

Lab0实验报告

Lab0实验报告

实验思考题

Thinking 0.1

Thinking 0.2

Thinking 0.3

Thinking 0.4

Thinking 0.5

Thinking 0.6

Thinking 0.7

实验难点

体会与感想

指导书反馈

实验思考题

Thinking 0.1

status 不一样。

add 之前的 status 表示没有跟踪某个文件的变化，使用 `git add` 之后才对文件进行跟踪。

修改 add 过之后的文件的 status 表示修改了某个文件，但还没有加入到暂存区中。

Thinking 0.2

- add the file 对应 `git add`
- stage the file 对应 `git add`
- commit 对应 `git commit`

Thinking 0.3

- 使用

```
1 | git checkout printf.c
```

- 依次使用

```
1 | git reset HEAD printf.c
2 | git checkout printf.c
```

- 使用

```
1 | git rm --cached Tucao.txt
```

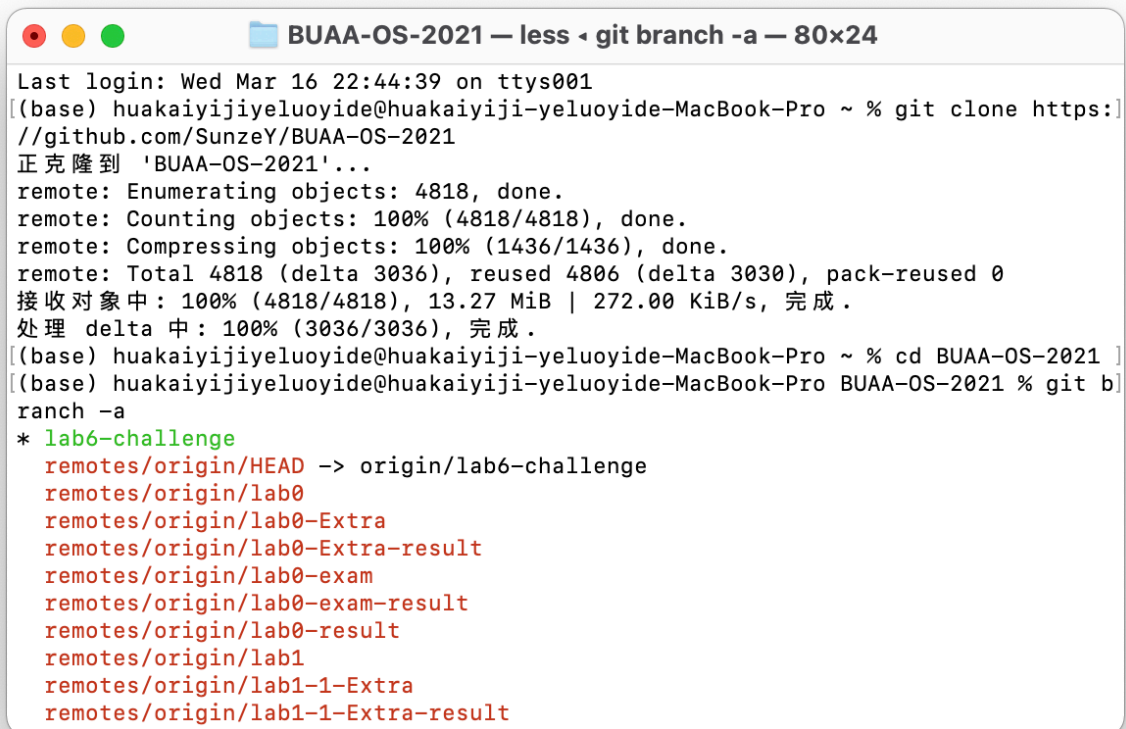
Thinking 0.4

`git reset --hard` 指令可以用来恢复文件的版本，既可以往回恢复，也可以往后“恢复”。

但是，`--hard` 是 `reset` 命令唯一的危险用法，它也是 `git` 会真正地销毁数据的几个操作之一。其他任何形式的 `reset` 调用都可以轻松撤消，但是 `--hard` 选项不能，因为它强制覆盖了工作目录中的文件。若该文件还未提交，`git` 会覆盖它从而导致无法恢复。使用时要小心谨慎。

