课程作业报告

课程名称：系统工具开发基础

学生姓名：张家玮

学号：23020007157

日期：2024 年 9 月 6 日

目录

Chapter 1

Shell脚本和工具

1.1 ls指令

为使ls命令能够完成如下要求的操作：

- 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

为了完成如上的要求我们需要加入一些后缀来实现这些功能：

- -l: 使用长格式输出，包括文件权限、链接数、所有者、组、文件大小和最后修改时间。
- -A: 列出所有文件，包括隐藏文件（以 . 开头的文件），但不包括 .（当前目录）和 ..（上一级目录）。
- -h: 以人类可读的格式显示文件大小。
- –sort=atime: 按访问时间排序。
- –color=auto: 以彩色文本显示输出结果。这个选项会根据文件类型显示不同的颜色，比如目录会以蓝色显示，普通文件以默认颜色显示。

在这里我们使用如下的指令,可以实现我们想要的功能。

```
ls -lAh --sort=time --color=auto
```

```
23724@Gavin MINGW64 ~
$ ls -lAh --sort=time --color=auto
total 186M
drwxr-xr-x 1 23724 197609    0 Sep  5 14:55  OneDrive/
drwxr-xr-x 1 23724 197609    0 Sep  5 11:55 'WPS Cloud Files'/
drwxr-xr-x 1 23724 197609    0 Sep  5 08:58  Videos/
drwxr-xr-x 1 23724 197609    0 Sep  4 15:42  Desktop/
drwxr-xr-x 1 23724 197609    0 Sep  4 14:38  Downloads/
-rw-r--r-- 1 23724 197609  19M Sep  4 10:42  NTUSER.DAT
drwxr-xr-x 1 23724 197609    0 Sep  1 17:45  Documents/
-rw-r--r-- 1 23724 197609 6.9K Aug 30 18:29  .bash_history
-rw-r--r-- 1 23724 197609   76 Aug 30 17:31  .gitignore_global
-rw-r--r-- 1 23724 197609  409 Aug 30 17:28  .gitconfig
-rw-r--r-- 1 23724 197609   20 Aug 30 16:32  .lesshst
drwxr-xr-x 1 23724 197609    0 Aug 27 17:10  .texlive2024/
drwxr-xr-x 1 23724 197609    0 Aug 24 17:14  .ssh/
-rw-r--r-- 1 23724 197609  743 Aug 24 17:02  mykey.pub
-rw-r--r-- 1 23724 197609 3.4K Aug 24 17:02  mykey
-rw-r--r-- 1 23724 197609  690 Aug 24 17:00  .viminfo
-rw-r--r-- 1 23724 197609  12K Aug 24 17:00  .id_rsa.swp
-rwxr-xr-x 1 23724 197609  66M Aug 23 10:40  Git-2.46.0-64-bit.exe*
-rw-r--r-- 1 23724 197609  64K Aug 19 21:55  NTUSER.DAT{98eeab7f-61eb-11ee-ac
-rw-r--r-- 1 23724 197609 512K Aug 19 21:55  NTUSER.DAT{98eeab7f-61eb-11ee-ac
drwxr-xr-x 1 23724 197609    0 Aug 19 11:40  WPSDrive/
drwxr-xr-x 1 23724 197609    0 Jul 20 22:13  Pictures/
drwxr-xr-x 1 23724 197609    0 Mar 14 20:23  .leigod/
drwxr-xr-x 1 23724 197609    0 Jan  9  2024  .config/
drwxr-xr-x 1 23724 197609    0 Dec 25  2023  .pintia/
drwxr-xr-x 1 23724 197609    0 Dec 10  2023  ansel/
drwxr-xr-x 1 23724 197609    0 Dec  9  2023  AppData/
drwxr-xr-x 1 23724 197609    0 Dec  3  2023  .obs-browser-plus/
drwxr-xr-x 1 23724 197609    0 Nov  6  2023  .vscode/
-rwxr-xr-x 1 23724 197609  91M Nov  6  2023  VSCodeUserSetup-x64-1.84.0.exe*
drwxr-xr-x 1 23724 197609    0 Nov  2  2023  sangfor/
drwxr-xr-x 1 23724 197609    0 Oct 27  2023  .dotnet/
drwxr-xr-x 1 23724 197609    0 Oct  8  2023  source/
drwxr-xr-x 1 23724 197609    0 Oct  4  2023  .templateengine/
drwxr-xr-x 1 23724 197609    0 Oct  3  2023  Searches/
drwxr-xr-x 1 23724 197609    0 Oct  3  2023  Links/
drwxr-xr-x 1 23724 197609    0 Oct  3  2023 'Saved Games'/
drwxr-xr-x 1 23724 197609    0 Oct  3  2023  Music/
drwxr-xr-x 1 23724 197609    0 Oct  3  2023  Favorites/
drwxr-xr-x 1 23724 197609    0 Oct  3  2023  Contacts/
```

