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**

**2021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** |
| 傅浩东 | 1190202105 | 1190202105@stu.hit.edu.cn |

目 录

[1 实验要求 1](#_Toc86397007)

[2 安装Git 1](#_Toc86397008)

[2.1 本地机器上安装Git 1](#_Toc86397009)

[2.2 申请Gitee/GitLab/GitHub帐号 2](#_Toc86397010)

[3 Git操作过程 3](#_Toc86397011)

[3.1 实验场景(1)：仓库创建与提交 3](#_Toc86397012)

[3.2 实验场景(2)：分支管理 7](#_Toc86397013)

[3.3 实验场景(3)：远程分支管理 12](#_Toc86397014)

[4 小结 14](#_Toc86397015)

# 实验要求

简要复述实验手册中要求达到的实验目标与过程。

**实验目标：**

1. 了解配置管理工具Git及相应用环境；

2. 熟练掌握Git的基本指令和分支管理指令；

3. 掌握Git支持软件配置管理的核心机理；

4. 在实践项目中使用Gitee/GitLab/GitHub管理自己的项目源代码。

**实验要求：**

1. 本次实验由个人单独完成；

2. 按照第三条步骤进行实验；

3. 遵循实验报告模板撰写报告；

. 上交文件：

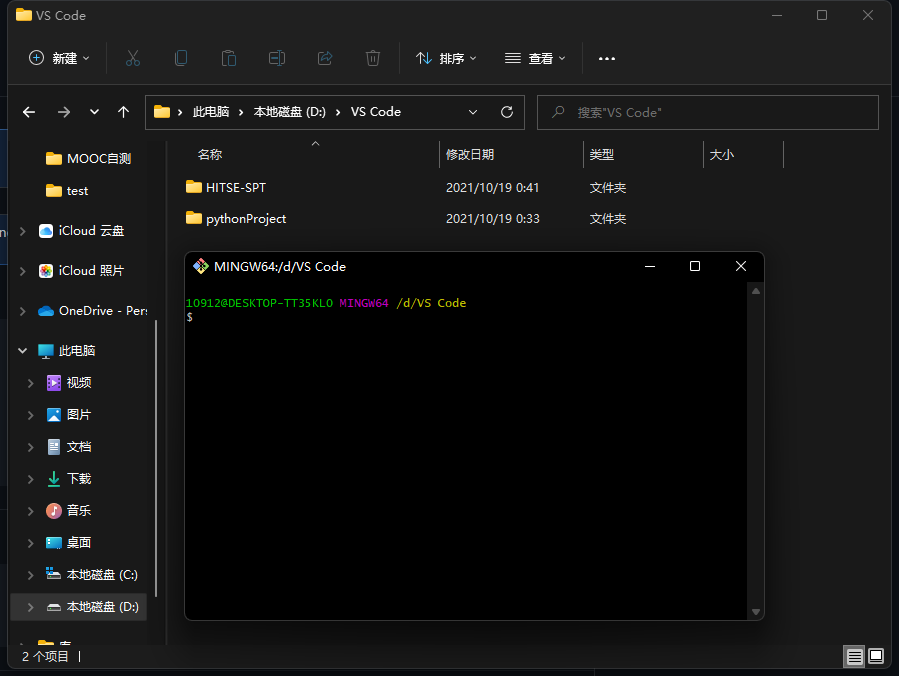
1. 实验报告文件命名：《软件过程与工具》实验2报告-学号-姓名.docx；
2. 上交时间：第9周周三（2021.11.03）24：00前；
3. 上交方式：发送到老师信箱：fgx@hit.edu.cn。

