

操作系统 Lab0实验报告

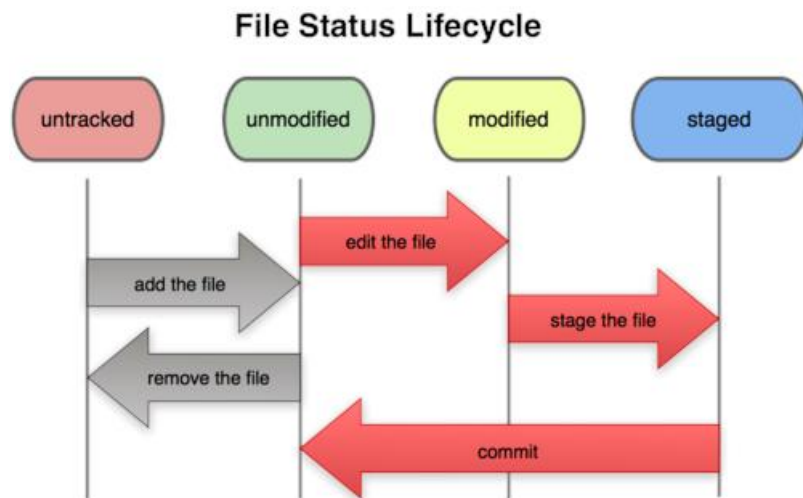
实验思考题

Thinking 0.1

在修改 `README.txt` 文件后，其状态和第一次 `add` 之前不同。

在第一次 `add` 之前，`git` 还没有追踪 `README.txt` 文件，因此此时显示 `Untracked files`。而将 `README.txt` 提交后它便存在于 `git` 的版本管理系统之中，再对其进行修改则会显示 `Modified` 以提示当前文件的内容与 `git` 上提交的版本有出入。

Thinking 0.2



- `add the file`: `git add` (跟踪文件)
- `stage the file`: `git add` (将修改后的文件加入暂存区)
- `commit`: `git commit`

