

Lab0 实验报告

19373135 田旗舰

一、实验思考题

思考 0.1

通过你的使用经验，简单分析 **CLI Shell**，**GUI Shell** 在你使用过程中的各自优劣（100 字以内）

CLI:

优点：节约计算机资源，运行效率高。

在熟记命令的前提下操作比 GUI 快。

功能强大。

稳定性好。

缺点：不方便操作，学习起来比较困难

需要记忆命令

GUI:

优点：方便操作，使用更熟悉。

图形界面美观。

缺点：运行效率比 CLI 低。

功能不如 CLI 强大

思考题 0.2

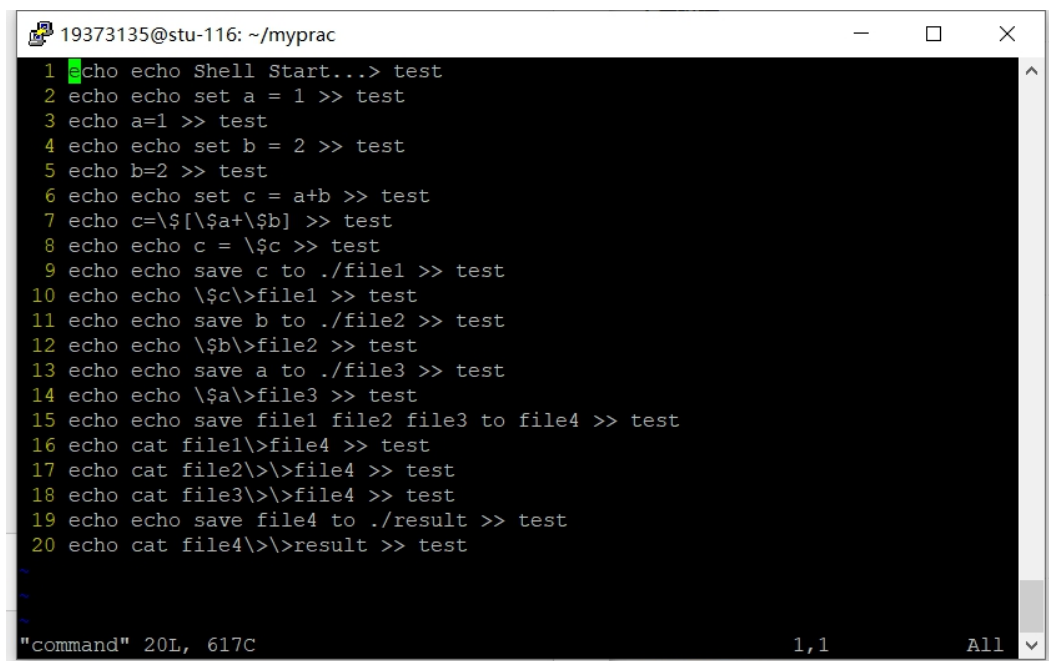
使用你知道的方法（包括重定向）创建下图内容的文件（文件命名为 **test**），将创建该文件的命令序列保存在 **command** 文件中，并将 **test** 文件作为批处理文件运行，将运行结果输出至 **result** 文件中。给出 **command** 文件和 **result** 文件的内容，并对最后的结果进行解释说明（可以从 **test** 文件的内容入手）

