

# 实验报告——Shell 工具和脚本，vim 编辑器，数据整理

姓名：徐亚齐 学号：23020007136

2024 年 8 月 30 日

## 1 实验实例

### 1. 在 /tmp 下新建一个名为 missing 的文件夹

新建文件夹需要用到 mkdir，建好后可以用 ls 来查看包含的文件名称，从而确定文件夹已经建好了。

```
Lenovo@LAPTOP-FGVNVP6K MINGW64 /d/系统开发工具基础 (master)
$ cd tmp

Lenovo@LAPTOP-FGVNVP6K MINGW64 /d/系统开发工具基础/tmp (master)
$ mkdir missing

Lenovo@LAPTOP-FGVNVP6K MINGW64 /d/系统开发工具基础/tmp (master)
$ ls
missing/
```

图 1: 实例 1

### 2. 用 touch 在 missing 文件夹中新建一个叫 semester 的文件

首先用 cd 进入刚建好的 missing 文件夹，然后再运行 touch semester 就可以了。

```

Lenovo@LAPTOP-FGVNVP6K MINGW64 /d/系统开发工具基础/tmp (master)
$ cd missing

Lenovo@LAPTOP-FGVNVP6K MINGW64 /d/系统开发工具基础/tmp/missing (master)
$ touch semester

Lenovo@LAPTOP-FGVNVP6K MINGW64 /d/系统开发工具基础/tmp/missing (master)
$ ls
semester

```

图 2: 实例 2

3. 将以下内容一行一行地写入 semester 文件:

`#!/bin/sh`

`curl --head --silent https://missing.csail.mit.edu`

在尝试将第一行写入文件时，因为有特殊元素的符号，所以需要用到单引号。将第一行用单引号包含起来就可以让有特殊含义的符号正常输入进文件。

```

Lenovo@LAPTOP-FGVNVP6K MINGW64 /d/系统开发工具基础/tmp (master)
$ echo '#!/bin/sh' > semester

Lenovo@LAPTOP-FGVNVP6K MINGW64 /d/系统开发工具基础/tmp (master)
$ echo curl --head --silent https://missing.csail.mit.edu >>semester

Lenovo@LAPTOP-FGVNVP6K MINGW64 /d/系统开发工具基础/tmp (master)
$ cat semester
#!/bin/sh
curl --head --silent https://missing.csail.mit.edu

```

图 3: 实例 3

4. 尝试成功执行这个文件，并用 `ls` 命令来获取它的相关信息

执行文件只需要将 `./semester` 输入到 shell 中并回车，用 `ls -l` 可以获得完整的信息

```

Lenovo@LAPTOP-FGVNVP6K MINGW64 /d/系统开发工具基础/tmp (master)
$ ./semester
HTTP/1.1 200 OK
Connection: keep-alive
Content-Length: 8285
Server: GitHub.com
Content-Type: text/html; charset=utf-8
Last-Modified: Thu, 08 Aug 2024 20:16:01 GMT
Access-Control-Allow-Origin: *
ETag: "66b52781-205d"
expires: Tue, 03 Sep 2024 04:29:41 GMT
Cache-Control: max-age=600
x-proxy-cache: MISS
X-GitHub-Request-Id: 3DA6:3257C3:11AA297:1209311:66D68E5D
Accept-Ranges: bytes
Age: 0
Date: Wed, 04 Sep 2024 03:06:48 GMT
Via: 1.1 varnish
X-Served-By: cache-nrt-rjtf7700068-NRT
X-Cache: HIT
X-Cache-Hits: 0
X-Timer: S1725419208.871085,VS0,VE213
Vary: Accept-Encoding
X-Fastly-Request-ID: f4371256c4d7c2ba0ae33286d8c23262bfdd235d

Lenovo@LAPTOP-FGVNVP6K MINGW64 /d/系统开发工具基础/tmp (master)
$ ls -l
total 1
drwxr-xr-x 1 Lenovo 197121 0 Sep  4 11:00 missing/
-rwxr-xr-x 1 Lenovo 197121 61 Sep  4 11:05 semester*

```

图 4: 实例 4

5. 使用 `ls` 命令进行如下操作:

所有文件 (包括隐藏文件)

文件打印以人类可以理解的格式输出 (例如, 使用 454M 而不是 454279954)

