

1、实验思考题

Thinking 0.1

Modified.txt 和 Untracked.txt 中所保存的status信息显然不一样，如下所示：

Untracked.txt:

```
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    README.txt
    Untracked.txt

nothing added to commit but untracked files present (use "git add" to track)
```

Modified.txt:

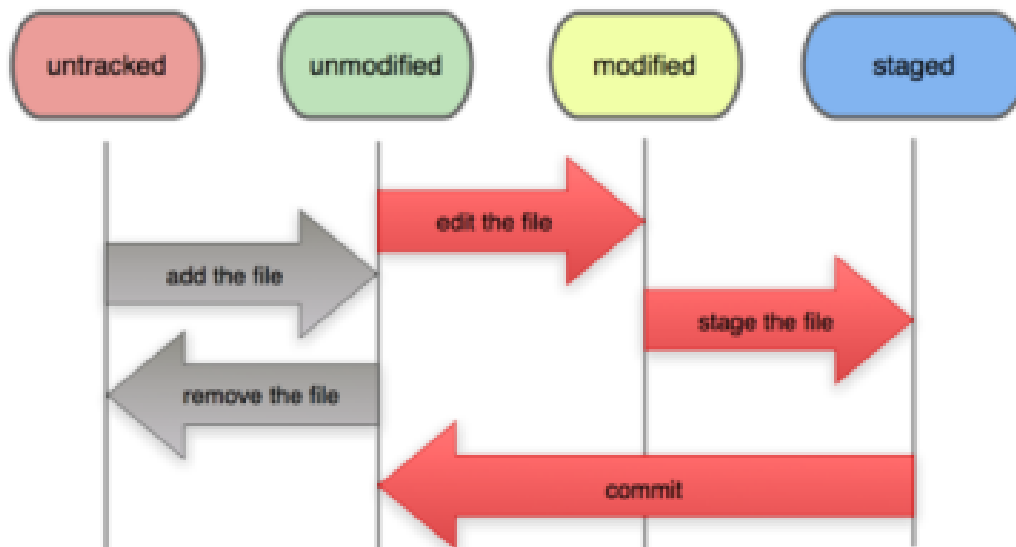
```
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   README.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    Modified.txt
    Stage.txt
    Untracked.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

根据第一次add前的status信息，可以知道，在第一次add前，文件 README.txt 处于未跟踪的状态，即未被git跟踪，对应下图的untracked状态；而最后一次修改未提交时，文件 README.txt 处于已修改态，也即未暂存态，对应下图的modified状态，此时该文件被存在修改内容，但修改内容未被暂存，需要使用 git add 将文件变为暂存态，并使用 git commit 提交修改，将文件变为未修改状态 (unmodified)，如果再继续remove该文件，则回到开始的untracked状态，跟踪丢失。

File Status Lifecycle



Thinking 0.2

在上图中：

- 1、`add the file`对应 `git add <file>` 指令；
- 2、`stage the file` 也对应 `git add <file>` 指令；
- 3、`commit`对应 `git commit` 指令。

Thinking 0.3

- 1、工作区本地文件 `printf.c` 被删除，可以使用 `git checkout printf.c` 从暂存区来恢复；
- 2、在使用 `git rm printf.c` 将暂存区的文件也删除，若还想要在工作区恢复，可以先使用 `git reset HEAD printf.c` 从HEAD恢复到暂存区，再使用 `git checkout printf.c` 从暂存区来恢复；
- 3、如果有工作区的文件 `Tucaao.txt` 已被添加到暂存区，可以使用 `git rm --cached Tucao.txt` 将该文件从暂存区删除，但不影响工作区，之后再使用 `git commit` 将暂存区文件提交到版本库。

Thinking 0.4

- 1、使用 `git log` 可以查看提交日志，每个提交版本都有其对应的哈希值；
- 2、使用 `git reset --hard HEAD^` 可以回退到上一次 `commit` 的版本，工作区和暂存区都会发生回退，加几个 `^` 符号就会向前回退几个版本，在某一版的更新出现问题，需要回到上一次提交的版本时，可以利用这一命令实现版本的回退和恢复；
- 3、使用 `git reset --hard <Hash-code>` 可以利用每个版本的哈希值（对应下图黄字部分）在不同版本之间进行切换，如果需要回退的版本太多，或者需要回退到指定版本时，可以采用此命令；
- 4、利用 `git` 工具可以进行版本的更新、切换，对版本内容进行维护，对于软件开发有着重要的作用。

