

实验报告

23020007067 李子昊

2024 年 9 月 6 日

目录

1	课后练习	1
1.1	shell 练习内容	1
1.2	vim 练习内容	4
1.3	数据整理练习内容	7
2	实例大纲	8
2.1	shell 和 vim 实例大纲	8
2.2	vim 实例	9
2.3	数据整理实例大纲	9
3	shell 与 vim 的实例展示	10
4	个人心得	20
5	github 网址	20

1 课后练习

1.1 shell 练习内容

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）

```
li@li-VMware-Virtual-Platform:~$ ls -a
.      音乐      break.sh    myfile      test1.sh
..     桌面     .cache     myfile.sh   test.sh
公共  abc.sh    case.sh    printf.sh   .viminfo
模板  algorithm.sh .config    .profile    while.sh
视频  array.sh  etc        snap
图片  .bash_history for.sh     .ssh
文档  .bash_logout function.sh .sudo_as_admin_successful
下载  .bashrc   .local     sum100.sh
li@li-VMware-Virtual-Platform:~$
```

图 1: ls -a

-文件打印以人类可以理解的格式输出

```
li@li-VMware-Virtual-Platform:~$ ls -h
公共  图片  音乐  algorithm.sh case.sh function.sh printf.sh test1.sh
模板  文档  桌面  array.sh     etc    myfile    snap    test.sh
视频  下载  abc.sh break.sh     for.sh myfile.sh sum100.sh while.sh
li@li-VMware-Virtual-Platform:~$
```

图 2: ls -t

-文件以最近访问顺序排序

```
li@li-VMware-Virtual-Platform:~$ ls -h
公共  图片  音乐  algorithm.sh case.sh function.sh printf.sh test1.sh
模板  文档  桌面  array.sh     etc    myfile    snap    test.sh
视频  下载  abc.sh break.sh     for.sh myfile.sh sum100.sh while.sh
li@li-VMware-Virtual-Platform:~$
```

图 3: ls -t

-以彩色文本显示输出结果

```
li@li-VMware-Virtual-Platform:~$ ls --color=auto
公共  图片  音乐  algorithm.sh case.sh function.sh printf.sh test1.sh
模板  文档  桌面  array.sh     etc     myfile    snap     test.sh
视频  下载  abc.sh break.sh    for.sh  myfile.sh  sum100.sh while.sh
li@li-VMware-Virtual-Platform:~$
```

图 4: ls -color=auto

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。

```
li@li-VMware-Virtual-Platform:~$ vim marco.sh
li@li-VMware-Virtual-Platform:~$ cat marco.sh
#!/bin/bash

marco(){
    echo "$(pwd)" > $HOME/marco_history.log
    echo "save pwd $(pwd)"
}

polo(){
    cd "$(cat "$HOME/marco_history.log")"
}
li@li-VMware-Virtual-Platform:~$ source marco.sh
li@li-VMware-Virtual-Platform:~$ marco
save pwd /home/li
li@li-VMware-Virtual-Platform:~$ cd
li@li-VMware-Virtual-Platform:~$ cd
li@li-VMware-Virtual-Platform:~$ polo
li@li-VMware-Virtual-Platform:~$
```

