

系统开发工具基础

Shell & Vim & 数据整理

王志巍 22020007161

2024 - 08 - 30

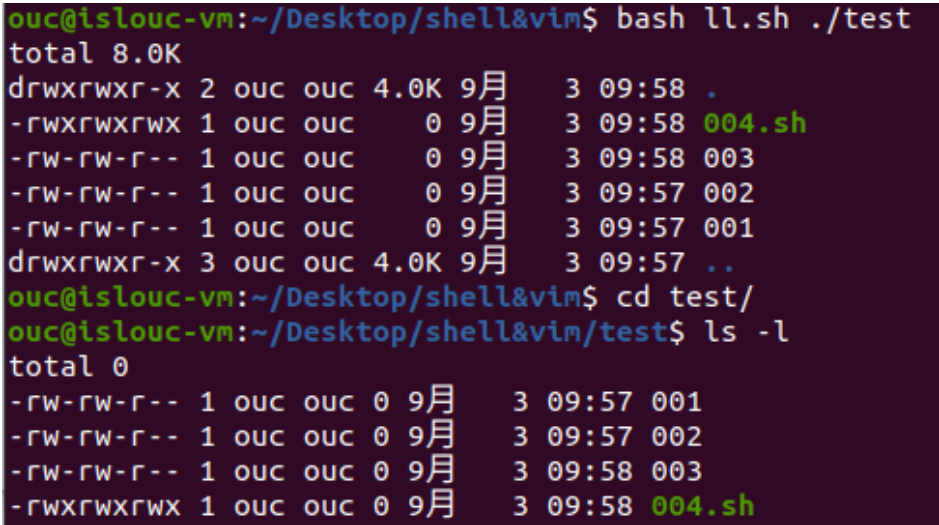
目录

1	Shell	1
1.1	ls -l	1
1.2	marco & polo	1
1.3	运行脚本直到它出错	2
1.4	find -exec	2
1.4.1	使用-exec gzip 压缩	2
1.4.2	用递归压缩一个目录下所有 html 文件	3
1.4.3	递归查找最近使用的文件	4
2	Vim	4
2.1	查看帮助手册	4
2.2	Vim 基础操作	5
2.2.1	常用模式切换	5
2.2.2	复制	5
2.2.3	黏贴	6
2.2.4	剪切	6
2.3	安装和配置 ctrlp.vim	7
2.3.1	下载	7
2.3.2	配置	7
2.3.3	使用	7
3	数据整理	7
3.1	引入	7
3.2	sed 正则表达式	8
3.2.1	统计	8
3.2.2	替换	9
4	实验心得	9
5	Github 链接	9

1 Shell

1.1 ls -l

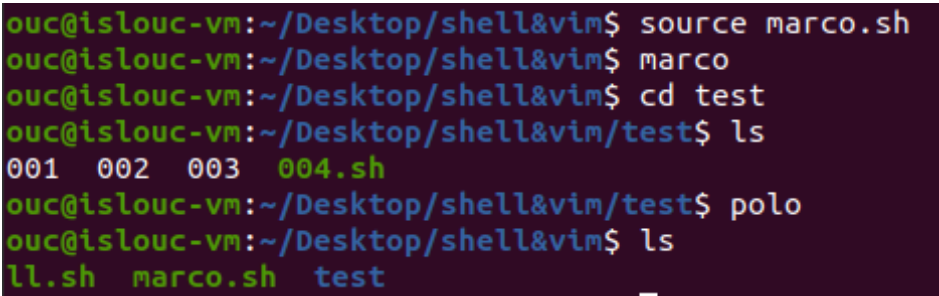
```
#!/bin/bash
relative_path=$1          #$1是第一个参数，第零个参数是脚本自身
if [ -z "$relative_path" ]; then
    echo "路径不能为空"
    exit 1
fi
ls -lah --time=atime --sort=time --color=auto "$relative_path"
```



