**胡柯洋 17373197**

**一、实验思考题**

**思考题0.1**

CLI Shell：纯文本方式工作，在熟悉相关指令后操作起来简便高效，可将命令保存为脚本以调用。支持批量操作，严格执行指令，软件要求低，适配性强。但操作不直观，人机交互性弱，上手难度大，若对指令不熟悉用起来举步维艰。

GUI Shell：图形界面工作，用直观的图标标识命令，通过鼠标、键盘作为输入工具，界面直观，人机交互性强，可以直接观察到执行结果，易于上手。但GUI软件效率低，需要花时间寻找相应图标。

**思考题0.2**

command文件内容：

echo 'echo Shell Start...' > test

echo 'echo set a = 1' >> test

echo 'a=1' >> test

echo 'echo set b = 2' >> test

echo 'b=2' >> test

echo 'echo set c = a+b' >> test

echo 'c=$[$a+$b]' >> test

echo 'echo c = $c' >> test

echo 'echo save c to ./file1' >> test

echo 'echo $c>file1' >> test

echo 'echo save b to ./file2' >> test

echo 'echo $b>file2' >> test

echo 'echo save a to ./file3' >> test

echo 'echo $a>file3' >> test

echo 'echo save file1 file2 file3 to file4' >> test

echo 'cat file1>file4' >> test

echo 'cat file2>>file4' >> test

echo 'cat file3>>file4' >> test

echo 'echo save file4 to ./result' >> test

echo 'cat file4>>result' >> test

result文件内容：

3

2

1

结果解释：

（1）命令行输出内容为：  
 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

先在命令行输出

Shell Start...

set a = 1

set b = 2

set c = a+b内容，并对a，b分别赋值为1，2；

c=$[$a+$b]将参数a与b的值之和赋给c

echo c = $c 在命令行输出”c =” + 变量c的值

接下来echo save c to ./file1

echo $c>file1

echo save b to ./file2

echo $b>file2

echo save a to ./file3

echo $a>file3 这些命令

在命令行输出 save c to ./file1

save b to ./file2

save a to ./file3

同时将c的值保存到file1，b的值保存到file2，a的值保存到file3

接下来一句命令在命令行输出 save file1 file2 file3 to file4

cat file1>file4

cat file2>>file4

cat file3>>file4

这三句命令将cat的输出结果重定向至file4，（第一句重定向并覆盖file4内容，后两句追加输出接在第一句的下方）

此时file4的内容应为 3

2

1

echo save file4 to ./result 在命令行输出 save file4 to ./result

cat file4>>result 将cat file4结果重定向输出到新建文件result

故最终result文件内容为 3

2

1

**思考题0.3**

add the file----git add <file>

staged the file---git add <file>

commit-------git commit

**思考题0.4**

1.

（1）若小明还没未add或commit过此文件，那他凉了

（2）若小明已经add或commit文件，则使用git checkout -- <file>

1）file自修改后还没有被放到暂存区，就回到和版本库一模一样的状态；

2）已经添加到暂存区后，就回到添加到暂存区后的状态；

2.

（1）同样地，若小明还未commit（未提交到版本库），那他继续凉

（2）若此文件已经提交到版本库git reset HEAD <file>撤销对暂存区的修改

或先使用git log查看版本控制历史记录，决定要回退到哪个版本，再git reset -hard+版本号

3. git rm --cache <file>删除错误添加到暂存区中的文件

**思考题0.5**

1.正确

Git-scm 官网教程中所述：“这是 Git 区别于其它版本控制系统的一个重要特性，Git 克隆的是该 Git 仓库服务器上的几乎所有数据，而不是仅仅复制完成你的工作所需要文件。 当你执行 git clone 命令的时候，默认配置下远程 Git 仓库中的每一个文件的每一个版本都将被拉取下来。”clone在本地复制了一个最新的仓库，当下的版本HEAD就是目前被检出的版本。

2.正确

这些操作都是在本地仓库中进行修改，与暂存区，本地版本库有关，不联网也可进行操作，并不会访问远程版本库。

3.错误

Git 克隆的是该 Git 仓库服务器上的几乎所有数据，而不是仅仅复制完成你的工作所需要文件。 当你执行 git clone 命令的时候，默认配置下远程 Git 仓库中的每一个文件的每一个版本都将被拉取下来。

4.正确

Git-scm 官网教程给出的克隆远程分支的例子。假设你的网络里有一个在 git.ourcompany.com 的 Git 服务器。 如果你从这里克隆，Git的clone 命令会为你自动将其命名为 origin，拉取它的所有数据，创建一个指向它的 master 分支的指针，并且在本地将其命名为 origin/master

**二、实验难点**

GUI向CLI的风格转变

Makefile的书写格式——command前面需加Tab

编写hello\_os.sh时中对变量的引入（$1,$2）

在对sed的输出重定项时，第一项需用>覆盖原文件内容，第二项之后用>>接在文件后

Gcc编译器的使用格式 (gcc -o fibo fibo.c)

**三、体会与感想**

课下需要花大量时间熟练命令行窗口的使用， 第一次接触命令行时觉得这个界面很不友好，多次使用后发现命令行的直接与便利，对LINUX命令行界面我需要进行更多的探索。在命令行风格感受到了vim编辑器的强大之处，在熟悉操作指令后对文档内容的操作都显得非常直观，特别方便。

第一次进行脚本文件的编写，了解到了在我们日常使用的图形窗口的后台在同时进行着多么复杂的操作，实验课给了我们直接接触系统内部运行模式的机会。

熟悉对Git强大的分支管理能力的引用，使用Putty连接远程服务器后，可在lab0与lab1分支之间自由切换，并且可以很方便地进行提交评测。

希望能在接下来的实验课上不断学习，熟悉命令行与Git的使用，成为一名合格的程序员。