实验一

班级： 21计科02 学号： B20210302211 姓名： 刘鑫 github: https://github.com/leonidluo/python_course

实验目的

1. Git基础，使用Git进行版本控制
2. Markdown基础，使用Markdown进行文档编辑

实验环境

- Git
- VSCode
- VSCode插件

实验内容

实验环境安装

1. 安装git
2. 在vscode中安装git与markdown相关插件

git基础

1. `git commit`
    - 用法：`git commit -m "提交消息"`
    - 作用：将暂存区中的更改提交到本地仓库，并创建一个新的提交记录。
2. `git branch bugFix`
    - 用法：`git branch <分支名>`
    - 作用：创建一个名为 "bugFix" 的新分支。
3. `git checkout bugFix`
    - 用法：`git checkout <分支名>`
    - 作用：切换到名为 "bugFix" 的分支，使你可以在该分支上进行操作。
4. `git merge bugFix`
    - 用法：在主分支上执行 `git merge <分支名>`，通常是在主分支上执行该命令。
    - 作用：将名为 "bugFix" 的分支合并到当前所在的分支中（通常是主分支），将 bugFix 分支的更改合并到当前分支中。
5. `git rebase main`
    - 用法：在 bugFix 分支上执行 `git rebase <目标分支名>`。
    - 作用：将当前分支（bugFix）的提交记录重新应用在 "main" 分支的顶部。相当于在 main 分支上进行变基操作，以包括 bugFix 分支的更改。

6. `git checkout bugFix^`
    - 用法：`git checkout <提交哈希值>^`
    - 作用：将 HEAD（当前分支）移动到 bugFix 分支的父提交上。即切换到 bugFix 分支最近的父提交。
7. `git branch -f bugFix head~2`
    - 用法：`git branch -f <分支名> <目标提交>`
    - 作用：将名为 "bugFix" 的分支强制设置为距离当前提交的父提交的第二层祖先。
8. `git reset HEAD^`
    - 用法：`git reset <参考位置>`
    - 作用：将当前分支的 HEAD（最新提交）移动到前一个提交上，取消最新的提交，并将更改保留在工作区。
9. `git revert HEAD`
    - 用法：`git revert <提交号>`
    - 作用：创建一个新的提交，撤销先前的提交（通过特定的提交号或引用），并将撤销的更改应用到当前分支上。

markdown基础

1. 标题
    - 用法：在文本前面添加 1 到 6 个 `#`，代表不同级别的标题。
    - 作用：用于标识标题的层级结构。
2. 粗体和斜体
    - 用法：使用 `**文本**` 表示**粗体**，使用 `*文本*` 或 `_文本_` 表示*斜体*。
    - 作用：强调或突出显示文本内容。
3. 链接
    - 用法：使用 `[链接文本](链接URL)` 表示链接。
    - 作用：在文本中添加超链接。
4. 列表
    - 用法：使用 `-` 或 `*` 开头表示无序列表，使用数字和 `.` 表示有序列表。
    - 作用：展示项目或事项的清单。
5. 引用
    - 用法：使用 `>` 开头表示引用
    - > 引用。
    - 作用：引用他人的内容或进行引用注释。

6. 代码块
    - 用法：使用三个 ``` 包裹代码块，或者在行内使用反引号 ` 表示代码。

```
git init
```

    - 作用：展示代码片段或内联代码。
7. 图片
    - 用法：使用 `![替代文本](图片URL)` 表示图片。
    - 作用：引入并显示图片。
8. 水平线
    - 用法：使用三个或更多的连字符 `---`、星号 `***` 或下划线 `___` 表示水平线。
    - 作用：在文档中创建水平分隔线。
9. 表格
    - 用法：使用 `|` 进行表格的列分隔，用 `-` 表示表头和表格内容之间的分隔行。
    - 作用：创建简单的表格。

实验问答

1. 版本控制是一种记录文件变更历史的系统，它可以追踪文件的修改、删除和添加，并提供回滚、协作和版本管理等功能。Git 是一种分布式版本控制系统，它具有以下优点：
    - 每个开发人员都可以在本地拥有完整的版本库，可以离线工作和进行本地提交，不需要依赖网络。
    - Git 提供灵活且强大的分支管理功能，可以轻松创建、切换、合并和删除分支，方便并行开发和版本控制。
    - 由于git每次的变更都会生成一个完整的文件快照，所以它非常快。用空间来换取时间。
2. 若要撤销还没有 Commit 的修改，可以使用以下命令：
    - `git checkout -- <文件名>`：撤销指定文件的所有修改。
    - `git checkout .` 或 `git checkout -- .`：撤销所有修改的文件。 注意：这些操作会丢失未提交的修改，请谨慎使用。

    若要检出已经以前的 Commit，可以使用以下命令：
    - `git checkout <Commit哈希值>`：检出指定的 Commit，并将 HEAD 指向该 Commit。
3. 在 Git 中，HEAD 是当前所在的分支指针，指向最近一次的提交。如果想让 HEAD 处于"detached HEAD"状态，可以使用以下命令：
    - `git checkout <Commit哈希值>`：检出指定的 Commit，并让 HEAD 进入"detached HEAD"状态。

4. 分支（Branch）是指向某个 Commit 的指针，它指向开发过程中的特定版本。可以使用以下命令创建和切换分支：
    - `git branch <分支名称>`：创建一个新的分支。
    - `git checkout <分支名称>`：切换到指定的分支。
5. 分支合并使用 `git merge` 命令进行合并操作，它将当前分支和目标分支的修改合并到一起。而 `git rebase` 命令是将当前分支的修改“衍合”到目标分支的最新提交之上，使提交历史变得更加线性整洁。
    - `git merge <目标分支>`：将目标分支合并到当前分支。
    - `git rebase <目标分支>`：将当前分支的修改衍合到目标分支。
6. 在 Markdown 格式的文本中使用标题、数字列表、无序列表和超链接的实际操作如下：
    - 标题：使用 `#` 符号表示标题的级别，例如 `# 标题一`、`## 标题二`。
    - 数字列表：使用数字和点号，例如 `1. 列表项一`、`2. 列表项二`。
    - 无序列表：使用 `-` 或 `*` 符号，例如 `- 列表项一`、`* 列表项二`。
    - 超链接：使用 `[链接文本](链接URL)` 的格式，例如 `[Google](https://www.google.com)`，得到Google。

实验总结

在学习**Git**使用方法和**Markdown**的实验中，我学会了如何使用Git进行版本控制和协作开发，以及如何使用Markdown语言进行文本编辑和格式化。通过Git，我可以轻松地创建和管理代码库，进行分支管理、合并和提交等操作，同时也能够与其他开发者进行协作开发。而Markdown则可以帮助我快速地创建格式化的文本，包括标题、列表、链接、图片等，使得文本更加易读易懂。总的来说，这次实验让我更加熟练地掌握了Git和Markdown的使用方法，为我的日后学习和工作打下了坚实的基础