

# Week2 Shell 工具和脚本、编辑器 (vim)、数据整理

黄琬晴

September 2025

本次实验的代码与实验报告上传在 github 仓库中:

<https://github.com/uuukyoo/systools>

## 目录

<b>1</b>	<b>Sell 工具和脚本</b>	<b>1</b>
1.1	检查 shell 是否满足实验要求 . . . . .	1
1.2	新建文件夹 . . . . .	1
1.3	新建文件 . . . . .	1
1.4	输入输出流重定向到文件 . . . . .	2
1.5	执行文件与权限 . . . . .	2
1.6	查看 chmod 使用手册 . . . . .	3
1.7	chmod 命令与权限 . . . . .	4
1.8	安装未安装的命令 . . . . .	5
1.9	管道 . . . . .	6
1.10	使用 ls 命令的不同参数来达成不同的目的 . . . . .	7
1.11	使用 bash 函数保存工作目录 . . . . .	9
1.12	使用 bash 函数查找脚本在失败前共运行了多少次 . . . . .	10

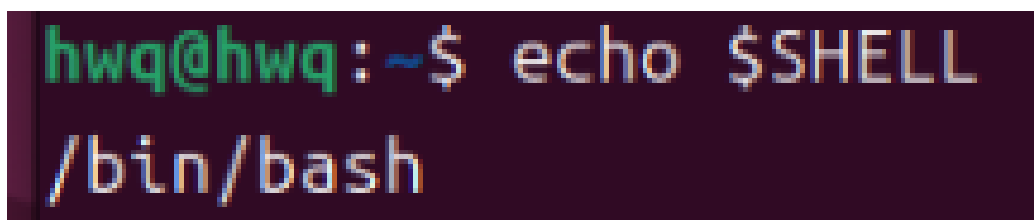
<b>2</b>	<b>编辑器 vim</b>	<b>13</b>
2.1	完成 vimtutor . . . . .	13
2.2	vim 意外中断 . . . . .	13
2.3	配置/.vimrc . . . . .	15
2.4	安装和配置一个插件: ctrlp.vim. . . . .	18
<b>3</b>	<b>数据整理</b>	<b>20</b>
3.1	sed 能否原地替换 . . . . .	20
3.2	开机时间统计 . . . . .	21
3.3	统计近三次开机的不同 . . . . .	22
3.4	统计 words 文件 . . . . .	23
<b>4</b>	<b>实验心得</b>	<b>23</b>

# 1 Sell 工具和脚本

## 1.1 检查 shell 是否满足实验要求

- 本课程需要使用类 Unix shell, 例如 Bash 或 ZSH。如果您在 Linux 或者 MacOS 上面完成本课程的练习, 则不需要做任何特殊的操作。如果您使用的是 Windows, 则您不应该使用 cmd 或是 Powershell; 您可以使用 Windows Subsystem for Linux 或者是 Linux 虚拟机。使用 `echo $SHELL` 命令可以查看您的 shell 是否满足要求。如果打印结果为 `/bin/bash` 或 `/usr/bin/zsh` 则是可以的

`echo $SHELL` 命令的结果为 `/bin/bash`, 说明我的 shell 符合本实验的要求

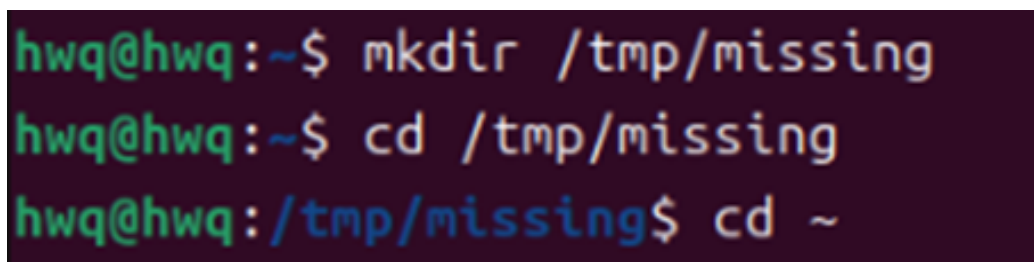


```
hwq@hwq:~$ echo $SHELL
/bin/bash
```

## 1.2 新建文件夹

- 在 `/tmp` 下新建一个名为 `missing` 的文件夹

使用 `mkdir` 命令新建目录, 再尝试用 `cd` 命令转到 `/tmp/missing` 文件夹下, 尝试成功, 说明新建目录成功



```
hwq@hwq:~$ mkdir /tmp/missing
hwq@hwq:~$ cd /tmp/missing
hwq@hwq:/tmp/missing$ cd ~
```

