项目管理文档

> **学号：2212452**
> **姓名：孟启轩**
> **专业：计算机科学与技术**

1. 实验要求

本次实验的目标是通过 Git 工具，掌握基本的版本控制操作，包括本地仓库初始化、文件提交、分支管理和远程仓库协作等技能。通过三大实验场景的操作，理解 Git 支持软件配置管理的机制，并在实践中体会其对个人开发和团队协作的支持。

实验内容包括：

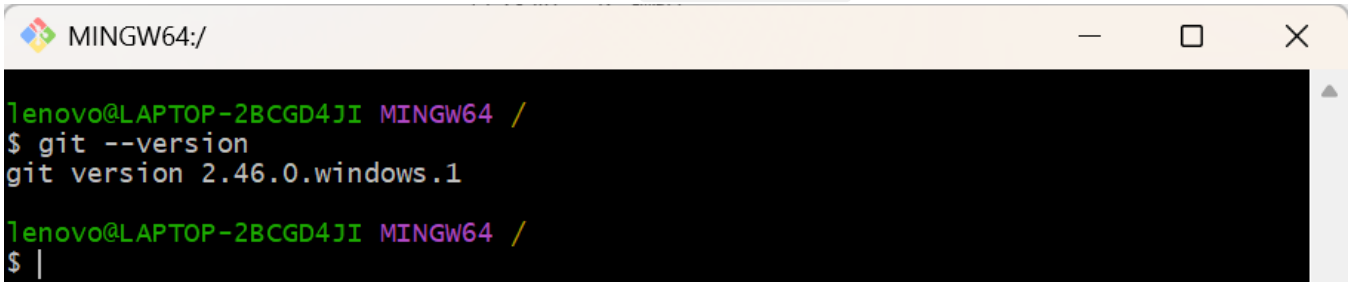
- 在本地初始化 Git 仓库并提交项目文件；
- 使用 Git 进行分支管理、文件修改与合并操作；
- 将本地仓库同步至 GitHub 远程仓库，完成协同开发流程；
- 完成实验报告撰写并提交。

2. 安装 Git

2.1 本地机器上安装 Git

使用 Windows 系统，在 Git 官网 下载并安装 Git，安装过程中保持默认选项。

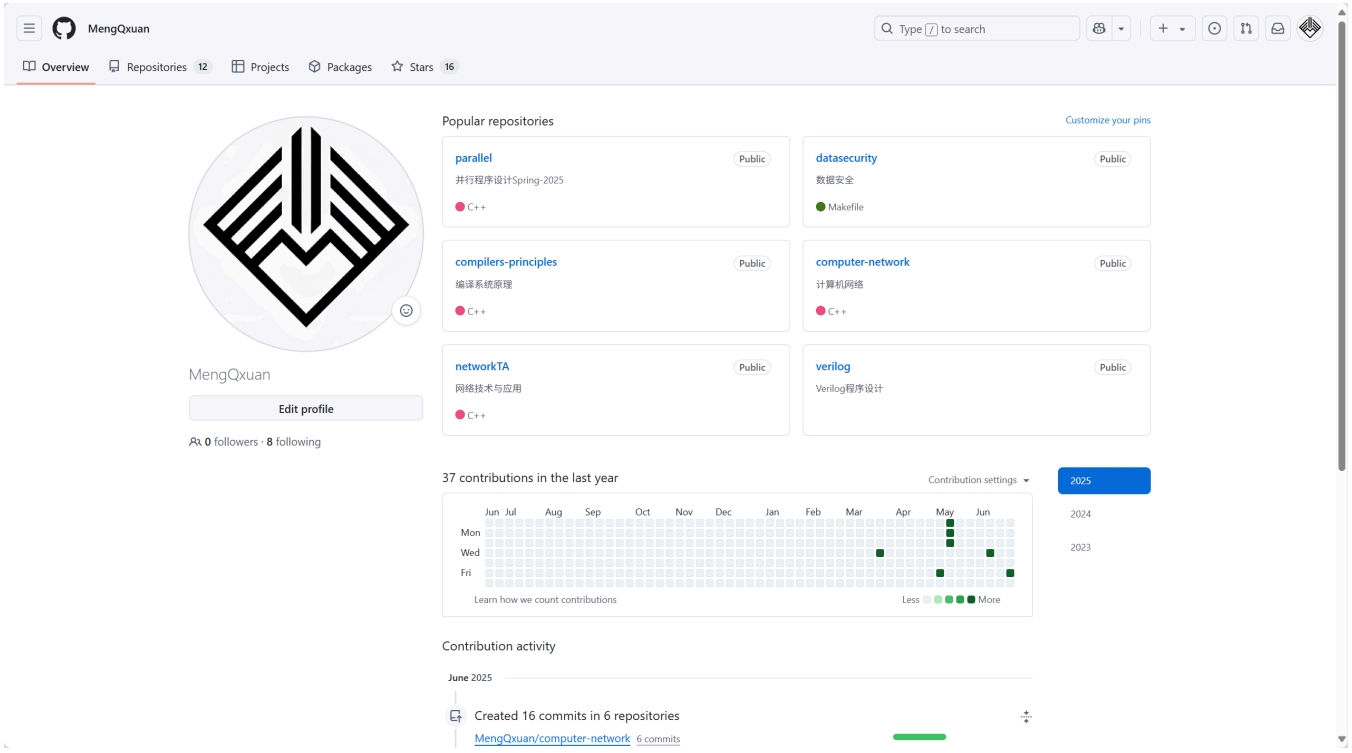
安装完成后，打开终端（Git Bash）输入命令 `git --version`：