文件以最近访问顺序排序

以彩色文本显示输出结果

获取所有的文件需要用: `-a`。以人类可以理解的格式输出需要用: `-h`。

以最近访问顺序排序需要用: `-t`。以彩色文本输出需要用: `-color=auto`。

```
Lenovo@LAPTOP-FGVNVP6K MINGW64 /d/系统开发工具基础/tmp (master)
$ ls -a
./ ../ missing/ semester*

Lenovo@LAPTOP-FGVNVP6K MINGW64 /d/系统开发工具基础/tmp (master)
$ ls -h
missing/ semester*

Lenovo@LAPTOP-FGVNVP6K MINGW64 /d/系统开发工具基础/tmp (master)
$ ls -t
semester* missing/

Lenovo@LAPTOP-FGVNVP6K MINGW64 /d/系统开发工具基础/tmp (master)
$ ls --color=auto
missing/ semester*

Lenovo@LAPTOP-FGVNVP6K MINGW64 /d/系统开发工具基础/tmp (master)
$ ls -l
total 1
drwxr-xr-x 1 Lenovo 197121 0 Sep  4 11:00 missing/
-rwxr-xr-x 1 Lenovo 197121 61 Sep  4 11:05 semester*
```

图 5: 实例 5

6. 编写两个 bash 函数 marco 和 polo 执行下面的操作。每当你执行 marco 时，当前的工作目录应当以某种形式保存，当执行 polo 时，无论现在处在什么目录下，都应当 cd 回到当时执行 marco 的目录。为了方便 debug，你可以把代码写在单独的文件 marco.sh 中，并通过 source marco.sh 命令，（重新）加载函数。通过 source 来加载函数，随后可以在 bash 中直接使用。

通过 vim marco.sh，用 vim 编辑好文件 marco.sh 的内容，然后执行 source marco.sh 命令，就可以使用 marco 和 polo 这两个新写好的函数了。

```

Lenovo@LAPTOP-FGVNVP6K MINGW64 /d/系统开发工具基础/tmp (master)
$ vim marco.sh

Lenovo@LAPTOP-FGVNVP6K MINGW64 /d/系统开发工具基础/tmp (master)
$ ls
marco.sh* missing/ semester*

Lenovo@LAPTOP-FGVNVP6K MINGW64 /d/系统开发工具基础/tmp (master)
$ source marco.sh

Lenovo@LAPTOP-FGVNVP6K MINGW64 /d/系统开发工具基础/tmp (master)
$ marco
save pwd /d/系统开发工具基础/tmp

Lenovo@LAPTOP-FGVNVP6K MINGW64 /d/系统开发工具基础/tmp (master)
$ cd

Lenovo@LAPTOP-FGVNVP6K MINGW64 ~ (main)
$ polo

Lenovo@LAPTOP-FGVNVP6K MINGW64 /d/系统开发工具基础/tmp (master)

```

图 6: 实例 6(1)

```

Lenovo@LAPTOP-FGVNVP6K MINGW64 /d/系统开发工具基础/tmp (master)
$ cat marco.sh
#!/bin/bash
marco(){
    echo "$(pwd)" > $HOME/marco_history.log
    echo "save pwd $(pwd)"
}
polo(){
    cd "$(cat "$HOME/marco_history.log")"
}

```

图 7: 实例 6(2)

## 7. 对所有文件进行操作, 创建一个 zip 压缩文件

首先用 touch 和 mkdir 创建一些需要被压缩的文件, 再用 find 命令将这些新创建的文件进行压缩。

```

Lenovo@LAPTOP-FGVNVP6K MINGW64 /d/系统开发工具基础/tmp (master)
$ mkdir html_root

Lenovo@LAPTOP-FGVNVP6K MINGW64 /d/系统开发工具基础/tmp (master)
$ cd html_root

Lenovo@LAPTOP-FGVNVP6K MINGW64 /d/系统开发工具基础/tmp/html_root (master)
$ touch {1..10}.html

Lenovo@LAPTOP-FGVNVP6K MINGW64 /d/系统开发工具基础/tmp/html_root (master)
$ mkdir html

Lenovo@LAPTOP-FGVNVP6K MINGW64 /d/系统开发工具基础/tmp/html_root (master)
$ cd html

Lenovo@LAPTOP-FGVNVP6K MINGW64 /d/系统开发工具基础/tmp/html_root/html (master)
$ touch xxxx.html

