

# 《计算机操作系统》实验报告

## 实验题目：SHELL 编程

姓名：严昕宇      学号：20121802      实验日期：2022.11.05

### 实验环境：

实验设备：Lenovo Thinkbook16+ 2022

操作系统：Ubuntu 22.04.1 LTS 64 位

### 实验目的：

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

### 实验内容：

1. 学习使用 vi 编辑程序
2. 编写 Shell 程序
3. 将程序文件设置为可执行文件（用 chmod 命令）
4. 在命令行方式中运行 Shell 程序

### 操作过程 1：

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

### 结果 1：



```
yanxinyu@Thinkbook16-2022: ~  
  
VIM - Vi IMproved  
  
版本 8.2.4919  
维护人 Bram Moolenaar 等  
修改者 team+vim@tracker.debian.org  
Vim 是可自由分发的开放源代码软件  
  
帮助乌干达的可怜儿童！  
输入 :help iccf<Enter>      查看说明  
  
输入 :q<Enter>              退出  
输入 :help<Enter> 或 <F1>   查看在线帮助  
输入 :help version8<Enter> 查看版本信息  
  
0,0-1 全部
```

**思考：**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 的三种工作方式各用在何时

- ① 插入方式。进入插入方式，屏幕下方有一行 “-----Insert-----”字样表示。需要输入文本的内容时用到插入方式，可以用退格键来纠正错误。在插入方式中按一下 <ESC>，即退出插入方式，进入转义命令方式。
- ② 转义命令方式。刚进入 vi 或退出插入方式，即为转义命令方式。这时键入的任何字符转义为特殊功能，如：移动、删除、替换等。大多数转义命令由一个或两个字母组成，操作时没有提示符，而且输入命令不需要按<ENTER>。
- ③ 末行命令方式 在转义命令方式中，按冒号“:”进入末行命令方式。屏幕最末一行的行首显示冒号作为命令提示。命令行输入后按<ENTER>开始执行。此时用户可进行文件的全局操作，如：全局查找、替换、文件读、写等。

- 用什么命令进入插入方式

用户输入命令	功能描述
i text <ESC>	在光标前插入新文本，ENTER 可重起一行
I text <ESC>	在当前行起始处插入新文本
a text <ESC>	在光标后输入新的文本
A text <ESC>	在当前行末尾输入新的文本
o text <ESC>	在当前行下产生新的一行并输入文本
O text <ESC>	在当前行上产生新的一行并输入文本

- 怎样退出插入方式

在插入方式中按一下<ESC>，即退出插入方式，进入转义命令方式。

- 文件怎样存盘

存盘过程只需键入命令：**:w**，命令 **w** 将文件以当前名字存入磁盘，并覆盖了文件先前的副本。但 **w** 命令不影响缓冲区的内容。

如果要将文件以不同的名字保存，例如 **newfile**，需键入命令：**:w newfile**，如果不存在名为 **newfile** 的文件，vi 编辑器将存入文件并给出文件的大小。如果这个文件已经存在，那么 vi 会通知你，但并不刷新文件的内容。

如果想刷新已存在的文件的内容，则可键入命令：**:w !newfile** 同样，感叹号“!”放弃了标准的 vi 防止覆盖文件的保护功能，强制刷新。

## 操作过程 2：

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)