图 5: vim 编译内容和 shell 操作指令

3. 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。

```
li@li-VMware-Virtual-Platform:~$ cat buggy.sh
n=$((RANDOM % 20 ))

if [[ n -eq 12 ]]; then
    echo "Something went wrong"
    >&2 echo "The error was using magic numbers"
    exit 1
fi

li@li-VMware-Virtual-Platform:~$ cat debug.sh
count=0
echo > out.log

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

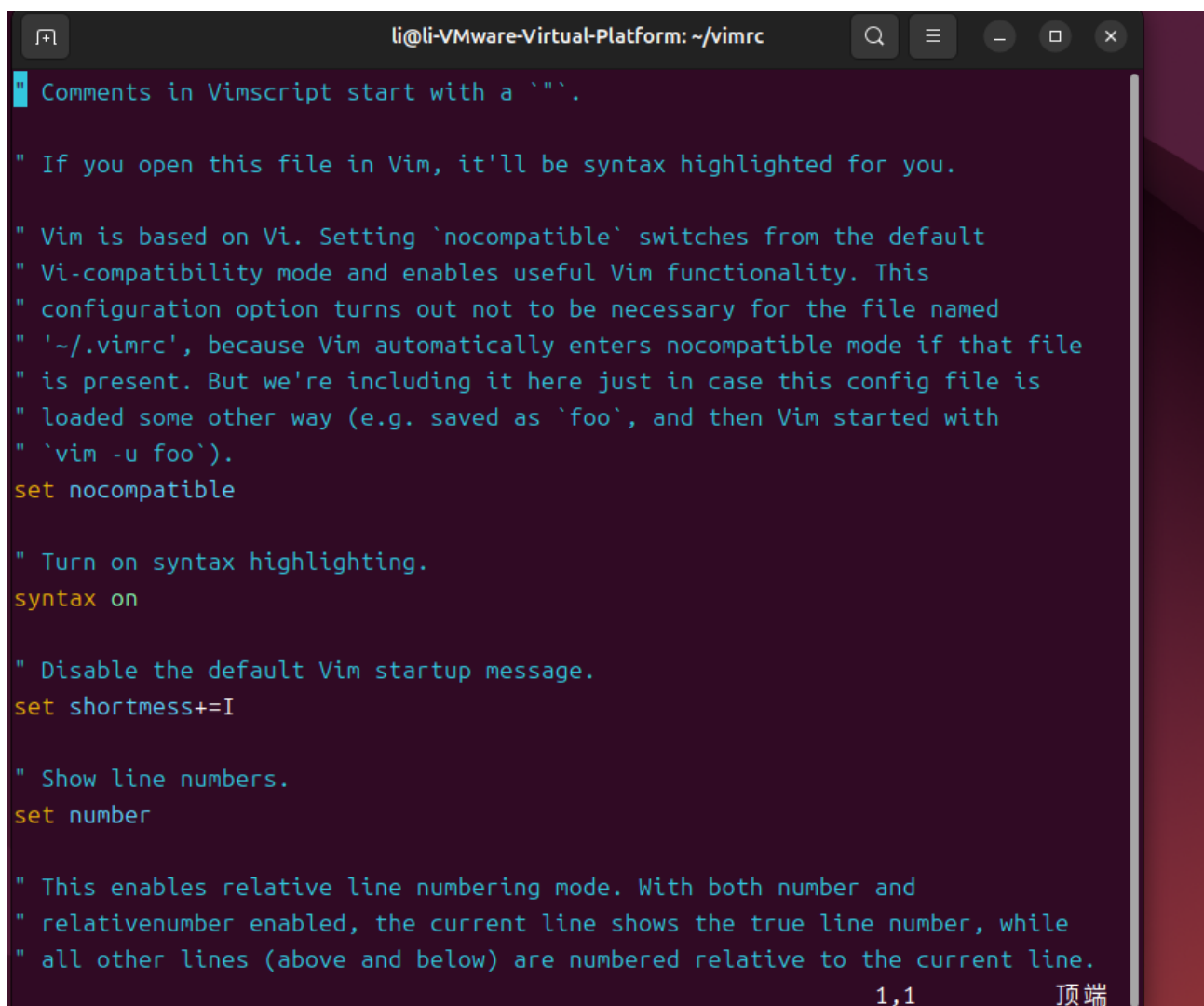
图 6: vim 编译内容

1.2 vim 练习内容

4. 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。

```
li@li-VMware-Virtual-Platform:~$ cd vimrc
li@li-VMware-Virtual-Platform:~/vimrc$ mv vimrc newfile
mv: 对 'vimrc' 调用 stat 失败: 没有那个文件或目录
li@li-VMware-Virtual-Platform:~/vimrc$ mv /home/li/下载/vimrc vimrc
li@li-VMware-Virtual-Platform:~/vimrc$ ls
vimrc
li@li-VMware-Virtual-Platform:~/vimrc$ vim vimrc
li@li-VMware-Virtual-Platform:~/vimrc$
```

图 7: shell 操作指令



