



上海大学

SHANGHAI UNIVERSITY

## 操作系统（一）实验报告

组 号 第 7 组

学 号 姓 名 20121034 胡才郁

实 验 序 号 6

日 期 2022 年 10 月 27 日

# 实验六 SHELL 编程

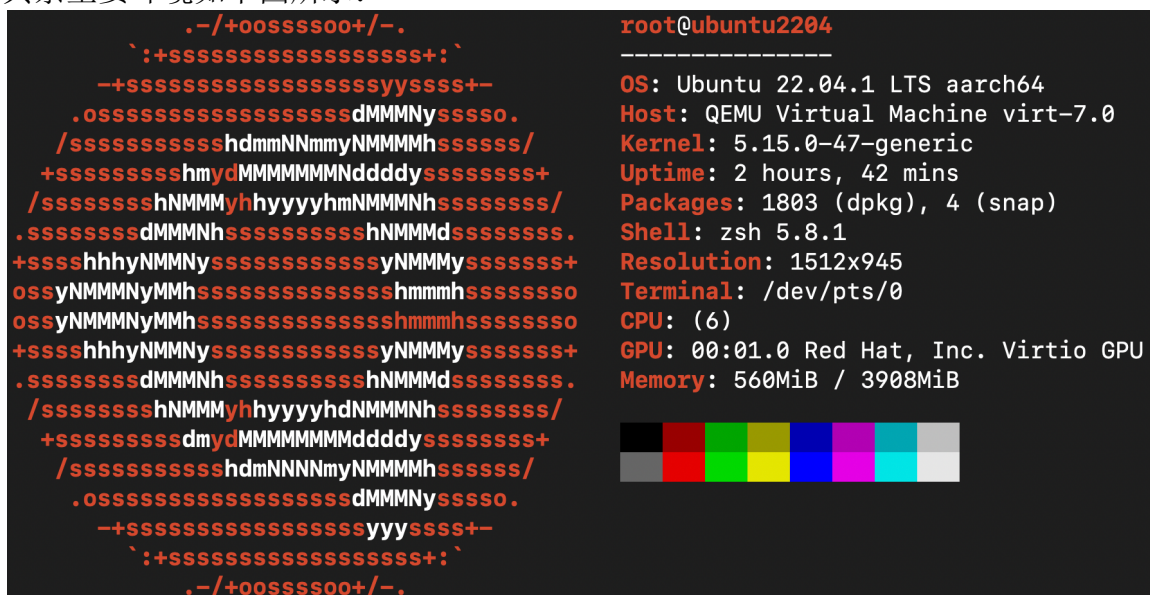
## 1 实验目的与要求

1. 掌握 vi 的三种工作方式，熟悉 vi 编辑程序的使用。
2. 学习 Shell 程序设计方法。掌握编程要领。

## 2 实验环境

- 操作系统: Ubuntu 22.04
- 宿主机: QEMU Virtual Machine

其余主要环境如下图所示：



```
root@ubuntu2204
-----
OS: Ubuntu 22.04.1 LTS aarch64
Host: QEMU Virtual Machine virt-7.0
Kernel: 5.15.0-47-generic
Uptime: 2 hours, 42 mins
Packages: 1803 (dpkg), 4 (snap)
Shell: zsh 5.8.1
Resolution: 1512x945
Terminal: /dev/pts/0
CPU: (6)
GPU: 00:01.0 Red Hat, Inc. Virtio GPU
Memory: 560MiB / 3908MiB

.-/+00SSSS00+/- .
`:+SSSSSSSSSSSSSSSS+:`
-+SSSSSSSSSSSSSSSSyySSSS+-
.OSSSSSSSSSSSSSSSSdMMMMySSSSO.
/SSSSSSSSSSShdmmNNmmyNMMMMhSSSSS/
+SSSSSSSSShmydMMMMMMNdddySSSSSSS+
/SSSSSSSSShNMMMyhhyyyyhmNMMNhSSSSSSS/
.SSSSSSSdMMMNhSSSSSSSSShNMMMdSSSSSSS.
+SSShhhyNMMNySSSSSSSSSSyNMMMySSSSSSS+
oSSyNMMMNyMMhSSSSSSSSSSShmmmhSSSSSSO
oSSyNMMMNyMMhSSSSSSSSSSShmmmhSSSSSSO
+SSShhhyNMMNySSSSSSSSSSyNMMMySSSSSSS+
.SSSSSSSdMMMNhSSSSSSSSShNMMMdSSSSSSS.
/SSSSSSSSShNMMMyhhyyyyhdNMMNhSSSSSSS/
+SSSSSSSSdmydMMMMMMNdddySSSSSSS+
/SSSSSSSSShdmNNNNmyNMMMMhSSSSSS/
.OSSSSSSSSSSSSSSSSdMMMMySSSSO.
-+SSSSSSSSSSSSSSSSyySSSS+-
`:+SSSSSSSSSSSSSSSS+:`
.-/+00SSSS00+/- .
```