1.2 marco & polo

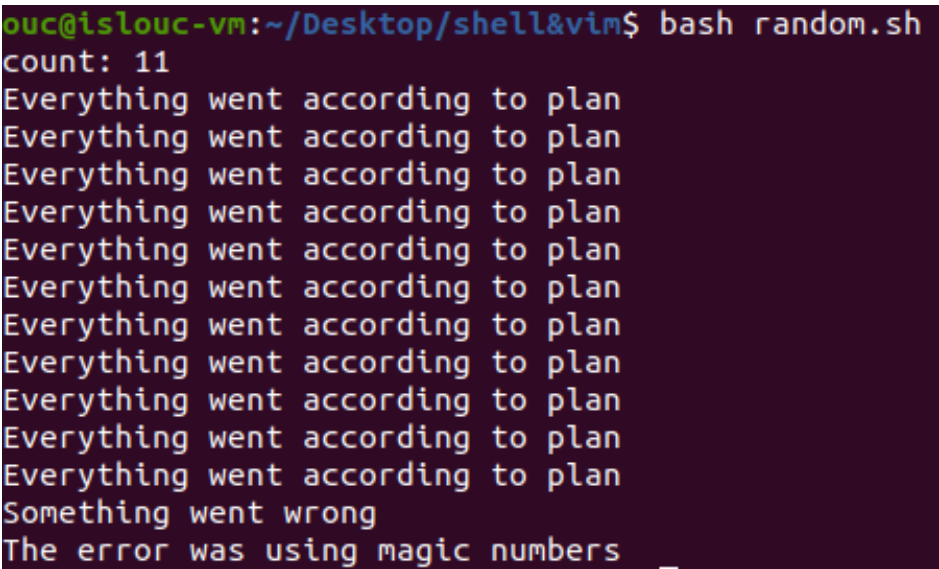
```
#执行 marco 时，当前的工作目录应当以某种形式保存
#当执行 polo 时，无论现在处在什么目录下，
#都应当 cd 回到当时执行 marco 的目录
```

```
#!/bin/bash
saved_dir=""
marco() {
    saved_dir=$(pwd)
}
polo() {
    if [ -z "$saved_dir" ]; then
        echo "空目录"
    else
        cd "$saved_dir" || echo "error"
    fi
}
#order1 || order2
#order1 失败则执行 order2
```



1.3 运行脚本直到它出错

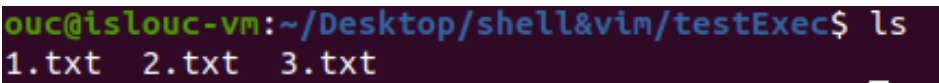
```
#!/usr/bin/env bash
#初始化日志
log_file="script_output.log"
> "$log_file"
#要运行的脚本
script_content='
n=$(( RANDOM % 100 ))
if [[ n -eq 42 ]]; then
    echo "Something went wrong"
    >&2 echo "The error was using magic numbers"
    exit 1
fi
echo "Everything went according to plan"
,
#计数
count=0
while true; do
    bash -c "$script_content" >> "$log_file" 2>&1
    if [[ $? -ne 0 ]]; then
        break
    fi
    count=$((count + 1))
done
echo "count: $count"
cat "$log_file"
```



```
ouc@islouc-vm:~/Desktop/shell&vim$ bash random.sh
count: 11
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
Everything went according to plan
Something went wrong
The error was using magic numbers
```