```
git@21210110:~/test_dir/learnGit$ git add test.txt
git@21210110:~/test_dir/learnGit$ git commit -m 2
[master 9488a15] 2
 1 file changed, 2 insertions(+)
 create mode 100644 test.txt
git@21210110:~/test_dir/learnGit$ vim test.txt
git@21210110:~/test_dir/learnGit$ git add test.txt
git@21210110:~/test_dir/learnGit$ git commit -m 3
[master c573ed0] 3
 1 file changed, 1 insertion(+)
git@21210110:~/test_dir/learnGit$ git log
commit c573ed0ccd6de9c2aa9eafc1b69af8d277608beb (HEAD -> master)
Author: Du Jianan <2603605747@qq.com>
Date:   Fri Mar 18 15:06:09 2022 +0800

    3

commit 9488a15c2be919f77c287334a293c977e6f98ed0
Author: Du Jianan <2603605747@qq.com>
Date:   Fri Mar 18 15:05:35 2022 +0800

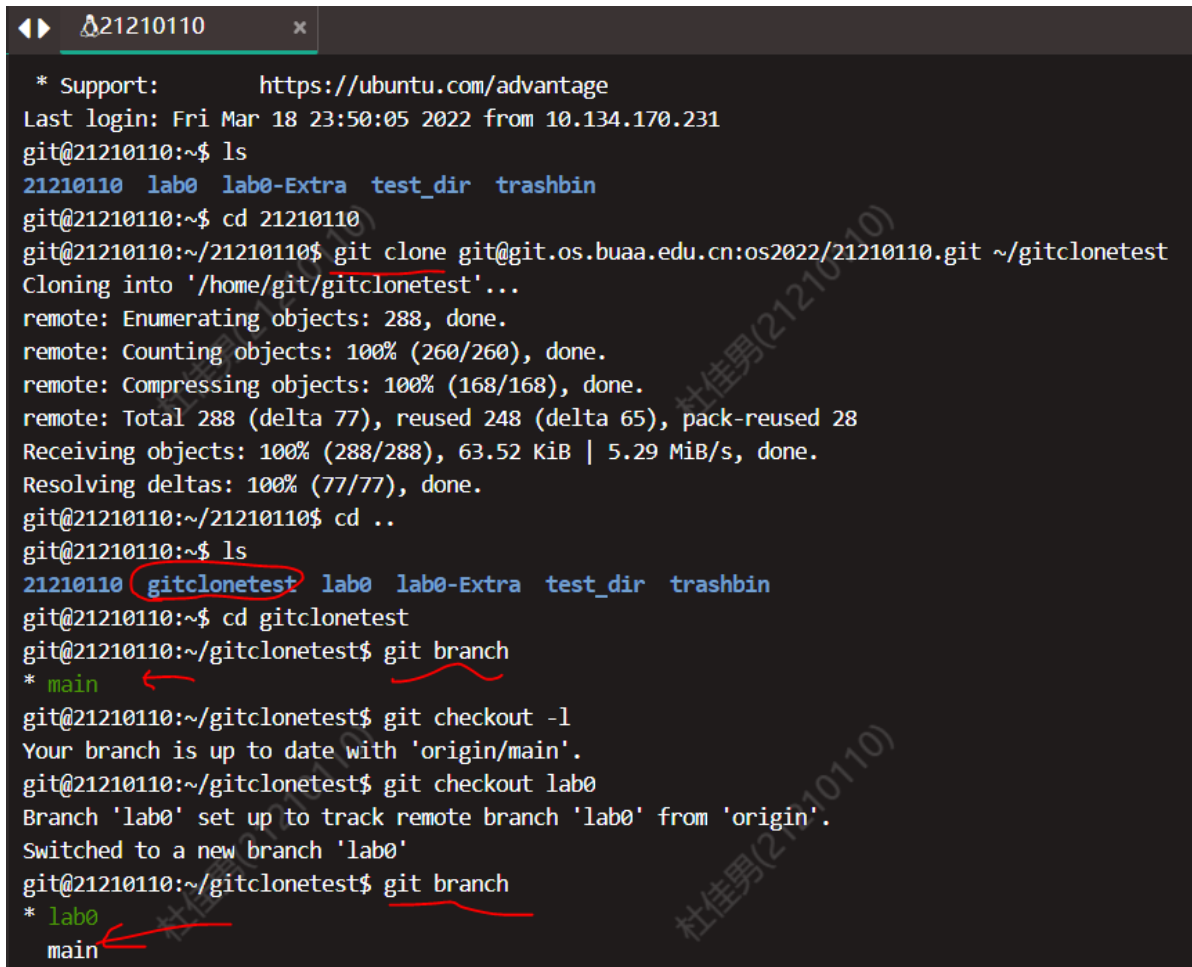
    2

commit f752c14dc07f18bfde63c55432aef893942f6f91
Author: Du Jianan <2603605747@qq.com>
Date:   Fri Mar 18 15:03:42 2022 +0800

    1
```

Thinking 0.5

1、正确，克隆时所有分支都被克隆，但需要使用 `git checkout <file>` 才能将分支检测出。如下图，对远程仓库进行克隆，使用 `git branch` 指令可以发现只检出了 `main` 分支，但可以通过 `git checkout` 跳转到其他分支，并将分支检出，而中间并没有其他的 `git clone` 或 `git pull` 等从远端拉取资源的操作，再使用 `ls` 可以发现存在远程仓库分支 `lab0` 里的文件，因此这一条正确。

A terminal window titled '21210110' showing a series of git commands and their outputs. The user clones a repository, checks out the 'lab0' branch, and lists the files. Red annotations highlight the 'gitclone' directory in the file list and the 'lab0' branch in the git branch output. A watermark '杜佳男(21210110)' is visible diagonally across the terminal output.

```
* Support:      https://ubuntu.com/advantage
Last login: Fri Mar 18 23:50:05 2022 from 10.134.170.231
git@21210110:~$ ls