具体实现的过程中思考下列问题：

echo Shell Start 与 **echo 'Shell Start'**效果是否有区别

echo %>file1 与 **echo "%>file1"** 效果是否有区别

command 文件内容

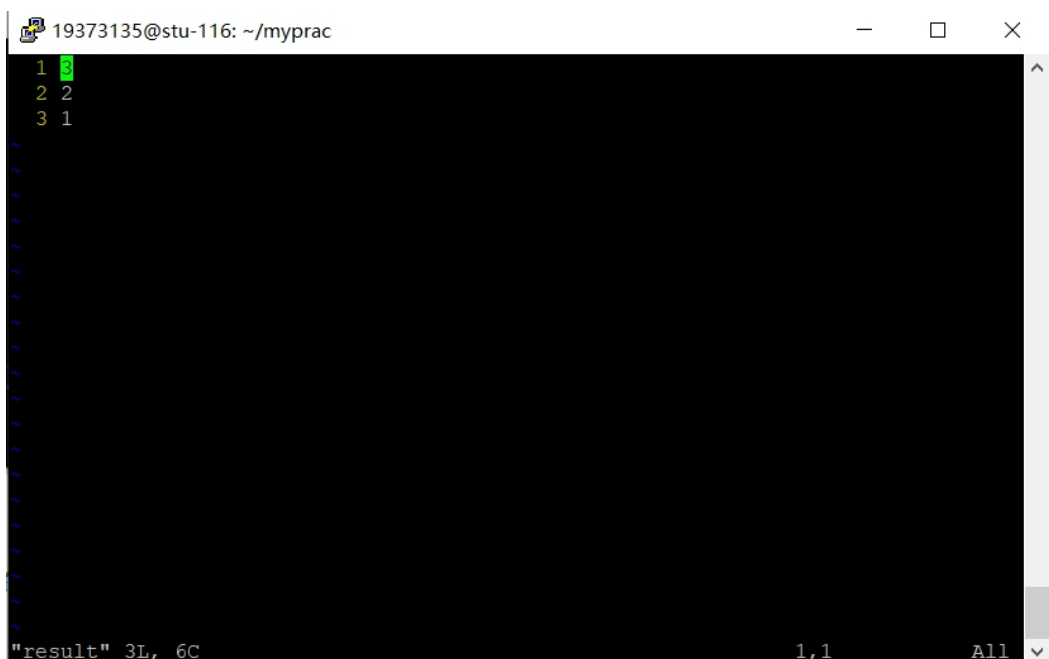


```
19373135@stu-116: ~/myprac
1 echo echo Shell Start...> test
2 echo echo set a = 1 >> test
3 echo a=1 >> test
4 echo echo set b = 2 >> test
5 echo b=2 >> test
6 echo echo set c = a+b >> test
7 echo c=\$[\$a+\$b] >> test
8 echo echo c = \$c >> test
9 echo echo save c to ./file1 >> test
10 echo echo \$c\>file1 >> test
11 echo echo save b to ./file2 >> test
12 echo echo \$b\>file2 >> test
13 echo echo save a to ./file3 >> test
14 echo echo \$a\>file3 >> test
15 echo echo save file1 file2 file3 to file4 >> test
16 echo cat file1\>file4 >> test
17 echo cat file2\>\>file4 >> test
18 echo cat file3\>\>file4 >> test
19 echo echo save file4 to ./result >> test
20 echo cat file4\>\>result >> test

"command" 20L, 617C 1,1 All
```

执行 bash command, bash test

result 文件中的结果



```
19373135@stu-116: ~/myprac
1 3
2 2
3 1

"result" 3L, 6C 1,1 All
```

test 文件的内容是将 a b c3 个变量分别赋值，并写入到对应文件 file3 file2 file1 中，再将文件内容写入到 file4 中。

echo ‘Shell Start’ 与 echo Shell Start 直接执行效果相同，但如果重定向到文件中则写入内容不同，分别为 echo ‘Shell Start’和 echo Shell Start。

echo \\$c>file1 将变量 c 写入 file1 中，echo "\\$c>file1"直接输出 \\$c>file1。

思考题 0.3

仔细看看这张图，思考一下箭头中的 `add the file`、`stage the file` 和 `commit` 分别对应的是 Git 里的哪些命令呢？

```
git add  git add  git commit
```

思考题 0.4

深夜，小明在做操作系统实验。困意一阵阵袭来，小明睡倒在了键盘上。等到小明早上醒来的时候，他惊恐地发现，他把一个重要的代码文件 `printf.c` 删除掉了。苦恼的小明向你求助，你该怎样帮他把代码文件恢复呢？

```
git checkout printf.c
```

正在小明苦恼的时候，小红主动请缨帮小明解决问题。小红很爽快地在键盘上敲下了 `git rm printf.c`，这下事情更复杂了，现在你又该如何处理才能弥补小红的过错呢？

```
git reset HEAD printf.c
```

```
git checkout printf.c
```