图 1. 实验环境

## 3 实验内容及其设计与实现

问题一：按本《实验指导》第三部分的内容。熟悉 vi 的三种工作方式。熟悉使用各种编辑功能。

思考题：试一试 vi 的三种工作方式各用在何时？用什么命令进入插入方式？怎样退出插入方式？文件怎样存盘？注意存盘后的提示信息。

答：vi 或 vim 主要分为命令模式、末行模式、编辑模式三种，灵活在三种模式之间切换来完成文本编辑工作。

①命令模式：

- 1) 在该模式中，可以输入命令来执行许多种功能；
- 2) 打开文件首先进入命令模式，它是使用 vim 编辑器的入口

②末行模式：

- 1) 将文件保存或退出 vi，也可以设置编辑环境，如寻找字符串、列出行号等；
- 2) 末行模式是 vim 编辑器对的出口，要退出 vim，必须要在末行模式下。

③编辑模式：可以对文本进行编辑操作。

Vi 或 Vim 的各种快捷键十分复杂，对于 Vim 各种操作的各种总结有很多，其中最著名的应当是 Vim keymap，如下图，对应每一个键位对应的操作。

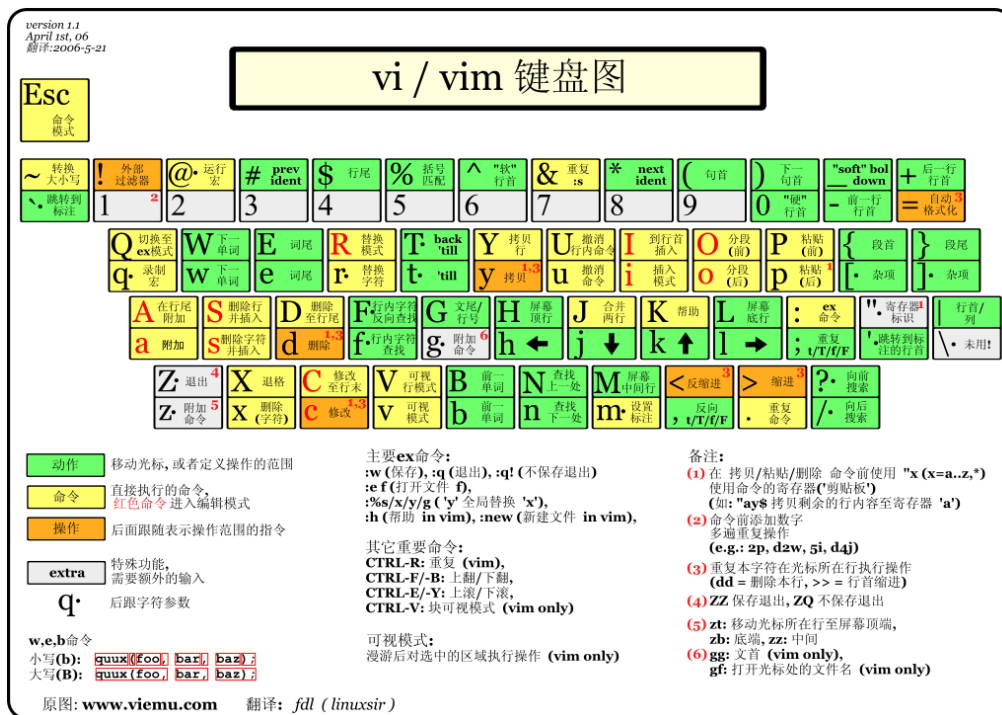


图 2. Vim Keymap

Vim 一定是所有 Linux 程序员都会经历的一个坎，曾经看到过一个笑话：

Q: 如何生成一个随机字符串?  
A: 让 Vim 新手退出 Vim.