得到的效果上图。

1.2 bash function

我们先创建一个文件夹开始编辑这几个函数

如下图中我们创建一个文件夹用于存放用于存放。具体的内容如下

MINGW64:/c/Users/23724

```
# 保存当前目录
marco() {
    export SAVED_DIR=$(pwd)
}

# 回到保存的目录
polo() {
    if [ -z "$SAVED_DIR" ]; then
        echo "No directory saved. Run marco first."
    else
        cd "$SAVED_DIR" || echo "Failed to change directory to $SAVED_DIR"
    fi
}

~
~
```

然后我们在使用source命令进行重载，最后测试一下这两个函数是否能够完成我们所预设的功能。

```
23724@Gavin MINGW64 ~
$ touch marco_polo.sh

23724@Gavin MINGW64 ~
$ vim marco_polo.sh

23724@Gavin MINGW64 ~
$ source marco_polo.sh

23724@Gavin MINGW64 ~
$ cd "C:\Users\23724\Desktop\系统工具开发\Shell"

23724@Gavin MINGW64 ~/Desktop/系统工具开发/Shell
$ marco

23724@Gavin MINGW64 ~/Desktop/系统工具开发/Shell
$ cd "C:\Users\23724\Desktop\系统工具开发\Vim"

23724@Gavin MINGW64 ~/Desktop/系统工具开发/Vim
$ polo

23724@Gavin MINGW64 ~/Desktop/系统工具开发/Shell
$
```

可以看到函数的功能运行正常。

1.3 bash脚本

为实现能够自动运行这个命令指导出错，我们可以写一个Bash脚本运行给我们的指令直到出错，同时可以加入一个计数器来显示出出错时运行了多少遍。

这个脚本的代码如下图

```
counter=0
#初始化计数器

#循环脚本直至出错
while true;do
        ((counter++))
        ./run.sh >> output.log 2>> error.log

        if [[ $? -ne 0 ]]; then
                echo "failed after $counter runs."
                echo "Standard Output:"
                cat output.log
                echo
                echo "Standard Error:"
                cat error.log
                break
        fi
done
~
```

然后我们运行这个脚本可以看到：

```
23724@Gavin MINGW64 ~/Desktop/系统工具开发/Shell
$ ./catch.sh
failed after 247 runs.
Standard Output:
Everything went according to plan
Everything went according to plan
Everything went according to plan
Everything went according to plan
Everything went according to plan
Everything went according to plan
Everything went according to plan
Everything went according to plan
Everything went according to plan
```

.....

```
Everything went according to plan
Everything went according to plan
Something went wrong

Standard Error:
The error was using magic numbers
```

可以看到这个脚本的标准输出和标准错误流，同时看到计数器显示是在运行了247次时发生了错误。

1.4 find命令

find 命令中的 -exec 参数是一个非常强大的功能，它允许你对 find 命令查找到的每个文件或目录执行特定的命令。这在自动化任务中非常有用，特别是当你需要对查找到的文件或目录进行一系列操作时。

```
find [path] [conditions] -exec [command] {} ;
```

- path:指定要查找的目录。
- conditions:指定查找条件。
- -exec [command]:指定要对查找到的每个文件执行的命令。
- 花括号:find 查找到的每个文件或目录的占位符。

为了满足我们的命令可以达到，递归地查找文件夹中所有地HTML文件，并且讲他们都压缩成zip文件。而且，要使得即使文件夹中包括空格，命令也能正确执行。
我们设计的命令如下：

```
find . -type f -name "*.html" -exec zip html_files.zip '{}' +
```

