

# 实验一 Git和Markdown基础

班级： 21计科03

学号： B20210302321

姓名： 向钟源

Github地址： <[Ch1rs \(github.com\)](#)>

Gitee地址:<[词不达意 \(Ch1rs\) - Gitee.com](#)>

## 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）。然后在这里通过问同学，我发现windows的cmd也能用，不一定要用git bash。

```
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)：

这里的话是版本问题 我一开始也遇到了这个问题

```
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。

### 1. 安装git

我已经安装好了，可以通过在cmd中输入`git --version`查看当前电脑中的git版本。通过`git config --global user.name "username"`

和`git config --global user.email "username@example"`配置git,以便让git知道是谁修改了项目。最后，我们最好设置每个项目中的主分支的默认名称，比如取个名字叫main的，意为主分支。在cmd中输入 `git config --global init.defaultBranch main`来取名，这意味着在用git管理的每一个新项目中，一开始都只有一个分支，它的名字叫做main。

### 2.创建项目

我在桌面新建了一个文件夹，用来测试和学习git的基本功能，并在这个文件夹中创建了一个.py文件，里面的内容是：

```
print("wo shi zhu")
```

### 3.忽略文件

在我们创建并运行了.py文件后，文件夹中会自动生成.pyc文件，这些文件存在目录名为`__pycache__`中,但是我们不希望提交它，所以我们想办法要让git忽略它，怎么办呢？我们可以在文件夹中新建一个名为.gitignore的特殊文件，点开这个文件并在其中输入：

```
__pycahe__/
```

这个操作直接可以让git忽略pycahe中的所有文件，当然如果我们希望git还要忽略别的一些文件，只需要往.gitignore文件中添加相应文件夹即可。

### 4.初始化仓库

我们已经创建了一个用来实验git的文件夹，里面放了两个文件，一个.py文件、一个.gitignore文件，现在可以初始化一个仓库了。怎么弄呢？打开终端，用cd+我们实验git的文件夹的位置，比如我的是 `cd C:\Users\ROG\Desktop\Python_code_2023_class\git_practise`，用以切换到这个文件夹下，这样才能在这文件夹下创建一个仓库。

切换后 输入 `git init`来创建一个空仓库。**仓库** 是程序中被git主动跟踪的一组文件。

ps:git用来管理仓库的文件都储存再隐藏的目录.git中，没事别乱动它，更别删除，否则项目的所有历史记录全没了。

## 5.检查状态

在执行其他操作之前，需要检查一下项目的状态。那么怎么检查呢？接着上步操作，也就是还是在这文件夹下输入 `git status`。从输出结果我们知道，现在位于分支main上。每当我们检查项目的状态时，输出都将指出我们位于分支main上。接下来的Untracked files下显示红色的文件是表明这两个文件还没有进行任何的commit,commit是指项目在特定时间点的快照。这里的快照，我理解为特定时间点的状态吧。

## 6.将文件加入仓库

很简单，检查完状态后，将两个新文件添加就行。也就是在当前文件夹下的终端中输入 `git add .`，这里的意思是将项目中未被跟踪的所有文件(不包括.gitignore忽略的)都加入之前创建的空仓库中。需要注意的是`git add .`只是将new file 添加到仓库中并没有提交。

## 7.执行提交

怎么提交？在终端中且在当前文件夹目录下输入命令: `git commit -m "started project"`，这里的意思是创建项目的快照，意思是开始项目，你看后面""号里的英文也知道，-m的意思是让git将接下来的message --"started project"记录到项目的历史记录中去。输出表示 我们位于main分支上 且有两个files被修改了。在提交后在`git status`一下来检查一下项目的状态，可以发现这个时候的仍位于main分支上，且工作树is clean.

## 8.查看提交历史

怎么查看？很简单 在项目文件夹目录下的终端中输入命令: `git log`。每次提交git会自动生成一个specia的cite id ,length is 40 字符，它记录的提交的人和时间的，这也就是为什么第一步中我们输入了咱们user的name和email,但是这种log太长，那咋办呢？ git提供了一个简单版 输入名利`git log --pretty=oneline`，顾名思义只有一行，显示了提交的cite id和提交记录的消息。

## 9. 第二次提交

为什么要第二次提交？问得好 我们写的代码不是一次就写好了的，可能需要多次修改，那这时候我们就需要多次提交。以我们的测试代码为例，之前里面只有一句，现在我们加一句后里面的内容是：

```
print("wo shi zhu")
print("wo shi shabi")
```

然后我们在`git status`一下查看当前项目的状态。可以发现，modified：我们创建的.py文件这个东西，这东西说明这个文件被修改了。那么怎么二次提交这种修改呢？只需要输入命令：`git commit -am "Extended message."` 我们发现这里的标志从-m变成了-am，标志-a让git将仓库中所有修改了的文件都加入当前commit,-m还是一样让git在提交历史中记录一条message.

