

系统开发工具基础实验报告（二）

22110032031 张希敏

2024 年 9 月 12 日

目录

1	练习主题	1
2	练习内容	1
2.1	四个题目	1
2.1.1	题目一	1
2.1.2	题目二	3
2.1.3	题目三	3
2.1.4	题目四	4
2.2	20 个实例	5
3	问题及解决方案	16
4	解题感悟	16
5	github 链接	16

1 练习主题

(1) Shell 工具和脚本

(2) 编辑器 (Vim)

(3) 数据整理

2 练习内容

2.1 四个题目

2.1.1 题目一

阅读 `man ls` , 然后使用 `ls` 命令进行如下操作:

- (1) 输出所有文件 (包括隐藏文件)。
- (2) 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)。
- (3) 文件以最近访问顺序排序。
- (4) 以彩色文本显示输出结果。

答: 阅读 `ls` 的手册, 加后缀如下:

- (1) 所有文件 (包括隐藏文件): `-a`
- (2) 文件打印以人类可以理解的格式输出: `-h`
- (3) 文件以最近访问顺序排序: `-t`
- (4) 以彩色文本显示输出结果: `-color=auto`

```
abc@abc-virtual-machine:~/Desktop$ man ls

ls(1)
NAME
ls - list directory contents

SYNOPSIS
ls [OPTION]... [FILE]...

DESCRIPTION
List information about the FILES (the current directory by default). Sort entries alphabetically if none of -ctfuwSUX nor --sort is specified.
Mandatory arguments to long options are mandatory for short options too.

-a, --all
do not ignore entries starting with .

-A, --almost-all
do not list implied . and ..

--author
with -l, print the author of each file

-b, --escape
print C-style escapes for nongraphic characters

--block-size=SIZE
with -l, scale sizes by SIZE when printing them; e.g., '--block-size=M'; see SIZE format below

-B, --ignore-backups
do not list implied entries ending with ~

-c with -lt: sort by, and show, ctime (time of last modification of file status information); with -l: show ctime and sort by name; otherwise: sort by ctime, newest first

-C list entries by columns

--color[=WHEN]
colorize the output; WHEN can be 'always' (default if omitted), 'auto', or 'never'; more info below

-d, --directory
list directories themselves, not their contents

-B, --dired

Manual page ls(1) line 1/208 19% (press h for help or q to quit)

abc@abc-virtual-machine:~/Desktop$ ls -l -a
total 40
drwxr-xr-x 6 abc abc 4096 9月 6 12:05 .
drwxr-xr-x 18 abc abc 4096 9月 12 12:50 ..
-rw-r----- 1 abc abc 3288 2月 1 2018 cp0_reg.v
drwxrwxrwx 5 abc abc 4096 4月 26 2019 func_test