只能说十分的生动形象了...Vim 确实对于新手不太友好，但是熟练使用 Vim 后会对于工作效率大幅度提升，不过学习的过程可能会异常复杂。由于 Vim 的十分灵活，可以自定义配置各种快捷键以及大量的插件支持，至少就我而言，尝试过使用 [Vimtutor](#) 来练习 Vim 的操作，但是对于 Vim 还是用不来，Vscod+远程 SSH 连接的方式还是更适合我。

## 问题二：创建和执行 Shell 程序

用前面介绍的 Vi 或其他文本编辑器编写 Shell 程序，并将文件以文本文件方式保存在相应的目录中。用 chmod 将文件的权限设置为可执行模式，如若文件名为 shdemo.h,则命令如下：

```
$ chmod 755 shdemo.h （文件主可读、写、执行，同组人和其他人可读和执行）
在提示符后执行 Shell 程序：
$ shdemo.h （直接键入程序文件名执行）
或 $ sh shdemo.h （执行 Shell 程序）
或 $ .shdemo.h （没有设置权限时可用点号引导）
```

答：可以使用 `umask -S` 指令查看默认文件、文件夹的权限。其中的-S 选项输出符号选项，帮助更清晰了解文件权限。通过下图输出可知，默认的文件权限为文件拥有者具有读、写、执行权限，同组用户具有读、执行，非同组的其他用户具有读、执行权限。

```
> umask -S
u=rwx,g=rx,o=rx
```

图 3. 查看默认权限

尽管在 Linux 操作系统下万物皆文件，但是目录(d)和文件(-)的三种权限之间可以允许的操作是不同的，具体而言如下表所示：

表 1. 文件与目录不同权限的操作

权限	文件	目录
读取权限 r	具有读取文件内容的权限	具有浏览目录与子目录的权限
写入权限 w	具有修改文件内容的权限	具有增加和删除目录内文件的权限
执行权限 x	具有执行文件的权限	具有进入目录的权限

可以将目录类比为抽屉，文件类比为抽屉中的日记本，如果目录具有读取权限 r，而文件没有，就相当于可以打开抽屉看到一个日记本，知道这些日记本的名字，知道日记本是谁的，但由于没有读取文件内容的权限，不能打开日记本阅读。

可能是出于安全的考虑，Linux 系统新建文件默认没有执行权限，其最大权限为 `rw-rw-rw-`（666），新建目录的最大权限为 `rwx-rwx-rwx`（777）。而默认的权限与 `umask` 命令有关。修改文件的权限之前，并没有权限执行新创建的 `shdemo.sh` 文件。使用 `chmod` 命令修改，加入权限类型后，即可执行。如下图所示：

```
> ./shdemo.sh
zsh: permission denied: ./shdemo.sh
> ls
total 12K
drwxr-xr-x 2 root root 4.0K Oct 27 01:02 .
drwxr-xr-x 9 root root 4.0K Oct 27 00:50 ..
-rw-r--r-- 1 root root 52 Oct 27 01:02 shdemo.sh
> chmod 755 shdemo.sh
> ls
total 12K
drwxr-xr-x 2 root root 4.0K Oct 27 01:02 .
drwxr-xr-x 9 root root 4.0K Oct 27 00:50 ..
-rwxr-xr-x 1 root root 52 Oct 27 01:02 shdemo.sh
> ./shdemo.sh
This text is for the question 2!
```

图 4. 权限修改前后运行情况

问题三：用 vi 编写《实验指导》“第四部分 Shell 程序设计”中的例 1，练习内部变量和位置参数的用法。

用 chmod 将文件的权限设置为可执行模式，并在提示符后键入命令行：

`./prog1.sh` 或 `$sh prog1.sh` #没有参数

屏幕显示：

Name not provided

在提示符后键入命令行：

`./prog1.sh Theodore` #有一个参数

屏幕显示：

Your name is Theodore #引用\$1 参数的效果

答：针对本问题，`$#`为内部变量，内部变量是 Linux 提供的一种特殊变量，具体而言`$#`表示传送给 Shell 程序的位置参数的个数。

```
#!/bin/bash
if [ $# == 0 ]
then
    echo "Name not provided."
else
    echo "Your name is " $1
fi
```

图 5. 问题三代码

与这个问题的代码完成相比，更值得讨论的是执行 Shell 脚本的方法。根据实验指导书的写法，实际上分为了 3 种运行 Shell 脚本的方式，分别为

1. `./` 相对路径引用
2. `sh` 用 `sh` 执行
3. 可执行文件名

观察下图分别为 1、2、3 的结果可以观察到，对于上述代码，只有第一种方式可以正常执行，而第 2 种直接报错 `unexpected operator`，第 3 种报错 `command not found`。关于其中的原因，接下来进行解释。

```
> ./prog1.sh
Name not provided.
> ./prog1.sh Tom
Your name is Tom
> sh ./prog1.sh
./prog1.sh: 2: [: 0: unexpected operator
Your name is
> sh ./prog1.sh Tom
./prog1.sh: 2: [: 1: unexpected operator
Your name is Tom
> prog1.sh
zsh: command not found: prog1.sh
```