最后我们再用`git status` 查看一下项目的状态，可以发现working tree clean且下面的提交历史中有两个提交

## 10.放弃修改

有些时候，我们需要放弃上一次进行的修改，因为可能做出的改变是冗余的甚至是错误的，所以这个时候我们需要放弃修改恢复到上一个可行状态，那么怎么做呢？比如 我在.py文件中做了以下修改：

```
print("wo shi zhu")
print("wo shi shabi")
print("oh shit")
```

保存运行后，git status 查看一下项目状态，这里和上面一样，会有一个modified：我们创建的.py文件这个东西，如果我们愿意我们可以提交，但是这次我们不想提交，我们想返回项目的上一个状态，那么怎么搞呢？很简单 在终端中输入命令: git restore .就行。

接下来 我们再用git status 查看一下项目的状态。此时返回vscode后发现print("oh shit")没了

git restore filename 可以放弃最后一次commit后对指定文件的修改

git restore . 可以放弃最后一次commit后对所有文件的修改

## 11.检出以前的提交

如果我们要检出提交历史中的任何提交，输入命令git checkout 引用id的前六个字符就行。

检出以前的提交后，将离开main分支，进入git所说的detached HEAD状态。HEAD表示当前提交项目的状态，之所以叫做detached，因为离开了具体的一个branch，比如这里的main. 如果需要回到main 可以输入命令:git switch - 来撤销上一个操作。除非你是高手，否则在检出以前的提交后，最好不要对项目做出任何修改。如果开发人员只有你一个，你又想放弃最近的所有提交并恢复到之前的状态，你可以这么做：git status 查看状态 git log --pretty=oneline后发现HEAD在main上，但我们想在另外一个提交记录上，那么我们输入命令 git reset --hard cite id的前六位，然后再git status一下查看状态，发现HEAD在另外一次记录上了。

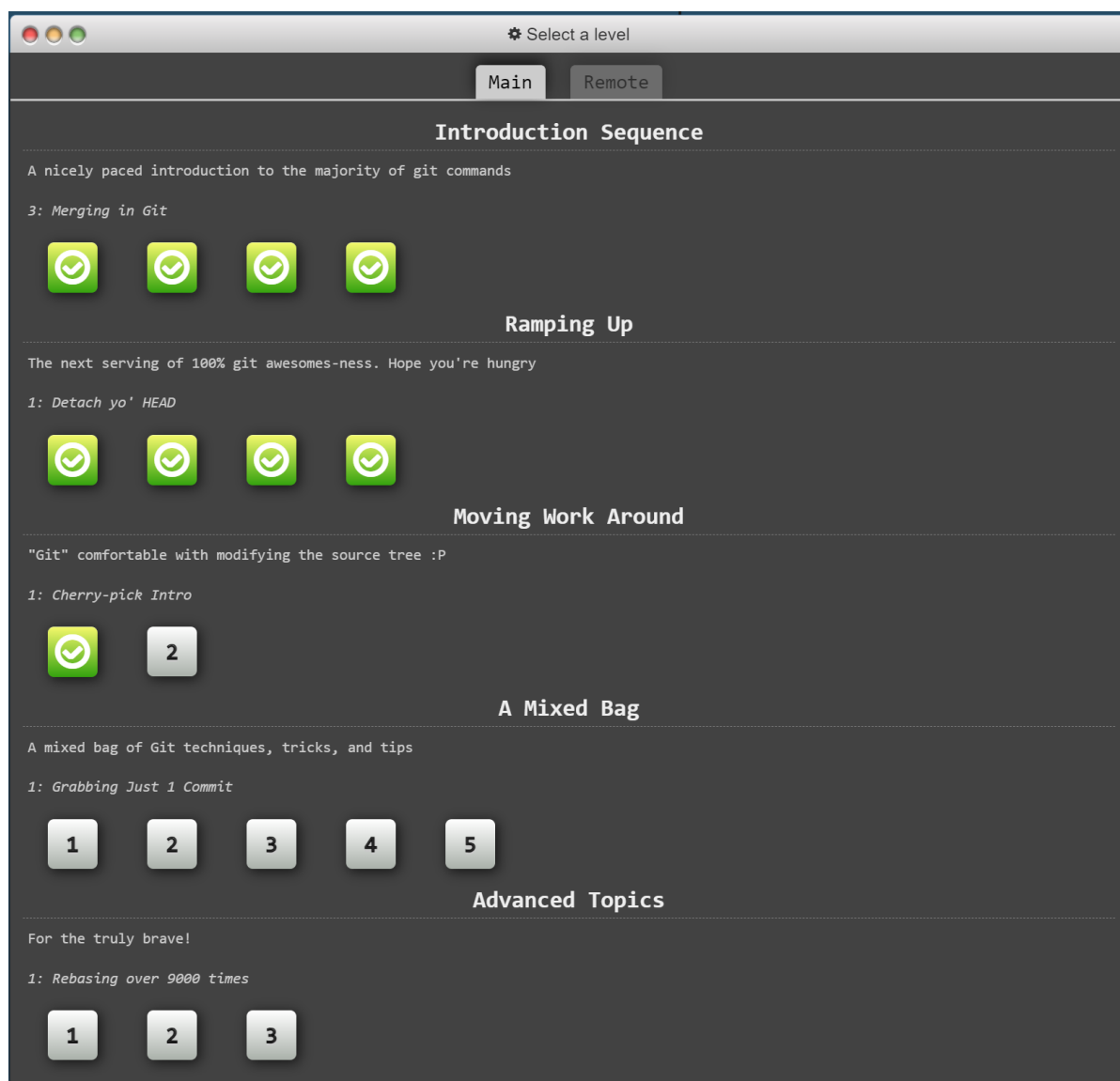
## 12.删除仓库

sometimes 仓库的历史记录乱了，我们又不知道怎么恢复，我们大力出奇迹，直接删了所有历史记录。怎么做呢？ git status查看状态后，输入命令 rm -rf .git/