## 1.3 新建文件

- `touch` 在 `missing` 文件夹中新建一个叫 `semester` 的文件

在新建文件之前，使用 `ls` 命令查看 `missing` 文件夹下的文件，此时为空使用 `touch` 命令在对应目录下新建文件后，再次使用 `ls` 查看，此时 `semester` 文件出现，说明新建 `semester` 文件成功

```
hwq@hwq:~$ ls /tmp/missing
hwq@hwq:~$ touch /tmp/missing/semester
hwq@hwq:~$ ls /tmp/missing
semester
```

## 1.4 输入输出流重定向到文件

- 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
```

第一行使用 `echo` 和 `>` 来将内容覆盖，第二行使用 `echo` 和 `>>` 来在上一行后追加第二行；并且由于内容中含有有特殊含义的 `'#'` 和 `'!'` 字符，所以得使用单引号将内容括起来。输入完成后使用 `cat` 命令来检查输入的正确性

```
hwq@hwq:/tmp/missing$ echo '#!/bin/sh' > semester
hwq@hwq:/tmp/missing$ echo 'curl --head --silent https://missing.csail.mit.edu'
>> semester
hwq@hwq:/tmp/missing$ cat semester
#!/bin/sh
curl --head --silent https://missing.csail.mit.edu
```

## 1.5 执行文件与权限

- 尝试执行这个文件。例如，将该脚本的路径 (`./semester`) 输入到您的 `shell` 中并回车。如果程序无法执行，请使用 `ls` 命令来获取信息并理解其不能执行的原因

尝试执行输入 `./semester` 尝试执行该文件，发现无法正常执行，显示权限不够

```
hwq@hwq:/tmp/missing$ ./semester
bash: ./semester: 权限不够
```

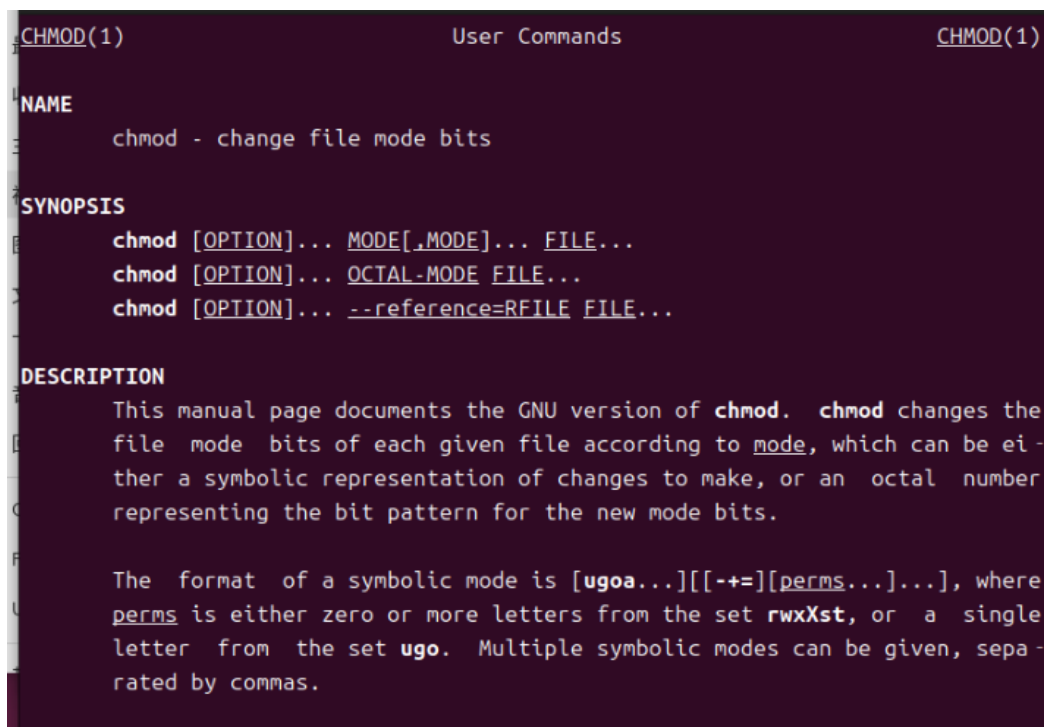
使用 `ls -l` 以长格式查看 `missing` 目录下的文件，发现 `semester` 文件没有执行权限，及 `x` 权限

