Lab0实验报告

思考题

Thinking 0.1

status在三个阶段都不一样

- add之前的status表示没有跟踪的文件Untracked files，使用add之后便进行了文件的跟踪
- 修改add之后status展示修改了的文件信息，没有加入暂存区

Thinking 0.2

```
git add # add the file
git add # stage the file
git commit # commit
```

Thinking 0.3

- ```
  git checkout -- print.c
  ```

- ```
  git reset HEAD print.c
  git checkout -- print.c
  # 先回调暂存区中的文件内容
  # 再从暂存区中恢复文件
  ```

- ```
  git rm --cached hello.txt
  ```

Thinking 0.4

- 使用 `git reset --hard HEAD^` 之后再使用 `git log`

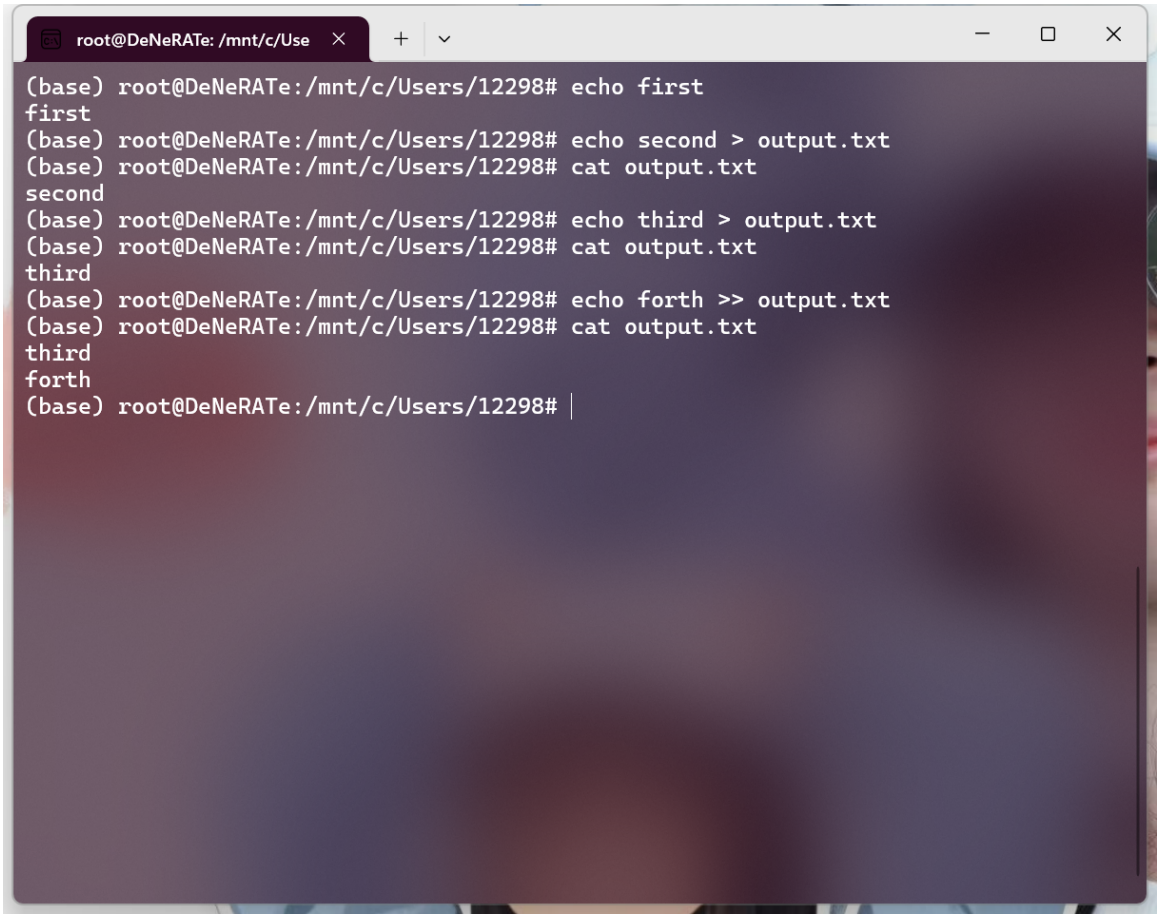
  提交说明为 3 的提交消失
  ```

- 使用 `git reset --hard <hash>` 之后再使用 `git log`

  提交说明为 2 的提交消失

- git 会保存每次提交后的工程状态，跳转到某一个版本之后，其会被视为当前状态

Thinking 0.5



Thinking 0.6

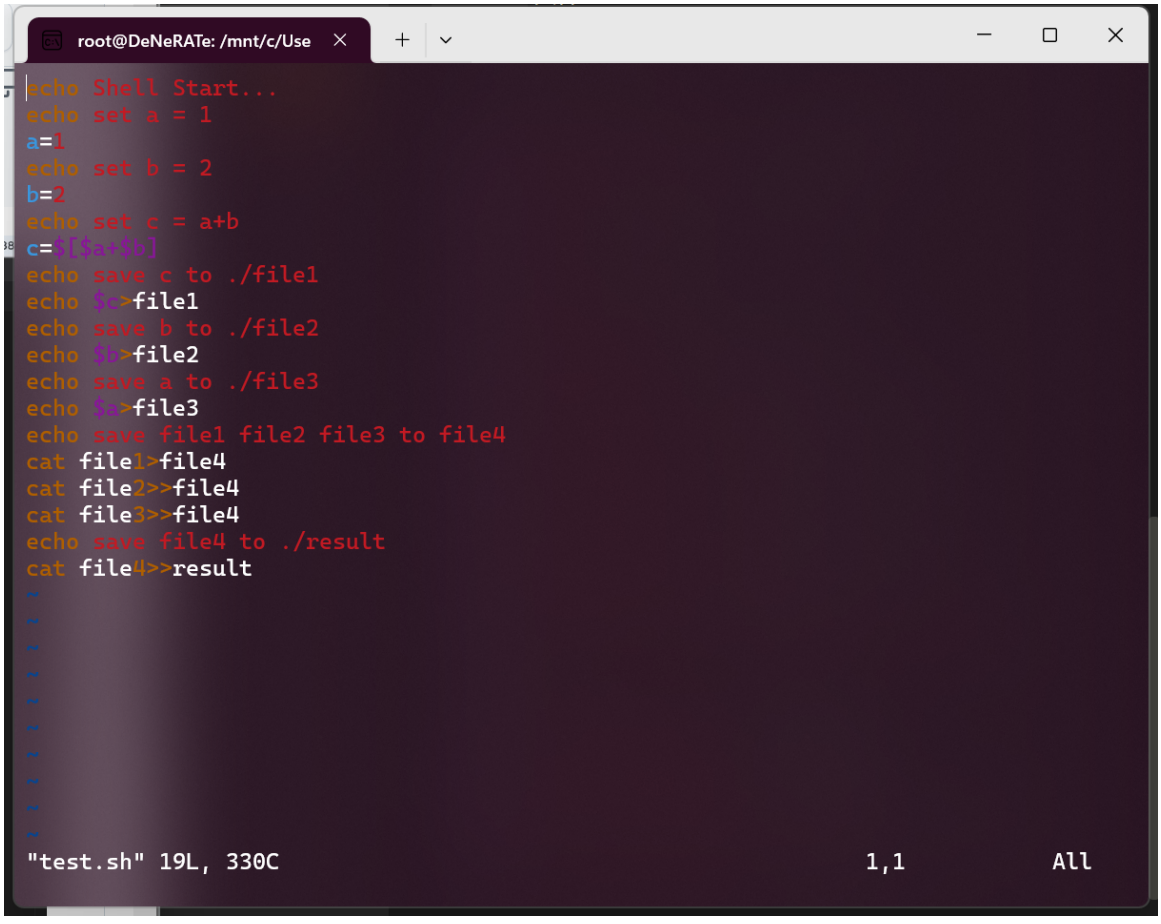
`command` 文件