处理完代码文件，你正打算去找小明说他的文件已经恢复了，但突然发现小明的仓库里有一个叫 `Tucao.txt`，你好奇地打开一看，发现是吐槽操作系统实验的，且该文件已经被添加到暂存区了，面对这样的情况，你该如何设置才能使 `Tucao.txt` 在不从工作区删除的情况下不会被 `git commit` 指令提交到版本库？

```
git rm --cached printf.c
```

思考题 0.5

思考下面四个描述，你觉得哪些正确，哪些错误，请给出你参考的资料或实验证据。

1.克隆时所有分支均被克隆，但只有 **HEAD** 指向的分支被检出。

正确

节选自官网

DESCRIPTION

Clones a repository into a newly created directory, creates remote-tracking branches for each branch in the cloned repository (visible using `git branch --remotes`), and creates and checks out an initial branch that is forked from the cloned repository's currently active branch.

After the clone, a plain `git fetch` without arguments will update all the remote-tracking branches, and a `git pull` without arguments will in addition merge the remote master branch into the current master branch, if any (this is untrue when `--single-branch` is given; see below).

This default configuration is achieved by creating references to the remote branch heads under `refs/remotes/origin` and by initializing `remote.origin.url` and `remote.origin.fetch` configuration variables.

2.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

```
# git push 用于从本地版本库推送到服务器远程仓库
$ git push
```

3.克隆时只有远程版本库 **HEAD** 指向的分支被克隆。

错误

When the repository to clone from is on a local machine, this flag bypasses the normal "Git aware" transport mechanism and clones the repository by making a copy of HEAD and everything under objects and refs directories. The files under `.git/objects/` directory are hardlinked to save space when possible.

4.克隆后工作区的默认分支处于 **master** 分支。

正确

初学者最容易犯的一个错误是，克隆代码后马上进行编译。但是克隆代码时默认处于 master 分支，而我们实验的代码测试不是在 master 分支上进行，所以首先要使用 `git checkout` 检出对应的 labx 分支，再进行测试。课上测试时也要先看清楚分支再进行代码编写和提交。