Thinking 0.3

- `git checkout -- printf.c`

文件被从工作区删除后可以从暂存区和版本库中找回。

- `git reset HEAD printf.c`

`git checkout -- printf.c`

先将最近一次 `commit` 版本中的文件拉回暂存区，再从暂存区中恢复该文件。

- `git rm Tucao.txt`

将该文件从暂存区删除。

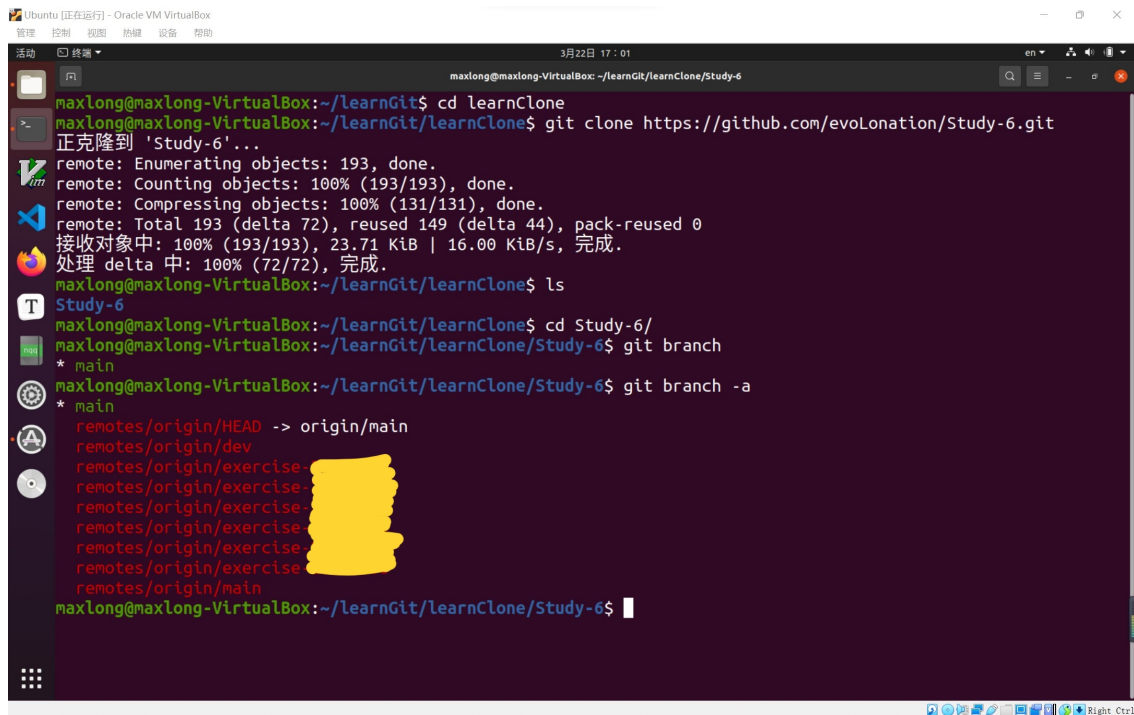
Thinking 0.4

- 使用 `git reset --hard HEAD^` 再使用 `git log` 后，提交说明为 3 的那次提交消失了。
- 使用 `git reset --hard <Hash-code>`，再使用 `git log` 后，提交说明为 2 的那次提交消失了。
- 对于这些操作，个人理解是 `git` 会保存每次提交后工程的状态，并生成一个 `Hash-code` 指向状态存储的地址，使用命令 `git reset --hard <Hash-code>` 时，`git` 便会使 `HEAD` 指针跳转到该

Hash-code 所指向的状态，将其视为当前状态。因此使用这个方式便可以向前或向后恢复工程的版本。

Thinking 0.5

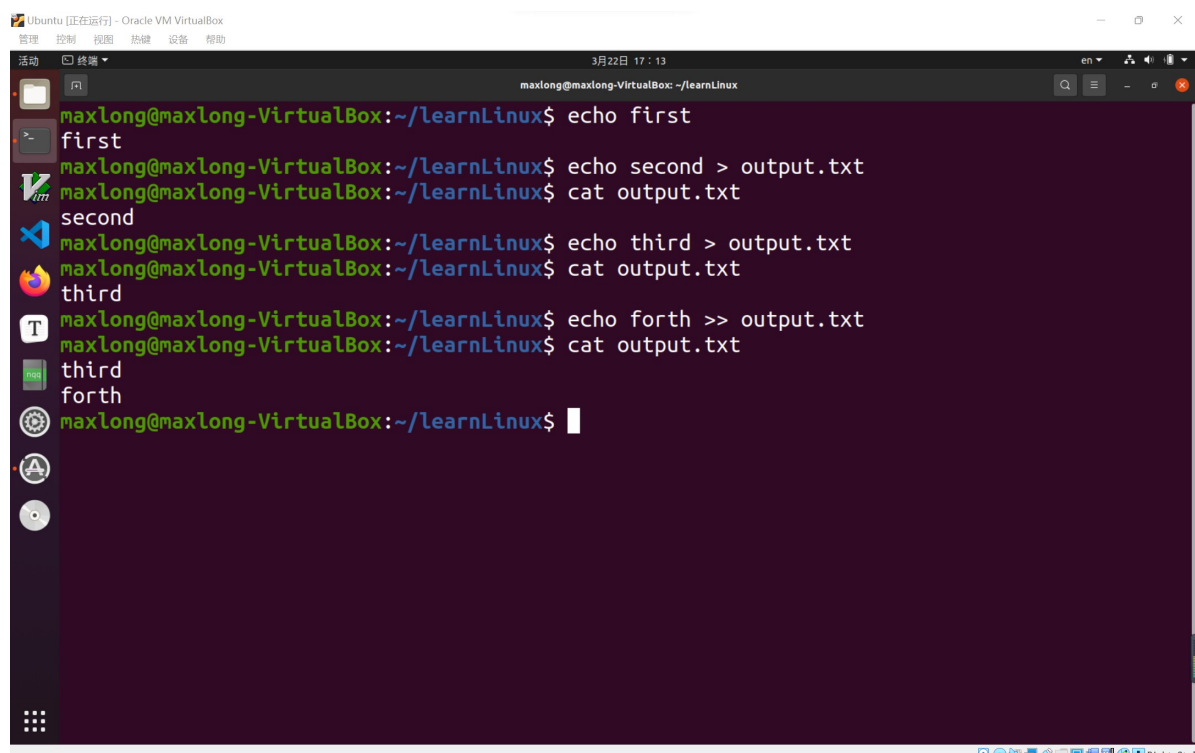
- 克隆时所有分支均被克隆，但只有HEAD指向的分支被检出。**克隆时只克隆远程库的 HEAD 指向的分支，需要使用 `git branch -a` 可以检出本地和远程的所有分支。**



```
maxlong@maxlong-VirtualBox: ~/learnGit/learnClone/Study-6
maxlong@maxlong-VirtualBox:~/learnGit$ cd learnClone
maxlong@maxlong-VirtualBox:~/learnGit/learnClone$ git clone https://github.com/evoLonation/Study-6.git
正克隆到 'Study-6'...
remote: Enumerating objects: 193, done.
remote: Counting objects: 100% (193/193), done.
remote: Compressing objects: 100% (131/131), done.
remote: Total 193 (delta 72), reused 149 (delta 44), pack-reused 0
接收对象中: 100% (193/193), 23.71 KiB | 16.00 KiB/s, 完成.
处理 delta 中: 100% (72/72), 完成.
maxlong@maxlong-VirtualBox:~/learnGit/learnClone$ ls
Study-6
maxlong@maxlong-VirtualBox:~/learnGit/learnClone$ cd Study-6/
maxlong@maxlong-VirtualBox:~/learnGit/learnClone/Study-6$ git branch
* main
maxlong@maxlong-VirtualBox:~/learnGit/learnClone/Study-6$ git branch -a
* main
remotes/origin/HEAD -> origin/main
remotes/origin/dev
remotes/origin/exercise
remotes/origin/exercise
remotes/origin/exercise
remotes/origin/exercise
remotes/origin/exercise
remotes/origin/exercise
remotes/origin/main
maxlong@maxlong-VirtualBox:~/learnGit/learnClone/Study-6$
```