# 安装Git

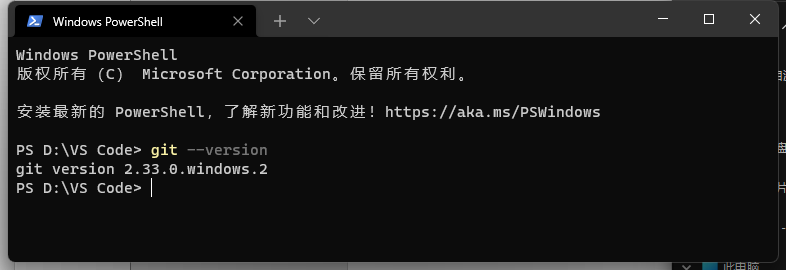
## 本地机器上安装Git

如果在Windows下安装Git，给出安装的Git版本号和在本地机器上安装Git后的运行界面（给出主要界面即可）。

Git Bash运行界面



Git版本号



## 申请Gitee/GitLab/GitHub帐号

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

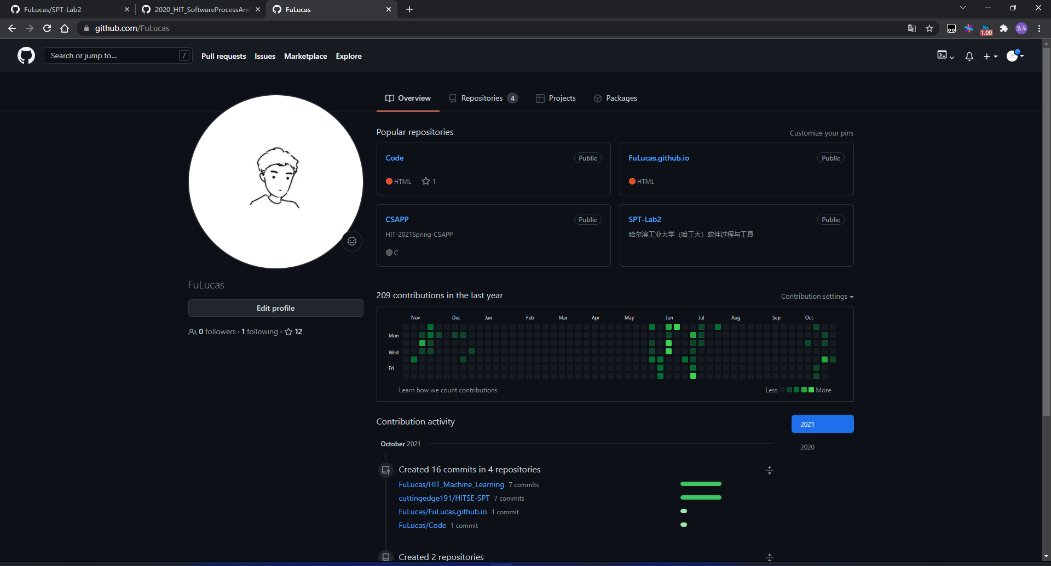
Github的账号名称：FuLucas

本实验项目的URL地址：<https://github.com/FuLucas/SPT-Lab2>

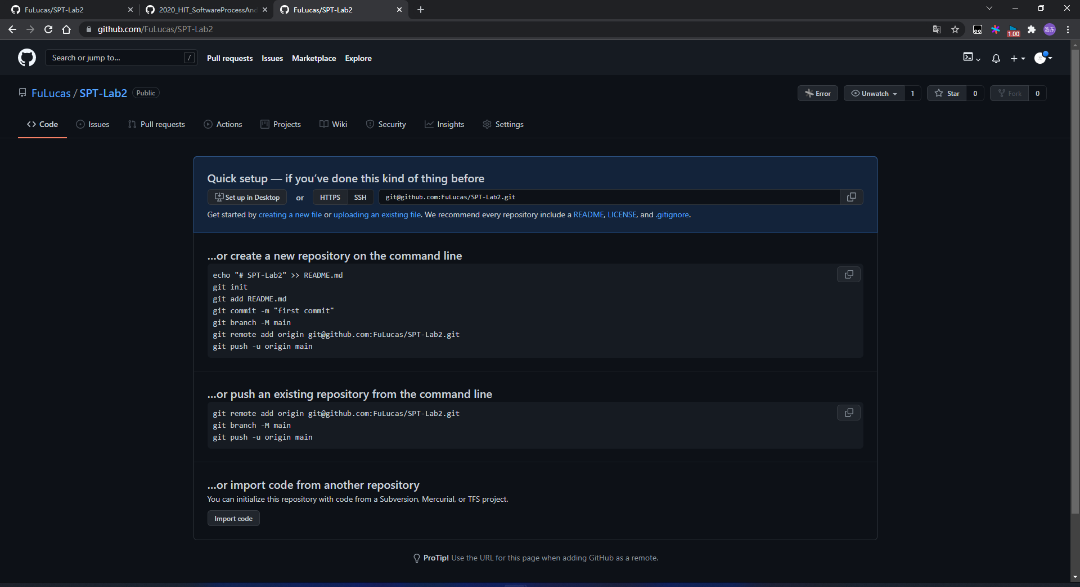
以及地址：<https://github.com/FuLucas/SPT_Lab2_R1_7>

GitHub网站上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

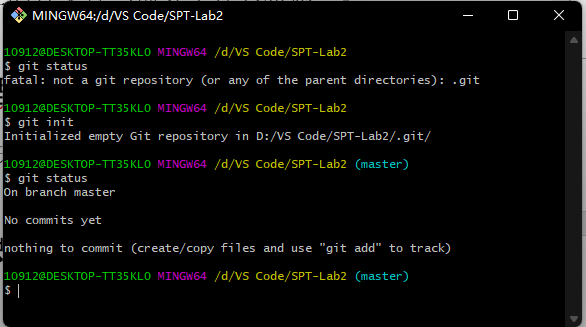
给出R0~R7的操作命令，并给出执行界面的截图（命令输入界面和结果界面）。

1. ***R0***：针对R1和R7，在进行每次Git操作之前，随时查看**工作区、暂存区、Git仓库的状态**，确认项目里的各文件当前处于什么状态；

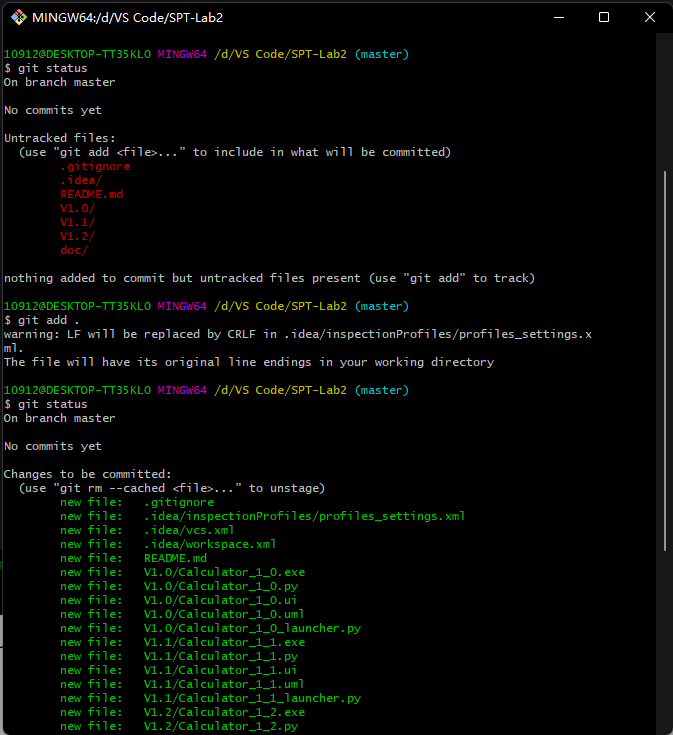
命令：git status

1. ***R1***：本地初始化一个Git仓库，将自己在Lab1中所创建项目的全部源文件加入进去，纳入Git管理；

初始化Git仓库

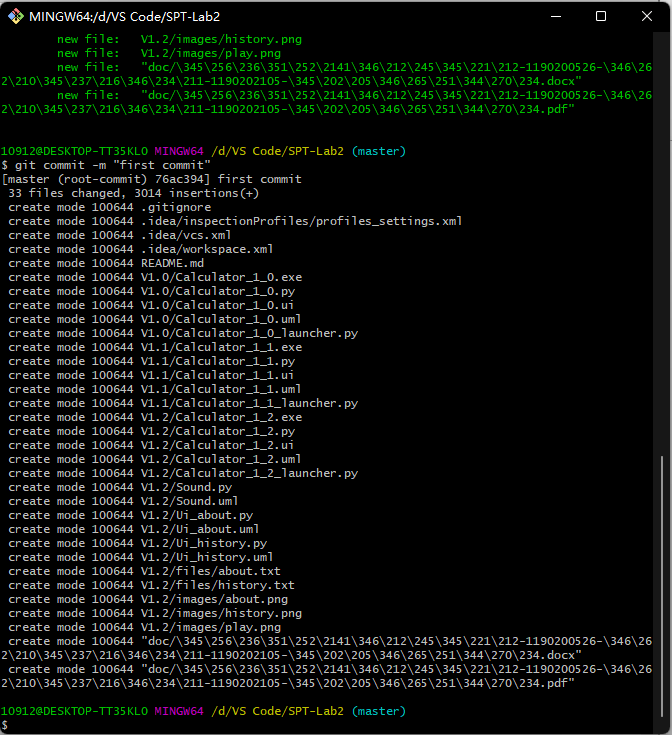


Lab1中所创建项目的全部源文件加入



1. ***R2***：提交

命令：git commit -m “commit information”