图 6. 三种方式运行 sh 文件

第一种情况使用./的方式执行可执行文件，观察代码第一行的#!/bin/bash，#在 shell 脚本种用作注释行，但是在第一行是一个例外，#后面的!会告诉 shell 用哪一个 shell 来执行脚本，在本例中是/bin 目录下的 bash。这个操作也就意味着在命令行敲下./prog.sh 时，这个命令行在我的实验环境中，此处为 zsh，他作为父进程，创建一个子进程/bin/bash 来运行我的脚本。

而第二种情况相当于使用二进制文件 sh 来执行脚本文件 prog1.sh，至于为什么在此处的运行结果会出错，观察下图：

```
> echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
> which sh
/usr/bin/sh
> readlink -f /usr/bin/sh
/usr/bin/dash
```

图 7. 寻找 sh 根源位置

当我在 shell 中敲下命令 sh 时，系统会优先在环境变量中有无名字为 sh 的文件，再在当前目录下寻找有无名字为 sh 的文件。这里使用 which 命令查看 sh 的绝对路径，发现路径为/usr/bin/sh，不过，这里还存在着软链接、硬链接等类似于“快捷方式”的问题，因此使用 readlink -f 命令查找到 sh 的最终指向为/usr/bin/dash。

到这里，第二种方式会报错的原因就比较显然了。此处相当于使用 dash 这个 Shell 来执行 prog1.h 这个文件，而不同的 shell 之间语法有差异，不可以完全兼容，具体到此处，在 dash 中，bash 的中括号语法[]在 dash 中不兼容，因此有了之前的报错 unexpected operator。

而第三种方式直接敲下 prog1.sh 时，系统会认为这是命令，自然出错。

综上，最建议的运行脚本方式是使用第一种方式./ 相对路径引用，如此使用可以在不清楚这个脚本的前提下保证不因为 shell 选择而出错。其次在清楚此脚本是由何种脚本语言编写的情况下，使用类似 bash prog1.sh 与 zsh prog.sh 此类方式执行。

**问题四：进一步修改程序 prog1.h，要求显示参数个数、程序名字，并逐个显示参数。**

**答：**此题主要是对 \$ 特殊参数以及循环编写的考察，将没有传入参数与传入参数的执行逻辑分开，代码如下图所示：

```
#!/bin/bash
echo "number of parameters:" $#
echo "Your file name is " $0
if [ $# == 0 ]
then
    echo "Name is not provided"
else
    for VAR in $*
    do
        echo $VAR
    done
fi
```

图 8. 问题四代码

运行结果如下图，标号 1 为无参数传入情况，标号 2 为有参数传入情况。

```
> ./prog1_4.sh
number of parameters: 0
Your file name is ./prog1_4.sh
Name is not provided
> ./prog1_4.sh 1 2 3 4
number of parameters: 4
Your file name is ./prog1_4.sh
1
2
3
4
```

图 9. 问题四运行结果

问题五：修改例 1 程序（即上面的 prog1.h），用 read 命令接受键盘输入。若没有输入显示第一种提示，否则第二种提示。

答：运行结果如下图所示，标号 1 为无参数传入情况，标号 2 为有参数传入情况。

```
> ./prog1_5.sh
Please input your name
Name is not provided
> ./prog1_5.sh
Please input your name
HCY
Your name is HCY
```

图 10. 问题五运行结果

问题六：用 vi 编写《实验指导》“第四部分 Shell 程序设计”中的例 2、例 3，练习字符串比较运算符、数据比较运算符和文件运算符的用法，观察运行结果。

答：例 2 部分：

```
#!/bin/bash
string1="The first one"
string1="The second one"
if [ string1 == string2 ]
then
    echo "string1 equal to string2"
else
    echo "string1 not equal to string2"
fi

if [ string1 ]
then
    echo "string1 is not empty"
else
    echo "string1 is empty"
fi

if [ -n string2 ]
then
    echo "string2 has a length grater than zero"
else
    echo "string2 has a length equal to zero"
fi
```

```
> ./prog6.sh
string1 not equal to string2
string1 is not empty
string2 has a length grater than zero
```

图 11. 例 2 代码与运行结果



例 3 部分：由于在我的实验环境下为 root 用户，拥有 write 权限，因此第 4 行输出与指导书上不同。

```
#!/bin/bash
if [ -d cppdir ]; then
    echo "cppdir is a directory"
else
    echo "cppdir is not a directory"
fi
if [ -f filea ]; then
    echo "filea is a regular file"
else
    echo "filea is not a regular file"
fi
if [ -r filea ]; then
    echo "filea has read permissione"
else
    echo "filea dose not have read permissione"
fi
if [ -w filea ]; then
    echo "filea has write permissione"
else
    echo "filea dose not have write permissione"
fi
if [ -x cppdir ]; then
    echo "cppdir has execute permissione"
else
    echo "cppdir dose not have execute permissione"
fi
```

```
> ./prog6_2.sh
cppdir is a directory
filea is a regular file
filea has read permissione
filea has write permissione
cppdir has execute permissione
```

