

哈尔滨工业大学 计算学部

2024 年秋季学期《开源软件开发实践》

Lab 1: Git 实战

姓名	学号	联系方式
岳翼博	2022210939	19716318612

目 录

1 实验要求.....	1
2 安装 Git	1
2.1 本地机器上安装 git	1
2.2 申请 github 帐号	1
3 Git 操作过程	3
3.1 实验场景(1): 仓库创建与提交	3
3.2 实验场景(2): 分支管理	8
3.3 实验场景(3): 在线 Git 练习	16
4 小结	21

1 实验要求

- (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 GitHub 管理自己的项目源代码。

2 安装 Git

2.1 本地机器上安装 git

Git 版本号:

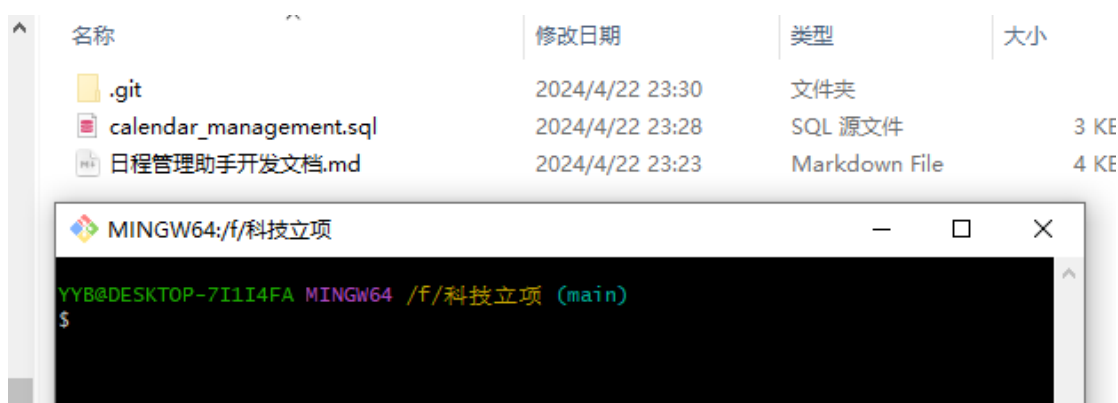


```
C:\Windows\system32\cmd.exe
Microsoft Windows [版本 10.0.19045.4291]
(c) Microsoft Corporation。保留所有权利。

C:\Users\YYB>git --version
git version 2.43.0.windows.1

C:\Users\YYB>
```

Git 运行界面:



2.2 申请 github 帐号

账号名称: obiyeuy

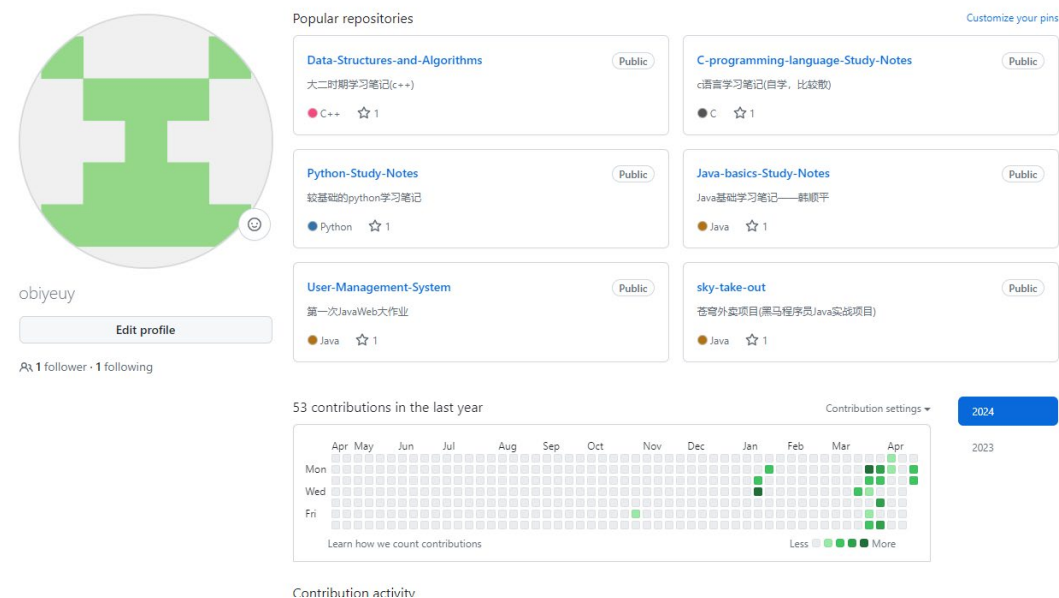
URL 地址:

1) 软件过程与项目管理笔记仓库:

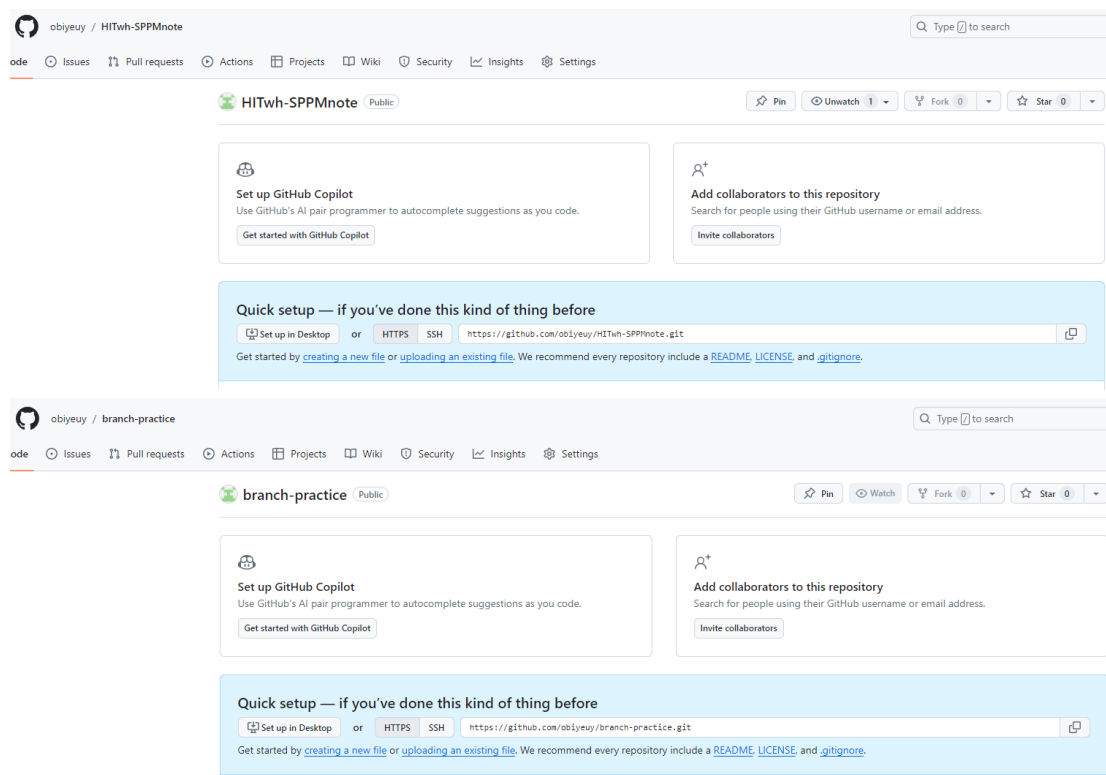
<https://github.com/obiyeuy/HITwh-SPPMnote.git>

2) 分支操作练习仓库: <https://github.com/obiyeuy/branch-practice.git>

账号信息截图:



项目信息截图:



3 Git 操作过程

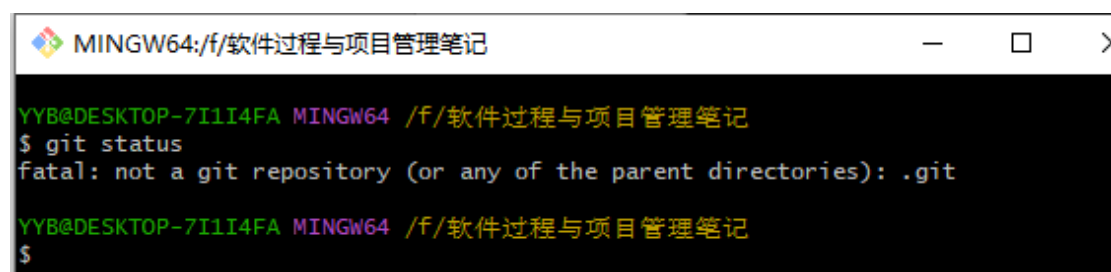
3.1 实验场景(1): 仓库创建与提交

(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

指令: `git status`

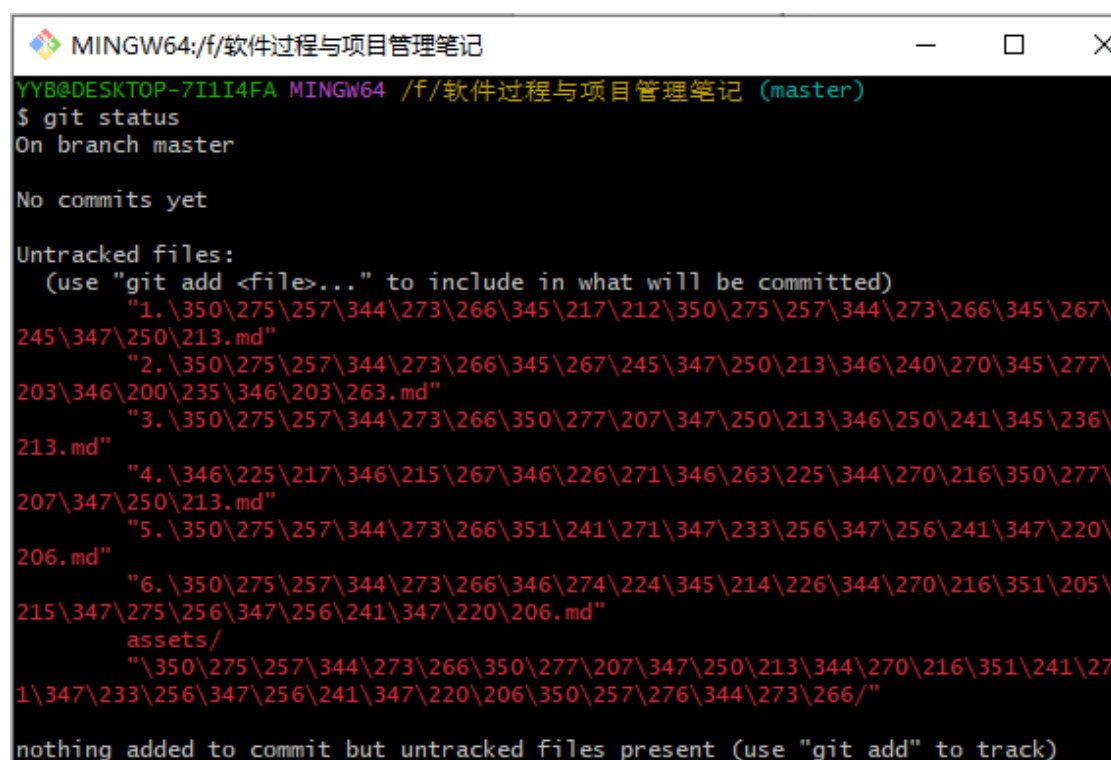
说明: 这个指令在未进行本地仓库初始化前是无法生效的, 下面分别是本人在初始化前和初始化后执行该指令得到的结果。

初始化前:



```
MINGW64:/f/软件过程与项目管理笔记
YYB@DESKTOP-7I1I4FA MINGW64 /f/软件过程与项目管理笔记
$ git status
fatal: not a git repository (or any of the parent directories): .git
YYB@DESKTOP-7I1I4FA MINGW64 /f/软件过程与项目管理笔记
$
```

初始化后:



```
MINGW64:/f/软件过程与项目管理笔记
YYB@DESKTOP-7I1I4FA MINGW64 /f/软件过程与项目管理笔记 (master)
$ git status
On branch master

No commits yet

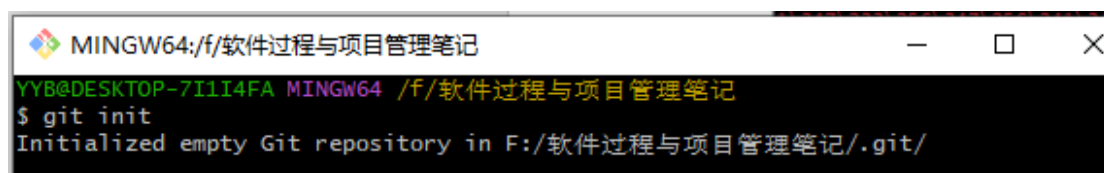
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    "1. \350\275\257\344\273\266\345\217\212\350\275\257\344\273\266\345\267\245\347\250\213.md"
    "2. \350\275\257\344\273\266\345\267\245\347\250\213\346\240\270\345\277\203\346\200\235\346\203\263.md"
    "3. \350\275\257\344\273\266\350\277\207\347\250\213\346\250\241\345\236\213.md"
    "4. \346\225\217\346\215\267\346\226\271\346\263\225\344\270\216\350\277\207\347\250\213.md"
    "5. \350\275\257\344\273\266\351\241\271\347\233\256\347\256\241\347\220\206.md"
    "6. \350\275\257\344\273\266\346\274\224\345\214\226\344\270\216\351\205\215\347\275\256\347\256\241\347\220\206.md"
    assets/
    "\350\275\257\344\273\266\350\277\207\347\250\213\344\270\216\351\241\271\347\233\256\347\256\241\347\220\206\350\257\276\344\273\266/"

nothing added to commit but untracked files present (use "git add" to track)
```

(2) R1: 本地初始化一个 Git 仓库, 将自己在某个项目中的全部源文件加入进去, 纳入 Git 管理

指令: `git init`

执行命令:



```

MINGW64:/f/软件过程与项目管理笔记
YYB@DESKTOP-7I1I4FA MINGW64 /f/软件过程与项目管理笔记
$ git init
Initialized empty Git repository in F:/软件过程与项目管理笔记/.git/

```

执行结果：在项目文件夹下生成一个名为.git 的文件夹

此电脑 > 本地磁盘 (F:) > 软件过程与项目管理笔记

名称	修改日期	类型	大小
.git	2024/4/23 22:47	文件夹	
assets	2024/4/23 22:41	文件夹	
软件过程与项目管理课件	2024/4/23 22:41	文件夹	
1.软件及软件工程.md	2024/4/20 20:22	Markdown File	8 KB
2.软件工程核心思想.md	2024/4/20 20:43	Markdown File	3 KB
3.软件过程模型.md	2024/4/20 20:52	Markdown File	8 KB
4.敏捷方法与过程.md	2024/4/20 21:20	Markdown File	4 KB
5.软件项目管理.md	2024/4/20 21:23	Markdown File	11 KB
6.软件演化与配置管理.md	2024/4/20 20:03	Markdown File	4 KB

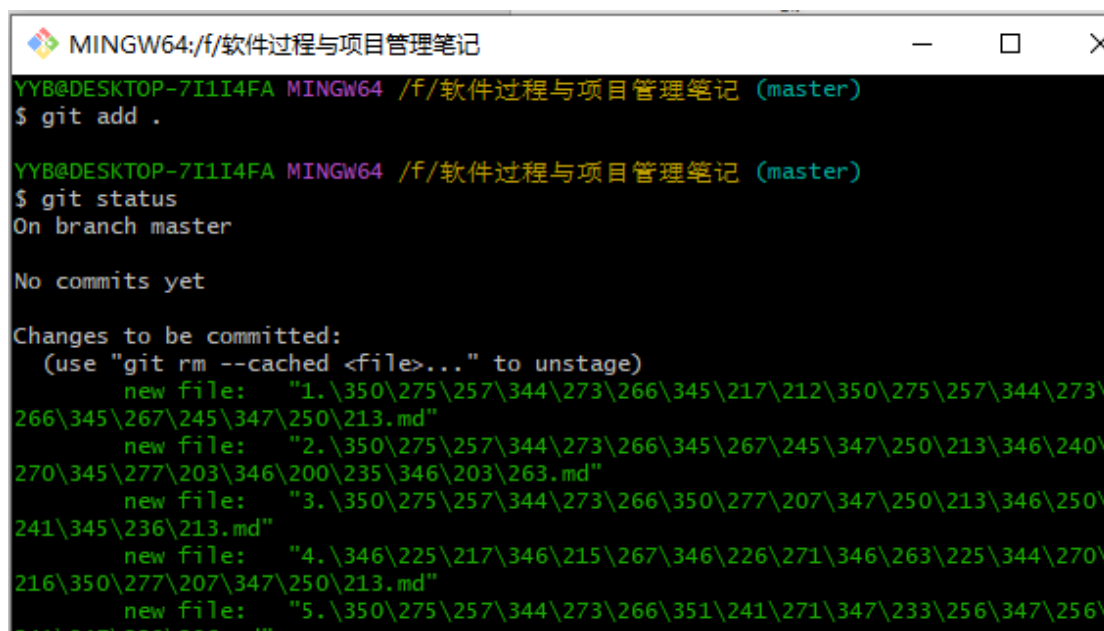
(3) R2: 提交(需要先将文件加入暂存区, 用 `git add` 命令)