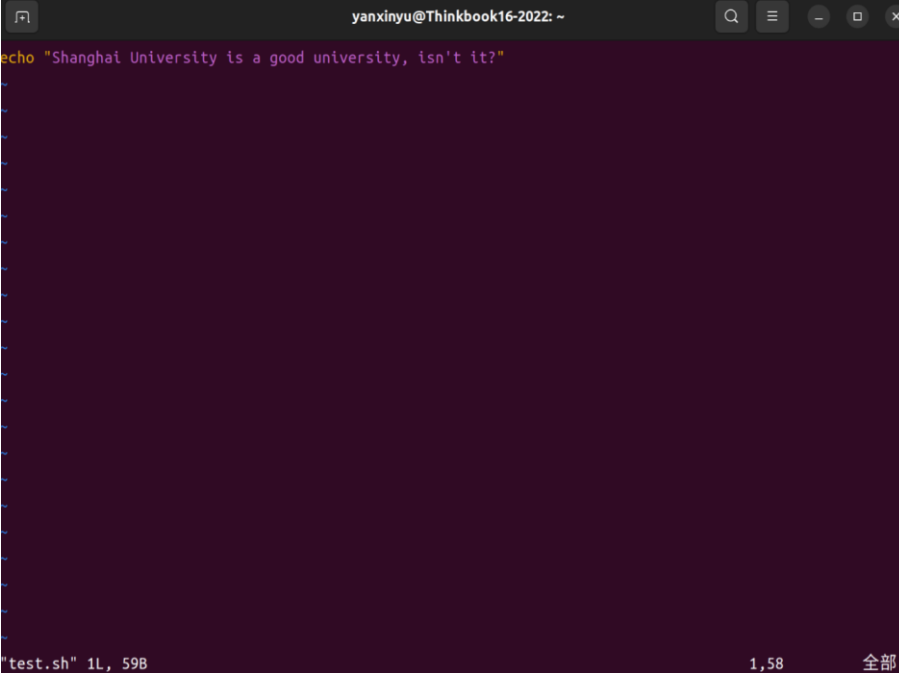
在提示符后执行 Shell 程序：

**\$ 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)

## 结果 2:

A terminal window titled 'yanxinyu@Thinkbook16-2022: ~' with search, menu, and window control icons in the title bar. The prompt is 'echo "Shanghai University is a good university, isn't it?"' and the output is 'Shanghai University is a good university, isn't it?'. The status bar at the bottom shows '"test.sh" 1L, 59B' on the left, '1,58' in the center, and '全部' on the right.

```
yanxinyu@Thinkbook16-2022: ~
echo "Shanghai University is a good university, isn't it?"
Shanghai University is a good university, isn't it?
"test.sh" 1L, 59B 1,58 全部
```

```
yanxinyu@Thinkbook16-2022:~$ vi test.sh
yanxinyu@Thinkbook16-2022:~$ chmod 755 test.sh
yanxinyu@Thinkbook16-2022:~$ ./test.sh
Shanghai University is a good university, isn't it?
yanxinyu@Thinkbook16-2022:~$
```

## 操作过程 3:

用 vi 编写《实验指导》“第四部分 Shell 程序设计”中的例 1(假设文件名为 prog1.h), 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.`

或`$sh prog1.`

屏幕显示:

Name not provided

在提示符后键入命令行:

`$/prog1.h Theodore`

#有一个参数

屏幕显示:

Your name is Theodore

#引用\$1 参数的效果

## 结果 3:

```
yanxinyu@Thinkbook16-2022: ~  
#!/bin/bash  
if [ $# == 0 ]  
then  
    echo "Name not provided"  
else  
    echo "Your name is " "$1"  
fi
```

"prog1.sh" 7L, 94B 7,2 全部

```
yanxinyu@Thinkbook16-2022:~$ vi prog1.sh  
yanxinyu@Thinkbook16-2022:~$ chmod 755 prog1.sh  
yanxinyu@Thinkbook16-2022:~$ ./prog1.sh  
Name not provided  
yanxinyu@Thinkbook16-2022:~$ ./prog1.sh YanXinyu  
Your name is  YanXinyu  
yanxinyu@Thinkbook16-2022:~$
```

### 单中括号 []:

① `bash` 的内部命令, `[]`和 `test` 是等同的。如果我们不用绝对路径指明, 通常我们用的都是 `bash` 自带的命令。`if/test` 结构中的左中括号是调用 `test` 的命令标识, 右中括号是关闭条件判断的。这个命令把它的参数作为比较表达式或者作为文件测试, 并且根据比较的结果来返回一个退出状态码。`if/test` 结构中并不是必须右中括号, 但新版 `bash` 中有此要求。

② `Test` 和 `[]`中可用的比较运算符只有 `=`和 `!=`, 两者都是用于字符串比较的, 不可用于整数比较, 整数比较只能使用 `-eq`, `-gt` 这种形式。无论是字符串比较还是整数比较都不支持大于号小于号。如果实在想用, 对于字符串比较可以使用转义形式, 如果比较 `"ab"`和 `"bc"`: `[ ab \< bc ]`, 结果为真, 也就是返回状态为 0。 `[]`中的逻辑与和逻辑或使用 `-a` 和 `-o` 表示。且 `[]`前后都有空格。