-rw-r----- 1 abc abc 716 9月 6 12:03 hilo.v
drwxrwxr-x 4 abc abc 4096 1月 12 2024 lab
-rw-r----- 1 abc abc 2298 2月 1 2018 regfile.v
drwxrwxrwx 2 abc abc 4096 4月 26 2019 rtl
-rw-r----- 1 abc abc 398 2月 1 2018 scu.v
drwxrwxrwx 4 abc abc 4096 8月 22 19:43 test_c

abc@abc-virtual-machine:~/Desktop$ ls -l -h
total 32K
-rw-r----- 1 abc abc 3.3K 2月 1 2018 cp0_reg.v
drwxrwxrwx 5 abc abc 4.0K 4月 26 2019 func_test
-rw-r----- 1 abc abc 716 9月 6 12:03 hilo.v
drwxrwxr-x 4 abc abc 4.0K 1月 12 2024 lab
-rw-r----- 1 abc abc 2.3K 2月 1 2018 regfile.v
drwxrwxrwx 2 abc abc 4.0K 4月 26 2019 rtl
-rw-r----- 1 abc abc 398 2月 1 2018 scu.v
drwxrwxrwx 4 abc abc 4.0K 8月 22 19:43 test_c

abc@abc-virtual-machine:~/Desktop$ ls -l -t
total 32
-rw-r----- 1 abc abc 716 9月 6 12:03 hilo.v
drwxrwxrwx 4 abc abc 4096 8月 22 19:43 test_c
drwxrwxr-x 4 abc abc 4096 1月 12 2024 lab
drwxrwxrwx 5 abc abc 4096 4月 26 2019 func_test
drwxrwxrwx 2 abc abc 4096 4月 26 2019 rtl
-rw-r----- 1 abc abc 3288 2月 1 2018 cp0_reg.v
-rw-r----- 1 abc abc 2298 2月 1 2018 regfile.v
-rw-r----- 1 abc abc 398 2月 1 2018 scu.v

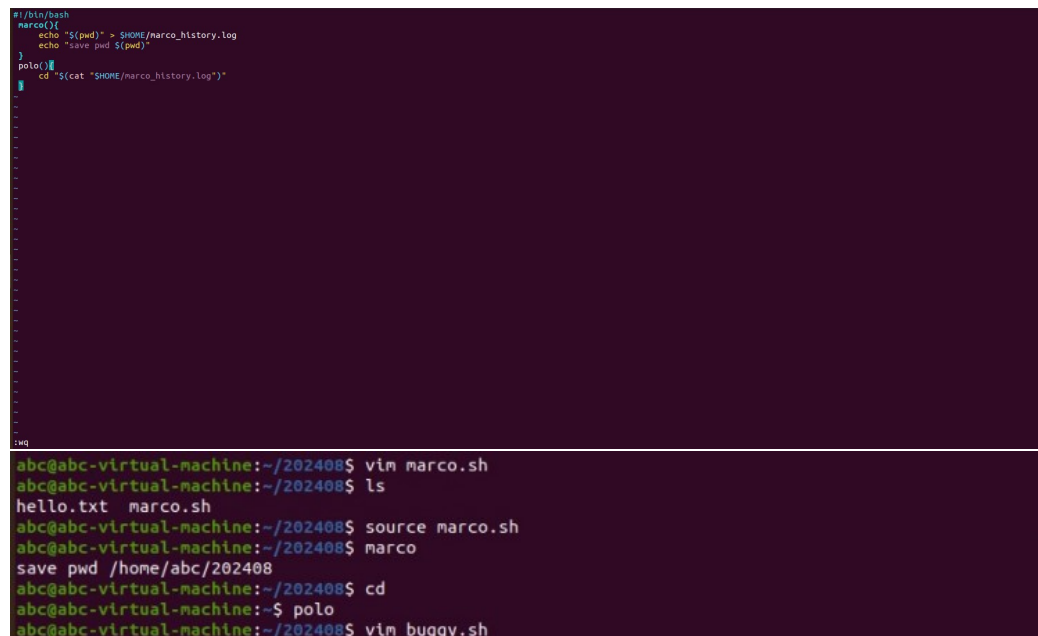
abc@abc-virtual-machine:~/Desktop$ ls -l --color=auto
total 32
-rw-r----- 1 abc abc 3288 2月 1 2018 cp0_reg.v
drwxrwxrwx 5 abc abc 4096 4月 26 2019 func_test
-rw-r----- 1 abc abc 716 9月 6 12:03 hilo.v
drwxrwxr-x 4 abc abc 4096 1月 12 2024 lab
-rw-r----- 1 abc abc 2298 2月 1 2018 regfile.v
drwxrwxrwx 2 abc abc 4096 4月 26 2019 rtl
-rw-r----- 1 abc abc 398 2月 1 2018 scu.v
drwxrwxrwx 4 abc abc 4096 8月 22 19:43 test_c

abc@abc-virtual-machine:~/Desktop$ ls -l --color=never
total 32
-rw-r----- 1 abc abc 3288 2月 1 2018 cp0_reg.v
drwxrwxrwx 5 abc abc 4096 4月 26 2019 func_test
-rw-r----- 1 abc abc 716 9月 6 12:03 hilo.v
drwxrwxr-x 4 abc abc 4096 1月 12 2024 lab
-rw-r----- 1 abc abc 2298 2月 1 2018 regfile.v
drwxrwxrwx 2 abc abc 4096 4月 26 2019 rtl
-rw-r----- 1 abc abc 398 2月 1 2018 scu.v
drwxrwxrwx 4 abc abc 4096 8月 22 19:43 test_c
```