```

图 8: 实例 7(1)

```

Lenovo@LAPTOP-FGVNVP6K MINGW64 /d/系统开发工具基础/tmp/html_root/html (master)
$ cd /d/系统开发工具基础/tmp

Lenovo@LAPTOP-FGVNVP6K MINGW64 /d/系统开发工具基础/tmp (master)
$ find html_root -name "*.html" -print0 | xargs -0 tar vcf html.zip
html_root/1.html
html_root/10.html
html_root/2.html
html_root/3.html
html_root/4.html
html_root/5.html
html_root/6.html
html_root/7.html
html_root/8.html
html_root/9.html
html_root/html/xxxx.html

```

图 9: 实例 7(2)

8. 编写一个命令或脚本递归的查找文件夹中最近使用的文件, 按照最近的使用时间列出来

为防止文件数量较多出现错误, 可以增加 `-mmin` 条件, 先将最近修改的文件进行初步筛选再交给 `ls` 进行排序显示, 只用输入 `find . -type f -mmin -60 -print0 | xargs -0 ls -lt | head -10` 命令就可以了。

```

Lenovo@LAPTOP-FGVNVP6K MINGW64 /d/系统开发工具基础 (master)
$ find . -type f -mmin -60 -print0 | xargs -0 ls -lt | head -10
-rw-r--r-- 1 Lenovo 197121 1 Sep 4 16:25 ./tmp/out.log
-rw-r--r-- 1 Lenovo 197121 10240 Sep 4 16:17 ./tmp/html.zip
-rw-r--r-- 1 Lenovo 197121 0 Sep 4 16:13 ./tmp/html_root/html/xxxx.html
-rw-r--r-- 1 Lenovo 197121 0 Sep 4 16:13 ./tmp/html_root/10.html
-rw-r--r-- 1 Lenovo 197121 0 Sep 4 16:13 ./tmp/html_root/9.html
-rw-r--r-- 1 Lenovo 197121 0 Sep 4 16:13 ./tmp/html_root/8.html
-rw-r--r-- 1 Lenovo 197121 0 Sep 4 16:13 ./tmp/html_root/7.html
-rw-r--r-- 1 Lenovo 197121 0 Sep 4 16:13 ./tmp/html_root/6.html
-rw-r--r-- 1 Lenovo 197121 0 Sep 4 16:13 ./tmp/html_root/5.html
-rw-r--r-- 1 Lenovo 197121 0 Sep 4 16:13 ./tmp/html_root/4.html

```

图 10: 实例 8

### 9. 练习使用 Vim, 在你自己的机器上重做演示。

资料演示部分让我们将一个有问题的 fizz buzz 实现修复, 利用 G,o,ww,ea,ggO 等一系列操作完成问题的修改。在修复中, 需要记住每个操作的表示方法, 否则会使修改的过程极为复杂, 修改后的内容:

```

10 import sys
9 def fizz_buzz(limit):
8     for i in range(1,limit+1):
7         if i % 3 == 0:
6             print('fizz',end='')
5         if i % 5 == 0:
4             print('buzz',end='')
3         if i % 3 and i % 5:
2             print(i)
1
11 def main():
1     fizz_buzz(int(sys.argv[1]))
2
3 if __name__ == '__main__':
4     main()
5
~
~

```

图 11: 实例 9

### 10. 熟悉 vim 编辑器的多种模式转换

vim 编辑器有多种模式, 在起初的正常模式下, 键入 i 进入插入模式, R 进入替换模式, v 进入可视 (一般) 模式, V 进入可视 (行) 模式, Ctrl-V 进入可视 (块) 模式, : 进入命令模式。

```
~  
~  
[No Name] [unix] (07:59 01/01/1970)  
-- INSERT --
```

图 12: 插入模式

```
~  
~  
[No Name] [unix] (07:59 01/01/1970)  
-- REPLACE --
```

图 13: 替换模式

```
~  
~  
[No Name] [unix] (07:59 01/01/1970)  
-- VISUAL --
```

图 14: 可视（一般）模式

```
~  
~  
[No Name] [unix] (07:59 01/01/1970)  
-- VISUAL LINE --
```

图 15: 可视（行）模式

```
~  
~  
[No Name] [unix] (07:59 01/01/1970)  
-- VISUAL BLOCK --
```

图 16: 可视（块）模式

### 11. 用 shell 工具给变量赋值，并尝试区分单引号和双引号的不同

在赋值时中间不能出现空格，否则不能正常工作。对于引号的区别，以单引号定义的字符串为原义字符串，其中的变量不会被转义，而双引号定义的字符串会将变量值进行替换。



```
Lenovo@LAPTOP-FGVNVP6K MINGW64 ~ (main)
$ foo=bar

Lenovo@LAPTOP-FGVNVP6K MINGW64 ~ (main)
$ echo "$foo"
bar

Lenovo@LAPTOP-FGVNVP6K MINGW64 ~ (main)
$ echo '$foo'
$foo
```

