思考题

Thinking 0.1 思考下列有关 Git 的问题：

- 在前述已初始化的 `~/learnGit` 目录下，创建一个名为 README.txt 的文件。执行命令 `git status > Untracked.txt`（其中的 `>` 为输出重定向，我们将在 0.6.3 中详细介绍）。
- 在 README.txt 文件中添加任意文件内容，然后使用 `add` 命令，再执行命令 `git status > Stage.txt`。
- 提交 README.txt，并在提交说明里写入自己的学号。
- 执行命令 `cat Untracked.txt` 和 `cat Stage.txt`，对比两次运行的结果，体会 README.txt 两次所处位置的不同。
- 修改 README.txt 文件，再执行命令 `git status > Modified.txt`。
- 执行命令 `cat Modified.txt`，观察其结果和第一次执行 `add` 命令之前的 status 是否一样，并思考原因。

■

```
git@23373175:~/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）
git@23373175:~/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

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

git@23373175:~/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
        Untracked.txt

修改尚未加入提交（使用 "git add" 和/或 "git commit -a"）
git@23373175:~/learnGit (master)$
```

第一次执行add之前的状态是未跟踪，因为这个时候仓库里没有任何关于README.txt的信息。

而执行了一次add之后再修改，状态是已修改，这时候仓库里已经有README.txt的信息了，只不过我们又修改了一次文件，所以状态显示已修改。

> **Thinking 0.2** 仔细看看0.10，思考一下箭头中的 add the file 、stage the file 和 commit 分别对应的是 Git 里的哪些命令呢？ ■

add the file 对应 git add

stage the file 也对应 git add

commit 对应 git commit

> **Thinking 0.3** 思考下列问题：
>
> 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？
> 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？
> 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？
>
> ■

1.

```
git checkout print.c
```

2.

```
git reset HEAD print.c//取消删除操作
git checkout print.c//从暂存区恢复文件
```

3.

```
git rm print.c
```

Thinking 0.4 思考下列有关 Git 的问题：

- 找到在 `/home/22xxxxxx/learnGit` 下刚刚创建的 `README.txt` 文件，若不存在则新建该文件。
- 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。
- 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[a]。
- 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。
- 找到提交说明为 1 的哈希值，执行命令 `git reset --hard <hash>` 后，再执行 `git log`，观察其变化。
- 现在已经回到了旧版本，为了再次回到新版本，执行 `git reset --hard <hash>`，再执行 `git log`，观察其变化。

■

[a]使用 `git log` 命令时，在 `commit` 标识符后的一长串数字和字母组成的字符串

- 提交操作

```
git@23373175:~/learnGit (master)$ vim README.txt
git@23373175:~/learnGit (master)$ git add
没有指定文件，也没有文件被添加。
提示：也许您想要执行 'git add .'?
提示：运行下面的命令来关闭本消息
提示："git config advice.addEmptyPathspec false"
git@23373175:~/learnGit (master)$ git add .
git@23373175:~/learnGit (master)$ git commit -m "1"
[master 2f4ce5a] 1
 4 files changed, 31 insertions(+), 1 deletion(-)
 create mode 100644 Modified.txt
 create mode 100644 Stage.txt
 create mode 100644 Untracked.txt
git@23373175:~/learnGit (master)$ vim README.txt
git@23373175:~/learnGit (master)$ git add .
git@23373175:~/learnGit (master)$ git commit -m "2"
[master af7127b] 2
 1 file changed, 1 insertion(+), 1 deletion(-)
git@23373175:~/learnGit (master)$ vim README.txt
git@23373175:~/learnGit (master)$ git add .
git@23373175:~/learnGit (master)$ git commit -m "3"
[master ebece83] 3
 1 file changed, 1 insertion(+), 1 deletion(-)
```

