BUAA_OS_Lab0

一、实验思考题

Thinking 0.1

- 执行cat Untracked.txt后的结果：README.txt文件位于untracked区

```
git@22373180:~/learnGit (master)$ cat Untracked.txt
位于分支 master
未跟踪的文件:
  （使用 "git add <文件>..." 以包含要提交的内容）
        README.txt
        Untracked.txt

提交为空，但是存在尚未跟踪的文件（使用 "git add" 建立跟踪）
```

- 执行cat Stage.txt后的结果：README.txt文件位于staged区

```
git@22373180:~/learnGit (master)$ cat Stage.txt
位于分支 master
要提交的变更:
  （使用 "git restore --staged <文件>..." 以取消暂存）
        新文件:     README.txt
        新文件:     Untracked.txt

未跟踪的文件:
  （使用 "git add <文件>..." 以包含要提交的内容）
        Stage.txt
```

- 修改修改README.txt 文件，再执行命令git status > Modified.txt。执行命令cat Modified.txt后的结果：

```
git@22373180:~/learnGit (master)$ vim README.txt
git@22373180:~/learnGit (master)$ git status > Modified.txt
git@22373180:~/learnGit (master)$ cat Modified.txt
位于分支 master
尚未暂存以备提交的变更：
  （使用 "git add <文件>..." 更新要提交的内容）
  （使用 "git restore <文件>..." 丢弃工作区的改动）
        修改：      README.txt

未跟踪的文件：
  （使用 "git add <文件>..." 以包含要提交的内容）
        Modified.txt
        Stage.txt

修改尚未加入提交（使用 "git add" 和/或 "git commit -a"）
```

结果和第一次执行 add 命令之前的 status 不一样。第一次执行 add 命令之前的README.txt的 status是未跟踪，修改后的status是未暂存以备提交的变更

原因是执行add命令前README.txt未被跟踪，后续执行add命令文件被跟踪修改后未提交，所以处于未暂存状态。

Thinking 0.2

> Thinking0.2 仔细看看0.10，思考一下箭头中的add thefile、stage thefile和 commit分别对应的是Git里的哪些命令呢?

首先对于任何一个文件, 在 Git 中都只有四种状态: 未跟踪（untracked）、未修改（unmodified）、已修改（modified）、已暂存（staged）:

未跟踪 表示没有跟踪（add）某个文件的变化，使用 `git add` 即可跟踪文件。

未修改 表示某文件在跟踪后一直没有改动过或者改动已经被提交。

已修改 表示修改了某个文件, 但还没有加入（add）到暂存区中。

已暂存 表示把已修改的文件放在下次提交（commit）时要保存的清单中。

这里使用一张图来说明文件的四种状态的转换关系:

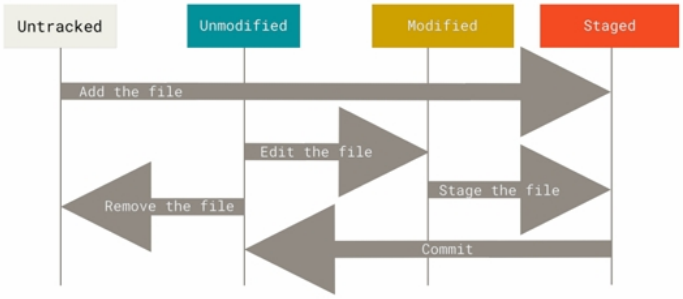


图 0.10: Git 中的四种状态转换关系

add the file 对应指令:

```
git add
```

stage the file 对应指令:

```
git add
```

commit 对应指令:

```
git commit
```

Thinking 0.3

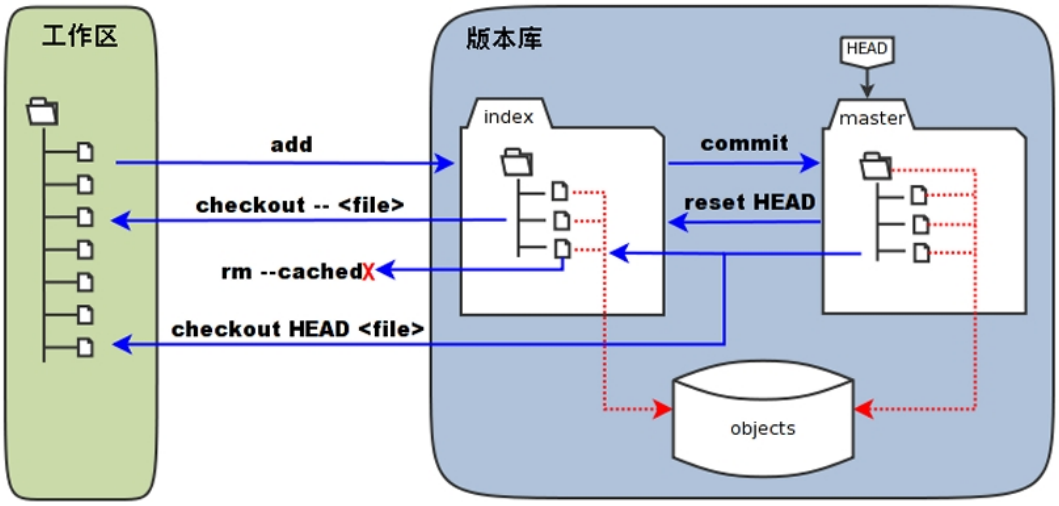


图 0.11: 工作区、暂存区和版本库

> 1、代码文件print.c 被错误删除时，应当使用什么命令将其恢复?

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

> 2、代码文件 print.c 被错误删除后，执行了 git rm print.c 命令，此时应当 使用什么命令将其恢复?

```
git checkout HEAD print.c
```

> 3、无关文件 hello.txt 已经被添加到暂存区时，如何在不删除此文件的前提下将其移出暂存区?