图 17: 实例 11

12. 在文件中读取/d/系统开发工具基础/vim/man\_db.conf 的内容，并删除第一行。

先用 touch 创建一个新文件，再用 vim 将新创建的文件打开，在命令模式下输入：r /d/系统开发工具基础/vim/man\_db.conf 命令，就可以读取文件中的内容，然后在正常模式下可以用 dd 删除第一行。

```
Lenovo@LAPTOP-FGVNVP6K MINGW64 /d/系统开发工具基础/vim (master)
$ touch man_db.conf

Lenovo@LAPTOP-FGVNVP6K MINGW64 /d/系统开发工具基础/vim (master)
$ vim man_db.conf
```

图 18: 实例 12 (1)

```
~
man_db.conf [dos] (11:36 05/09/2024)
:r /d/系统开发工具基础/vim/man_db.conf
```

图 19: 实例 12 (2)

```
1 My Project
  1 Ingenuity Paradigm
  2 Convoluted Perspicacious
  3 Intricate
  4 Nuanced
  5 Elucidate
  6 Circumvent
  7 Quixotic
  8 Concomitant
  9 Oblique
 10 Ambivalent
 11 Eponymous
 12 Sate
 13 Facetious
 14 Dilemma
 15 Ingenuous
 16 Capricious
 17 Soporific
 18 Perfunctory
~
~
~
man_db.conf [dos] (11:36 05/09/2024)
1 more line; before #1 3 seconds ago
```

图 20: 实例 12 删除前

```
1 Ingenuity Paradigm
  1 Convoluted Perspicacious
  2 Intricate
  3 Nuanced
  4 Elucidate
  5 Circumvent
  6 Quixotic
  7 Concomitant
  8 Oblique
  9 Ambivalent
 10 Eponymous
 11 Sate
 12 Facetious
 13 Dilemma
 14 Ingenuous
 15 Capricious
 16 Soporific
 17 Perfunctory
~
```

图 21: 实例 12 删除后

13. 将文件中第一行到第 20 行的 ous 替换成 OUS，并在替换后完成恢复。

在 vim 编辑器中进入命令模式，执行：1, 20 s/ous/OUS/g 命令，就可以完成替换。若要恢复文件原来的内容，可以在正常模式下用 u 来实现。

```
man_db.conf[+] [dos] (11:36 05/09/2024)
: 1, 20 s/ous/OUS/g
```

图 22: 实例 13

```
12 Elucidate
11 Circumvent
10 Quixotic
9 Concomitant
8 Oblique
7 Ambivalent
6 EponymOUS
5 Satisfy
4 Facetious
3 Dilemma
2 Perfunctory
1 Ingenuity Paradigm
13 Convoluted Perspicacious
1 Intricate
```

图 23: 实例 13 替换后

```
6 Elucidate
5 Circumvent
4 Quixotic
3 Concomitant
2 Oblique
1 Ambivalent
7 Eponymous
1 Satisfy
2 Facetious
3 Dilemma
4 Perfunctory
5 Ingenuity Paradigm
6 Convoluted Perspicacious
7 Intricate
```

图 24: 实例 13 恢复后

14. 用 vim 编辑器将文件的第 10 行到第 14 行内容移动到文件的底部。

先将第 10 到第 14 行内容复制，用命令:10,14 y, 再通过 G 移动到最后一行，将复制的内容用 p 粘贴，最后在命令模式下执行: 10, 14 d, 从而将多余的内容删除，完成要求。

```
18 My Project
17 Ingenuity Paradigm
16 Convoluted Perspicacious
15 Intricate
14 Nuanced
13 Elucidate
12 Circumvent
11 Quixotic
10 Concomitant
9 oblique
8 Ambivalent
7 Eponymous
6 Satisfy
5 Facetious
4 Dilemma
3 Ingenuous
2 Capricious
1 Soporific
19 Perfunctory
```