1. 手工对Lab1的某个文件进行修改

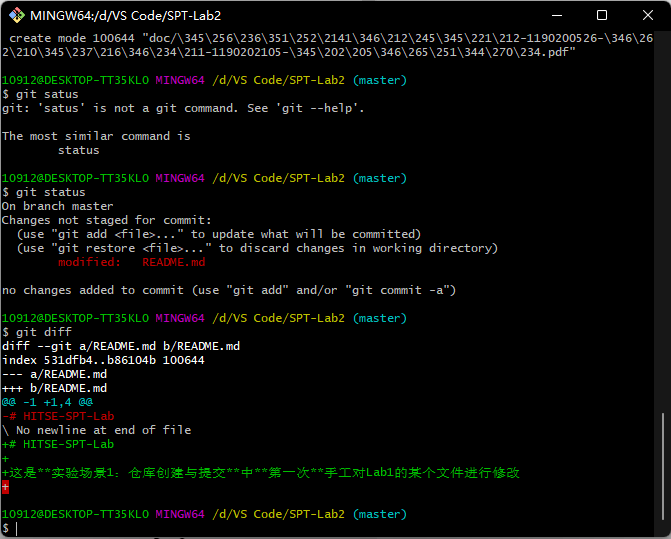
对README.md增加一行：“这是**实验场景1：仓库创建与提交**中**第一次**手工对Lab1的某个文件进行修改”，如下改变前后对比：





1. ***R3***：查看上次提交之后都有哪些文件修改、具体修改内容是什么（查看修改后的文件和暂存区域中相应文件的差别）；

查看差别命令：git diff



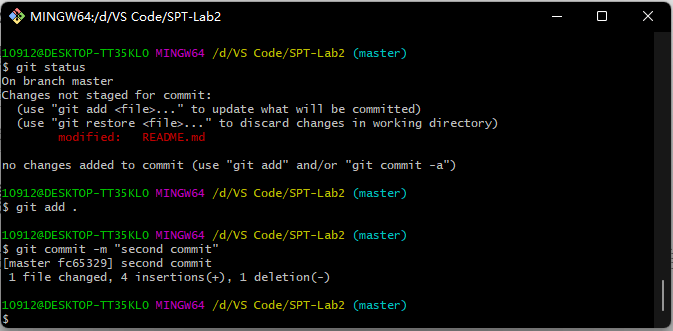
**文件修改**：README.md

**具体修改内容**：增加一行：“这是**实验场景1：仓库创建与提交**中**第一次**手工对Lab1的某个文件进行修改”

1. ***R4***：重新提交；

Git命令前状态

命令：git add . 以及git commit -m “second commit”



