实验一 Git和Markdown基础

班级： 21计科02

学号： B20210302210

姓名： 陶鑫

Github地址：https://github.com/Muyu-ikun/python-report.git

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

1. Git
2. VSCode
3. VSCode插件

实验内容和步骤k

第一部分 实验环境的安装

1. 安装git，从git官网下载后直接点击可以安装：git官网地址
2. 从Github克隆课程的仓库：课程的仓库地址，运行git bash应用（该应用包含在git安装包内），在命令行输入下面的命令（命令运行成功后，课程仓库会默认存放在Windows的用户文件夹下）

```
git clone https://github.com/zhoujing204/python_course.git
```

如果你在使用`git clone`命令时遇到SSL错误，请运行下面的git命令(这里假设你的Git使用了默认安装目录)：

```
git config --global http.sslCAInfo "C:/Program Files/Git/mingw64/ssl/certs/ca-bundle.crt"
```

或者运行下面的命令:

```
git config --global http.sslVerify false
```

该仓库的课程材料后续会有更新，如果需要更新课程材料，可以在本地课程仓库的目录下运行下面的命令：

```
git pull
```

3. 注册Github账号，创建一个新的仓库，用于存放实验报告和实验代码。
4. 安装VScode，下载地址：Visual Studio Code
5. 安装下列VScode插件
    - GitLens
    - Git Graph
    - Git History
    - Markdown All in One
    - Markdown Preview Enhanced
    - Markdown PDF
    - Auto-Open Markdown Preview
    - Paste Image
    - markdownlint

第二部分 Git基础

教材《Python编程从入门到实践》P440附录D：使用Git进行版本控制，按照教材的步骤，完成Git基础的学习。

第三部分 learngitbranching.js.org

访问learngitbranching.js.org，如下图所示完成Main部分的Introduction Sequence和Ramping Up两个小节的学习。


Learngitbranching.js.org

上面你学习到的git命令基本上可以应付百分之九十以上的日常使用，如果你想继续深入学习git，可以：

- 继续学习learngitbranching.js.org后面的几个小节（包括Main和Remote）
- 在日常的开发中使用git来管理你的代码和文档，用得越多，记得越牢
- 在git使用过程中，如果遇到任何问题，例如：错误删除了某个分支、从错误的分支拉取了内容等等，请查询git-flight-rules

第四部分 Markdown基础

查看Markdown cheat-sheet，学习Markdown的基础语法

使用Markdown编辑器（例如VScode）编写本次实验的实验报告，包括实验过程与结果、实验考查和实验总结，并将其导出为 **PDF格式** 来提交。

实验过程与结果

请将实验过程中编写的代码和运行结果放在这里，注意代码需要使用markdown的代码块格式化，例如Git命令行语句应该使用下面的格式：


Git命令

显示效果如下：

```
git init：在当前目录中初始化一个新的Git仓库。
git clone <仓库URL>：克隆一个远程仓库到本地。
git add <文件>：将文件添加到暂存区。
git commit -m "<提交信息>"：提交暂存区的文件到本地仓库。
git status：查看当前仓库的状态，包括已修改、已暂存和未跟踪的文件。
git log：查看提交历史记录。
git branch：查看本地分支。
git checkout <分支>：切换到指定的分支。
git merge <分支>：将指定分支的更改合并到当前分支。
git pull：从远程仓库拉取最新的变更。
git push：将本地仓库的变更推送到远程仓库。
git remote add <远程仓库名称> <仓库URL>：添加一个远程仓库。
git remote -v：查看当前仓库关联的远程仓库。
git diff：查看当前工作区与暂存区之间的差异。
git reset <文件>：将文件从暂存区移出，但保留在工作区。
git rm <文件>：从版本控制中删除文件
```

如果是Python代码，应该使用下面代码块格式，例如：

 Python代码

显示效果如下：

```
def add_binary(a,b):
    return bin(a+b)[2:]
```

代码运行结果的文本可以直接粘贴在这里。

注意：不要使用截图，Markdown文档转换为Pdf格式后，截图可能会无法显示。

实验考查

请使用自己的语言回答下面的问题，这些问题将在实验检查时用于提问和答辩，并要求进行实际的操作。

1. 什么是版本控制？使用Git作为版本控制软件有什么优点？

- 版本控制是一种记录和管理文件变更的系统。它可以帮助团队协同工作，跟踪文件的修改历史，恢复到以前的版本，解决冲突等。Git作为版本控制软件具有以下优点：
- 分布式：每个人都有完整的仓库副本，可以在本地进行工作，不依赖于网络连接。 高效：Git使用了一种增量存储和快照的方式来保存文件变更，使得存储和传输变得非常高效。
- 强大的分支和合并功能：Git提供了强大的分支和合并功能，使得团队可以并行工作，并将各个分支的变更合并到主分支中。 安全性：Git使用哈希值来标识文件和提交，确保数据的完整性和安全性。

2. 如何使用Git撤销还没有Commit的修改？如何使用Git检出（Checkout）已经以前的Commit？（实际操作）

- git checkout -- <文件>：撤销对文件的修改，恢复到最近一次的提交状态。
- git reset HEAD <文件>：将文件从暂存区移出，但保留在工作区。

- 要检出已经以前的提交，可以使用以下命令： git checkout <提交ID>：将工作区恢复到指定提交的状态。

3. Git中的HEAD是什么？如何让HEAD处于detached HEAD状态？（实际操作）

- git checkout <提交ID>：将HEAD指向指定的提交，进入detached HEAD状态。

4. 什么是分支（Branch）？如何创建分支？如何切换分支？（实际操作）

- git branch <分支名>：创建一个新的分支。

5. 如何合并分支？git merge和git rebase的区别在哪里？（实际操作）

- git merge <分支名>：将指定分支的更改合并到当前分支。
- git merge和git rebase的区别在于合并的方式不同：
- git merge：将两个分支的更改合并为一个新的提交，保留各自的提交历史。
- git rebase：将当前分支的更改应用到目标分支的最新提交上，形成一条线性的提交历史。

6. 如何在Markdown格式的文本中使用标题、数字列表、无序列表和超链接？（实际操作）

- 标题：使用 # 符号表示标题级别，例如 # 标题 表示一级标题。
- 数字列表：使用数字和点号表示，例如 1. 项目一 表示一个有序列表项。
- 无序列表：使用 - 或 * 符号表示，例如 - 项目一 表示一个无序列表项。
- 超链接：使用 链接文本 的格式表示超链接。

实验总结

在此次实验中，我学会了使用git来控制工程代码版本，使用git来管理我的代码，同时也有助于他人远程协助修改代码，我学会了使用git bash来连接远程仓库同时上传代码到我的仓库，可以更好的保存自己的代码不用担心意外损坏导致工程功亏一篑。