```
li@li-VMware-Virtual-Platform: ~/vimrc
" 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
" relativenumber enabled, the current line shows the true line number, while
" all other lines (above and below) are numbered relative to the current line.
1,1 顶端
```

图 8: shell 操作指令

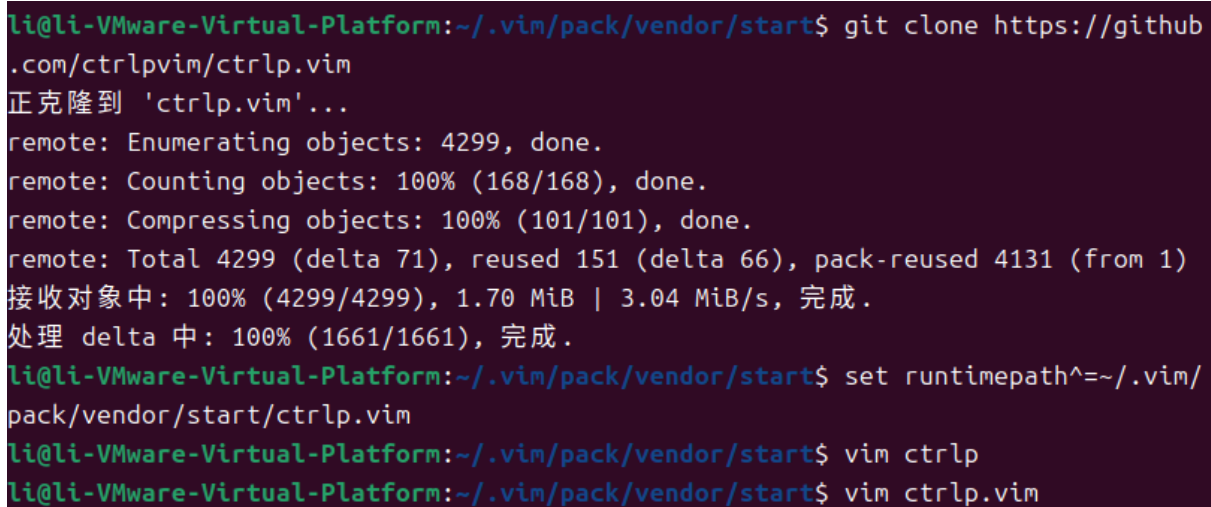
5. 安装和配置一个插件: ctrlp.vim.

5.1 用 `mkdir -p ~/.vim/pack/vendor/start` 创建插件文件夹

5.2 下载这个插件:`cd ~/.vim/pack/vendor/start; git clone https://github.com/ctrlpvim/ctrlp.vim`

5.3 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.

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

A terminal window with a dark background and light green text. The prompt is 'li@li-VMware-Virtual-Platform:~/.vim/pack/vendor/start\$'. The user enters 'git clone https://github.com/ctrlpvim/ctrlp.vim'. The output shows the cloning progress: '正克隆到 \'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)', '接收对象中: 100% (4299/4299), 1.70 MiB | 3.04 MiB/s, 完成.', '处理 delta 中: 100% (1661/1661), 完成.'. The user then enters 'set runtimepath^=~/.vim/pack/vendor/start/ctrlp.vim'. The prompt changes to 'li@li-VMware-Virtual-Platform:~/.vim/pack/vendor/start\$'. The user enters 'vim ctrlp'. The prompt changes to 'li@li-VMware-Virtual-Platform:~/.vim/pack/vendor/start\$'. The user enters 'vim ctrlp.vim'.

```
li@li-VMware-Virtual-Platform:~/.vim/pack/vendor/start$ git clone https://github
.com/ctrlpvim/ctrlp.vim
正克隆到 '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)
接收对象中: 100% (4299/4299), 1.70 MiB | 3.04 MiB/s, 完成.
处理 delta 中: 100% (1661/1661), 完成.
li@li-VMware-Virtual-Platform:~/.vim/pack/vendor/start$ set runtimepath^=~/.vim/
pack/vendor/start/ctrlp.vim
li@li-VMware-Virtual-Platform:~/.vim/pack/vendor/start$ vim ctrlp
li@li-VMware-Virtual-Platform:~/.vim/pack/vendor/start$ vim ctrlp.vim
```

图 9: shell 操作指令

```
li@li-VMware-Virtual-Platform: ~/.vim/pack/vendor/start
" =====
" Netrw Directory Listing                                (netrw v173)
"   /home/li/.vim/pack/vendor/start/ctrlp.vim
"   Sorted by      name
"   Sort sequence: [\\/]$,\\<core\\%(\\.\\d\\+\\)\\=\\>,\\.h$,\\.c$,\\.cpp$,\\~\\=\\*$,*,\\.o$,\\
"   Quick Help: <F1>:help  -:go up dir  D:delete  R:rename  s:sort-by  x:special
" =====
../
./
.git/
.github/
autoload/
doc/
plugin/
.gitignore
LICENSE
readme.md
~
~
~
~
~
~
~
"ctrlp.vim" 是目录                                1,1      全部
```

图 10: shell 操作指令

1.3 数据整理练习内容

1. 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？还有一个很有挑战性的问题：哪个组合从未出现过？

