Lab0实验报告

> 学号：21371220
>
> 姓名：杨硕

一、思考题

Thinking 0.1

- `Untracked.txt`的内容如下：

```
位于分支 learnGit

尚无提交

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）
```

- `Stage.txt`的内容如下：

```
位于分支 learnGit

尚无提交

要提交的变更:
   （使用 "git rm --cached <文件>..." 以取消暂存）
         新文件:    README.txt

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

- `Modified.txt`的内容如下：

```
位于分支 learnGit
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
        Untracked.txt

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

从上可以看到，`Untracked.txt`的第二、三行为"尚未提交"，"未跟踪的文件"，`Stage.txt`的第二、三行为"尚未提交"，"要提交的变更"，`Modified.txt`的第三行为"尚未暂存以备提交的变更"，这表明，在`README.txt`文件新建时，处于未跟踪的状态；向`README.txt`中添加内容并执行`add`命令后，文件处于暂存的状态；在修改`README.txt`后，文件处于被修改的状态。

Thinking 0.2

- `add the file`对应命令`git add`
- `stage the file`对应命令`git add`
- `commit`对应命令`git commit`

Thinking 0.3

1. 代码文件`print.c`被错误删除时，应当使用`git checkout -- print.c`命令
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`git reset HEAD print.c`和`git checkout -- print.c`命令恢复
3. 无关文件`hello.txt`已经被添加到暂存区时，在不删除此文件的前提下，使用`git rm --cached hello.txt`将其移出暂存区

Thinking 0.4

- 第一次使用`git log`查看时，显示有三次提交

```
commit 3382669eccc4afd2fff1a74e19d7b3d045f5bdf1 (HEAD -> learnGit)
Author: 杨硕 <21371220@buaa.edu.cn>
Date:   Mon Mar 6 08:23:45 2023 +0800

    3

commit 6b6e11be81cb5cd5bce8b77fda6fff61bb83e230
Author: 杨硕 <21371220@buaa.edu.cn>
Date:   Mon Mar 6 08:22:58 2023 +0800

    2
```

```
commit 3385755b25488ddf2ea77fe5262fcc35360921f1
Author: 杨硕 <21371220@buaa.edu.cn>
Date:   Mon Mar 6 08:19:29 2023 +0800

    1
```

- 执行命令`git reset --hard HEAD^`后，再执行`git log`查看，只显示前两次提交

```
commit 6b6e11be81cb5cd5bce8b77fda6fff61bb83e230 (HEAD -> learnGit)
Author: 杨硕 <21371220@buaa.edu.cn>
Date:   Mon Mar 6 08:22:58 2023 +0800

    2

commit 3385755b25488ddf2ea77fe5262fcc35360921f1
Author: 杨硕 <21371220@buaa.edu.cn>
Date:   Mon Mar 6 08:19:29 2023 +0800

    1
```

- 执行命令`git reset --hard 3385755`后，再执行`git log`查看，只显示第一次提交

```
commit 3385755b25488ddf2ea77fe5262fcc35360921f1 (HEAD -> learnGit)
Author: 杨硕 <21371220@buaa.edu.cn>
Date:   Mon Mar 6 08:19:29 2023 +0800

    1
```

- 执行`git reset --hard 3382669`，再执行`git log`，回到最初的版本

```
commit 3382669eccc4afd2fff1a74e19d7b3d045f5bdf1 (HEAD -> learnGit)
Author: 杨硕 <21371220@buaa.edu.cn>
Date:   Mon Mar 6 08:23:45 2023 +0800

    3

commit 6b6e11be81cb5cd5bce8b77fda6fff61bb83e230
Author: 杨硕 <21371220@buaa.edu.cn>
Date:   Mon Mar 6 08:22:58 2023 +0800

    2

commit 3385755b25488ddf2ea77fe5262fcc35360921f1
Author: 杨硕 <21371220@buaa.edu.cn>
Date:   Mon Mar 6 08:19:29 2023 +0800

    1
```

Thinking 0.5

- `echo first`，屏幕输出`first`
- `echo second > output.txt`，向`output.txt`文件中输出了`second`
- `echo third > output.txt`，向`output.txt`文件中输出了`third`，并覆盖了之前的`second`
- `echo forth >> output.txt`，向`output.txt`文件中追加输出了`forth`

Thinking 0.6

`command`文件内容如下：

```
 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`文件内容如下：

```
3
2
1
```

根据`test`文件内容，首先赋值变量`a`和`b`分别为1和2，赋值变量`c`为`a`和`b`的和3，将`c`值输出到`file1`文件，`b`值输出到`file2`，`a`值输出到`file3`，然后`file1``file2``file3`依次追加输出到`file4`文件

`echo echo Shell Start`与`echo 'echo Shell Start'`效果没有区别，都是输出字符串`echo Shell Start`

`echo echo $c>file1`与`echo 'echo $c>file1'`效果有区别，前者是把变量c的值输出到`file1`，后者是输出字符串`echo $c>file1`

二、实验难点

Exercise 0.4.2

题目要求编写两个`Makefile`，且分别处于上下级目录，且在上级目录`csc`调用`make`命令，可同时在下级目录`code`生成2个`.o`文件和在当前目录生成一个可执行文件。

首先想到，可以在上级目录的`Makefile`写入命令对下级目录的`Makefile`进行操作，于是写出如下命令：

```
fibo:
    cd code
    make
```

运行测试后发现，执行`make`命令，只会不断重复进入`code`目录，经过查阅资料发现原因，`cd`命令只对当前行有效，即下一条`make`命令仍在当前`csc`目录执行。找到原因后，将命令改为：

```
fibo:
    cd code && make
```

可顺利进入下级目录`code`执行`make`

接下来编写下级目录的`makefile`，如下：

```
all: fibo.c main.c
    gcc -c fibo.c
    gcc -c main.c
```

结果报错，找不到`fibo.h`文件，查看文件结构后发现，`fibo.h`并不在`code`目录中，于是想到利用`gcc`的`-I`选项解决：

```
all: fibo.c main.c
    gcc -c fibo.c -I ../include
    gcc -c main.c -I ../include
```

根据题目要求，完善上级目录的`Makefile`

```
fibo: ./code/fibo.c ./code/main.c
    cd code && make
    gcc -o fibo code/fibo.o code/main.o
clean:
    rm -f code/fibo.o code/main.o
```

三、实验体会

第一次接触`Linux`操作系统和`Git`，对各种操作和命令都不熟悉，只能照葫芦画瓢，模仿预习教程和指导书上的操作进行尝试，经历了各种异常和报错，感觉非常棘手。

多次阅读预习教程和指导书后，陌生感减少了很多，可以自主实现一些基本操作。第二周在上机时间顺利完成了`lab0`课下实验，大大增强了信心。

当然，目前的我只会一些基础中的基础的命令与操作，在以后的课程中还需不断学习，不断记忆，掌握更多的有力工具。