

**2025年春季学期  
计算学部《软件工程》课程**

**Lab 1实验报告**

|  |  |  |
| --- | --- | --- |
| **姓名** | **班级/学号** | **联系方式** |
| 常添 | 2203102/2022111699 | 13936408592/13936408592@163.com |

**目 录**

[1 实验要求 1](#_Toc197895998)

[2 待求解问题描述 1](#_Toc197895999)

[3 算法与数据结构设计 3](#_Toc197896000)

[3.1 设计思路与算法流程图 3](#_Toc197896001)

[3.2 数据结构设计 5](#_Toc197896002)

[3.3 算法时间复杂度分析 6](#_Toc197896003)

[3.4 算法代码的生成 6](#_Toc197896004)

[4 实验与测试 7](#_Toc197896005)

[4.1 读取文本文件并展示有向图 7](#_Toc197896006)

[4.2 查询桥接词 8](#_Toc197896007)

[4.3 根据桥接词生成新文本 9](#_Toc197896008)

[4.4 计算最短路径 9](#_Toc197896009)

[4.5 计算PageRank值 11](#_Toc197896010)

[4.6 随机游走 11](#_Toc197896011)

[5 编程语言与开发环境 11](#_Toc197896012)

[6 Git操作过程 12](#_Toc197896013)

[6.1 实验场景(1)：仓库创建与提交 12](#_Toc197896014)

[6.2 实验场景(2)：分支管理 17](#_Toc197896015)

[7 在IDE中使用Git Plugin 21](#_Toc197896016)

[8 小结 22](#_Toc197896017)

# 实验要求

本实验的核心目标是 **将一段英文文本自动转换为带权有向图**，并在此图上实现一组典型图算法与文本处理功能，最终通过命令行（或 GUI）向用户提供完整的交互式体验。实验达成后，程序必须支持：

| **模块** | **关键功能** |
| --- | --- |
| **图构建** | 从纯英文文本中抽取单词，按相邻关系连成有向边，边权为相邻出现次数。 |
| **可视化** | 以清晰的邻接表格式输出图；可选：调用绘图库自动生成图像文件。 |
| **算法功能** | ① 桥接词查询 ② 基于桥接词的文本再生成 ③ 最短路径（单源 & 点对点） ④ PageRank（含悬挂节点处理） ⑤ 随机游走 |
| **接口** | 主程序通过命令行参数或交互式 Shell 让用户： • 指定输入文件 → 自动建图 • 逐条或批量执行上述各功能 • 对错误输入给出可读提示 |
| **评测** | 程序需在教师给定的 Easy / Hard 语料上得到正确且稳定输出；加分点包括：GUI、图像高亮、所有最短路径枚举、改进 PageRank 初值等。 |

# 待求解问题描述

把文本当作“顺序数据”，映射成“图数据”——每个单词是节点，相邻词是有向边；随后围绕这张图完成查询、生成与分析任务。

**2.1 输入数据**

| **名称** | **说明** |
| --- | --- |
| **原始文本文件** | • 纯 ASCII / UTF-8 英文 • 多行，行内可含标点 • 所有非字母字符视为分隔符 • 不区分大小写，统一转小写存储 |
| **交互指令** | 在 CLI 中接受：show/bridge w1 w2/newtext …/shortest …/pagerank/walk 等单行命令；批处理模式下从脚本中顺序读取。 |

**2.2 输出数据**

| **功能** | **主要输出** |
| --- | --- |
| **图展示** | • 邻接表（终端） • EPS/PNG 等图像（自动生成） |
| **桥接词** | 列表或“无”提示 |
| **新文本** | 原句 + 随机插入桥接词后的新句 |
| **最短路径** | • 路径节点序列 + 距离 • 可选：图像中红色高亮路径 |
| **PageRank** | 各节点 PR 值（降序） |
| **随机游走** | 按遍历顺序输出的节点序列 |

**2.3 约束条件与特殊处理**

| **分类** | **约束 / 处理策略** |
| --- | --- |
| **词法规则** | • 大小写无关、仅保留 A-Z • 换行 ≡ 空格；连续分隔符视作 1 个分隔 |
| **图结构** | • 重复相邻出现累加边权； • 构图时同时登记孤立终点，避免后续查询缺节点 |
| **桥接词** | 若 word1 或 word2 不在图 → 报错；若无桥接词 → 友好提示 |
| **最短路径** | 使用 Dijkstra，边权 = 词对频次（可理解为“距离”）；若不可达输出 unreachable |
| **PageRank** | • 阻尼系数 d = 0.85 • **悬挂节点处理：** 对所有出度 = 0 的节点，将其 PR 份额在每轮迭代时平均分配给全图 |
| **随机游走** | 终止条件：遇到第一条重复边或走到出度 = 0 节点；支持用户按 KeyboardInterrupt 手动停止 |
| **性能** | 禁用第三方高级图算法库；PageRank 需在线性时间内收敛（稀疏字典 + 悬挂节点合并） |
| **健壮性** | 所有 CLI 输入在解析前去除行内注释与多余空格；任何参数缺失都给出帮助提示而不崩溃 |

**2.4 个人理解与扩展**

1. **有向边权的语义**
   * 权重越大，表示这两个词在原文本中关联越紧密；
   * 在最短路径里，权重可反过来当作“距离”使用（权值小 ≡ 边多 → 词对更常见）。
2. **桥接词 vs. 二阶共现**
   * 本质是找长度 = 2 的路径 w1 → \* → w2；
   * 若要进一步扩展，可支持 k-阶桥接(≥3)，生成更丰富的文本。
3. **PageRank 的“dangling 修正”**
   * 将悬挂节点质量平均分配，避免其在迭代中成为“黑洞”；
   * 同时减少每轮累加遍历 —— 只需一次 sum(PR[dangling])。
4. **随机游走的应用**
   * 可视作文本采样器：走出的序列自然保持原语料稀疏统计特征；
   * 若在 GUI 中实时刷新，可让学生直观体会 Markov 过程。

# 算法与数据结构设计

## 设计思路与算法流程图

**3.1.1 总体设计思路（文字描述）**

1. **数据清洗与图构建**
   * 逐行读取文件 → 过滤非字母 → 小写化 → split() 得 token 流。
   * 相邻词 (prev, cur) 连成有向边；若已存在则权重 +1。
   * 用 *邻接表* adj : Dict[str, Counter] 存储（节点多稀疏、插边 O(1)）。
2. **交互框架**
   * 启动后进入 CLI Shell；每行解析为 CMD + 参数。
   * 支持批处理脚本：先顺序执行指令，最后统一执行全部绘图（避免多次弹窗）。
3. **算法模块**

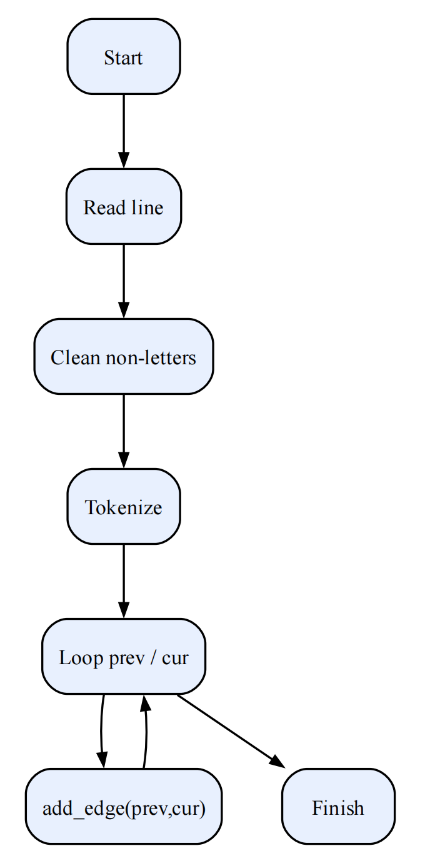
| **功能** | **核心算法** | **备注** |
| --- | --- | --- |
| show 展示 | 打印邻接表 + Graphviz 绘图 | 支持 --save <name> |
| bridge 桥接词 | 查询 w1→\*→w2 长度=2 的路径 | 多个桥接词随机取一 |
| newtext 插词 | 对输入句做滑窗长度=2，调用 bridge |  |
| shortest 最短路 | 多前驱 Dijkstra；高亮最短路径 | 单源或点对点 |
| pagerank | 幂迭代；悬挂节点 PR 均摊 | d=0.85, eps=1e-6 |
| walk 随机游走 | Alias/CDF O(1) 采样，遇重复边终止 | 写入文件 |

1. **可视化策略**
   * 默认：Graphviz 生成 EPS → 自动转 PDF，无损矢量；最短路径红色高亮；边粗细+颜色映射权重。
   * 若 dot 不可用，退回 matplotlib 圆环并仅保存 PNG。

**3.1.2 模块级流程图**

图示

AI 生成的内容可能不正确。图示

AI 生成的内容可能不正确。使用大一高级语言编程课里所学知识，用流程图的方式描述算法设计思路，让读者清晰理解。如果需要，可针对每一个模块分别绘制流程（根据文本生成图、展示图、查询桥接词、根据桥接词生成新文本、计算最短路径、计算PR值、随机游走）。