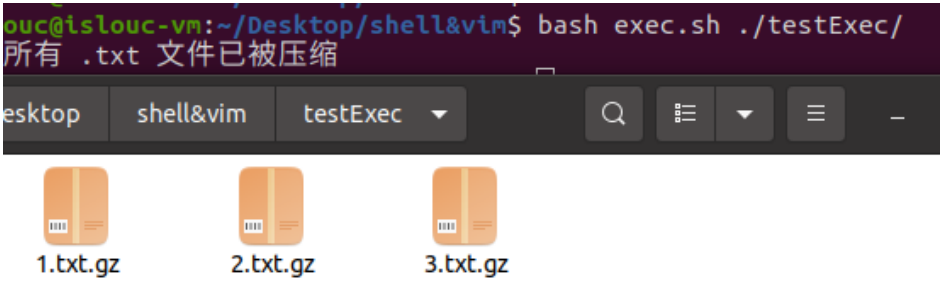
1.4 find -exec

1.4.1 使用-exec gzip 压缩



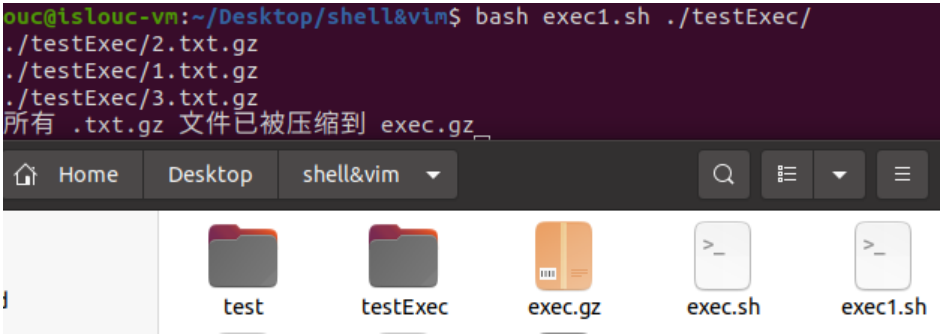
```
ouc@islouc-vm:~/Desktop/shell&vim/testExec$ ls
1.txt 2.txt 3.txt
```

```
#!/bin/bash
relative_path=$1
find "$relative_path" -type f -name "*.txt" -exec gzip {} \;
echo "所有 .txt 文件已被压缩"
```



运行脚本后发现，它将每个“.txt”文件分别压缩了，但我希望他们被压缩为一个压缩文件。故修改代码。

```
#!/bin/bash
relative_path=$1
archive_name="exec.gz"
find "$relative_path" -type f -name "*.txt.gz" -print0 |
tar -czvf "$archive_name" --null -T -
echo "所有 .txt.gz 文件已被压缩到 $archive_name"
#print0 用空字符分隔文件名，用以处理文件名中包含空格或特殊字符的情况
```



1.4.2 用递归压缩一个目录下所有 html 文件

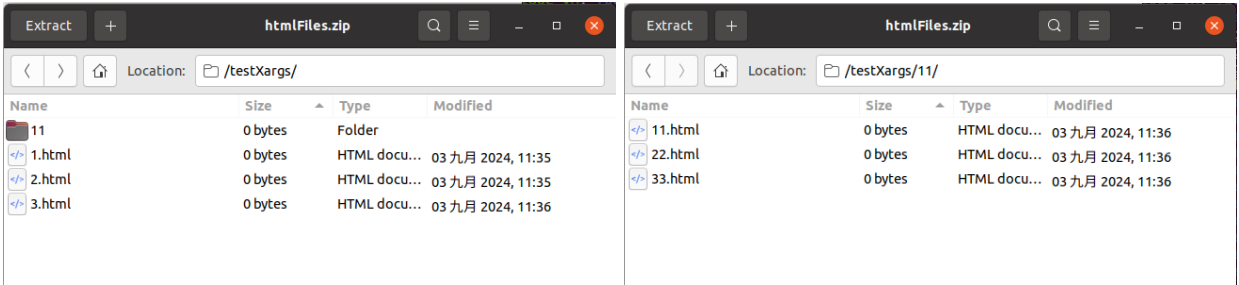
编写一个命令，它可以递归地查找文件夹中所有的 HTML 文件，并将它们压缩成 zip 文件。注意，即使文件名中包含空格，您的命令也应该能够正确执行



建立嵌套目录，每个目录下都有 html 文件

```
#!/bin/bash
relative_path=$1
zip_name="htmlFiles.zip"
find "$relative_path" -type f -name "*.html" -print0 |
xargs -0 zip "$zip_name"
echo "所有 .html 文件已被压缩到 $zip_name"
```

```
ouc@islouc-vm:~/Desktop/shell&vim$ bash html2zip.sh ./testXargs/
adding: testXargs/11/11.html (stored 0%)
adding: testXargs/11/33.html (stored 0%)
adding: testXargs/11/22.html (stored 0%)
adding: testXargs/3.html (stored 0%)
adding: testXargs/1.html (stored 0%)
adding: testXargs/2.html (stored 0%)
所有 .html 文件已被压缩到 htmlFiles.zip
```