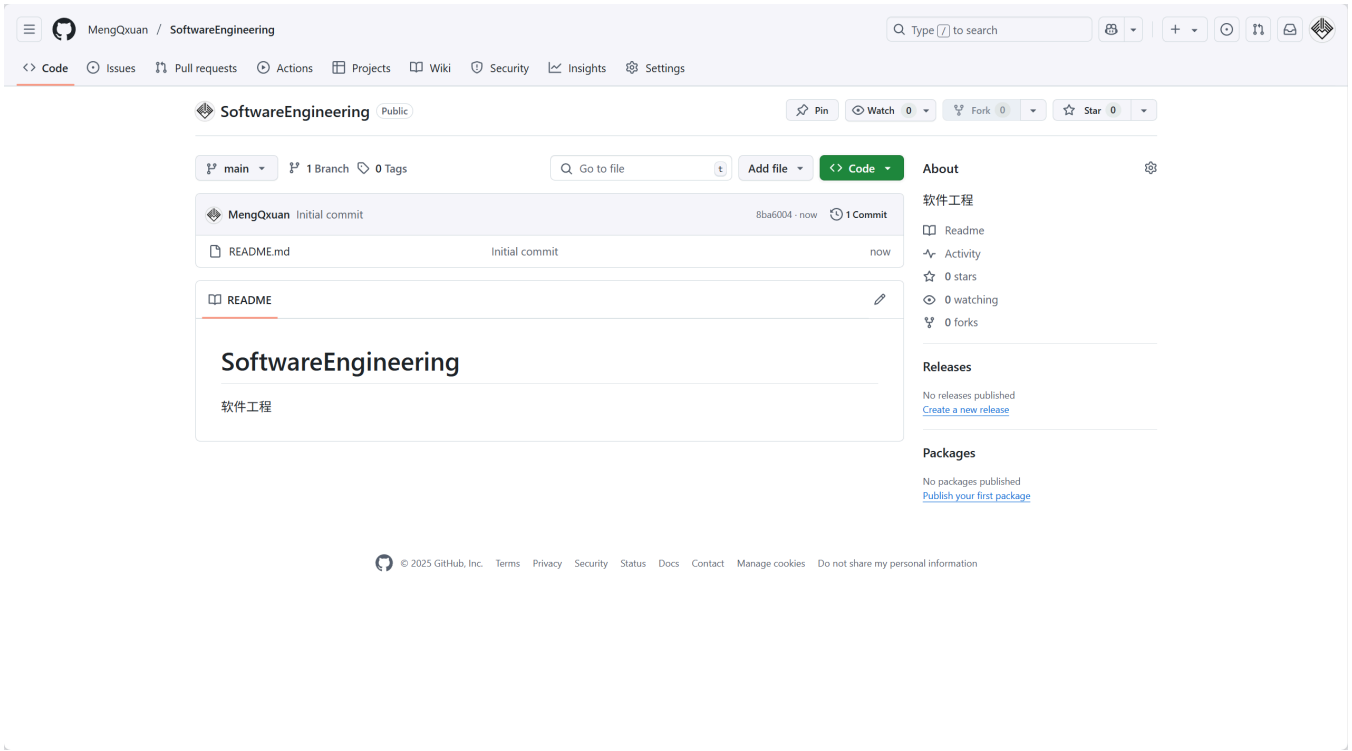


2.2 申请 GitHub 账号

在 https://github.com 注册账号并登录，账号信息如下：

- 用户名：`MengQxuan`
- 实验项目仓库地址：https://github.com/MengQxuan/SoftwareEngineering





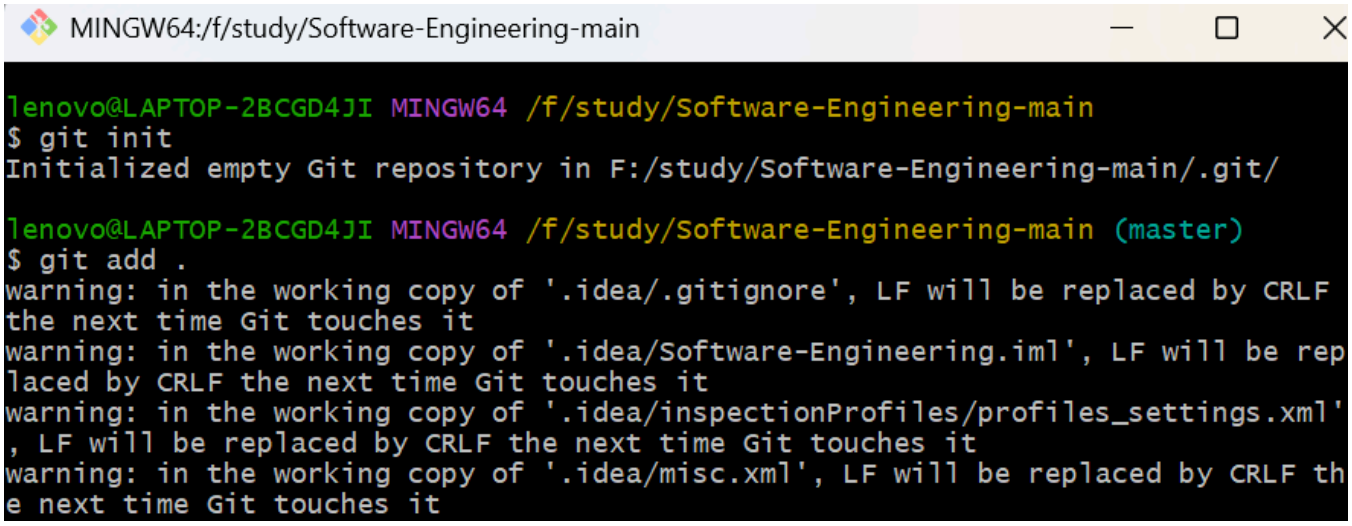
3. Git 操作过程

3.1 实验场景(1)：仓库创建与提交（R0~R7）

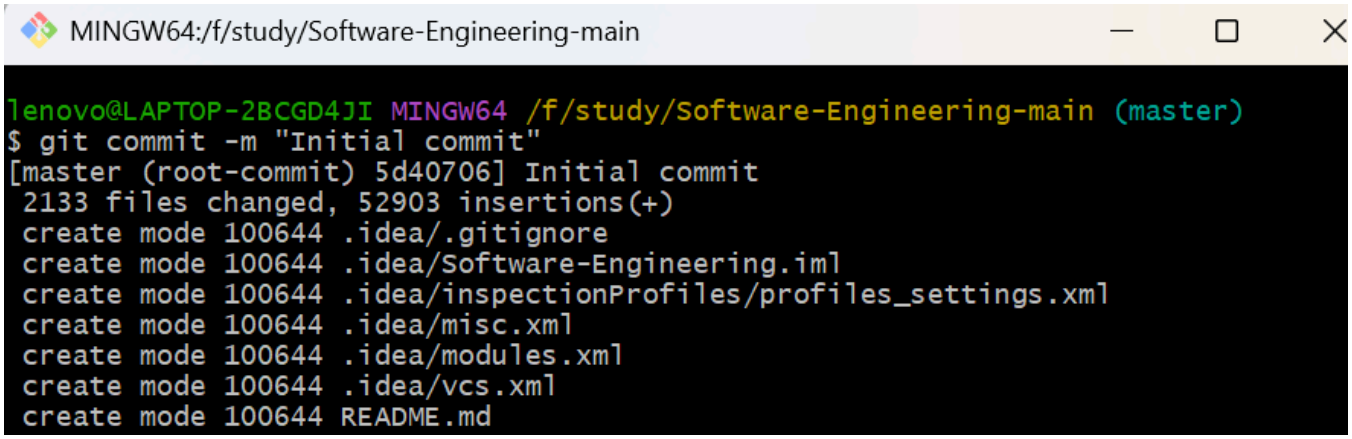
R0：查看当前状态

```
lenovo@LAPTOP-2BCGD4JI MINGW64 /f/study/Software-Engineering-main (master)
$ git status
On branch master
nothing to commit, working tree clean

lenovo@LAPTOP-2BCGD4JI MINGW64 /f/study/Software-Engineering-main (master)
$
```

R1：初始化仓库并添加源文件



R2：提交



R3：手工对团队作业中的3个文件进行修改并查看修改内容

```
lenovo@LAPTOP-2BCGD4JI MINGW64 /f/study/Software-Engineering-main (master)
$ git diff
warning: in the working copy of 'README.md', LF will be replaced by CRLF the nex
t time Git touches it
warning: in the working copy of 'config.py', LF will be replaced by CRLF the nex
t time Git touches it
warning: in the working copy of 'run.py', LF will be replaced by CRLF the next t
ime Git touches it
diff --git a/README.md b/README.md
index 1e59095..24cec20 100644
--- a/README.md
+++ b/README.md
@@ -26,9 +26,8 @@ python run.py

 * 数据库的初步搭建并和项目连接，登录和跳转的前后端逻辑基本实现
 * 注册功能初步实现
-* 感觉可以注册和登录分离成两个页面）
+* 感觉可以注册和登录分离成两个页面
 * 前后端已经调通了，数据库交互问题也不大
-* 中期阶段应该就差可视化部分了
 * 智能问答已实现
 * 图片识别实现
 * 体长预测实现
diff --git a/config.py b/config.py
index 4b712f1..1f23311 100644
--- a/config.py
+++ b/config.py
@@ -3,6 +3,6 @@
 DB_CONFIG = {
     "host": "localhost",
     "user": "root",
-    "password": "Mqx20041205",
+    "password": "Mqx111222333",
     "database": "smart_ocean_ranch"
 }
\ No newline at end of file
