# 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。

Untraced.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

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

此时由于README.txt没有add，因此为未跟踪的文件。

> 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。

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
```

此时add了README.txt到暂存区，被追踪但没有提交。

> 提交 README.txt，并在提交说明里写入自己的学号。

```
git@21371477:~/learnGit (master)$ git commit -m "21371477"
[master 8ffd4d1] 21371477
 2 files changed, 8 insertions(+)
 create mode 100644 README.txt
 create mode 100644 Untracked.txt
```

> 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。

README.txt从工作区到暂存区，再到本地仓库。

> 修改 README.txt 文件，再执行命令 git status > Modified.txt。

Modified.txt：

```
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
```

此时由于README.txt已被跟踪，但发生了修改且未被提交到暂存区。

> 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。

此时由于README.txt已被跟踪，但发生了修改且未被提交到暂存区。

## 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 file
- stage the file 对应 git add file
- 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?

- `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?

1. `git reset HEAD print.c`
2. `git checkout -- printf.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 rm --cached hello.txt`

## Thinking 0.4

思考下列有关Git的问题：

> 找到在/home/21xxxxxx/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(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)。

> 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 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 ，再执行 git log，观察其变化。

`git log`：起始和第二次`git reset --hard <hash>`后

```
git@21371477:~/learnGit (master)$ git log
commit 9c05ba7b7d34db85d938b1734aa2e16a94c44ef0 (HEAD -> master)
Author: 吴建宇 <21371477@buaa.edu.cn>
Date:   Mon May 8 01:08:09 2023 +0800

    3

commit 6a21e6b16b1e169e3394f2e5234b553208b8720c
Author: 吴建宇 <21371477@buaa.edu.cn>
Date:   Mon May 8 01:07:53 2023 +0800

    2

commit a2e7b0ce3485f1eb3788dbe165a18d6ae3e7e0e3
Author: 吴建宇 <21371477@buaa.edu.cn>
Date:   Mon May 8 01:07:15 2023 +0800

    1

commit 8ffd4d13b341e196abc312fa03841c13690230a9
Author: 吴建宇 <21371477@buaa.edu.cn>
Date:   Mon May 8 00:59:33 2023 +0800

    21371477
```

`git log`：`git reset --hard HEAD^`后

```
commit 6a21e6b16b1e169e3394f2e5234b553208b8720c (HEAD -> master)
Author: 吴建宇 <21371477@buaa.edu.cn>
Date:   Mon May 8 01:07:53 2023 +0800

    2

commit a2e7b0ce3485f1eb3788dbe165a18d6ae3e7e0e3
Author: 吴建宇 <21371477@buaa.edu.cn>
Date:   Mon May 8 01:07:15 2023 +0800

    1

commit 8ffd4d13b341e196abc312fa03841c13690230a9
Author: 吴建宇 <21371477@buaa.edu.cn>
Date:   Mon May 8 00:59:33 2023 +0800

    21371477
```

`git log`：`git reset --hard <hash>`后

```
git@21371477:~/learnGit (master)$ git log
commit a2e7b0ce3485f1eb3788dbe165a18d6ae3e7e0e3 (HEAD -> master)
Author: 吴建宇 <21371477@buaa.edu.cn>
Date:   Mon May 8 01:07:15 2023 +0800

    1

commit 8ffd4d13b341e196abc312fa03841c13690230a9
Author: 吴建宇 <21371477@buaa.edu.cn>
Date:   Mon May 8 00:59:33 2023 +0800

    21371477
```

`git reset --hard HEAD^`后回到上一个版本，即版本2，通过`git reset --hard <hash>`可以回到`<hash>`对应的版本。

## Thinking 0.5

执行如下命令，并查看结果：

> echo first

```
git@21371477:~/learnGit (master)$ echo first
first
```

> echo second > output.txt

```
git@21371477:~/learnGit (master)$ echo second > output.txt
git@21371477:~/learnGit (master)$ cat output.txt
second
```

> echo third > output.txt

```
git@21371477:~/learnGit (master)$ echo third > output.txt
git@21371477:~/learnGit (master)$ cat output.txt
third
```

> echo forth >> output.txt

```
git@21371477:~/learnGit (master)$ echo forth >> output.txt
git@21371477:~/learnGit (master)$ cat output.txt
third
forth
```

## 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.

command:

```
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
```

test:

```
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
```

`echo`对于不包含转义字符和取变量的内容，是否加单引号结果一样。但如果存在转义字符或取变量时，需要加单引号或反斜杠才能原样输出字符串，实现不进行转义或取变量。

# 难点分析

## Makefile

1. 缩进或空格错误：目标和依赖间需要空格；命令前需要Tab缩进。
2. 变量使用：`$(varname)`引用变量。
3. 构建顺序：`make`默认构建第一个目标。
4. 预定义变量：`CC`为C语言编译器。

## Git

1. 分支管理：切换分支时需要切换到git根目录，并通过`git status`查看当前状态。
2. 恢复操作：
    - `git checkout`：恢复被修改的文件到上一个提交状态，或者切换到其他分支上。

      `git checkout <commit/branch> <file>`用于还原指定文件到指定的提交或分支。
    - `git reset`：回退提交或取消暂存的文件。有三个主要的模式：`--soft`、`--mixed`和`--hard`。

      `--soft`模式将HEAD指针移动到指定的提交，但保留工作目录和暂存区的修改；

      `--mixed`模式（默认模式）将HEAD指针和暂存区都移动到指定的提交，但保留工作目录的修改；

      `--hard`模式将HEAD指针、暂存区和工作目录都移动到指定的提交，丢弃所有修改。
    - `git revert`：创建一个新的提交，撤销之前的提交。

      `git revert <commit>`会撤销指定的提交，生成一个新的提交来还原指定提交的更改。这种方式是通过提交新的变更来撤销之前的提交，因此历史记录中会包含撤销的提交。

## Shell

1. 脚本权限问题：使用`chmod +x script.sh`命令来为脚本添加执行权限。
2. 注释问题：Shell脚本使用`#`作为注释符号。在注释行的开头使用`#`。
3. 变量赋值问题：在变量赋值时`var="value"`，要确保没有在等号两边添加空格。
4. 文件路径问题：在处理文件路径时，确保使用正确的路径分隔符。在Unix和Linux系统上，路径分隔符是斜杠（/），而在Windows系统上是反斜杠（\）。
5. 条件语句问题：在使用条件语句（if-else）时，要使用方括号（[ ]）并在它们周围留出空格。条件表达式应该在方括号内，并且方括号与其他部分之间应该有空格。
6. 命令执行问题：在Shell脚本中执行外部命令时，确保命令的路径正确。可以使用绝对路径或设置正确的环境变量来执行命令。

# 实验体会

- 对CLI环境操作以及相关工具的使用还不熟悉，体会到sed、awk等工具的强大。
- 通过ssh和tmux实现的远程编程可以使得切换机器不中断任务。
- 希望可以增加样例讲解，以及希望增加对这些工具的练习，感觉第一遍学完还是有点蒙。