图 12. 例 3 代码与运行结果

问题七：修改例 2 程序，使在程序运行中能随机输入字符串，然后进行字符串比较。

答：以下是比较字符串时需要注意的几点：

1. 必须在二元运算符和操作数之间使用空格。
2. 始终在变量名称周围使用双引号以避免任何单词拆分或通配问题。
3. Bash 不按“类型”隔离变量，变量根据上下文被视为整数或字符串。

```
#!/bin/bash
read -p "please input string1: " -r string1
read -p "please input string2: " -r string2
if [ "$string1" == "$string2" ]; then
    echo "string1 equal to string2"
else
    echo "string1 not equal to string2"
fi
if [ "$string1" ]; then
    echo "string1 is not empty"
else
    echo "string1 is empty"
fi
if [ -n "$string2" ]; then
    echo "string2 has a length greater than zero"
else
    echo "string2 has a length equal to zero"
fi
```

```
> ./prog7.sh
please input string1: abc
please input string2: abcde
string1 not equal to string2
string1 is not empty
string2 has a length greater than zero
```

图 13. 问题七代码与运行结果



问题八：修改例 3 程序，使在程序运行中能随机输入文件名，然后进行文件属性判断。

答：两种类型分别进行测试：

```
#!/bin/bash
read -p "Please input the file path: " -r file
if [ -d $file ]; then
    echo "$file is a directory"
else
    echo "$file is not a directory"
fi
if [ -f $file ]; then
    echo "$file is a regular file"
else
    echo "$file is not a regular file"
fi
if [ -r $file ]; then
    echo "$file has read permissione"
else
    echo "$file dose not have read permissione"
fi
if [ -w $file ]; then
    echo "$file has write permissione"
else
    echo "$file dose not have write permissione"
fi
if [ -x $file ]; then
    echo "$file has execute permissione"
else
    echo "$file dose not have execute permissione"
fi
```

```
> ./prog8.sh
Please input the file path: filea
filea is not a directory
filea is a regular file
filea has read permissione
filea has write permissione
filea has execute permissione
> ./prog8.sh
Please input the file path: cppdir
cppdir is a directory
cppdir is not a regular file
cppdir has read permissione
cppdir has write permissione
cppdir has execute permissione
```

图 14. 问题八代码与运行结果

问题九：用 vi 编写《实验指导》“第四部分 Shell 程序设计”中的例 4、例 5、例 6、例 7，掌握控制语句的用法，观察运行结果。

答：例四：

```
#!/bin/bash
for filename in $(ls); do
    cp $filename backup/$filename
    if [ $? -ne 0 ]; then
        echo "copy $filename failed"
    fi
done
```

```
> ./prog9_1.sh
cp: -r not specified; omitting directory 'backup'
./prog9_1.sh: line 4: [: -ne: binary operator expected
cp: -r not specified; omitting directory 'cppdir'
./prog9_1.sh: line 4: [: -ne: binary operator expected
./prog9_1.sh: line 4: [: -ne: binary operator expected
./prog9_1.sh: line 4: [: -ne: binary operator expected
./prog9_1.sh: line 4: [: -ne: binary operator expected
./prog9_1.sh: line 4: [: -ne: binary operator expected
./prog9_1.sh: line 4: [: -ne: binary operator expected
./prog9_1.sh: line 4: [: -ne: binary operator expected
./prog9_1.sh: line 4: [: -ne: binary operator expected
./prog9_1.sh: line 4: [: -ne: binary operator expected
./prog9_1.sh: line 4: [: -ne: binary operator expected
./prog9_1.sh: line 4: [: -ne: binary operator expected
./prog9_1.sh: line 4: [: -ne: binary operator expected
./prog9_1.sh: line 4: [: -ne: binary operator expected
./prog9_1.sh: line 4: [: -ne: binary operator expected
./prog9_1.sh: line 4: [: -ne: binary operator expected
./prog9_1.sh: line 4: [: -ne: binary operator expected
./prog9_1.sh: line 4: [: -ne: binary operator expected
./prog9_1.sh: line 4: [: -ne: binary operator expected
```

图 15. 例四代码与运行结果

例五:

```
#!/bin/bash
loopcount=0
result=0
while [ $loopcount -lt 10 ]
do
    loopcount=$((loopcount + 1))
    result=$((result + (loopcount * 2)))
done
echo "result is $result"
```

图 16. 例五代码

例六:

```
#!/bin/bash
loopcount=0
result=0
until [ $loopcount -ge 10 ]; do
    loopcount=$((loopcount + 1))
    result=$((result + (loopcount * 2)))
done
echo "result is $result"
```