压缩后压缩文件内部目录

1.4.3 递归查找最近使用的文件

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。更通用的做法，你可以按照最近的使用时间列出文件吗？

```
#!/bin/bash
relative_path=$1
find "$relative_path" -type f -exec ls -ltu {} +

#ls -ltu 列出文件信息并按访问时间排序
#+表示以批处理的方式执行命令
```

```
ouc@islouc-vm:~/Desktop/shell&vim$ bash fetch.sh .
-rw-rw-r-- 1 ouc ouc 78 9月 3 12:05 ./fetch.sh
-rw-rw-r-- 1 ouc ouc 1006 9月 3 11:48 ./htmlFiles.zip
-rw-rw-r-- 1 ouc ouc 186 9月 3 11:46 ./html2zip.sh
-rw-rw-r-- 1 ouc ouc 0 9月 3 11:45 ./testXargs/11/11.html
-rw-rw-r-- 1 ouc ouc 0 9月 3 11:45 ./testXargs/11/22.html
-rw-rw-r-- 1 ouc ouc 0 9月 3 11:45 ./testXargs/11/33.html
-rw-rw-r-- 1 ouc ouc 0 9月 3 11:45 ./testXargs/1.html
-rw-rw-r-- 1 ouc ouc 0 9月 3 11:45 ./testXargs/2.html
-rw-rw-r-- 1 ouc ouc 0 9月 3 11:45 ./testXargs/3.html
-rw-rw-r-- 1 ouc ouc 195 9月 3 11:24 ./exec.gz
-rw-rw-r-- 1 ouc ouc 205 9月 3 11:23 ./exec1.sh
-rw-rw-r-- 1 ouc ouc 26 9月 3 11:21 ./testExec/1.txt.gz
-rw-rw-r-- 1 ouc ouc 26 9月 3 11:21 ./testExec/2.txt.gz
-rw-rw-r-- 1 ouc ouc 26 9月 3 11:21 ./testExec/3.txt.gz
-rw-rw-r-- 1 ouc ouc 128 9月 3 11:06 ./exec.sh
-rw-rw-r-- 1 ouc ouc 429 9月 3 10:42 ./script_output.log
-rw-rw-r-- 1 ouc ouc 450 9月 3 10:42 ./random.sh
-rwxrwxrwx 1 ouc ouc 346 9月 3 10:10 ./marco.sh
-rwxrwxrwx 1 ouc ouc 168 9月 3 09:58 ./ll.sh
-rwxrwxrwx 1 ouc ouc 0 9月 3 09:58 ./test/004.sh
-rw-rw-r-- 1 ouc ouc 0 9月 3 09:58 ./test/003
-rw-rw-r-- 1 ouc ouc 0 9月 3 09:57 ./test/002
-rw-rw-r-- 1 ouc ouc 0 9月 3 09:57 ./test/001
```

2 Vim

2.1 查看帮助手册

```
vimtutor
```

```
Lesson 1.1:  MOVING THE CURSOR

** To move the cursor, press the h,j,k,l keys as indicated. **
      ^
      k
    < h   l >      Hint:  The h key is at the left and moves left.
      j          The l key is at the right and moves right.
      v          The j key looks like a down arrow.

1. Move the cursor around the screen until you are comfortable.

2. Hold down the down key (j) until it repeats.
   Now you know how to move to the next lesson.

3. Using the down key, move to lesson 1.2.

NOTE: If you are ever unsure about something you typed, press <ESC> to place
      you in Normal mode.  Then retype the command you wanted.

NOTE: The cursor keys should also work.  But using hjkl you will be able to
      move around much faster, once you get used to it.  Really!
```

2.2 Vim 基础操作

2.2.1 常用模式切换

一般模式

在插入模式下按 “ESC” ， 进入一般模式	
h：	向左移动光标
j：	向下移动光标
k：	向上移动光标
l：	向右移动光标
dd：	删除当前行
u：	撤销上一个操作。
Ctrl + r：	重做上一个撤销的操作