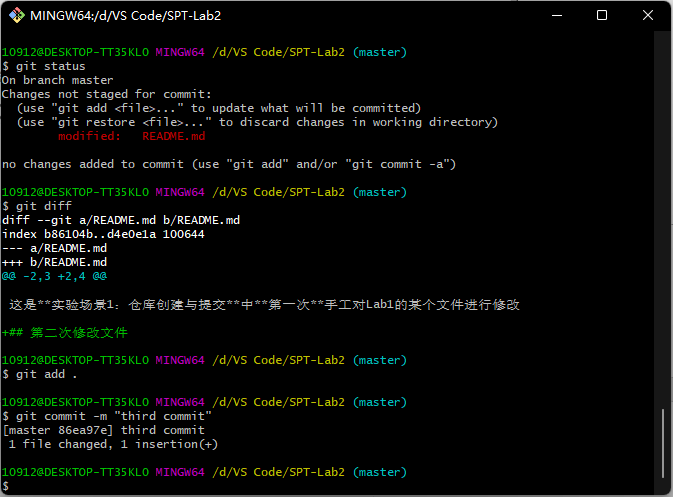
1. 再次对Lab1的某个文件进行修改；

依然修改文件README.md，修改后的状态如下：



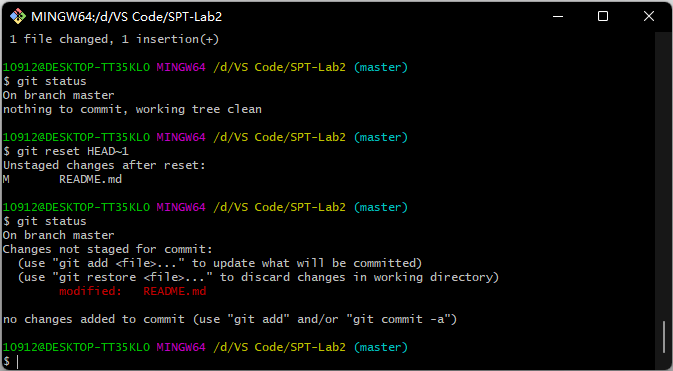
1. ***R5***：重新提交；

提交前状态以及重新提交



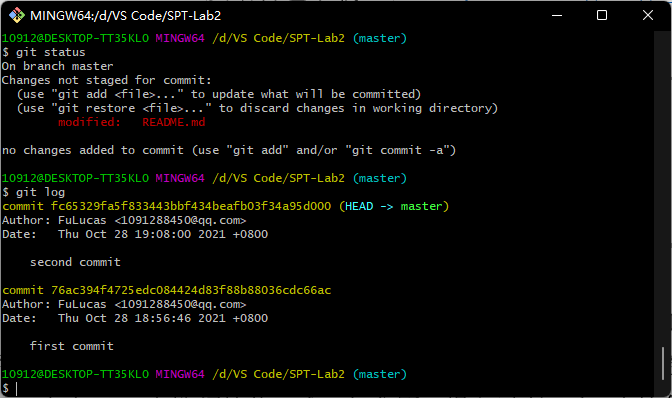
1. ***R6***：把最后一次提交撤销；

命令：git reset HEAD~1



1. ***R7***：查询提交记录。

命令：git log

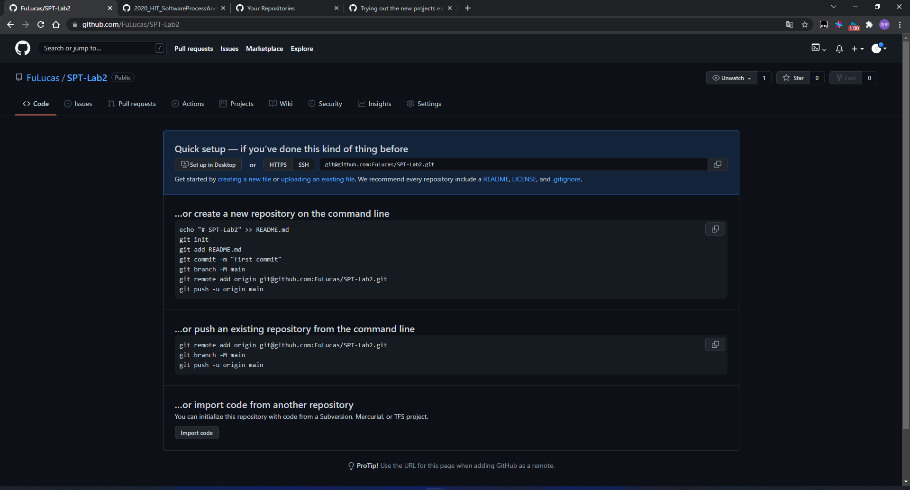


## 实验场景(2)：分支管理

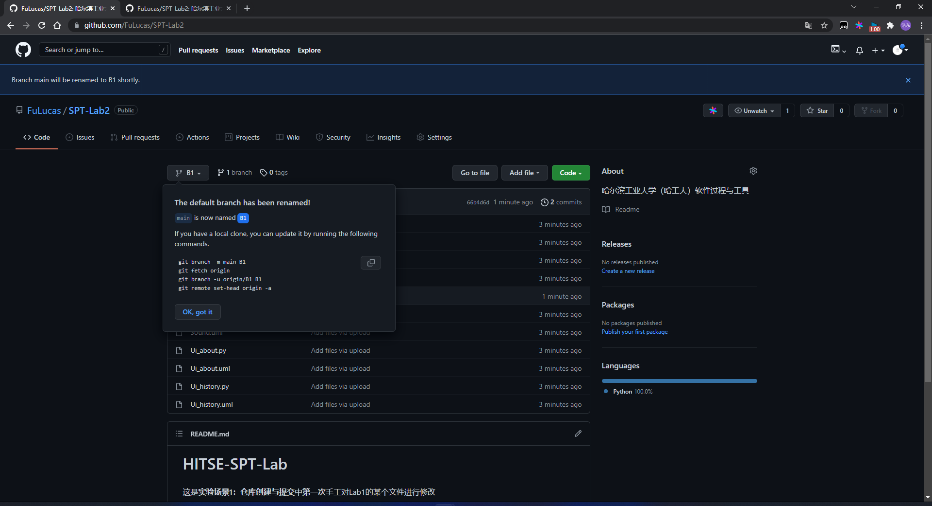
给出R8~R15的操作命令，并给出执行界面的截图（命令输入界面和结果界面）。

1. 在GitHub上，通过Web界面建立一个Project，将不少于10个文件（程序代码、文档等）加入进去，形成初始分支B1；在B1基础上建立两个并行的分支B2、B3，手工对B2和B3上的某些文件进行不同程度的修改并提交；