```
li@li-VMware-Virtual-Platform:~$ cat /usr/share/dict/words | tr "[:upper:]" "[:lower:]" | grep -E "^([a]*a){3}.*$" | grep -v "'s$" | wc -l
854
li@li-VMware-Virtual-Platform:~$ ^([a]*a){3}.*(?<'s)$
cat /usr/share/dict/words | tr "[:upper:]" "[:lower:]" | grep -E "^([a]*a){3}.*(?<'s)$^a){3}.*$" | grep -v "'s$" | wc -l
grep: 警告：表达式以 ? 开头
0
li@li-VMware-Virtual-Platform:~$ cat /usr/share/dict/words | tr "[:upper:]" "[:lower:]" | grep -E "^([a]*a){3}.*$" | grep -v "'s$" | sed -E "s/.*([a-z]{2})$/\1/" | sort | uniq -c | sort | tail -n3
      8 am
      8 ce
      9 ca
li@li-VMware-Virtual-Platform:~$ cat /usr/share/dict/words | tr "[:upper:]" "[:lower:]" | grep -E "^([a]*a){3}.*$" | grep -v "'s$" | sed -E "s/.*([a-z]{2})$/\1/" | sort | uniq -c | sort | tail -n3
      8 am
      8 ce
      9 ca
li@li-VMware-Virtual-Platform:~$ vim all.sh
li@li-VMware-Virtual-Platform:~$ ./all.sh > all.txt
bash: ./all.sh: 权限不够
li@li-VMware-Virtual-Platform:~$ chmod +x all.sh
找不到命令 "chmod", 您的意思是:
  "chmod" 命令来自 Debian 软件包 coreutils (9.4-2ubuntu2)
尝试 sudo apt install <deb name>
li@li-VMware-Virtual-Platform:~$ chmod +x all.sh
li@li-VMware-Virtual-Platform:~$ ./all.sh > all.txt
li@li-VMware-Virtual-Platform:~$ cat /usr/share/dict/words | tr "[:upper:]" "[:lower:]" | grep -E "^([a]*a){3}.*$" | grep -v "'s$" | sed -E "s/.*([a-z]{2})$/\1/" | sort | uniq > occurrence.txt
li@li-VMware-Virtual-Platform:~$ diff --unchanged-group-format='< (cat occurrence.txt) <(cat all.txt) | wc -l
564
```

图 11: shell 操作指令

2. 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

```
li@li-VMware-Virtual-Platform:~$ touch input.txt
li@li-VMware-Virtual-Platform:~$ sed -i.bak s/REGEX/SUBSTITUTION/ input.txt
```

图 12: shell 操作指令

2 实例大纲

2.1 shell 和 vim 实例大纲

表 1: shell 与 vim 共用的实例展示

1	for	for 循环求出 1 到 100 的和
2	case	使用 case 来进行选择分支
3	01 23...	传入参数执行文件
4	expr $a + / - b$	对数值进行简单的加减运算
5	while	用 while 循环输出 1 到 5
6	case+while	用 case 终止 while 循环
7	创建 function	创建 fun 函数
8	echo 文字 > 文件	定向输入并且覆盖原有文件
9	echo 文字 » 文件	在文件末尾后面添加文字
10	date	显示当前时间
11	-eq	判断语句的使用
12	array	创建并输出数组
13	printf	采用多种方式用 printf 输出
14	shell	传入参数执行文件
15	whereis	查找命令的二进制文件、源文件及帮助文档位置
16	cal	显示日历
17	env	显示当前所有的环境变量
18	mkdir	创建一个新的目录
19	rmdir	删除一个空目录
20	df	显示磁盘使用空间

2.2 vim 实例

表 2: vim 的实例展示大纲

1	:wq	保存然后退出
2	:e 文件名	打开要编辑的文件
3	:ls	显示打开的缓存
4	Ctrl+v	可视化块
5	x	删除字符
6	jj <i>i</i>	插入文字到行尾
7	jj.	重复第二个打印
8	:s	(替换) 命令
9	ci(改变当前括号内的内容
10	:sp / :vsp	分割窗口

2.3 数据整理实例大纲

表 3: 数据整理的实例展示

1	tr "[:upper:]" "[:lower:]"	大小写转换
2	(<i>[a]</i> * <i>a</i>)3. * [<i>s</i>]	查找一个以 a 结尾的字符串三次
3	grep -v "s"	匹配结尾为 ' s 的结果，然后取反
4	-unchanged-group-format="	两个文件中相同的内容设置为空字符串
5	sed -i.bak s/REGEX/SUBSTITUTION/ input.txt	自动创建一个后缀为.bak 的备份文件

3 shell 与 vim 的实例展示

用 vim 编译，然后用 cat 查看文件内容，再用 bash 运行程序。

1.for 循环求出 1 到 100 的和。