```
hwq@hwq:/tmp/missing$ ls -l
总计 4
-rw-rw-r-- 1 hwq hwq 61  9月 10 10:09 semester
```

所以猜想该文件无法正常执行的原因是因为它没有执行权限，用户无法执行没有对该文件的执行权限就无法执行它

## 1.6 查看 `chmod` 使用手册

输入 `man chmod` 来查看 `chmod` 的使用手册



从使用手册中可以得知,可以使用 `chmod +x semester` 命令来为 `semester` 文件增加执行权限来解决其无法执行的问题

其中的 `+` 表示为文件增加权限,与之类似的还有`-`表示去除权限,`=` 表示将文件对应的用户权限重新设为 `=` 后的内容,常见权限有 `r`(read 读), `w`(write 写), `x`(execute 执行) 等

## 1.7 chmod 命令与权限

- 使用 `chmod` 命令改变权限,使 `./semester` 能够成功执行,不要使用 `sh semester` 来执行该程序。您的 `shell` 是如何知晓这个文件需要使用 `sh` 来解析呢?

使用 `chmod +x semester`, `+x` 表示添加执行权限, `semester` 表示是为 `semester` 文件添加执行权限,再次使用 `ls -l` 查看,此时就可以发现 `semester` 文件已有执行权限

```
hwq@hwq:/tmp/missing$ chmod +x semester
hwq@hwq:/tmp/missing$ ls -l
总计 4
-rwxrwxr-x 1 hwq hwq 61  9月 10 10:09 semester
```

再次尝试执行，还是无法成功，由显示的 `curl: not found` 可知，这此无法成功的原因大概是 `curl` 命令未安装

```
hwq@hwq:/tmp/missing$ ./semester
./semester: 2: curl: not found
```

## 1.8 安装未安装的命令

使用 `sudo apt install curl` 命令安装 `curl` 命令

```
hwq@hwq:/tmp/missing$ sudo apt install curl
[sudo] hwq 的密码：
正在读取软件包列表... 完成
正在分析软件包的依赖关系树... 完成
正在读取状态信息... 完成
下列【新】软件包将被安装：
  curl
升级了 0 个软件包，新安装了 1 个软件包，要卸载 0 个软件包，有 248 个软件包未被升级。
需要下载 226 kB 的归档。
解压缩后会消耗 534 kB 的额外空间。
获取:1 http://mirrors.tuna.tsinghua.edu.cn/ubuntu noble-updates/main amd64 curl
amd64 8.5.0-2ubuntu10.6 [226 kB]
已下载 226 kB，耗时 3秒 (75.3 kB/s)
正在选中未选择的软件包 curl。
(正在读取数据库 ... 系统当前共安装有 195950 个文件和目录。)
准备解压 .../curl_8.5.0-2ubuntu10.6_amd64.deb ...
正在解压 curl (8.5.0-2ubuntu10.6) ...
正在设置 curl (8.5.0-2ubuntu10.6) ...
正在处理用于 man-db (2.12.0-4build2) 的触发器 ...
```

再次尝试执行，执行成功，现在可以看见执行结果

```

hwq@hwq:/tmp/missing$ ./semester
HTTP/2 200
server: GitHub.com
content-type: text/html; charset=utf-8
last-modified: Thu, 28 Aug 2025 13:37:00 GMT
access-control-allow-origin: *
etag: "68b05b7c-2002"
expires: Tue, 09 Sep 2025 15:36:28 GMT
cache-control: max-age=600
x-proxy-cache: MISS
x-github-request-id: 2092:E3D4C:9730D1:9FF3B5:68C04721
accept-ranges: bytes
date: Wed, 10 Sep 2025 02:11:42 GMT
via: 1.1 varnish
age: 0
x-served-by: cache-tyo11923-TYO
x-cache: HIT
x-cache-hits: 1
x-timer: S1757470302.842254,VS0,VE180
vary: Accept-Encoding
x-fastly-request-id: f4c25c6fbb3bc97fb03ca16b84153e3b3d949d3a
content-length: 8194

```

shell 知晓这个文件需要使用 `sh` 来解析,是因为先前我们在编写 `semester` 时,将 `#!/bin/sh` 写在了文件的第一行,执行它时,这一行的功能就是告诉操作系统需要用 `/bin/sh` 作为解释器,也就是告诉 shell 这个文件需要使用 `sh` 来解析

## 1.9 管道

- 使用 `|` 和 `>`, 将 `semester` 文件输出的最后更改日期信息,写入主目录下的 `last-modified.txt` 的文件中

执行 `semester` 文件的结果通过管道传递,由 `grep` 过滤出包含 `last-modified` 的那一行,将该行内容写入到主目录的 `last-modified.txt` 文件中可以用 `cat` 查看文件内容,可以发现与上次执行结果的 `last-modified` 相一致

```

bash: ./semester: 没有那个文件或目录
hwq@hwq:/tmp/missing$ ./semester | grep last-modified > ~/last-modified.txt
hwq@hwq:/tmp/missing$ cat ~/last-modified.txt
last-modified: Thu, 28 Aug 2025 13:37:00 GMT

```



## 1.10 使用 ls 命令的不同参数来达成不同的目的

- 阅读 man ls ，然后使用 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）
  - 文件以最近修改顺序排序
  - 以彩色文本显示输出结果

使用-a 参数可以显示出包括隐藏文件的所有文件，对比 ls 和 ls -a 的结果可以看出，显示中多了许多以“.”开头的隐藏文件

