实验一 Git和Markdown基础

班级： 21计科03

学号： B20210302317

姓名： 覃思敏

Github地址：https://github.com/xx12qq/Experiment.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

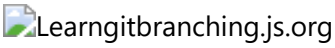
- GitLens
- Git Graph
- Git History
- Markdown All in One
- Markdown Preview Enhanced
- Markdown PDF
- Auto-Open Markdown Preview
- Paste Image
- markdownlint

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。

如何将markdown文件转换为pdf格式的文件？

- 安装vscode插件Markdown PDF，安装后重启vscode，打开markdown文件，按下Ctrl+Shift+P，输入Markdown PDF: Export (pdf)，回车即可导出pdf文件。
- 使用Google Chrome浏览器，在Github网站或者Gitee网站打开你的仓库，浏览你的markdown文件，按下Ctrl+P，选择打印，选择目标打印机为另存为PDF，点击保存即可导出pdf文件。

实验过程与结果

基础篇

1. 提交

```
- git commit
- git commit
```

2. 创建分支

```
- git branch bugFix //创建分支
- git commit
- git checkout bugFix //切换分支
- git commit
- git checkout -b bugFix //创建并切换
```

3. 合并方法一 git merge

```
git checkout -b bugFix //创建并切换
git commit
git checkout main //切换main分支
git commit
git merge bugFix 合并到main分支
```

1. 合并方法二 git rebase

```
git checkout -b bugFix
git commit
git checkout main //切换main分支
git commit
git checkout bugFix
git rebase main //合并到main
```

高级篇

1. 分离head

```
git checkout c4 //head指向c4
```

2. 相对引用

- 使用 ^ 向上移动 1 个提交记录
- 使用 ~ 向上移动多个提交记录，如 ~3

```
git checkout bugFix^
```

3. 相对引用2

```
git branch -f main C6
git checkout C1
git branch -f bugFiX HEAD^
```

1. 撤销变更 两种方法用来撤销变更: 1.git reset(本地)，2.git revert(远程) pushed 是远程分支，local 是本地分支

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

显示效果如下：

```
git init
git add .
git status
git commit -m "first commit"
```

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？
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）
3. Git中的HEAD是什么？如何让HEAD处于detached HEAD状态？（实际操作）
4. 什么是分支（Branch）？如何创建分支？如何切换分支？（实际操作）
5. 如何合并分支？git merge和git rebase的区别在哪里？（实际操作）
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）

答：

1.版本控制（Version Control）是一种记录文件内容变化，以便将来查阅特定版本历史的系统。它对于在软件开发、文档撰写等领域非常重要，可以帮助团队协作、追溯问题和管理项目。

Git 是一种分布式版本控制系统，它具有以下优点：

1. **分布式系统**：每个开发者都拥有完整的代码仓库，可以在本地进行版本控制，不依赖于中央服务器。这样就可以在没有网络连接的情况下工作，并且可以避免单点故障。

2. **快速**：Git 的设计使其非常高效，因此在处理大型项目时也能保持较快的速度。

3. **分支支持**：Git 非常强大的一个功能就是分支管理。开发者可以轻松地创建、合并、删除分支，这对于同时进行多个特性开发或修复问题非常有用。

4. **版本跟踪**：Git 可以精确地记录每个文件的修改历史，包括谁在什么时候做了哪些修改。

5. **易于协作**：多人协作时，可以轻松地将各自的工作合并到主分支中，而不会导致代码冲突。
6. **可扩展性**：Git 允许用户通过插件和定制脚本来扩展其功能。
7. **开源和广泛采用**：Git 是一个开源项目，有庞大的社区支持，同时也是许多项目中使用的首选版本控制系统。
8. **稳定性**：Git 经过了长时间的使用和测试，被认为是一个非常稳定可靠的版本控制系统。