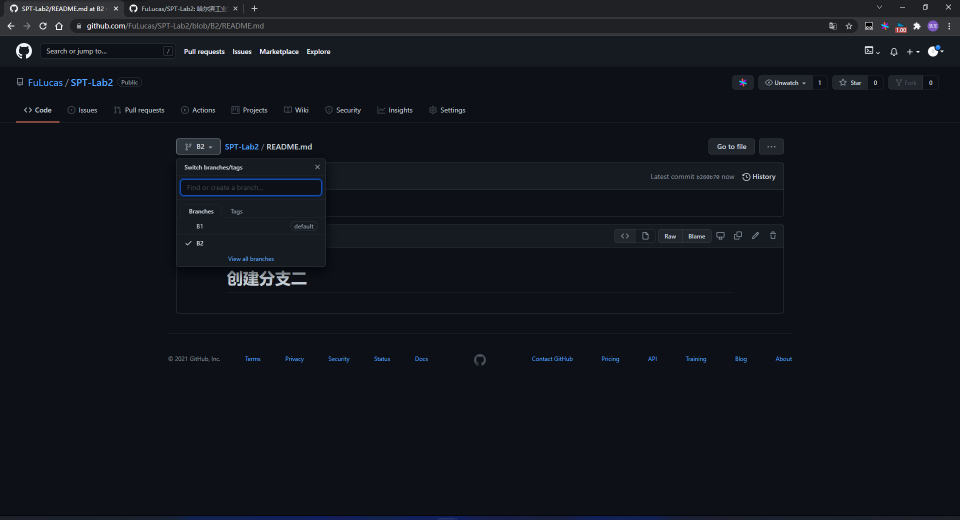
通过Web界面建立一个Project



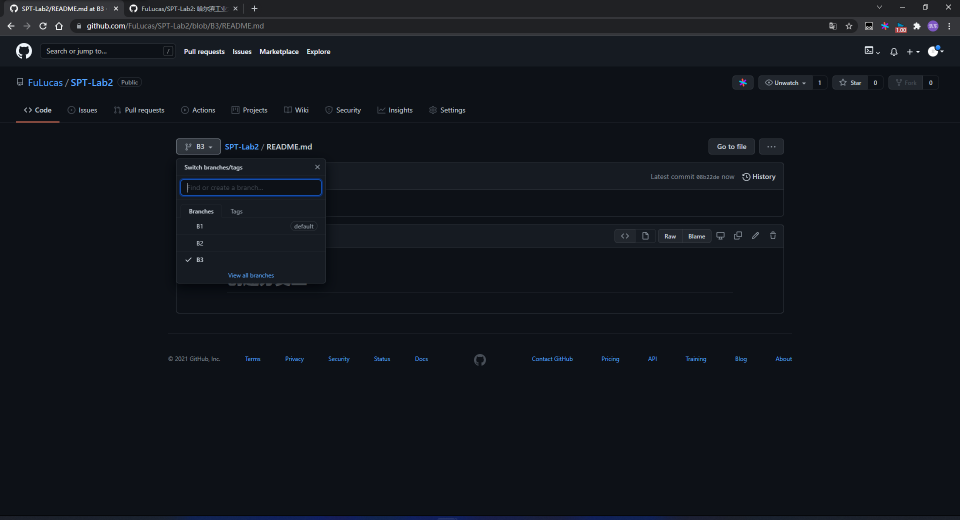
将不少于10个文件（程序代码、文档等）加入到初始分支B1

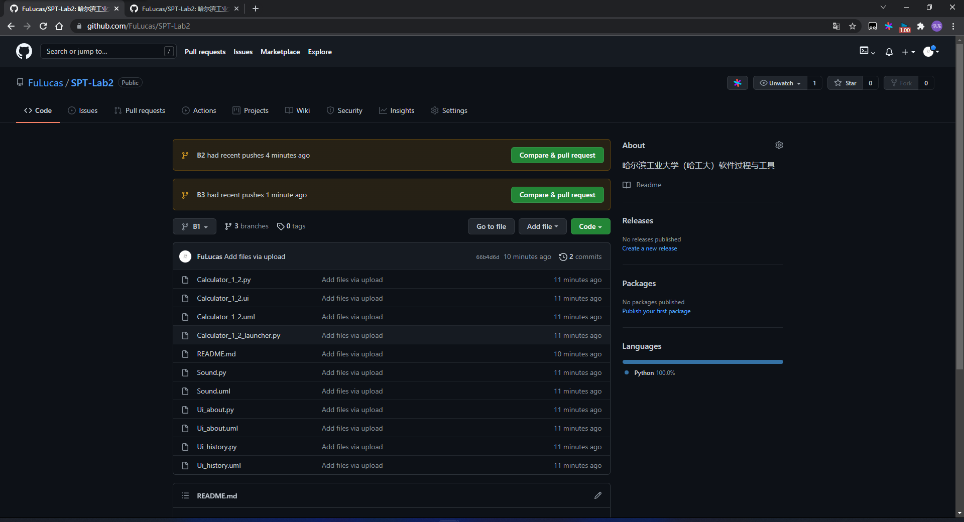


B1基础上分支B2，修改README.md文件



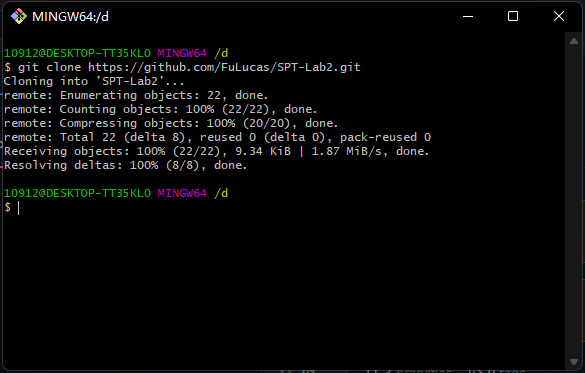
B1基础上分支B3，修改README.md文件





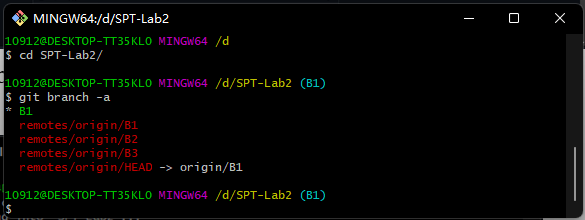
1. R8：从GitHub上(URL)克隆一个已有的Git仓库到本地；

克隆Git仓库到本地：git clone URL/SSH



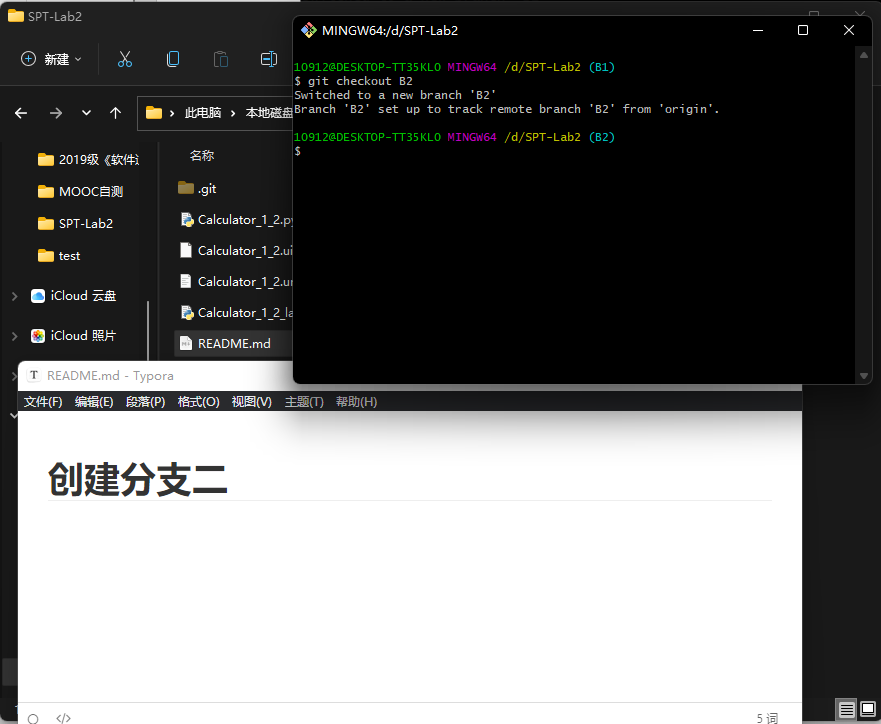
1. R9：获得该仓库的全部分支；

查看clone到本地的分支

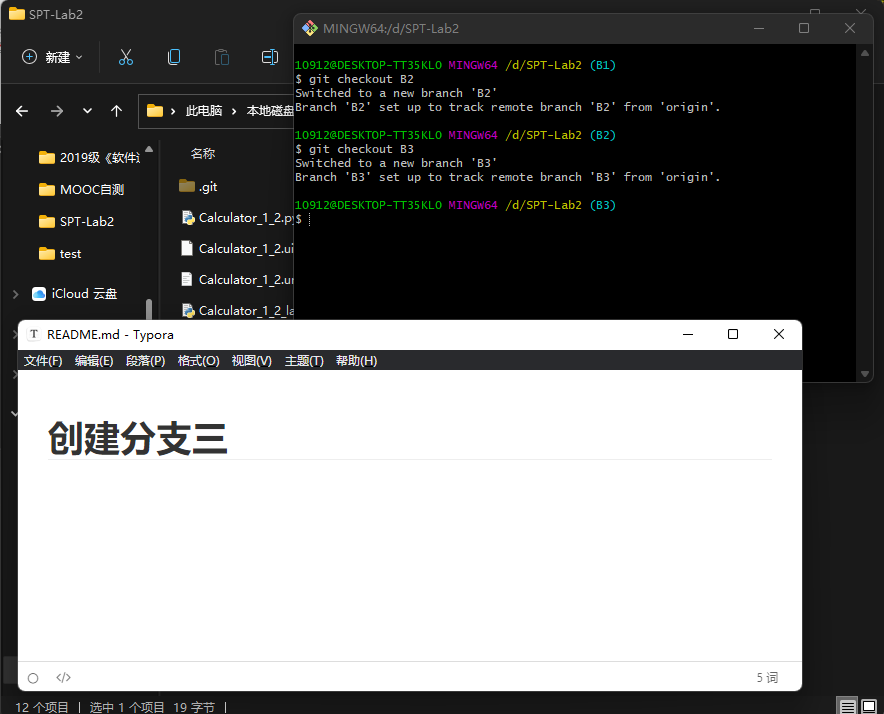


切换分支命令：git checkout B2/B3

查看分支B2

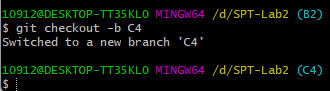


查看分支三



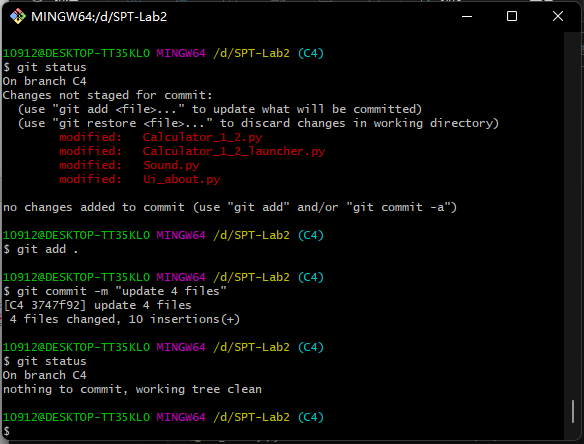
1. R10：在B2分支基础上创建一个新分支C4；

创建分支命令：git checkout -b C4