插入模式

在一般模式下， 按	
i：	在光标前插入
a：	在光标后插入
o：	在当前行下方插入新行
O：	在当前行上方插入新行

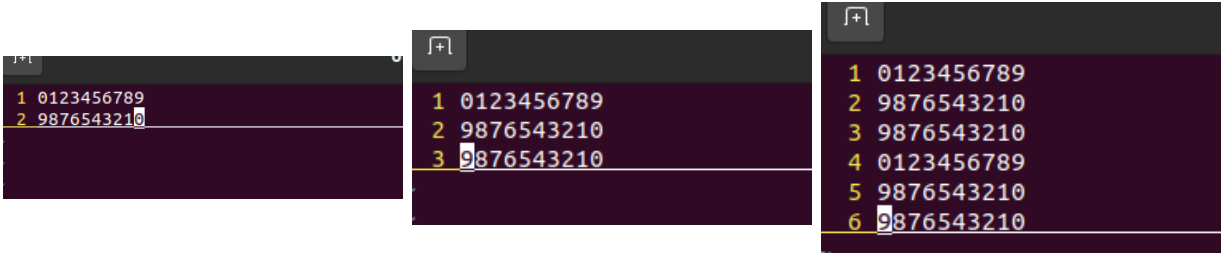
命令模式

在一般模式下按 “： ” ， 进入命令模式	
:w	保存文件。
:q	退出Vim。
:wq & :x	保存并退出Vim。
:q!	强制退出Vim， 不保存更改。
:e <filename>	打开文件。
:set number	显示行号。
:set nonumber	隐藏行号。