21210110 lab0 lab0-Extra test_dir trashbin
git@21210110:~$ cd 21210110
git@21210110:~/21210110$ git clone git@git.os.buaa.edu.cn:os2022/21210110.git ~/gitclonetest
Cloning into '/home/git/gitclonetest'...
remote: Enumerating objects: 288, done.
remote: Counting objects: 100% (260/260), done.
remote: Compressing objects: 100% (168/168), done.
remote: Total 288 (delta 77), reused 248 (delta 65), pack-reused 28
Receiving objects: 100% (288/288), 63.52 KiB | 5.29 MiB/s, done.
Resolving deltas: 100% (77/77), done.
git@21210110:~/21210110$ cd ..
git@21210110:~$ ls
21210110 gitclonetest lab0 lab0-Extra test_dir trashbin
git@21210110:~$ cd gitclonetest
git@21210110:~/gitclonetest$ git branch
* main
git@21210110:~/gitclonetest$ git checkout -l
Your branch is up to date with 'origin/main'.
git@21210110:~/gitclonetest$ git checkout lab0
Branch 'lab0' set up to track remote branch 'lab0' from 'origin'.
Switched to a new branch 'lab0'
git@21210110:~/gitclonetest$ git branch
* lab0
main
```

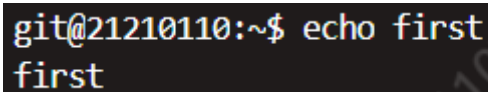
2、正确，`git log`、`git status`、`git checkout` 和 `git commit` 等操作均为对本地库的操作，不访问远程库，只有当使用 `git push` 命令时才会将本地库更新至远程库。通过网页访问远程仓库即可发现内容未被修改，截图此处不列出。

3、错误，由第一小问即可知。

4、正确，可以看到刚 clone 的工作区处于 `main` 分支，即此时的 `master` 分支。

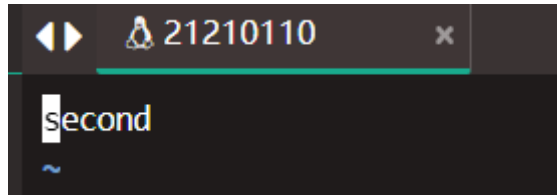
Thinking 0.6

1、执行 `echo first`，会在命令行回显 `first`；



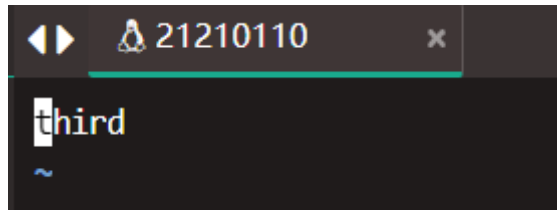
```
git@21210110:~$ echo first
first
```

2、执行 `echo second > output.txt`，会将输出 `second` 重定向到文件 `output.txt` 中，如果文件不存在，则创建该文件；



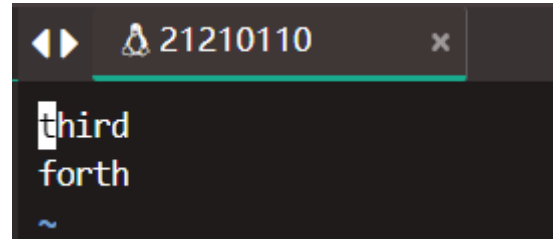
```
21210110
second
~
```

3、执行 `echo third > output.txt`，会将输出 `third` 重定向到文件 `output.txt` 中，如果文件已存在，则覆盖原有内容；