2.1.2 题目二

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。

答： 如图。代码文件已上传至 github。



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

abc@abc-virtual-machine:~/202408$ vim marco.sh
abc@abc-virtual-machine:~/202408$ ls
hello.txt marco.sh
abc@abc-virtual-machine:~/202408$ source marco.sh
abc@abc-virtual-machine:~/202408$ marco
save pwd /home/abc/202408
abc@abc-virtual-machine:~/202408$ cd
abc@abc-virtual-machine:~$ polo
abc@abc-virtual-machine:~/202408$ vim buggy.sh
```

2.1.3 题目三

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。

答： 如图。代码文件已上传至 github。


```

abc@abc-virtual-machine:~/202408$ mkdir html_root
abc@abc-virtual-machine:~/202408$ cd html_root
abc@abc-virtual-machine:~/202408/html_root$ touch {1..10}.html
abc@abc-virtual-machine:~/202408/html_root$ mkdir html
abc@abc-virtual-machine:~/202408/html_root$ cd html
abc@abc-virtual-machine:~/202408/html_root/html$ touch xxxx.html

```

然后执行 find 命令，输入 `find . -type f -name "*.html" | xargs -d '\n' tar -cvzf html.zip`

```

abc@abc-virtual-machine:~/202408$ find . -type f -name "*.html" | xargs -d '\n' tar -cvzf html.zip
./html_root/2.html
./html_root/3.html
./html_root/7.html
./html_root/9.html
./html_root/4.html
./html_root/1.html
./html_root/5.html
./html_root/8.html
./html_root/html/xxxx.html
./html_root/6.html
./html_root/10.html
abc@abc-virtual-machine:~/202408$ ls
buggy.sh  debug.sh  html_root  html.zip  marco.sh  out.log  pidwait.sh

```

2.2 20 个实例

(1) **shell:** 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？

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

当文件数量较多时，上面的解答会得出错误结果，解决办法是增加 `-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：

`find . -type f -mmin -60 -print0 | xargs -0 ls -lt | head -10`

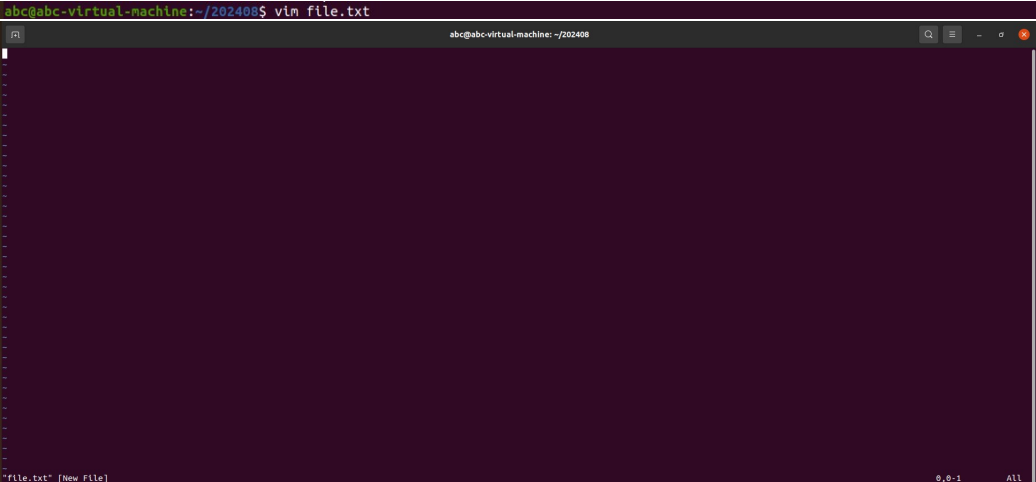
```

abc@abc-virtual-machine:~/202408$ find . -type f -print0 | xargs -0 ls -lt | head -1
-rw-rw-r-- 1 abc abc 225 9月 12 21:26 ./html.zip
abc@abc-virtual-machine:~/202408$ find . -type f -mmin -60 -print0 | xargs -0 ls -lt | head -10
-rw-rw-r-- 1 abc abc 225 9月 12 21:26 ./html.zip
-rwxrwxrwx 1 abc abc 230 9月 12 21:25 ./debug.sh
-rw-rw-r-- 1 abc abc 153 9月 12 21:18 ./marco.sh

```

(2) **vim:** 用 vim 编辑器创建一个文件。

```
abc@abc-virtual-machine:~/202408$ vim file.txt
```




(3) **vim:** 按下 i 键切换输入模式，在文件中输入一段文字。

```
abc@abc-virtual-machine: ~/202408
```



(4) **vim:** 保存对文件的修改并退出 vim。(按 esc 键然后输入:wq)
另: :q 退出 (关闭窗口) :w 保存 (写)

```
This is a sentence.
```



(5) **vim:** 输入:help :w 打开:w 命令的帮助文档

```
:help :w
-----
4. Writing
Note: When the 'write' option is off, you are not able to write any file.