```
hwq@hwq:~$ ls
公共  下载      client.key  lock      snap      virtual.py
模板  音乐      file_to_send.txt lock.c     unlock     wk3_1_vir.py
视频  桌面      httpserver.py received_file.txt unlock.c   wk3_2_vir.py
图片  ca.crt    lab5_vir.py  server2.py vir.crt    wk4vir.py
文档  client.crt last-modified.txt server.py  vir.key

hwq@hwq:~$ ls -a
.          .bash_logout  .lessht      unlock
..         .bashrc      .local        unlock.c
公共       .cache       lock          .viminfo
模板       ca.crt       lock.c        vir.crt
视频       client.crt   .profile      vir.key
图片       client.key   received_file.txt virtual.py
文档       .config     server2.py    .virtual.py.swp
下载       file_to_send.txt server.py      wk3_1_vir.py
音乐       httpserver.py snap          wk3_2_vir.py
桌面       lab5_vir.py  .ssh         wk4vir.py
.bash_history last-modified.txt .sudo_as_admin_successful
```

使用-h 参数可以实现文件打印以人类可以理解的格式输出，首先使用-l 可以输出文件的详细信息，可以看见此时显示的 4096 并没有具体的单位

```
hwq@hwq:~$ ls -l
总计 140
drwxr-xr-x 2 hwq hwq 4096 12月 15 2024 公共
drwxr-xr-x 2 hwq hwq 4096 12月 15 2024 模板
drwxr-xr-x 2 hwq hwq 4096 12月 15 2024 视频
drwxr-xr-x 3 hwq hwq 4096 9月 5 09:53 图片
drwxr-xr-x 2 hwq hwq 4096 12月 15 2024 文档
drwxr-xr-x 2 hwq hwq 4096 12月 15 2024 下载
drwxr-xr-x 2 hwq hwq 4096 12月 15 2024 音乐
drwxr-xr-x 2 hwq hwq 4096 12月 15 2024 桌面
```

接着使用了 `ls -l -h`，可以看见 4096 变成了 4.0k，更易于人类理解

```
hwq@hwq:~$ ls -l -h
总计 140K
drwxr-xr-x 2 hwq hwq 4.0K 12月 15 2024 公共
drwxr-xr-x 2 hwq hwq 4.0K 12月 15 2024 模板
drwxr-xr-x 2 hwq hwq 4.0K 12月 15 2024 视频
drwxr-xr-x 3 hwq hwq 4.0K 9月 5 09:53 图片
drwxr-xr-x 2 hwq hwq 4.0K 12月 15 2024 文档
drwxr-xr-x 2 hwq hwq 4.0K 12月 15 2024 下载
drwxr-xr-x 2 hwq hwq 4.0K 12月 15 2024 音乐
drwxr-xr-x 2 hwq hwq 4.0K 12月 15 2024 桌面
```

使用 `-t` 可以按修改时间排序

```
hwq@hwq:~$ ls -t
图片      client.key      wk3_1_vir.py  lock.c      视频
last-modified.txt  ca.crt        server2.py    unlock      文档
lab5_vir.py      wk4vir.py     httpserver.py unlock.c     下载
vir.crt         file_to_send.txt  virtual.py   snap       音乐
vir.key        received_file.txt  server.py    公共      桌面
client.crt     wk3_2_vir.py    lock        模板
```

使用 `-color=auto` 参数来以彩色文本输出，但是由于我的 shell 本来就是以彩色文本输出，所以输出并没有区别

```
hwq@hwq:~$ ls -l --color=auto
总计 140
drwxr-xr-x 2 hwq hwq 4096 12月 15 2024 公共
drwxr-xr-x 2 hwq hwq 4096 12月 15 2024 模板
drwxr-xr-x 2 hwq hwq 4096 12月 15 2024 视频
drwxr-xr-x 3 hwq hwq 4096 9月 5 09:53 图片
drwxr-xr-x 2 hwq hwq 4096 12月 15 2024 文档
drwxr-xr-x 2 hwq hwq 4096 12月 15 2024 下载
drwxr-xr-x 2 hwq hwq 4096 12月 15 2024 音乐
drwxr-xr-x 2 hwq hwq 4096 12月 15 2024 桌面
-rw----- 1 hwq hwq 1001 5月 16 22:03 ca.crt
-rw----- 1 hwq hwq 985 5月 16 22:03 client.crt
-rw----- 1 hwq hwq 1704 5月 16 22:03 client.key
-rw----- 1 hwq hwq 302 5月 6 21:59 file_to_send.txt
-rw-rw-r-- 1 hwq hwq 1234 3月 21 17:01 httpserver.py
-rw----- 1 hwq hwq 3430 5月 16 23:05 lab5_vir.py
-rw-rw-r-- 1 hwq hwq 46 9月 5 09:43 last-modified.txt
-rwxrwxr-x 1 hwq hwq 16168 12月 25 2024 lock
-rw-rw-r-- 1 hwq hwq 1015 12月 25 2024 lock.c
-rw-rw-r-- 1 hwq hwq 302 5月 6 20:23 received_file.txt
-rw-rw-r-- 1 hwq hwq 1324 4月 4 21:01 server2.py
```

## 1.11 使用 bash 函数保存工作目录

- 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

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

新建 marco.sh 文件，内容如下

