实验报告 1
Git 与 LaTeX

洪子翔

2024 年 9 月 3 日

目录

1 实验内容 **3**

1.1 Git 3

1.1.1 初始配置 3

1.1.2 开始 3

1.1.3 提交到 github 4

1.1.4 跟踪状态 5

1.1.5 提交更改 6

1.1.6 跳过暂存提交 7

1.1.7 删除文件 7

1.1.8 移动或重命名文件 8

1.1.9 查看日志 9

1.1.10 查找特定提交信息 10

1.2 LaTeX 11

1.2.1 基本文档结构 11

1.2.2 标题与目录 11

1.2.3 文章层次 12

1.2.4 彩色字体 13

1.2.5 特殊字符 13

1.2.6 列表 13

1.2.7 表格 14

1.2.8 图表 14

1.2.9 插入代码 15

1.2.10 公式 16

2 心得 **16**

1 实验内容

项目链接：https://github.com/SoraRosa0514/system-dev-tools/tree/main/1

1.1 Git

这一部分内容使用样例项目”alice” 完成。
链接：https://github.com/SoraRosa0514/alice

1.1.1 初始配置

可以使用`git config`实现 Git 的各种配置。配置变量按优先级从低到高分为三级：

1. Git 路径下的/etc/gitconfig：对该系统全体用户生效。
2. 用户路径下的 /.gitconfig：对该用户生效。
3. 仓库中的.git/config：对该仓库生效。

排在后面的 config 会覆盖前者。
初次使用 Git 需要初始化用户信息，包括用户名和邮箱地址，代码如下：

```
git config user.name "Alice"
git config user.email AliceRiddel@example.com
```

如果希望一劳永逸，可以在`config`后加上`--global`，此处的配置将作用于整个系统。要查看配置结果，只需要把配置的命令去掉末尾的值即可，例如：

```
$ git config user.name
Alice
```

要查看所有配置内容，输入`$ git config --list`

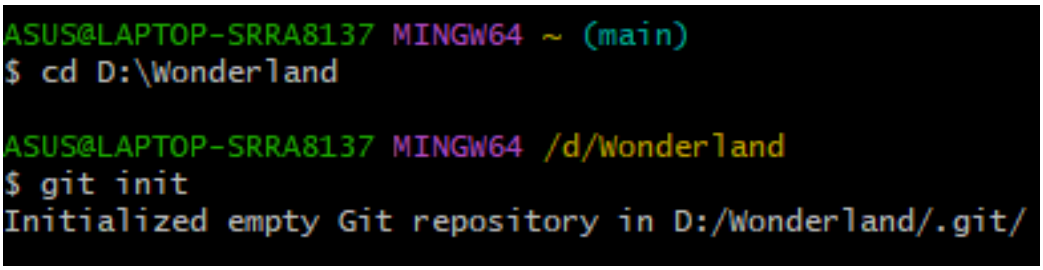
1.1.2 开始

可以新建仓库或克隆已有仓库。

1. 新建仓库

   首先移动到对应目录，再使用`git init`

   成功后，应该可以在对应目录找到.git 文件夹。



2. 克隆仓库

   使用`git clone <repo> <directory>`

   其中 repo 是项目地址，可以使用 ssh,git,https 等多种协议；directory 是要保存到的本地地址。

   以 missing semester 项目为例：

   https:

   ```
   git clone https://github.com/missing-semester-cn/missing-semester-cn.github
   ```

   git:

   ```
   git clone git://...
   ```

   ssh:

   ```
   git clone git@github.com:...
   ```

1.1.3 提交到 github

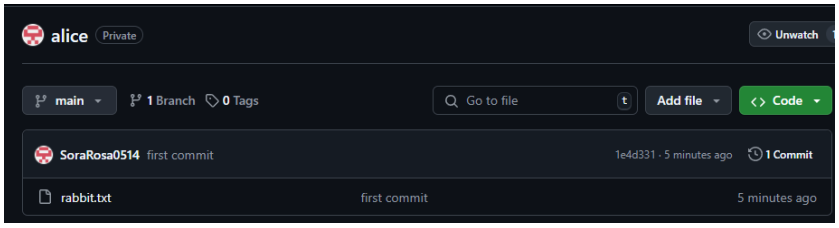
首先在 github 初始化新仓库，之后直接按照提示输入命令即可。

```
ASUS@LAPTOP-SRRA8137 MINGW64 /d/Wonderland (master)
$ git remote add origin https://github.com/SoraRosa0514/alice.git

ASUS@LAPTOP-SRRA8137 MINGW64 /d/Wonderland (master)
$ git branch -M main

ASUS@LAPTOP-SRRA8137 MINGW64 /d/Wonderland (main)
$ git push -u origin main
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 231 bytes | 231.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/SoraRosa0514/alice.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
```

结果如图所示：



1.1.4 跟踪状态

可以用`git status`查看仓库状态。如果在 Wonderland 目录下创建文件”Potion.txt”，再运行这个命令，会看到如下内容：

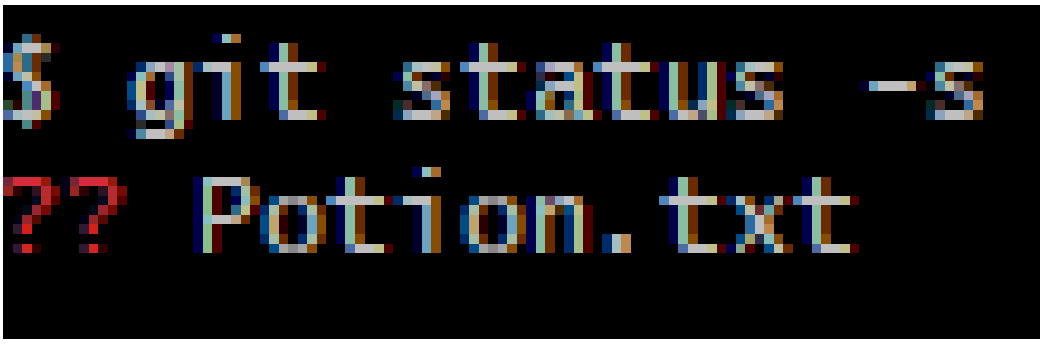
```
ASUS@LAPTOP-SRRA8137 MINGW64 /d/Wonderland (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

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

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

从这里我们能看出：首先我们在 main 分支，并且没有偏差。此外，git 发现了目录下未被跟踪的”Potion.txt”。如果要提交文件，必须跟踪它。
暂时不要管它。如果觉得提示太复杂，用`git status -s`即可打开简略模式，

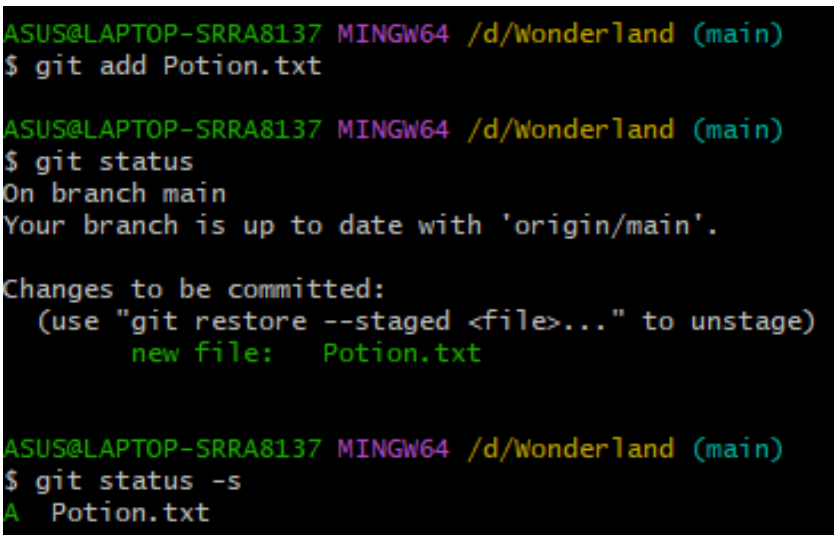
效果如下



前面的?? 代表未跟踪。已跟踪的文件的提示之后会讲解。

1.1.5 提交更改

输入`git add <filename>`就可以跟踪未跟踪的文件。



如图所示，这个命令成功让 git 追踪文件，并将修改放入暂存区，简略模式下由绿 A 标记表示新增的文件。

需要注意：如果之后再对文件进行修改，需要再次暂存更改。例如在 Potion.txt 写下“喝下就会变大的药”，此时再检查状态：