2.2.2 复制

按行复制

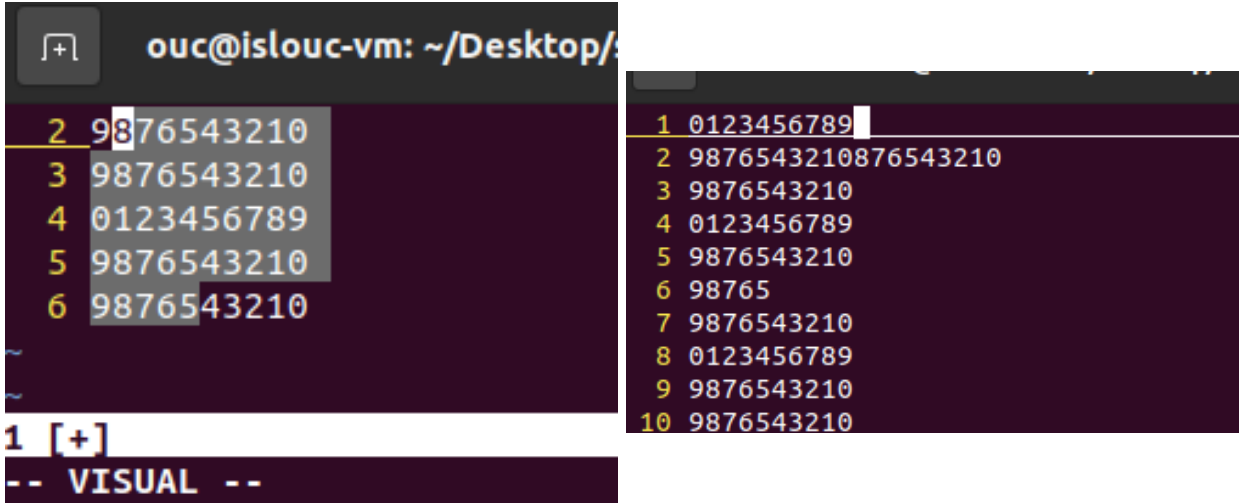
yy	复制当前行
Nyy, yN	复制从当前行开始的N行， N为数字



图：按行复制

选取复制

step1	按v进入字符可视模式
step2	用h,j,k,l选取要复制的文本
step3	按y复制



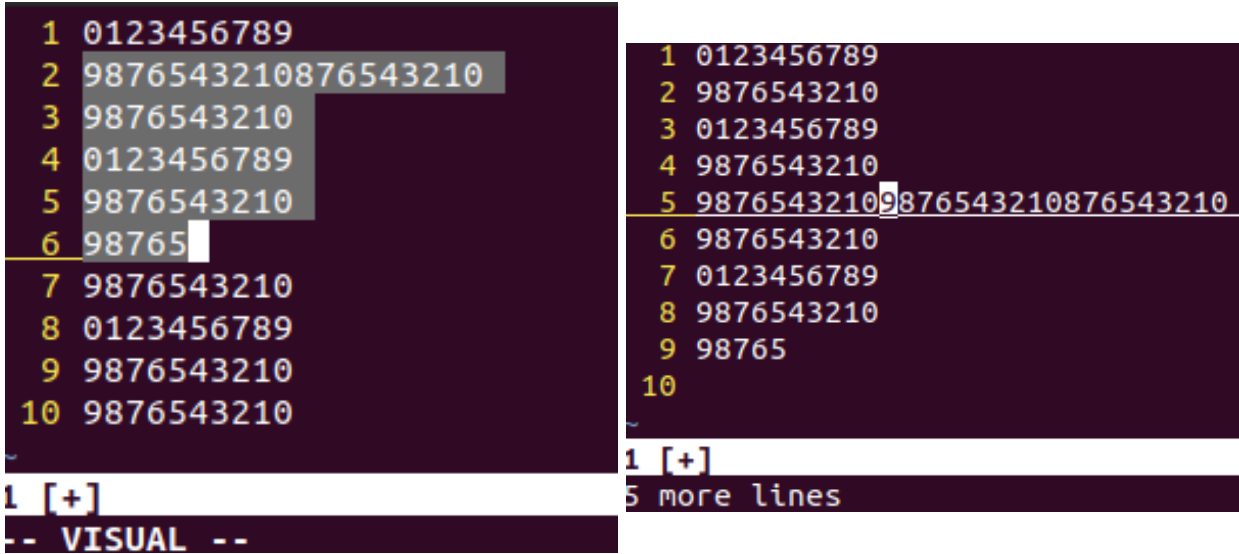
图：选取复制

2.2.3 黏贴

p	粘贴复制或剪切的内容到当前光标位置后
P	粘贴复制或剪切的内容到当前光标位置前

2.2.4 剪切

dd	剪切当前行。
Ndd, dN	剪切从当前行开始的N行。
选取剪切也同样，进入字符可视模式选取，按d剪切	



图：选取剪切

2.3 安装和配置 ctrlp.vim

2.3.1 下载

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

```
ouc@islouc-vm:~/Desktop$ mkdir -p ~/.vim/pack/vendor/start
ouc@islouc-vm:~/Desktop$ cd ~/.vim/pack/vendor/start
ouc@islouc-vm:~/vim/pack/vendor/start$ git clone https://github.com/ctrlpvim/ctrlp.vim
Cloning into 'ctrlp.vim'...
remote: Enumerating objects: 4299, done.
remote: Counting objects: 100% (168/168), done.
remote: Compressing objects: 100% (101/101), done.
remote: Total 4299 (delta 71), reused 151 (delta 66), pack-reused 4131 (from 1)
Receiving objects: 100% (4299/4299), 1.70 MiB | 3.08 MiB/s, done.
Resolving deltas: 100% (1661/1661), done.
```

2.3.2 配置

```
vim ~/.vimrc
```

```
set runtimepath^=~/.vim/bundle/ctrlp.vim

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

```
39 set runtimepath^=~/.vim/bundle/ctrlp.vim
40
41 let g:ctrlp_map = '<c-p>'
42 let g:ctrlp_cmd = 'CtrlP'
43 let g:ctrlp_working_path_mode = 'ra'
44
~/vimrc [+]
-- INSERT --
```

2.3.3 使用

CtrlP 是一个强大的 Vim 插件，用于快速在项目文件夹中定位和打开文件。通过自定义快捷键，如文档中的“Ctrl + P”，用户可以在当前路径及其子目录中高效搜索文件，极大地提升了文件导航和编辑的效率。

```
cd <项目目录>
vim
Ctrl + P
```

```
1 1,1 All
> sub/sub1
> sub/sub2
> 2.txt
> 3.txt
prt path <mru>={ files }=<buf> <-> /home/ouc/Desktop/shell&vim/vims
>>>
```

3 数据整理

3.1 引入


```
ouc@islouc-vm:~/Desktop$ journalctl | grep -i intel | tail -n 5
9月 03 14:42:46 islouc-vm kernel: ACPI: FACP 0x00000000BFEFEE73 0000F4 (v04 INTEL 440BX
9月 03 14:42:46 islouc-vm kernel: smpboot: CPU0: Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz
9月 03 14:42:46 islouc-vm kernel: agpgart-intel 0000:00:00.0: Intel 440BX Chipset
9月 03 14:42:46 islouc-vm kernel: agpgart-intel 0000:00:00.0: ACP aperture is 256M @ 0x0
9月 03 14:42:46 islouc-vm kernel: intel_pstate: CPU model not supported
```