```
21210110
third
~
```

4、执行 `echo forth >> output.txt`，会将输出 `forth` 重定向到文件 `output.txt` 中，如果文件已存在，则将内容接在原有内容之后。



```
21210110
third
forth
~
```

Thinking 0.7

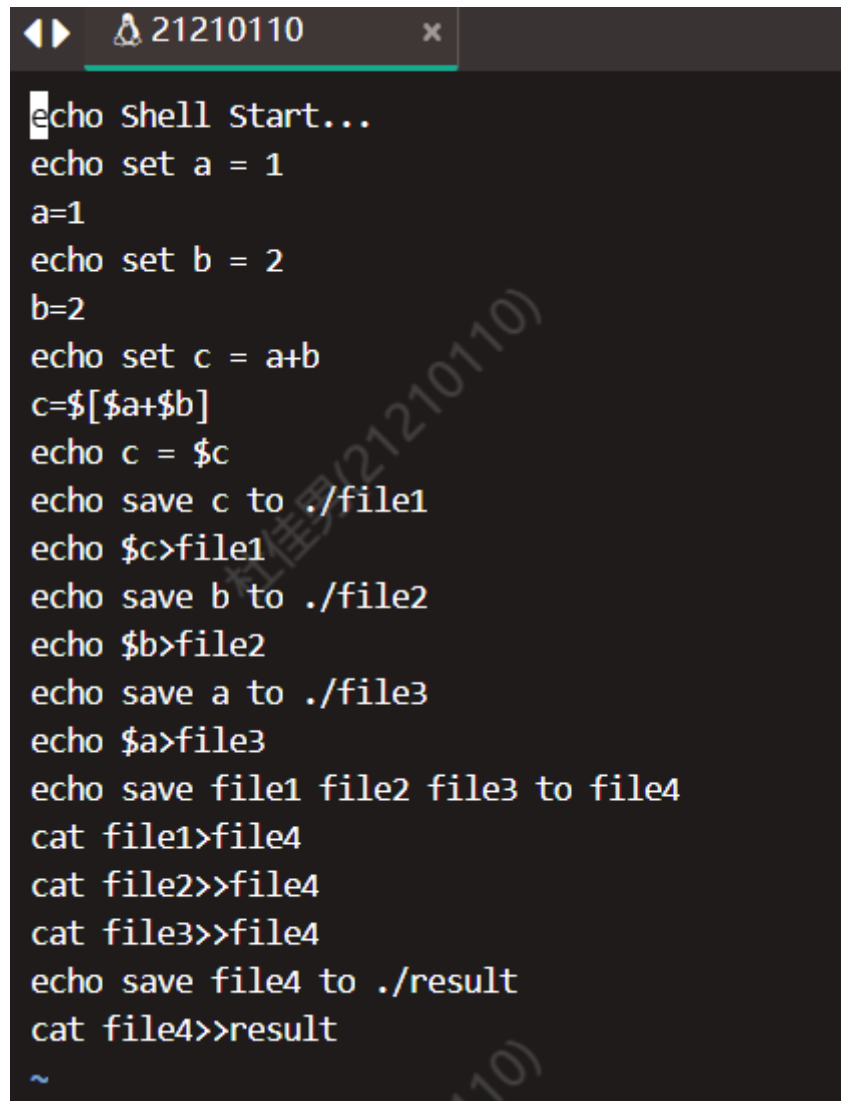
1、使用 bash 批处理运行 command 文件得到 test 文件，命令的执行界面如下图。