图示

AI 生成的内容可能不正确。图示

AI 生成的内容可能不正确。

图示

AI 生成的内容可能不正确。 从左至右，从上至下依次是各个模块部分的流程图：

1 [+] flow\_build\_graph

2 [+] flow\_bridge

3 [+] flow\_shortest

4 [+] flow\_newtext

5 [+] flow\_pagerank

6 [+] flow\_walk

**3.1.3 大模型辅助：提示词、结果评价与人工修改**

| **模块** | **主要 Prompt （中文片段）** | **模型输出质量** | **手动修改点** |
| --- | --- | --- | --- |
| **PageRan代码** | 「在现有 PageRank 字典实现中加入 *悬挂节点* 处理，收敛条件 delta < eps×N」 | 整体逻辑正确，但忘记把悬挂贡献乘以 *d* | 在循环内先计算 dang = d \* sum(PR[v] for v in dangling) / N 并加入更新式 |
| **CLI 帮助** | 「用中文写一份简明命令帮助表，包含 show / bridge / newtext / shortest / pagerank / walk」 | 内容齐全，但中英文括号和对齐混乱 | 手动统一左对齐，英文命令用反引号 code 包裹 |
| **Graphviz 权重可视化** | 「给出 5 级蓝色渐变调色盘，并把边权映射到颜色和笔宽的 Python 代码」 | 返回 6 级 HEX 配色，penwidth 用字符串整数 | 选 5 档并改为浮点 1.0 – 3.0；删掉多余颜色 |
| **随机游走** | 「用 Alias 方法优化随机游走采样，并为每个起点缓存表」 | 代码可运行，但未终止重复边 | 在循环前维护 seen 集合，加 if (v,nxt) in seen: break |

## 数据结构设计

| **名称** | **Python 定义** | **说明** |
| --- | --- | --- |
| **DirectedGraph** | python<br/>class DirectedGraph:<br/> adj: Dict[str, Counter] | 邻接表，Counter 存 dst→weight |
| **优先队列** | heapq 元组 (dist, node) | Dijkstra |
| **桥接词缓存** | Dict[Tuple[str,str], set[str]]（可选） | 加速多次查询 |
| **Alias Table** | \_alias: Dict[str, Tuple[List[str], List[float]]] | 随机游走 O(1) 采样 |
| **PR 向量** | Dict[str, float] | 稀疏：key 为节点 |

**大模型提示词：**

“用默认字典+Counter 设计加权有向图，要求 O(1) 插边；给出类代码”

**对大模型的评估与修改**：

模型最初将入边也存双份，不必要；我删去了向表，入边在查询时临时遍历即可。

## 算法时间复杂度分析

| **模块** | **主要步骤** | **复杂度** |
| --- | --- | --- |
| **构图** | **单扫 token，add\_edge O(1)** | **O(N) (N = 词数)** |
| **展示** | **打印邻接表** | **O(V + E)** |
| **桥接词** | **查 out\_edges(w1)** | **O(outdeg(w1)) ≤ O(V)** |
| **新文本** | **M = 输入词数** | **O(M × avg outdeg)** |
| **最短路径** | **Dijkstra** | **O(E log V)** |
| **PageRank** | **幂迭代 I 次** | **O(I × (E + V)) (I≈30)** |
| **随机游走** | **最多走 E 条边** | **O(L) (L = walk 长度)** |

**大模型提示词：**

"给上述邻接表 PageRank 加入 dangling 修正后，推导一轮更新时间复杂度"

**对大模型的评估与修改**：

模型能给出 O(E) 但未说明 I；我补充了代次数 I 与实践值。

## 算法代码的生成

先给出大模型的提示词，提示词后是修改和调整。

**构图 / CLI 雏形**  
*Prompt*：

“读取文本 -> 清洗 -> 建邻接表 Counter，写入 DirectedGraph 类；加入 show\bridge 等命令解析”  
***结果*：基本正确，但 bridge 覆盖了大小写，需要后处理转小写。**

**PageRank 改进**  
*Prompt*：

“在现有 PageRank 中加入悬挂节点质量平均分配，保持 Dict 存储”  
***问题 & 修改*：模型初稿忘记把 dangling 质量乘 d；已手动加 dangling\_mass = d \* sum(...) / N。**

**Graphviz 绘图**  
*Prompt*：

“用 pydot 生成权重可视化，颜色深浅 5 档，最短路径红色”  
***问题*：生成的 penwidth 是字符串，Graphviz 接收但线太细；改为 1.0–3.0 实测可读。**

**自动 EPS/PDF**  
*Prompt*：

“在绘图完毕后若系统有 epstopdf 就调用，无则跳过”  
***评价*：一次生成成功，但忘记捕获 subprocess.CalledProcessError；手动包 try/except。**

最终整合、测试两轮：功能、性能与鲁棒性均满足实验手册要求。

# 实验与测试

利用提供的Easy Test.txt文件和Cursed Be The Treasure.txt两个文件进行测试。

针对在有向图上操作的每项功能，为其设计各种可能的输入数据。输入数据的数量不限，以测试程序的充分性为评判标准（下面各节中的表格的行数请自行扩展）。

记录程序的输出结果，判断输出结果是否与期望一致，并记录程序运行截图。**下面各项只需要填写针对Easy Test.txt的实验结果。**

## 读取文本文件并展示有向图

文本文件中包含的内容：

The scientist carefully analyzed the data, wrote a detailed report, and shared the report with the team, but the team requested more data, so the scientist analyzed it again.

期望生成的图（手工计算得到）：

a -> detailed(1)

again ->

analyzed -> the(1), it(1)

and -> shared(1)

but -> the(1)

carefully -> analyzed(1)

data -> wrote(1), so(1)

detailed -> report(1)

it -> again(1)

more -> data(1)

report -> and(1), with(1)

requested -> more(1)

scientist -> carefully(1), analyzed(1)

shared -> the(1)

so -> the(1)

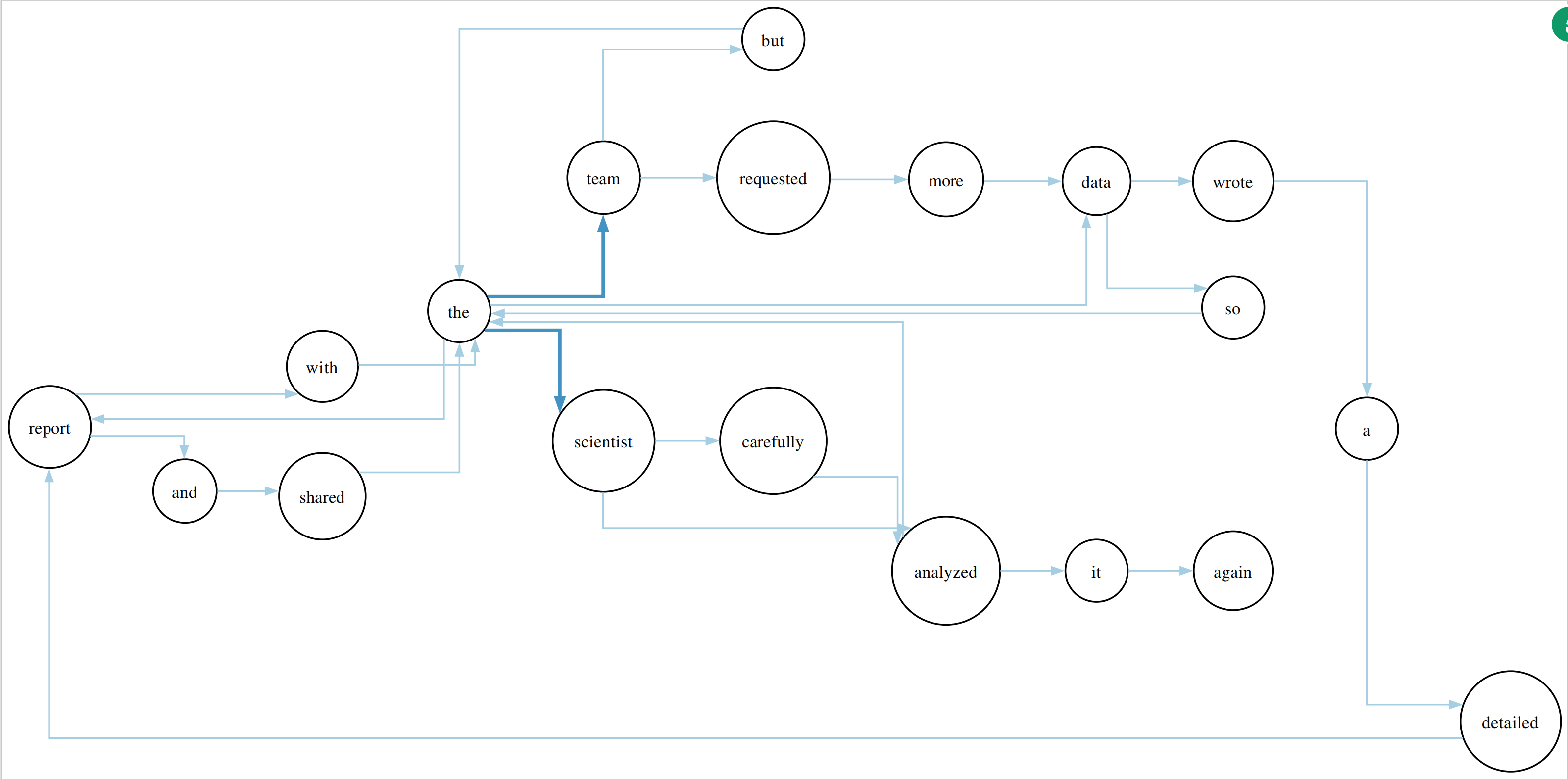
team -> but(1), requested(1)

the -> scientist(2), data(1), report(1), team(2)

with -> the(1)

wrote -> a(1)