```
journalctl | grep -i intel | tail -n 5
```

这条代码将全部系统日志转换为了仅包含 intel 的日志，并且输出最近的 5 条。这就是一种简单的数据整理。大多数情况下，数据整理需要明确哪些工具可以被用来达成特定数据整理的目的，并且明白如何组合使用这些工具。

3.2 sed 正则表达式

3.2.1 统计

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。

```
grep -i 'a.*a.*a' /usr/share/dict/words |
grep -vi "'s$" | wc -l
#wc -l 计算文件行数
```

```
ouc@islouc-vm:~/Desktop$ grep -i 'a.*a.*a' /usr/share/dict/words | grep -vi "'s$" | wc -l
847
```

这些单词中，出现频率前三的末尾两个字母是什么？

```
grep -i 'a.*a.*a' /usr/share/dict/words | grep -vi "'s$" |
sed 's/.*\(..\)$/\1/' | tr 'A-Z' 'a-z' | sort |
uniq -c | sort -nr | head -n 3
#sort | uniq 用来去重，因为uniq只能识别相邻行，所以要先按字母序排序
```

```
ouc@islouc-vm:~/Desktop$ grep -i 'a.*a.*a' /usr/share/dict/words | grep -vi "'s$" | sed 's/.*\(..\)$/\1/' | tr 'A-Z' 'a-z' | sort | uniq -c | sort -nr | head -n 3
101 an
63 ns
54 as
```

共存在多少种词尾两字母组合？

```
grep -i 'a.*a.*a' /usr/share/dict/words |
grep -vi "'s$" | sed 's/.*\(..\)$/\1/' |
tr 'A-Z' 'a-z' | sort | uniq | wc -l
```

```
ouc@islouc-vm:~/Desktop$ grep -i 'a.*a.*a' /usr/share/dict/words | grep -vi "'s$" | sed 's/.*\(..\)$/\1/' | tr 'A-Z' 'a-z' | sort | uniq | wc -l
111
```

哪个组合从未出现过？

```
echo {a..z}{a..z} | tr ' ' '\n' > all

grep -i 'a.*a.*a' /usr/share/dict/words |
grep -vi "'s$" | sed 's/.*\(..\)$/\1/' |
tr 'A-Z' 'a-z' | sort | uniq > used