```
#!/bin/bash
marco(){
    echo "$(pwd)" > $HOME/marco_history.log
    echo "save pwd $(pwd)"
}
polo(){
    cd "$(cat "$HOME/marco_history.log")"
}

~
~
```

键入 source marco.sh 加载函数

```
hwq@hwq:~$ source marco.sh
```

此时输入 marco 之后，当前工作目录已保存转到其他目录后输入 polo，就转回原来保存的工作目录

```
hwq@hwq:~$ cat /usr/share/dict/words | tr "[:upper:]" "[:lower:]" | grep -E "^(
^a]*a){3}.*$" | grep -v "'s$" | sed -E "s/.*([a-z]{2})$/\1/" | sort | uniq -c |
sort | tail -n3
      8 am
      8 ce
      9 ca
```

## 1.12 使用 bash 函数查找脚本在失败前共运行了多少次

- 假设您有一个命令，它很少出错。因此为了在出错时能够对其进行调试，需要花费大量的时间重现错误并捕获输出。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。加分项：报告脚本在失败前共运行了多少次。

```
#!/usr/bin/env bash

n=$(( RANDOM % 100 ))

if [[ n -eq 42 ]]; then
    echo "Something went wrong"
    && echo "The error was using magic numbers"
    exit 1
fi

echo "Everything went according to plan"
```

将题目中给到的脚本保存在 buggy.sh 中并将如下内容保存在 debug\_for.sh 中

```
#!/usr/bin/env bash

count=0
echo > out.log

while true
do
    ./buggy.sh &>> out.log
    if [[ $? -ne 0 ]]; then
        cat out.log
        echo "failed after $count times"
        break
    fi
    ((count++))
done
```

done

```
hwq@hwq:~$ vim buggy.sh
hwq@hwq:~$ vim debug_for.sh
```

为它们添加执行权限并执行 debug\_for.sh

```
hwq@hwq:~$ ./debug_for.sh
bash: ./debug_for.sh: 权限不够
hwq@hwq:~$ chmod +x buggy.sh
hwq@hwq:~$ chmod +x debug_for.sh
hwq@hwq:~$ ./debug_for.sh
```

由运行结果可知在失败之前运行了 78 次

```
Everything went according to plan
Everything went according to plan
Something went wrong
The error was using magic numbers
failed after 78 times
hwq@hwq:~$
```

通过 out.log 中的记录来验证 debug\_for.sh 的结果，确实为 78 次

```
hwq@hwq:~$ cat out.log | grep Everything | wc -l
78
```

## 2 编辑器 vim

### 2.1 完成 vimtutor

- 完成 vimtutor。备注：它在一个 80x24 (80 列, 24 行) 终端窗口看起来效果最好

在终端中输入 vimtutor 命令，完成对 vim 的基本操作的练习

```
要了解更多信息请输入 :help vimrc-intro

-----
                          第七讲第三节：补全功能

          ** 使用 CTRL-D 和 <TAB> 可以进行命令行补全 **

1. 请确保 Vim 不是在以兼容模式运行： :set nocompatible
2. 查看一下当前目录下已经存在哪些文件，输入： :!ls  或者  :!dir
3. 现在输入一个目录的起始部分，例如输入： :e
4. 接着按 CTRL-D 键，Vim 会显示以 e 开始的命令的列表。
5. 然后按 <TAB> 键，Vim 会补全命令为 :edit 。
6. 现在添加一个空格，以及一个已有文件的文件名的起始部分，例如： :edit FILE
7. 接着按 <TAB> 键，Vim 会补全文件名(如果它是惟一匹配的)。
```

### 2.2 vim 意外中断

若在使用 vim 编辑文件时意外退出



再次尝试打开文件时就会显示如下





```
hwq@hwq:~$ rm .test.swp
```

此时可以输入 `vim -r test` 来使用交换文件来恢复原始文件

```
使用交换文件 ".test.swp"
原始文件 "~/test"
恢复完毕。请确定一切正常。
(你可能想要将这个文件另存为别的文件名
再运行 diff 与原文件比较以检查是否有改变)
你现在可以删除 .swp 文件了。

请按 ENTER 或其它命令继续
```

输入：`wq` 保存即可

```
123
do not save
```

保存完后即可将交换文件删除

## 2.3 配置`~/.vimrc`

- 下载我们的 `vimrc`，然后把它保存到 `~/.vimrc`。通读这个注释详细的文件（用 Vim!），然后观察 Vim 在这个新的设置下看起来和使用起来

有哪些细微的区别。

将 ~/.vimrc 文件设置为提供的 vimrc 文件

```
Comments in Vimscript start with a `\"`.