程序实际生成的图：



二者是否一致：

√ **一致**

给出实际运行得到结果的界面截图。

图形用户界面, 应用程序

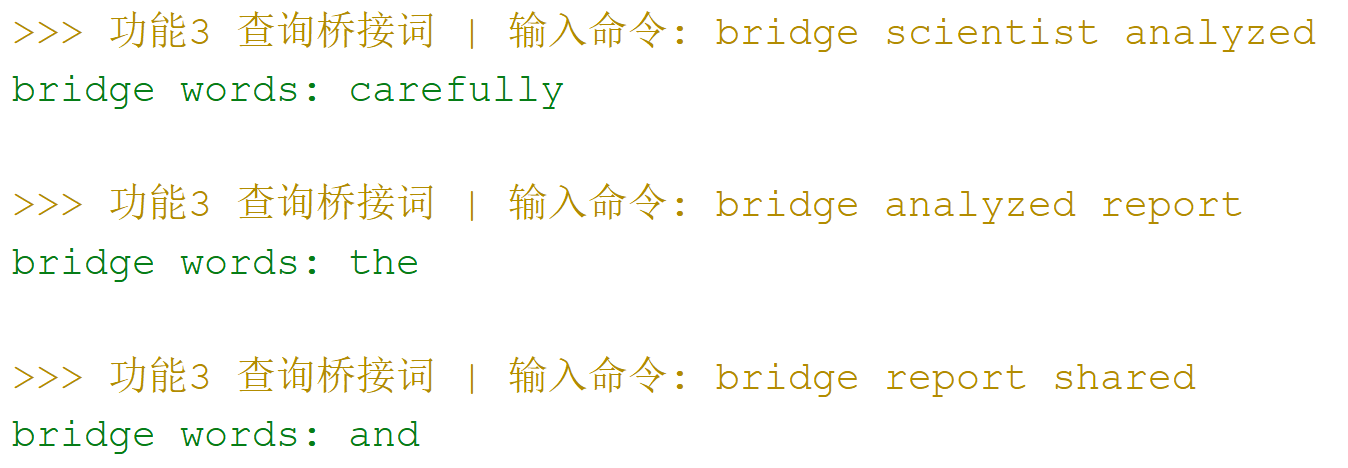
AI 生成的内容可能不正确。

文本, 信件

AI 生成的内容可能不正确。

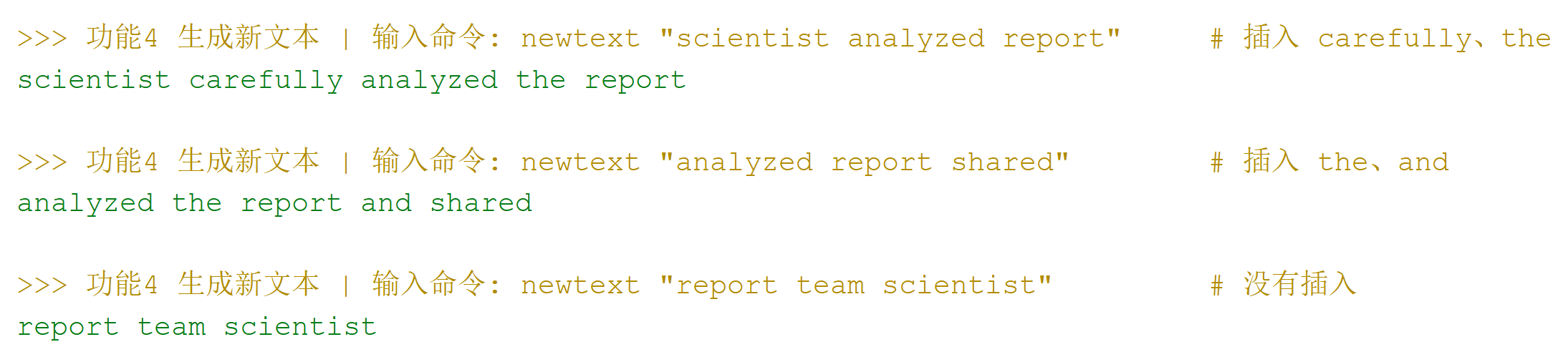
## 查询桥接词

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 输入（2个单词） | 期望输出 | 实际输出 | 运行是否正确 |
| 1 | scientist analyzed | carefully | carefully | √ |
| 2 | analyzed report | the | the | √ |
| 3 | report shared | and | and | √ |



## 根据桥接词生成新文本

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 输入（一行文本） | 期望输出 | 实际输出 | 运行是否正确 |
| 1 | scientist analyzed report | scientist carefully analyzed the report | scientist carefully analyzed the report | √ |
| 2 | analyzed report shared | analyzed the report and shared | analyzed the report and shared | √ |
| 3 | report team scientist | report team scientist | report team scientist | √ |



## 计算最短路径

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 输入（两个单词、或一个单词） | 期望输出 | 实际输出 | 运行是否正确 |
| 1 | scientist again | dist=3: scientist -> analyzed -> it -> again | dist=3: scientist -> analyzed -> it -> again | √ |
| 2 | scientist team | dist=4: scientist -> analyzed -> the -> team | dist=4: scientist -> analyzed -> the -> team | √ |
| 3 | scientist # 单源：到所有节点 | 见下表 | 与下表一致 | √ |
| 4 | alpha omega  # 不可达示例 | unreachable | unreachable | √ |

scientist的单源路径表：

scientist -> carefully dist=1: scientist -> carefully

scientist -> analyzed dist=1: scientist -> analyzed

scientist -> the dist=2: scientist -> analyzed -> the

scientist -> it dist=2: scientist -> analyzed -> it

scientist -> again dist=3: scientist -> analyzed -> it -> again

scientist -> data dist=3: scientist -> analyzed -> the -> data

scientist -> report dist=3: scientist -> analyzed -> the -> report

scientist -> team dist=4: scientist -> analyzed -> the -> team

scientist -> wrote dist=4: scientist -> analyzed -> the -> data -> wrote

scientist -> so dist=4: scientist -> analyzed -> the -> data -> so

scientist -> and dist=4: scientist -> analyzed -> the -> report -> and

scientist -> with dist=4: scientist -> analyzed -> the -> report -> with

scientist -> shared dist=5: scientist -> analyzed -> the -> report -> and -> shared

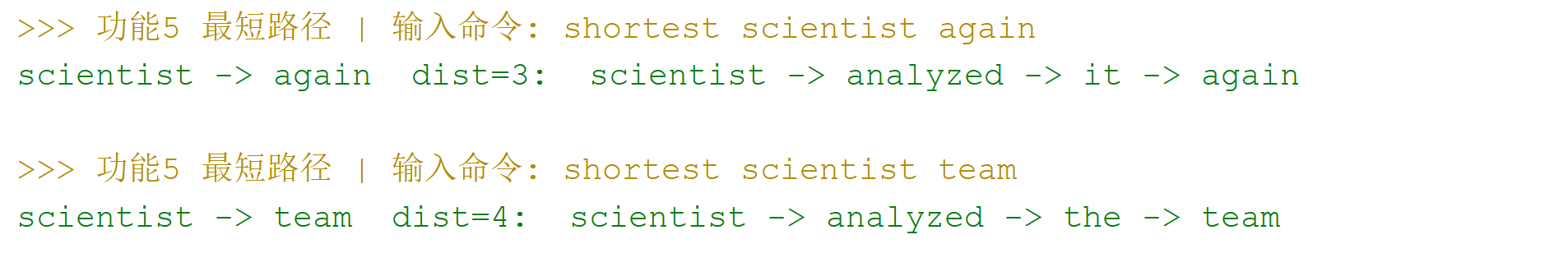
scientist -> but dist=5: scientist -> analyzed -> the -> team -> but

scientist -> requested dist=5: scientist -> analyzed -> the -> team -> requested

scientist -> a dist=5: scientist -> analyzed -> the -> data -> wrote -> a

scientist -> detailed dist=6: scientist -> analyzed -> the -> data -> wrote -> a -> detailed

scientist -> more dist=6: scientist -> analyzed -> the -> team -> requested -> more