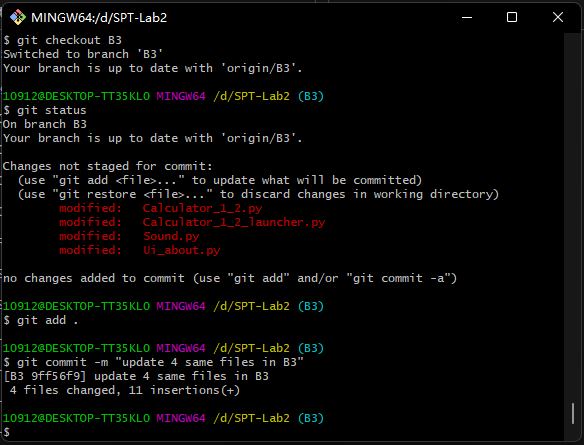


1. R11：在C4上，对4个文件进行修改并提交；

修改如下四个文件并提交

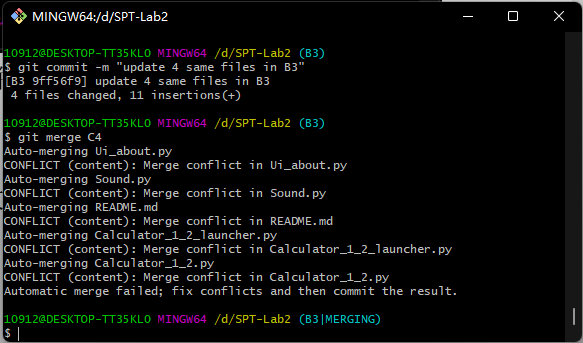


1. R12：在B3分支上对同样的4个文件做不同修改并提交；

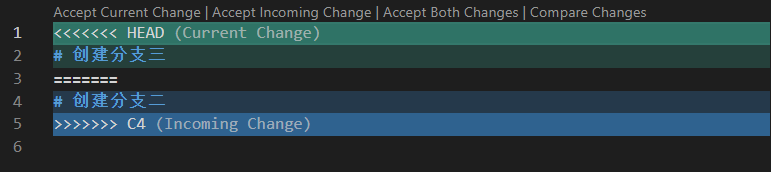


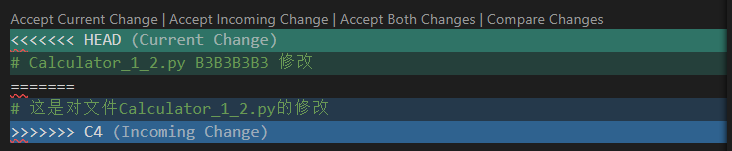
1. R13：将C4和B3分支合并，若有冲突，手工消解；

命令：git merge C3(when in B3)

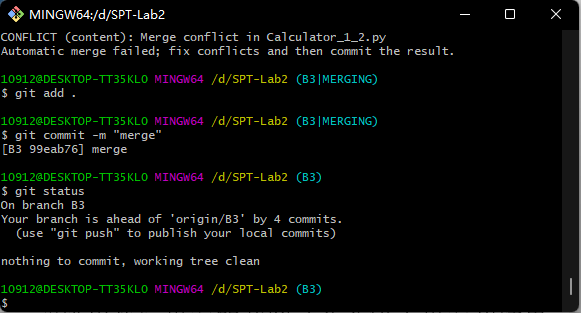


文件发生冲突，例如：



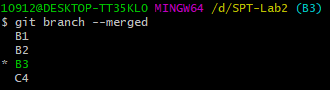


Accept Both Changes, then add, commit:

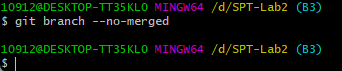


1. R14：查看目前哪些分支已经合并、哪些分支尚未合并；

已经合并分支命令：git branch –-merged

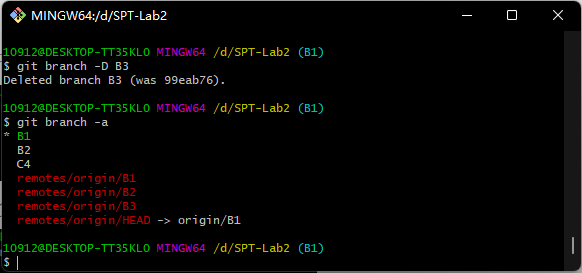


尚未合并分支命令：git branch –-no-merged



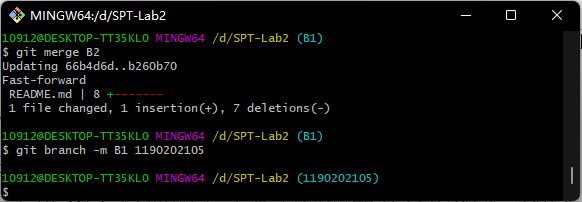
1. R15：将C4和B3合并后的分支删除，将尚未合并的分支合并到一个新分支上，分支名字为你的学号。

