Lab0 实验报告

> 姓名：文柳懿
> 学号：21351002

1.思考题

Thinking 0.1

- 对比 `Untracked.txt` 和 `Stage.txt`：
  `Untracked.txt` 显示 `README.txt` 文件未被跟踪；而 `Stage.txt` 则显示 `README.txt` 文件已被添加到暂存区.

- 查看 `Modified.txt`：
  `Modified.txt` 显示 `README.txt` 已被修改但未暂存。与第一次执行 `add` 命令之前相比： `add` 之前， `README.txt` 被视作未跟踪的文件；执行后， `README.txt` 已被跟踪，修改后有尚未暂存的变更。

Thinking 0.2

- `Add the file` 对应： `git add` 指令.
  文件从工作目录添加至暂存区，Git开始跟踪该文件的变更。
- `Stage the file` 对应： `git add` 指令.
  将文件的当前状态（自上次提交以来的所有变更）添加到暂存区。
- `Commit` 对应： `git commit` 指令.
  将暂存区中的文件变更，保存为一个新的提交，加入Git仓库。

Thinking 0.3

- `print.c` 被错误删除时：
  （1）若未提交：执行 `git restore --source=HEAD print.c`，从最近的提交（HEAD）中恢复 `print.c` 文件。
  （2）若已提交：找到 `print.c` 存在的最后一次提交，例如知道 `print.c` 在 `commit-1` 中存在，执行 `git checkout commit-1 -- print.c`.

- `print.c` 被错误删除后，执行了 `git rm print.c` 命令：
  找到 `print.c` 存在的最后一次提交，例如知道 `print.c` 在 `commit-1` 中存在，检出该文件：
  执行 `git checkout commit-1 -- print.c`.
  这将 `print.c` 恢复至工作区，再执行 `git add print.c` 添加至暂存区.

- 在不删除文件的前提下，将 `hello.txt` 移出暂存区：
  `git restore --staged hello.txt` 或 `git reset HEAD hello.txt` ,会将 `hello.txt` 加入工作区.

Thinking 0.4

- 执行 `git reset --hard HEAD^` 后：版本回退
  执行 `git log`，发现最后一个提交(说明为 `3` 的提交)已消失，指针回退了一个提交。
- 执行 `git reset --hard <hash>` 后（hash值对应提交说明为"1"）：回退到特定版本
  执行 `git log`，回到提交说明为"1"的版本。
- 执行 `git reset --hard <hash>` 后（hash值对应提交说明为"3"）：回到新版本
  执行 `git log`，回到提交说明为"3"的版本。

Thinking 0.5

- `echo first`:
  终端输出 `first`.
- `echo second > output.txt`:
  将 `second` 写入 `output.txt` 文件。
  若文件之前不存在，则创建；若已存在，则清空并重新写入 `second`.
- `echo third > output.txt`:
  文件 `output.txt` 清空，并重新写入 `third`.
- `echo forth >> output.txt`:
  将 `forth` 追加至 `output.txt` 末尾.
- 执行 `cat output.txt`:
  终端输出:

```
third
forth
```

Thinking 0.6

`command`文件内容:

```
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
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
```

将`test`文件作为批处理文件运行后，执行`./test > result`将结果输出至`result`文件中.
`result`文件内容:

```
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
```

- `echo echo Shell Start`与`echo` `echo Shell Start`的区别:
  前者会输出文字 "echo Shell Start";
  后者会执行`echo Shell Start`命令，输出命令的结果"Shell Start".

- `echo echo $c>file1` 与 `echo``echo $c>file1` 的区别:
  前者:若 `$c` 未被定义，则输出 `echo` 至 `file1`.
  若 `$c` 被定义某个值，假设为 `hi`，则输出 `echo hi` 至 `file1`.
  后者:反引号表示命令替换。先执行 `echo $c>file1` 这个子命令，将其标准输出替换到当前位置。而 `echo $c>file1` 命令本身将 `$c` 的值（若定义）输出至 `file1`，即重定向至文件，因此标准输出为空。
  因此，执行 `echo``echo $c>file1`，输出一个空行至标准输出，`file1` 被 `$c` 的值（若定义）替换。

2.难点分析

Exercise 0.2

- 循环/条件结构

```
while [ $a -le 100 ]
do
    if [ $a -gt 70 ]
    then

    elif [ $a -gt 40 ]
    then

    fi
    a=$[$a+1]       #change variable
done
```

Exercise 0.3

命令 `bash search.sh file int result`，获取 `file` 中含有 `int` 字符串所在行数，输出至 `result`.

- 重定向
- `grep -n` 输出包含字符串的行号
- `awk` 使用分隔符 `:`，将分割后第一个参数输出至 `result`
  `cat $1 | grep -n $2 | awk -F : '{print $1}' >> $3`

Exercise 0.4

将 `fibo.c` 中所有的 `char` 字符串更改为 `int` 字符串。

```
source_file="$1"
old_string="$2"
new_string="$3"
sed -i "s/$old_string/$new_string/g" "$source_file"
```

- 注意：不能直接使用`sed –i "s/$2/$3/g" $s1`，该命令意为：将`$2`替换为`$3`.

3.体会

多查看指导书，并对比不同命令的细微区别。