

系统工具开发基础实验

曹瑜 22020007007

September 4, 2024

1 实验内容

Shell 工具和脚本
编辑器 (vim)
数据整理

2 实验目的

(1) 通过实践掌握 Shell 的基本操作，包括命令行界面的使用、文件系统的导航、文件的创建、编辑与删除等基本命令，以及环境变量的设置与管理。学习 Shell 脚本的语法、变量、条件判断、循环控制、函数定义等编程基础，能够编写简单的脚本完成特定任务。这不仅能提高个人工作效率，也是 Linux 系统管理的基础技能之一。

(2) 熟悉 Vim 编辑器的启动、退出以及三种基本模式（命令模式、插入模式、底行模式）之间的切换。掌握在 Vim 中进行文本编辑的基本操作，如光标移动、文本删除、复制粘贴等。通过实验，能够学习 Vim 的高级功能，如搜索、替换、多文件编辑、代码折叠、语法高亮等，提升编辑效率和编程体验。

(3) 学习将某种格式存储的数据转换成另外一种格式

3 课上实验

3.1 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）：-a

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）：-h

文件以最近访问顺序排序：-t

以彩色文本显示输出结果-color=auto

```

ouc@islouc-vn:~/Desktop$ man ls
ouc@islouc-vn:~/Desktop$ ls -a
.
..
cachegrind.out.18174
callgrind.out.18207
cryp.py
Ghidra.desktop
ph
'sql-labs-mysql.sh'
systemtool
test
test1.obj
test2.obj
test3.obj
test_all_protection.obj
test.c
test_char_protection.obj
test_no_protection.obj
test.obj
ouc@islouc-vn:~/Desktop$ ls -h
cachegrind.out.18174
callgrind.out.18207
cryp.py
Ghidra.desktop
ph
'sql-labs-mysql.sh'
systemtool
test
test1.obj
test2.obj
test3.obj
test_all_protection.obj
test.c
test_char_protection.obj
test_no_protection.obj
test.obj
ouc@islouc-vn:~/Desktop$ ls -t
systemtool
test
cryp.py
test1.obj
test2.obj
test3.obj
test_all_protection.obj
test.c
test_char_protection.obj
test_no_protection.obj
test.obj
ph.c
test.obj
ph
ouc@islouc-vn:~/Desktop$ ls --color=auto
cachegrind.out.18174
callgrind.out.18207
cryp.py
Ghidra.desktop
ph
'sql-labs-mysql.sh'
systemtool
test
test1.obj
test2.obj
test3.obj
test_all_protection.obj
test.c
test_char_protection.obj
test_no_protection.obj
test.obj

```

Figure 1: 3.1 ls 命令

3.2 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。

```

1 marco(){
2     echo "$(pwd)" > $HOME/marco_history.log
3     echo "save pwd $(pwd)"
4 }
5 polo(){
6     cd "$(cat "$HOME/marco_history.log")"
7 }
8
~
~

```

Figure 2: 3.2 bash 函数

```

ouc@islouc-vm:~/Desktop$ vim marco.sh
ouc@islouc-vm:~/Desktop$ source marco.sh
ouc@islouc-vm:~/Desktop$ marco
save pwd /home/ouc/Desktop
ouc@islouc-vm:~/Desktop$ cd
ouc@islouc-vm:~$ polo

```

Figure 3: 3.2 执行截图

3.3 创建一个 zip 压缩文件

3.3.1 创建所需的文件

```

mkdir html_root
cd html_root
touch {1..10}.html
mkdir html
cd html
touch xxxx.html

```

```

ouc@islouc-vm:~/Desktop/systemtool/html_root$ mkdir html_root
ouc@islouc-vm:~/Desktop/systemtool/html_root$ cd html_root
ouc@islouc-vm:~/Desktop/systemtool/html_root/html_root$ touch {1..10}.html
ouc@islouc-vm:~/Desktop/systemtool/html_root/html_root$ mkdir html
ouc@islouc-vm:~/Desktop/systemtool/html_root/html_root$ cd html
ouc@islouc-vm:~/Desktop/systemtool/html_root/html_root/html$ touch xxxx.html
ouc@islouc-vm:~/Desktop/systemtool/html_root/html_root/html$ ls
xxxx.html
ouc@islouc-vm:~/Desktop/systemtool/html_root/html_root/html$ cd ..
ouc@islouc-vm:~/Desktop/systemtool/html_root/html_root$ ls
10.html 1.html 2.html 3.html 4.html 5.html 6.html 7.html 8.html 9.html html
ouc@islouc-vm:~/Desktop/systemtool/html_root/html_root$

```

Figure 4: 3.3.1 创建文件

3.3.2 执行 find 命令

```

find . -type f -name "*.html" | xargs -d '\n' tar -cvzf html.zip

```

```

ouc@islouc-vm:~/Desktop/systemtool/html_root$ find . -type f -name "*.html" | xargs -d '\n' tar -cvzf html.zip
./8.html
./5.html
./10.html
./html/xxxx.html
./3.html
./1.html
./6.html
./html_root/8.html
./html_root/5.html
./html_root/10.html
./html_root/html/xxxx.html
./html_root/3.html
./html_root/1.html
./html_root/6.html
./html_root/4.html
./html_root/2.html
./html_root/9.html
./html_root/7.html
./4.html
./2.html
./9.html
./7.html

```

Figure 5: 3.3.2 执行 find 命令

3.4 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

```
find . -type f -print0 | xargs -0 ls -lt | head -1
```

```

ouc@islouc-vm:~/Desktop/systemtool/html_root$ find . -type f -print0 | xargs -0 ls -lt | head
-rw-rw-r-- 1 ouc ouc 357 8月 30 09:54 ./html.zip
-rw-rw-r-- 1 ouc ouc  0 8月 30 09:47 ./html_root/html/xxxx.html
-rw-rw-r-- 1 ouc ouc  0 8月 30 09:47 ./html_root/10.html
-rw-rw-r-- 1 ouc ouc  0 8月 30 09:47 ./html_root/1.html
-rw-rw-r-- 1 ouc ouc  0 8月 30 09:47 ./html_root/2.html
-rw-rw-r-- 1 ouc ouc  0 8月 30 09:47 ./html_root/3.html
-rw-rw-r-- 1 ouc ouc  0 8月 30 09:47 ./html_root/4.html
-rw-rw-r-- 1 ouc ouc  0 8月 30 09:47 ./html_root/5.html
-rw-rw-r-- 1 ouc ouc  0 8月 30 09:47 ./html_root/6.html
-rw-rw-r-- 1 ouc ouc  0 8月 30 09:47 ./html_root/7.html
ouc@islouc-vm:~/Desktop/systemtool/html_root$ find . -type f -print0 | xargs -0 ls -lt | head -1
-rw-rw-r-- 1 ouc ouc 357 8月 30 09:54 ./html.zip