文本, 信件

AI 生成的内容可能不正确。

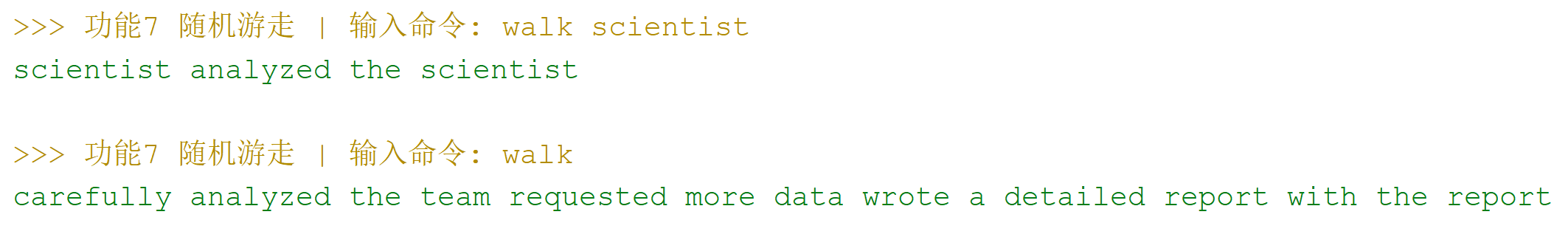
## 计算PageRank值

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 单词 | 期望输出 | 实际输出 | 运行是否正确 |
| 1 | the | 0.1351 | 0.1351 | √ |
| 2 | a | 0.0359 | 0.0359 | √ |
| 3 | but | 0.0214 | 0.0214 | √ |

## 随机游走

该功能无输入，让你的程序执行多次，分别记录结果。

|  |  |  |
| --- | --- | --- |
| 序号 | 实际输出 | 程序运行是否正确 |
| 1 | scientist analyzed the scientist | √ |
| 2 | the report with the | √ |
| 3 | carefully analyzed the scientist carefully | √ |



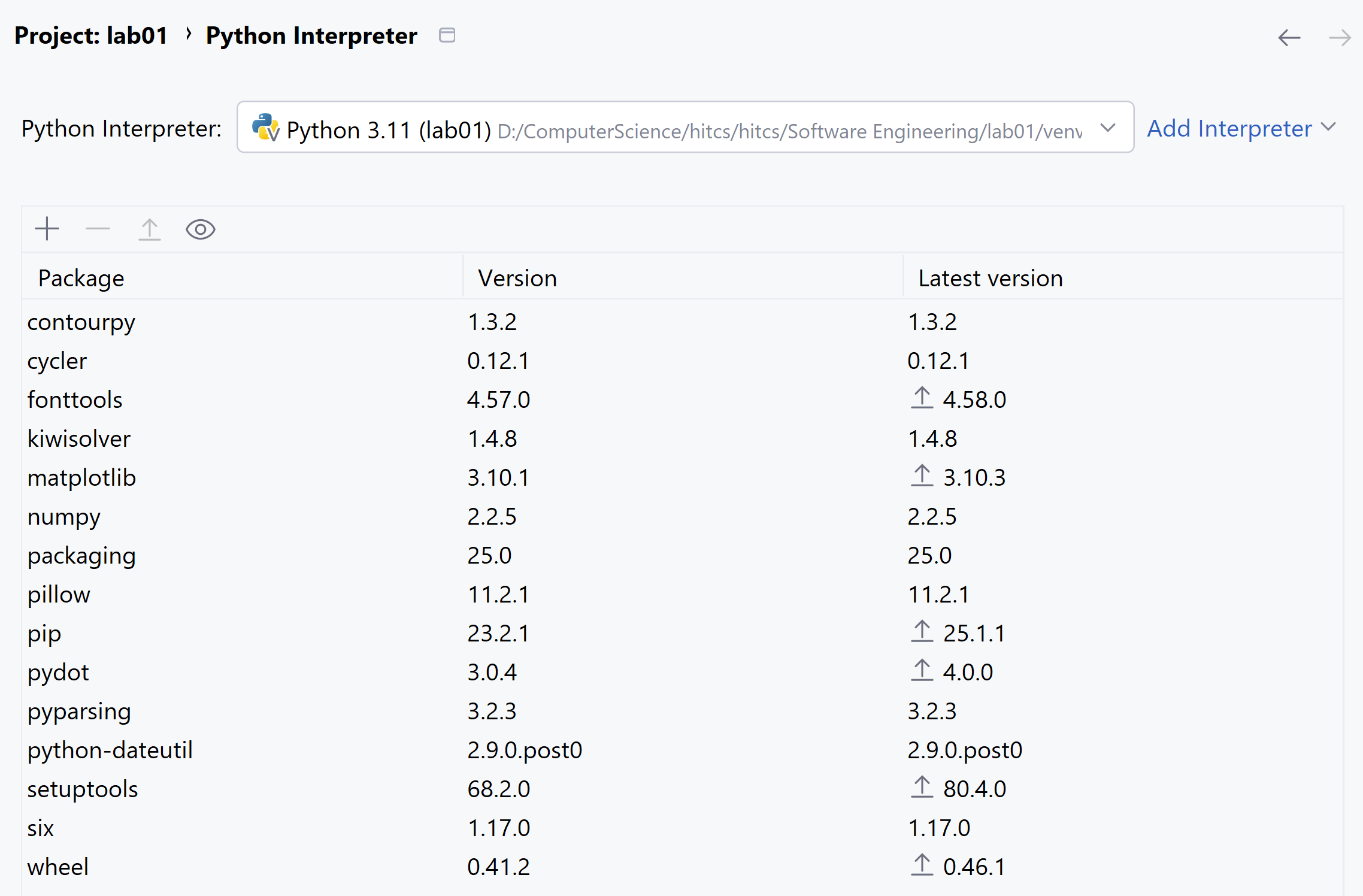
文本

AI 生成的内容可能不正确。

# 编程语言与开发环境

Python版本：Python 3.11，IDE版本：Jetbrains Pycharm 2024.1.1

依赖包/库函数：



采用的大模型：

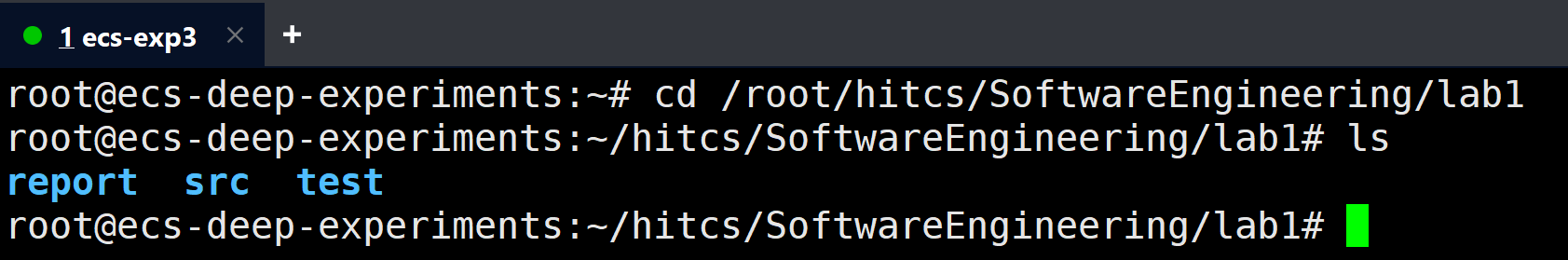
徽标

AI 生成的内容可能不正确。

# Git操作过程

## 实验场景(1)：仓库创建与提交

进入项目目录：



**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；**

文本

AI 生成的内容可能不正确。

图形用户界面, 文本

AI 生成的内容可能不正确。

第一次提交：§ **R2：提交；**

文本

AI 生成的内容可能不正确。

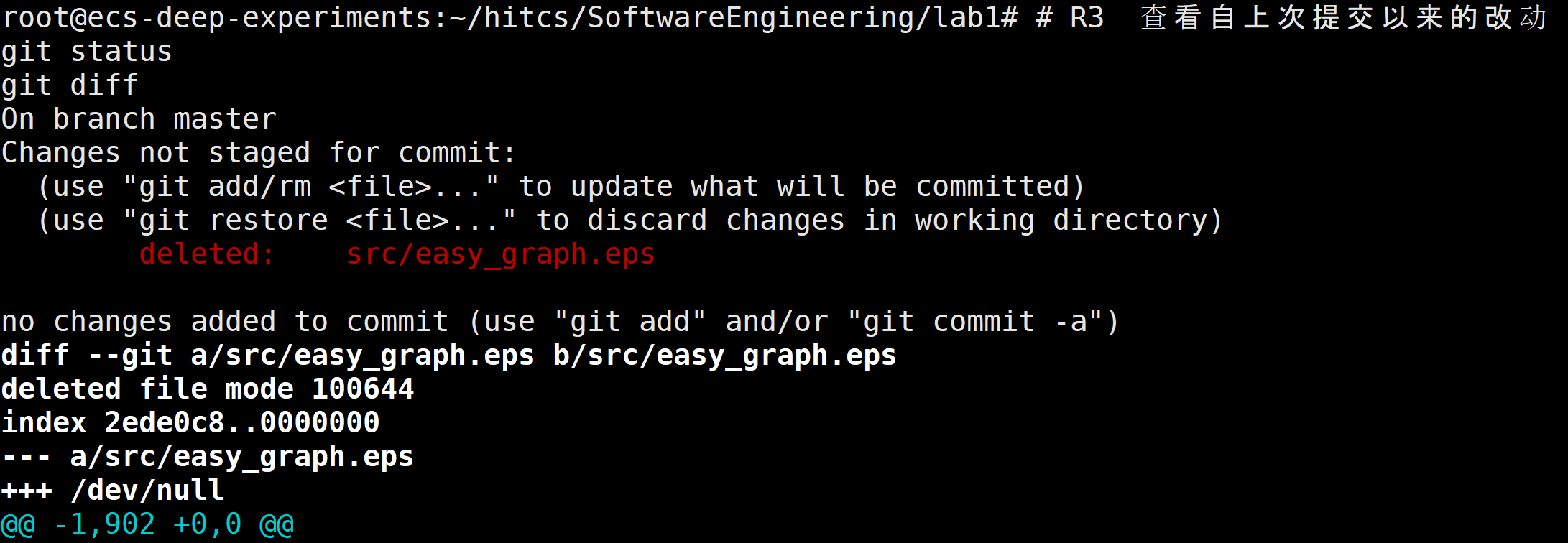
§ **R0：在进行每次git操作之前，随时查看工作区、暂存区、git仓库的**