diff --git a/run.py b/run.py
index 5ad9a2e..745df30 100644
--- a/run.py
+++ b/run.py
@@ -4,4 +4,5 @@ from app import create_app
 app = create_app()

 if __name__ == '__main__':
-    app.run(host='0.0.0.0', port=5000, debug=True)
\ No newline at end of file
+    app.run(host='0.0.0.0', port=5000, debug=True)
+    # app.run(debug=True)
\ No newline at end of file
--cached
```

R4: 重新提交

```
lenovo@LAPTOP-2BCGD4JI MINGW64 /f/study/Software-Engineering-main (master)
$ git add .
warning: in the working copy of 'README.md', LF will be replaced by CRLF the nex
t time Git touches it
warning: in the working copy of 'config.py', LF will be replaced by CRLF the nex
t time Git touches it
warning: in the working copy of 'run.py', LF will be replaced by CRLF the next t
ime Git touches it

lenovo@LAPTOP-2BCGD4JI MINGW64 /f/study/Software-Engineering-main (master)
$ git commit -m "Second edit"
[master 397dbb6] Second edit
 3 files changed, 4 insertions(+), 4 deletions(-)
```

R5：再次对这3个文件进行修改并重新提交

```
lenovo@LAPTOP-2BCGD4JI MINGW64 /f/study/Software-Engineering-main (master)
$ git add .
warning: in the working copy of 'README.md', LF will be replaced by CRLF the next t
ime Git touches it
warning: in the working copy of 'config.py', LF will be replaced by CRLF the next t
ime Git touches it
warning: in the working copy of 'run.py', LF will be replaced by CRLF the next time
 Git touches it

lenovo@LAPTOP-2BCGD4JI MINGW64 /f/study/Software-Engineering-main (master)
$ git commit -m "Third edit"
[master bc14363] Third edit
 3 files changed, 3 insertions(+), 5 deletions(-)
```