```

Figure 6: 3.4 查找文件夹中最近使用的文件

4 课后实验

4.1 Shell 工具和脚本

4.1.1 基本的 Shell 脚本

保存文件、用 `chmod +x` 命令赋予文件执行权限、运行脚本

```
#!/bin/bash
1 echo "Hello World!"
```

Figure 7: 4.1.1 bash 函数

```
ouc@islouc-vm:~/Desktop/systemtool/shell$ vim hello_world.sh
ouc@islouc-vm:~/Desktop/systemtool/shell$ chmod +x hello_world.sh
ouc@islouc-vm:~/Desktop/systemtool/shell$ ./hello_world.sh
Hello World!
ouc@islouc-vm:~/Desktop/systemtool/shell$
```

Figure 8: 4.1.1 运行截图

4.1.2 文件处理

功能：统计当前目录中的普通文件数目

```
2 #!/bin/bash
1 echo "Counting files in the current directory:"
3 ls -l | grep ^- | wc -l
```

Figure 9: 4.1.2 bash 函数

```

ouc@islouc-vm:~/Desktop/systemtool/shell$ vim file_count.sh
ouc@islouc-vm:~/Desktop/systemtool/shell$ chmod +x file_count.sh
ouc@islouc-vm:~/Desktop/systemtool/shell$ ./file_count.sh
Counting files in the current directory:
2
ouc@islouc-vm:~/Desktop/systemtool/shell$

```

Figure 10: 4.1.2 运行截图

4.1.3 条件判断和循环

功能：根据用户输入的数字判断其大小

```

9 #!/bin/bash
8 echo "Enter a number:"
7 read num
6
5 if [ $num -gt 10 ]; then
4     echo "The number is greater than 10."
3 else
2     echo "The number is 10 or less."
1 fi

```

Figure 11: 4.1.3 bash 函数

```

ouc@islouc-vm:~/Desktop/systemtool/shell$ vim check_number.sh
ouc@islouc-vm:~/Desktop/systemtool/shell$ chmod +x check_number.sh
ouc@islouc-vm:~/Desktop/systemtool/shell$ ./check_number.sh
Enter a number:
8
The number is 10 or less.
ouc@islouc-vm:~/Desktop/systemtool/shell$

```

Figure 12: 4.1.3 运行截图

4.1.4 参数传递

功能：根据提供的参数输出参数

```

6 #!/bin/bash
5 if [ $# -eq 0 ]; then
4     echo "No arguments provided."
3 else
2     echo "Hello, \"$1!\""
1 fi

```

Figure 13: 4.1.4 bash 函数

```

ouc@islouc-vm:~/Desktop/systemtool/shell$ vim greet_user.sh
ouc@islouc-vm:~/Desktop/systemtool/shell$ chmod +x greet_user.sh
ouc@islouc-vm:~/Desktop/systemtool/shell$ ./greet_user.sh CaoYu
Hello, CaoYu!
ouc@islouc-vm:~/Desktop/systemtool/shell$

```

Figure 14: 4.1.4 运行截图

4.1.5 检查网络连通性

```

4 #!/bin/bash
3 # 检查网络连通性
2 HOST="google.com"
1 ping -c 4 $HOST

```

Figure 15: 4.1.5 bash 函数

```

ouc@islouc-vm:~/Desktop/systemtool/shell$ vim ping_check.sh
ouc@islouc-vm:~/Desktop/systemtool/shell$ chmod +x ping_check.sh
ouc@islouc-vm:~/Desktop/systemtool/shell$ ./ping_check.sh
PING google.com (59.24.3.174) 56(84) bytes of data.

--- google.com ping statistics ---
4 packets transmitted, 0 received, 100% packet loss, time 3078ms

```

Figure 16: 4.1.5 运行截图

4.1.6 检查文件是否存在

这个脚本检查一个指定的文件是否存在并输出相应的信息

```

8 #!/bin/bash
7 echo "please input filename:"
6 read filename
5 # 检查文件是否存在
4 if [ -e "$filename" ]; then
3     echo "文件 '$filename' 存在。"
2     else
1         echo "文件 '$filename' 不存在。"
fi

```

Figure 17: 4.1.6 bash 函数

```

ouc@islouc-vm:~/Desktop/systemtool/shell$ vim check_file.sh
ouc@islouc-vm:~/Desktop/systemtool/shell$ chmod +x check_file.sh
ouc@islouc-vm:~/Desktop/systemtool/shell$ ./check_file.sh
please input filename:
time.sh
文件 'time.sh' 存在。
ouc@islouc-vm:~/Desktop/systemtool/shell$

```

Figure 18: 4.1.6 运行截图

4.1.7 列出当前系统的时间

这个脚本显示当前的系统日期和时间


```

4 #!/bin/bash
3
2 echo "当前系统时间是:"
1 date
5

```

Figure 19: 4.1.7 bash 函数

```

ouc@islouc-vm:~/Desktop/systemtool/shell$ vim time.sh
ouc@islouc-vm:~/Desktop/systemtool/shell$ chmod +x time.sh
ouc@islouc-vm:~/Desktop/systemtool/shell$ ./time.sh
当前系统时间是:
2024年 09月 03日 星期二 23:57:03 CST
ouc@islouc-vm:~/Desktop/systemtool/shell$

```

Figure 20: 4.1.7 运行截图

4.1.8 计算文件行数

这个脚本在检查文件是否存在的基础上计算指定文件的行数

```

11 #!/bin/bash
10
9 echo "请输入文件名:"
8 read filename
7
6 if [ -f "$filename" ]; then
5     lines=$(wc -l < "$filename")
4     echo "$filename 文件的行数是: $lines"
3     else
2         echo "文件不存在。"
1 fi

```

Figure 21: 4.1.8 bash 函数

```

ouc@islouc-vm:~/Desktop/systemtool/shell$ vim file_linecount.sh
ouc@islouc-vm:~/Desktop/systemtool/shell$ chmod +x file_linecount.sh
ouc@islouc-vm:~/Desktop/systemtool/shell$ ./file_linecount.sh
请输入文件名:
file_rename.sh
file_rename.sh 文件的行数是: 15
ouc@islouc-vm:~/Desktop/systemtool/shell$