指令:

将文件加入暂存区: `git add` 文件名

将暂存区的文件提交到本地仓库: `git commit -m "此次提交的备注信息"`

加入暂存区执行结果:



```

MINGW64:/f/软件过程与项目管理笔记
YYB@DESKTOP-7I1I4FA MINGW64 /f/软件过程与项目管理笔记 (master)
$ git add .

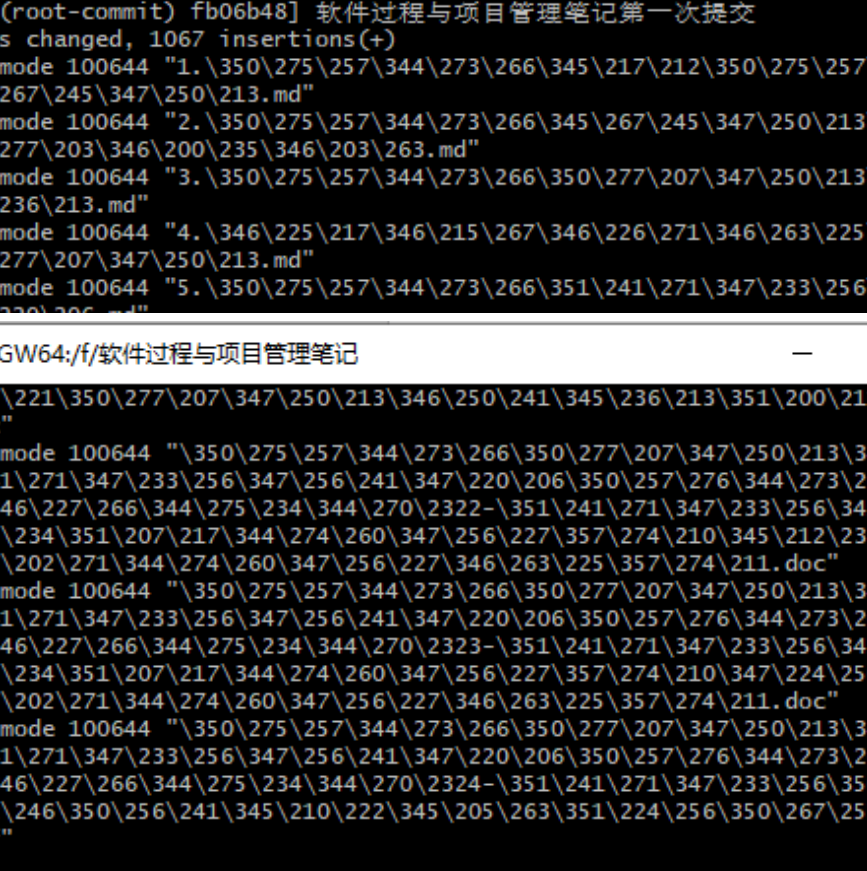
YYB@DESKTOP-7I1I4FA MINGW64 /f/软件过程与项目管理笔记 (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   "1.\350\275\257\344\273\266\345\217\212\350\275\257\344\273\266\345\267\245\347\250\213.md"
    new file:   "2.\350\275\257\344\273\266\345\267\245\347\250\213\346\240\270\345\277\203\346\200\235\346\203\263.md"
    new file:   "3.\350\275\257\344\273\266\350\277\207\347\250\213\346\250\241\345\236\213.md"
    new file:   "4.\346\225\217\346\215\267\346\226\271\346\263\225\344\270\216\350\277\207\347\250\213.md"
    new file:   "5.\350\275\257\344\273\266\351\241\271\347\233\256\347\256\344\247\228\206.md"

```

提交执行结果:



The screenshot shows a Windows terminal window with the title "MINGW64:/f/软件过程与项目管理笔记". The terminal displays the following commands and output:

```

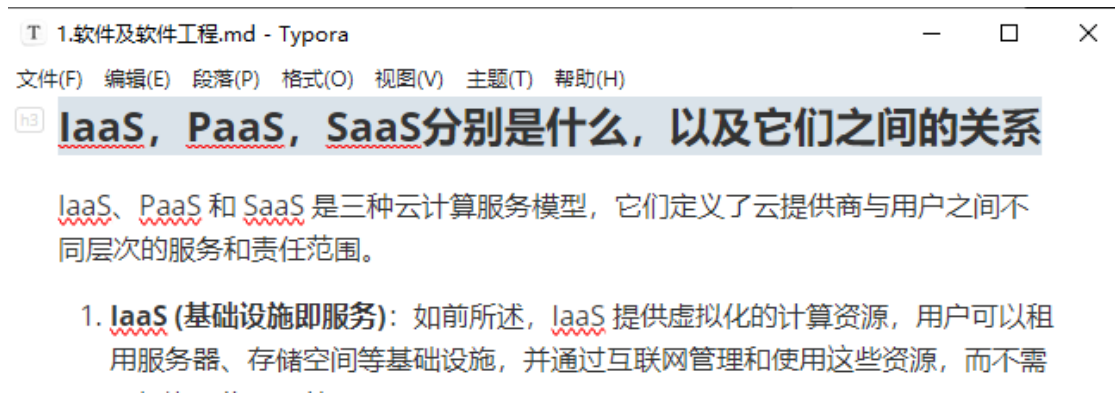
YYB@DESKTOP-7I1I4FA MINGW64 /f/软件过程与项目管理笔记 (master)
$ git commit -m "软件过程与项目管理笔记第一次提交"
[master (root-commit) fb06b48] 软件过程与项目管理笔记第一次提交
45 files changed, 1067 insertions(+)
 create mode 100644 "1.\350\275\257\344\273\266\345\217\212\350\275\257\344\273\266\345\267\245\347\250\213.md"
 create mode 100644 "2.\350\275\257\344\273\266\345\267\245\347\250\213\346\240\270\345\277\203\346\200\235\346\203\263.md"
 create mode 100644 "3.\350\275\257\344\273\266\350\277\207\347\250\213\346\250\241\345\236\213.md"
 create mode 100644 "4.\346\225\217\346\215\267\346\226\271\346\263\225\344\270\216\350\277\207\347\250\213.md"
 create mode 100644 "5.\350\275\257\344\273\266\351\241\271\347\233\256\347\256\344\273\266\345\217\212\350\275\257\344\273\266\345\267\245\347\250\213\346\240\270\345\277\203\346\200\235\346\203\263.md"
 create mode 100644 "\345\217\221\350\277\207\347\250\213\346\250\241\345\236\213\351\200\211\345\236\213.doc"
 create mode 100644 "\350\275\257\344\273\266\350\277\207\347\250\213\344\270\216\351\241\271\347\233\256\347\256\241\347\220\206\350\257\276\344\273\266\345\271\263\346\227\266\344\275\234\344\270\2322-\351\241\271\347\233\256\345\267\245\344\275\234\351\207\217\344\274\260\347\256\227\357\274\210\345\212\237\350\203\275\347\202\271\344\274\260\347\256\227\346\263\225\357\274\211.doc"
 create mode 100644 "\350\275\257\344\273\266\350\277\207\347\250\213\344\270\216\351\241\271\347\233\256\347\256\241\347\220\206\350\257\276\344\273\266\345\271\263\346\227\266\344\275\234\344\270\2323-\351\241\271\347\233\256\345\267\245\344\275\234\351\207\217\344\274\260\347\256\227\357\274\210\347\224\250\344\276\213\347\202\271\344\274\260\347\256\227\346\263\225\357\274\211.doc"
 create mode 100644 "\350\275\257\344\273\266\350\277\207\347\250\213\344\270\216\351\241\271\347\233\256\347\256\241\347\220\206\350\257\276\344\273\266\345\271\263\346\227\266\344\275\234\344\270\2324-\351\241\271\347\233\256\350\277\233\345\272\246\350\256\241\345\210\222\345\205\263\351\224\256\350\267\257\345\276\204.pdf"

YYB@DESKTOP-7I1I4FA MINGW64 /f/软件过程与项目管理笔记 (master)
$ git status
On branch master
nothing to commit, working tree clean