**状态，确认项目里的各文件当前处于什么状态；**



提交后，第一次修改，删除了一个eps文件：**手工对提交的部分文件进行修改；**

§ **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；**



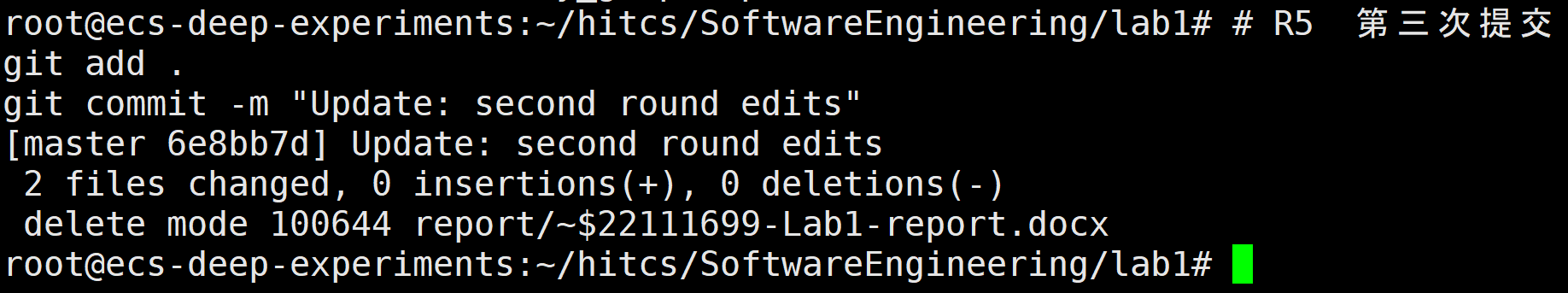
第二次提交：**R4：重新提交；**

文本

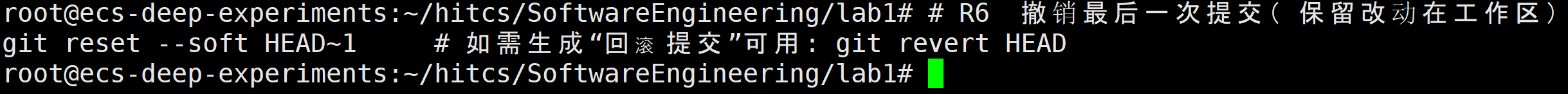
AI 生成的内容可能不正确。

第二次修改：删除了一个临时文件，修改了一个.docx文件。**再次对部分文件进行修改；**

第三次提交：§ **R5：重新提交**



§**R6：把最后一次提交撤销；**

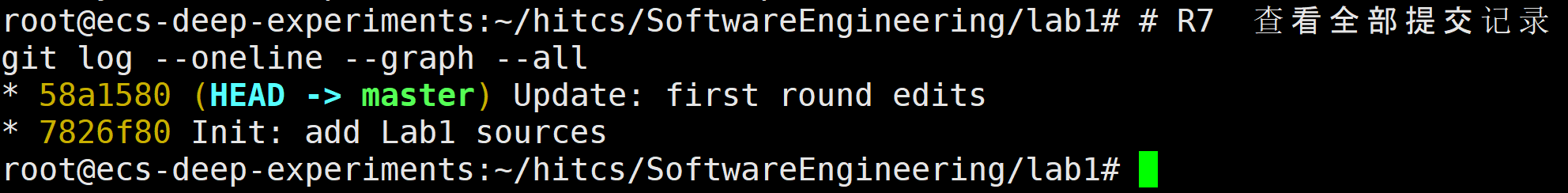


查看状态：§ **R0：在进行每次git操作之前，随时查看工作区、暂存区、git仓库的**

**状态，确认项目里的各文件当前处于什么状态；**



§**R7：查询该项目的全部提交记录；**

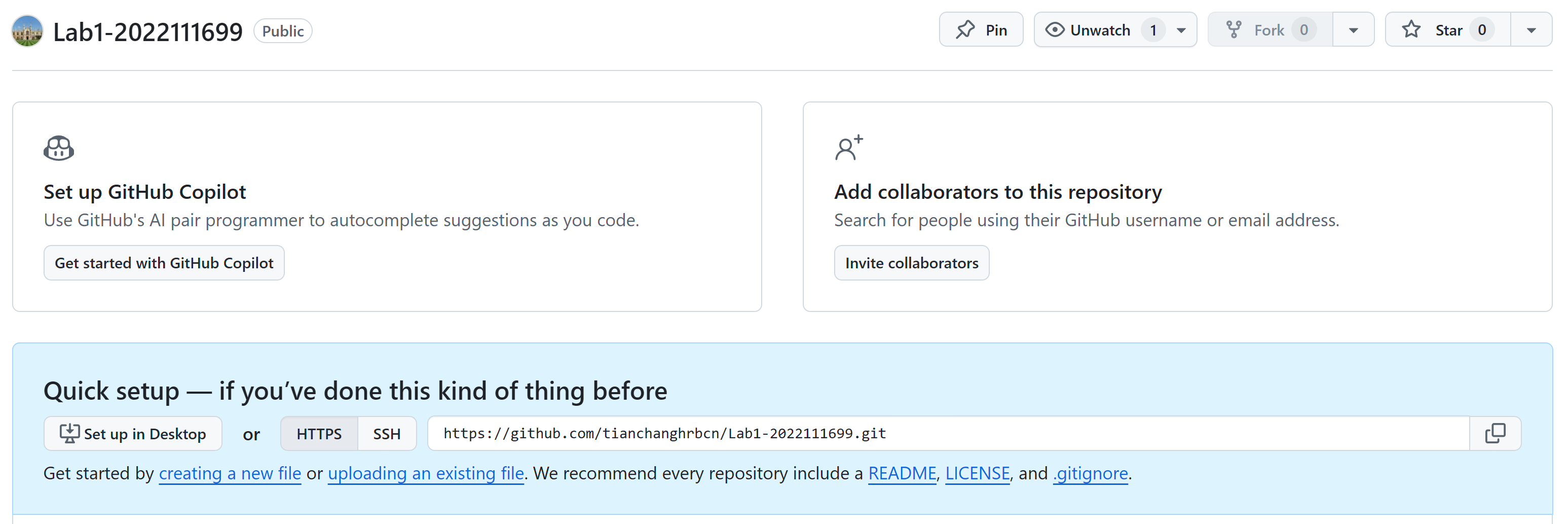


§**R8：在GitHub上创建名为“Lab1-学号”的仓库，并在本地仓库建立相**

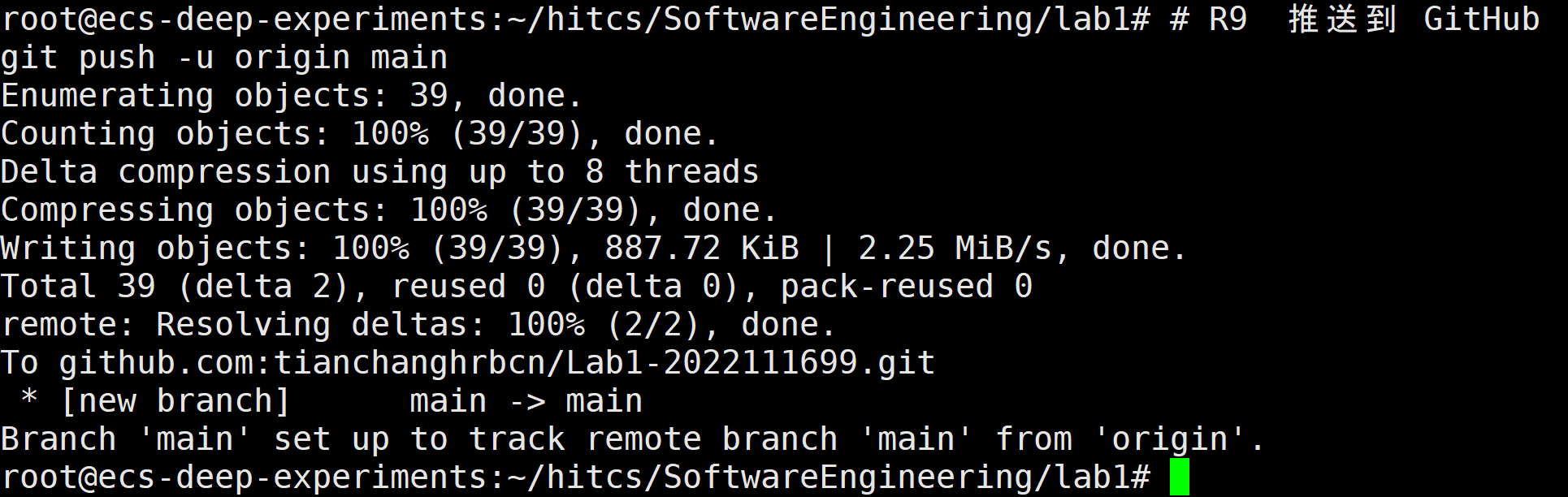