- 版本回退

```
git@23373175:~/learnGit (master)$ git log
commit ebece838c05fd19349934290443add332a686864 (HEAD -> master)
Author: 占永杰 <23373175@buaa.edu.cn>
Date:   Tue Mar 11 09:49:40 2025 +0800

    3

commit af7127b83db9ddc8978134a372278a8d2acbcfe2
Author: 占永杰 <23373175@buaa.edu.cn>
Date:   Tue Mar 11 09:49:19 2025 +0800

    2

commit 2f4ce5a3f242c1fec935a89eff27c72a16580feb
Author: 占永杰 <23373175@buaa.edu.cn>
Date:   Tue Mar 11 09:48:54 2025 +0800

    1
```

```
git@23373175:~/learnGit (master)$ git reset --hard HEAD^
HEAD 现在位于 af7127b 2
git@23373175:~/learnGit (master)$ git log
commit af7127b83db9ddc8978134a372278a8d2acbcfe2 (HEAD -> master)
Author: 占永杰 <23373175@buaa.edu.cn>
Date:   Tue Mar 11 09:49:19 2025 +0800

    2

commit 2f4ce5a3f242c1fec935a89eff27c72a16580feb
Author: 占永杰 <23373175@buaa.edu.cn>
Date:   Tue Mar 11 09:48:54 2025 +0800

    1
```

```
git@23373175:~/learnGit (master)$ git reset --hard 2f4ce5a3
HEAD 现在位于 2f4ce5a 1
git@23373175:~/learnGit (master)$ git log
commit 2f4ce5a3f242c1fec935a89eff27c72a16580feb (HEAD -> master)
Author: 占永杰 <23373175@buaa.edu.cn>
Date:   Tue Mar 11 09:48:54 2025 +0800

    1
```

```
git@23373175:~/learnGit (master)$ git reset --hard ebece838
HEAD 现在位于 ebece83 3
git@23373175:~/learnGit (master)$ git log
commit ebece838c05fd19349934290443add332a686864 (HEAD -> master)
Author: 占永杰 <23373175@buaa.edu.cn>
Date:   Tue Mar 11 09:49:40 2025 +0800

    3

commit af7127b83db9ddc8978134a372278a8d2acbcfe2
Author: 占永杰 <23373175@buaa.edu.cn>
Date:   Tue Mar 11 09:49:19 2025 +0800

    2

commit 2f4ce5a3f242c1fec935a89eff27c72a16580feb
Author: 占永杰 <23373175@buaa.edu.cn>
Date:   Tue Mar 11 09:48:54 2025 +0800

    1
```

Thinking 0.5 执行如下命令, 并查看结果

- `echo first`
- `echo second > output.txt`
- `echo third > output.txt`
- `echo forth >> output.txt`

```
git@23373175:~ $ echo first
first
git@23373175:~ $ echo second > output.txt
git@23373175:~ $ cat output.txt
second
git@23373175:~ $ echo third > output.txt
git@23373175:~ $ cat
23373175/                   learnGit/
.ansible/                   .lesshst
.bash_history               output.txt
.cache/                     .ssh/
ctags_test.c                .sudo_as_admin_successful
.gitconfig                  tags
hello                       TEST
hello.c                     tutor
hello.ks                    .vim/
hello.o                     .viminfo
hello.os                    .vimrc
helloworld.c
git@23373175:~ $ cat output.txt
third
git@23373175:~ $ echo forth >> output.txt
git@23373175:~ $ cat output.txt
third
forth
```

>会覆盖原来内容

>>会加入到原来内容的后面

Thinking 0.6 使用你知道的方法（包括重定向）创建下图内容的文件（文件命名为 test），将创建该文件的命令序列保存在 command 文件中，并将 test 文件作为批处理文件运行，将运行结果输出至 result 文件中。给出 command 文件和 result 文件的内容，并对最后的结果进行解释说明（可以从 test 文件的内容入手）。具体实现的过程中思考下列问题: echo echo Shell Start 与 echo `echo Shell Start` 效果是否有区别; echo echo $c>file1 与 echo `echo $c>file1` 效果是否有区别. ■

```
git@23373175:~ $ vim command
git@23373175:~ $ chmod +x command
git@23373175:~ $ ./command
git@23373175:~ $ chmod +x test
git@23373175:~ $ ./test > result
git@23373175:~ $ cat result
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
3
2
1
```


难点分析

make的在使用的时候需要理清楚文件之间的依赖关系，已经make里面嵌套make 的时候，需要理清楚哪些是外部make做的事情，哪些是内部make做的事情。使用命令的时候需要清楚执行文件在哪个目录下，应使用正确的相对路径。

实验体会

刚开始接触命令行的时候有些不习惯，但是习惯了之后发现效率非常高，通过简单的命令就可以完成大量操作。熟练掌握git,vim,grep，管道，shell脚本，make等工具之后对PC的使用效率会大大提高