```

(4) **R3**: 手工对笔记中的某个文件进行修改, 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)

1) 首先对文件进行修改,我删除了“1.软件及软件工程.md”文件中的一行文字(IaaS, PaaS, SaaS 分别是什么,以及它们之间的关系):



2) 查看文件修改情况: `git status`


```
YYB@DESKTOP-7I1I4FA MINGW64 /f/软件过程与项目管理笔记 (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   "1.\350\275\257\344\273\266\345\217\212\350\275\257\344\273\266\345\267\245\347\250\213.md"

no changes added to commit (use "git add" and/or "git commit -a")
```

可以看见“1....md”文件被标红了(此处中文被转码了,我的项目中只有“1.软件及软件工程.md”符合图片),可以看出 git 已经发现此文件已被修改

3) 查看具体变动情况: git diff

```
YYB@DESKTOP-7I1I4FA MINGW64 /f/软件过程与项目管理笔记 (master)
$ git diff
diff --git "a/1.\350\275\257\344\273\266\345\217\212\350\275\257\344\273\266\345\267\245\347\250\213.md" "b/1.\350\275\257\344\273\266\345\217\212\350\275\257\344\273\266\345\267\245\347\250\213.md"
index 5089336..d89bb0a 100644
--- "a/1.\350\275\257\344\273\266\345\217\212\350\275\257\344\273\266\345\267\245\347\250\213.md"
+++ "b/1.\350\275\257\344\273\266\345\217\212\350\275\257\344\273\266\345\267\245\347\250\213.md"
@@ -176,7 +176,7 @@

-### IaaS, PaaS, SaaS分别是什么，以及它们之间的关系
+### IaaS, PaaS, SaaS分别是什么，以及它们之间的关系

IaaS、PaaS 和 SaaS 是三种云计算服务模型，它们定义了云提供商与用户之间不同层次的服务和责任范围。
```

结果中出现了-### IaaS, PaaS, SaaS 分别是什么，以及它们之间的关系，git 发现文件此处被删除

(5) R4: 重新提交

这步指令与 R2 相同

将文件加入暂存区: git add 文件名

将暂存区的文件提交到本地仓库: git commit -m “此次提交的备注信息”

执行结果:



```
MINGW64:/f/软件过程与项目管理笔记
YYB@DESKTOP-7I1I4FA MINGW64 /f/软件过程与项目管理笔记 (master)
$ git add .

YYB@DESKTOP-7I1I4FA MINGW64 /f/软件过程与项目管理笔记 (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   "1.\350\275\257\344\273\266\345\217\212\350\275\257\344\273\266\345\267\245\347\250\213.md"
```

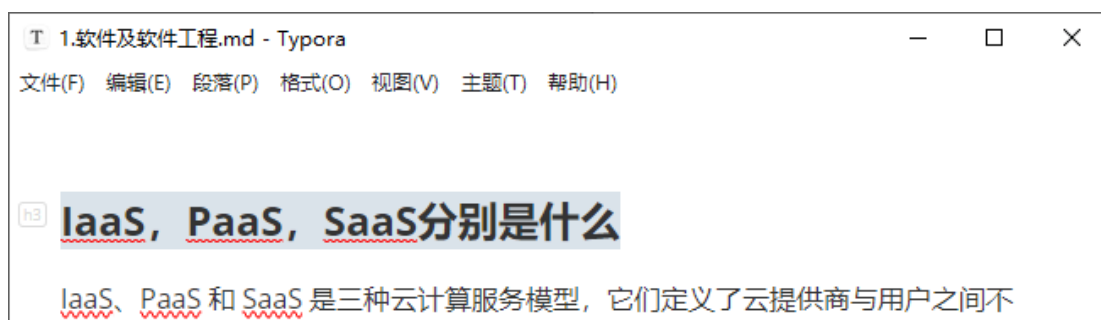


```
YYB@DESKTOP-7I1I4FA MINGW64 /f/软件过程与项目管理笔记 (master)
$ git commit -m "软件过程与项目管理笔记第二次提交"
[master 857d656] 软件过程与项目管理笔记第二次提交
1 file changed, 1 insertion(+), 1 deletion(-)

YYB@DESKTOP-7I1I4FA MINGW64 /f/软件过程与项目管理笔记 (master)
$ git status
On branch master
nothing to commit, working tree clean
```

(6) R5: 再次对项目文件中某个文件进行修改，重新提交

我修改文件仍是“1.软件及软件过程.md”，此次添加内容，为了与上面删除的区分，此次添加文字是“IaaS, PaaS, SaaS 分别是什么”



使用的指令仍然是: `git add 文件名` 和 `git commit -m “此次提交的备注信息”`，执行结果如下:

```
MINGW64:/f/软件过程与项目管理笔记

YYB@DESKTOP-7I1I4FA MINGW64 /f/软件过程与项目管理笔记 (master)
$ git add .

YYB@DESKTOP-7I1I4FA MINGW64 /f/软件过程与项目管理笔记 (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   "1.\350\275\257\344\273\266\345\217\212\350\275\257\344\273\266\345\267\245\347\250\213.md"

YYB@DESKTOP-7I1I4FA MINGW64 /f/软件过程与项目管理笔记 (master)
$ git commit -m "软件过程与项目管理笔记第三次提交"
[master 237da6d] 软件过程与项目管理笔记第三次提交
1 file changed, 2 insertions(+), 1 deletion(-)

YYB@DESKTOP-7I1I4FA MINGW64 /f/软件过程与项目管理笔记 (master)
$ git status
On branch master
nothing to commit, working tree clean
```

(7) R6: 把最后一次提交撤销

指令: `git reset --hard HEAD~`

首先使用 `git reflog` 查看历史的提交版本，执行结果如下:

```
YYB@DESKTOP-7I1I4FA MINGW64 /f/软件过程与项目管理笔记 (master)
$ git reflog
237da6d (HEAD -> master) HEAD@{0}: commit: 软件过程与项目管理笔记第三次提交
857d656 HEAD@{1}: commit: 软件过程与项目管理笔记第二次提交
fb06b48 HEAD@{2}: commit (initial): 软件过程与项目管理笔记第一次提交
```

可以看出包括初始化提交在内已经提交过三次，且当前的 HEAD 指向的是最后一次提交，则执行撤销提交指令执行结果如下：

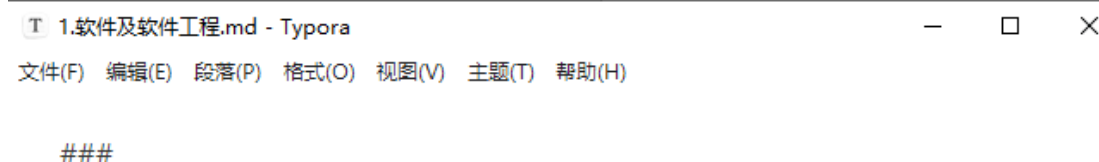
```
YYB@DESKTOP-7I1I4FA MINGW64 /f/软件过程与项目管理笔记 (master)
$ git reset --hard HEAD~
HEAD is now at 857d656 软件过程与项目管理笔记第二次提交
```