comm -23 all used > unused
paste -sd, unused | sed 's/,/, /g'
```

```
ouc@islouc-vm:~/Desktop/shell&vim$ paste -sd, unused | sed 's/./, /g'
ab, af, ai, aj, ao, ap, aq, au, av, aw, bb, bc, bd, be, bf, bg, bh, bi, bj, bk, bl, bm, bn, bp, bq, br, bs, bt, bu, bv, bw, bx, by, bz, cb
, cc, cd, cf, cg, ch, ci, cj, cl, cm, cn, co, cp, cq, cr, cu, cv, cw, cx, cy, cz, db, dc, dd, df, dg, dh, di, dj, dk, dl, dm, dn, dp, dq,
dr, dt, du, dv, dw, dx, dy, dz, eb, ec, ef, eg, eh, ei, ej, ek, el, em, en, eo, ep, eq, et, eu, ev, ew, ex, ey, ez, fb, fc, fd, fe, ff, fg
, fh, fi, fj, fk, fl, fm, fn, fo, fp, fq, fr, fs, fu, fv, fw, fx, fy, fz, gb, gc, gd, gf, gg, gh, gi, gj, gk, gl, gm, gn, go, gp, qq, gr,
gt, gu, gv, gw, gx, gy, gz, hb, hc, hd, hf, hg, hh, hi, hj, hk, hl, hm, hn, hp, hq, hr, ht, hu, hv, hw, hx, hy, hz, ib, id, if, ig, ih, ii
, ij, ik, il, im, iq, ir, iu, iv, iw, ix, iy, iz, 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, kb, kc, kd, ke, kf, kg, kh, kj, kk, kl, km, kn, ko, kp, kq, kr, kt, ku, kv, kw, kx, ky, kz, lb, lc, ld, lf, lg, lh, li, lj, lk
, ll, lm, ln, lo, lp, lq, lr, lt, lu, lv, lw, lx, lz, mb, mc, md, mf, mg, mh, mi, nj, mk, ml, mn, mo, mp, mq, mr, mt, mu, mv, mw, mx, my,
mz, nb, nc, nf, nh, nj, nk, nl, nm, nn, no, np, nq, nr, nu, nv, nw, nx, ny, nz, ob, oc, od, oe, of, og, oh, oi, oj, ok, ol, om, op, oq, ot
, ou, ow, ox, oy, oz, pb, pc, pd, pe, pf, pg, pi, pj, pk, pl, pm, pn, po, pp, pq, pr, pt, pu, pv, pw, px, 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, rb, rc, rf, rg, rh, ri, rl, rm, rn, ro, rp, rq, rr, rt, ru, rv
, rw, rx, rz, sb, sc, sd, sf, sg, sj, sk, sl, sn, sp, sq, sr, ss, sv, sw, sx, sz, tb, tc, td, tf, tg, ti, tj, tk, tl, tm, tn, to, tp, tq,
tr, tu, tv, tw, tx, tz, ub, uc, ud, uf, ug, uh, ui, uj, uk, ul, um, un, uo, up, uq, ur, uu, uv, uw, ux, uy, uz, vb, vc, vd, ve, vf, vg, vh
, vi, vj, vk, vl, vm, vn, vp, vq, vr, vs, vt, vu, vv, vw, vx, vy, vz, wb, wc, wd, we, wf, wg, wh, wi, wj, wk, wl, wm, wn, wo, wp, wq, wr,
ws, wt, wu, ww, wx, wy, wz, xa, xb, xc, xd, xe, xf, xg, xh, xi, xj, xk, xl, xm, xn, xo, xp, xq, xr, xs, xt, xu, xv, xw, xx, xy, xz, yb
, yc, yd, ye, yf, yg, yh, yi, yj, yk, yl, ym, yn, yo, yp, yq, yr, yt, yu, yv, yw, yx, yy, yz, zb, zc, zd, ze, zf, zg, zh, zj, zk, zl, zm,
zn, zo, zp, zq, zr, zs, zt, zu, zv, zw, zx, zy, zz
```

3.2.2 替换

```
sed s/REGEX/SUBSTITUTION/ file > file
```

这条指令看上去可以将文件中的字符原地替换，但这是无法实现的。通过 man sed 指令，我们可以知道,” sed ***** file1 > file2 “指令会先将 file2 清空，因此这条指令不仅不能原地替换字符，反而会清空原本的文件。

```
sed -i.bak s/REGEX/SUBSTITUTION/ file
```

修改后，这条指令可以在进行原地替换的同时，生成一个.bak 备份文件。

```
ouc@islouc-vm:~/Desktop/shell&vim$ vim sebSwap
ouc@islouc-vm:~/Desktop/shell&vim$ cat sebSwap
hello world
Hello world

ouc@islouc-vm:~/Desktop/shell&vim$ sed -i.bak 's/hello/hi/' sebSwap
ouc@islouc-vm:~/Desktop/shell&vim$ cat sebSwap
hi world
Hello world
ouc@islouc-vm:~/Desktop/shell&vim$ cat sebSwap.bak
hello world
Hello world
```

从结果可以看出，seb 指令是严格匹配,” hello “被替换为了” hi “，而” Hello “没有。而 grep 指令的筛选无关大小写。

4 实验心得

本次实验学习了 Shell 脚本、Vim 文本编辑器、使用 seb 正则表达式的数据整理相关的知识。通过本次实验学会了轻量的 Shell 脚本语言。在写 Shell 脚本的同时，也对 Vim 的使用进行了大量的练习。虽然在文本编辑时，仍然感觉效率上，不如高图形化的 Windows，不过我认为更多的是熟练度的问题，Vim 光标的精确移动是平时使用鼠标时难以达到的。此外设计了许多含有 seb 正则表达式的指令，对 Linux 系统的 words 文件中的数据进行整理。过程中，管道符号” | “对指令的设计帮助很大，减去了很多中间文件。

5 Github 链接

```
https://github.com/Starry-Sky-OUC/gitTest
```