**应的远程仓库；**

文本

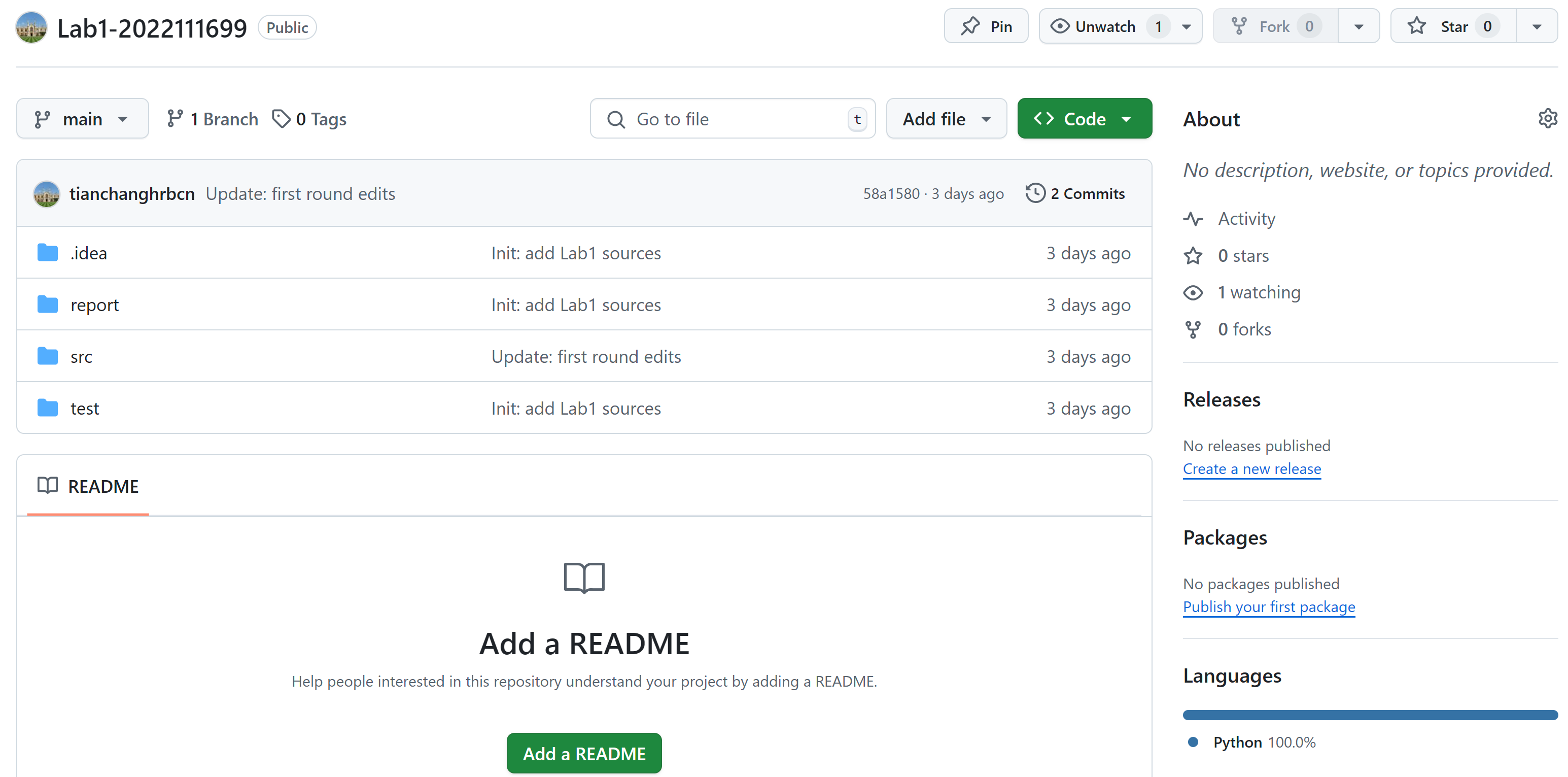
AI 生成的内容可能不正确。



§**R9：将之前各步骤得到的本地仓库全部内容推送到GitHub的仓库中；**

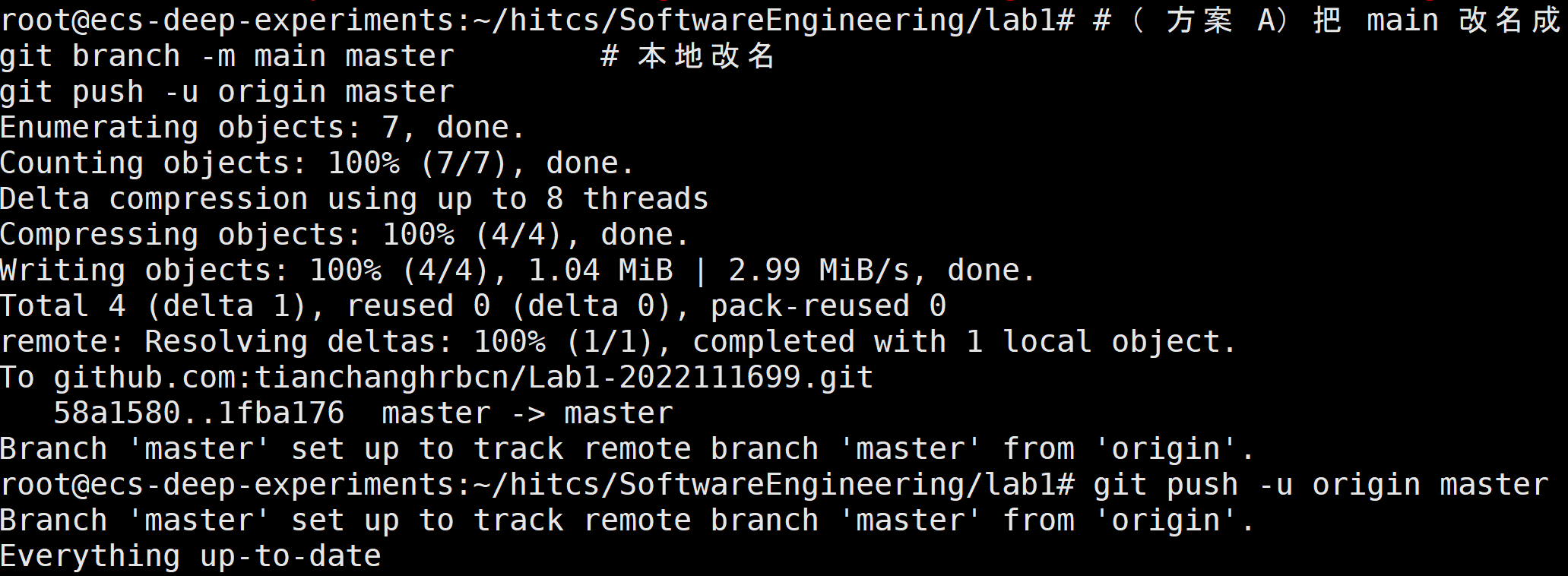
****

**仓库界面截图：**

****

## 实验场景(2)：分支管理

从场景1继续，由于我的github默认分支是main，所以先在本地和远程改分支名：



图形用户界面, 文本, 应用程序

AI 生成的内容可能不正确。

接下来按照步骤进行：

**§ R1：获得本地Lab1仓库的全部分支，切换至分支master；**

文本

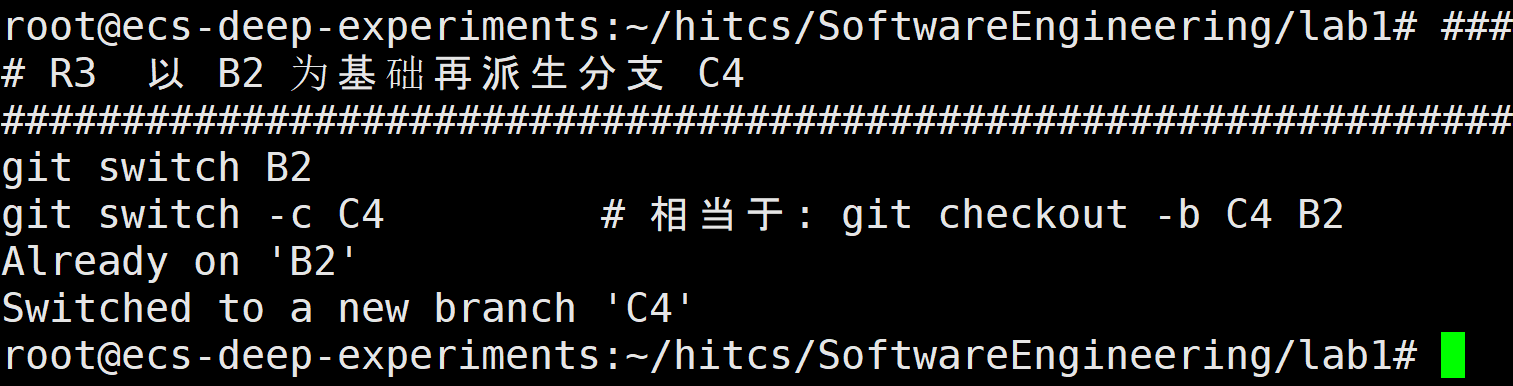
AI 生成的内容可能不正确。

§ **R2：在master基础上建立两个分支B1、B2；**

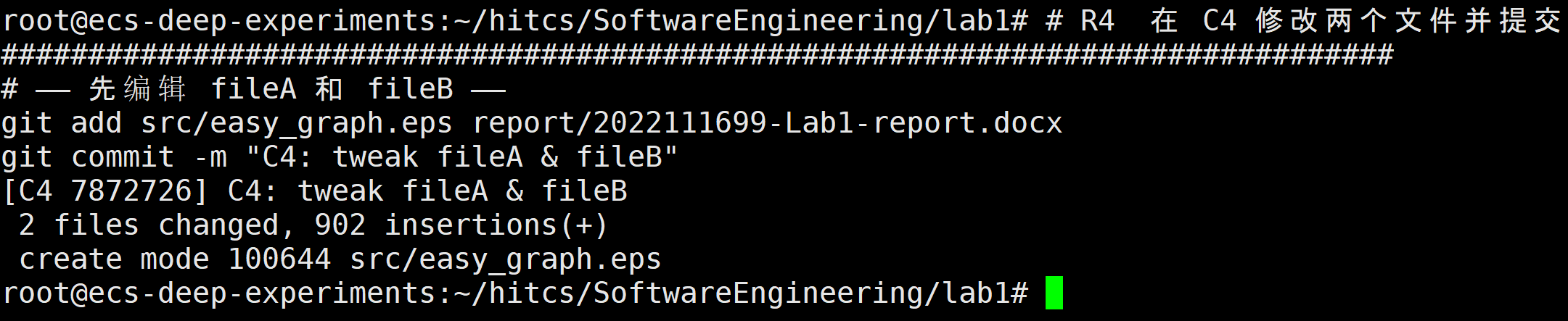
文本

AI 生成的内容可能不正确。

§ **R3：在B2分支基础上创建一个新分支C4；**



§ **R4：在C4上，对2个文件进行修改并提交；**



§ **R5：在B1分支上对同样的2个文件做不同修改并提交；**