If you open this file in Vim, it'll be syntax highlighted for you.

Vim is based on Vi. Setting `nocompatible` switches from the default
Vi-compatibility mode and enables useful Vim functionality. This
configuration option turns out not to be necessary for the file named
`~/.vimrc`, because Vim automatically enters nocompatible mode if that file
is present. But we're including it here just in case this config file is
loaded some other way (e.g. saved as `foo`, and then Vim started with
`vim -u foo`).
set nocompatible

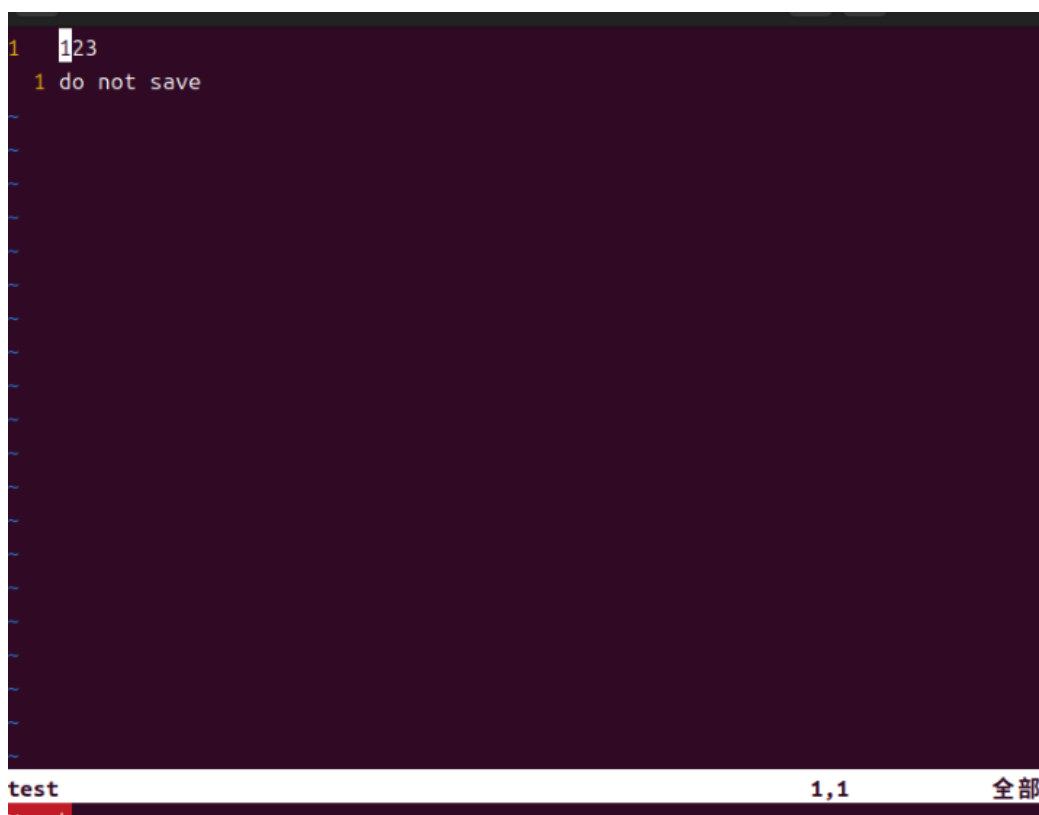
Turn on syntax highlighting.
syntax on

Disable the default Vim startup message.
set shortmess+=I

Show line numbers.
set number

This enables relative line numbering mode. With both number and
```

再次打开 vim，比较显著的差别有，方向键被禁用了，鼠标可以定位，下方有如“use J”这样的提示，并且出现了类似行号和状态栏的东西