- 克隆出的工作区中执行 `git log`、`git status`、`git checkout`、`git commit`等操作不会去访问远程版本库。**正确**
- 克隆时只有远程版本库 HEAD 指向的分支被克隆。**正确**
- 克隆后工作区的默认分支处于 master 分支。**正确**

Thinking 0.6



```
maxlong@maxlong-VirtualBox: ~/learnLinux
maxlong@maxlong-VirtualBox:~/learnLinux$ echo first
first
maxlong@maxlong-VirtualBox:~/learnLinux$ echo second > output.txt
maxlong@maxlong-VirtualBox:~/learnLinux$ cat output.txt
second
maxlong@maxlong-VirtualBox:~/learnLinux$ echo third > output.txt
maxlong@maxlong-VirtualBox:~/learnLinux$ cat output.txt
third
maxlong@maxlong-VirtualBox:~/learnLinux$ echo forth >> output.txt
maxlong@maxlong-VirtualBox:~/learnLinux$ cat output.txt
third
forth
maxlong@maxlong-VirtualBox:~/learnLinux$
```

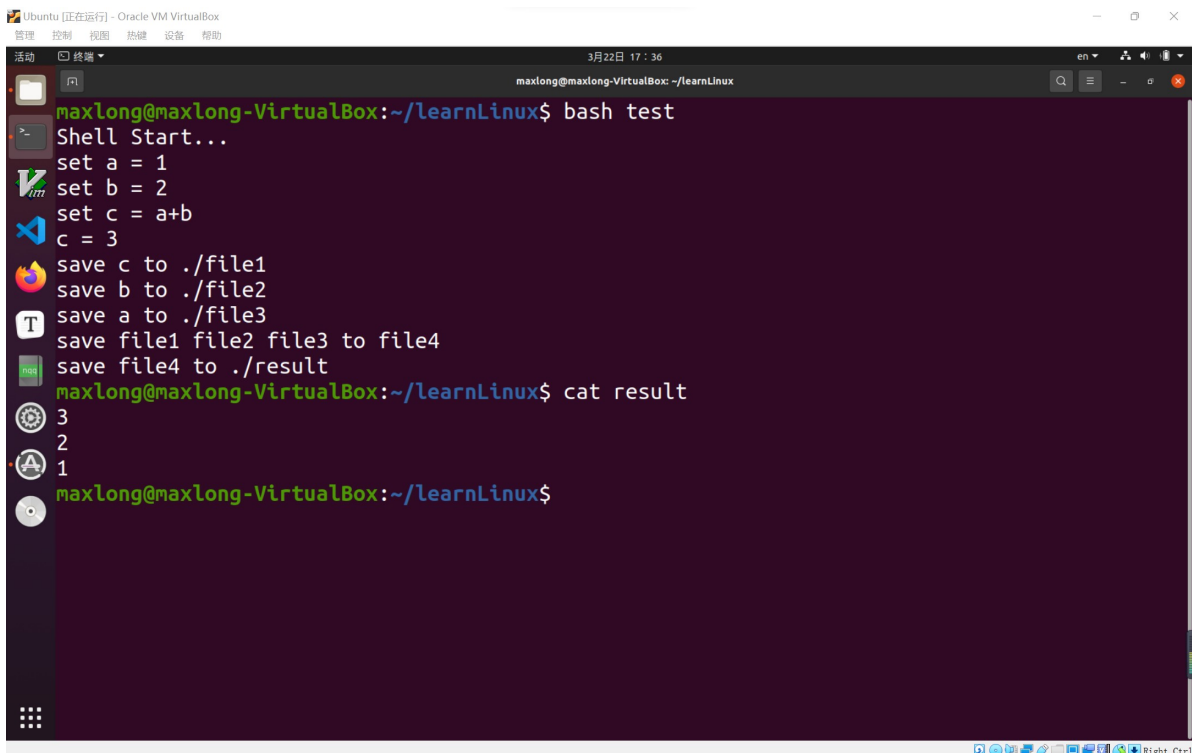
Thinking 0.7

command 文件:

```
1 touch test
2 echo echo Shell Start... > test
3 echo echo set a = 1 >> test
4 echo a=1 >> test
5 echo echo set b = 2 >>test
6 echo b=2 >> test
7 echo echo set c = a+b >> test
8 echo c=\$[\$a+\$b] >> test
9 echo echo c = \$c >> test
10 echo echo save c to ./file1 >> test
11 echo echo \$c\>file1 >> test
12 echo echo save b to ./file2 >> test
13 echo echo \$b\>file2 >> test
14 echo echo save a to ./file3 >> test
15 echo echo \$a\>file3 >> test
16 echo echo save file1 file2 file3 to file4 >> test
17 echo cat file1\>file4 >> test
18 echo cat file2\>\>file4 >> test
19 echo cat file3\>\>file4 >> test
20 echo echo save file4 to ./result >> test
21 echo cat file4\>\>result >> test
```

result 文件:

```
1 3
2 2
3 1
```



The screenshot shows a terminal window titled 'maxlong@maxlong-VirtualBox: ~/learnLinux'. The user has executed the command 'bash test'. The output of the script is as follows:

```
Shell Start...
set a = 1
set b = 2
set c = a+b
c = 3
save c to ./file1
save b to ./file2
save a to ./file3
save file1 file2 file3 to file4
save file4 to ./result
```

After running 'cat result', the terminal displays the contents of the result file:

```
3
2
1
```