执行这个命令就可以将当前目录及其子目录中所有的 .html 文件压缩并且不会忽略文件命中有空格的文件。如图可以看到所有的.html文件都成功压缩包括文件名带有空格的

```
23724@Gavin MINGW64 ~/Desktop/系统工具开发/Shell/find命令
$ find . -type f -name "*.html" -exec zip html_files.zip '{}' +
  adding: 1 2.html (188 bytes security) (stored 0%)
  adding: 1.html (188 bytes security) (stored 0%)
  adding: 2.html (188 bytes security) (stored 0%)
  adding: 3.html (188 bytes security) (stored 0%)
```

1.5 recent脚本

为了达到能够递归的查找文件夹中最近使用的文件，我们在这里创建一个recent.sh文件来进行操作在其中我们加入如下命令：

```
find . -type f -printf '%T@ %p\n' | sort -n | awk '{print $2}'
```

这个指令可以做到打印文件的最后访问时间，打印文件的路径并按照数值排序，从最早访问的文件到最晚访问的文件。我们在这次报告的latex文件夹下使用一次这个脚本。

```
23724@Gavin MINGW64 ~/Desktop/系统工具开发/LaTex/week2
$ ./recent.sh
ouc.png
1.png
2.png
3.png
5.png
6.png
4.png
7.png
main.tex
main.aux
main.out
main.toc
main.fls
main.pdf
main.synctex.gz
main.log
main.fdb_latexmk
recent.sh
```

可以看到我们在这次报告中使用的文件明细.

Chapter 2

编辑器(Vim)

2.1 vimtutor

通过vimtutor我们可以很好地入门vim地使用，在终端直接输入vimtutor即可进入vim的教程中，通过完成教程我们了解到一些关于vim的基础操作
我们可以学习到以下一些内容：

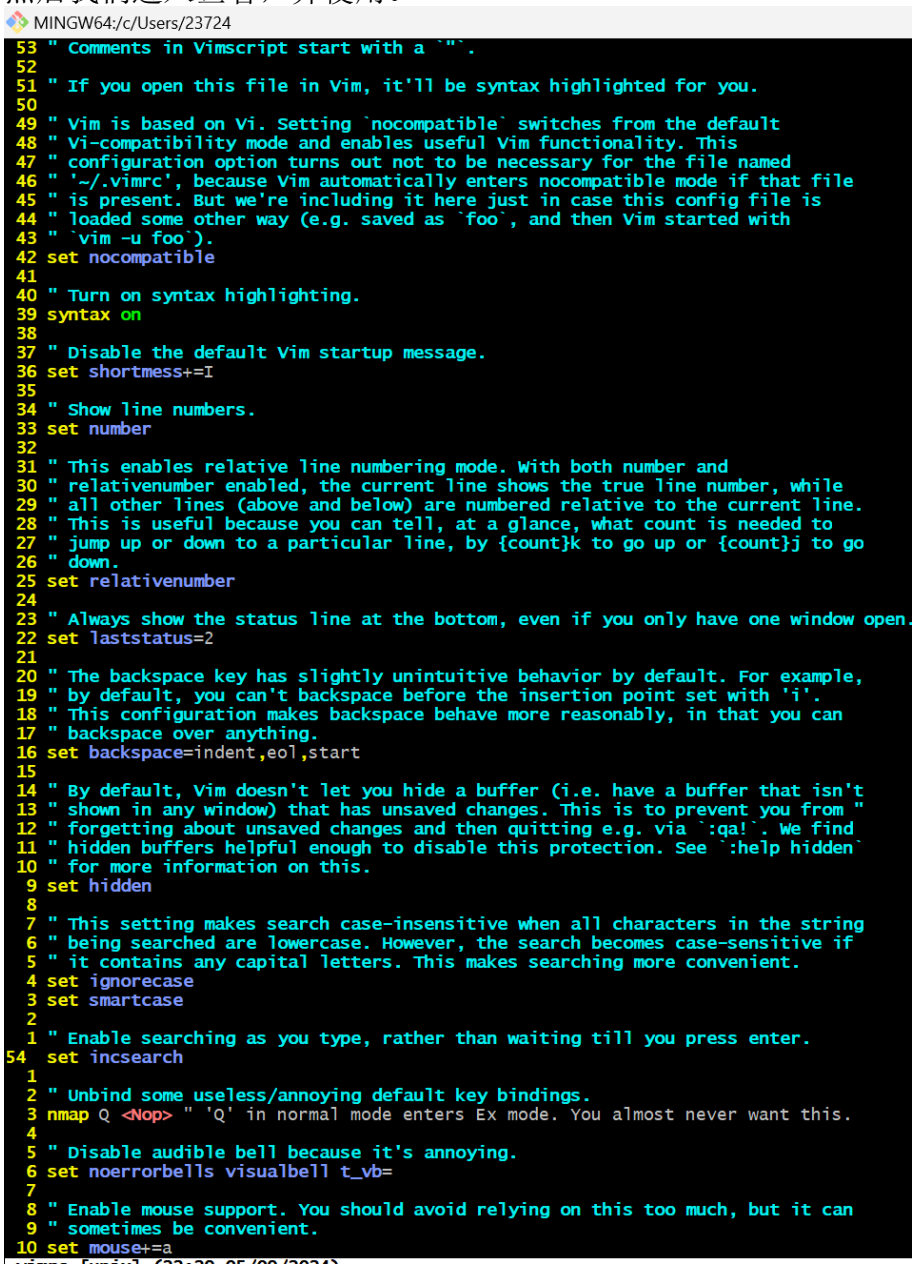
- 进入和退出 Vim
- 光标移动
- 插入和编辑文本
- 搜索和替换
- 复制、剪切和粘贴
- Visual 模式
- 多窗口操作
- 宏