```
1 123
1 do not save
```

test 1,1 全部

在看了文件内容后，可以得知它实现的功能：set nocompatible 关闭 vi 兼容模式，启用 vim 的增强功能

syntax on 打开语法高亮。

set shortmess+=I 禁用 Vim 启动时的欢迎消息。

set number / set relativenumber 打开行号，同时启用相对行号（当前行为实际行号，其余显示相对行号）。

set laststatus=2 始终显示状态栏。

set backspace=indent,eol,start 改善退格键行为，使其能删除缩进、换行符、插入点之前的内容。

set hidden 允许切换到未保存的缓冲区，而不会强制保存。

set ignorecase 搜索默认忽略大小写。

set smartcase 如果搜索词中包含大写字母，则大小写敏感。

set incsearch 输入搜索内容时就实时显示匹配，而不是等回车。

nmap Q <Nop> 禁用 Q 键进入 Ex 模式。

`set noerrorbells visualbell t_vb=` 关闭报错提示音和屏幕闪烁。

`set mouse+=a` 启用鼠标支持

禁止使用方向键（提示使用用 `hjkl` 来移动）

## 2.4 安装和配置一个插件：ctrlp.vim.

1. 用 `mkdir -p ~/.vim/pack/vendor/start` 创建插件文件夹
2. 下载这个插件：`cd ~/.vim/pack/vendor/start; git clone https://github.com/ctrlpvim/ctrlp.vim`
3. 阅读这个插件的 文档。尝试用 `CtrlP` 来在一个工程文件夹里定位一个文件，打开 Vim, 然后用 Vim 命令控制行开始 `:CtrlP`.
4. 自定义 `CtrlP`：添加 `configuration` 到你的 `~/.vimrc` 来用按 `Ctrl-P` 打开 `CtrlP`

创建文件夹，并转到该文件夹下下载该插件

```
hwq@hwq:~$ mkdir -p ~/.vim/pack/vendor/start
hwq@hwq:~$ cd ~/.vim/pack/vendor/start
hwq@hwq:~/.vim/pack/vendor/start$ git clone https://github.com/ctrlpvim/ctrlp.vim
```

在`~/.vimrc`中添加如下内容

```
set runtimepath^=~/.vim/pack/vendor/start/ctrlp.vim
```

通过`vim ~/.vim/pack/vendor/start/ctrlp.vim/doc/ctrlp.txt`进入插件的文档目录来阅读该文件的文档

```

n - When bigger than 1, disable caching and use the number as the limit to
    enable caching again.

Note: you can quickly purge the cache by pressing <F5> while inside CtrlP.

                                *'g:ctrlp_clear_cache_on_exit'*
Set this to 0 to enable cross-session caching by not deleting the cache files
upon exiting Vim: >
    let g:ctrlp_clear_cache_on_exit = 1
*

                                *'g:ctrlp_cache_dir'*
Set the directory to store the cache files: >
    let g:ctrlp_cache_dir = $HOME.'/.cache/ctrlp'
*

                                *'g:ctrlp_show_hidden'*
Set this to 1 if you want CtrlP to scan for dotfiles and dotdirs: >
    let g:ctrlp_show_hidden = 0
*

Note: does not apply when a command defined with |g:ctrlp_user_command| is
being used.

241,1      13%

```

在 vim 命令行里输入:CtrlP 即可开始使用 CtrlP 插件的功能

```

doc/ctrlp.txt      241,1      13%
> autoload/ctrlp/mixed.vim
> autoload/ctrlp.vim
> plugin/ctrlp.vim
> readme.md
> autoload/ctrlp/undo.vim
> autoload/ctrlp/line.vim
> autoload/ctrlp/tag.vim
> autoload/ctrlp/dir.vim
> doc/ctrlp.cnx
> LICENSE
prt path <mru>={ files }=<buf> <-> <ome/hwq/.vim/pack/vendor/start/ctrlp.vim

```

在 ~/.vimrc 中添加如下内容

```
let g:ctrlp_map = '<c-p>'
let g:ctrlp_cmd = 'CtrlP'
let g:ctrlp_working_path_mode = 'ra'
```

再次使用 vim 查看一个文件，此时只需按下 Ctrl+P 即可打开文件查找

```
#      +#+      +#+      +#+      +#+ +#+      +#+      #
#      ##      ##      ##      ##      ##      ##      ##      #
doc/ctrlp.txt 1,1 顶端
> autoload/ctrlp/mixed.vim
> autoload/ctrlp.vim
> plugin/ctrlp.vim
> readme.md
> autoload/ctrlp/undo.vim
> autoload/ctrlp/line.vim
> autoload/ctrlp/tag.vim
> autoload/ctrlp/dir.vim
> doc/ctrlp.cnx
> LICENSE
prt path <mru>={ files }=<buf> <-> <ome/hwq/.vim/pack/vendor/start/ctrlp.vim
>>>
```

:

## 3 数据整理

### 3.1 sed 能否原地替换

- 进行原地替换听上去很有诱惑力，例如：`sed s/REGEX/SUBSTITUTION/ input.txt > input.txt`。但是这并不是一个明智的做法，为什么呢？还是说只有 sed 是这样的？查看 `man sed` 来完成这个问题

因为在运行 `sed s/REGEX/SUBSTITUTION/ input.txt > input.txt` 时，`>input.txt` 会在 sed 启动前就执行，所以在 sed 启动前 `input.txt` 就已清空，当

sed 再尝试读取时，input.txt 就已是空文件，所以结果只能为空而无法得到正确的结果，并且还会丢失原来的 input.txt 文件的内容

并且不只是 sed 有问题，输出重定向会先将目标文件清空，所以任何的类似的原地替换的命令都会导致文件清空无法得到正确结果，如