test 文件:

```

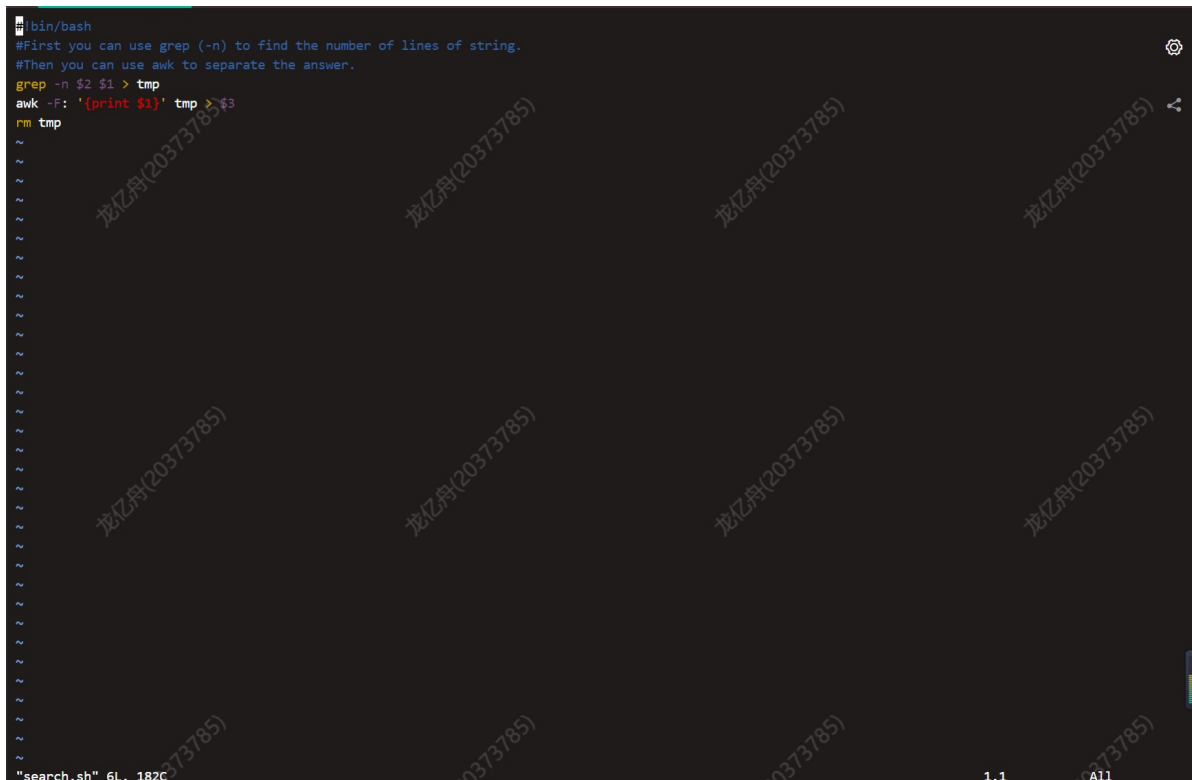
1 echo Shell Start... #屏幕打印
2 echo set a = 1 #屏幕打印
3 a=1 #设置变量a的值为1
4 echo set b = 2 #屏幕打印
5 b=2 #设置变量b的值为2
6 echo set c = a+b #屏幕打印
7 c=$((a+b)) #设置变量c的值为a+b的结果
8 echo c = $c #屏幕打印输出变量c的值
9 echo save c to ./file1 #屏幕打印
10 echo $c>file1 #将c的值输出到file1中
11 echo save b to ./file2 #屏幕打印
12 echo $b>file2 #将b的值输出到file2中
13 echo save a to ./file3 #屏幕打印
14 echo $a>file3 #将a的值输出到file3中
15 echo save file1 file2 file3 to file4 #屏幕打印
16 cat file1>file4 #将file1的内容写入file4
17 cat file2>>file4 #将file2的内容添加到file4
18 cat file3>>file4 #将file3的内容添加到file4
19 echo save file4 to ./result #屏幕打印
20 cat file4>>result #将file4的内容添加到result

```

- `echo echo Shell Start` 与 `echo 'echo Shell Start'` 的效果没有区别
- `echo echo $c>file1` 是把 `echo $c` 的内容写入 `file1` ; `echo 'echo $c>file1'` 是把 `echo $c>file1` 输出到屏幕

实验难点

脚本文件中对于变量的应用



```

bin/bash
#First you can use grep (-n) to find the number of lines of string.
#Then you can use awk to separate the answer.
grep -n $2 $1 > tmp
awk -F: '{print $1}' tmp > $3
rm tmp