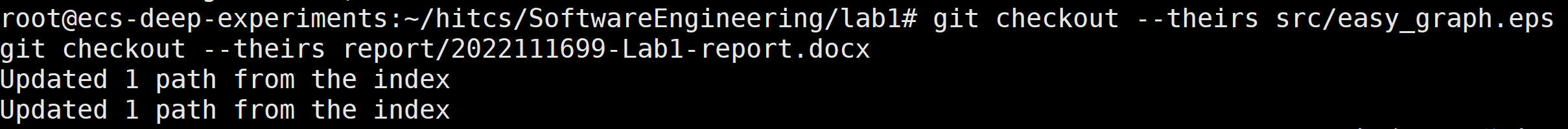


§ **R6：将C4合并到B1分支，若有冲突，手工消解；**

文本

AI 生成的内容可能不正确。

修改冲突：

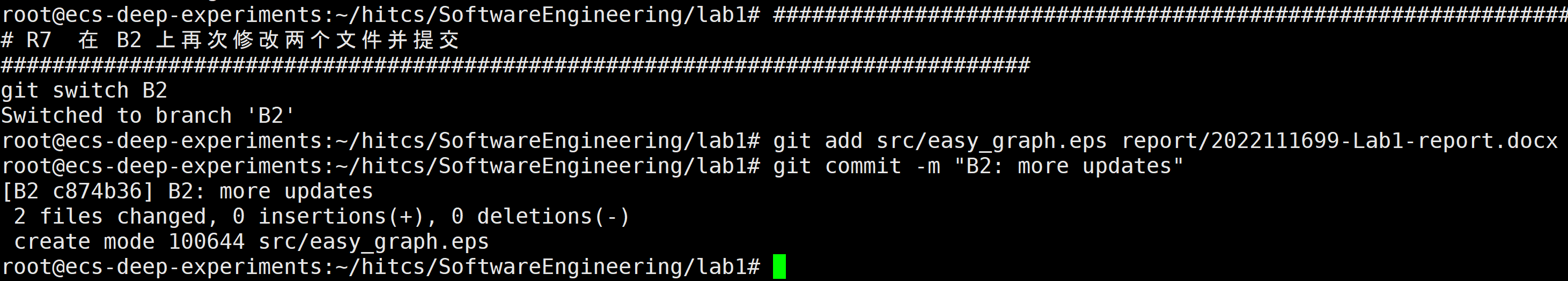


再次提交：

文本

AI 生成的内容可能不正确。

§ **R7：在B2分支上对2个文件做修改并提交；**



§ **R8：查看目前哪些分支已经合并、哪些分支尚未合并；**

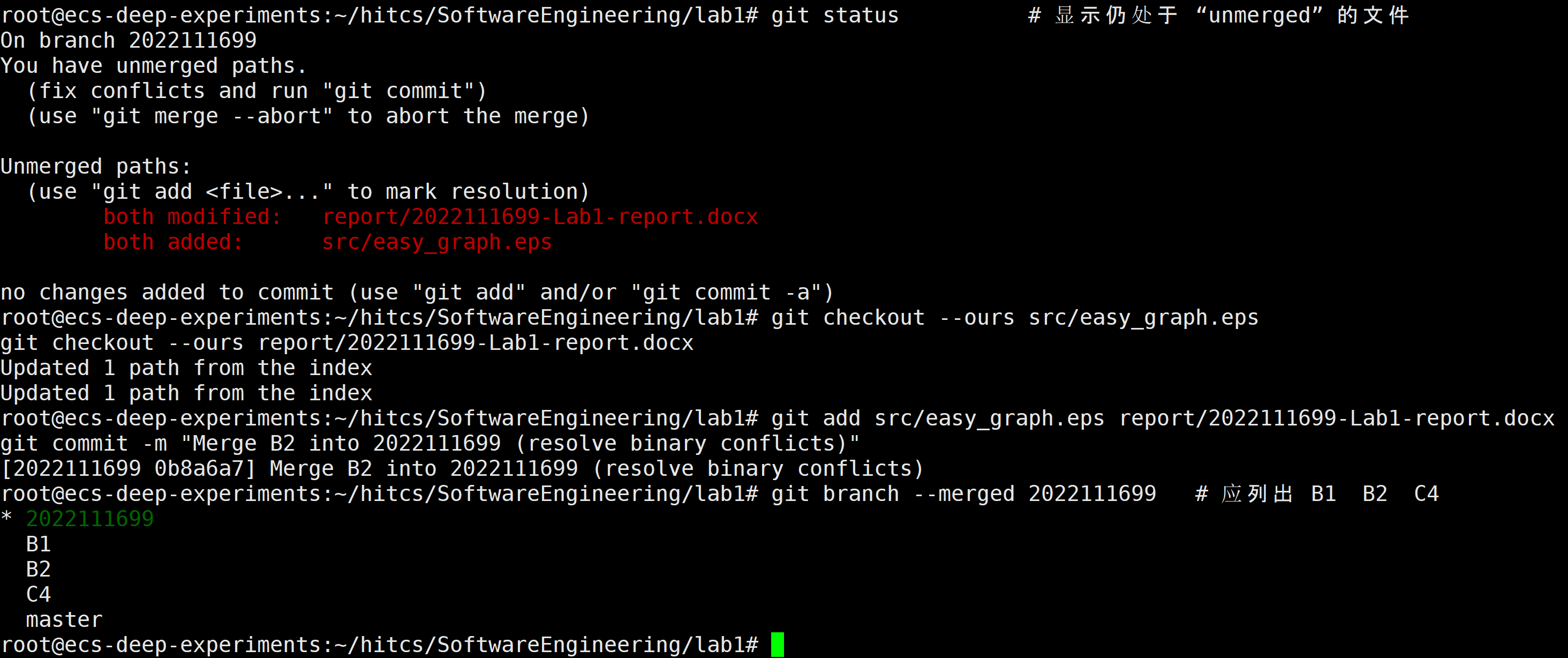
文本

AI 生成的内容可能不正确。

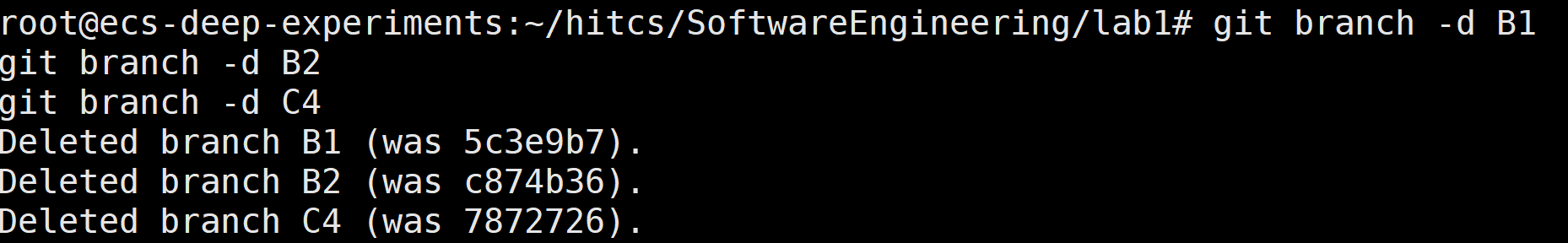
§ **R9：将已经合并的分支删除，将尚未合并的分支合并到一个新分支上，**

**分支名字为你的学号；**

**首先检查分支状态，处理每个冲突的文件，确定需要哪个版本：**



删除多余的分支：

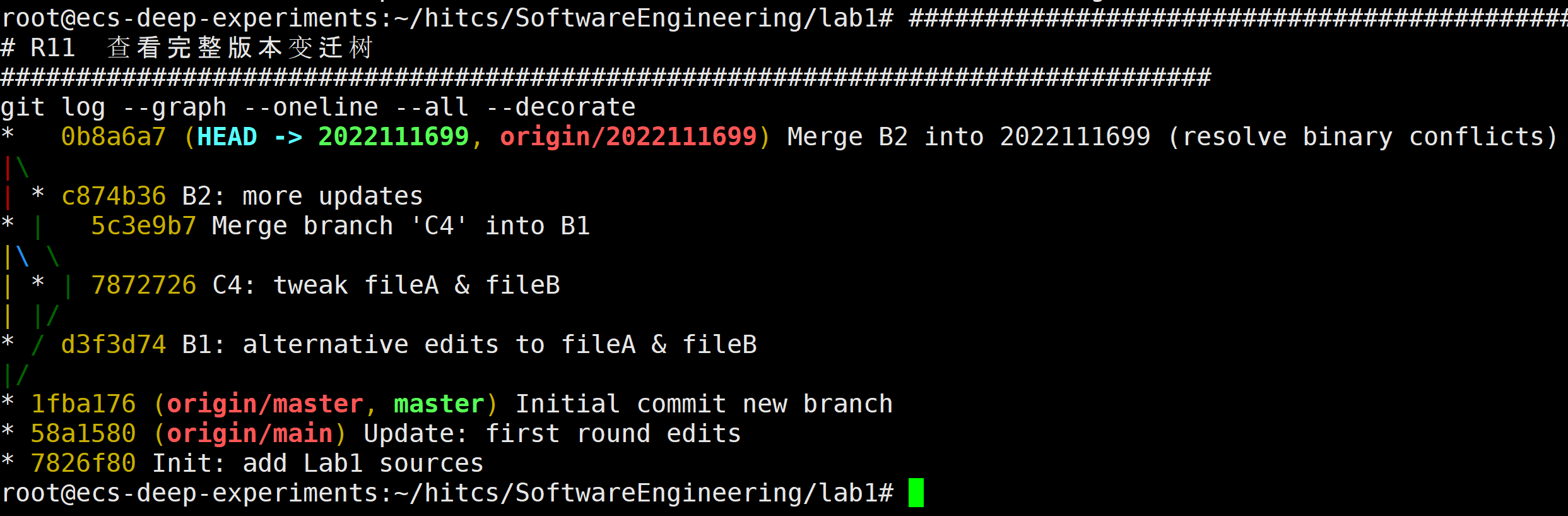


§ **R10：将本地以你的学号命名的分支推送到GitHub上自己的仓库内；**

文本

AI 生成的内容可能不正确。

§ **R11：查看完整的版本变迁树；**



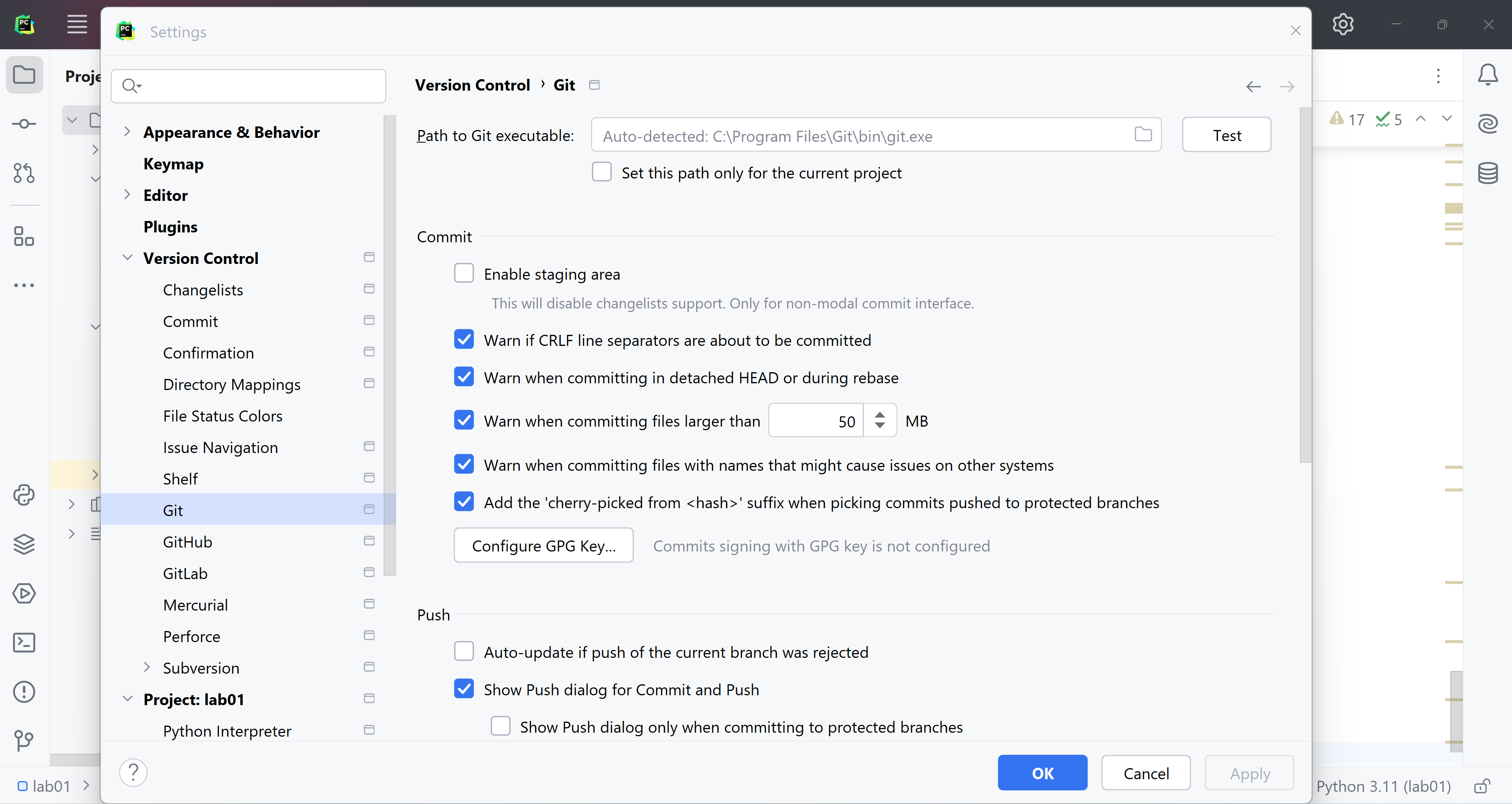