图 17. 例六代码与运行结果

```
> ./prog9_2.sh
result is 110
> ./prog9_3.sh
result is 110
```

图 18. 例五、例六运行结果

例七:

```
#!/bin/bash
select item in continue finish; do
    if [ "$item" = "finish" ]; then
        break
    fi
done

1) continue
2) finish
[#? 1
[#? 1
[#? 2
```

图 19. 例七代码与运行结果

问题十：用 vi 编写《实验指导》“第四部分 Shell 程序设计”中的例 8 及例 9 掌握条件语句的用法，函数的用法，观察运行结果。

答：这两个例子练习了 bash 中的 case 语法与函数的写法。例：

```
#!/bin/bash
case $1 in
01|1) echo "Month is January";;
02|2) echo "Month is February";;
03|3) echo "Month is March";;
04|4) echo "Month is April";;
05|5) echo "Month is May";;
06|6) echo "Month is June";;
07|7) echo "Month is July";;
08|8) echo "Month is August";;
09|9) echo "Month is September";;
10|10) echo "Month is October";;
11|11) echo "Month is November";;
12|12) echo "Month is December";;
*) echo "Invalid parameter";;
esac
```

```
> ./prog10_1.sh 1
Month is January
> ./prog10_1.sh 3
Month is March
```

图 20. 例八代码与运行结果

例 9：

```
#!/bin/bash
displaymonth ( ) { #定义函数
case $1 in
01|1) echo "Month is January";;
02|2) echo "Month is February";;
03|3) echo "Month is March";;
04|4) echo "Month is April";;
05|5) echo "Month is May";;
06|6) echo "Month is June";;
07|7) echo "Month is July";;
08|8) echo "Month is August";;
09|9) echo "Month is September";;
10|10) echo "Month is October";;
11|11) echo "Month is November";;
12|12) echo "Month is December";;
*) echo "Invalid parameter";;
esac
}
displaymonth 8 #调用函数
displaymonth 12
```

```
> ./prog10_2.sh
Month is August
Month is December
```

图 21. 例九代码与运行结果

问题十一：编程，在屏幕上显示用户主目录名（HOME）、命令搜索路径（PATH），并显示由位置参数指定的文件的类型和操作权限。

答：使用 echo 命令输出相应环境变量即可。

```
#!/bin/bash
echo "$HOME"
echo "$PATH"
ls -l "$1"
```

```
> ./prog11.sh prog3.sh
/root
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
-rwxrwxrwx 1 root root 95 Oct 27 02:27 prog3.sh
```

图 22. 问题十一代码与运行结果

思考题：到此为止你对 Shell 有所认识了吧？怎么样？自己再编两个程序：

- ① 做个批处理程序，体会一下批处理概念。
- ② 做个菜单，显示系统环境参数。将此程序设置为人人可用。

答：对于第 1 个问题，编写了一个计算两个日期之间的时间之差：

```
#!/bin/bash
date1="Jan 1, 2020"
date2="May 1, 2020"

time1=$(date -d "$date1" +%s)
time2=$(date -d "$date2" +%s)

diff=$(expr $time2 - $time1)
secondsinday=$(expr 24 \* 60 \* 60)
days=$(expr $diff / $secondsinday)

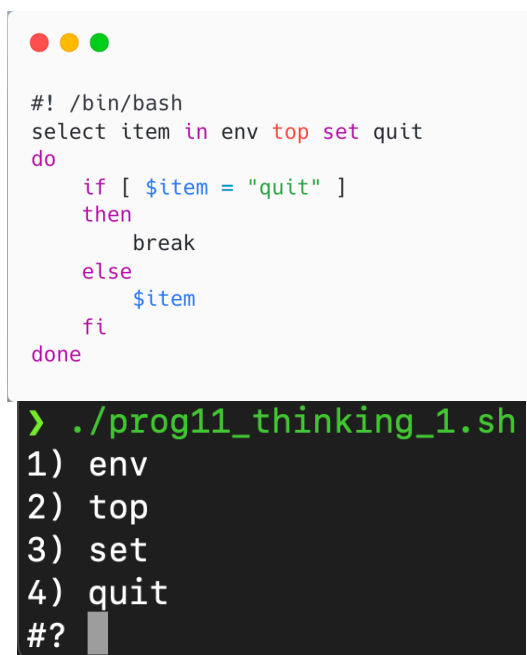
echo "The difference between $date2 and $date1 is $days
days"
```

```
> ./mydate.sh
The difference between May 1, 2022 and Jan 1, 2022 is 120 days
```