:w [r]te [++opt]
    Write the whole buffer to the current file. This is
    the normal way to save changes to a file. It fails
    when the 'readonly' option is set or when there is
    another reason why the file can't be written.
    For ++opt see ++opt, but only ++bin, ++nabin, ++ff
    and ++enc are effective.

:w[r]te! [++opt]
    Like ":write", but forcefully write when 'readonly' is
    set or there is another reason why writing was
    refused.
    Note: This may change the permission and ownership of
    the file and break (symbolic) links. Add the 'u' flag
    to change the file and break (symbolic) links. Add the 'u' flag

Editing.txt [help] 80 873,8-97 50%
```

(6) vim: 输入:e 文件名 打开要编辑的文件

```
:e file.txt
This is a sentence.

"file.txt" 3L, 20C 1,19 All
```

(7) vim: 输入:ls 显示打开的缓存

```
:ls
1 # "a.txt" line 1
2 %a "file.txt" line 1
Press ENTER or type command to continue
```

(8) vim: 打开一个文件，移动光标，在第二行行头和倒数第二行行尾插入文字。

移动方式:

基本移动: hjkl (左, 下, 上, 右)

词: w (下一个词), b (词初), e (词尾)

行: 0 (行初), ^ (第一个非空格字符), \$ (行尾)

屏幕: H (屏幕首行), M (屏幕中间), L (屏幕底部)

翻页: Ctrl-u (上翻), Ctrl-d (下翻)

文件: gg (文件头), G (文件尾)

行数: : 行数 <CR> 或者行数 G (行数为行数)

杂项: % (找到配对, 比如括号或者 /* */ 之类的注释对)

查找: f 字符, t 字符, F 字符, T 字符

查找/到向前/向后在本行的字符

, / ; 用于导航匹配

搜索: /正则表达式, n / N 用于导航匹配

```
This is a sentence
\c@part=\count184
\c@section=\count185
\c@subsection=\count186
\c@subsubsection=\count187
\c@paragraph=\count188
\c@subparagraph=\count189
\c@figure=\count190
\c@table=\count191
\abovecaptionskip=\skip48
\belowcaptionskip=\skip49
\bibindent=\dimen140
}
(e:/texlive/2024/texmf-dist/tex/latex/ctex/ctex.sty
(e:/texlive/2024/texmf-dist/tex/latex/l3kernel/expl3.sty
Package: expl3 2024-02-20 L3 programming layer (loader)

(e:/texlive/2024/texmf-dist/tex/latex/l3backend/l3backend-xetex.def
File: l3backend-xetex.def 2024-02-20 L3 backend support: XeTeX
\g_graphics_track_int=\count192
\l__pdf_internal_box=\box51
\g__pdf_backend_object_int=\count193
\g__pdf_backend_annotation_int=\count194
\g__pdf_backend_link_int=\count195
))
Package: ctex 2022/07/14 v2.5.10 Chinese adapter in LaTeX (CTEX)

(e:/texlive/2024/texmf-dist/tex/latex/ctex/ctexhook.sty
Package: ctexhook 2022/07/14 v2.5.10 Document and package hooks (CTEX)
)
(e:/texlive/2024/texmf-dist/tex/latex/ctex/ctexpatch.sty
Package: ctexpatch 2022/07/14 v2.5.10 Patching commands (CTEX)
)
(e:/texlive/2024/texmf-dist/tex/latex/base/fix-cm.sty
Package: fix-cm 2020/11/24 v1.1t fixes to LaTeX
)

-

This is a sentence
aaaaaaaaaaaaaaaaaaaaaa\c@part=\count184
\c@section=\count185
\c@subsection=\count186
\c@subsubsection=\count187
\c@paragraph=\count188
\c@subparagraph=\count189
\c@figure=\count190
\c@table=\count191
\abovecaptionskip=\skip48
\belowcaptionskip=\skip49
\bibindent=\dimen140
}
(e:/texlive/2024/texmf-dist/tex/latex/ctex/ctex.sty
(e:/texlive/2024/texmf-dist/tex/latex/l3kernel/expl3.sty
Package: expl3 2024-02-20 L3 programming layer (loader)