R6 & R7：撤销最后一次提交并查看提交记录

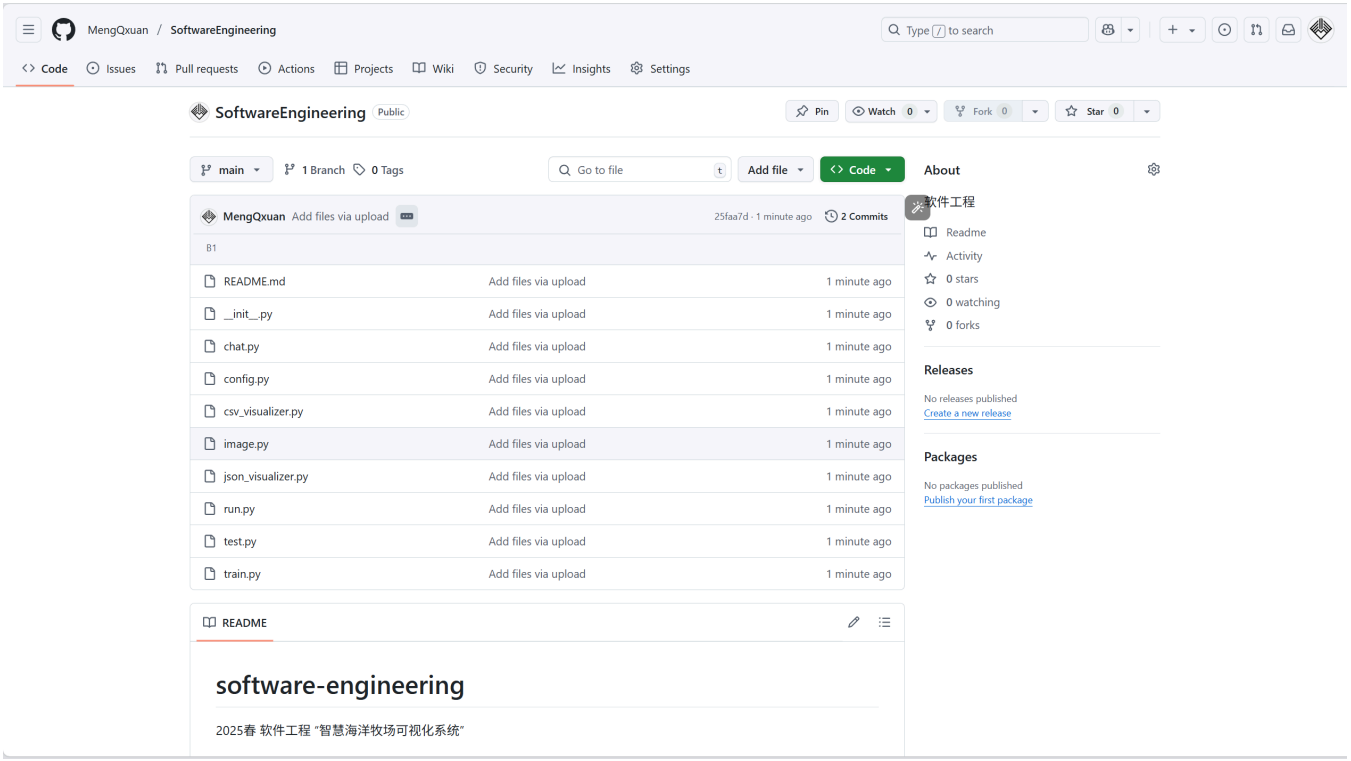
```
lenovo@LAPTOP-2BCGD4JI MINGW64 /f/study/Software-Engineering-main (master)
$ git reset --soft HEAD~1

lenovo@LAPTOP-2BCGD4JI MINGW64 /f/study/Software-Engineering-main (master)
$ git log --oneline
397dbb6 (HEAD -> master) Second edit
bd6fd9e Edit 3 files
5d40706 Initial commit
```

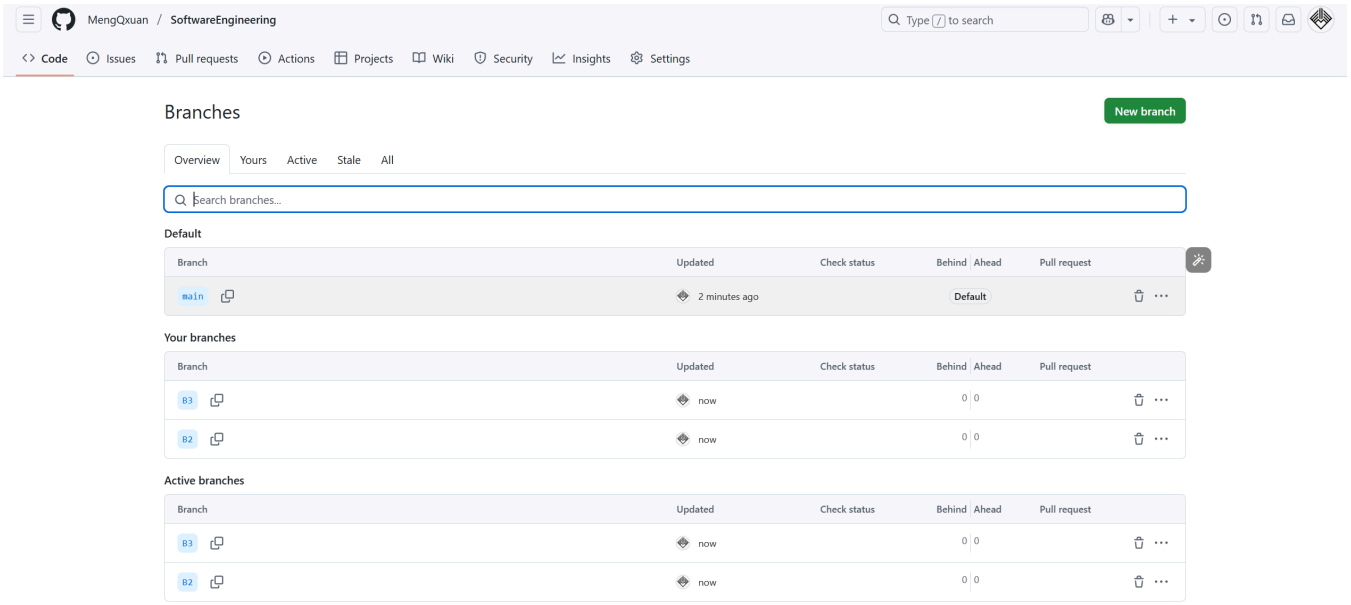
```
lenovo@LAPTOP-2BCGD4JI MINGW64 /f/study/Software-Engineering-main (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   README.md
        modified:   config.py
        modified:   run.py
```