```
ASUS@LAPTOP-SRRA8137 MINGW64 /d/Wonderland (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   Potion.txt

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:   Potion.txt


ASUS@LAPTOP-SRRA8137 MINGW64 /d/Wonderland (main)
$ git status -s
AM Potion.txt
```

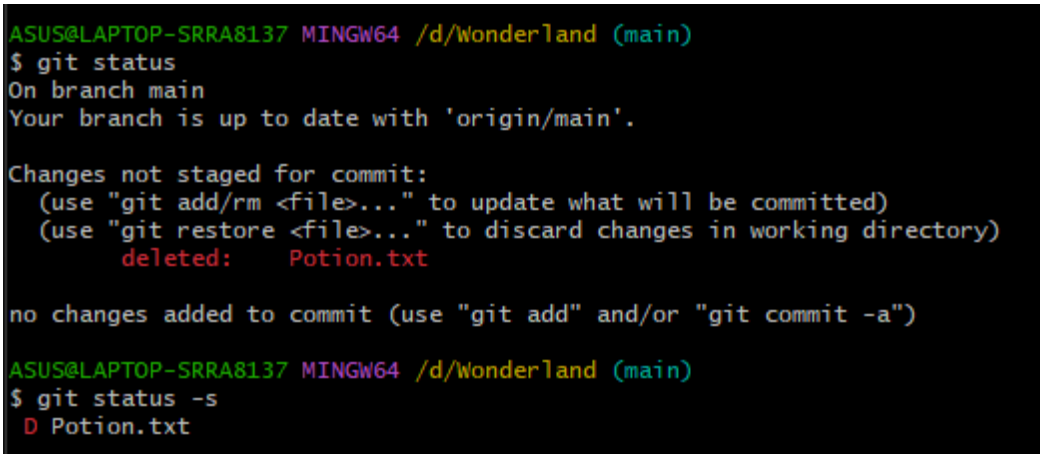
提示更改未暂存，用第二列的红 M 表示。再次用`git add <filename>`即可。之后输入`git commit -m "Test commit"`提交修改，并留下“Test commit”的留言方便阅读。
为了在 github 上看到修改，还需要使用`git push origin main`推送上去。

1.1.6 跳过暂存提交

如果修改了许多文件，且它们都被追踪，直接输入
`git commit -a`
可以跳过暂存步骤。

1.1.7 删除文件

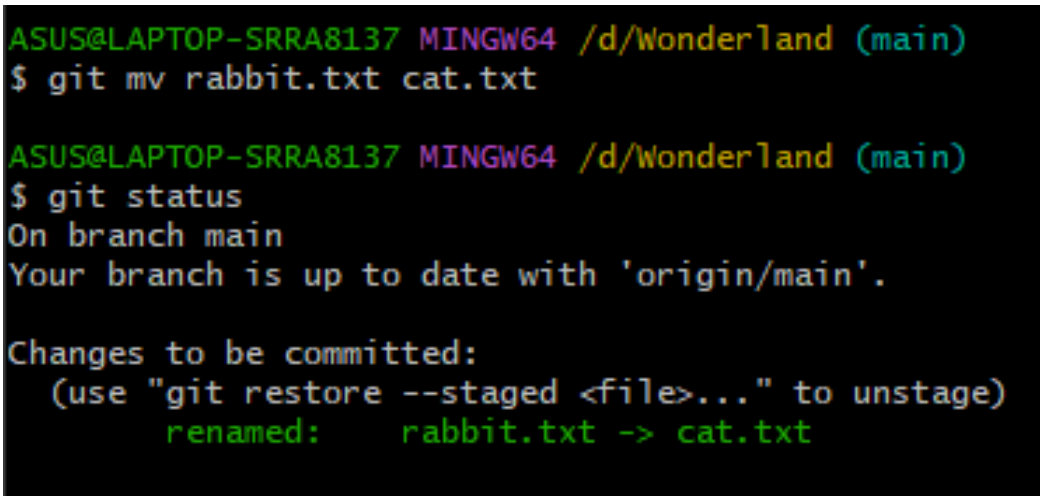
直接在工作目录删除文件是不可行的。假如本地删除 Potion.txt 再跟踪状态：



与增加文件一样,必须将修改放到暂存区。这里需要使用`git rm <filename>`才行，之后提交修改。

1.1.8 移动或重命名文件

只需使用`git mv <file> <newfile>`
例如将目录下的 rabbit.txt 变成 cat.txt：



1.1.9 查看日志

输入git log即可查看历史更改。

通过日志我们能看到每次提交的作者，时间和说明等，时间更近的提交在前。

要想让日志更好读，尝试 `git log --all --graph --decorate`