(e:/texlive/2024/texmf-dist/tex/latex/l3backend/l3backend-xetex.def
File: l3backend-xetex.def 2024-02-20 L3 backend support: XeTeX
\g_graphics_track_int=\count192
\l__pdf_internal_box=\box51
\g__pdf_backend_object_int=\count193
\g__pdf_backend_annotation_int=\count194
\g__pdf_backend_link_int=\count195
))
Package: ctex 2022/07/14 v2.5.10 Chinese adapter in LaTeX (CTEX)

(e:/texlive/2024/texmf-dist/tex/latex/ctex/ctexhook.sty
Package: ctexhook 2022/07/14 v2.5.10 Document and package hooks (CTEX)
)
(e:/texlive/2024/texmf-dist/tex/latex/ctex/ctexpatch.sty
Package: ctexpatch 2022/07/14 v2.5.10 Patching commands (CTEX)
)
(e:/texlive/2024/texmf-dist/tex/latex/base/fix-cm.styaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
Package: fix-cm 2020/11/24 v1.1t fixes to LaTeX
)

-

```

(9) vim: 进入可视化模式选择部分内容。

可视化: v

可视化行: V

可视化块: Ctrl+v

可以用移动命令来选中。

```
This is a sentence
aaaaaaaaaaaaaaaaaaaaaa\c@part=\count184
\c@section=\count185
\c@subsection=\count186
\c@subsubsection=\count187
\c@paragraph=\count188
\c@subparagraph=\count189
\c@figure=\count190
\c@table=\count191
\abovecaptionskip=\skip48
\belowcaptionskip=\skip49
\bibindent=\dimen140
}
(e:/texlive/2024/texmf-dist/tex/latex/ctex/ctex.sty
(e:/texlive/2024/texmf-dist/tex/latex/l3kernel/expl3.sty
Package: expl3 2024-02-20 L3 programming layer (loader)

(e:/texlive/2024/texmf-dist/tex/latex/l3backend/l3backend-xetex.def
File: l3backend-xetex.def 2024-02-20 l3 backend support: XeTeX
\g_graphics_track_int=\count192
\l_pdf_internal_box=\box51
\g_pdf_backend_object_int=\count193
\g_pdf_backend_annotation_int=\count194
\g_pdf_backend_link_int=\count195
))
Package: ctex 2022/07/14 v2.5.10 Chinese adapter in LaTeX (CTEX)

(e:/texlive/2024/texmf-dist/tex/latex/ctex/ctexhook.sty
Package: ctexhook 2022/07/14 v2.5.10 Document and package hooks (CTEX)
)
(e:/texlive/2024/texmf-dist/tex/latex/ctex/ctexpatch.sty
Package: ctexpatch 2022/07/14 v2.5.10 Patching commands (CTEX)
)
(e:/texlive/2024/texmf-dist/tex/latex/base/fix-cm.styaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
Package: fix-cm 2020/11/24 v1.1t fixes to LaTeX

.
.
.
.
-- VISUAL BLOCK --
```

(10) vim: 编辑文件, 在最后一行行头删除 7 个词。(7dw)

基本编辑操作:

O / o 在之上/之下插入行

d 移动命令 删除移动命令

例如, dw 删除词, d\$ 删除到行尾, d0 删除到行头。

c 移动命令 改变移动命令

例如, cw 改变词

比如 d 移动命令 再 i

x 删除字符 (等同于 dl)

s 替换字符 (等同于 xi)

可视化模式 + 操作

选中文字, d 删除或者 c 改变

u 撤销, <C-r> 重做

y 复制 / “yank”（其他一些命令比如 d 也会复制）

p 粘贴

改变字符的大小写

```
(e:/texlive/2024/texmf-dist/tex/latex/ctex/ctexhook.sty
Package: ctexhook 2022/07/14 v2.5.10 Document and package hooks (CTEX)
)
(e:/texlive/2024/texmf-dist/tex/latex/ctex/ctexpatch.sty
Package: ctexpatch 2022/07/14 v2.5.10 Patching commands (CTEX)
)
(e:/texlive/2024/texmf-dist/tex/latex/base/flx-cm.styaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
Package: flx-cm 2020/11/24 v1.1t fixes to LaTeX
)
-
(e:/texlive/2024/texmf-dist/tex/latex/ctex/ctexpatch.sty
Package: ctexpatch 2022/07/14 v2.5.10 Patching commands (CTEX)
)
(e:/texlive/2024/texmf-dist/tex/latex/base/flx-cm.styaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
1/24 v1.1t fixes to LaTeX
)
-
```

(11) vim: 修改以下实例 fizz buzz，修复主函数没有被调用的问题。

