# 实验思考题

## 0.1

CLI与GUI都是一种通过可视化界面来向用户提供操作系统的一些服务的手段。功能上可以说差别不大，但实现过程有所不同

- CLI不需要图形化界面，成本低功耗低，通过字符操作使得用户输入命令快，但对于初学者不友好
- GUI通过鼠标配合操作，可以让任何不了解操作系统的人也轻松使用计算机，但不适合学习操作系统本身的学生学习使用

## 0.2

- command：`cat "string" > test.sh`
  - 其中，string为图片内容的某一行，逐行进行
  - 如果图片中所示的是一个文件的话，设其文件名为`file.txt`，则command为：`cat file.txt > test.sh`
- result：

  `3`

  `2`

  `1`
- 解释

  file4依次经历了file1覆盖写，file2拼接写，file3拼接写

## 0.3

- add: git add
- stage: git add
- commit: git commit

## 0.4

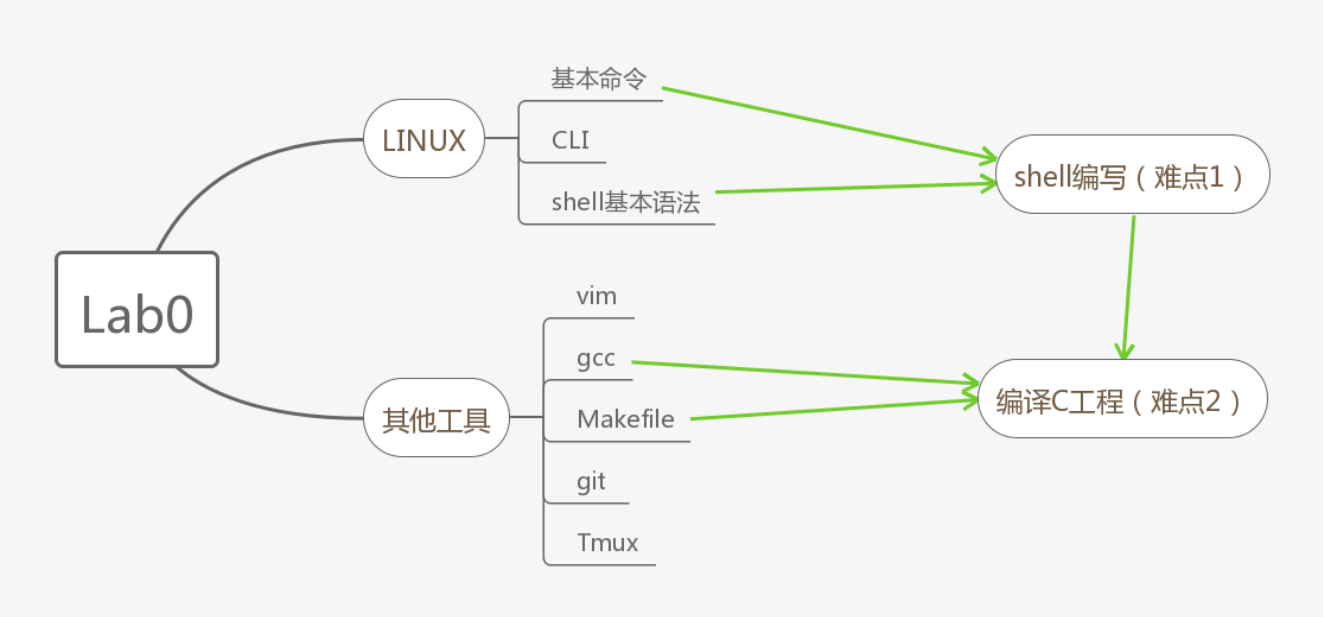
- git checkout -- printf.c
- git reset HEAD -- printf.c

  git checkout -- printf.c
- git rm - cached printf.c

## 0.5

1. 错误 实验证据：通过github验证
2. 正确 实验证据：本地git commit如果不push不会影响远程库+通过github验证
3. 错误 实验证据：https://blog.csdn.net/qq_28903377/article/details/82978583 +通过github验证
4. 正确 实验证据：通过github验证

# 难点




# 体会与感想

### 难度评价

总体难度约等于计组P1，即与OS内核本身无关的东西（与CPU本身无关的）比较多，还特别杂，但又是继续利用Linux等工具学习的必经之路（ISE的操作，Verilog的语法），我当时给P1打的难度分为5/10，是除了P7以外最高的

### 花费时间

很惭愧，虽然linux完全小白，但是因为OO的缘故，分配的时间并不多。重点学习的也就是shell的基本指令，顺便仿照awk的功能做了一个逐字符处理文件的脚本（虽然课上并没有用到）

面对课上题目，尤其是gcc与Makefile相关，更是无从下手，最后只好靠cp或者rm等指令狼狈过关

### 感想

还是要在Lab1前再好好地过一遍Makefile和gcc的用法，顺便为了防止这次只重视文本处理而险些挂掉的惨案，提前看一下其他两款工具（模拟器和交叉编译工具链）的使用。

# 指导书反馈

建议加入更多的样例，比如很多指令的具体用法（就是那种敲到终端就能有效果的例子，不是形如“grep [] xxx []"这样的通用类型）之类的，或者加入一些对于“选项”“命令”之类的描述，比如，“awk -F blabla,其中这个F是blabla”，更方便像我这种完全没有接触过linux和命令行的人学习~~（p.s. 如果课程组的本意是要让我们自学，练习查阅资料，那这条建议作废）~~

建议仿照lab0提交文件树/样例那样，给lab1提供更多的资料，或者在cscore上面放出更多的资料~~（p.s. 如果课程组本意是让我们练习查看指导书（将来可能的说明书），并锻炼学习陌生事物的能力的话，那这条建议作废）~~

# 残留难点

Makefile的巧妙用法，由于各种主观原因和极其微小的客观原因（时间），我并没有掌握，只会用基本的“target：”，“变量名或者宏（也许这个称呼并不准确）”，gcc的基本语法，除此之外就完全当做shell脚本来写了。这一部分课下我要更加投入的去学习