总的来说，Git 是一个功能强大、高效、稳定且广泛采用的版本控制系统，适用于各种规模的项目和团队。

2. 当你在Git中需要撤销还没有提交（commit）的修改时，可以使用以下命令：

撤销未提交的修改：

1. **查看当前修改的状态：**

```
git status
```

2. **撤销单个文件的修改：**

```
git checkout -- <文件名>
```

这将会撤销对指定文件的修改，恢复到最后一次提交的状态。

3. **撤销所有未提交的修改：**

```
git checkout .
```

这将会撤销所有未提交的修改。

检出已经以前的Commit：

如果你想切换到之前的某一个提交（Commit），可以使用 `git checkout` 命令，但请注意，这会使你处于"分离头指针（detached HEAD）"状态，即你在一个不属于任何分支的状态下。

1. **查看所有的提交历史：**

```
git log
```

找到你想要切换的提交的哈希值（commit hash）。

2. **切换到特定的提交：**

```

```

```
git checkout <commit哈希值>
```

这会将你的工作目录和代码库切换到指定的提交状态。

如果你想创建一个新的分支来保留这个状态，可以使用以下命令：

```
git checkout -b <新分支名> <commit哈希值>
```

请务必注意，当你处于“分离头指针”状态时，任何新的提交将不会属于任何分支，所以你可能需要创建一个新的分支来保存你的工作。

注意：在使用 `git checkout` 命令时，请务必确保你没有未提交的修改，因为它会丢弃你当前的修改。如果你有未提交的修改，可以先使用 `git stash` 命令将其保存起来，然后再切换到其他提交。

3. 在Git中，`HEAD` 是一个特殊的指针，它指向当前工作目录所对应的提交（Commit）。换句话说，它表示你当前所在的代码状态。

当 `HEAD` 处于 "detached HEAD" 状态时，意味着它指向的是一个具体的提交，而不是一个分支。这种情况通常发生在你直接切换到一个特定的提交，而不是一个分支上。

下面是如何将 `HEAD` 置于 "detached HEAD" 状态：

1. **查看所有的提交历史：**

```
git log
```

找到你想要切换到的特定提交的哈希值（commit hash）。

2. **切换到特定的提交：**

```
git checkout <commit哈希值>
```

这会将你的工作目录和代码库切换到指定的提交状态。

如果你想创建一个新的分支来保留这个状态，可以使用以下命令：

```
git checkout -b <新分支名> <commit哈希值>
```

现在，`HEAD` 将会指向你选择的提交，处于 "detached HEAD" 状态。

注意：在 "detached HEAD" 状态下进行的修改不会属于任何分支，如果你想保留这些修改，你可以使用 `git checkout -b <新分支名>` 来创建一个新的分支。

4. 在Git中，分支（Branch）是指向某个特定提交（Commit）的一个引用，它可以让你在开发过程中同时处理多个独立的工作流。

以下是关于分支的操作：

1. 查看当前分支：

```
git branch
```

这会列出所有本地的分支，并在当前分支前加上一个星号。

2. 创建分支：

```
git branch <分支名>
```

这会创建一个新的分支，但不会自动切换到新创建的分支。

3. 切换分支：

```
git checkout <分支名>
```

这会让你切换到指定的分支。

或者可以将创建和切换分支合并成一个命令：

```
git checkout -b <分支名>
```

这会创建一个新分支并立即切换到该分支。

5. 在 Git 中，合并分支有两种常见的方法：使用 git merge 和使用 git rebase。git merge 用于将一个分支的更改合并到另一个分支中。它会创建一个新的合并提交，将两个分支的更改整合在一起。git rebase 用于将一个分支的更改应用到另一个分支上。它会将每个提交逐个应用到目标分支上，并重新创建这些提交。

git merge实际操作：

```
git checkout target_branch
git merge source_branch
```

git rebase实际操作：

```
git checkout source_branch
git rebase target_branch
```

6. 实际操作：

标题

```
# 一级标题
## 二级标题
### 三级标题
```

数字列表

```
1. 第一项
2. 第二项
3. 第三项
```

无序列表

```
- 第一项
- 第二项
- 第三项
```

超链接

```
[链接文字](链接地址)
```

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。

本次实验用到了git的基本用法，使用了git、git bash、markdown、vscode等工具，熟悉了git的基本操作，包括：创建仓库、克 隆仓库、提交修改、查看提交历史、创建分支、切换分支、合并分支等。