```
def fizz_buzz(limit):
    for i in range(limit):
        if i % 3 == 0:
            print('fizz')
        if i % 5 == 0:
            print('fizz')
        if i % 3 and i % 5:
            print(i)

def main():
    fizz_buzz(10)
```

答: -G 文件尾

-o 向下打开一个新行

-输入 “if name ...”

```
def fizz_buzz(limit):
    for i in range(limit):
        if i % 3 == 0:
            print('fizz')
        if i % 5 == 0:
            print('fizz')
        if i % 3 and i % 5:
            print(i)

def main():
    fizz_buzz(10)
if __name__ == '__main__':
    main()
-
-
```

(12) vim: 继续修改实例 fizz buzz, 修复从 0 而不是 1 开始的问题。

答: -搜索 /range

-ww 向后移动两个词

-i 插入文字, “1, “

-ea 在 limit 后插入, “+1”

```
def fizz_buzz(limit):
    for i in range(1, limit+1):
        if i % 3 == 0:
            print('fizz')
        if i % 5 == 0:
            print('fizz')
        if i % 3 and i % 5:
            print(i)

def main():
    fizz_buzz(10)
if __name__ == '__main__':
    main()
```

(13) vim: 继续修改实例 fizz buzz, 在 5 的整数倍的时候打印 “buzz”。

答: 搜索 /fizz, 按 n 搜索下一个, 一直到第三个 “fizz”

键入 ci', 再将 “fizz” 改为 “buzz”

```
def fizz_buzz(limit):
    for i in range(1, limit+1):
        if i % 3 == 0:
            print('fizz')
        if i % 5 == 0:
            print('buzz')
        if i % 3 and i % 5:
            print(i)

def main():
    fizz_buzz(10)
if __name__ == '__main__':
    main()
```

(14) vim: 继续修改实例 fizz buzz, 在 15 的整数倍的时候在不同行打印 “fizz” 和 “buzz”。

答: -jj\$i 插入文字到行尾

-加入 “, end=’ ’”

-jj. 重复第二个打印

```
def fizz_buzz(limit):
    for i in range(1, limit+1):
        if i % 3 == 0:
            print('fizz', end='')
        if i % 5 == 0:
            print('buzz', end='')
        if i % 3 and i % 5:
            print(i)

def main():
    fizz_buzz(10)
if __name__ == '__main__':
    main()
```

(15) vim: 继续修改实例 fizz buzz，采用硬编码的参数 10 而不是从命令控制行读取参数。

答: gg 回到文件头，O 向上打开新行

键入内容 “import sys”，然后回到 normal 模式

键入 /10，跳到 10 文字处

键入 ci(命令，添加内容: int(sys.argv[1])

```
import sys
def fizz_buzz(limit):
    for i in range(1, limit+1):
        if i % 3 == 0:
            print('fizz', end='')
        if i % 5 == 0:
            print('buzz', end='')
        if i % 3 and i % 5:
            print(i)