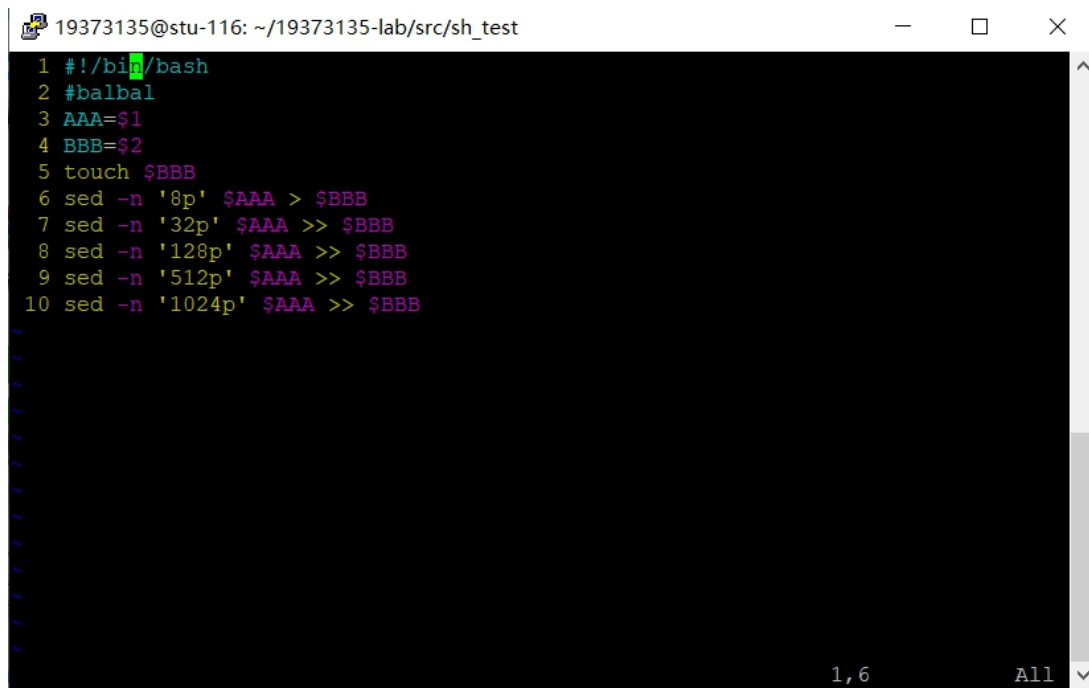
二、实验难点图示

本次实验是为了认识操作系统实验环境并掌握操作系统实验所需的基本工具而进行的铺垫性实验，仅仅是熟悉 Linux 操作系统（Ubuntu），了解控制终端，掌握一些常用工具并能够脱离可视化界面进行工作，总体而言难度并不高。

但我在上机以及课下的学习中还是遇到了一些困难，主要是由于以前从未接触过 Linux 操作系统，对于各类工具也不够熟练，基本上所有的内容都是边学边做的状态。

具体来说实验的难点有：

1.shell 编程

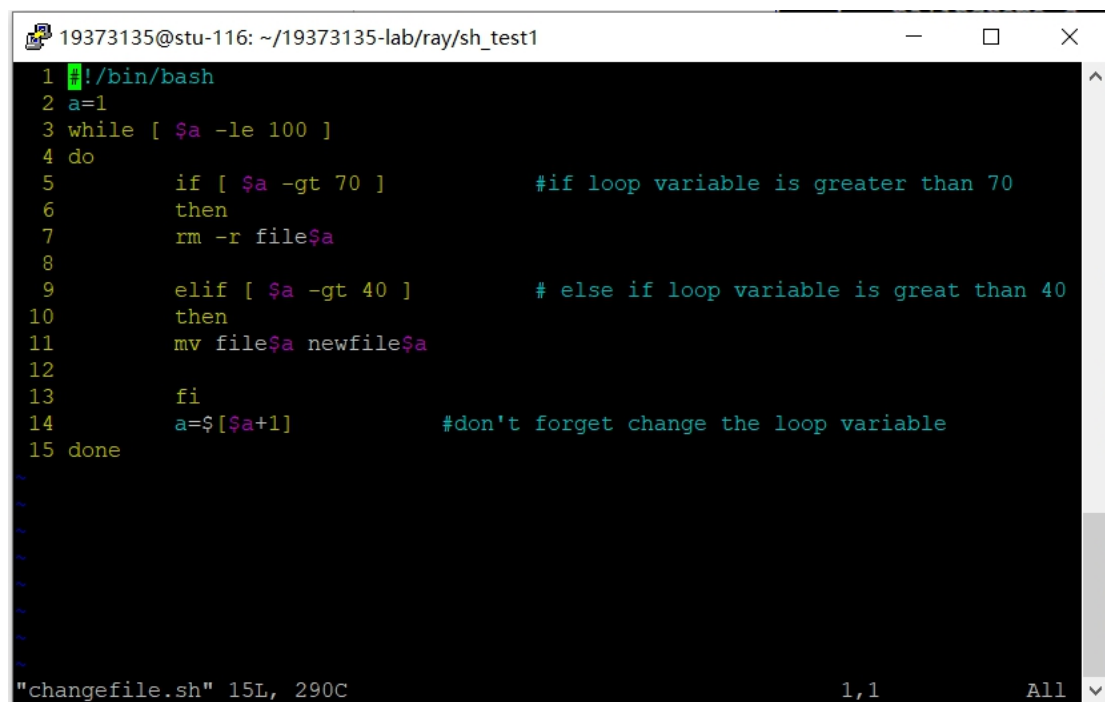


```
19373135@stu-116: ~/19373135-lab/src/sh_test
1 #!/bin/bash
2 #balbal
3 AAA=$1
4 BBB=$2
5 touch $BBB
6 sed -n '8p' $AAA > $BBB
7 sed -n '32p' $AAA >> $BBB
8 sed -n '128p' $AAA >> $BBB
9 sed -n '512p' $AAA >> $BBB
10 sed -n '1024p' $AAA >> $BBB
```