3.2 实验场景(2)：分支管理（R8~R15)

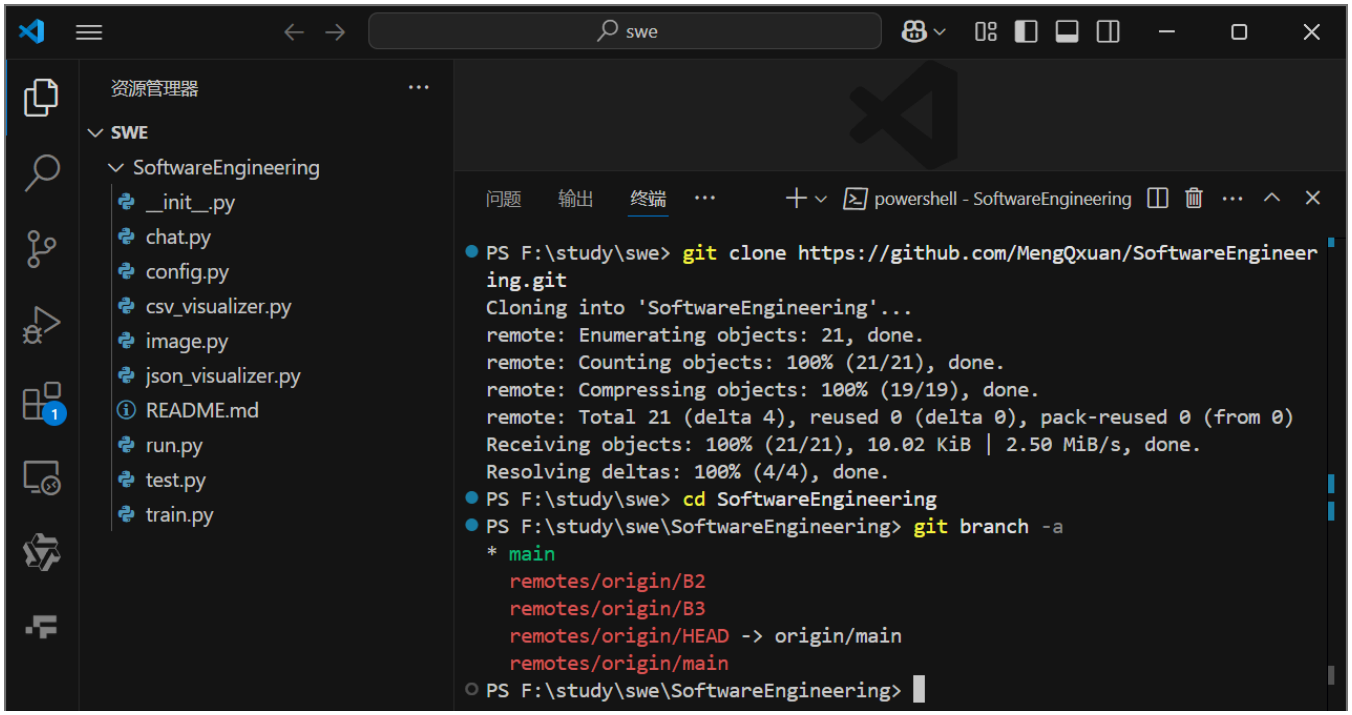
在 GitHub Web 上创建并操作分支：

1. 主分支 `B1` （main）上传了不少于10个文件；
2. 创建两个并行分支：`B2` 与 `B3`；
3. 在 Web 界面中对 `B2`、`B3` 上的文件进行了不同修改，并提交。