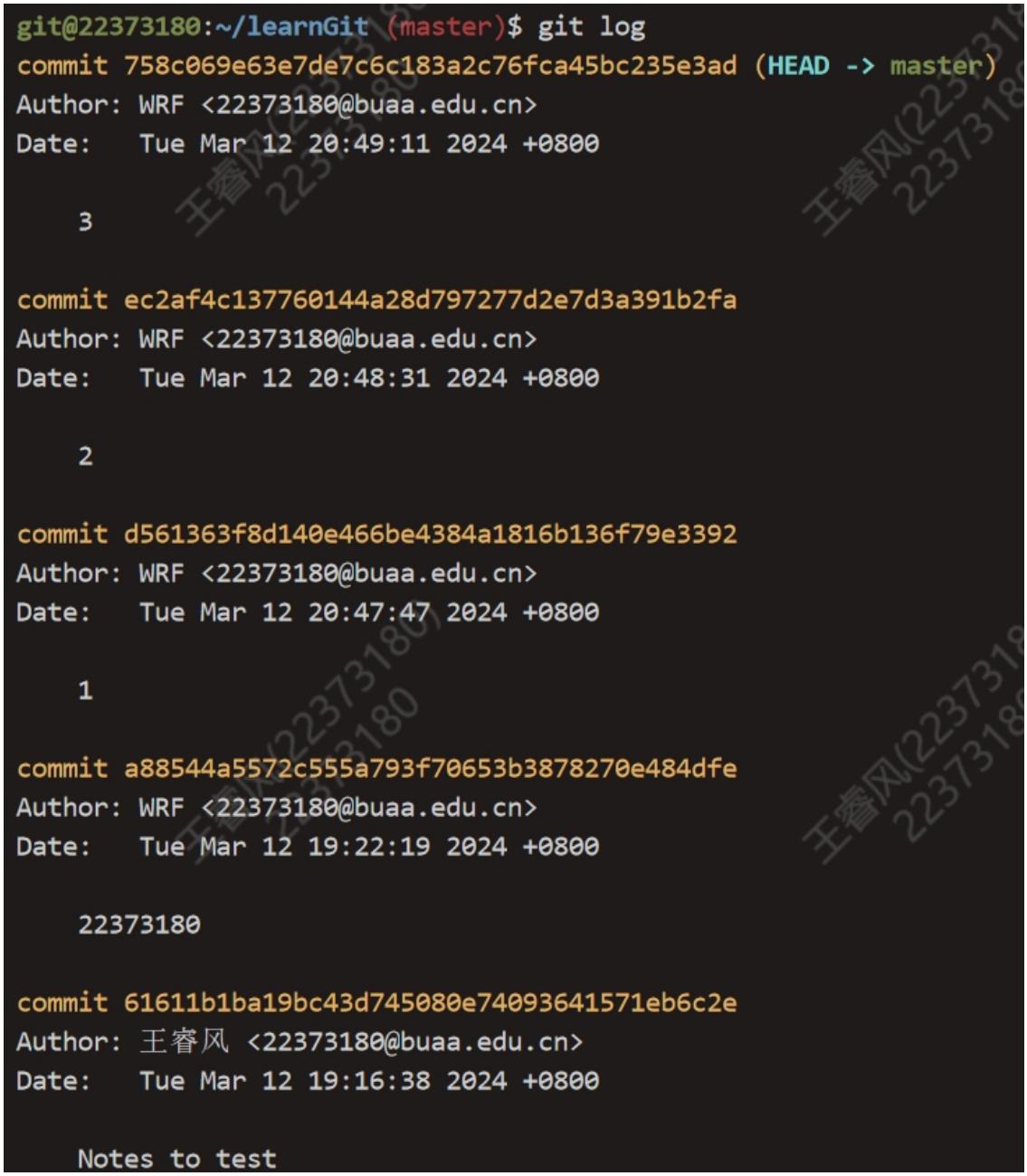
```
git reset HEAD hello.txt
```

Thinking 0.4

- 在文件里加入Testing 1，git add，git commit，提交说明记为1。模仿上述做法，把1分别改为2和3，再提交两次。

```
git@22373180:~/learnGit (master)$ vim README.txt
git@22373180:~/learnGit (master)$ git add .
git@22373180:~/learnGit (master)$ git commit -m "1"
[master d561363] 1
 3 files changed, 24 insertions(+), 1 deletion(-)
 create mode 100644 Modified.txt
 create mode 100644 Stage.txt
git@22373180:~/learnGit (master)$ vim README.txt
git@22373180:~/learnGit (master)$ git add .
git@22373180:~/learnGit (master)$ git commit -m "2"
[master ec2af4c] 2
 1 file changed, 1 insertion(+)
git@22373180:~/learnGit (master)$ vim README.txt
git@22373180:~/learnGit (master)$ git add .
git@22373180:~/learnGit (master)$ git commit -m "3"
[master 758c069] 3
 1 file changed, 1 insertion(+)
```

- 使用git log命令查看提交日志，看是否已经有三次提交，记下提交说明为3的哈希值。



- 进行版本回退。执行命令git reset --hard HEAD^后，再执行git log，观 察其变化。

```
git@22373180:~/learnGit (master)$ git reset --hard HEAD^
HEAD 现在位于 ec2af4c 2
git@22373180:~/learnGit (master)$ git log
commit ec2af4c137760144a28d797277d2e7d3a391b2fa (HEAD -> master)
Author: WRF <22373180@buaa.edu.cn>
Date:   Tue Mar 12 20:48:31 2024 +0800

    2

commit d561363f8d140e466be4384a1816b136f79e3392
Author: WRF <22373180@buaa.edu.cn>
Date:   Tue Mar 12 20:47:47 2024 +0800

    1

commit a88544a5572c555a793f70653b3878270e484dfe
Author: WRF <22373180@buaa.edu.cn>
Date:   Tue Mar 12 19:22:19 2024 +0800

    22373180

commit 61611b1ba19bc43d745080e74093641571eb6c2e
Author: 王睿风 <22373180@buaa.edu.cn>
Date:   Tue Mar 12 19:16:38 2024 +0800

    Notes to test
```

- 找到提交说明为1的哈希值，执行命令git reset --hard后，再执 行git log，观察其变化。

```
git@22373180:~/learnGit (master)$ git reset --hard d561363f8d140e466be4384a1816b1
36f79e3392
HEAD 现在位于 d561363 1
git@22373180:~/learnGit (master)$ git log
commit d561363f8d140e466be4384a1816b136f79e3392 (HEAD -> master)
Author: WRF <22373180@buaa.edu.cn>
Date:   Tue Mar 12 20:47:47 2024 +0800

    1

commit a88544a5572c555a793f70653b3878270e484dfe
Author: WRF <22373180@buaa.edu.cn>
Date:   Tue Mar 12 19:22:19 2024 +0800

    22373180

commit 61611b1ba19bc43d745080e74093641571eb6c2e
Author: 王睿风 <22373180@buaa.edu.cn>
Date:   Tue Mar 12 19:16:38 2024 +0800

    Notes to test
```