§ **R12：在Github上以web页面的方式查看你的Lab1仓库的当前状态。**

图形用户界面, 文本, 应用程序, 电子邮件

AI 生成的内容可能不正确。

# 在IDE中使用Git Plugin

**在Pycharm 2024中打开git设置，并点击OK：**

****

**Ctrl+K打开commit界面：**

**图形用户界面, 文本, 应用程序

AI 生成的内容可能不正确。图形用户界面, 文本, 应用程序

AI 生成的内容可能不正确。**

**可以发现有3个文件added，2个文件modified，点击Commit完成提交，点击Commit and Push完成提交并推送。**

****

**推送成功。**

# 小结

**关键体会**

**LLM 最擅长“生成草稿”：速度快、覆盖面广，但不能省掉人工验证。**

**Prompt 质量 = 结果质量：将需求拆得越细、越结构化，模型输出越贴合。**

**人机协同最佳模式:**

**模型：批量产出模板 / 提供灵感**

**人类：做规范统一、错误修复、性能 Benchmark、最终把关**

**结论: 大模型让“原型—验证—改进”的迭代周期大幅缩短，并在文档与可视化部分节省了至少 50 %的时间；但在正确性、鲁棒性和工程化方面仍需人工主导。合理利用其“加速器”角色，而不是完全替代开发者，是目前最有效的策略。**