R8 & R9：克隆远程仓库到本地并获取全部分支

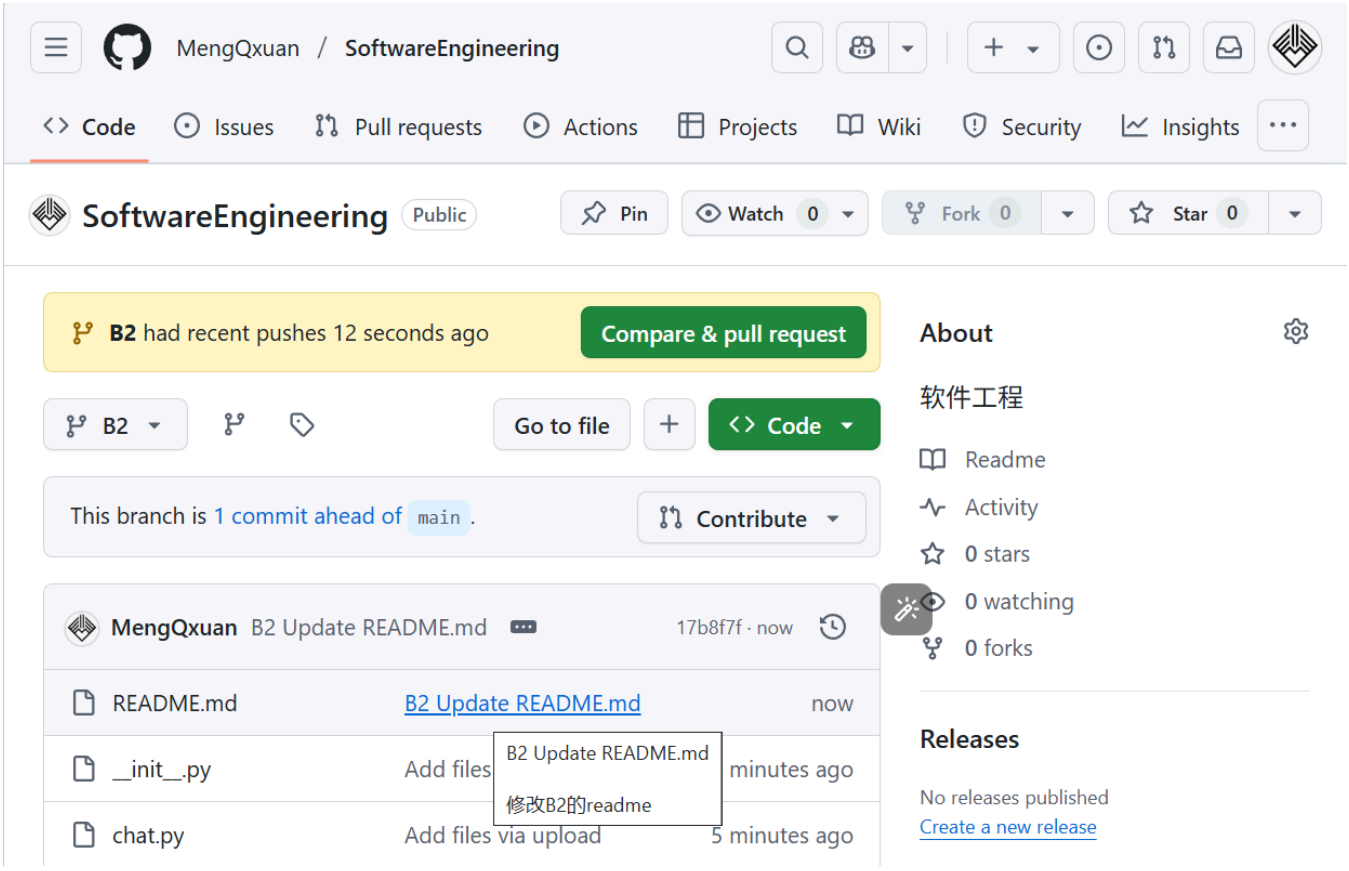


R10：基于 B2 创建新分支 C4

```
PS F:\study\swe\SoftwareEngineering> git checkout -b B2 origin/B2
branch 'B2' set up to track 'origin/B2'.
Switched to a new branch 'B2'
PS F:\study\swe\SoftwareEngineering> git checkout -b C4
Switched to a new branch 'C4'
PS F:\study\swe\SoftwareEngineering>
```

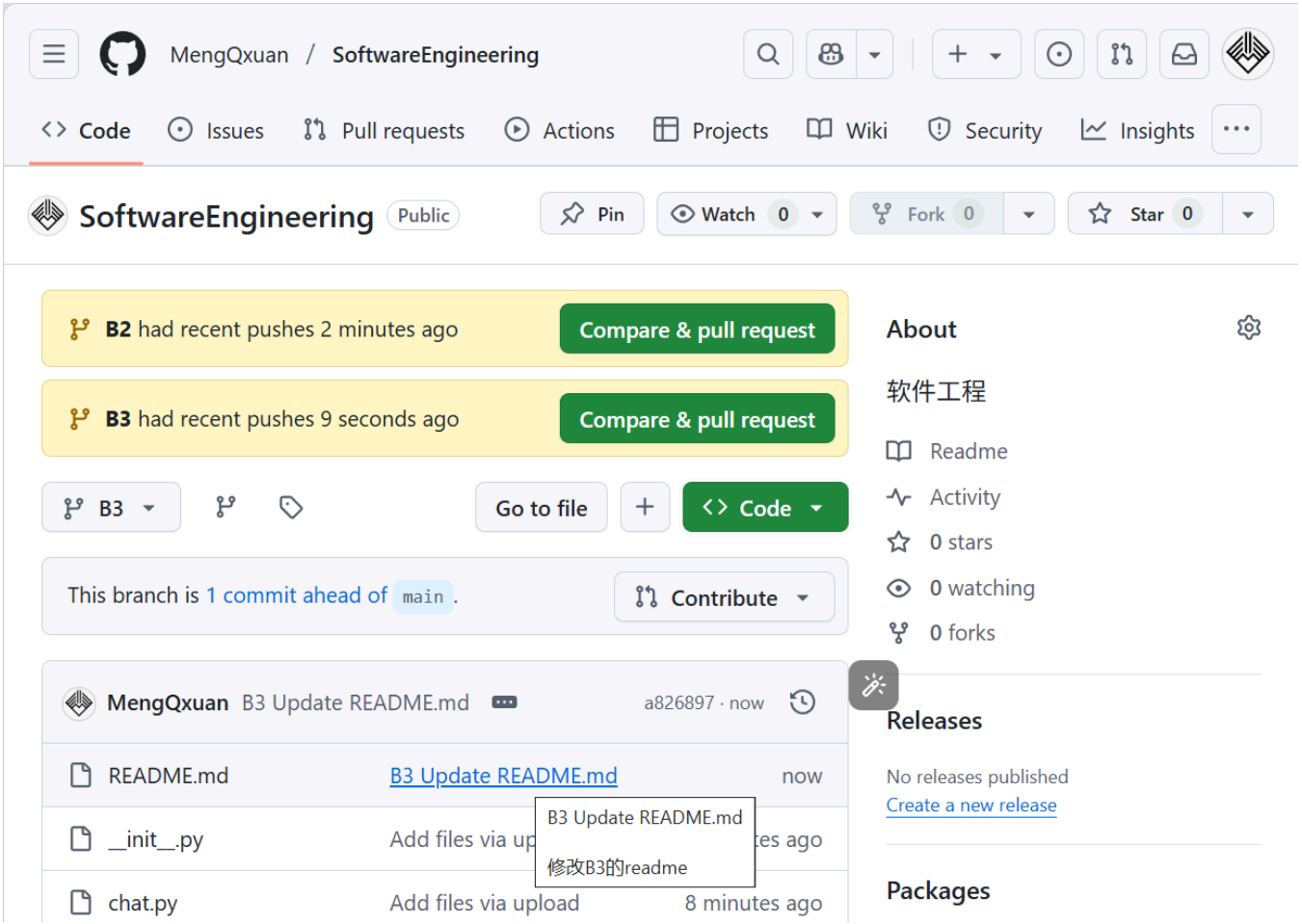
R11：C4 上修改 4 个文件并提交

```
PS F:\study\swe\SoftwareEngineering>  git add chat.py image.py README.md run.py
PS F:\study\swe\SoftwareEngineering> git commit -m "C4 updated 4 files"
[C4 fa3e938] C4 updated 4 files
 4 files changed, 17 insertions(+), 7 deletions(-)
PS F:\study\swe\SoftwareEngineering>
```



