

操作系统 LAB0 实验报告

报告人：21373037 杨博文

一. 思考题

1. Untracked, Staged, Modified 如图所示

```
git@21373037:~/21373037 (lab0)$ cat Untracked.txt
位于分支 lab0
您的分支与上游分支 'origin/lab0' 一致。

未跟踪的文件:
  (使用 "git add <文件>..." 以包含要提交的内容)
    README.txt
    Untracked.txt

提交为空, 但是存在尚未跟踪的文件 (使用 "git add" 建立跟踪)
```

```
git@21373037:~/21373037 (lab0)$ cat Stage.txt
位于分支 lab0
您的分支与上游分支 'origin/lab0' 一致。

要提交的变更:
  (使用 "git restore --staged <文件>..." 以取消暂存)
    新文件:    README.txt

未跟踪的文件:
  (使用 "git add <文件>..." 以包含要提交的内容)
    Stage.txt
    Untracked.txt
```

```
git@21373037:~/21373037 (lab0)$ cat Modified.txt
位于分支 lab0
您的分支领先 'origin/lab0' 共 1 个提交。
  (使用 "git push" 来发布您的本地提交)

尚未暂存以备提交的变更:
  (使用 "git add <文件>..." 更新要提交的内容)
  (使用 "git restore <文件>..." 丢弃工作区的改动)
    修改:    README.txt

未跟踪的文件:
  (使用 "git add <文件>..." 以包含要提交的内容)
    Modified.txt
    Stage.txt
    Untracked.txt

修改尚未加入提交 (使用 "git add" 和/或 "git commit -a")
```

README.txt 文件在未 add 前，处于未跟踪的状态，在 add 后，处于暂存区中，即处在 staged 暂存状态，而在对该文件修改后且未再次 add 时，该文件处于已修改的状态。一旦初次跟踪该文件（add）后，除非删除该文件，否则将不会再处于未跟踪的状态，若对其进行修改，即使仍需再次 add,那也是从已修改状态转移到暂存状态。

2. add the file: git add

stage the file: git add

commit the file: git commit -m

3. (1)git checkout --print.c

(2)git checkout HEAD print.c

(3)git rm --cached hello.txt

4.提交三次后版本信息如下图所示：



```
commit 9cb93acfc690fbfeb8851150b467b3ef930e82ee (HEAD -> lab0)
Author: 杨博文 <21373037@buaa.edu.cn>
Date:   Fri Mar 3 19:15:36 2023 +0800

    3

commit eae22040a9fed8cf590557efb181b72e85a1c911
Author: 杨博文 <21373037@buaa.edu.cn>
Date:   Fri Mar 3 19:15:07 2023 +0800

    2

commit 4ef025e3ac2b453bc3c63666011d84069cad7404
Author: 杨博文 <21373037@buaa.edu.cn>
Date:   Fri Mar 3 19:14:17 2023 +0800

    1
```

进行第一次 HEAD 回退后如下图所示, 可见往后回退一个版本到 2。

```
commit eae22040a9fed8cf590557efb181b72e85a1c911 (HEAD -> lab0)
Author: 杨博文 <21373037@buaa.edu.cn>
Date:   Fri Mar 3 19:15:07 2023 +0800

    2

commit 4ef025e3ac2b453bc3c63666011d84069cad7404
Author: 杨博文 <21373037@buaa.edu.cn>
Date:   Fri Mar 3 19:14:17 2023 +0800

    1
```

进行一次说明为 1 的哈希值回退后如下图所示, 可见往后回退到版本 1 处。

```
commit 4ef025e3ac2b453bc3c63666011d84069cad7404 (HEAD -> lab0)
Author: 杨博文 <21373037@buaa.edu.cn>
Date:   Fri Mar 3 19:14:17 2023 +0800

    1
```

进行一次说明为 3 的哈希值回退后如下图所示, 可见恢复到了版本 3 处

```
git@21373037:~/21373037 (lab0)$ git reset --hard 9cb93acfc690fbfeb8851150b467b3ef930e82ee
HEAD 现在位于 9cb93ac 3
git@21373037:~/21373037 (lab0)$ git log
commit 9cb93acfc690fbfeb8851150b467b3ef930e82ee (HEAD -> lab0)
Author: 杨博文 <21373037@buaa.edu.cn>
Date:   Fri Mar 3 19:15:36 2023 +0800

    3
```