```
git@21210110:~/test_dir/learnBash$ ls
command my.sh my2.sh
git@21210110:~/test_dir/learnBash$ bash command
git@21210110:~/test_dir/learnBash$ ls
command my.sh my2.sh test
git@21210110:~/test_dir/learnBash$ bash test
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
git@21210110:~/test_dir/learnBash$ ls
command file1 file2 file3 file4 my.sh my2.sh result test
```

2、command 文件内容如下，其中部分需要 echo 的内容因包含 \$、>、>> 等特殊字符，需要加单引号以忽略其特殊含义当作普通文本，其余回显内容不含特殊字符的部分未标单引号。以思考题为例，echo Shell Start 中没有特殊字符，因此 echo echo Shell Start 和 echo 'echo Shell Start' 效果一样，作用都是回显 echo Shell Start；echo \$c>file 中有特殊字符，命令 echo echo \$c>file 的效果是把 echo echo \$c 的输出重定向到文件 file，且此时的 \$c 具有特殊含义，指代变量 c 的值，因此整条命令的实际效果是将字符串 "echo"+空格+变量 c 的值 写入到文件 file 里，而 echo 'echo \$c>file' 因单引号忽视特殊字符，其实际效果为回显字符串 echo echo \$c。（这里建议题目中把文件名改成 *.sh，以更直观地以高亮形式阅读代码）

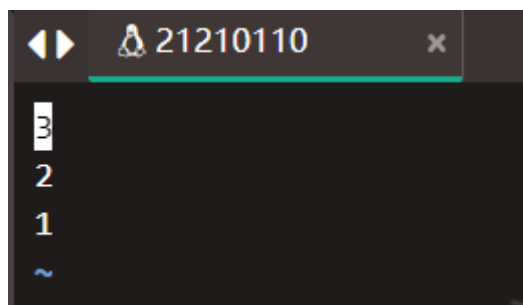
```
21210110 x
echo echo Shell Start... > test
echo echo set a = 1 >> test
echo a=1 >> test
echo echo set b = 2 >> test
echo b=2 >> test
echo echo set c = a+b >> test
echo 'c=${a+$b}' >> test
echo 'echo c = $c' >> test
echo echo save c to ./file1 >> test
echo 'echo $c>file1' >> test
echo echo save b 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
~
~
```

3、`test` 文件内容如下，与指导书一致。