```

Figure 22: 4.1.8 运行截图

4.1.9 重命名文件

功能：给指定文件重命名为新的名字

```

14 #!/bin/bash
13
12 echo "请输入要重命名的文件名:"
11 read oldname
10
9 echo "请输入新文件名:"
8 read newname
7
6 if [ -f "$oldname" ]; then
5     mv "$oldname" "$newname"
4     echo "文件已重命名为 $newname"
3 else
2     echo "文件 $oldname 不存在。"
1 fi

```

Figure 23: 4.1.9 bash 函数

```

ouc@islouc-vm:~/Desktop/systemtool/shell$ vim file_rename.sh
ouc@islouc-vm:~/Desktop/systemtool/shell$ chmod +x file_rename.sh
ouc@islouc-vm:~/Desktop/systemtool/shell$ ./file_rename.sh
请输入要重命名的文件名:
file_count.sh
请输入新文件名:
count.sh
文件已重命名为 count.sh
ouc@islouc-vm:~/Desktop/systemtool/shell$

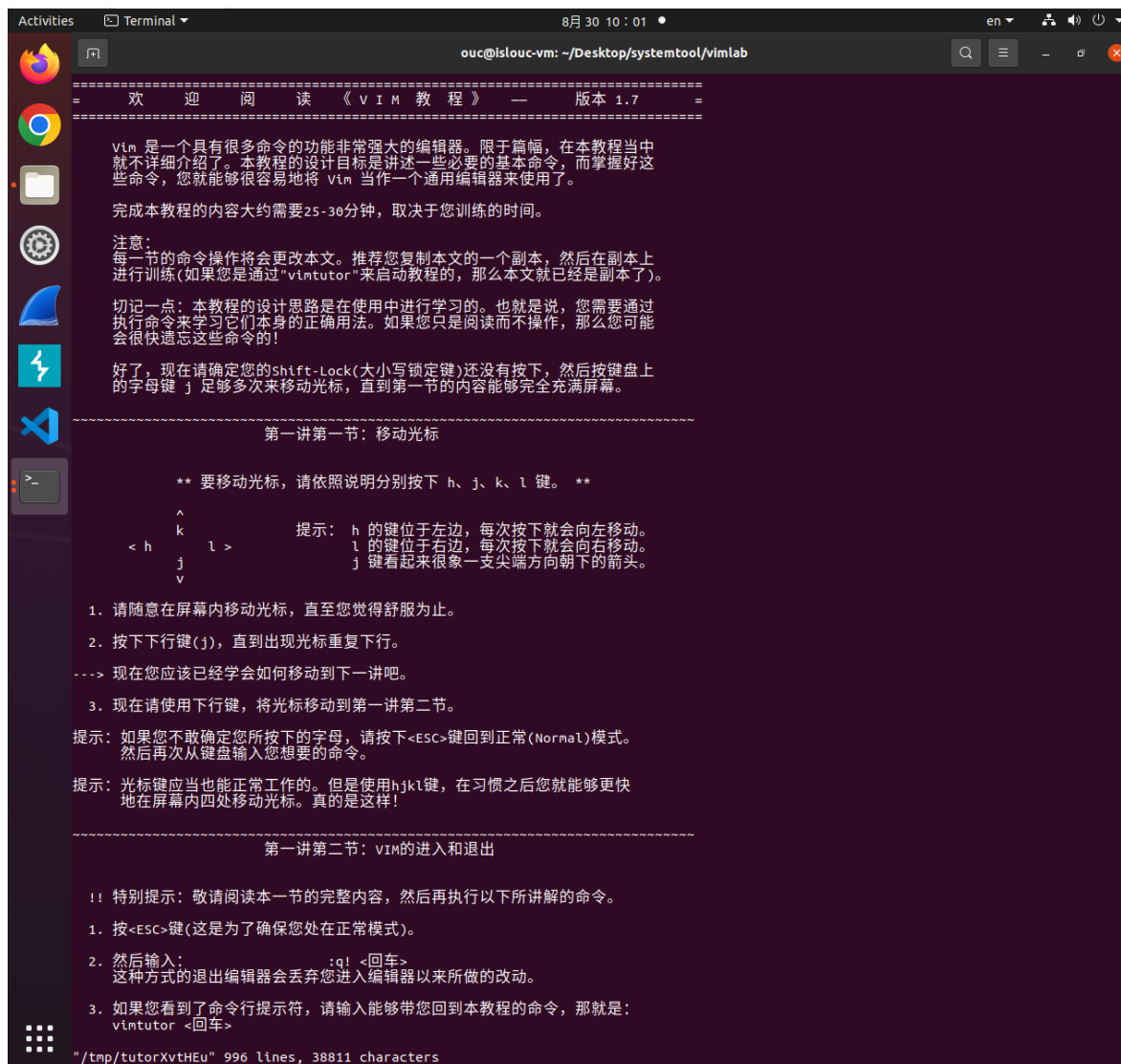
```

Figure 24: 4.1.9 运行截图

4.2 编辑器 (vim)

4.2.1 完成 vintutor

vintutor



```
=====
= 欢 迎 阅 读 《 V I M 教 程 》 — 版 本 1.7 =
=====

vim 是一个具有很多命令的功能非常强大的编辑器。限于篇幅，在本教程当中
就不详细介绍了。本教程的设计目标是讲述一些必要的基本命令，而掌握好这
些命令，您就能够很容易地将 vim 当作一个通用编辑器来使用了。

完成本教程的内容大约需要25-30分钟，取决于您训练的时间。

注意：
每一节的命令操作将会更改本文。推荐您复制本文的一个副本，然后在副本上
进行训练(如果您是通过“vintutor”来启动教程的，那么本文就已经是副本了)。

切记一点：本教程的设计思路是在使用中进行学习的。也就是说，您需要通过
执行命令来学习它们本身的正确用法。如果您只是阅读而不操作，那么您可能
会很快忘记这些命令的！

好了，现在请确定您的Shift-Lock(大小写锁定键)还没有按下，然后按键盘上
的字母键 j 足够多次来移动光标，直到第一节的内容能够完全充满屏幕。

-----
第一讲第一节：移动光标

** 要移动光标，请依照说明分别按下 h、j、k、l 键。 **

      ^
      k           提示： h 的键位于左边，每次按下就会向左移动。
< h   l >         l 的键位于右边，每次按下就会向右移动。
      j           j 键看起来很像一支尖端方向朝下的箭头。
      v

1. 请随意在屏幕内移动光标，直至您觉得舒服为止。
2. 按下下行键(j)，直到出现光标重复下行。
---> 现在您应该已经学会如何移动到下一讲吧。
3. 现在请使用下行键，将光标移动到第一讲第二节。

提示：如果您不敢确定您所按下的字母，请按下<ESC>键回到正常(Normal)模式。
      然后再次从键盘输入您想要的命令。

提示：光标键应当也能正常工作的。但是使用hjkl键，在习惯之后您就能够更快
      地在屏幕内四处移动光标。真的是这样！

-----
第一讲第二节：VIM的进入和退出

!! 特别提示：敬请阅读本一节的完整内容，然后再执行以下所讲解的命令。

1. 按<ESC>键(这是为了确保您处在正常模式)。
2. 然后输入：          :q! <回车>
   这种方式的退出编辑器会丢弃您进入编辑器以来所做的改动。
3. 如果您看到了命令行提示符，请输入能够带您回到本教程的命令，那就是：
   vintutor <回车>

"/tmp/tutorXvtHEu" 996 lines, 38811 characters
```