图 23. 思考题 1 代码与运行结果

观察运行结果，发现符合预期，这个脚本主要练习了 shell 中的数值计算方式。值得一提的是，bash 中默认是不支持浮点数运算的，在 bash 中使用浮点数运算可以使用“awk”脚本，另一种 shell 语言 zsh 是支持浮点数运算的。

第 2 个问题代码以及运行结果如下图所示，其中 `env top set` 分别为查看系统参数的命令，输入 `quit` 时退出。



```
#!/bin/bash
select item in env top set quit
do
    if [ $item = "quit" ]
    then
        break
    else
        $item
    fi
done

> ./prog11_thinking_1.sh
1) env
2) top
3) set
4) quit
#?
```

图 24. 思考题 2 代码与运行结果

## 4 讨论

### 1. Linux 的 Shell 有什么特点？

普通意义上的 shell 就是可以接受用户输入命令的程序。它之所以被称作 shell 是因为它隐藏了操作系统低层的细节。同样的 Linux 下的图形用户界面 GNOME 和 KDE，有时也被叫做“虚拟 shell”或“图形 shell”。Linux 操作系统下的 shell 既是用户交互的界面，也是控制系统的脚本语言。

Shell 在几乎所有操作系统上都很常见，因为它高效且易于更新，监控计算机系统并执行例行备份。使用 Shell 时，无需切换语法，因为 shell 的语法和命令与在命令行中输入的相同。此外，编写 shell 脚本既简单又快捷，它的启动速度快，而且易于调试。

Linux 中的 shell 有多种类型，例如 `zsh`、`bash`、`sh`、`dash` 等等。本次实验中，使用的 shell 语法是 `bash`。

### 2. 怎样进行 Shell 编程？如何运行？有什么条件？

怎样进行 Shell 编程这个问题太宽泛了。就我个人而言，学习的书籍是《Linux 命令行与 shell 脚本编程大全》这本书，这本书前几章介绍 Linux 相关环境，后几章介绍具体的脚本编写的语法等等，并且有配套练习的资源可供下载，学习价值很高。Shell 编程对于不同类型的 Shell 语法之间不能完全兼容，主流的是 `bash`，`zsh` 等等也有许多开发者在使用，Shell 编程用到时忘记了再查，反复练习效率会高。

关于 Shell 运行的几种方式，在回答问题三的扩展时已经有了具体的提及以及理由。总结而言为最建议的运行脚本方式是使用 `./` 相对路径引用，如此使用可以在不清楚这个脚本的前提下保证不因为 shell 选择而出错。其次在清楚此脚本是由何种脚本语言编写的前提下，使用类似 `bash prog1.sh` 与 `zsh prog.sh` 此类方式执行。

Shell 脚本运行的条件与逻辑与其他一般的编程语言类似，具体有下面几种方式：

1. 顺序执行：程序从上到下顺序执行
2. 选择执行：程序执行过程中，根据条件的不同，进行选择不同分支继续执行
3. 循环执行：程序执行过程中需要重复执行多次某段语句

3. vi 编辑程序有几种工作方式？查找有关的详细资料，熟练掌握屏幕编辑方式、转移命令方式以及末行命令的操作。学习搜索、替换字符、字和行，行的复制、移动，以及在 vi 中执行 Shell 命令的方式。

Vi 编辑程序有 3 种方式，对于 Vim 操作的练习我尝试过 Vimtutor。Vimtutor 是一个帮助练习 Vim 的 Linux 下的软件，具体界面如下：



```
=====
=  Welcome to the VIM Tutor - Version 1.7  =
=====

Vim is a very powerful editor that has many commands, too many to
explain in a tutor such as this. This tutor is designed to describe
enough of the commands that you will be able to easily use Vim as
an all-purpose editor.

The approximate time required to complete the tutor is 30 minutes,
depending upon how much time is spent with experimentation.

ATTENTION:
The commands in the lessons will modify the text. Make a copy of this
file to practice on (if you started "vimtutor" this is already a copy).

It is important to remember that this tutor is set up to teach by
use. That means that you need to execute the commands to learn them
properly. If you only read the text, you will forget the commands!

Now, make sure that your Caps-Lock key is NOT depressed and press
the j key enough times to move the cursor so that lesson 1.1
completely fills the screen.
~~~~~
Lesson 1.1: MOVING THE CURSOR

** To move the cursor, press the h,j,k,l keys as indicated. **

      ^
      k          Hint: The h key is at the left and moves left.
    < h      l >      The l key is at the right and moves right.
      j          The j key looks like a down arrow.
      v

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.