- 现在已经回到了旧版本，为了再次回到新版本，执行git reset --hard ，再执行git log，观察其变化。

```
git@22373180:~/learnGit (master)$ git reset --hard 758c069e63e7de7c6c183a2c76fca4
5bc235e3ad
HEAD 现在位于 758c069 3
git@22373180:~/learnGit (master)$ git log
commit 758c069e63e7de7c6c183a2c76fca45bc235e3ad (HEAD -> master)
Author: WRF <22373180@buaa.edu.cn>
Date:   Tue Mar 12 20:49:11 2024 +0800

    3

commit ec2af4c137760144a28d797277d2e7d3a391b2fa
Author: WRF <22373180@buaa.edu.cn>
Date:   Tue Mar 12 20:48:31 2024 +0800

    2

commit d561363f8d140e466be4384a1816b136f79e3392
Author: WRF <22373180@buaa.edu.cn>
Date:   Tue Mar 12 20:47:47 2024 +0800

    1

commit a88544a5572c555a793f70653b3878270e484dfe
Author: WRF <22373180@buaa.edu.cn>
Date:   Tue Mar 12 19:22:19 2024 +0800

    22373180

commit 61611b1ba19bc43d745080e74093641571eb6c2e
Author: 王睿风 <22373180@buaa.edu.cn>
Date:   Tue Mar 12 19:16:38 2024 +0800

    Notes to test
```

Thinking 0.5

```
echo first
```

```
git@22373180:~ $ echo first
first
```

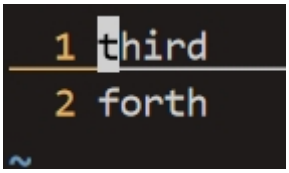
```
echo second > output.txt
```

```
1 second
```

```
echo third > output.txt
```

```
1 third
~
```

```
echo forth >> output.txt
```

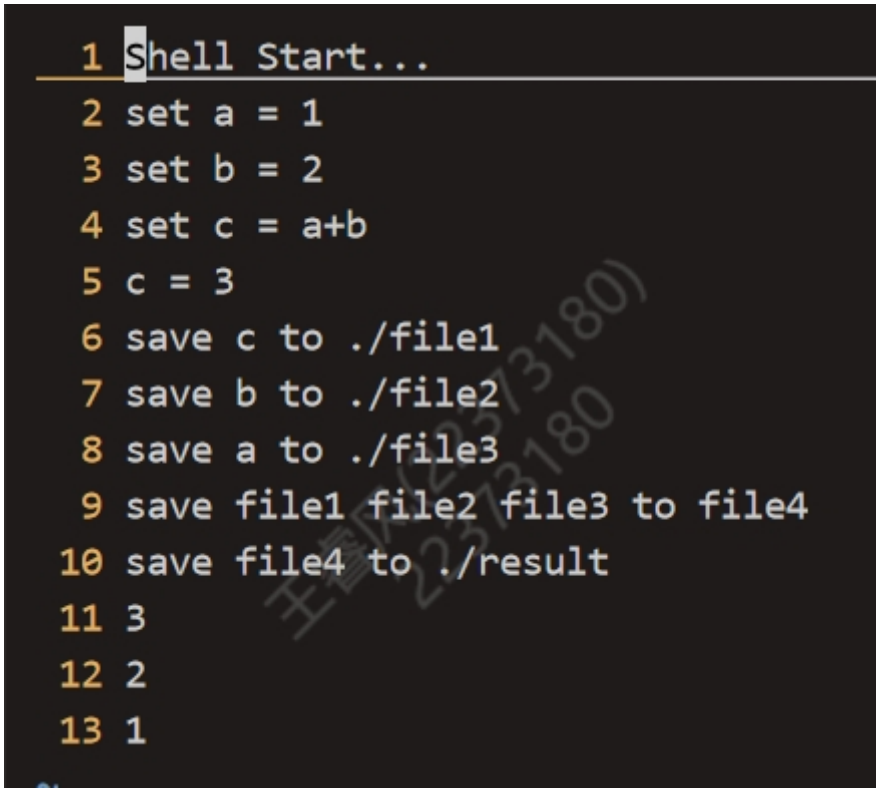


Thinking 0.6

command文件内容：

```
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
```

result文件内容：



test文件命令：

```
echo ...
```

将...中的内容输出到result中

```
a=1
b=2
c=$[$a+$b]
```

将a赋值为1，b赋值为2，c赋值为3

```
echo $c>file1
echo $b>file2
echo $a>file3
```

将a输出到为file3，c输出到为file1，b输出到为file2

```
cat file1>file4
cat file2>>file4
cat file3>>file4
cat file4>>result
```

将file1 file2 file3的值依次输出到file4中，再将file4的值输出到result尾部。

```
echo echo Shell Start
echo 'echo Shell Start'
```

效果无区别

```
echo echo $c>file1
echo 'echo $c>file1'
```

效果有区别，上面的输出结果为echo，下面输出结果为echo $c>file1

二、实验难点分析

bash脚本编写

[sed指令的使用]