2.2 vimrc区别

我们使用如下指令将你给我的vimrc替换

```
cp /c/Users/23724/Desktop/vimrc ~/.vimrc
```

然后我们进入查看，并使用。



可以发现：

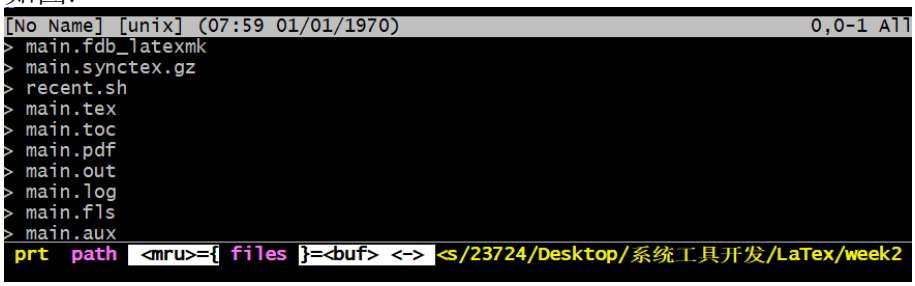
- 启用了语法高亮来根据文件类型进行颜色高亮。
- 在每一行的左侧显示行号
- 可以使用退格键来进行删除
- 进行搜索操作时可以忽略大小写
- 可以使用鼠标来点击文本，切换窗口等操作

2.3 Vim插件

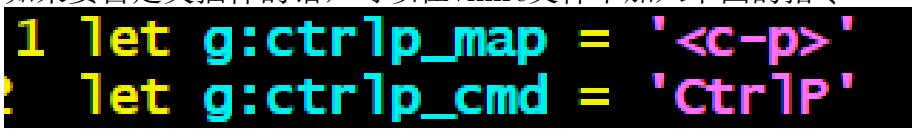
我们在终端先创建一个文件夹用来存放Vim插件，任何从GitHub上克隆ctrlp.Vim插件即以此执行下面的命令：

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

然后我们进入任意一个工程文件夹，输入vim，进入然后就可以通过”CtrlP”这个指令来打开一个文件搜索界面，你可以开始输入文件名来搜索并打开文件。
如图:



如果要自定义插件的话，可以在vimrc文件中加入下图的指令



就可以做到按下Ctrl-P，来使用这个插件。

2.4 进一步修改vimrc和更多插件

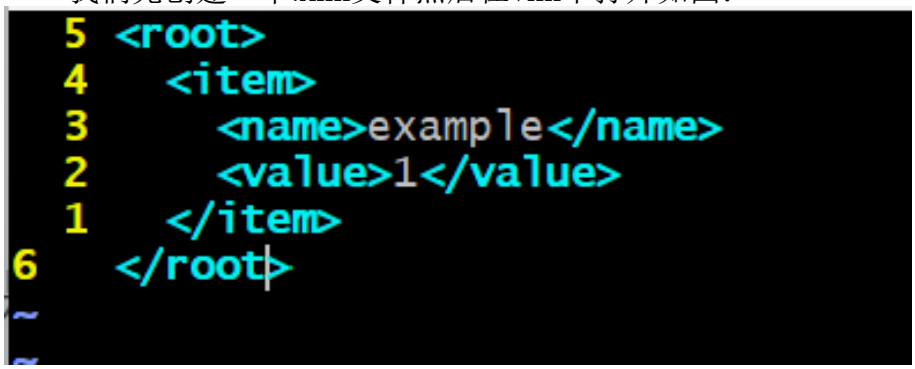
我们可以先安装一个vim-plug用于管理我们的插件然后我修改vimrc文件，加入如下信息

```
call plug#begin('~/.vim/plugged')

Plug 'jiangmiao/auto-pairs'

call plug#end()
```