Figure 25: 4.2.1 完成 vintutor

4.2.2 下载 vimrc, 然后把它保存到 /.vimrc

```
ouc@islouc-vm:~/Desktop/systemtool/vinlab$ vim --version
VIM - Vi IMproved 8.1 (2018 May 18, compiled Mar 14 2024 09:29:25)
包含补丁: 1-213, 1840, 214-579, 1969, 580-1848, 4975, 5023, 2110, 1849-1854, 1857, 1855-1857, 1331, 1858, 1858-1859, 1873, 1860-1969
, 1992, 1970-1992, 2010, 1993-2068, 2106, 2069-2106, 2108, 2107-2109, 2109-2111, 2111-2112, 2112-2269, 3612, 3625, 3669, 3741, 1847
修改者 team+vim@tracker.debian.org
编译器 team+vim@tracker.debian.org
巨型版本 无图形界面。 可使用(+)与不可使用(-)的功能:
+acl                    -farsi                    -mouse_sysmouse      -tag_any_white
+arabic                 +file_in_path           +mouse_urxvt         -tcl
+autocmd                +find_in_path           +mouse_xterm         +termguicolors
+autocmdnr             +float                  +multi_byte          +terminal
+autoservername         +folding                +multi_lang          +terminfo
+balloon_eval           -footer                 +mzscheme            +termresponse
+balloon_eval_term     +fork()                 +netbeans_intg       +textobjects
-browse                 +gettext                +num64               +textprop
++builtin_terms         -hangul_input           +packages            +timers
+byte_offset            +iconv                  +path_extra          +title
+channel                +insert_expand          -perl                -toolbar
+cindent                +job                    +persistent_undo     +user_commands
-clientserver           +jumplist               +postscript          +vartabs
+clipboard              +keymap                 +printer             +vertsplit
+cmdline_compl          +lambda                 +profile             +virtualedit
+cmdline_hist           +langmap                +python              +visual
+cmdline_info           +libcall                +python3             +visualextra
+comments               +linebreak              +quickfix            +viminfo
+conceal                +lispindent             +reltime             +vreplace
+cryptv                 +listcmds               +rightleft           +wildignore
+cscope                 +localmap               -ruby                +wildmenu
+cursordbind            -lua                    +scrollbind          +windows
+cursormshape           +menu                   +signs               +writebackup
+dialog_con             +mksession              +smartindent         -X11
+diff                   +modify_fname           +sound               -xfontset
+digraphs               +mouse                  +spell               -xim
-dnd                    -mouseshape             +startuptime         -xpm
-ebcdic                 +mouse_dec              +statusline          -xsmp
+emacs_tags             +mouse_gpm              -sun_workshop        -xterm_clipboard
+eval                   -mouse_jsbterm          +syntax              -xterm_save
+ex_extra               +mouse_netterm          +tag_binary
+extra_search           +mouse_ssr              -tag_old_static

系统 vimrc 文件: "$VIM/vimrc"
用户 vimrc 文件: "$HOME/.vimrc"
第二用户 vimrc 文件: "~/.vim/vimrc"
用户 exrc 文件: "$HOME/.exrc"
defaults title: "$VIMRUNTIME/defaults.vim"
$VIM 预设值: "/usr/share/vim"
编译方式: gcc -c -I. -Iproto -DHAVE_CONFIG_H -Wdate-time -g -O2 -fdebug-prefix-map=/build/vim-exHr0L/vim-8.1.2269=. -fstack-protector-strong -Wformat -Werror=format-security -D_REENTRANT -U_FORTIFY_SOURCE -D_FORTIFY_SOURCE=1
链接方式: gcc -Wl,-Bsymbolic-functions -Wl,-z,relro -Wl,-z,now -Wl,--as-needed -o vim -lm -ltninfo -lnsl -lselinux -lcanbe
rra -lacl -lattr -lgpm -ldl -L/usr/lib/python3.8/config-3.8-x86_64-linux-gnu -lpthread -ldl -lutil -lm -lm
```

Figure 26: 4.2.2 下载 vimrc

4.2.3 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

```
ouc@islouc-vm:~/Desktop/systemtool/vinlab$ mkdir -p ~/.vim/pack/vendor/start
ouc@islouc-vm:~/Desktop/systemtool/vinlab$ cd ~/.vim/pack/vendor/start
ouc@islouc-vm:~/Desktop/systemtool/vinlab$ 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 149 (delta 66), pack-reused 4131 (from 1)
Receiving objects: 100% (4299/4299), 1.70 MiB | 1.23 MiB/s, done.
Resolving deltas: 100% (1661/1661), done.
```

Figure 27: 4.2.3 下载插件

3. 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.

```
[未命名]
> ctrlp.vim/autoload/ctrlp.vim
> ctrlp.vim/plugin/ctrlp.vim
> ctrlp.vim/doc/ctrlp.txt
> ctrlp.vim/doc/ctrlp.cnx
> ctrlp.vim/readme.md
> ctrlp.vim/autoload/ctrlp/undo.vim
> ctrlp.vim/autoload/ctrlp/line.vim
> ctrlp.vim/autoload/ctrlp/tag.vim
> ctrlp.vim/autoload/ctrlp/dir.vim
> ctrlp.vim/LICENSE
prt path <mru>={ files }=<buf> <->
>>> _
```