R12：在 B3 上对相同文件进行不同修改并提交

```
PS F:\study\swe\SoftwareEngineering> git checkout B3
branch 'B3' set up to track 'origin/B3'.
Switched to a new branch 'B3'
PS F:\study\swe\SoftwareEngineering> git add chat.py image.py README.md run.py
PS F:\study\swe\SoftwareEngineering> git commit -m "B3 updated same 4 files diffe
rently"
[B3 8c84302] B3 updated same 4 files differently
 4 files changed, 4 insertions(+), 5 deletions(-)
PS F:\study\swe\SoftwareEngineering>
```



R13：合并 C4 和 B3

```
PS F:\study\swe\SoftwareEngineering> git checkout B3
Already on 'B3'
Your branch is ahead of 'origin/B3' by 1 commit.
  (use "git push" to publish your local commits)
PS F:\study\swe\SoftwareEngineering> git merge C4
Auto-merging README.md
Auto-merging chat.py
Auto-merging image.py
Auto-merging run.py
Merge made by the 'ort' strategy.
 README.md | 12 ++++++++++++
 chat.py   |  1 -
 image.py  | 10 +++++-----
 run.py    |  2 +-
 4 files changed, 18 insertions(+), 7 deletions(-)
PS F:\study\swe\SoftwareEngineering>
```

R14：查看哪些分支已/未合并

```
PS F:\study\swe\SoftwareEngineering> git branch --merged
  B2
* B3
  C4
  main
PS F:\study\swe\SoftwareEngineering> git branch --no-merged
PS F:\study\swe\SoftwareEngineering>
```

R15：删除合并分支并创建新分支合并未合并内容

```
PS F:\study\swe\SoftwareEngineering> git branch -d C4
Deleted branch C4 (was fa3e938).
PS F:\study\swe\SoftwareEngineering> git checkout -b 2212452
Switched to a new branch '2212452'
PS F:\study\swe\SoftwareEngineering> git merge B2
Already up to date.
PS F:\study\swe\SoftwareEngineering>
```