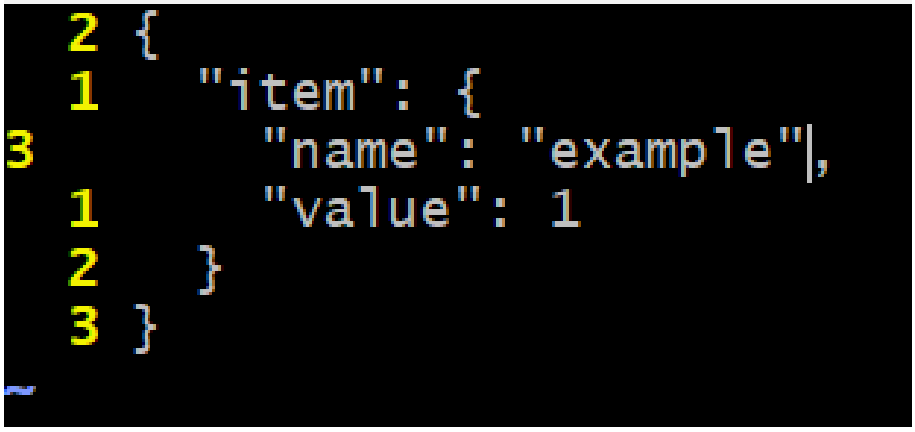
可以成功启用，然后输入:PlugInstall可以按照配置的插件。这里我们加入了自动补齐括号操作。

2.5 宏

我们先创建一个.xml文件然后在vim中打开如图：



然后我们输入qa开始宏的录制；使用指令来将.xml文件转换为JOSN文件，然后输入q结束录制，再输入@a来使用，可以看到文件变化如下图：



成功将 XML 转换到 JSON。

Chapter 3

数据整理

3.1 交互式正则表达式

交互式正则表达式是一种学习、构建和测试正则表达式的工具。
主要有下面一些功能

- 实时匹配
- 语法高亮
- 错误提示
- 替换功能
- 支持多种语言

下面是我在Bash中使用的一些简单的正则表达式

```
grep -E "[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}" file.txt
```

这个可以查找文件中匹配日期的行。

```
sed -E 's/([0-9]{4})-([0-9]{2})-([0-9]{2})/\3-\2-\1/g' file.txt
```

这个可以将文件中的日期格式从 yyyy-mm-dd 转换为 dd-mm-yyyy。

```
grep -E "[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}" file.txt
```

这个指令可以在这个文本文件查找包含电子邮件的行。

这是一个强大的功能。

3.2 统计单词

我在桌面上创建了一个words.txt文件用于我们来演示具体操作。我们可以使用grep和sed命令来处理文本，这些都是强大的搜索工具。

3.2.1 包含至少三个 a 且不以 's 结尾的单词个数

```
grep -i 'a' "C:\Users\23724\Desktop\words.txt"
| grep -v 's$' | grep -iE '(.*a){3}' | wc -l
```

- grep -i 'a': 匹配包含至少一个 'a' 的单词。
- grep -v 's$': 排除以 's' 结尾的单词。
- grep -iE '(.*a)3': 匹配包含至少三个 'a' 的单词。
- wc -l: 统计符合条件的单词数量。

```
23724@Gavin MINGW64 ~/Desktop
$ grep -i 'a' "C:\Users\23724\Desktop\words.txt" | grep -v 's$' | grep -iE '(.*a
){3}' | wc -l
5
```

3.2.2 统计出现频率前三的末尾两个字母组合

```
grep -i 'a' "C:\Users\23724\Desktop\words.txt" | grep -v 's$'
| grep -iE '(.*a){3}' | sed -E 's/[^a-zA-Z]*$//'
| sed -E 's/.*([a-zA-Z]{2})$/\1/' | sort | uniq -c
| sort -nr | head -n 3
```

