实验一 Git和Markdown基础

班级： 21计科3

学号： B20200202217

姓名： 黄澍

Github地址：https://github.com/Hassium1/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

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
```

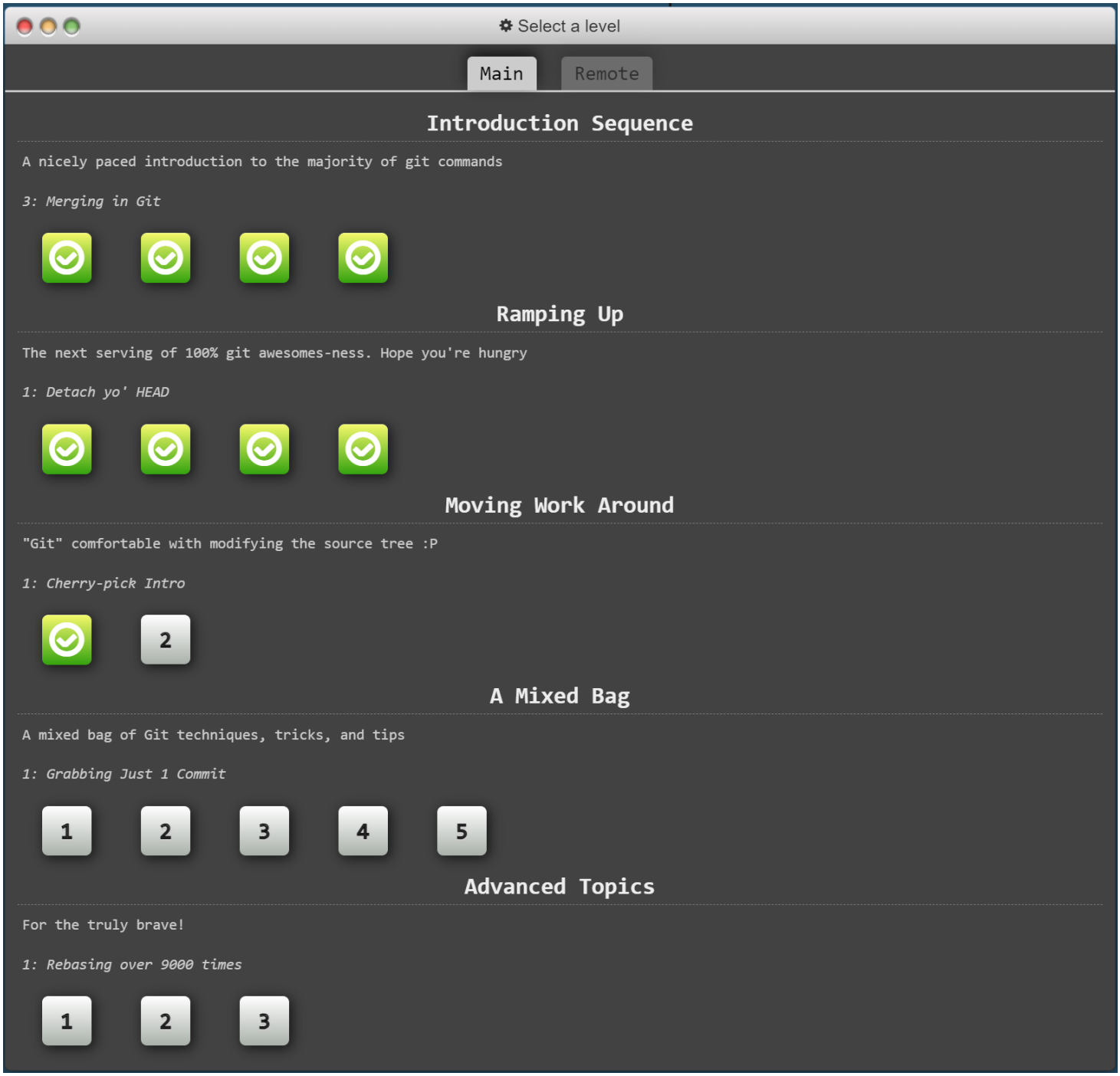
3. 注册Github账号，创建一个新的仓库，用于存放实验报告和实验代码。
4. 安装VScode，下载地址：Visual Studio Code
5. 安装下列VScode插件
    - GitLens
    - Git Graph
    - Git History
    - Markdown All in One
    - Markdown Preview Enhanced
    - Markdown PDF
    - Auto-Open Markdown Preview
    - Paste Image
    - markdownlint

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。



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

第二部分实验过程：

初始化仓库：