```
ASUS@LAPTOP-SRRA8137 MINGW64 /d/Wonderland (main)
$  git log --all --graph --decorate
* commit a99e5b8aff85a3d5eb2af850cc1f0b14336004e7 (HEAD -> main, origin/main)
| Author: SoraRosa0514 <140384996+SoraRosa0514@users.noreply.github.com>
| Date:   Tue Sep 3 16:31:20 2024 +0800
|
|     test rename
|
* commit 81fedcd31fd0792c99a4b912b06dd43cf22e3ee4
| Author: SoraRosa0514 <140384996+SoraRosa0514@users.noreply.github.com>
| Date:   Tue Sep 3 16:18:28 2024 +0800
|
|     test delete
|
* commit 75d0b9d8711159ba1171981880e998a4fa5760da
| Author: SoraRosa0514 <140384996+SoraRosa0514@users.noreply.github.com>
| Date:   Tue Sep 3 15:59:06 2024 +0800
|
|     Test commit
|
* commit 1e4d331889ef7076845743ae9a393e0b7d04fb92
  Author: SoraRosa0514 <140384996+SoraRosa0514@users.noreply.github.com>
  Date:   Fri Aug 30 18:07:23 2024 +0800

      first commit
```

这种模式会用彩色的线标注不同提交的分支关系。不过这个仓库没有。

1.1.10 查找特定提交信息

通过在`git log`加上参数，可以方便地查找一些东西。
比如说，我们尝试查看 cat.txt 的最后一次修改：

```
ASUS@LAPTOP-SRRA8137 MINGW64 /d/Wonderland (main)
$ git log -1 cat.txt
commit a99e5b8aff85a3d5eb2af850cc1f0b14336004e7 (HEAD -> main, origin/main)
Author: SoraRosa0514 <140384996+SoraRosa0514@users.noreply.github.com>
Date:   Tue Sep 3 16:31:20 2024 +0800

    test rename
```

此处使用`git log -1 cat.txt`，其中”-x” 代表 “倒数第 x 次更改”。

1.2 LaTeX

1.2.1 基本文档结构

一个最简单的文档可以这样实现：

```
1  \documentclass{ article }
2  \begin{document}
3      Test  test .
4  \end{document}
```

实现的效果如图1.2.1。

Test test.

其中，第一行确定了文件的格式，除了 article 之外，还包括 report，book 等，也可以制定纸张类型、字体大小等参数。第二行和第四行分别是文件环境的开头和结尾，二者之间的内容是文档的主体。begin 之前的内容是预处理命令，可以引入宏包，设置参数等；end 之后的部分则会被忽略。

1.2.2 标题与目录

首先是标题，包括大标题，作者名和日期。