- 第一个sed：去除行尾非字母字符，如标点符号。
- 第二个sed；提取最后两个字母组合。

```
23724@Gavin MINGW64 ~/Desktop
$ grep -i 'a' "C:\Users\23724\Desktop\words.txt" | grep -v 's$' | grep -iE '(.*a
){3}' | sed -E 's/[^a-zA-Z]*$//' | sed -E 's/.*([a-zA-Z]{2})$/\1/' | sort | uniq
 -c | sort -nr | head -n 3
      1 us
      1 ng
      1 fe
```

3.2.3 统计总共有多少种词尾两字母组合

```
grep -i 'a' /usr/share/dict/words | grep -v 's$'
| grep -iE '(.*a){3}' | sed 's/.*\(..\)$/\1/' | sort | uniq | wc -l
```

这个会统计唯一的词尾两字母组合数量。

```
23724@Gavin MINGW64 ~/Desktop
$ grep -i 'a' "C:\Users\23724\Desktop\words.txt" | grep -v 's$' | grep -iE '(.*a
){3}' | sed 's/.*\(..\)$/\1/' | sort | uniq | wc -l
4
```

3.2.4 找出从未出现过的词尾两字母组合

先生成一个存储所有词尾两字母的.txt文件然后再与我们的文件进行对比

```
23724@Gavin MINGW64 ~/Desktop
$ echo {a..z}{a..z} > all_combinations.txt

23724@Gavin MINGW64 ~/Desktop
$ grep -i 'a' "C:\Users\23724\Desktop\words.txt" | grep -v 's$' | grep -iE '(.*a
){3}' | sed 's/.*\(..\)$/\1/' | sort | uniq > existing_combinations.txt

comm -23 all_combinations.txt existing_combinations.txt
aa ab ac ad ae af ag ah ai aj ak al am an ao ap aq ar as at au av aw ax ay az ba
 bb bc bd be bf bg bh bi bj bk bl bm bn bo bp bq br bs bt bu bv bw bx by bz ca c
b cc cd ce cf cg ch ci cj ck cl cm cn co cp cq cr cs ct cu cv cw cx cy cz da db
dc dd de df dg dh di dj dk dl dm dn do dp dq dr ds dt du dv dw dx dy dz ea eb ec
 ed ee ef eg eh ei ej ek el em en eo ep eq er es et eu ev ew ex ey ez fa fb fc f
d fe ff fg fh fi fj fk fl fm fn fo fp fq fr fs ft fu fv fw fx fy fz ga gb gc gd
ge gf gg gh gi gj gk gl gm gn go gp gq gr gs gt gu gv gw gx gy gz ha hb hc hd he
 hf hg hh hi hj hk hl hm hn ho hp hq hr hs ht hu hv hw hx hy hz ia ib ic id ie i
f ig ih ii ij ik il im in io ip iq ir is it iu iv iw ix iy iz ja jb jc jd je jf
jg jh ji jj jk jl jm jn jo jp jq jr js jt ju jv jw jx jy jz ka kb kc kd ke kf kg
 kh ki kj kk kl km kn ko kp kq kr ks kt ku kv kw kx ky kz la lb lc ld le lf lg l
h li lj lk ll lm ln lo lp lq lr ls lt lu lv lw lx ly lz ma mb mc md me mf mg mh
mi mj mk ml mm mn mo mp mq mr ms mt mu mv mw mx my mz na nb nc nd ne nf ng nh ni
 nj nk nl nm nn no np nq nr ns nt nu nv nw nx ny nz oa ob oc od oe of og oh oi o
j ok ol om on oo op oq or os ot ou ov ow ox oy oz pa pb pc pd pe pf pg ph pi pj
pk pl pm pn po pp pq pr ps pt pu pv pw px py pz qa qb qc qd qe qf qg qh qi qj qk
 ql qm qn qo qp qq qr qs qt qu qv qw qx qy qz ra rb rc rd re rf rg rh ri rj rk r
l rm rn ro rp rq rr rs rt ru rv rw rx ry rz sa sb sc sd se sf sg sh si sj sk sl
sm sn so sp sq sr ss st su sv sw sx sy sz ta tb tc td te tf tg th ti tj tk tl tm
 tn to tp tq tr ts tt tu tv tw tx ty tz ua ub uc ud ue uf ug uh ui uj uk ul um u
n uo up uq ur us ut uu uv uw ux uy uz va vb vc vd ve vf vg vh vi vj vk vl vm vn
vo vp vq vr vs vt vu vv vw vx vy vz wa wb wc wd we wf wg wh wi wj wk wl wm wn wo
 wp wq wr ws wt wu wv ww wx wy wz xa xb xc xd xe xf xg xh xi xj xk xl xm xn xo x
p xq xr xs xt xu xv xw xx xy xz ya yb yc yd ye yf yg yh yi yj yk yl ym yn yo yp
yq yr ys yt yu yv yw yx yy yz za zb zc zd ze zf zg zh zi zj zk zl zm zn zo zp zq
 zr zs zt zu zv zw zx zy zz
```