对于 shell 的编程不熟练，对于 shell 的理解也不太到位。

在 task3 中需要用到 sed 和重定向的相关知识，尽管教程中有所讲解，但我在上机时仍然花费了大量的时间，最后又查阅了相关的资料才理解了什么是重定向，尽管现在看了并不是很难，但在上机时仍是一个难点。

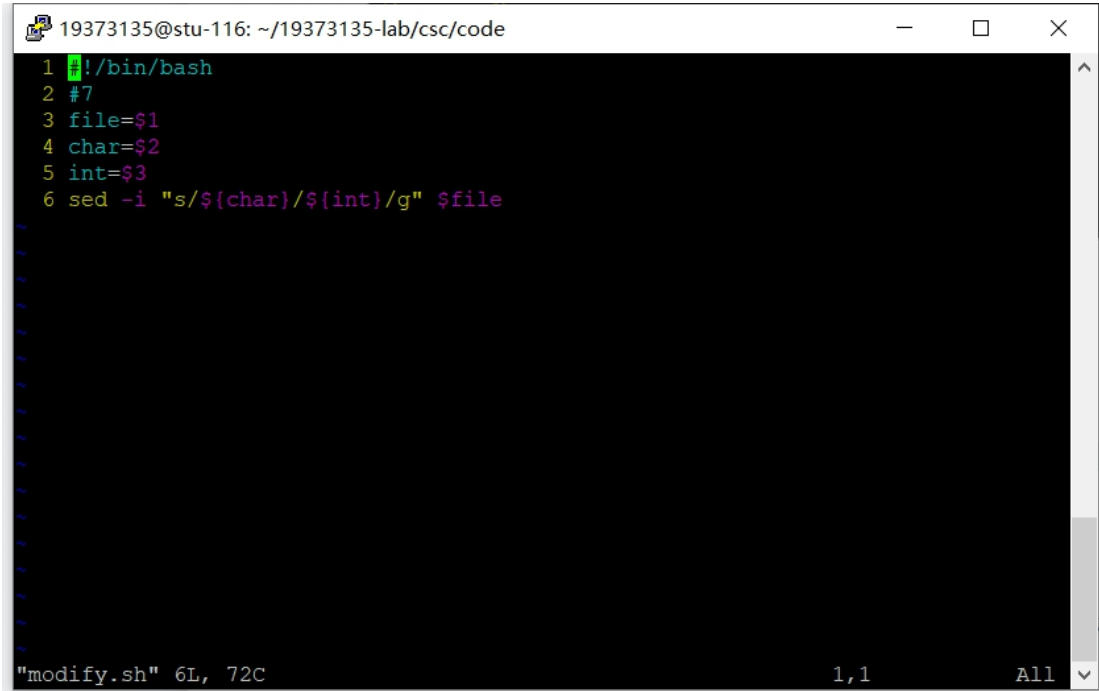
此外，在 task5 中对于 shell 的流程控制也不够熟练，对于 if else 的结构学习了一些时间。



```
19373135@stu-116: ~/19373135-lab/ray/sh_test1
1 #!/bin/bash
2 a=1
3 while [ $a -le 100 ]
4 do
5     if [ $a -gt 70 ]           #if loop variable is greater than 70
6     then
7         rm -r file$a
8     elif [ $a -gt 40 ]        # else if loop variable is great than 40
9     then
10        mv file$a newfile$a
11    fi
12    a=$((a+1))                #don't forget change the loop variable
13 done
```