5. 执行第一条语句后终端回显 first

```
git@21373037:~/21373037 (lab0)$ echo first
first
```

执行第二条语句后打开 output.txt, 其中内容为 second

```
1 second
~
~
~
```

执行第三条语句后 output.txt 如下图，可见原内容被覆盖掉了。

```
1 third
~
```

执行第四条语句后 output.txt 如下图，可见新输出的文本追加在了原内容的后面。

```
1 third
2 forth
~
```

6. echo echo Start Shell 输出 echo Start Shell

echo 'echo Start Shell 输出 Start Shell

echo echo \$c>file 在 file 中输出 echo

Echo 'echo \$c>file`在 file 中无输出

为得到目标文本，command 如图

```

1 #!/bin/bash
2
3 echo "echo Shell Start" > test
4 echo "echo set a = 1" >> test
5 echo "a=1" >> test
6 echo "echo set b = 2" >> test
7 echo "b=2" >> test
8 echo "echo set c = a+b" >> test
9 echo "c=\${a+$b}" >> test
10 echo "echo c = $c" >> test
11 echo "echo save c to ./file1" >> test
12 echo "echo $c>file1" >> test
13 echo "echo save b to ./file2" >> test
14 echo "echo $b>file2" >> test
15 echo "echo save a to ./file3" >> test
16 echo "echo $a>file3" >> test
17 echo "echo save file1 file2 file3 to file4" >> test
18 echo "cat file1>file4" >> test
19 echo "cat file2>>file4" >> test
20 echo "cat file3>>file4" >> test
21 echo "echo save file4 to ./result" >> test
22 echo "cat file4>>result" >> test

```

Test 文件如题目要求所示:

```

echo Shell Start...
echo set a = 1
a=1
echo set b = 2
b=2
echo set c = a+b
c=${a+$b}
echo c = $c
echo save c to ./file1
echo $c>file1
echo save b to ./file2
echo $b>file2
echo save a to ./file3
echo $a>file3
echo save file1 file2 file3 to file4
cat file1>file4
cat file2>>file4
cat file3>>file4
echo save file4 to ./result
cat file4>>result

```

将 test 赋予权限，作为批处理文件运行后，终端上显示：

```
git@21373037:~/21373037 (lab0)$ ./test
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
```

Result 文件中内容为：

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

为创建 test 而在 command 脚本的输入中，用双引号将想要输出的文字括起来即可，注意面对\$时应使用\转义。而执行 test 文件时，c,b,a 三个参数所对应的值被存入不同的 file 文件中，最后再将三个文件中的内容依次重定向到 result 文件中，最后结果即为上图，输出 3, 2, 1。

二. 难点分析

本次 lab0 考察的是对于 Linux 命令行，Makefile，Bash 脚本的基本操作编写方法，本身不算太难，其中我遇到了三个难点。

第一个难点是 Makefile 的嵌套调用\$(MAKE) -C <子目录>，意思为调用子目录的 makefile，实现类似于函数调用的机制。第二个难点在于两个 object 目标文件的链接，我们知道预处理只是加入了

头文件的引用，然后并没有把实际内容放进来，这一步在链接操作中完成，可使用 `gcc <目标文件> <目标文件>` 的方式来得到最后可执行代码。在预处理时，用到的头文件会进入系统自带的库中寻找，若该头文件是自己编写的该如何定位寻找呢，我们可以使用 `gcc -I <文件路径>` 的方式定位到手写的头文件。最后是由于各种参数的混淆与记忆不清，比如 `sed` 的各种使用方法，经常会使用错误或记不清，需要再次巩固记忆。

三. 实验体会

本次 LAB0 实验我总共用时三小时左右，实验本身并无很硬核的难点，但由于我没有观看教学视频，导致很多东西都是在网上现学现卖，后来才得知视频中涵盖了我大部分的困惑点，以后要注意多看教程，这样才能少走弯路。同时我深感各种指令(尤其是文本指令)的繁琐，想必记忆并熟练运用好需要一点功夫，争取在闭卷的 LAB0 课下考试中取得优异成绩。

我的实验报告到此结束，谢谢。