Figure 28: 4.2.3.3 CtrlP

4. 自定义 CtrlP: 添加 configuration 到你的 `/.vimrc` 来用按 Ctrl-P 打开 CtrlP

```
let g:ctrlp_map = '<c-p>'
let g:ctrlp_cmd = 'CtrlP'
let g:ctrlp_working_path_mode = 'ra' #设置默认路径为当前路径
```

```

8 set mouse+=a
7
6 " Try to prevent bad habits like using the arrow keys for movement
5 " not the only possible bad habit. For example, holding down the t
4 " for movement, rather than using more efficient movement commands
3 " bad habit. The former is enforceable through a .vimrc, while we
2 " how to prevent the latter.
1 " Do this in normal mode...
72 noremap <C-p> :CtrlP<CR>
1 noremap <Left> :echoe "Use h"<CR>
2 noremap <Right> :echoe "Use l"<CR>
3 noremap <Up> :echoe "Use k"<CR>
4 noremap <Down> :echoe "Use j"<CR>
5 " ...and in insert mode
6 inoremap <Left> <ESC>:echoe "Use h"<CR>
7 inoremap <Right> <ESC>:echoe "Use l"<CR>
8 inoremap <Up> <ESC>:echoe "Use k"<CR>
9 inoremap <Down> <ESC>:echoe "Use j"<CR>

```

Figure 29: 4.2.3.4 自定义 CtrlP

4.2.4 进一步自定义 `/.vimrc` 和安装更多插件

1. 安装 vim-plug

```
$ curl -fLo ~/.vim/autoload/plug.vim --create-dirs \
https://raw.githubusercontent.com/junegunn/vim-plug/master/plug.vim
```

```

ouc@islouc-vm:~/Desktop/systemtool/vimlab$ curl -fLo ~/.vim/autoload/plug.vim --create-dirs \
> https://raw.githubusercontent.com/junegunn/vim-plug/master/plug.vim
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total   Spent    Left     Speed

  0     0    0     0    0     0      0      0 --:--:--  0:02:57 --:--:--    0
  0     0    0     0    0     0      0      0 --:--:--  0:03:22 --:--:--    0

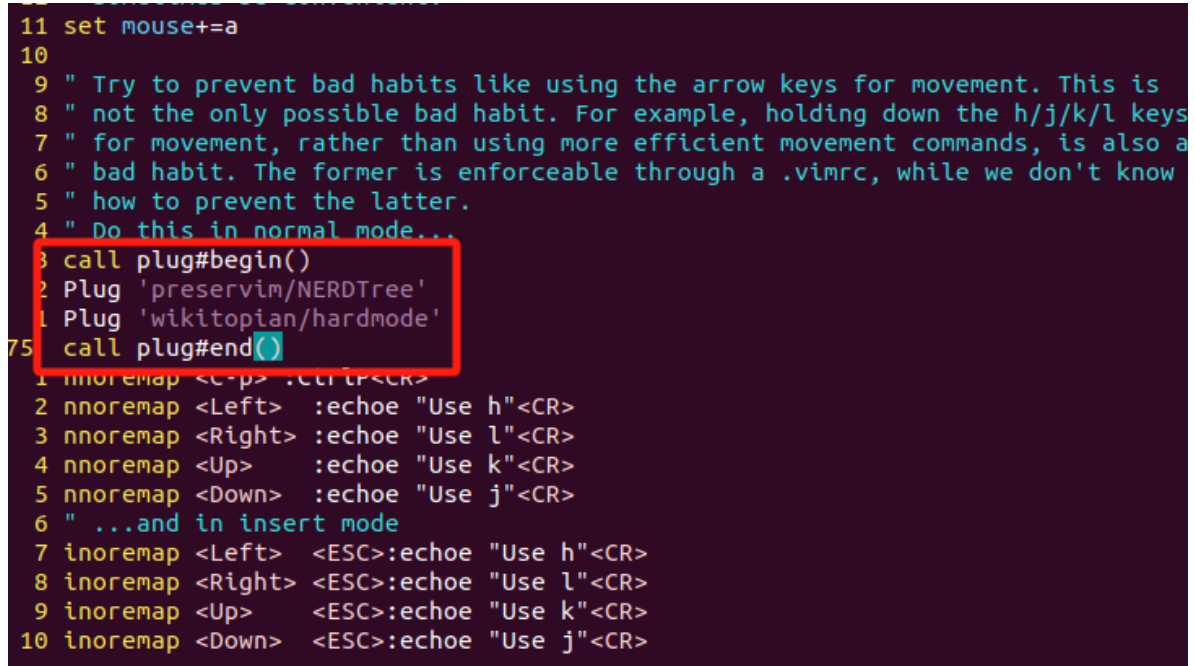
  0     0    0     0    0     0      0      0 --:--:--  0:03:23 --:--:--    0
100 84223 100 84223    0     0  405      0  0:03:27  0:03:27 --:--:-- 23684
ouc@islouc-vm:~/Desktop/systemtool/vimlab$

```

Figure 30: 4.2.4.1 安装 vim-plug

2. 修改 ~/.vimrc

```
call plug#begin()  
Plug 'preservim/NERDTree' #需要安装的插件 NERDTree  
Plug 'wikitopian/hardmode' #安装 hardmode  
call plug#end()
```



The screenshot shows a .vimrc file with the following content:

```
11 set mouse+=a  
10  
9 " Try to prevent bad habits like using the arrow keys for movement. This is  
8 " not the only possible bad habit. For example, holding down the h/j/k/l keys  
7 " for movement, rather than using more efficient movement commands, is also a  
6 " bad habit. The former is enforceable through a .vimrc, while we don't know  
5 " how to prevent the latter.  
4 " Do this in normal mode...  
3 call plug#begin()  
2 Plug 'preservim/NERDTree'  
1 Plug 'wikitopian/hardmode'  
75 call plug#end()  
1 inoremap <C-p> :CtrlP<CR>  
2 nnoremap <Left> :echoe "Use h"<CR>  
3 nnoremap <Right> :echoe "Use l"<CR>  
4 nnoremap <Up> :echoe "Use k"<CR>  
5 nnoremap <Down> :echoe "Use j"<CR>  
6 " ...and in insert mode  
7 inoremap <Left> <ESC>:echoe "Use h"<CR>  
8 inoremap <Right> <ESC>:echoe "Use l"<CR>  
9 inoremap <Up> <ESC>:echoe "Use k"<CR>  
10 inoremap <Down> <ESC>:echoe "Use j"<CR>
```

A red box highlights the lines 3 through 75, which contain the plugin installation code.