```
li@li-VMware-Virtual-Platform:~$ vim sum100.sh
li@li-VMware-Virtual-Platform:~$ cat sum100.sh
sum=0
for i in {1..100}
do
    sum=$((sum+i))
done
echo "1到100的和为$sum"
li@li-VMware-Virtual-Platform:~$ bash sum100.sh
1到100的和为5050
```

图 13: for 循环

2. 使用 case 来进行选择分支。

```
li@li-VMware-Virtual-Platform:~$ vim case.sh
li@li-VMware-Virtual-Platform:~$ cat case.sh
echo "请输入一个1-9的数字： "
read num
case $num in
    [1,9])
        echo "you are right!"
        ;;
    *)
        echo "you are wrong"
        ;;
esac
li@li-VMware-Virtual-Platform:~$ bash case.sh
请输入一个1-9的数字：
10
you are wrong
```

图 14: case 示例

3. 传入参数执行文件。

```
li@li-VMware-Virtual-Platform:~$ vim test.sh
li@li-VMware-Virtual-Platform:~$ cat test.sh
#!/bin/bash
echo "Shell传参实例： ";
echo "可执行文件名:$0";
echo "第一个参数： $1";
echo "第二个参数： $2";
echo "第三个参数： $3";
li@li-VMware-Virtual-Platform:~$ chmod +x test.sh
li@li-VMware-Virtual-Platform:~$ ./test.sh 1 2 3
Shell传参实例：
可执行文件名:./test.sh
第一个参数： 1
第二个参数： 2
第三个参数： 3
li@li-VMware-Virtual-Platform:~$
```

图 15: shell 传参实例

4. 对数值进行简单的加减运算。

```

li@li-VMware-Virtual-Platform:~$ vim algorithm.sh
li@li-VMware-Virtual-Platform:~$ cat algorithm.sh
#!/bin/bash
a=10
b=20

val=`expr $a + $b`
echo "a + b = $val"

val=`expr $a - $b`
echo "a - b = $val"
li@li-VMware-Virtual-Platform:~$ chmod +x algorithm.sh
li@li-VMware-Virtual-Platform:~$ ./algorithm.sh
a + b = 30
a - b = -10

```

图 16: 基本运算

5. 用 while 循环输出 1 到 5。

```

li@li-VMware-Virtual-Platform:~$ vim while.sh
li@li-VMware-Virtual-Platform:~$ cat while.sh
#!/bin/bash
num=1
while(($num<=5))
do
    echo $num
    let "num++"
done
li@li-VMware-Virtual-Platform:~$ chmod +x while.sh
li@li-VMware-Virtual-Platform:~$ ./while.sh
1
2
3
4
5
li@li-VMware-Virtual-Platform:~$

```

图 17: while 循环

6. 用 case 终止 while 循环。

```
li@li-VMware-Virtual-Platform:~$ vim break.sh
li@li-VMware-Virtual-Platform:~$ cat break.sh
while :
do
    echo "请输入一个数字"
    read num
    case $num in
        1|2|3) echo "please input again"
                ;;
        *) echo "return"
            break
            ;;
    esac
done
li@li-VMware-Virtual-Platform:~$ chmod +x break.sh
li@li-VMware-Virtual-Platform:~$ ./break.sh
请输入一个数字
1
please input again
请输入一个数字
4
return
li@li-VMware-Virtual-Platform:~$
```

图 18: while+case 判断

7. 创建 fun 函数

