实验一 Git和Markdown基础

班级： 21计科02

学号： B20210302226

姓名：刘培钰

Github地址：https://github.com/kapeibala/demo.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

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
```

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

第二部分 GIT基础

D1 配置git

Git命令

```
git config --global user.name "shixiaoxiya"
git config --global user.email "shixiaoxiya@example.com"
```

D2 创建项目

创建一个要进行版本控制的项目。在系统中创建一个文件夹，将其命名为git_practice.在这个文件中，创建一个简单的Python程序：

```
print("Hello Git world")
```

D3 忽略文件

D4 初始化仓库

编写的Git代码

```
git init
```

D5 检查状态

编写的Git代码

```
git status
```

运行结果

```
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   .gitignore
        new file:   hello_git.py
```

D6 将文件加入仓库

编写的Git代码

```
git add .
git status
```

运行结果

```
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   .gitignore
        new file:   hello_git.py
```

D7 执行提交

编写的Git代码

```
git commit -m "Started project"
```

运行结果

```
[main (root-commit) e2dabc0] Started project
 2 files changed, 2 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 hello_git.py
```

编写的Git代码

```
git status
```

运行结果

```
On branch master
nothing to commit, working tree clean
```

D8 查看历史提交记录

编写的Git代码

```
git log
```

运行结果

```
Author: kapeibala <1011265239@qq.com>
Date:   Mon Sep 25 14:22:08 2023 +0800

    Started project
```

D9 第二次提交

编写的Git代码

```
git status
```

运行结果

```
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:   prac.py

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

D10 放弃修改

编写的Git代码

```
git restore .
git status
```

运行结果

```
On branch master
nothing to commit, working tree clean
```

D11 检出以前的提交

编写的Git代码

```
git log --pretty=oneline
git checkout cea13d
git switch
```

D12 删除仓库

编写的Git代码

```
git status
rm -rf .git/
git init
git status
git add .
git commit -m "Starting over."
 git status
```

第三部分 learngitbranching.js.org

一、基础篇

1. Git Commit
   编写的Git代码

```
git commit
git commit
```

2. Git Branch

```
git branch bugFix
git checkout bugFix
```

3. Git Merge

```
git checkout -b bugFix
git commit
git checkout master
git commit
git merge bugFix
```

4. Git Rebase

```
git checkout -b bugFix
git commit
git checkout master
git commit
git checkout bugFix
git rebase master
```

二、高级篇

1. 分离HEAD

```
git checkout c4
```

2. 相对引用（^）

```
git checkout bugFix^
或
git chekcout bugFix
git checkout HEAD^
```

3. 相对引用2（~）

```
git branch -f master c6
git branch -f bugFix c0
git checkout c1
```

4. 撤销变更

```
git reset HEAD^
git checkout pushed
git revert HEAD
```

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?

答:
版本控制是一种在开发的过程中管理源代码和文档的变更的系统。它可以帮助开发者跟踪代码和内容的历史版本,这样可以方便查看和恢复以前的版本。
Git 是一款流行的分布式版本控制软件,作为版本控制工具使用Git 主要有以下优点:

①分布式管理。Git是一个分布式的版本控制系统,开发者可以在本地建立一个代码仓库,可以离线在本地提交和管理

②强大的分支管理 。Git 提供强大的分支管理功能,可以创建新分支进行开发而不影响主分支,非常适合团队协作。

③强大的合并追踪能力。Git可以智能地处理复杂的合并操作,可以高效解决代码冲突。

④数据完整性保证。Git的数据所有存档使用SHA-1哈希算法计算,可以确保代码不被篡改。

⑤灵活的工作流模式 。Git支持各种工作流模式,可以非常灵活地定制开发流程。

⑥大量开源项目使用。包括Linux内核在内的大量开源项目使用Git进行版本控制。

⑦巨大的开源社区支持。Git拥有强大的社区支持,存在大量资源可供学习参考。

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）
   撤销未提交的修改

- 先修改文件,比如修改README.md

```
git add README.md
```

- 此时使用git status可以看到修改已被暂存

```
git status
```

- 使用git restore可以撤销修改

```
git restore README.md
```

- 再看git status,可以看到修改已被撤销
  检出以前的提交
  首先查看提交日志

```
git log
```

- 比如要检出上一个提交,找到其commit id

- 使用git checkout + commit id就可以把代码库检出到那个提交
- 再使用git log查看,当前状态已经是之前的提交了

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

- Git中的HEAD表示当前工作区所在的提交版本。
- 首先确保工作区没有未提交的修改:

```
git status
```

- 查看当前HEAD指向的提交版本:

```
git log
```

- 找到要检出的过去版本的commit id
- 此时再查看git status,可以看到HEAD处于detached状态:

```
HEAD is now at
```

3. 什么是分支（Branch）？如何创建分支？如何切换分支？（实际操作）

创建和切换分支的具体操作:

- 首先查看当前所在分支:

```
git branch
```

- 这会列出所有分支,当前分支前会有*号标记。
- 创建新分支,命名为new-branch:

```
git branch new-branch
```

- 再次查看分支列表,可以看到新分支new-branch已创建。
  切换到新分支:

```
git checkout new-branch
```

- 现在HEAD切换到了new-branch,表示当前工作区在这个分支上。
  在new-branch上进行开发,并提交。
  完成后,可以通过git checkout master切回master分支。

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

合并分支

假设当前在master分支上:

- 创建新分支feature:

```
git checkout -b feature
```

- 切换回master分支

```
git checkout master
```

- 将feature合并到master上

```
git merge feature
```

git merge vs git rebase

- git merge 在master上创建一个新的合并提交
- git rebase 会把feature里的每个提交逐个playback到master上,重写项目历史
  rebase操作:
  在feature分支上的开发历史:A-B-C
  切换到master,执行:

```
git rebase feature
```

- 现在master上的历史变成:A'-B'-C',feature分支的开发成果直接加入master主线上

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#号可以表示不同级别的标题,例如:

```
# 一级标题
## 二级标题
### 三级标题
```

数字列表
使用数字加点可以创建数字列表,例如:

```
- 第一项
- 第二项
- 第三项
```

超链接
使用显示文本可以创建超链接,例如:

```
[点我去Google](https://www.google.com)
```

实验总结

总结一下这次实验你学习和使用到的知识，例如：编程工具的使用、数据结构、程序语言的语法、算法、编程技巧、编程思想。

在这次实验中,我学习和使用到了以下知识:

1. 编程工具Git的使用

- 学习了Git的基本概念和命令,如初始化仓库、添加和提交修改、查看提交历史等
- 在learngitbranching.js.org上通过可视化的方式进一步理解和掌握Git的分支、合并、回滚等概念
- 学习使用Git进行版本控制,可以系统地管理不同版本的代码

2. Markdown语言的基本语法

- 学习了Markdown的基本语法,如标题、列表、代码块、链接等格式
- 可以使用Markdown编写文档,所见即所得地生成格式化的文档
- 编写了本次实验报告的实验过程和结果、考查题回答等内容

3. 提高了学习新知识、新工具的能力

- 自主学习了Git和Markdown的语法和用法
- 通过练习来掌握新工具的使用技巧
- 在使用过程中积极查阅资料解决问题

4. 培养了良好的编程思维和习惯

- 注重编写可读性好、结构清晰的代码
- 养成使用版本控制系统管理代码的习惯
- 培养编写文档与代码同步的习惯

通过本次实验,我对版本控制和文档编写工具有了更深入的理解,也强化了自主学习新知识的能力。这些都是提高编程技能必不可少的环节,也为今后的项目实践奠定了基础。