再次执行 `git reflog` 查看版本情况，特别注意版本的序列号：

```
YYB@DESKTOP-7I1I4FA MINGW64 /f/软件过程与项目管理笔记 (master)
$ git reflog
857d656 (HEAD -> master) HEAD@{0}: reset: moving to HEAD~
237da6d HEAD@{1}: commit: 软件过程与项目管理笔记第三次提交
857d656 (HEAD -> master) HEAD@{2}: commit: 软件过程与项目管理笔记第二次提交
fb06b48 HEAD@{3}: commit (initial): 软件过程与项目管理笔记第一次提交
```

可以看见撤销操作后版本的序列号从“237da6d”变成了“857d656”，这就是第二次提交的序列号，说明撤销操作成功。

查看对应文件，发现第二次修改的结果被撤回：



```
1.软件及软件工程.md - Typora
文件(F) 编辑(E) 段落(P) 格式(O) 视图(V) 主题(T) 帮助(H)

###

laaS、PaaS 和 SaaS 是三种云计算服务模型，它们定义了云提供商与用户之间不
###后的内容消失了，说明修改被撤销了。
```

(8) R7: 查询提交记录

指令：`git log`

执行结果：

```
YYB@DESKTOP-7I1I4FA MINGW64 /f/软件过程与项目管理笔记 (master)
$ git log
commit 857d656c56c00c4414e18a1785345aa17e28c045 (HEAD -> master)
Author: obiyeyu <yueyibohitwh@126.com>
Date:   Wed Apr 24 08:20:29 2024 +0800

    软件过程与项目管理笔记第二次提交

commit fb06b48661b918ea092c6a1f9a7d6d14f76bda9e
Author: obiyeyu <yueyibohitwh@126.com>
Date:   Tue Apr 23 22:54:53 2024 +0800

    软件过程与项目管理笔记第一次提交
```

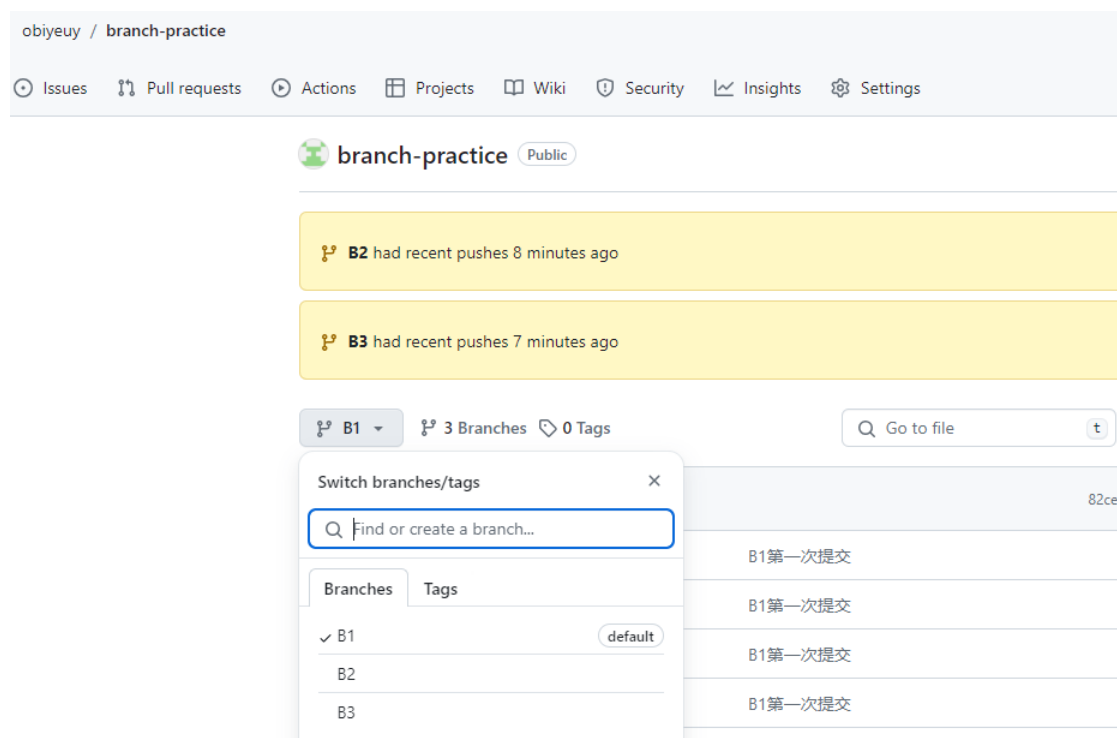
此处看出 `git log` 并不会显示撤销前的提交记录，只会显示结果到当前 HEAD 指针指向的版本，以及该版本之前所有的提交记录。

3.2 实验场景(2)：分支管理

当前准备：在 Github 中建立了一个 Project，其中有三个分支，主分支中有 21 个 java 文件，名为 B1；在此基础上建立了并行的两个分支 B2、B3，并对 B2

和 B3 中的文件进行了不同的修改(其中 B2 中 “CategoryDTO.java” 的文件相较于 B1 中的少了注释; B3 中 “CategoryPageQueryDTO.java” 文件相较于 B1 中的少了注释)

准备情况如下:



(1) R1: 从 Gitee/GitLab/Github 上克隆一个已有的 Git 仓库到本地

指令: `git clone URL`

执行结果:

```
MINGW64:/f/Git分支练习

YYB@DESKTOP-7I1I4FA MINGW64 /f/Git分支练习
$ git init
Initialized empty Git repository in F:/Git分支练习/.git/

YYB@DESKTOP-7I1I4FA MINGW64 /f/Git分支练习 (master)
$ git clone https://github.com/obiyeuy/branch-practice.git
Cloning into 'branch-practice'...
remote: Enumerating objects: 29, done.
remote: Counting objects: 100% (29/29), done.
remote: Compressing objects: 100% (18/18), done.
remote: Total 29 (delta 11), reused 29 (delta 11), pack-reused 0
Receiving objects: 100% (29/29), 5.74 KiB | 225.00 KiB/s, done.
Resolving deltas: 100% (11/11), done.
```

执行后本地文件夹中出现所有上传到远程的文件(图中为部分文件):

此电脑 > 本地磁盘 (F:) > Git分支练习 > branch-practice

名称	修改日期	类型	大小
CategoryDTO.java	2024/4/24 17:45	Java 源文件	1 KB
CategoryPageQueryDTO.java	2024/4/24 17:45	Java 源文件	1 KB
DataOverViewQueryDTO.java	2024/4/24 17:45	Java 源文件	1 KB
DishDTO.java	2024/4/24 17:45	Java 源文件	1 KB
DishPageQueryDTO.java	2024/4/24 17:45	Java 源文件	1 KB
EmployeeDTO.java	2024/4/24 17:45	Java 源文件	1 KB
EmployeeLoginDTO.java	2024/4/24 17:45	Java 源文件	1 KB
EmployeePageQueryDTO.java	2024/4/24 17:45	Java 源文件	1 KB
GoodsSalesDTO.java	2024/4/24 17:45	Java 源文件	1 KB
OrdersCancelDTO.java	2024/4/24 17:45	Java 源文件	1 KB

(2) R2: 获得该仓库的全部分支，并创建分支 B1,B2

指令: `git branch -a`

执行结果如下:

```
MINGW64:/f/Git分支练习/branch-practice

YYB@DESKTOP-7I1I4FA MINGW64 /f/Git分支练习/branch-practice (B1)
$ git branch -a
* B1
remotes/origin/B1
remotes/origin/B2
remotes/origin/B3
remotes/origin/HEAD -> origin/B1
```

可以看出当前分支是本地的 B1(master)分支，但是远程的 B2、B3 分支都可以在本地被查询到。

(3) R3: 在 B2 分支基础上创建一个新分支 C4

1)先切换到 B2 分支

指令: `git checkout B2`

```
YYB@DESKTOP-7I1I4FA MINGW64 /f/Git分支练习/branch-practice (B1)
$ git checkout B2
Switched to a new branch 'B2'
branch 'B2' set up to track 'origin/B2'.
```

2)创建 C4 分支并切换到 C4 分支

指令: `git branch C4` `git checkout C4`

```
YYB@DESKTOP-7I1I4FA MINGW64 /f/Git分支练习/branch-practice (B2)
$ git branch C4

YYB@DESKTOP-7I1I4FA MINGW64 /f/Git分支练习/branch-practice (B2)
$ git checkout C4
Switched to branch 'C4'

YYB@DESKTOP-7I1I4FA MINGW64 /f/Git分支练习/branch-practice (C4)
$
```