```
li@li-VMware-Virtual-Platform:~$ vim function.sh
li@li-VMware-Virtual-Platform:~$ chmod +x function.sh
li@li-VMware-Virtual-Platform:~$ cat function.sh
#!/bin/bash
fun(){
    echo "函数"
    echo "函数结束"
}
fun
li@li-VMware-Virtual-Platform:~$ ./function.sh
函数
函数结束
li@li-VMware-Virtual-Platform:~$
```

图 19: 创建函数

8. 定向输入并且覆盖原有文件

```
li@li-VMware-Virtual-Platform:~$ echo "输出定向且覆盖原有文件" > abc.sh
li@li-VMware-Virtual-Platform:~$ cat abc.sh
输出定向且覆盖原有文件
li@li-VMware-Virtual-Platform:~$ echo "123" >abc.sh
li@li-VMware-Virtual-Platform:~$ cat abc.sh
123
li@li-VMware-Virtual-Platform:~$
```

图 20: 覆盖输入

9. 在文件末尾后面添加文字。

```
li@li-VMware-Virtual-Platform:~$ echo "123" >abc.sh
li@li-VMware-Virtual-Platform:~$ cat abc.sh
123
li@li-VMware-Virtual-Platform:~$ echo "输出定向且不覆盖原有文件" >> abc.sh
li@li-VMware-Virtual-Platform:~$ cat abc.sh
123
输出定向且不覆盖原有文件
li@li-VMware-Virtual-Platform:~$
```

图 21: 不覆盖输入

10. 显示当前时间。

```
2024年 08月 31日 星期六 20:25:21 CST
li@li-VMware-Virtual-Platform:~$ echo "现在的时间为：`date`"
现在的时间为：2024年 08月 31日 星期六 20:25:52 CST
li@li-VMware-Virtual-Platform:~$
```

图 22: 显示时间

11. 判断语句的使用。

```
li@li-VMware-Virtual-Platform:~$ vim test1.sh
li@li-VMware-Virtual-Platform:~$ cat test1.sh
num1=100
num2=100
if test ${num1} -eq ${num2}
then
    echo "两个数相等"
else
    echo "不相等"
fi
li@li-VMware-Virtual-Platform:~$ chmod +x test1.sh
li@li-VMware-Virtual-Platform:~$ ./test1.sh
两个数相等
li@li-VMware-Virtual-Platform:~$
```

图 23: 判断语句

12. 创建并输出数组。

```
li@li-VMware-Virtual-Platform:~$ vim array.sh
li@li-VMware-Virtual-Platform:~$ cat array.sh
#!/bin/bash
array=(A B C D)
echo "the first element:${array[0]}"
echo "the second element:${array[1]}"
echo "the third element:${array[2]}"
echo "the fourth element:${array[3]}"
li@li-VMware-Virtual-Platform:~$ chmod +x array.sh
li@li-VMware-Virtual-Platform:~$ ./array.sh
the first element:A
the second element:B
the third element:C
the fourth element:D
li@li-VMware-Virtual-Platform:~$
```

图 24: 数组

13. 采用多种方式用 printf 输出

```
li@li-VMware-Virtual-Platform:~$ printf "hello world\n"
hello world
li@li-VMware-Virtual-Platform:~$ vim printf.sh
li@li-VMware-Virtual-Platform:~$ cat printf.sh
printf "%-10s %-3d %c %-4.2f\n" 你好 20 a 4.234
li@li-VMware-Virtual-Platform:~$ chmod +x printf.sh
li@li-VMware-Virtual-Platform:~$ ./printf.sh
你好      20  a 4.23
li@li-VMware-Virtual-Platform:~$
```

图 25: printf 输出

14. 传入参数执行文件。

```
li@li-VMware-Virtual-Platform:~$ vim test.sh
li@li-VMware-Virtual-Platform:~$ cat test.sh
#!/bin/bash
echo "Shell传参实例： ";
echo "可执行文件名:$0";
echo "第一个参数： $1";
echo "第二个参数： $2";
echo "第三个参数： $3";
li@li-VMware-Virtual-Platform:~$ chmod +x test.sh
li@li-VMware-Virtual-Platform:~$ ./test.sh 1 2 3
Shell传参实例：
可执行文件名:./test.sh
第一个参数： 1
第二个参数： 2
第三个参数： 3
li@li-VMware-Virtual-Platform:~$
```

图 26: shell 传参实例

15. 查找命令的二进制文件、源文件及帮助文档位置