切换到分支B1删除合并后的分支B3：git branch -D B3



将B1和B2和并到新分支1190202105上：

（先合将B2并到到B1上，将B1改名为1190202105）

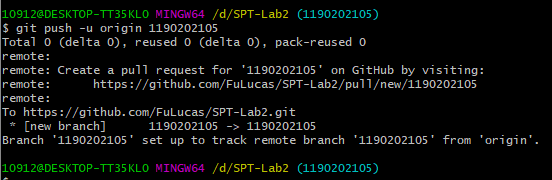


## 实验场景(3)：远程分支管理

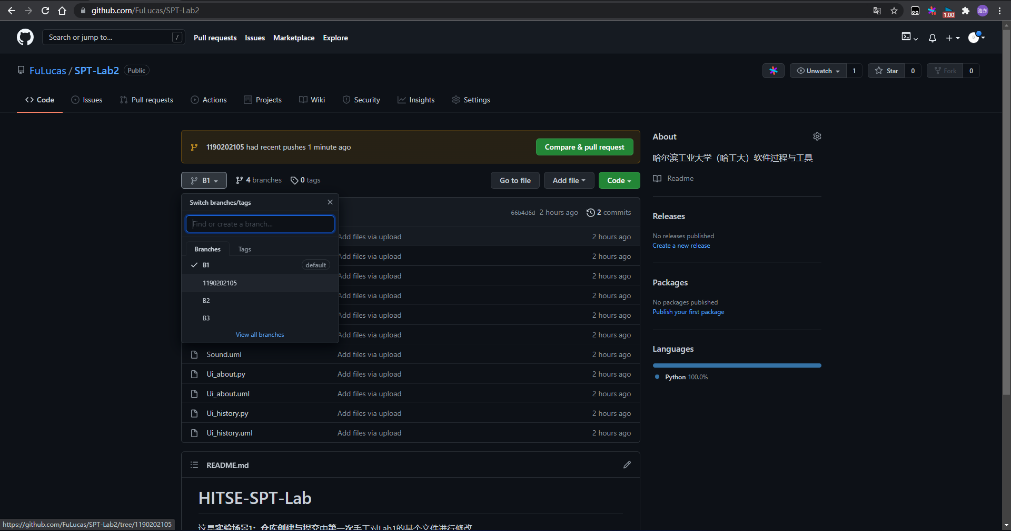
给出R16~R18的操作命令，并给出执行界面的截图（命令输入界面和结果界面）。

1. R16：将本地以你的学号命名的分支推送到GitHub上；

推送命令：git push -u origin 1190202105



Web演示：



1. R17：将R1到R7各步骤得到的结果推送到GitHub上；

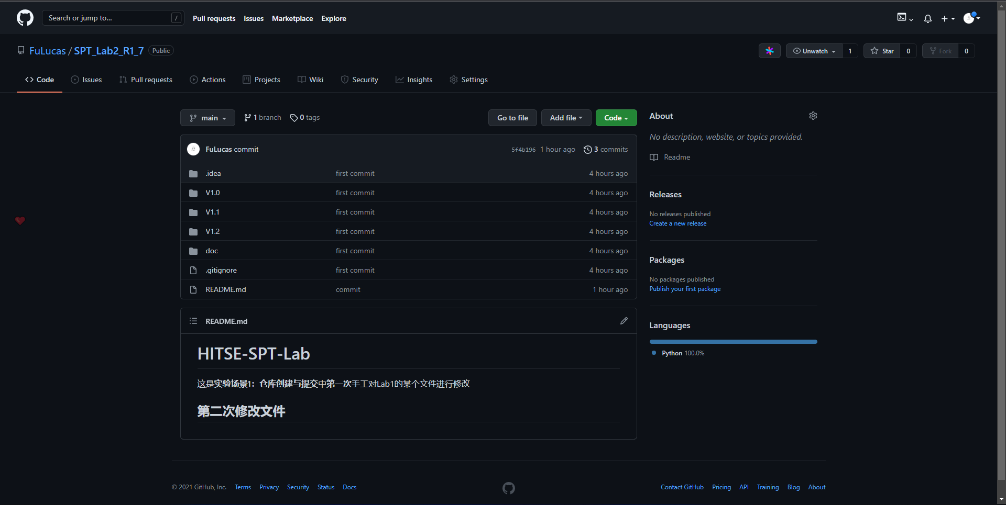
新建一个远程仓库按照GitHub提示：



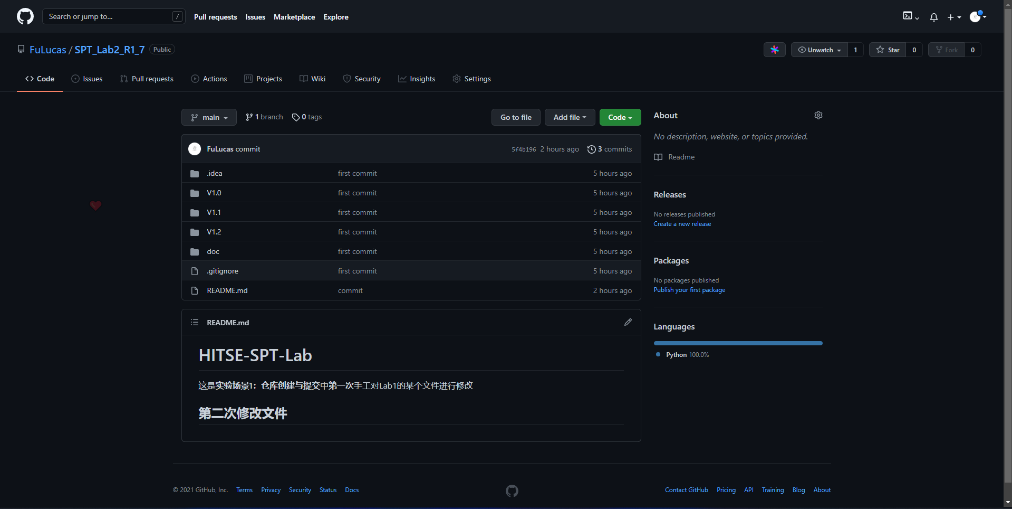
本地仓库与远程仓库关联：

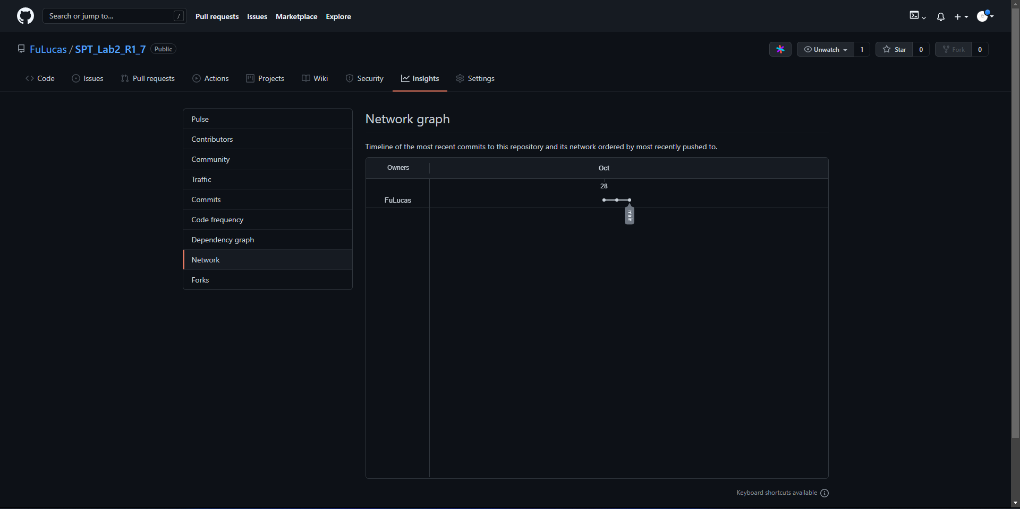


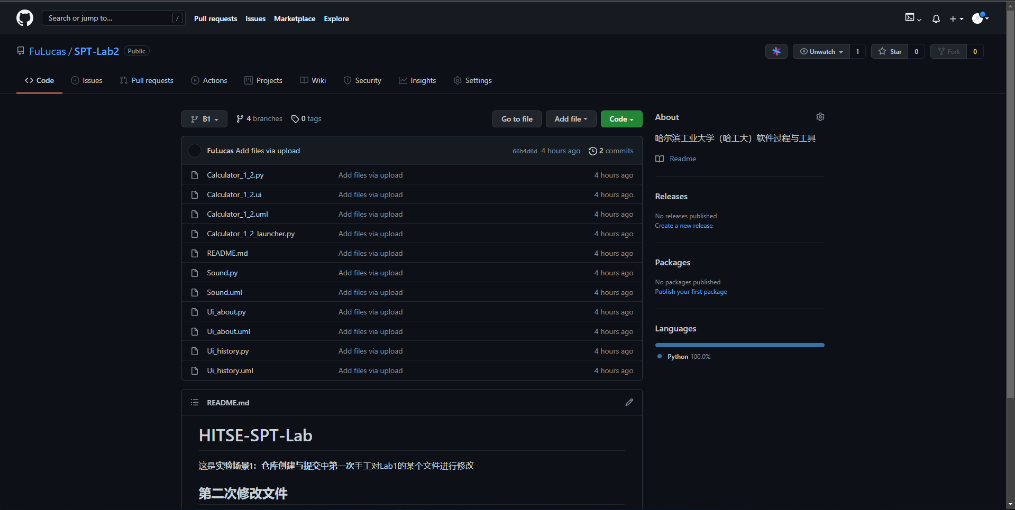
将R1到R7的结果推送到GitHub：

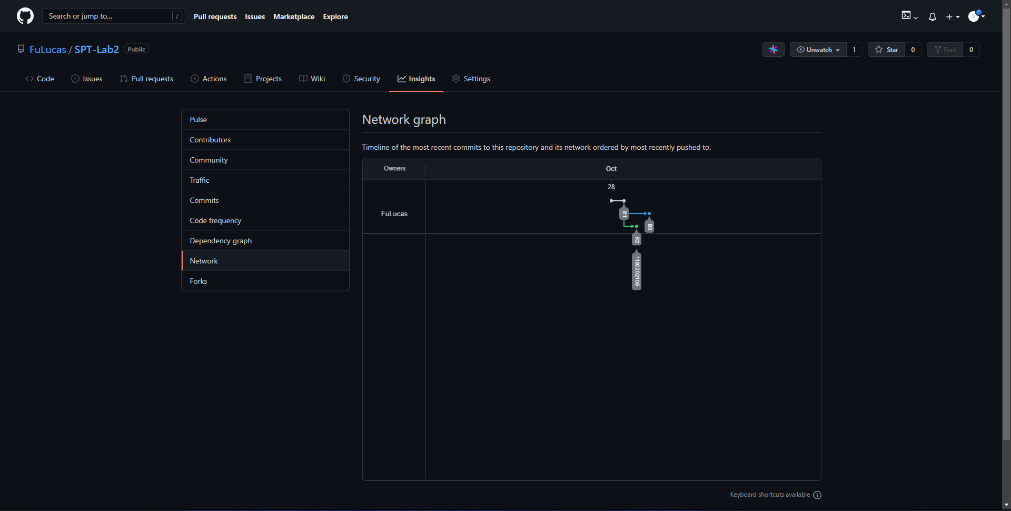


1. R18：在GitHub上以Web页面的方式查看你的两个仓库的当前状态。









# 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？

**性能**： Git 在版本控制系统方面提供最佳性能，提交、分支、合并都经过优化以获得比其他系统更好的性能。

**分布式**： Git 本质上是分布式的，分布式意味着存储库或完整的代码库被镜像到开发人员的系统上，以便多人能在它上面工作。

**暂存区**： Git 有一个称为“**索引**”或“**暂存区**”的中间阶段，可以在完成提交之前格式化和修改提交。

1. 在个人开发和团队开发中，Git起到的作用有何主要差异？

**个人开发中**，Git主要的作用是版本控制，便于个人进行迭代增量管理、创建分支、版本回退等，同时也能规范开发流程，避免产生差错。**团队开发中**，Git主要的作用是帮助团队协作，可以实现一人创建多人开发。实际项目开发中，通常都是以团队开发为主，为了维护线上主干代码的稳定，通常会采取创建分支-开发-测试-合并-上线的形式进行实际操作。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，开发完成后再讲修改后的文件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。同时每个工程师可以负责不同的模块，当代码合并时可以查看冲突，再来进行沟通，增加了团队合作的效率。

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： SVN，同Git相比的优缺点如下：