Figure 31: 4.2.4.2 修改 ~/.vimrc

3. 在 vim 命令行中执行:PlugInstall

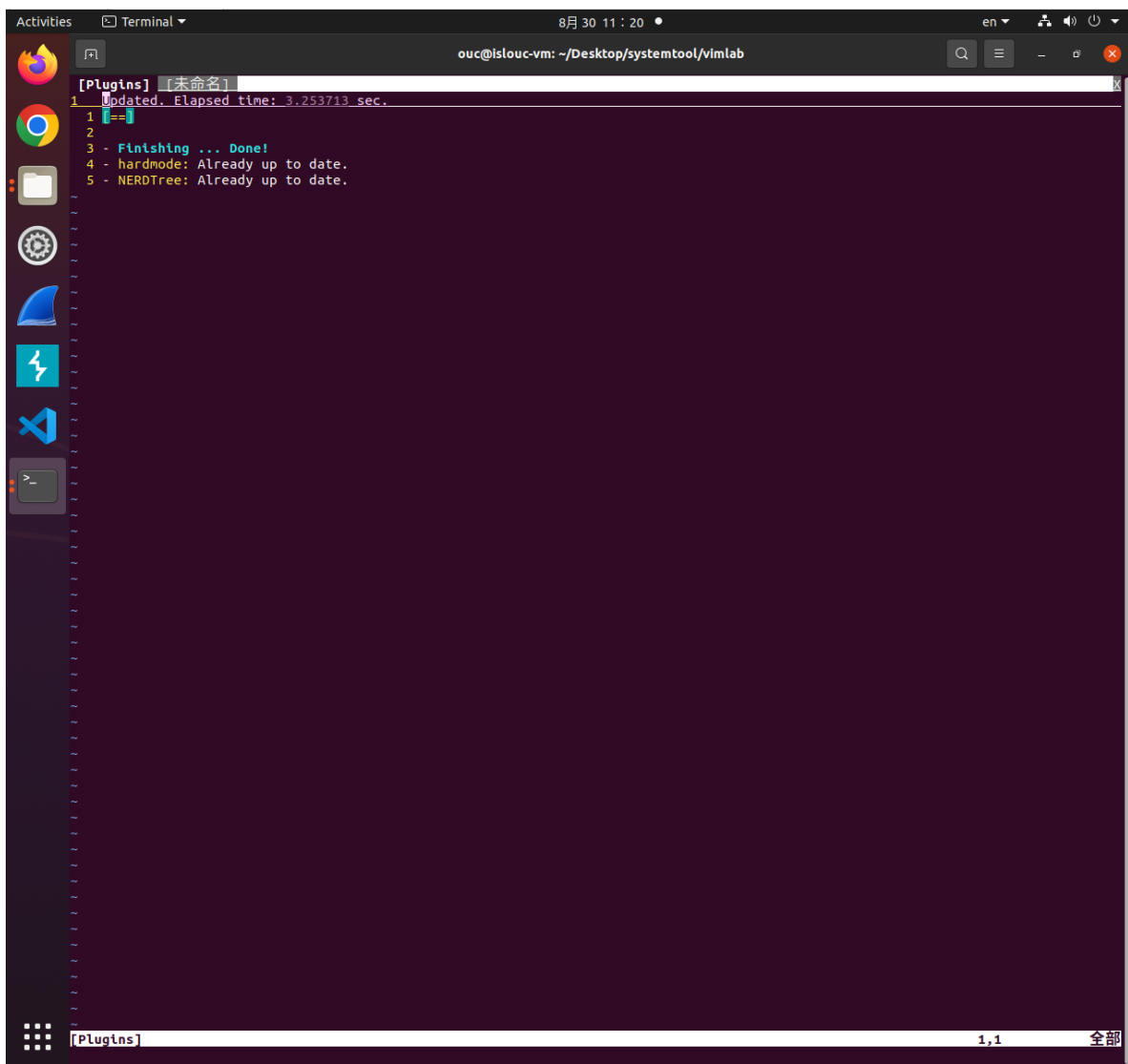


Figure 32: 4.2.4.3 在 vim 命令行中执行:PlugInstall

4.2.5 水平分屏和垂直分屏

:split filename: 水平分屏打开 filename。

:vsplit filename: 垂直分屏打开 filename。

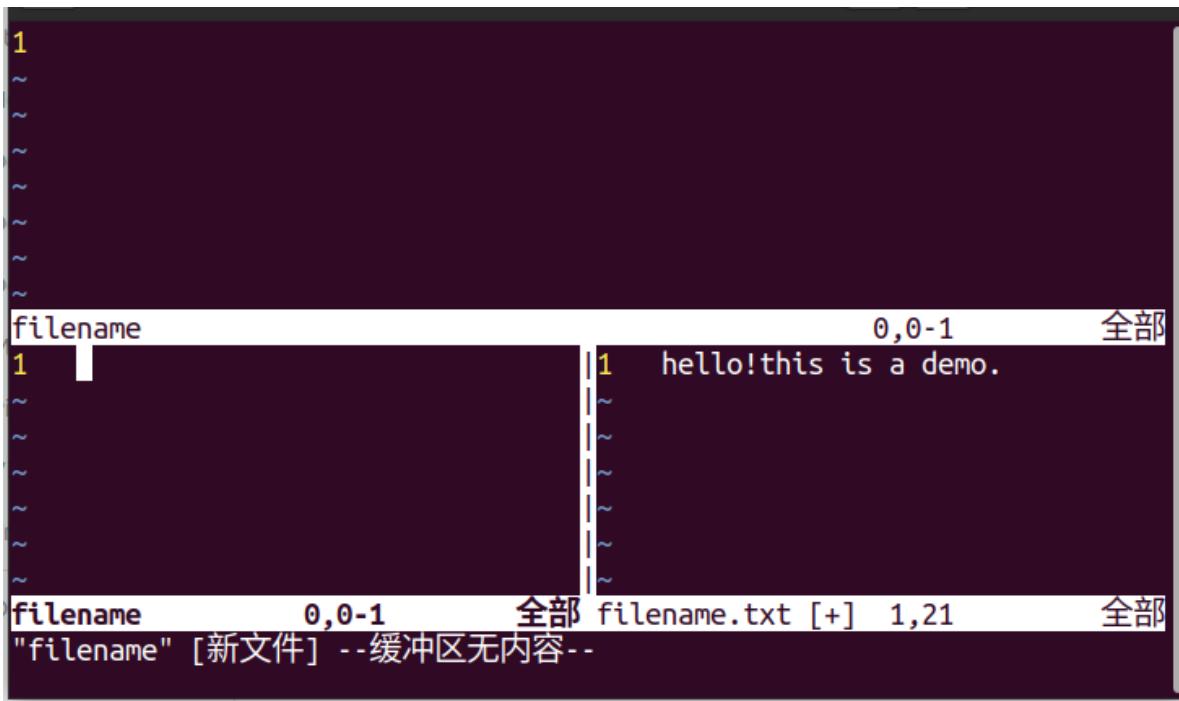


Figure 33: 4.2.5 水平分屏和垂直分屏

4.2.6 查找和替换

查找文本：输入:/Hello 查找”Hello”。

替换文本：输入：

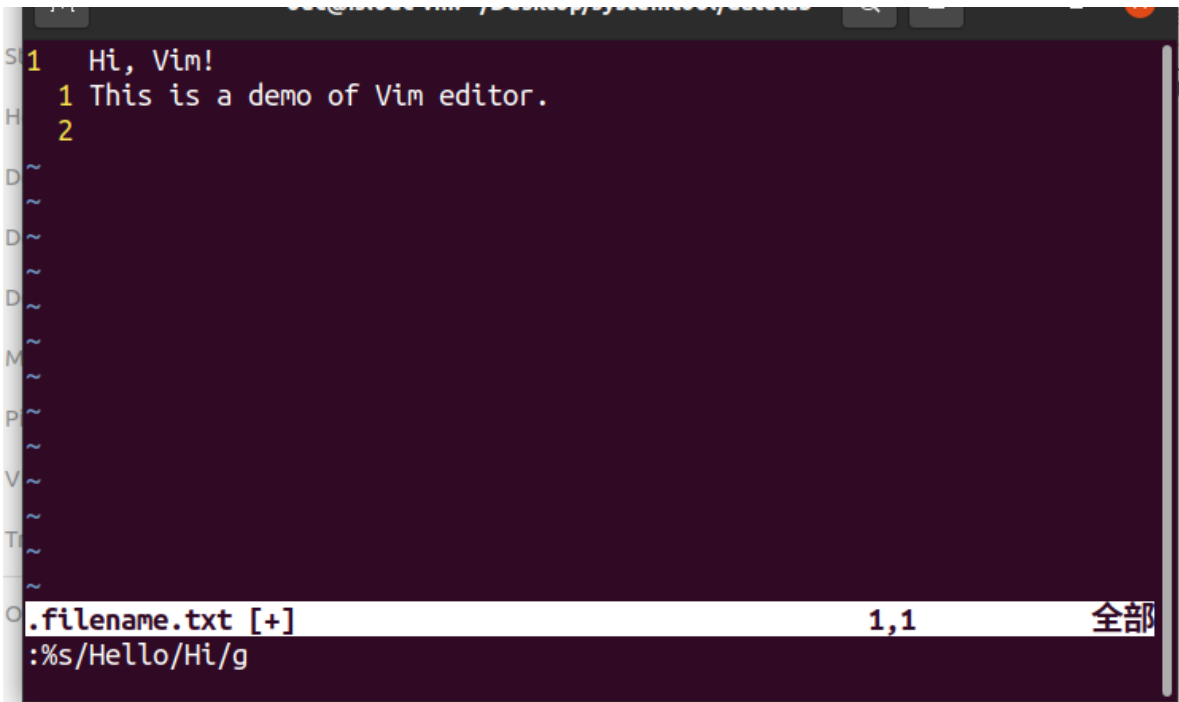
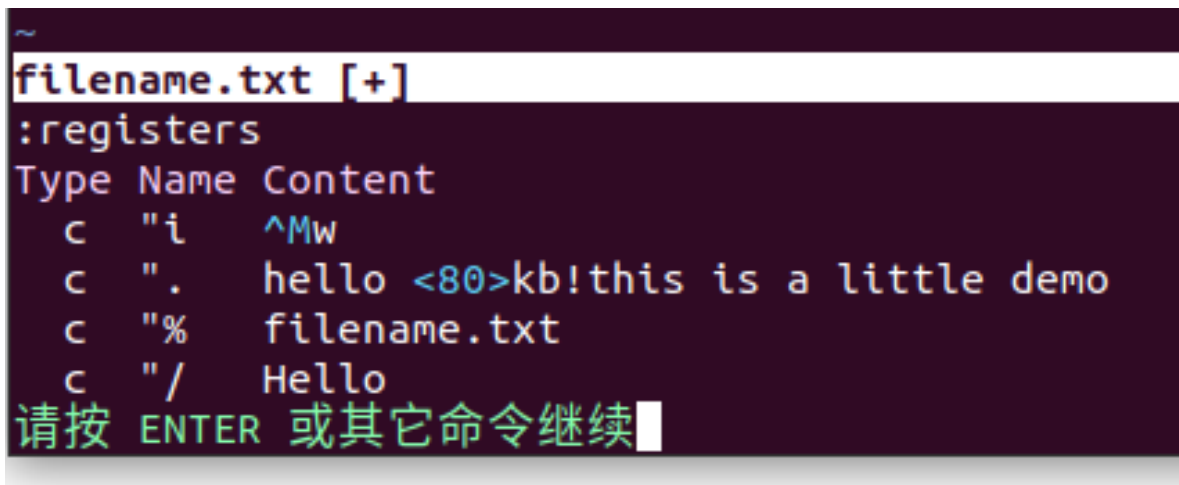


Figure 34: 4.2.6 查找和替换

4.2.7 使用注册表

查看所有寄存器的内容



The screenshot shows a debugger window with a dark background. At the top, there's a title bar with a tilde symbol. Below it, the text 'filename.txt [+]' is displayed. The main area shows the command ':registers' followed by a table of registers. The table has three columns: 'Type', 'Name', and 'Content'. There are four rows of data. The first row shows 'c' for type, 'i' for name, and '^Mw' for content. The second row shows 'c' for type, '.' for name, and 'hello <80>kb!this is a little demo' for content. The third row shows 'c' for type, '%' for name, and 'filename.txt' for content. The fourth row shows 'c' for type, '/' for name, and 'Hello' for content. At the bottom, there's a green prompt '请按 ENTER 或其它命令继续' followed by a cursor.