```
#!/usr/bin/env bash

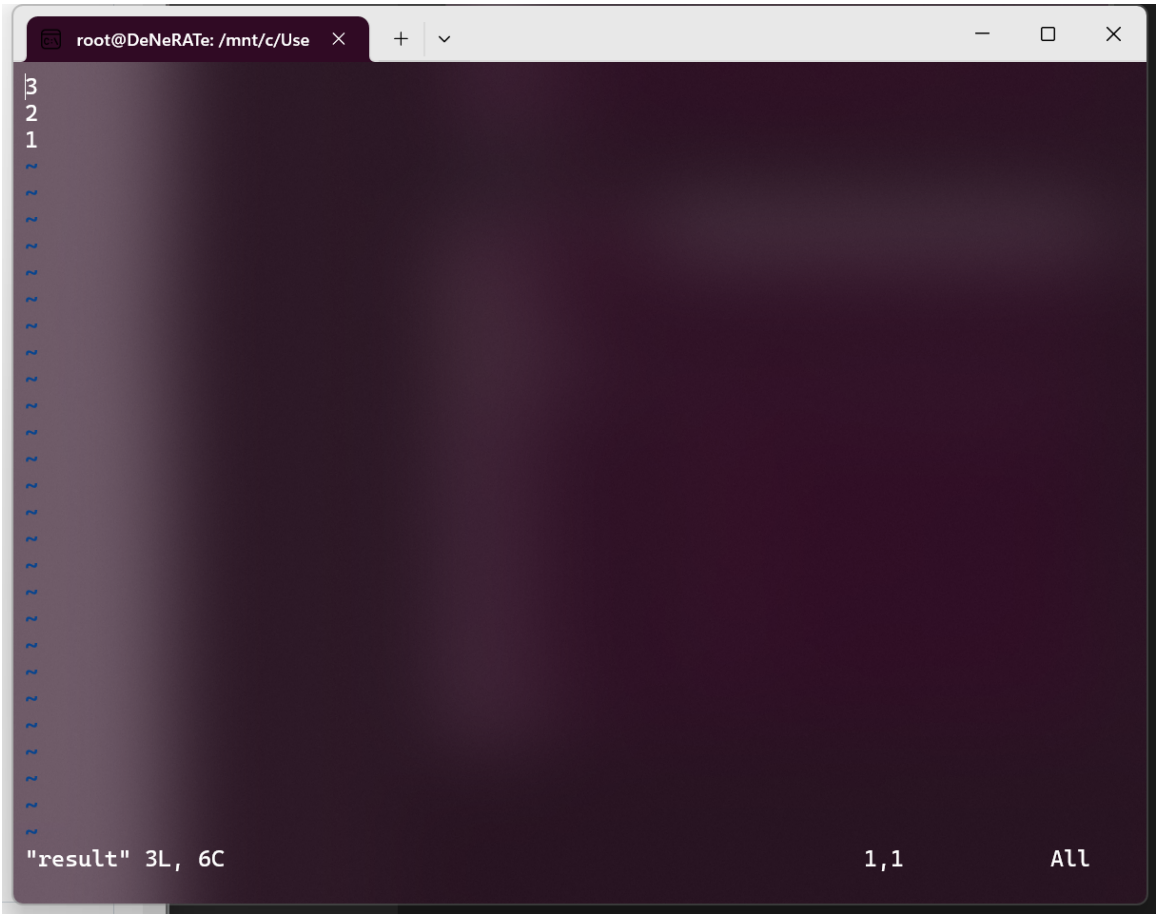
echo 'echo Shell Start...' > test.sh
echo 'echo set a = 1' >> test.sh
echo 'a=1' >> test.sh
echo 'echo set b = 2' >> test.sh
echo 'b=2' >> test.sh
echo 'echo set c = a+b' >> test.sh
echo 'c=$[$a+$b]' >> test.sh
echo 'echo save c to ./file1' >> test.sh
```

```
echo 'echo $c>file1' >> test.sh
echo 'echo save b to ./file2' >> test.sh
echo 'echo $b>file2' >> test.sh
echo 'echo save a to ./file3' >> test.sh
echo 'echo $a>file3' >> test.sh
echo 'echo save file1 file2 file3 to file4' >> test.sh
echo 'cat file1>file4' >> test.sh
echo 'cat file2>>file4' >> test.sh
echo 'cat file3>>file4' >> test.sh
echo 'echo save file4 to ./result' >> test.sh
echo 'cat file4>>result' >> test.sh
```

`test.sh` 文件



`result` 文件



解释分析