来删除目录.git，再次查询状态后发现啥也没了。然后我们再git init创建一个新的空仓库并查看状态，发现又回到了初始状态，等待着第一次提交，后面的操作又回到了上述的一切操作，太tm神奇了。

## 第三部分 learngitbranching.js.org

访问[learngitbranching.js.org](https://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。



我的学习完成记录:



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](https://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](https://git-flight-rules.com)

## 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。

### 基本语法

#### 1.标题

通过#+空格+标题名字 创建一级标题，同理加N个#就可以创建N级标题

2.字体加粗

通过在需要加粗的字体的两侧分别加上\*\*即可 比如: **粗体**

3.字体变成斜体字体

通过在需要变成斜体的字体的两侧分别加上\*即可 比如: *斜体*

4.块引用

通过在需要引用效果的语句块前加上>+空格即可 比如:

```
> sdsdsdsdswdsd
```

5.有序列表

```
1. xxxxx
1. xxxxx
1. xxxxx
```

6.无序列表

- + 空格

- xxxxx
- xxxx
- xxxx

7.code

如果我们需要让很短的一个代码语句特殊的显示 可以通过在代码语句两侧+`实现 比如:

```
print
```

8.水平线


如果我们需要使用一些记号使得我们的文章的某两个部分分割开来 可以使用通过---+空格来使用水平线 比如:

```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
-----
XXXXXXXXXXXX
```

9.链接

有些时候我们需要放置一些外部的链接 可以通过[title](地址例如(<https://www.example.com>))来通过文章访问

10.图像

插入图像: titile

# 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:

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

显示效果如下：

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

我的实验过程中遇到的一些Git命令行语句：

书上的：

1.向git提供用户名和邮箱地址

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

2.初始化仓库

```
git init
```

3.检查项目状态

```
git status
```

4.将文件加入仓库

```
git add
```

5.提交

```
git commit
git commit -m "started project"
```

6.查看提交历史

```
git log
git log --pretty=oneline
```

7.再次提交修改了的文件



```
git commit -am "Extend greeting"
```

#### 8.放弃修改的文件

```
git restore filename  
git restore .
```

#### 9.检查以前提交的修改 cea13d 是通过git log --pretty=oneline查的的引用id的前6个字符

```
git checkout cea13d
```

#### 10.删除仓库

```
rm -f .git/
```

### learngitbranching网站上的

```
git commit  
git branch  
git checkout  
git merge  
git rebase  
git cherry-pick
```

如果是Python代码，应该使用下面代码块格式，例如：

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

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。

**作为提交git的实验所创建的python文件:**

```
print("wo shi zhu")  
print("wo shi yi tou zhu")
```

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：**版本控制就是对我们编写的代码进行管理和追踪，Git具有分布式、高效、强大的分支管理和合并功能，以及可靠的数据完整性等优点，使其成为开发团队中首选的版本控制软件。

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）

**答：**`git restore` 命令用来撤销没有提交的修改；先使用 `git log --pretty=oneline` 命令查一下引用ID，再用 `git checkout` 引用ID的前六位 查以前的提交

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

**答：**HEAD是一个对当前所在分支的符号引用，也就是指向我们正在其基础上进行工作的提交记录，HEAD总是指向当前分支上最近的一次提交记录；分离的HEAD就是让其指向了某个具体的提交记录而不是分支，所以我们输入名利 `git checkout` 引用id前六位

4. 什么是分支（Branch）？如何创建分支？如何切换分支？（实际操作）

**答：**Branch就是指向某个提交记录的有名指针；可以通过命令 `git branch new_branch_name` 创建分支；可以通过命令 `git checkout branch_name` 来切换分支

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

**答：**通过命令 `git merge branch_name` 来合并分支；`git merge` 在 Git 中合并两个分支时会产生一个特殊的提交记录，它有两个 parent 节点,而 `git rebase` 就是取出一系列的提交记录，“复制”它们，然后在另外一个地方逐个的放下去。`git rebase` 的优势就是可以创造更线性的提交历史,使得代码库的提交历史将会变得异常清晰。

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：**标题：#空格标题名(几级标题就用几个#); 数字列表：1.空格+内容后回车第二行自动2.; 无序列表：-空格+内容回车后第二行自动补全-空格；超链接：[titlie](#)

## 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。

- 下了vscode并且根据老师提供的帮助配置好了python的环境，闲的没事把pycharm也下了
- 学会了什么是git以及git的一些基本操作
- 学会了markdown的一些基本语法，可以用来写笔记写文档