def main():
    fizz_buzz(int(sys.argv[1]))
if __name__ == '__main__':
    main()
```

(16) vim: 自定义 Vim。

Vim 有一个位于 ~/.vimrc 的文本配置文件（包含 Vim 脚本命令）。

我们提供一个文档详细的基本设置。下载我们提供的 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。

答：通读注释并实践可得该设置与默认设置的区别有以下几点：

(1) 语法高亮：会根据文件的类型自动进行语法高亮。

(2) 禁用默认启动消息：启动 Vim 时，不再显示默认的启动消息。

(3) 显示行号：每一行的左侧都会显示该行的行号，这有助于快速定位到代码中的特定位置。

(4) 相对行号：当同时启用 `number` 和 `relativenumber` 时，当前行的行号显示绝对行号，而其他行则显示相对于当前行的偏移量。

(5) 始终显示状态行：即使只打开一个窗口，Vim 也会在底部显示状态行。

(6) Backspace 键行为改进：可以在任何位置使用 Backspace 键进行回退，包括插入点之前的位置。

(7) 允许隐藏未保存的缓冲区：通过 `set hidden`，可以隐藏包含未保存更改的缓冲区，而不会收到警告。

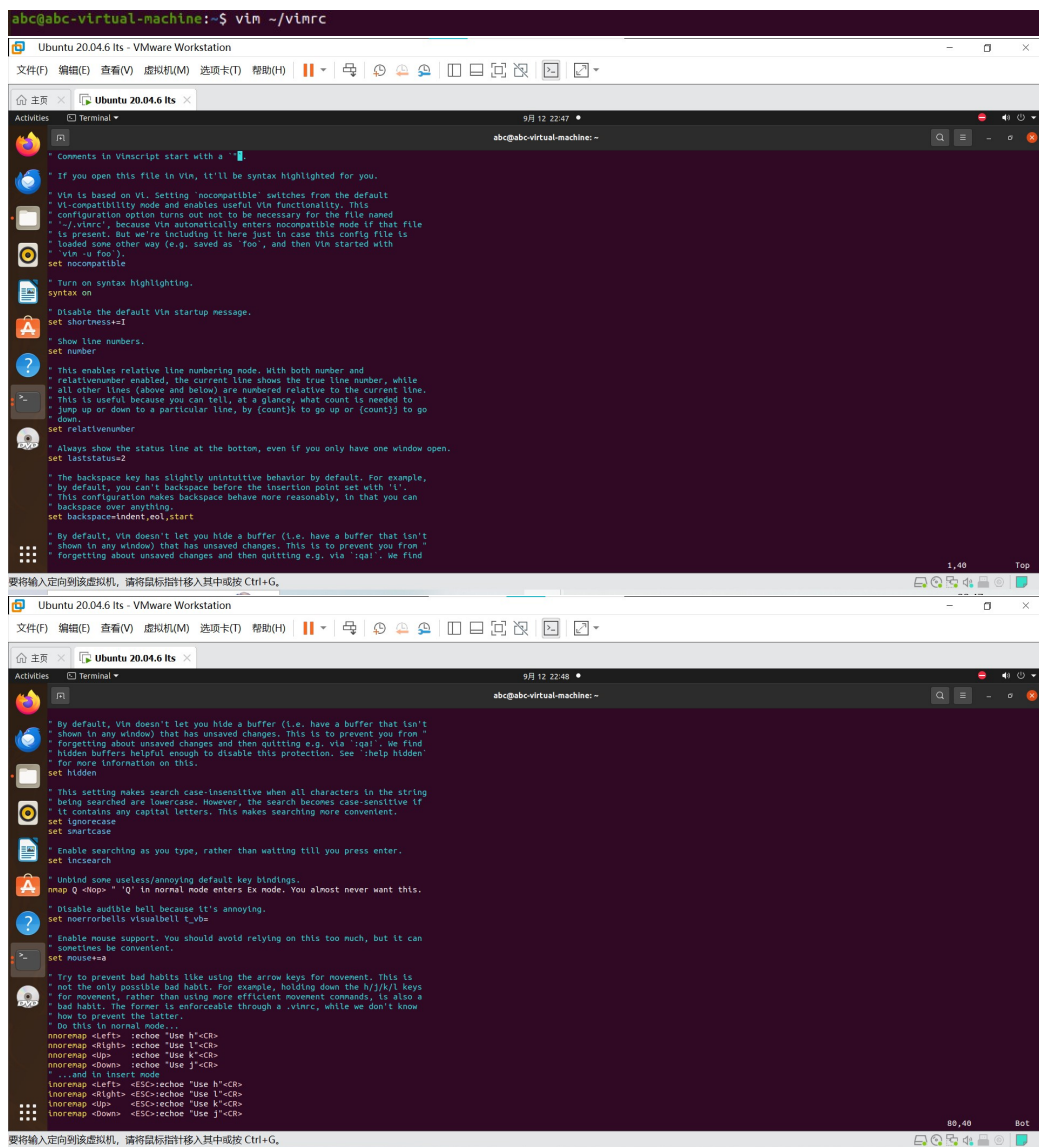
(8) 智能的搜索大小写敏感性：当搜索字符串全为小写时，搜索是大小写不敏感的；但如果搜索字符串包含大写字母，则搜索变为大小写敏感。

(9) 即时搜索：启用 `incsearch` 后，Vim 会随着您键入搜索字符串而即时显示搜索结果，而无需按 Enter 键。

(10) 重新映射方向键：通过 `nnoremap` 和 `inoremap` 命令，Vim 被配置为在尝试使用方向键时显示消息，鼓励用户使用更高效的移动命令。

(11) 禁用声音提示：`noerrorbells` 和 `visualbell` 的设置确保在出现错误时 Vim 不会发出声音，而是通过屏幕上的视觉提示来通知用户。

(12) 启用鼠标支持：`set mouse+=a` 允许用户在需要时使用鼠标进行选择 and 滚动。



(17) 数据整理：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。

答：输入 `cat /usr/share/dict/words | tr "[:upper:]" "[:lower:]" | grep -E "[a]*a)3.*$" | grep -v "'s$" | wc -l` 统计 words 文件中包含至少三个 a 且不以's 结尾的单词个数，847 个。