```
git init
Initialized empty Git repository in D:/git_practice/.git/
```

检查仓库状态：

```
git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore
        hello_git.py

nothing added to commit but untracked files present (use "git add" to track)
```

将文件加入仓库：

```
git add .

git status
On branch master

No commits yet

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

执行提交：

```
git commit -m "Started project"
[master (root-commit) ba75051] Started project
 2 files changed, 2 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 hello_git.py

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

查看提交历史：

```
git log
commit ba750514f3fb94063fd57ad7c2c7515a9809c112 (HEAD -> master)
Author: Hassium1 <116058435+Hassium1@users.noreply.github.com>
Date:   Mon Sep 25 11:15:21 2023 +0800

    Started project
```

第二次提交：

```
git status
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:   hello_git.py

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

git commit -am "Extending greeting."
[master 8095258] Extending greeting.
 1 file changed, 2 insertions(+), 1 deletion(-)

git status
On branch master
nothing to commit, working tree clean

git log --pretty=oneline
8095258b9ffe988e0313ef8360145c31d059d41c (HEAD -> master) Extending greeting.
ba750514f3fb94063fd57ad7c2c7515a9809c112 Started project
```

放弃修改：

```
git restore .
```

检出以前的提交：

```
git checkout [被哈希加密的id前六位]
Note: switching to 'ba7505'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

  git switch -c <new-branch-name>

Or undo this operation with:

  git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at ba75051 Started project
```

删除仓库：

```
rm -rf .git/
```

第三部分实验过程：

```
提交：
git commit
新建分支：
git branch [branch name]
合并分支：(非线性)
git merge [branch name]
合并分支：(线性)
git rebase [branch name]
HEAD在提交树上向上移动：
git checkout HEAD~[num]
强制移动：
git branch -f [branch neme] HEAD~[num]
撤销变更：(本地)
git reset HEAD~[num]
撤销变更：(远程)
git revert HEAD
```

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？

   答：版本控制是对软件开发过程中各种程序代码、配置文件和说明文档等文件变更进行控制.
   使用git作为版本控制软件可以比较方便地管理和开发项目，git是分布式的控制系统相互对立的修改代码不会影响其他人的开发进程。
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撤销还没有commit的修改应该使用：

   ```
   git restore .
   ```

   使用Git检出（Checkout)已经以前的Commit：

   ```
   git checkout [被哈希加密的id前六位或者七位]
   ```

3. Git中的HEAD是什么？如何让HEAD处于detached HEAD状态？（实际操作）
   答：HEAD表示当前提交的项目状态。
   先查看提交记录:

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

   然后检出以前的commit，HEAD就会离开分支，并进入detached HEDA状态：

   ```
   git checkout + 被哈希加密的id前六位或者七位
   ```

4. 什么是分支（Branch）？如何创建分支？如何切换分支？（实际操作）
   答：分支（Branch)是项目的一个版本。
   创建分支：

   ```
   git branch [branch name]
   ```

   切换分支：

   ```
   git checkout [branch name]
   ```

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

```
git merge [brand name]
```

git merge和git rebase的区别：
git merge合并后的分支历史将保留原始分支的提交历史，同时添加一个新的合并提交。最终的分支历史看起来像是一个分支在另一个分支之上合并的结果。
git rebase合并后的分支历史将被重写为一条线性的提交历史，看起来像是所有更改都是按顺序进行的。原始分支的提交将被重新应用到目标分支上，形成一个新的提交历史。

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）
   答：使用标题：在文字前加上#号
   使用数字列表：在每个列表项前添加数字并紧跟一个英文句点。数字不必按数学顺序排列，但是列表应当以数字 1 起始。
   使用无序列表：在每个列表项前面添加破折号 (-)、星号 (*) 或加号 (+) 。缩进一个或多个列表项可创建嵌套列表。
   使用超链接：

```
[超链接显示名](超链接地址 "超链接title")
```
后续的title可加可不加，比较关键的还是超链接的地址。

实验总结

在本次实验中我学会了如何使用git来对自己的项目进行版本控制，
以及不同的合并分支的方法，也学会了一些编写markdown文档的语法。总的来说难度不大，比较轻松。