```
echo Shell Start... # 输出内容到屏幕
echo set a = 1 # 输出内容到屏幕
a=1 # 设置变量 a = 1
echo set b = 2 # 输出内容到屏幕
b=2 # 设置变量 b = 2
echo set c = a+b # 输出内容到屏幕
c=$[$a+$b] # 计算 c = a + b
echo save c to ./file1 # 输出内容到屏幕
echo $c>file1 # 将 c 的值重定向到 file1 中
echo save b to ./file2 # 输出内容到屏幕
echo $b>file2 # 将 b 的值重定向到 file2 中
echo save a to ./file3 # 输出内容到屏幕
echo $a>file3 # 将 a 的值重定向到 file3 中
echo save file1 file2 file3 to file4 # 输出内容到屏幕
cat file1>file4 # file1 的内容输出到 file4
cat file2>>file4 # file2 的内容接到 file4 后
cat file3>>file4 # file3 的内容接到 file4 后
echo save file4 to ./result # 输出内容到屏幕
cat file4>>result # file4 的内容接到 result 后
```

- `echo echo Shell Start` 和 `echo 'echo Shell Start'` 效果上没有区别
- `echo echo $c > file1` 和 `echo 'echo $c > file1'` 效果有区别，前者将 `echo $c` 的内容写入 file1；后者将 `echo $c>file1` 输出到屏幕

实验难点

- Linux Shell语言的语法规范，需要通过查阅资料和不断练习，**总结**

> - *`sed -i 's/$2/$3/g' $1` 在 `modify.sh` 中不会按预期工作，因为 `sed` 不能直接解析 **shell** 变量，需要用双引号 `"..."` 来正确解析变量。*
>
> *`-n` 只输出经过sed处理的内容，否则输出文件所有的内容*
>
> *`-i` 表示直接修改到文件中，否则只是展现修改之后长什么样*
>
> *`'s/str1/str2/'` 时，必须加最后一个 `/`，之后可以加数字表示第几次出现或者使用 `g` 表示全部，最后加上 `i` 可以实现忽略大小写*
>
> *`-e` 多个连续指令依次执行*
>
> *`&` 代表 `str1`，可以用作突出作用*
>
> > *小技巧：对于sed替换单词，可以在str1 和str2 两侧加上空格*
> >
> > ```
> > sed 's/ the / this /'
> > ```
>
> ```
> # 替换
> sed -i "s/$2/$3/g" "$1"
> # 删除 my.txt 文件的第二行到最后一行（$ 符号表示到最末尾）
> sed '2,$d' my.txt
> # 第一条是第四行后添加一个 str
> sed -e '4a\str ' -e 's/str/aaa/' my.txt
> ```
>
> - *Linux awk命令*
>
> *用于对文本内容进行分析*
>
> *如 `39: int a;` 提取行数输出为 `39`*
>
> ```
> grep "int a=1" file.txt | awk -F: '{print $1}' > output.txt
> ```

- *`grep` 常用配套*

  `grep -n PATTERN FILE` 显示行号

  `-i` 对大小写不敏感（不区分）

  `-w` 按整个单词匹配

  `-v` 找到没有出现 `PATTERN` 的行

- *`mv` 可以用于移动目录（直接移动目录内所有的内容）*
- *`cp` 配套 `cp -r SOURCE TARGET` 可以递归复制目录及其子目录内容*
- *`rm` 配套 `rm -r` 递归删除目录及其内容*

  另外 `rm -f` 强制删除

- *`cut` 命令，对于每一行进行数据处理*

  相当于 `split()` 函数，`cut -c 3` 按字符分割，`-d '.' -f 2` 为按指定分隔符分割；`-f` 指定输出的列

  输出一定位置或范围的字符 `cut -c 3,5-7`

  `cut -f 1-3` `cut -f 13-` 默认按csv文件格式进行每行的分割

  ```
  echo '0000 192.168.1.100 192.168.100.1'
  |cut -d ' ' -f 2
  192.168.1.100
  echo '0000 192.168.1.100 192.168.100.1'
  |cut -d ' ' -f 2 |cut -d '.' -f 4
  100
  echo '0000 192.168.1.100 192.168.100.1'
  |cut -d ' ' -f 2 |cut -d '.' -f 4|cut -c 1
  1
  ```

- Makefile的编写，在 Exercise 4 中，需要调用内外两层的 Makefile，需要通过查阅资料进行学习和理解

  - *`make -C` 用于 **切换到指定目录** 并在该目录执行 `make` 命令*

    ```
    make -C <目录> <目标>
    ```

    e.g. `make -C code clean`

  - *默认 gcc 的 include 路径为默认路径*

> *如果需要在当前目录下进行查找，则需要加上 `-I[INCLUDE PATH]` 参数*
>
> ```
> gcc -Iinclude code/fibo.c main.c -o fibo
> ```
>
> - *Makefile 中如果使用变量，获取值使用 `()`*
>
> ```
> INCP=-I../include
> all: fibo.c main.c
>     gcc $(INCP) -c fibo.c -o fibo.o
>     gcc $(INCP) -c main.c -o main.o
> ```

- git 工具的学习，git 是软件工程开发的重要工具，实验为大家打开了一个大门，需要我们在此多多实践，熟能生巧

体会与感想

本次实验主要是关于Linux、git、makefie的基本使用，由于之前有一定基础，总体难度不高，但对于初学者来说知识体量庞大。当然，也是需要大量时间来进行学习实践沉淀才可以获得优秀的使用经验！