```

search.sh" 6L, 182C 1,1 All

我在编写 `Exercise0.3` 中的 `search.sh` 文件时耗费的时间比较多。

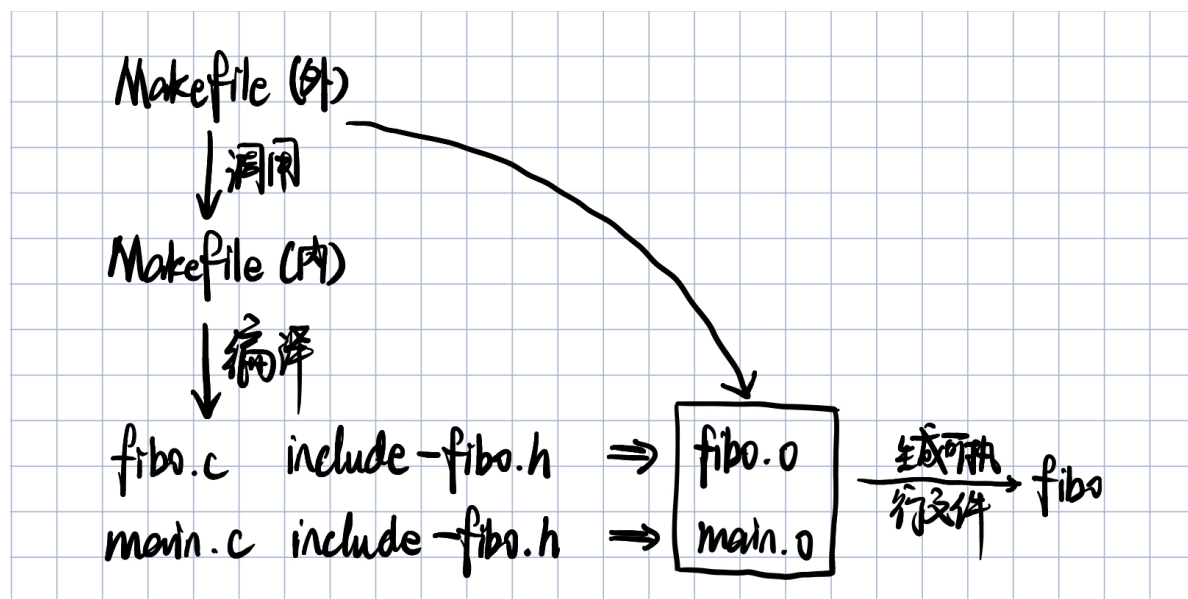
主要原因是在编写 `awk -F: '{print $1}' tmp > $3` 指令时，对于引号内外变量的运用有混淆，由此查阅了 `awk` 命令的相关文档，才处理好了对命令的编写。

除此之外，在刚开始编写时并没有想到利用 `tmp` 文件作数据的中转，而是使用了一个内部变量，导致程序运行比较混乱，无法达到要求的运行效果，经过多次尝试，发现使用一个文件进行数据中转，再在脚本结束时删除该文件是一个可行的方法，由此才完整地解决了本题。

Makefile的编写

在 `Exercise0.4` 中，需要编写内外两个 `Makefile` 使外层 `Makefile` 调用内层 `Makefile` 达到编译完整程序的效果。而由于指导书中介绍 `Makefile` 的部分比较简洁，因此需要自己查阅较多资料来理解 `Makefile` 的编写规则 and 不同参数的用法。

在了解了编写 `Makefile` 的基本方法，并翻阅了讨论区其他同学的间接之后，我才成功梳理出了本题的编译逻辑，并成功完成了 `Makefile` 的编写。



体会与感想

由于本次实验是在冬奥上岗期间完成的，因此并没有能够抽出完整的时间来进行整体学习，而是断断续续经过了两三天才完成。

在完成实验的过程中，经常会有忘记前一天学习过的内容而需要重新翻看指导书和笔记的情况，导致整体的效率并不高，而且在学习过程中，由于战线拉得比较长，导致学习时前后知识的关联性和整体性不是很高，有可能会影响到对我自己对内容的掌握。在未来的实验中，我争取能够抽出完整的时间来进行学习和实践，提升学习过程的整体性和延续性。

总体来说，本次实验的难度并不高，但对于初学者来说仍有一定难度，如 `git`，`Makefile`，`shell` 等内容都是初次接触，因此需要一定时间进行熟悉和练习，希望经过接下来的学习能够更加熟练地应用这些工具，辅助我进行实验和开发，提升学习和实践的效率。