(4) R4: 在 C4 上，对 2 个文件进行修改并提交

1)进行文件的修改

为了方便检查，我直接按照 Windows 资源管理器的默认字典序对最后两个

文件进行修改。修改的方式是简单的在文本最后添加一行注释(//C4 分支所添加的注释)。

此电脑 > 本地磁盘 (F:) > Git分支练习 > branch-practice

名称	修改日期	类型	大小
OrdersSubmitDTO.java	2024/4/24 17:45	Java 源文件	1 KB
PasswordEditDTO.java	2024/4/24 17:45	Java 源文件	1 KB
SetmealDTO.java	2024/4/24 17:45	Java 源文件	1 KB
SetmealPageQueryDTO.java	2024/4/24 17:45	Java 源文件	1 KB
ShoppingCartDTO.java	2024/4/24 17:45	Java 源文件	1 KB
UserLoginDTO.java	2024/4/24 17:45	Java 源文件	1 KB

修改的结果使用 `git status` 指令查看状态:

```
YYB@DESKTOP-7I1I4FA MINGW64 /f/Git分支练习/branch-practice (C4)
$ git status
On branch C4
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   ShoppingCartDTO.java
        modified:   UserLoginDTO.java

no changes added to commit (use "git add" and/or "git commit -a")
```

说明修改成功。

2)提交

对于提交过程,使用的指令仍然是: `git add 文件名` 和 `git commit -m` “此次提交的备注信息”, 执行指令后结果如下:

```
YYB@DESKTOP-7I1I4FA MINGW64 /f/Git分支练习/branch-practice (C4)
$ git add .

YYB@DESKTOP-7I1I4FA MINGW64 /f/Git分支练习/branch-practice (C4)
$ git commit -m "C4第一次提交"
[C4 172a04f] C4第一次提交
2 files changed, 2 insertions(+)

YYB@DESKTOP-7I1I4FA MINGW64 /f/Git分支练习/branch-practice (C4)
$ git status
On branch C4
nothing to commit, working tree clean
```

(5) R5: 在 B3 分支上对同样的 2 个文件做出不同修改并提交

1)先换分支到 B3

指令: `git checkout B3`

执行结果:

```
YYB@DESKTOP-7I1I4FA MINGW64 /f/Git分支练习/branch-practice (C4)
$ git checkout B3
Switched to a new branch 'B3'
branch 'B3' set up to track 'origin/B3'.

YYB@DESKTOP-7I1I4FA MINGW64 /f/Git分支练习/branch-practice (B3)
$
```

2)修改文件

此次仍然在文本末添加注释(//B3 分支所添加的注释), 结果如下:

```
YYB@DESKTOP-7I1I4FA MINGW64 /f/Git分支练习/branch-practice (B3)
$ git status
On branch B3
Your branch is up to date with 'origin/B3'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   ShoppingCartDTO.java
        modified:   UserLoginDTO.java

no changes added to commit (use "git add" and/or "git commit -a")
```

3)提交

使用的指令仍然是: `git add 文件名` 和 `git commit -m “此次提交的备注信息”`, 执行指令后结果如下:

```
YYB@DESKTOP-7I1I4FA MINGW64 /f/Git分支练习/branch-practice (B3)
$ git add .

YYB@DESKTOP-7I1I4FA MINGW64 /f/Git分支练习/branch-practice (B3)
$ git commit -m "B3第二次提交"
[B3 44f8f81] B3第二次提交
2 files changed, 2 insertions(+)

YYB@DESKTOP-7I1I4FA MINGW64 /f/Git分支练习/branch-practice (B3)
$ git status
On branch B3
Your branch is ahead of 'origin/B3' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```

(6) R6: 将 C4 和 B3 分支合并, 若有冲突, 手工消解

合并分支指令: `git merge 分支名`

这个命令是将指定的分支名合并到当前所在分支上, 我在这里是将 C4 分支合并到 B3 分支, 因此使用的指令是 `git merge C4`。执行后结果如下:

```
YYB@DESKTOP-7I1I4FA MINGW64 /f/Git分支练习/branch-practice (B3)
$ git merge C4
Auto-merging ShoppingCartDTO.java
CONFLICT (content): Merge conflict in ShoppingCartDTO.java
Auto-merging UserLoginDTO.java
CONFLICT (content): Merge conflict in UserLoginDTO.java
Automatic merge failed; fix conflicts and then commit the result.

YYB@DESKTOP-7I1I4FA MINGW64 /f/Git分支练习/branch-practice (B3|MERGING)
$
```

提示有两个文件有冲突产生, 下面进入文件进行修改(此处用 UserLoginDTO.java 文件为例):

Git 已对此文件冲突部分进行了基本处理:


```

}
<<<<<<< HEAD
//B3分支所添加的注释
=====
//C4分支所添加的注释
>>>>>>> C4

```

处理后此处内容:

```

}
//B3分支所添加的注释
//C4分支所添加的注释
//手动处理了冲突合并的问题

```

对另一个文件进行相同处理，就完成了手动解决冲突。

处理完冲突后仍然需要进行提交:

```

YYB@DESKTOP-7I1I4FA MINGW64 /f/Git分支练习/branch-practice (B3|MERGING)
$ git add .

YYB@DESKTOP-7I1I4FA MINGW64 /f/Git分支练习/branch-practice (B3|MERGING)
$ git status
On branch B3
Your branch is ahead of 'origin/B3' by 1 commit.
  (use "git push" to publish your local commits)

All conflicts fixed but you are still merging.
  (use "git commit" to conclude merge)

Changes to be committed:
  modified:   CategoryDTO.java
  modified:   ShoppingCartDTO.java
  modified:   UserLoginDTO.java

YYB@DESKTOP-7I1I4FA MINGW64 /f/Git分支练习/branch-practice (B3|MERGING)
$ git commit -m "C4与B3合并至B3分支"
[B3 d7452eb] C4与B3合并至B3分支

YYB@DESKTOP-7I1I4FA MINGW64 /f/Git分支练习/branch-practice (B3)
$

```

可以看见命令行尾部的分支提示从“(B3|MERGING)”变成了“(B3)”，说明冲突解决，合并成功。

(7) R7: 在 B2 分支上对某个文件做出修改并提交

先换分支到 B2，然后修改文件并提交(同 R5)

指令: git checkout B2

(8) R8: 查看目前哪些分支已经合并、哪些分支尚未合并。

查看已经合并的分支: **git branch --merged**

查看尚未合并的分支: **git branch --no-merged**

执行结果如下:


```
YYB@DESKTOP-7I1I4FA MINGW64 /f/Git分支练习/branch-practice (B3)
$ git branch --merged
B1
B2
* B3
C4

YYB@DESKTOP-7I1I4FA MINGW64 /f/Git分支练习/branch-practice (B3)
$ git branch --no-merged
```

(8) R9: 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, 分支名字为学号

从 R14 中看出一共有 4 个分支。而要删除 B3 与 C4 合并的分支, 因此剩下需要合并的分支就是 B1 和 B2。

删除分支的指令: `git branch -d 分支名`

强制删除分支的指令: `git branch -D 分支名`

执行结果如下:

```
YYB@DESKTOP-7I1I4FA MINGW64 /f/Git分支练习/branch-practice (B1)
$ git branch -D B3
Deleted branch B3 (was d7452eb).
```

注意: 此处需要先切换到其他分支才能强制删除, 否则会报如下错:

```
YYB@DESKTOP-7I1I4FA MINGW64 /f/Git分支练习/branch-practice (B3)
$ git branch -D B3
error: cannot delete branch 'B3' used by worktree at 'F:/Git分支练习/branch-practice'
```

将 B2 分支合并到 B1(master)分支, 指令: `git merge 分支名`, 结果如下:

```
YYB@DESKTOP-7I1I4FA MINGW64 /f/Git分支练习/branch-practice (B1)
$ git merge B2
Updating 82cefb2..804de92
Fast-forward
 CategoryDTO.java | 8 ++++----
 1 file changed, 4 insertions(+), 4 deletions(-)
```

然后将当前分支改名为我的学号(2022210939)。

分支改名指令: `git branch -m 旧名称 新名称`

执行结果如下:

```
YYB@DESKTOP-7I1I4FA MINGW64 /f/Git分支练习/branch-practice (B1)
$ git branch -m B1 2022210939

YYB@DESKTOP-7I1I4FA MINGW64 /f/Git分支练习/branch-practice (2022210939)
$
```

命令行末尾变成了“(2022210939)”, 说明分支名修改成功。

(10) R10: 将本地以你的学号命名的分支推送到 Github 上

需要先将远程仓库取一个本地别名, 否则会比较麻烦。

指令: `git remote add origin 远程仓库 URL`

然后将以学号命名的分支推送到远程仓库

指令: `git push -u 远程仓库别名 推送分支名`

执行结果如下:

```
YYB@DESKTOP-7I1I4FA MINGW64 /f/Git分支练习/branch-practice (2022210939)
$ git push -u origin 2022210939
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for '2022210939' on GitHub by visiting:
remote:   https://github.com/obiyeuy/branch-practice/pull/new/2022210939
remote:
To https://github.com/obiyeuy/branch-practice.git
 * [new branch]      2022210939 -> 2022210939
branch '2022210939' set up to track 'origin/2022210939'.
```

(11) R11: 将 R1 到 R7 各步骤得到的结果推送到 Github 上

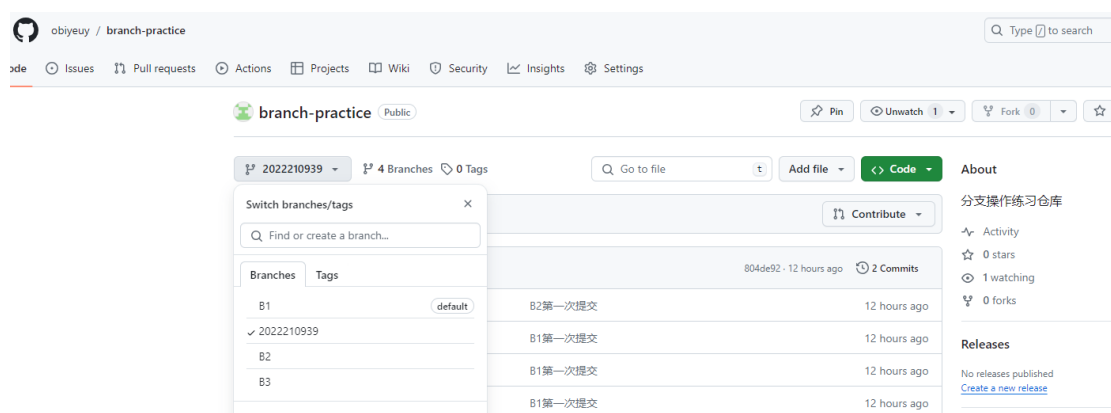
指令: `git push -u` 远程仓库别名 推送分支名

执行结果:

```
YYB@DESKTOP-7I1I4FA MINGW64 /f/软件过程与项目管理笔记 (master)
$ git push -u origin master
Enumerating objects: 51, done.
Counting objects: 100% (51/51), done.
Delta compression using up to 8 threads
Compressing objects: 100% (51/51), done.
Writing objects: 100% (51/51), 31.13 MiB | 25.85 MiB/s, done.
Total 51 (delta 4), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (4/4), done.
To https://github.com/obiyeuy/HITwh-SPPMnote.git
 * [new branch]      master -> master
branch 'master' set up to track 'origin/master'.
```

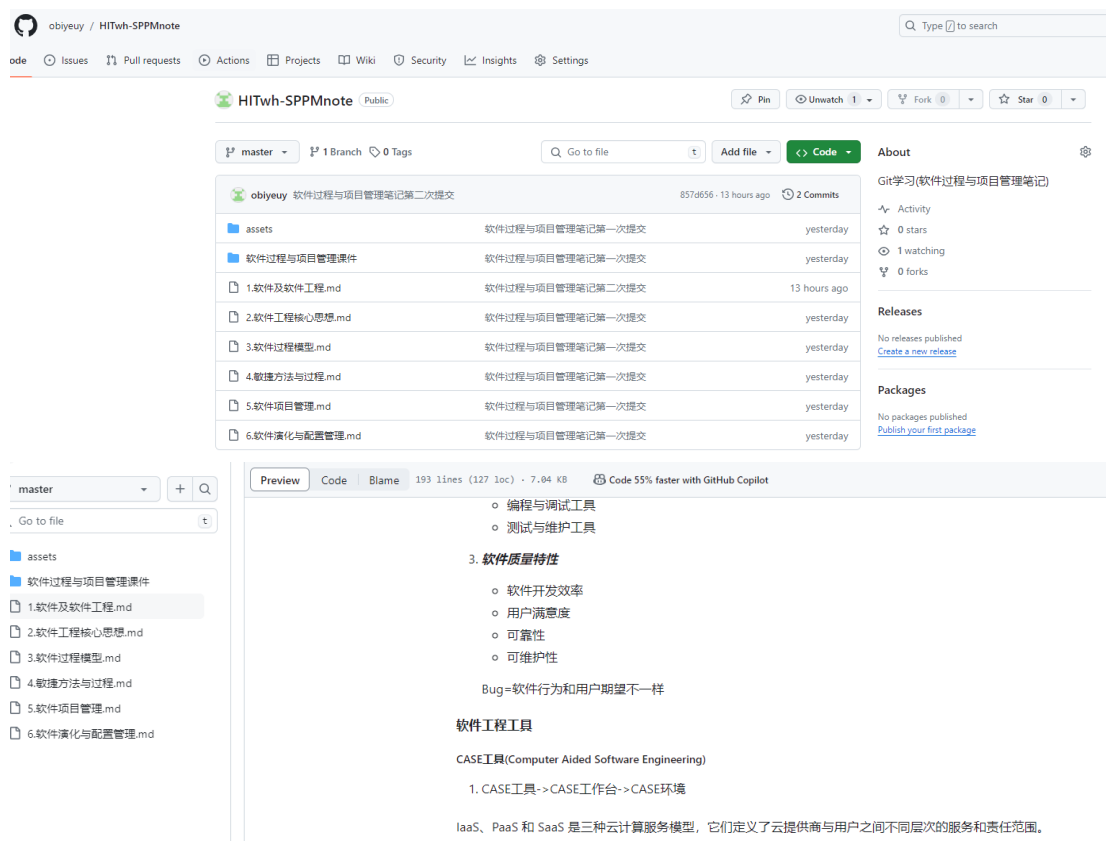
(12) R12: 在 Github 上以 Web 页面的方式查看两个仓库的当前状态。

分支练习的仓库，其状态如下：



可以发现提交的以本人学号命名的仓库已经可以在分支中找到，并且原来在本地对各种分支的操作并不会影响远程仓库的状态。

对于软件过程与项目管理笔记所保存的仓库，其状态如下：



3.3 实验场景(3)：在线 Git 练习

(一) 主要页面-基础篇

任务 1:

操作命令集

git commit

git commit

任务 2:

操作命令集

git branch bugFix

git checkout bugFix

任务 3:

操作命令集

git branch bugFix

git checkout bugFix

git commit

git checkout main

git commit

git merge bugFix

任务 4:

操作命令集

git branch bugFix

