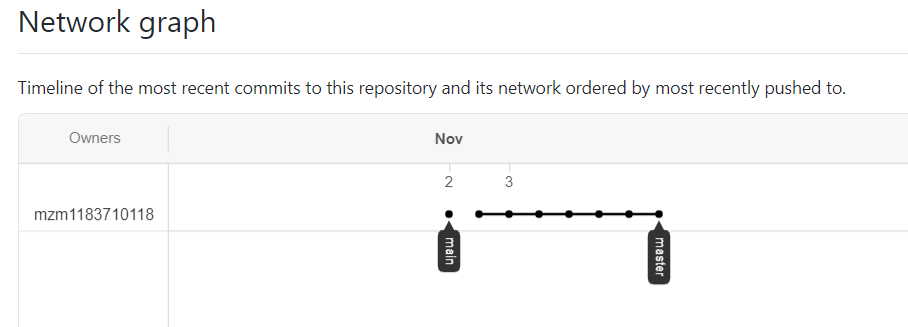




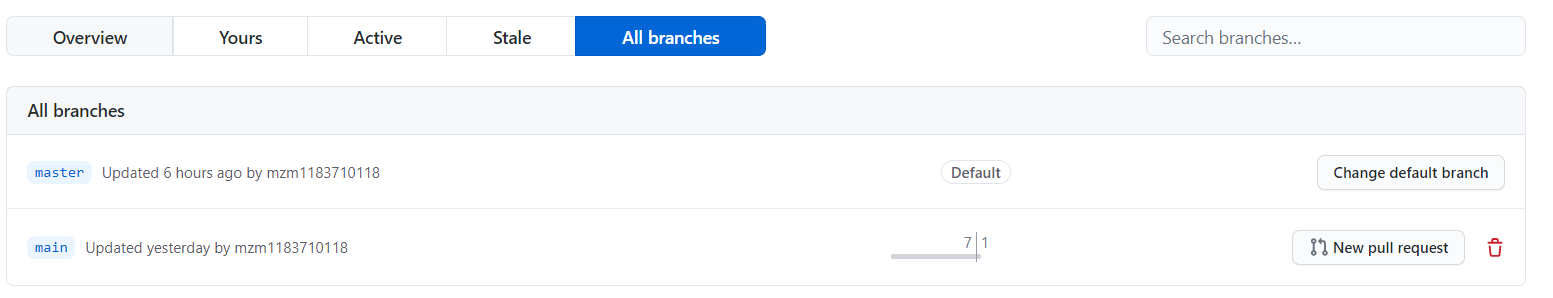
* R18指令：

第一个仓库：

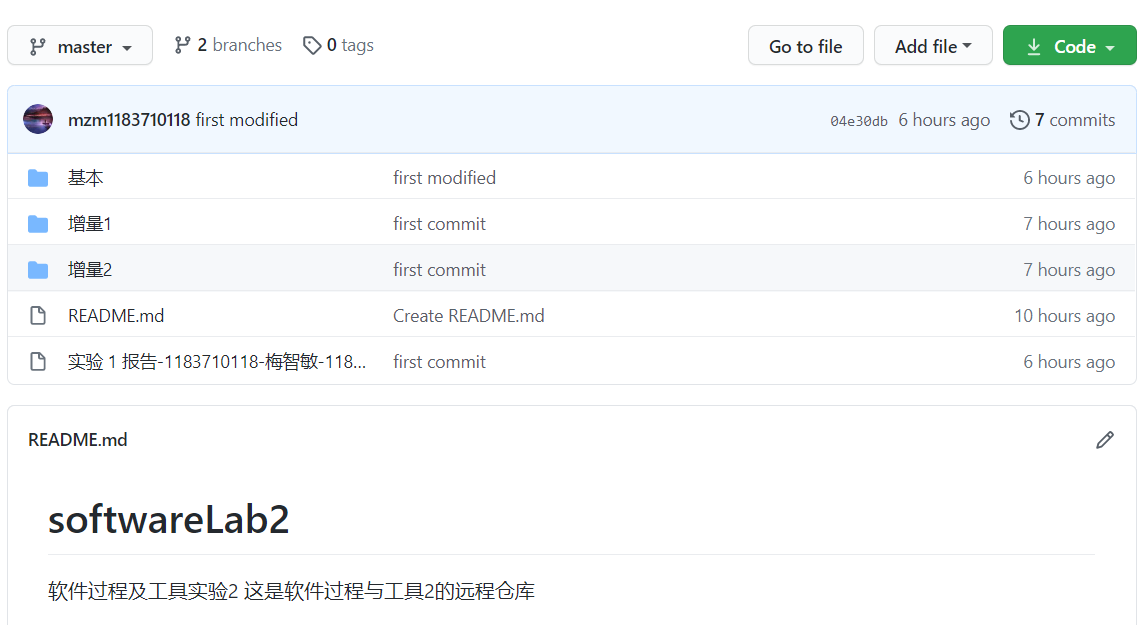
提交情况



分支情况

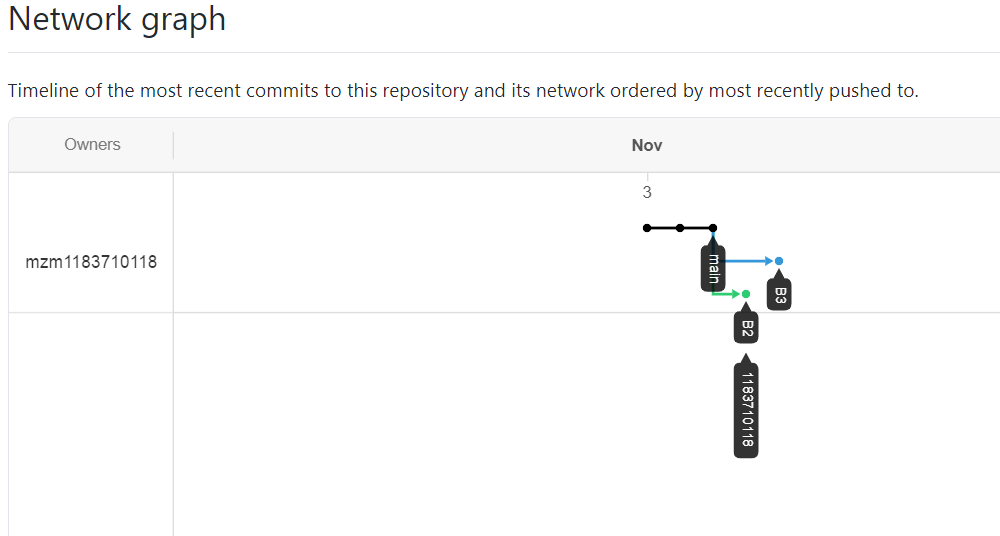


Master分支内容

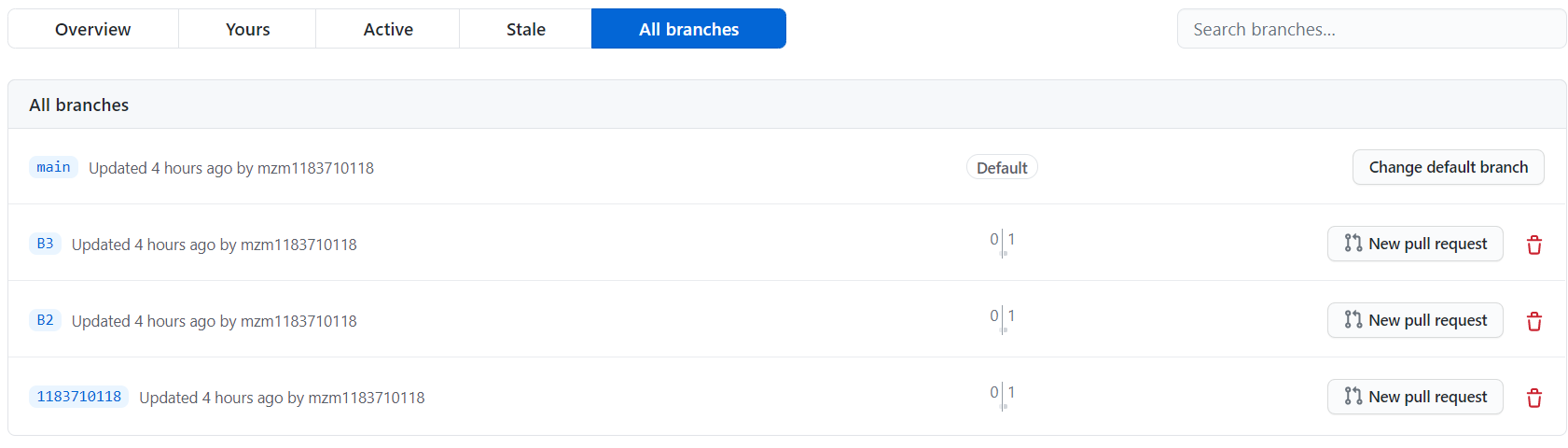


第二个仓库：

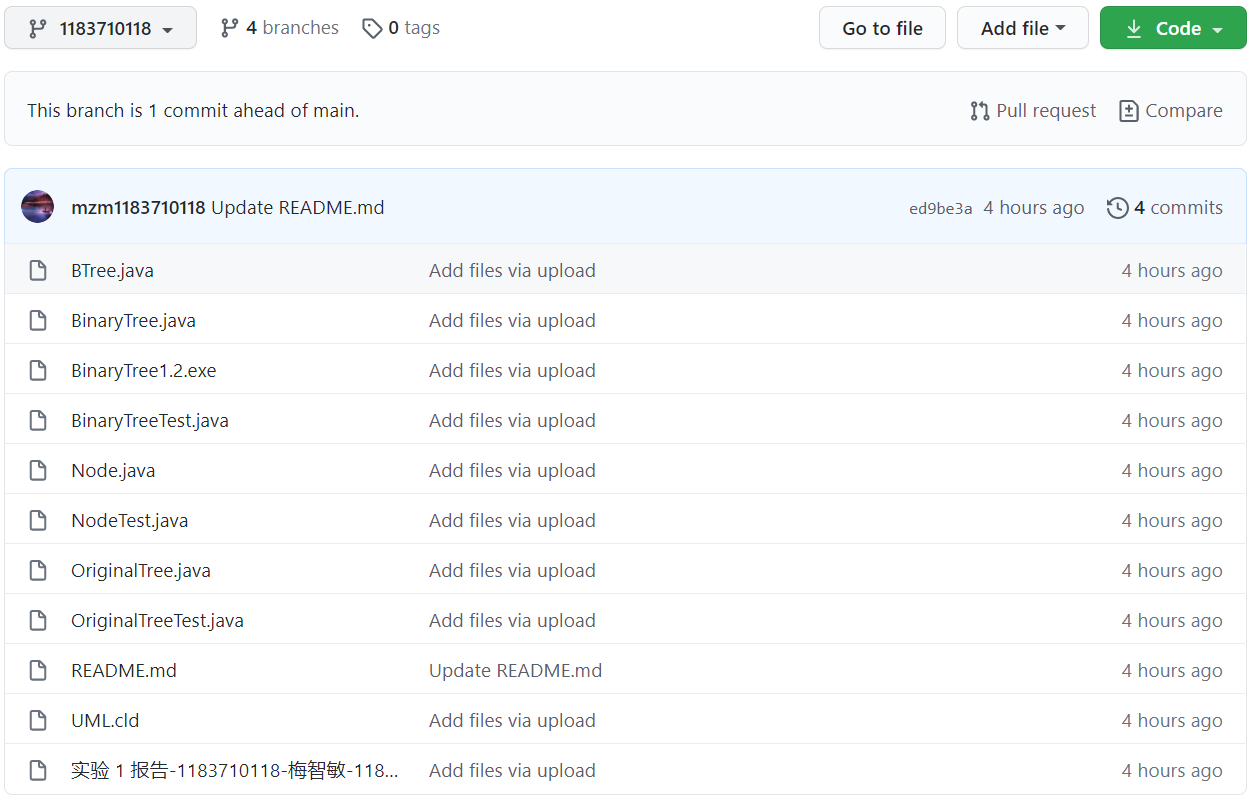
提交情况



分支情况



1183710118分支内容



# 小结

对本次实验过程和结果的思考：

1. 比较之前的开发经验，使用Git的优点？