优点：可以**在服务端统一控制好访问权限，**易于管理，保证安全性；操作简单，易上手；管理方便，逻辑明确；

缺点： SVN是集中式管理系统，这就导致容易出现单点故障问题，即当服务器故障宕机，在故障期间用户无法提交代码，有丢失数据的风险。提交必须有网络连接（非本地存储的版本库）。**分支管理不灵活，SVN分支是一个完整的目录，且这个目录拥有完整的实际文件（代价昂贵），这些操作都是在服务端进行同步的。**

不同版本控制系统的差别：

首先，他们最大的差别就是：**Git是分布式的而SVN是集中式的**。

Git把内容按元数据方式存储（分支廉价），而SVN是按文件（分支昂贵）；

Git概念相对复杂，命令多，而SVN则就要更加简单易于上手；

Git没有一个全局的版本号，而SVN有；

Git的每一个终端都相当于一个仓库，用户每次是克隆仓库的镜像。

1. 在什么情况下适合使用Git、什么情况下没必要使用Git？

在代码量较小或者实现功能比较简单的时候没必要使用Git，Git的版本控制功能在小量代码工作中显得复杂且没有必要。开发人员少也没有必要使用Git，因为Git分布式版本管理目的就在于协调团队中各种代码的冲突。

相对的，当项目复杂代码量大，需要多个增量等，以及开发团队庞大时Git就适合用Git。

1. 其他相关思考。

**对非线性开发的有力支持**

Git 支持快速分支和合并，并包括用于可视化和导航非线性开发历史的特定工具。在 Git 中，一个核心假设是更改被合并的频率比写入的要多，因为它会传递给不同的审阅者。

**与现有系统和协议的兼容性**

存储库可以通过HTTP、FTP或 Git 协议通过普通套接字或ssh发布。Git 还有一个 CVS 服务器模拟，它允许使用现有的 CVS 客户端和 IDE 插件来访问 Git 存储库。

**历史的密码认证**

Git 历史的存储方式使得特定版本的 ID取决于导致该提交的完整开发历史。一旦发布，就不可能在不被注意的情况下更改旧版本。

**基于工具包的设计**

Git 被设计为一组用C 语言编写的程序和几个为这些程序提供包装器的 shell 脚本。尽管为了速度和可移植性，这些脚本中的大多数已经用 C 重写了，但设计仍然存在，并且很容易将组件链接在一起。

**可插拔合并策略**

作为工具包设计的一部分，Git 有一个明确定义的不完全合并模型，它有多种算法来完成它，最终告诉用户它无法自动完成合并，需要手动编辑。

**垃圾堆积直到收集**

中止操作或撤销更改将在数据库中留下无用的悬空对象。这些通常只是通缉对象不断增长的历史的一小部分。当存储库中创建了足够多的松散对象时，Git 将自动执行垃圾收集。可以使用显式调用垃圾收集git gc。