```
grep foo file.txt > file.txt
awk '{print $1}' file.txt > file.txt
```

等指令都是如此

## 3.2 开机时间统计

- 找出您最近十次开机的开机时间平均数、中位数和最长时间。

编写脚本 getlog.sh 如下，来获取最近十次的开机时间

```
#!/bin/bash
for i in {0..9}; do
    journalctl -b-$i | grep "Startup finished in"
done
```

将执行结果存入到 starttime.txt 中

```
hwq@hwq:~$ ./getlog.sh > starttime.txt
```

查找出最长时间，最短时间，平均数中位数分别为：27.067s, 25.056s, 25.8622s, 26.051s

```
hwq@hwq:~$ cat starttime.txt | grep "systemd\[1\]" | sed -E "s/.*=\ (.*?)s\./\1/" | sort | tail -n1
27.067
hwq@hwq:~$ cat starttime.txt | grep "systemd\[1\]" | sed -E "s/.*=\ (.*?)s\./\1/" | sort -r | tail -n1
25.056
hwq@hwq:~$ cat starttime.txt | grep "systemd\[1\]" | sed -E "s/.*=\ (.*?)s\./\1/" | paste -sd+ | bc -l | awk '{print $1/10}'
25.8622
hwq@hwq:~$ cat starttime.txt | grep "systemd\[1\]" | sed -E "s/.*=\ (.*?)s\./\1/" | sort | paste -sd\ | awk '{print ($5+$6)/2}'
26.051
```

### 3.3 统计近三次开机的不同

- 查看之前三次重启启动信息中不同的部分

将上一题的 getlog.sh 改为获得近三次启动信息

```
#!/bin/bash
for i in {0..2}; do
    journalctl -b-$i | grep "Startup finished in"
done
```

将近三次启动信息存储到文件 last3.txt 中，查找除时间戳之外的不同

```
hwq@hwq:~$ ./getlog.sh > last3.txt
hwq@hwq:~$ cat last3.txt | sed -E "s/.*pi\ (.*?)\1/" | sort | uniq -c | sort |
awk '$1!=3 { print }'
    1 9月 12 09:40:05 hwq systemd[1652]: Startup finished in 141ms.
    1 9月 12 09:40:26 hwq systemd[1]: Startup finished in 3.209s (kernel) + 22
.838s (userspace) = 26.047s.
    1 9月 16 08:48:17 hwq systemd[1655]: Startup finished in 130ms.
    1 9月 16 08:48:37 hwq systemd[1]: Startup finished in 3.048s (kernel) + 23
.007s (userspace) = 26.055s.
    1 9月 16 09:36:29 hwq systemd[1631]: Startup finished in 114ms.
    1 9月 16 09:36:49 hwq systemd[1]: Startup finished in 2.299s (kernel) + 23
.746s (userspace) = 26.045s.
```



### 3.4 统计 words 文件

- 统计 words 文件 (/usr/share/dict/words) 中包含至少三个 a 且不以's 结尾的单词个数。这些单词中，出现频率前三的末尾两个字母是什么？ sed 的 y 命令，或者 tr 程序也许可以帮你解决大小写的问题。共存在多少种词尾两字母组合？

查找包含至少三个 a 且不是以's 结尾的单词个数，为 854 个

```
hwq@hwq:~$ cat /usr/share/dict/words | tr "[:upper:]" "[:lower:]" | grep -E "^([a]*a){3}.*$" | grep -v "'s$" | wc -l
854
```

频率前三的末尾两个字母分别是 am, ce, ca

```
hwq@hwq:~$ cat /usr/share/dict/words | tr "[:upper:]" "[:lower:]" | grep -E "^([a]*a){3}.*$" | grep -v "'s$" | sed -E "s/.*([a-z]{2})$/\1/" | sort | uniq -c | sort | tail -n3
      8 am
      8 ce
      9 ca
```

共存在 112 种词尾两字母组合

```
hwq@hwq:~$ cat /usr/share/dict/words | tr "[:upper:]" "[:lower:]" | grep -E "^([a]*a){3}.*$" | grep -v "'s$" | sed -E "s/.*([a-z]{2})$/\1/" | sort | uniq | wc -l
112
```

## 4 实验心得

通过本次实验我学习了 shell 的使用方法，如何自定义 vim 编辑器的功能、插件等，还学习了基本的数据整理方法

之前我对它们的了解仅仅在大概会用但是不是很了解的程度上，每次如果遇到了什么记不住的指令或者不知道如何解决的问题都是现用现搜，通过本次学习，本来只在我大脑里有模糊概念的知识变得轮廓清晰了起来