```
abc@abc-virtual-machine:~/Desktop$ cat /usr/share/dict/words | tr "[:upper:]" "[:lower:]" | grep -E "^[a]*a){3}.*$" | grep -v "'s$" | wc -l
847
```

(18) 数据整理： 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。

答：输入 `cat /usr/share/dict/words | tr "[:upper:]" "[:lower:]" | grep -E "[a-z]{3}.*" | grep -v "'s$" | sed -E "s/.*([a-z]{2})$/\1/" | sort | uniq -c | sort | tail -n3`，结果如图。

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

(19) 数据整理： 共存在多少种词尾两字母组合？

答：输入 `cat /usr/share/dict/words | tr "[:upper:]" "[:lower:]" | grep -E "[a-z]{3}.*" | grep -v "'s$" | sed -E "s/.*([a-z]{2})$/\1/" | sort | uniq | wc -l`，共 111 种。

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

(20) 数据整理： 多少种组合从未出现过？（首先我们要生成一个包含全部组合的列表，然后再使用上面得到的出现的组合，比较二者不同即可。）

答：写脚本生成包含全部组合的列表，再输入 `cat /usr/share/dict/words | tr "[:upper:]" "[:lower:]" | grep -E "[a-z]{3}.*" | grep -v "'s$" | sed -E "s/.*([a-z]{2})$/\1/" | sort | uniq > occurrence.txt` 得到出现的组合，然后比较两者不同 `diff -unchanged-group-format=" <(cat occurrence.txt) <(cat all.txt) | wc -l`，结果共 565 种。脚本已上传至 github。


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

abc@abc-virtual-machine:~/Desktop$ vi all.sh
abc@abc-virtual-machine:~/Desktop$ chmod 777 all.sh
abc@abc-virtual-machine:~/Desktop$ ./all.sh > all.txt
abc@abc-virtual-machine:~/Desktop$ cat all.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
cb
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
fd
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
if
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
lh
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
oj
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
rl
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
un
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
xp
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 问题及解决方案

(1) 问题：在运行脚本时报错，如下图。

```
abc@abc-virtual-machine:~/202408$ ./buggy.sh
bash: ./buggy.sh: Permission denied
```

解决方案：修改脚本文件权限（chmod 命令），再运行脚本。

```
abc@abc-virtual-machine:~/202408$ chmod 777 buggy.sh
abc@abc-virtual-machine:~/202408$ ./buggy.sh
Everything went according to plan
```

4 解题感悟

本实验中，我深入学习了 Shell 工具、Vim 编辑器以及数据整理的相关操作，不仅巩固了理论知识，还通过实践提升了技能。

在 Shell 工具方面，我通过练习题目，加深了对 Shell 命令的理解，并提升了编写脚本的能力，这对我在复杂环境中调试很有帮助。编写一个能够自动记录错误输出的 bash 脚本也让我认识到了脚本在自动化和错误追踪中的重要性。

Vim 编辑器的使用让我体验到了 vim 文本编辑器的强大与灵活。Vim 的每一个命令都体现了其设计的精妙。

数据整理部分的实验让我对文本处理工具有了更深入的了解。我学习了如何使用正则表达式和文本处理工具的组合来精确匹配和过滤数据，锻炼了我的数据处理能力。

5 github 链接

<https://github.com/zxm2580/xtkfgjjc-202408.git>