> 在src/sh_test 目录下，有一个file 文件和hello_os.sh 文件。hello_os.sh 是 一个未完成的脚本文档，请同学们借助shell编程的知识，将其补完，以实现通过命令bash hello_os.sh AAA BBB，在hello_os.sh 所处的目录新建一个名为 BBB 的文件，其内容为AAA文件的第8、32、128、512、1024行的内容提取(AAA文件行数一定超过1024行)。[注 意：对于命令bashhello_os.sh AAABBB，AAA及BBB可为任何合法文件的名称，例如 bashhello_os.sh filehello_os.c，若已有hello_os.c文件，则将其原有内容覆盖]

```
#!/bin/bash
sed -n '8p, 32p, 128p, 512p, 1024p' $1 > $2
```

使用sed指令将参数$1文件中指定行输出到$2中

> 在Lab0工作区的csc/code目录下，存在fibo.c、main.c，其中fibo.c有点小 问题，还有一个未补全的modify.sh文件，将其补完，以实现通过命令bash modify.sh fibo.c char int，可以将fibo.c中所有的char字符串更改为int字符串。[注意：对于 命令bashmodify.sh fibo.ccharint，fibo.c可为任何合法文件名，char及int可 以是任何字符串，评测时评测modify.sh的正确性，而不是检查修改后fibo.c的正确性]

```
#!/bin/bash
sed -i "s/$2/$3/g" $1
```

[循环语句的使用]

> 在Lab0工作区ray/sh_test1目录中，含有100个子目录file1~file100，还存 在一个名为changefile.sh的文件，将其补完，以实现通过命令bashchangefile.sh，可 以删除该目录内file71~file100共计30个子目录，将file41~file70共计30个子目录 重命名为newfile41~newfile70。[注意：评测时仅检测changefile.sh的正确性]

```
#!/bin/bash
a=1
while [ $a -le 100 ]
do
    if [ $a -gt 70 ]
    then
        rm -r file$a
    elif [ $a -gt 40 ]
    then
        mv file$a newfile$a
    fi
    a=$((a+1))
done
```

[grep和awk指令的使用 重定向和管道的运用]

> 在Lab0工作区的ray/sh_test2目录下，存在一个未补全的search.sh文件， 将其补完，以实现通过命令bash search.sh file int result，可以在当前目录下生成 result文件，内容为file文件含有int字符串所在的行数，即若有多行含有int字符串 需要全部输出。[注意：对于命令bashsearch.sh file int result，file及result可 为任何合法文件名称，int可为任何合法字符串，若已有result文件，则将其原有内容覆 盖，匹配时大小写不忽略]

```
#!/bin/bash
grep -n $2 $1 | awk -F : '{prin $1}' > $3
```

文件操作

> 补全后的palindrome.c、Makefile、hello_os.sh依次复制到路径dst/palindrome.c, dst/Makefile,dst/sh_test/hello_os.sh[注意：文件名和路径必须与题目要求相同]

```
mv palindrome.c ../dst
mv Makefile ../dst
mv hello_os.sh ../dst/sh_test
```

Makefile编写

> Lab0工作区的csc/code/fibo.c成功更换字段后(bashmodify.shfibo.cchar int)，现已有csc/Makefile和csc/code/Makefile，补全两个Makefile文件，要求在 csc目录下通过命令make可在csc/code目录中生成fibo.o、main.o，在csc目录中生 成可执行文件fibo，再输入命令makeclean后只删除两个.o文件。[注意：不能修改 fibo.h和main.c文件中的内容，提交的文件中fibo.c必须是修改后正确的fibo.c，可 执行文件fibo作用是输入一个整数n(从stdin输入n)，可以输出斐波那契数列前n项， 每一项之间用空格分开。比如n=5，输出11235]

```
all: fibo
fibo: code/fibo.o code/main.o
    gcc -o fibo code/fibo.o code/main.o
code/fibo.o: code/fibo.c
    gcc -c code/fibo.c -o code/fibo.o -I include
code/main.o: code/main.c
    gcc -c code/main.c -o code/main.o -I include
clean:
    rm -r code/main.o code/fibo.o
```

三、实验体会

由于上机前没有仔细看教程，导致对Linux一些指令用法的不熟悉、对Makefile和bash脚本写法不熟悉。这也导致了我在上机时写的很慢。在仔细阅读教程后，终于对自己犯的一些错误恍然大悟，对git、Linux命令、bash脚本编写和Makefile编写有了进一步的认识。总体而言，本次实验难度不大，主要是考察我们对基础知识的理解。