```
git checkout bugFix
```

```
git commit
```

```
git checkout main
```

```
git commit
```

```
git checkout bugFix
```

```
git rebase main
```

(二) 主要页面-高级篇

任务 1:

操作命令集

```
git checkout C4
```

任务 2:

操作命令集

```
git checkout bugFix^
```

任务 3:

操作命令集

```
git branch -f main C6
```

```
git checkout HEAD^
```

```
git branch -f bugFix C0
```

任务 4:

操作命令集

```
git reset local^
```

```
git checkout pushed
```

```
git revert pushed
```

(三) 主要页面-移动提交记录

任务 1:

操作命令集

```
git cherry-pick C3 C4 C7
```

任务 2:

操作命令集

```
git rebase -i HEAD~4
```

(四) 主要页面-杂项

任务 1:

操作命令集

```
git rebase -i HEAD~4
```

```
git branch -f main C4'
```

任务 2:

操作命令集

```
git rebase -i HEAD~2
```

```
git commit --amend
```

```
git rebase -i HEAD~2
```

```
git branch -f master
```

任务 3:

操作命令集

```
git checkout master
```

git cherry-pick newImage

git commit --amend

git cherry-pick caption

任务 4:

操作命令集

git tag v0 C1

git tag v1 C2

git checkout v1

任务 5:

操作命令集

git commit

(五) 主要页面-高级话题*

任务 1:

操作命令集

git rebase main bugFix

git rebase bugFix side

git rebase side another

git branch -f main another

任务 2:

操作命令集

git branch bugWork HEAD~^2~

任务 3:

操作命令集

git checkout one

git cherry-pick C4 C3 C2

git checkout two

git cherry-pick C5 C4 C3 C2

git branch -f three C2

(六) 远程页面-Git 远程仓库

任务 1:

操作命令集

git clone

任务 2:

操作命令集

git commit

git checkout o/main

git commit

任务 3:

操作命令集

git fetch

任务 4:

操作命令集

git pull

任务 5:

操作命令集

git clone

git fakeTeamwork 2

git commit

git pull

任务 6:

操作命令集

git commit

git commit

git push

任务 7:

操作命令集

git clone

git fakeTeamwork 1

git commit

git pull --rebase

git push

任务 8:

操作命令集

git reset --hard o/main

git checkout -b feature C2

git push origin feature

(七) 远程页面-Git 远程仓库高级操作

任务 1:

操作命令集

git fetch

git rebase o/main side1

git rebase side1 side2

git rebase side2 side3

git rebase side3 main

git push

任务 2:

操作命令集

git checkout main

git pull origin main

git merge side1

git merge side2

git merge side3

git push origin main

任务 3:

操作命令集

git checkout -b side o/main

git commit

git pull --rebase

git push

任务 4:

操作命令集

git push origin main

git push origin foo

任务 5:

操作命令集

git push origin foo:main

git push origin main^:foo

任务 6:

操作命令集

git fetch origin C2:foo

git fetch origin C3:foo

git fetch origin C5:main

git fetch origin C6:main

git checkout foo

git merge main

任务 7:

操作命令集

git pull origin :bar

git push origin :foo

任务 8:

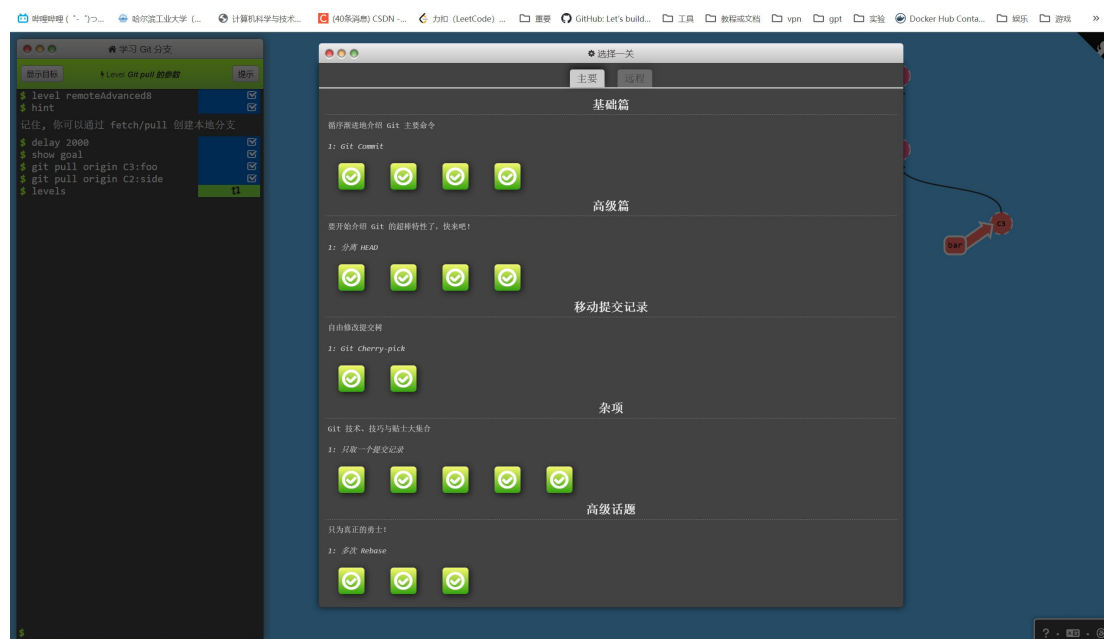
操作命令集

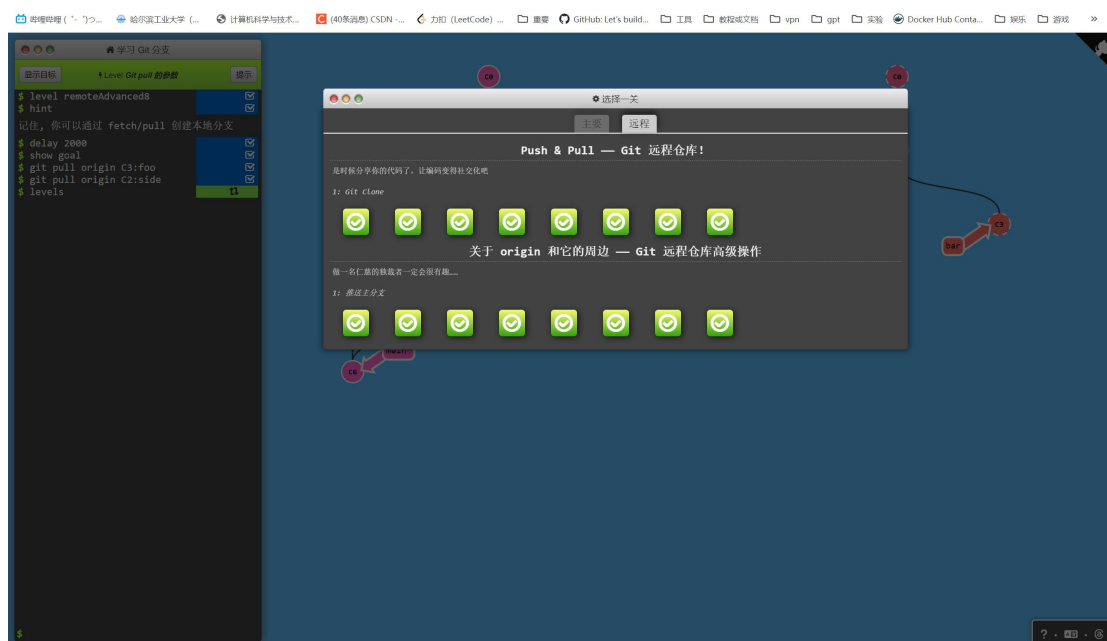
git pull origin C3:foo

git pull origin C2:side

(八) 通关后的主界面截图

完成规定任务后，“主页”和“远程”截图各一张，如下图所示：





4 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

对于个人开发：

- 1)Git 的分支系统允许无限制的同时进行多个独立的工作流，这使得测试、开发、修补 bug 和添加新功能等工作能够同时并行进行，并且能够随时切换。
- 2)我们使用 Git 可以轻松跟踪和理解项目的历史变化，有利于故障排查和功能追溯。
- 3)而且每一个 Git 工作目录都是一个完整的仓库，拥有完整的历史记录和版本跟踪能力，不依赖网络或者中央服务器。

对于团队开发：

- 1)Git 支持多人并行开发，每个开发者在本地工作并提交更改，然后将更改推送到中央仓库。这降低了多人协作开发时的冲突和阻塞。
- 2)Git 是开源的，有着丰富的第三方工具和服务的支持，如 GitHub、GitLab、Gitee 等，使得源码的托管、团队协作、代码审查等工作变得更加方便。

（2）在个人开发和团队开发中，Git 起到的作用有何主要差异？

- 1)个人开发中，Git 的主要作用是版本控制、备份和恢复以及代码审查。
- 2)团队开发中，Git 的主要作用是并行开发、代码集成以及冲突解决。

3)总的来说, Git 在个人开发和团队开发中的核心作用都是版本控制和代码管理,但是在团队协作过程中,还有明显的并行开发、集成和冲突解决的需求。