Thinking 0.5

1. 正确。克隆 `gitHub` 用户 `SunzeY` 的 `BUAA_OS_2021` 仓库，使用 `git branch -a` 命令查看，可以看到所有分支。但是只有 `HEAD` 指向的分支被检出。
2. 正确。在我们做作业时要提交到远程。若不使用 `git push`，则所有访问都是在本地库中进行，不会访问到远程库。
3. 错误。由问题1可知，克隆时所有分支均被克隆。
4. 错误。如图所示。分支名字只是个代号。问题1中，`clone` 下来后，默认分支是 `lab6-challenge` 分支，但是分支中存在 `master` 分支。默认分支可以设置，与设置的 `default` 分支相同。



```

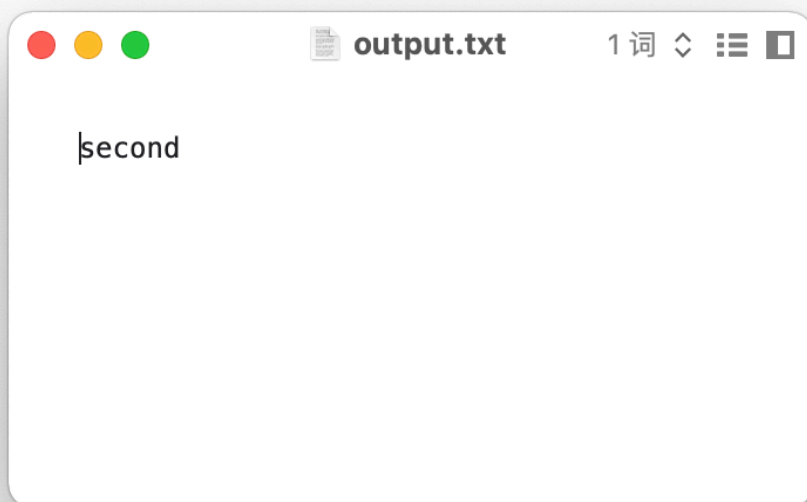
BUAA-OS-2021 — less ◀ git branch -a — 80x24
Last login: Wed Mar 16 22:44:39 on ttys001
[(base) huakaiyijiyeluoyide@huakaiyiji-yeluoyide-MacBook-Pro ~ % git clone https://github.com/SunzeY/BUAA-OS-2021]
正克隆到 'BUAA-OS-2021'...
remote: Enumerating objects: 4818, done.
remote: Counting objects: 100% (4818/4818), done.
remote: Compressing objects: 100% (1436/1436), done.
remote: Total 4818 (delta 3036), reused 4806 (delta 3030), pack-reused 0
接收对象中: 100% (4818/4818), 13.27 MiB | 272.00 KiB/s, 完成.
处理 delta 中: 100% (3036/3036), 完成.
[(base) huakaiyijiyeluoyide@huakaiyiji-yeluoyide-MacBook-Pro ~ % cd BUAA-OS-2021 ]
[(base) huakaiyijiyeluoyide@huakaiyiji-yeluoyide-MacBook-Pro BUAA-OS-2021 % git branch -a]
* lab6-challenge
remotes/origin/HEAD -> origin/lab6-challenge
remotes/origin/lab0
remotes/origin/lab0-Extra
remotes/origin/lab0-Extra-result
remotes/origin/lab0-exam
remotes/origin/lab0-exam-result
remotes/origin/lab0-result
remotes/origin/lab1
remotes/origin/lab1-1-Extra
remotes/origin/lab1-1-Extra-result
  