答：可以很方便地进行版本控制管理；适合分布式开发，强调个体；速度快、灵活；允许离线工作；任意2个开发者之间可以很容易地解决冲突。

1. 在个人开发和团队开发中，Git起到的作用有何主要差异？

答：在个人开发中，Git主要的作用是版本控制，便于个人进行创建分支、版本回滚等。

而在团队开发中，Git主要的作用是帮助团队协作，每个开发者从远程仓库中pull所需要的分支内容，然后在本地进行开发。开发完成后再push到远程仓库，这样就能高效地完成团队协作而不需要点对点通信。

1. 之前是否用过其他的版本控制软件？如果有，同Git相比有哪些优缺点？如果没有查阅资料对比一下不同版本控制系统的差别。

答：之前用过SVN，同Git相比的优缺点如下

优点：

每个人都能看到其他用户在做什么；管理员可以给用户分配权限

缺点：

1. 如果中央服务器出现故障，那么在故障期间，用户无法提交代码，无法协同工作，有丢失数据的风险。
2. 提交必须有网络连接（非本地存储的版本库）
3. 并非每次提交都会成功，如果两人修改了一份文件，并且对方先于你提交，则系统会提示你“请先更新再提交”，然后解决冲突  之后才能提交

不同版本控制系统的差别：

* GIT是分布式的，SVN不是
* GIT把内容按元数据方式存储，而SVN是按文件
* GIT分支和SVN的分支不同
* GIT没有一个全局的版本号，而SVN有
* GIT的内容完整性要优于SVN

1. 在什么情况下适合使用Git、什么情况下没必要使用Git？

答：当你的项目有多个版本时，适合使用Git

当你的项目很小型，没有较多版本时，没必要使用Git

1. 其他相关思考。

答：使用Git有许多好处，能够大大便捷我们的开发过程

* 本地拥有版本库，随时进行版本后退
* 非常简单的建立分支
* 速度更快，特别是熟悉git命令后
* 指定和若干不同的远端代码仓库进行交互