图 25: 实例 14 开始内容

```
mal@UBUNTU [~] (11.30 03/09/2024)
:10,14 d
```

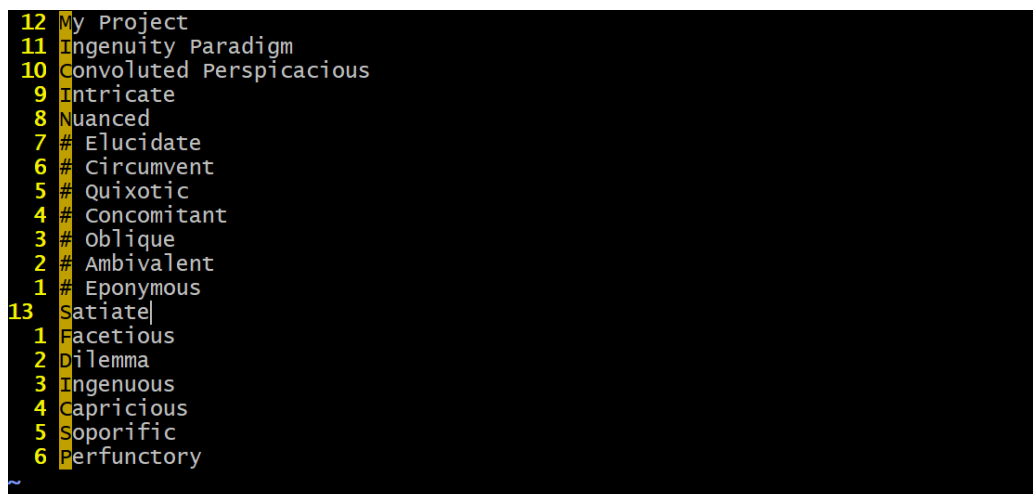
图 26: 实例 14 删除内容

```
9 My Project
8 Ingenuity Paradigm
7 Convoluted Perspicacious
6 Intricate
5 Nuanced
4 Elucidate
3 Circumvent
2 Quixotic
1 Concomitant
10 Dilemma
1 Ingenuous
2 Capricious
3 Soporific
4 Perfunctory
5 oblique
6 Ambivalent
7 Eponymous
8 Satisfy
9 Facetious
```

图 27: 实例 14 结束内容

### 15. Vim 给文件 README 的 6 到 12 行一次性添加 # 注释.

首先用 vim 打开 README 文件, 在命令模式输入:6,12s/ / # /就可以完成一次性注释。(其中, s 为替换命令, 表示行的开始), 注释后的结果如下:

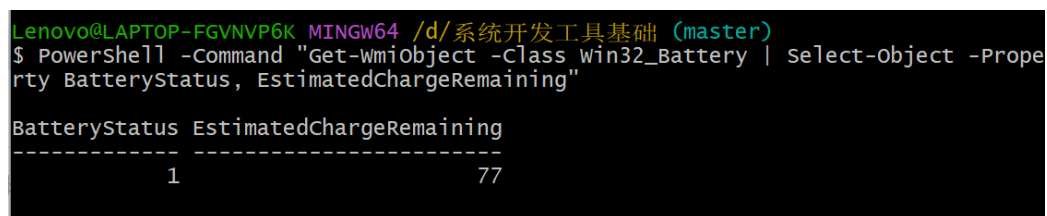


```
12 My Project
11 Ingenuity Paradigm
10 Convoluted Perspicacious
9 Intricate
8 Nuanced
7 # Elucidate
6 # Circumvent
5 # Quixotic
4 # Concomitant
3 # Oblique
2 # Ambivalent
1 # Eponymous
13 Satiated
1 Facetious
2 Dilemma
3 Ingenuous
4 Capricious
5 Soporific
6 Perfunctory
```

图 28: 实例 15

### 16. 写一段命令获取笔记本的电量信息。

Windows 操作系统中, /sys 目录不存在, 但可以使用 PowerShell 脚本获取笔记本的电池电量, 用 PowerShell -Command "Get-WmiObject -Class Win32\_Battery | Select-Object -Property BatteryStatus, EstimatedChargeRemaining" 即可获得电池的状态 (如是否正在充电、是否满电等) 和剩余电量 (以百分比形式)。



```
Lenovo@LAPTOP-FGVNVP6K MINGW64 /d/系统开发工具基础 (master)
$ PowerShell -Command "Get-WmiObject -Class Win32_Battery | Select-Object -Property BatteryStatus, EstimatedChargeRemaining"

BatteryStatus EstimatedChargeRemaining
-----
1 77
```