```

Thinking 0.6

如图所示

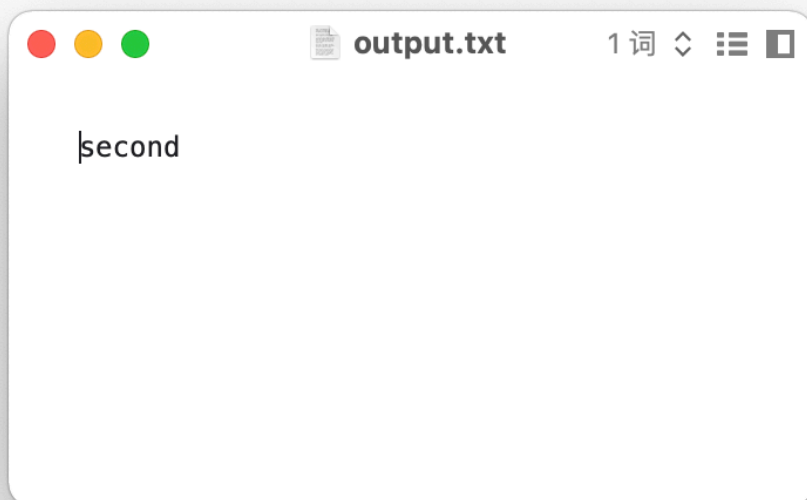
1. 执行 `echo first` 和 `echo second > output.txt`

```
Last login: Wed Mar 16 23:03:15 on ttys001
[(base) huakaiyijiyeluoyide@huakaiyiji-yeluoyide-MacBook-Pro ~ % echo first
first
[(base) huakaiyijiyeluoyide@huakaiyiji-yeluoyide-MacBook-Pro ~ % echo second > ou
tput.txt
(base) huakaiyijiyeluoyide@huakaiyiji-yeluoyide-MacBook-Pro ~ %
```



2. 执行 `echo third > output.txt`

```
Last login: Wed Mar 16 23:03:15 on ttys001
[(base) huakaiyijiyeluoyide@huakaiyiji-yeluoyide-MacBook-Pro ~ % echo first
first
[(base) huakaiyijiyeluoyide@huakaiyiji-yeluoyide-MacBook-Pro ~ % echo second > ou
tput.txt
(base) huakaiyijiyeluoyide@huakaiyiji-yeluoyide-MacBook-Pro ~ %
```



3. 执行 `echo forth >> output.txt`

```
Last login: Wed Mar 16 23:03:15 on ttys001
(base) huakaiyijiyeluoyide@huakaiyiji-yeluoyide-MacBook-Pro ~ % echo first
first
(base) huakaiyijiyeluoyide@huakaiyiji-yeluoyide-MacBook-Pro ~ % echo second > output.txt
(base) huakaiyijiyeluoyide@huakaiyiji-yeluoyide-MacBook-Pro ~ % echo third > output.txt
(base) huakaiyijiyeluoyide@huakaiyiji-yeluoyide-MacBook-Pro ~ % echo forth >> output.txt
(base) huakaiyijiyeluoyide@huakaiyiji-yeluoyide-MacBook-Pro ~ %
```



Thinking 0.7

Command.sh 文件如下:

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

result 文件如下:

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

The screenshot shows a macOS terminal window with two tabs: `command.sh` and `test.sh`. The `command.sh` tab is active, showing a script that sets variables `a` and `b`, calculates `c = a + b`, and saves the results to files `file1`, `file2`, `file3`, and `file4`. The `test.sh` tab shows a similar script. A separate window titled `Lab0 --zsh-- 80x24` shows the execution of `command.sh` and `test.sh`, with the output of `test.sh` displayed in a small window titled `result`.

```
command.sh
1 #!/usr/bin/env bash
2
3 echo 'echo Shell Start...' > test.sh
4 echo 'echo set a = 1' >> test.sh
5 echo 'a=1' >> test.sh
6 echo 'echo set b = 2' >> test.sh
7 echo 'b=2' >> test.sh
8 echo 'echo set c = a+b' >> test.sh
9 echo 'c=$((a+b))' >> test.sh
10 echo 'echo save c to ./file1' >> test.sh
11 echo 'echo $c>file1' >> test.sh
12 echo 'echo save b to ./file2' >> test.sh
13 echo 'echo $b>file2' >> test.sh
14 echo 'echo save a to ./file3' >> test.sh
15 echo 'echo $a>file3' >> test.sh
16 echo 'echo save file1 file2 file3 to file4' >> test.sh
17 echo 'cat file1>file4' >> test.sh
18 echo 'cat file2>>file4' >> test.sh
19 echo 'cat file3>>file4' >> test.sh
20 echo 'echo save file4 to ./result' >> test.sh
21 echo 'cat file4>result' >> test.sh

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

`echo echo Shell Start` 与 `echo 'echo Shell Start'` 效果完全一样:

```

Last login: Wed Mar 16 23:18:29 on ttys001
(base) huakaiyijiyeluoyide@huakaiyiji-yeluoyide-MacBook-Pro ~ % echo echo Shell Start
echo Shell Start
(base) huakaiyijiyeluoyide@huakaiyiji-yeluoyide-MacBook-Pro ~ % echo 'echo Shell Start'
echo Shell Start
(base) huakaiyijiyeluoyide@huakaiyiji-yeluoyide-MacBook-Pro ~ %

```

`echo echo \${c}>file1` 与 `echo 'echo \${c}>file1'` 效果不一样。

前者新建了 file1 文件，内容为 `echo \${c}`，后者直接在终端显示 `echo \${c}>file1`。如图所示。

```

Last login: Wed Mar 16 23:22:47 on ttys001
(base) huakaiyijiyeluoyide@huakaiyiji-yeluoyide-MacBook-Pro ~ % echo echo \${c}>file1
(base) huakaiyijiyeluoyide@huakaiyiji-yeluoyide-MacBook-Pro ~ % echo 'echo \${c}>file1'
echo \${c}>file1
(base) huakaiyijiyeluoyide@huakaiyiji-yeluoyide-MacBook-Pro ~ %

```



实验难点

本次实验需要掌握并熟悉较多的命令操作。主要为熟悉 Linux 的 `bash` 基本命令，还有使用 `vim` 编辑文本，`gcc` 命令行，编写 `Shell` 脚本、`make` 文件。本次实验的主要难度在于编写 `Shell` 脚本和 `make` 文件，主要原因是第一次接触。遇到的困难就是 `make` 文件的复合调用以及链接库文件，也就是这次 `Exercise 0.4` 的部分。

一开始我觉得不需要编写 `code` 文件夹中的 `Makefile` 文件也可以完成，但是考虑到题目的要求，应该是外层 `Makefile` 调用内层 `Makefile` 完成需求。于是通过查阅资料以及看课程指导书下方的讨论区，从而了解到如何通过外层 `Makefile` 调用内层 `Makefile`，以及如何连接到不是本文件夹下的库文件，从而写出下方的两个 `Makefile`。

`/csc` 目录下的 `Makefile`:

```
1 all:
2   cd ./code && make fibo
3
4 clean:
5   rm ./code/*.o
```

`/csc/code` 目录下的 `Makefile`:

```
1 fibo:
2   gcc -c fibo.c
3   gcc -c main.c -I../include
4   gcc fibo.o main.o -o ../fibo
```

`Shell` 脚本以及 `Makefile` 文件中的赋值语句中，参数与等于号之间不应有空格，若有空格系统会判定空格为分隔符，从而语句非法。

体会与感想

本实验并不涉及操作系统课程的主体部分，仅仅是一个对 `shell` 指令的练习。还是由于命令行使用的不熟练，界面不适应等原因，在学习这些指令，尤其是在 `shell` 脚本与 `makefile` 的使用中，以及完成作业上仍然还是花了较多的时间。在后期进行真正的操作系统架构的设计时，仍然需要多次调用这些指令，因此掌握它们尤为重要，在接下来的时间中，我仍应该多对这些指令进行联系，从而让自己接下来的操作系统实验进行得更加顺利。

指导书反馈

条目或许分的太细了，导致经常找不到想要查阅的地方。同时 `Thinking` 部分与指导书内的选择题要是能单独拿出来，食用起来或许更方便？