```
21210110 x
echo Shell Start...
echo set a = 1
a=1
echo set b = 2
b=2
echo set c = a+b
c=${a+$b}
echo c = $c
echo save c to ./file1
echo $c>file1
echo save b to ./file2
echo $b>file2
echo save a to ./file3
echo $a>file3
echo save file1 file2 file3 to file4
cat file1>file4
cat file2>>file4
cat file3>>file4
echo save file4 to ./result
cat file4>>result
~
```

4、`result` 文件内容如下，在 `test` 文件中，`a` 的值被设为 1，`b` 的值被设为 2，`c` 的值被设为 `a` 和 `b` 的和，`a` 的值被回显重定向到文件 `file1`，`b` 的值被回显重定向到文件 `file2`，`c` 的值为 3，被回显重定向到文件 `file3`，然后 `file1` 的内容被输出到 `file4`，`file2` 和 `file3` 的内容被接在 `file4` 原有内容之后，因此得到如下的文件内容。

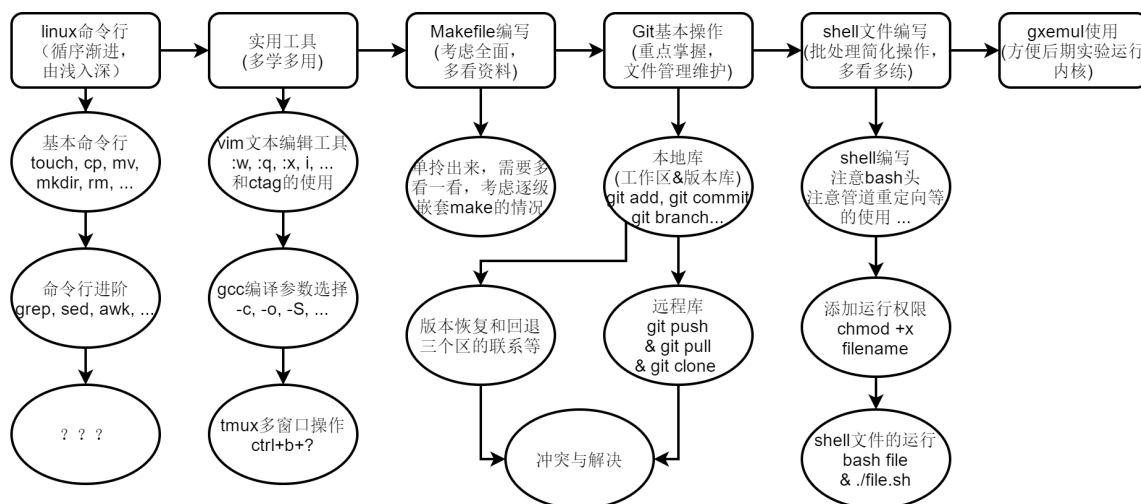


```
21210110 x
3
2
1
~
```

2、实验难点图示

个人最难的部分是Git的基本操作。

知识结构



3、体会与感受

难度评价适中，实验花费时间约4~5h，大部分时间用于阅读指导书和实操各项知识点，熟悉各种命令的使用，实验报告用时约1.5h。整体感受不错，有学到新知识，虽然内容比较多但总体来说处于可接受范围内，多看多练多学可以轻松过关。git的内容比较乱，有些难以理解，但多动手实操一会，情况会有明显地改善，目前程度的shell和makefile难度简单，无压力。往后瞄了一眼lab1，感觉相比lab1，lab0的难度真的已经算很简单了.....希望后面的实验不会太吃力。

4、指导书反馈

- 42页的Thinking 0.7，建议将 `command` 的文件名改为 `command.sh`，使用批处理文件格式，更直观地以高亮形式阅读代码，同样的，建议将 `test` 文件名改为 `test.sh`，可以在代码里加类似 `#!/bin/bash` 的语句以在 `test.sh` 文件生成sh文件的第一行，方便直接使用 `./test.sh`；
- 实验中发现，关于 `awk` 指令，如果不使用管道或者其他中间文件，如果源文件和重定向文件相同，也即使用 `awk` 对源文件进行修改，如 `awk -F : '{print $2}' > filename`，不能直接用 `>` 重定向，需要将指令改写为 `awk -F : '{print $2 > "filename"}' > filename`，如果是在 `bash` 文件里，`filename` 是该文件的参数，假定为 `$1`，还需要加单引号忽略特殊符号，为 `awk -F : '{print $2 > "'$1'"}' > $1`
- 指导书内容已经很全面啦，助教辛苦！

5、残留难点

- 错误使用 `git pull` 造成的分叉现象，debug困难；
- 还存在很多不熟悉的 `git` 指令，对 `git` 的使用还不够熟练；