Type	Name	Content
c	"i	^Mw
c	".	hello <80>kb!this is a little demo
c	"%	filename.txt
c	"/	Hello

Figure 35: 4.2.7 使用注册表

4.3 数据整理

4.3.1 统计符合条件的单词数量

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

这个命令的作用是：

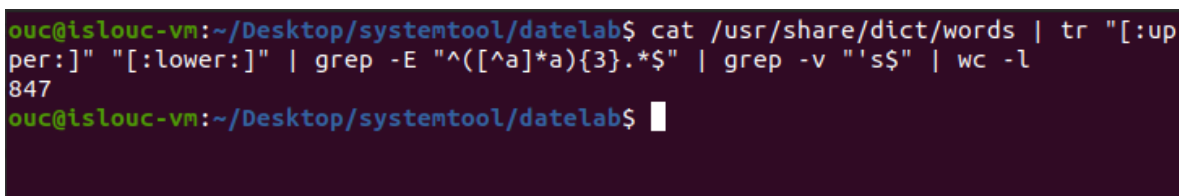
cat /usr/share/dict/words: 显示文件内容。

tr "[:upper:]" "[:lower:]": 将所有字母转换为小写。

grep -E "[a]*a){3}.*\$": 筛选出包含至少三个字母'a'的单词。

grep -v "'s\$": 去掉以's'结尾的单词。

wc -l: 统计行数，即符合条件的单词总数。



The screenshot shows a terminal window with a dark background. The prompt is 'ouc@islouc-vm:~/Desktop/systemtool/datelab\$'. The command being executed is 'cat /usr/share/dict/words | tr "[:upper:]" "[:lower:]" | grep -E "^([a]*a){3}.*\$" | grep -v "'s\$" | wc -l'. The output of the command is '847'. The prompt is now 'ouc@islouc-vm:~/Desktop/systemtool/datelab\$' followed by a cursor.

Figure 36: 4.3.1 统计符合条件的单词数量

得到了结果 847，表示文件中符合条件的单词有 850 个。

4.3.2 统计出现频率前三的末尾两个字母

```
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
```

这个命令的作用是：

sort：将提取出来的字母对排序。

uniq -c：统计每个字母对的出现次数。

sort：再次排序，按出现次数排序。

tail -n3：获取出现次数最多的三个字母对。

```
ouc@islouc-vm:~/Desktop/systemtool/datelab$ 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
```

Figure 37: 4.3.2 统计出现频率前三的末尾两个字母

由结果可以知道出现频率最高的三个末尾两个字母组合是 am(8 次)、ce(8 次) 和 ca(9 次)

4.3.3 统计有多少种不同的词尾两字母组合

```
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
```

这个命令的作用是：

sort：将提取出的字母对排序。

uniq：去重，得到不同的字母对。

wc -l：统计不同字母对的数量。

```
ouc@islouc-vm:~/Desktop/systemtool/datelab$ cat /usr/share/dict/words | tr "[:upper:]" "[:lower:]" | grep -E "^(^[a]*a){3}.*$" | grep -v "'s$" | sed -E "s/.*([a-z]{2})$/\1/" | sort | uniq | wc -l
111
ouc@islouc-vm:~/Desktop/systemtool/datelab$
```

Figure 38: 4.3.3 统计有多少种不同的词尾两字母组合

最后得到结果可以知道不同的组合数量为 111

4.3.4 查找从未出现过的字母组合

1. 将生成的所有可能字母组合保存到 all.txt 文件中

```
#!/bin/bash
for i in {a..z}; do
    for j in {a..z}; do
        echo "$i$j"
    done
done
```

将上述脚本保存为 all.sh 并运行 ./all.sh > all.txt

```

6 #!/bin/bash
5 for i in {a..z}; do
4     for j in {a..z}; do
3         echo "$i$j"
2     done
1 done
7

```

Figure 39: 4.3.4.1 编辑 all.sh

```

ouc@islouc-vm:~/Desktop/systemtool/datelab$ vim all.sh
ouc@islouc-vm:~/Desktop/systemtool/datelab$ chmod +x all.sh
ouc@islouc-vm:~/Desktop/systemtool/datelab$ ./all.sh > all.txt
ouc@islouc-vm:~/Desktop/systemtool/datelab$

```

Figure 40: 4.3.4.1 运行./all.sh > all.txt

2. 从文件中提取出现过的字母组合并保存到 occurrence.txt 文件中

```

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 > occurrence.txt

```

```

ouc@islouc-vm:~/Desktop/systemtool/datelab$ cat /usr/share/dict/words | tr "[:upper:]" "[:lower:]" | grep -E "^(^[a]*a){3}.*$" | grep -v "'s$" | sed -E "s/.*([a-z]{2})$/\1/" | sort | uniq > occurrence.txt
ouc@islouc-vm:~/Desktop/systemtool/datelab$

```

Figure 41: 4.3.4.2 提取字母组合并保存

3. 比较 all.txt 和 occurrence.txt, 找出未出现的组合

```

diff --unchanged-group-format='' <(cat occurrence.txt) <(cat all.txt) | wc -l

```

这个命令将 occurrence.txt 和 all.txt 中的差异内容输出, 并计算差异的行数, 即未出现的字母组合数量。

```
ouc@islouc-vm:~/Desktop/systemtool/datelab$ diff --unchanged-group-format='' <(cat occurance.txt) <(cat all.txt) | wc
-l
565
ouc@islouc-vm:~/Desktop/systemtool/datelab$
```

Figure 42: 4.3.4.3 计算差异的行数

4.3.5 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？

答：如果在原地替换的过程中出现了错误或者操作中断，可能会导致文件内容丢失或损坏。使用正则处理文件

```
sed -i.bak s/REGEX/SUBSTITUTION/ input.txt
```

可以自动创建一个后缀为.bak 的文件

```
ouc@islouc-vm:~/Desktop/systemtool/datelab$ vim input.txt
ouc@islouc-vm:~/Desktop/systemtool/datelab$ sed -i.bak 's/REGEX/SUBSTITUTION/' input.txt
ouc@islouc-vm:~/Desktop/systemtool/datelab$ ls
all.sh  all.txt  input.txt  input.txt.bak  occurance.txt
ouc@islouc-vm:~/Desktop/systemtool/datelab$
```

Figure 43: 4.3.5 原地替换

5 实验总结

在实验使用 shell 工具和编写脚本的过程中，我发现这大大提高了工作效率，这些实验使我对 shell 编程有了更深入的理解，并能更有效地应用这些工具解决实际问题。

在 vim 编辑器实验中，我学会了基本的 vim 编辑模式，比如插入模式、替换模式等，了解了 vim 的界面本身也是一种程序语言，在完成课后练习的过程中尽管遇到了一些困难，但是在寻求了同学的帮助之后基本学会了 vim 编辑器的一些基本操作，受益匪浅。

在实验数据整理中，我学会了一些对数据处理的方式，比如将某种格式存储的数据转换成另外一种格式，但是对于实验的一些命令代码我不是很理解，之后我通过查找资料理解了这些命令都是什么意思，更好的掌握了这个实验的知识。

6 github 链接

<https://github.com/Caoyu2233/Systemtools.git>