图 29: 实例 16

17. 编写一段 bash 脚本，运行如下的脚本直到它出错，将它的标准输出和标准错误流记录到文件，并在最后输出所有内容。

可以用 while 循环完成该脚本，然后运行一下看看结果。

```
Lenovo@LAPTOP-FGVNVP6K MINGW64 /d/系统开发工具基础 (master)
$ vim ./debug_for.sh

Lenovo@LAPTOP-FGVNVP6K MINGW64 /d/系统开发工具基础 (master)
$ ./debug_for.sh
failed after 34 runs
something went wrong
The error was using magic numbers
```

图 30: 实例 17

18. 请尝试用 vim 编辑器打开帮助文档

在正常模式下键入：进入命令模式，然后输入:help 标题 命令就可以打开帮助文档了。注意区分类似于 help :w 和 help w 的区别。下图为输入:help :w 后的结果。

```
BE CAREFUL! This might consume a lot of time, as the search of
'/u/user_x/**' includes '/u/user_x/work/**' and
'/u/user_x/work/release/**'. So '/u/user_x/work/release/**' is searched
three times and '/u/user_x/work/**' is searched twice.

In the above example you might want to set path to:
:set path=**,/u/user_x/**
This searches:
/u/user_x/work/release/**
/u/user_x/**
This searches the same directories, but in a different order.

Note that completion for ":find", ":sfind", and ":tabfind" commands do not
currently work with 'path' items that contain a URL or use the double star
with depth limiter (/usr/**2) or upward search (;) notations.

vim:tw=78:ts=8:noet:ft=help:norl:
editing txt[help] [-] [0] [unix] (18-15 29/07/2024) 1853 2 Rot
```

图 31: 实例 18

19. 统计文件中包含至少三个 a 且不以's 结尾的单词个数。

通过 cat 语句并根据要求编写条件进行统计。

```

Lenovo@LAPTOP-FGVNVP6K MINGW64 ~/.ssh (main)
$ cat /d/系统开发工具基础/README | tr "[:upper:]" "[:lower:]" | grep -E "^\([a]*a\){3}.*$" | grep -v "'s$" | wc -l
0

```

图 32: 实例 19

## 20. 统计一个文件中词尾两字母组合有多少种组合从未出现过

需要生成一个包含全部组合的列表，然后再使用已经得到的出现的组合，比较二者不同就可以了。可以选择将两个文件中相同的内容设置为空字符串，剩下的内容就是不同的内容。

```

Lenovo@LAPTOP-FGVNVP6K MINGW64 /d/系统开发工具基础 (master)
$ ./all.sh > all.txt

Lenovo@LAPTOP-FGVNVP6K MINGW64 /d/系统开发工具基础 (master)
$ cat /d/系统开发工具基础/README | tr "[:upper:]" "[:lower:]" | grep -E "^\([a]*a\){3}.*$" | grep -v "'s$" | sed -E "s/.*([a-z]{2})$/\1/" | sort | uniq > occurrence.txt

Lenovo@LAPTOP-FGVNVP6K MINGW64 /d/系统开发工具基础 (master)
$ diff --unchanged-group-format="" <(cat occurrence.txt) <(cat all.txt) | wc -l
676

```

图 33: 实例 20

## 2 实验结果

实验链接：

## 3 心得体会

在学习 shell 的时候，我发现它有许多的基本命令需要记住，只有掌握了基础的内容才能为以后更好的学习打下基础，但如果光评看资料是远远不够的，较多的命令容易被遗忘。在学习的过程中，我们更应该进行实际的操作，将每一个命令都执行一遍，遇到问题就上网查询或询问他人，这样才能加深我们对新学的东西的印象，同时也可以更块地理解每一个命令语句，从而在应用它们时更加熟练，从而提高效率。

Vim 作为最常见的文本编辑器之一，它真正做到了可以只用键盘进行操作，在提高效率方面有很大的意义。在最开始学习 vim 时，我被它多种的模式和通过按键实现各种各样的操作所震撼，同时也发现要学会它需要记忆的东西很多。但不需要鼠标的操作方式比其他的要方便很多，它也让我看到了学习一门新的工具的重要性。虽然我习惯应用熟悉的工具，但我在学习 Vim 的时候却意识到，如果能熟练应用它，效率一定会得到提高。所以，在以为我也会尝试用它编辑一些文件，争取将 Vim 熟练掌握。

无论是 shell，Vim 还是数据整理，都不是一两天就能完全掌握的，所以我们更应该将它们的相关知识应用在实际的操作中，通过不断地学习与应用发现并解决不会的地方，努力提高工作的效率。