"changeFile.sh" 15L, 290C

在 task7 中也涉及到 shell 编程，对于 sed 命令用双引号表示传入参数（或者用在单引号的命令中用单引号表示传入参数）也是一个难点。

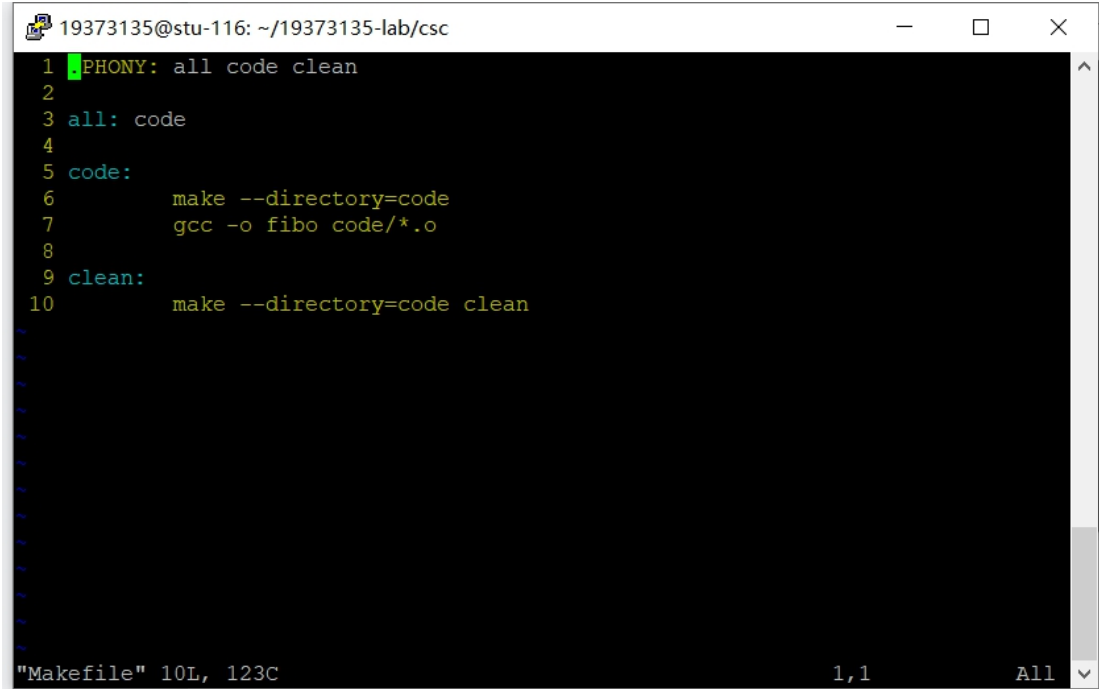


```
19373135@stu-116: ~/19373135-lab/csc/code
1  !/bin/bash
2  #7
3  file=$1
4  char=$2
5  int=$3
6  sed -i "s/${char}/${int}/g" $file

"modify.sh" 6L, 72C 1,1 All
```

2.makefile 编写

在 task8 中对 makefile 的编写是我认为 lab0 中最难的一点



```
19373135@stu-116: ~/19373135-lab/csc
1  .PHONY: all code clean
2
3  all: code
4
5  code:
6      make --directory=code
7      gcc -o fibo code/*.o
8
9  clean:
10     make --directory=code clean

"Makefile" 10L, 123C 1,1 All
```

对于 .PHONY 的理解，利用 make --directory=code 调用子目录的 makefile，以及对于 gcc 选项的应用对我来说都是比较难的地方。

三、体会与感想

学习新东西的开始总是痛苦的，但熟悉以后就会好很多。

lab0 总体难度适中，可能对于熟悉 Linux 的同学来说较为简单，而对于我这样的新手来说有一定的难度。但同时，对于我的提升也是很大的，很有收获。

我在 lab0 的学习时间大概为 8 个小时，包括上机的 3 小时和课下的 5 小时，但对我来说还不太够，我会再花费一些时间学习 lab0 的相关操作，以便在下周的上机中更加熟练。

四、指导书反馈

感觉实例少了一些，例如在重定向输入输出与 sed 等工具的讲解后最好能加上一些实际操作的截图让我们更直观地了解其含义。（当然我理解课程组可能是考虑到希望我们自己实际操作，但我仍觉得这样的方法更有利于新手的学习）

五、残留难点

lab0 中感觉最难的还是 makefile 的编写，虽然解决了 task8，但还有很多的知识没有掌握，而官方的教程感觉看不太下去，希望能了解一下大家都是怎么从 0 开始学习 makefile 的编写的。