```
1  ...
2  \ title {Doc 1}
3  \author{ Alice }
4  \date{\today}
5  ...
6  \begin{document}
7  \ maketitle
8  \end{document}
```

实现的效果如图1.2.2。

Doc 1

Alice

August 29, 2024

插入标题时，title date 和 author 指令设置标题参数，大括号内是对应的内容。设置完之后，使用 maketitile 指令插入标题。
添加目录可以使用`\tableofcontents`，可能需要多次编译。

1.2.3 文章层次

对 article 格式而言，有 section,subsection 和 subsubsection 三种层次，代码为

```
\section{}
\subsection{}
\subsubsection{}
```

效果如图：

1 一级标题

1.1 二级标题

1.1.1 三级标题

1.2.4 彩色字体

彩色字体的适配需要 color 宏包。
让字体变色的代码是`{\color{colorname}text}`
改变背景色的则是`\colorbox{colorname}{text}`
以蓝底白色的字 water 为例，代码为：

```
\colorbox{blue}{\color{white}{water}}
```

water

1.2.5 特殊字符

$ % ^ & _ { } ~ \
上述文字是特殊字符，需要转义或特殊指令才能显示：

```
\# \$ \% \^{} \& \_ \{ \} \~{} \textbackslash
```

1.2.6 列表

分有序和无序两种。有序列表的代码为：

```
\begin{enumerate}
    \item A
    \item B
\end{enumerate}
```

实现效果：

1. A
2. B

无序列表：

```
\begin{itemize}
    \item A
    \item B
\end{itemize}
```

实现：

- A
- B

1.2.7 表格

以一段代码举例：

```
\begin{tabular}{|l|l|}
Apples & Green \\
\hline
Strawberries & Red \\
\cline{2-2}
Orange & Orange \\
\end{tabular}
```

Apples	Green
Strawberries	Red
Orange	Orange

第一行第二个大括号的内容是表格结构，由 l,r,c 和 | 组成。前三项每个字母对应一列，分别是左对齐、中对齐和右对齐。| 是竖边框。中间的内容，用 & 分隔每一列，双反斜线换行，`\hline`代表完整横边框，`\cline{x-y}`代表从 x 列到 y 列的横边框。

1.2.8 图表

要插入图表（图片）需要引入 graphicx 宏包，以随便的一张图为例：

图 1: 我是样例

```
\begin{figure}[h]
    \centering
    \includegraphics[width=0.5\linewidth]{4.png}
    \caption{我是样例}
    \label{img4}
\end{figure}
```

图表同样是一种环境，中间各行分别代表居中、[行宽 0.5 倍] 文件名 4.png，标题和标签。

begin 后的方括号内标记了图表的放置位置，可以是 h（适应）,t（页首）,b（页尾）等，加! 代表强制。

更精确的控制需要 float 宏包的支持。

1.2.9 插入代码

行内代码可以用`\verb||` 实现，例如：

`print("Hello tex!)`

长段代码需要引入 listings 宏包，基本格式是：

```
\begin{lstlisting}[language=C++]
    cout << "Hello World!";
\textbackslash end{lstlisting}
```

begin 后的方括号内表明代码语言，中间输入即可。

1.2.10 公式

行间公式可以使用`\[...\]`，比如

$$1+2=3$$

`\[1+2=3\]` 长公式可以用 equation 环境。同样以 1+2=3 为例。

$$1+2=3 \tag{1}$$

要输入上下标，上标用 ^{...}，下标用 _{...}
比如

$$a_{down}^{up}$$

要输入分数，用\frac{分子}{分母}
比如

$$\frac{2}{5}$$

希腊字母的输入使用\接希腊字母英文名称的格式，大小写由名称首字母大小写决定。
更复杂的输入最好使用 amsmath 宏包。

2 心得

这节课的学习让我学到了两种非常重要的工具：便利的远程版本控制工具 Git 及平台 GitHub，以及适合学术写作的排版系统 LaTeX。虽然相比之下这两种工具较难使用，但更加专业且便捷。可以预料到，以后我将经常和它们打交道。