3.3 原地替代

进行原地替换并不安全，这里有一个例子sed s/REGEX/SUBSTITUTION/ input.txt ¿ input.txt。

这样做可能会有很多的风险：

- 在用¿将输出写回原始文件的时候，可能会遇到文件已经被截断的问题从而导致原始内容丢失。
- 执行这个指令时如果出现错误的话，文件会损坏，导致文本消失。

不只有sed有这个风险，别的工具像grep，perl也会有这样的问题。为了安全起见我们可以使用sed的-i选项来进行原地编辑，这是一种比较安全的做法。

3.4 一些数据整理的常见问题解决

3.4.1 统计文本

我们可以使用wc命令来完成。wc指令的常见参数如下：

- -l:只统计行数
- -w:只统计单词数
- -c:只统计字符数

3.4.2 IP访问次数

我们可以使用grep和wc指令来一起完成这个操作。

```
grep "IP" access.log | wc -l
```

这是一个实例代码grep来获得日志文件中与该IP地址匹配的行，然后使用wc -l来计数行数即访问次数。

3.4.3 常见数据整理

当我们有了一个数据文本需要处理的时候，可以使用awk命令来完成；下面我会给出一些常见的用法。

```
awk '{sum+=$1}END{print"Average:",sum/NR}' numbers.txt
awk '{sum+=$1;sumsq+=$1*$1}END{mean=sum/NR;variance=
(sumsq/NR)-(mean*mean);␣print"Variance:",variance}' numbers.txt
awk '{sum+=$1;sumsq+=$1*$1}END{mean=sum/NR;variance=
(sumsq/NR)-(mean*mean);print"Standard␣Deviation:",sqrt(variance)}'
numbers.txt
```

你可以调用这三个指令来分别完成计算平均数，方差和标准差。

Chapter 4

学习心得

4.1 感受

本周学习了Shell脚本和工具，Vim编辑器和数据整理的一些操作。学习过程中认识到了Shell脚本的强大，可以处理很多的复杂工作，使其自动化，大大提高我们的效率我们可以通过Bash脚本来便于我们对文件进行处理，还能用其来操作文本，管理系统任务。 Shell脚本中的管道功能可以将很多命令串联在一起完成复杂的操作，这是学习的重难点。初试以后我已经感受到了Shell脚本的强大。 Vim编辑器是一个功能非常强大的编辑器，还能够添加多种插件来提高我们的效率，这种编辑模式尽可能地让我们能够通过键盘完成所有的操作，来提高效率，让我们在代码编辑和文本处理上更加得心应手。宏命令是学习的重难点，可以帮助我们处理重复性任务。数据整理是一个很困难的学习项目，它的命令十分不好编写，这还只是简单的数据处理，还需要我们来编写脚本实现数据的自动化处理，这样才能发挥数据处理的作用。

4.2 学习中遇到的困难

4.2.1 Shell脚本

在学习Shell脚本时，发现管道是一个很强大的工具，它将将一个命令的输出作为下一个命令的输入来使用。看起来简单但是还有很多难点，自己编写尝试的命令组合经常不符合逻辑而导致出错。说明对命令的输出格式和下个命令的输入格式理解不到位，学习中也通过多次调试等来一步步纠正。

4.2.2 Vim

Vim使用过程中在宏的运用上遇到问题，经常在录制过程中出错，还是说明对Vim的操作不够熟悉。只能通过多用来解决问题。

4.2.3 数据处理

数据处理个人感觉十分抽象，正则表达式也很难于读和编写，目前没有什么好的解决办法。

Chapter 5

参考资料

- Missing Semester CN
- Google
- Wikipedia
- Missing Semester GitHub Repository
- 正则表达式

如果想要查看相关实例练习的源代码可以查询我的GitHub仓库
GitHub Repository