```
li@li-VMware-Virtual-Platform:~$ whereis ls
ls: /usr/bin/ls /usr/share/man/man1/ls.1.gz
```

图 27: whereis

16. 显示日历

```
正在处理用于 GNOME 3 (27.12.0 - 100.0.0) 的触发器...
li@li-VMware-Virtual-Platform:~$ cal
      九月 2024
日 一 二 三 四 五 六
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30
```

图 28: cal

17. 显示当前所有的环境变量

```
li@li-VMware-Virtual-Platform:~$ env
SHELL=/bin/bash
SESSION_MANAGER=local/li-VMware-Virtual-Platform:@/tmp/.ICE-unix/1669,unix/li-VMware-Virtual-Platform:/tmp/.ICE-unix/1669
QT_ACCESSIBILITY=1
COLORTERM=truecolor
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/etc/xdg
XDG_MENU_PREFIX=gnome-
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
GNOME_SHELL_SESSION_MODE=ubuntu
SSH_AUTH_SOCK=/run/user/1000/keyring/ssh
MEMORY_PRESSURE_WRITE=c29tZSAyMDAwMDAgMjAwMDAwMAA=
XMODIFIERS=@im=ibus
DESKTOP_SESSION=ubuntu
GTK_MODULES=gail:atk-bridge
PWD=/home/li
LOGNAME=li
XDG_SESSION_DESKTOP=ubuntu
XDG_SESSION_TYPE=wayland
SYSTEMD_EXEC_PID=1698
XAUTHORITY=/run/user/1000/.mutter-Xwaylandauth.C12UT2
HOME=/home/li
USERNAME=li
```

图 29: env

18. 创建一个新的目录

```
li@li-VMware-Virtual-Platform:~$ mkdir asd
li@li-VMware-Virtual-Platform:~$ cd asd
li@li-VMware-Virtual-Platform:~/asd$ cd ..
li@li-VMware-Virtual-Platform:~$
```

图 30: mkdir

19. 删除一个空目录

```
li@li-VMware-Virtual-Platform:~/asd$ cd ..
li@li-VMware-Virtual-Platform:~$ rmdir asd
li@li-VMware-Virtual-Platform:~$ cd asd
bash: cd: asd: 没有那个文件或目录
li@li-VMware-Virtual-Platform:~$
```

图 31: rmdir

20. 显示磁盘使用空间

```
li@li-VMware-Virtual-Platform:~$ df
文件系统      1K的块    已用    可用  已用% 挂载点
tmpfs          396104    2048   394056    1% /run
/dev/sda2     10215700 6593928 3081256   69% /
tmpfs         1980516      0 1980516    0% /dev/shm
tmpfs          5120      8    5112    1% /run/lock
tmpfs         396100    124   395976    1% /run/user/1000
/dev/sr0        90140   90140      0 100% /media/li/CDROM
/dev/sr1       6057964 6057964      0 100% /media/li/Ubuntu 24.04.1 LTS amd64
li@li-VMware-Virtual-Platform:~$
```

图 32: df

4 个人心得

使用 Shell 脚本的最大优势在于其强大而灵活的系统控制能力，尤其在 Unix 和 Linux 系统上表现突出。它提供了一种简洁易用的方式，能够将复杂的系统任务如文件处理、网络管理、系统维护自动化。Shell 脚本几乎都是系统命令的组合，简单易学，特别适合小型任务，如批量处理文件或执行系统管理任务，并且具备良好的跨平台兼容性。

然而，Shell 脚本的可读性和维护性是一个值得关注的方面。编写时应注重使用有意义的变量名、结构化代码、以及详细的注释，尤其在复杂的逻辑处理中，这可以帮助后续维护和理解。同时，Shell 脚本的调试相对较弱，建议在脚本中加入错误处理和日志记录，使用 `set -e` 和 `trap` 等命令，避免因简单错误导致脚本崩溃。

对于性能优化，Shell 脚本在处理复杂或大规模任务时，可能表现不如其他编程语言高效。为此，尽量使用内建命令，如 `awk`、`sed`、`grep`，并减少不必要的外部命令调用和文件读写。Shell 脚本在系统管理、批量操作、自动化部署等场景中表现优秀，但在性能要求高的复杂任务中，建议结合其他语言使用。

5 github 网址

<https://github.com/newbeginnerlzh/git-task01.git>