3.3 实验场景(3)：远程分支管理（R16~R18）

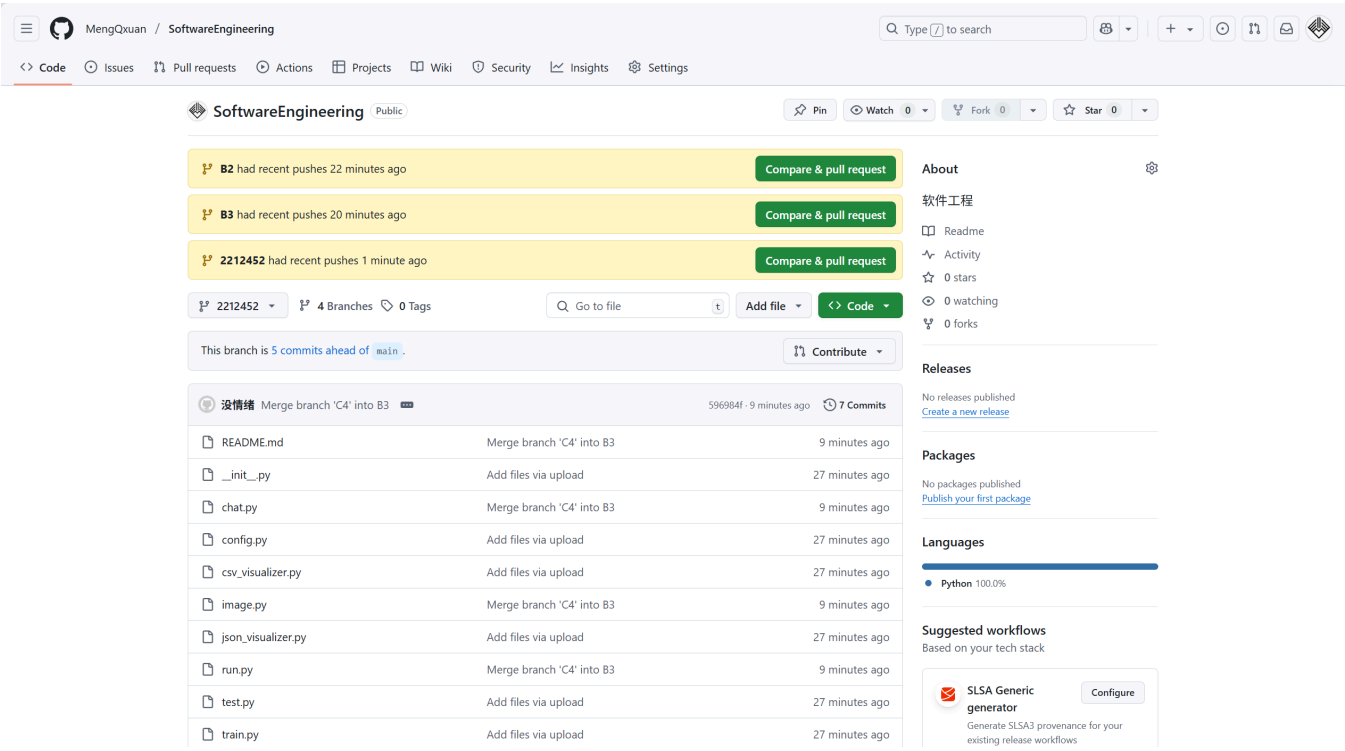
R16：将学号分支推送到远程仓库

```
PS F:\study\swe\SoftwareEngineering> git push origin 2212452
Enumerating objects: 28, done.
Counting objects: 100% (25/25), done.
Delta compression using up to 16 threads
Compressing objects: 100% (18/18), done.
Writing objects: 100% (18/18), 1.70 KiB | 871.00 KiB/s, done.
Total 18 (delta 15), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (15/15), completed with 6 local objects.
remote:
remote: Create a pull request for '2212452' on GitHub by visiting:
remote:      https://github.com/MengQxuan/SoftwareEngineering/pull/new/2212452
remote:
To https://github.com/MengQxuan/SoftwareEngineering.git
 * [new branch]      2212452 -> 2212452
PS F:\study\swe\SoftwareEngineering>
```

R17：将 R1~R7 操作结果推送到远程

```
PS F:\study\swe\SoftwareEngineering> git push origin main
Everything up-to-date
PS F:\study\swe\SoftwareEngineering>
```

R18：在 GitHub Web 查看两个分支状态



4. 小结

比较之前的开发经验，使用git的优点?

通过此次实验，深刻体会到了 Git 在版本控制中的重要作用。与我之前的开发经验相比，Git 能够自动记录每一次提交、变更与分支修改，并可方便地回滚和对比历史版本，Git 给予了我更高的控制能力，更好的版本管理能力，更好的协作能力，可以与他人协作开发，提升开发效率。

在个人开发和团队开发中，git起到的作用有何主要差异?

- 在个人开发中，Git 可以清晰地记录开发过程和代码版本演化过程；
- 在团队协作中，Git 的分支和合并机制能有效避免多人修改同一文件引发的冲突。

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。

在本次实验之前，我尚未使用过其他版本控制系统。但通过查阅相关资料，了解到目前主流的版本控制系统主要包括 Git、SVN（Subversion）、CVS等，它们各自具有不同的设计理念与适用场景。

对比维度	Git（分布式）	SVN（集中式）
架构	分布式，每个开发者有完整仓库副本	集中式，所有历史记录保存在中央服务器
离线操作	支持绝大多数操作（提交、查看历史等）	必须联网才能提交、查看历史等操作
分支管理	分支创建快速，操作灵活，适合并行开发	分支操作繁琐，不推荐频繁使用
速度与性能	本地操作快，适合大项目	每次操作需联网，性能依赖网络
学习曲线	操作概念多，学习曲线相对较陡峭	操作较直观，适合初学者
合并与冲突处理	更智能的合并工具和冲突提示	合并处理能力相对较弱

此外，CVS 是一种更早期的集中式版本控制系统，已经逐渐被 SVN 和 Git 取代。它不支持原子提交，分支功能也较弱。

通过对比可以看出，**Git 更适合现代软件开发中的协作与频繁分支管理，功能强大，灵活性高，已成为主流版本控制工具。**

在什么情况下适合使用git、什么情况下没必要使用git?

- 适合场景：中大型项目、团队开发、开源协作
- 不适合场景：小型个人笔记、一次性代码实验等可直接保存在本地

其他相关思考

通过本次实验，我不仅掌握了 Git 的基本使用方法，还对 Git 背后“分布式版本控制”的设计理念有了更深刻的理解。实验过程中，尤其是在操作多个分支、合并代码并解决冲突的环节，我切实体会到 Git 在团队协作中的强大优势。分支操作的高效性和灵活性使得不同成员可以并行开发而不会相互干扰，而合并与回滚机制的支持也为项目的可控性和可维护性提供了坚实保障。

总的来说，这次实验不仅提升了我对版本控制工具的实际操作能力，也让我认识到软件工程中“配置管理”的重要性。它是确保项目开发高效、稳定、可追溯的重要工具。未来在更复杂的项目开发中，我会更加注重使用 Git 来规范开发流程、降低协作成本。