"/var/folders/q_/fmntmw611zs92mryr0g0y3ph000gn/T//tutorPkVzGw" 972 lines, 33584 bytes
```

图 25. Vimtutor 界面

关于搜索、替换字符、字和行，行的复制、移动，以及在 vi 中执行 Shell 命令的方式在 Vimtutor 中都有详细的操作练习。Vim 的强大之处不仅仅在于原生的完全使用键盘代替鼠标的操作，还有许多方便的插件，例如 Vim 代码补全，Vim 快捷键扩展等等。

#### 4. 编写一个具有以下功能的 Shell 程序。

- (1) 把当前目录下的文件目录信息输出到文件 filedir.txt 中；
- (2) 在当前目录下建立一个子目录，目录名为 testdir2 ；
- (3) 把当前目录下的所有扩展名为 c 的文件以原文件名复制到子目录 testdir2 中；
- (4) 把子目录中的所有文件的存取权限改为不可读。（提示：用 for 循环控制语句实现，循环的控制列表用 'ls' 产生。）
- (5) 在把子目录 testdir2 中所有文件的目录信息追加到文件 filedir.txt 中；
- (6) 把你的用户信息追加到文件 filedir.txt 中；
- (7) 分屏显示文件 filedir.txt

具体的代码与结果如下图所示：

```
#!/bin/bash
ls -l >filedir.txt
mkdir testdir2
tmp=$(find ./*.c)
cp "$tmp" testdir2
cd testdir2 || exit
for filename in *; do
    [[ -e "$filename" ]] ||
breakhmod 311 "$filename"
done
ls -l testdir2 >>filedir.txt
id >>filedir.txt
less fildir.txt
```

图 26. 讨论题 4 代码

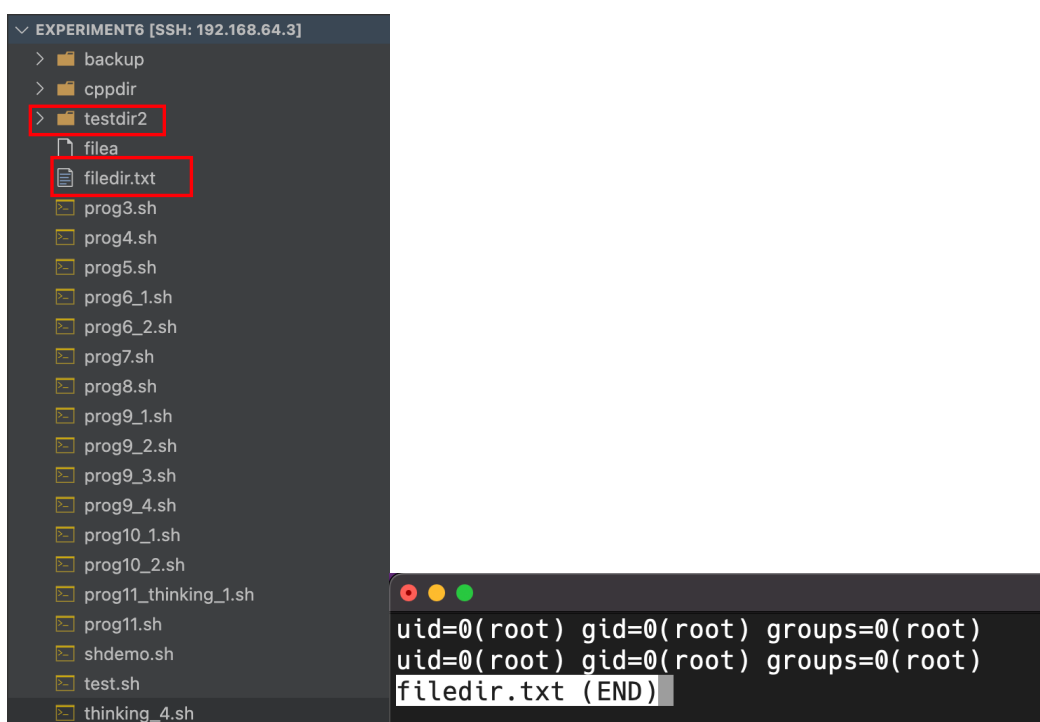


图 27. 讨论题 4 结果

可以看到由于 less 的分屏作用，最后脚本停留在了上图界面，并且 testdir2 目录也成功创建，filedir.txt 文件也存储在其中。

## 5 收获与体会

本次实验主要涉及了 Shell 编程，Shell 编程的基础在服务器编程方面比较重要，如果工作去运维岗位的话，使用 Shell 编写脚本一定是必会技能。对于 Vim，就算用其他的编辑器，至少要学会基本操作，毕竟从零安装 Linux 配置环境时没有那么多编辑器可以选择，还是需要用 Vim 更改，例如 sshd 文件中的登陆权限。