③ 字符范围。用作正则表达式的一部分, 描述一个匹配的字符范围。作为 `test` 用途的中括号内不能使用正则。

④ 在一个 `array` 结构的上下文中, 中括号用来引用数组中每个元素的编号。

### 操作过程 4:

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。

### 结果 4:

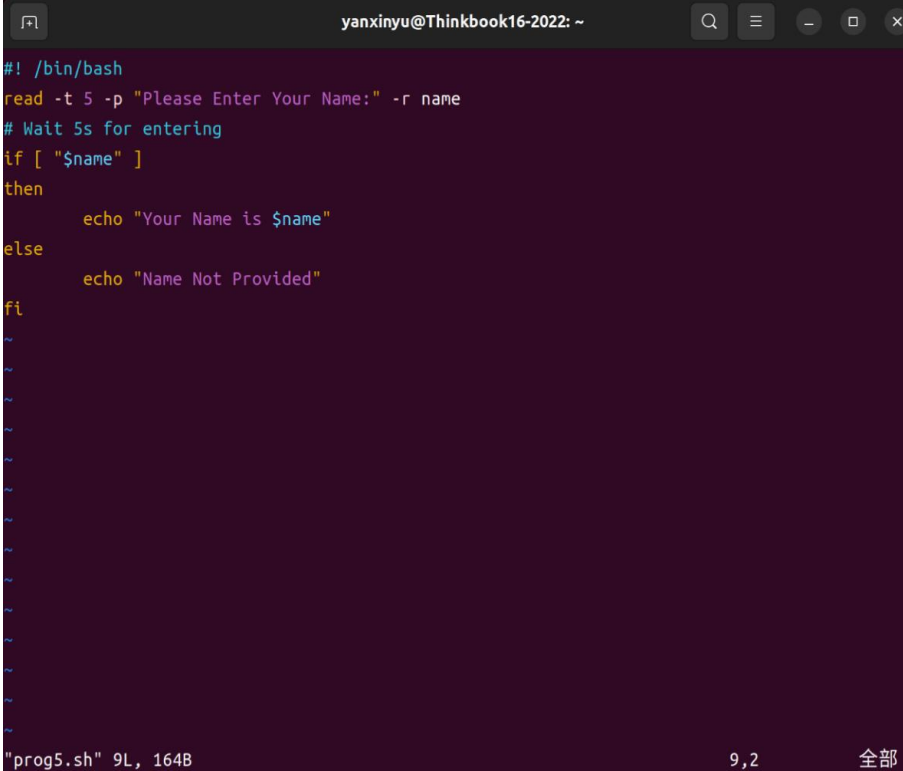
```
#!/bin/bash
echo "The Number of Parameter is $#"
```

```
yanxinyu@Thinkbook16-2022:~$ vi prog4.sh
yanxinyu@Thinkbook16-2022:~$ chmod 755 prog4.sh
yanxinyu@Thinkbook16-2022:~$ ./prog4.sh 1 2 3 4
The Number of Parameter is 4
The Program is ./prog4.sh
1
2
3
4
yanxinyu@Thinkbook16-2022:~$
```

## 操作过程 5:

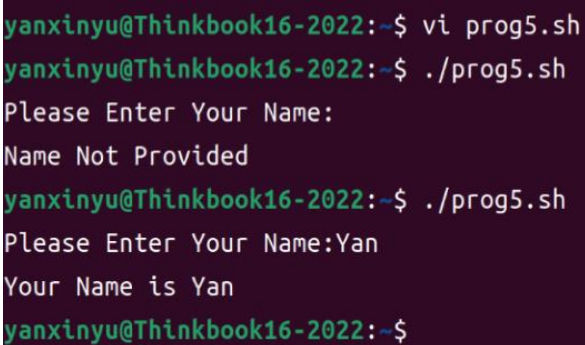
修改例 1 程序（即上面的 prog1.h），用 read 命令接受键盘输入。若没有输入显示第一种提示，否则第二种提示。

## 结果 5:



```
#!/bin/bash
read -t 5 -p "Please Enter Your Name:" -r name
# Wait 5s for entering
if [ "$name" ]
then
    echo "Your Name is $name"
else
    echo "Name Not Provided"
fi
```

"prog5.sh" 9L, 164B 9,2 全部



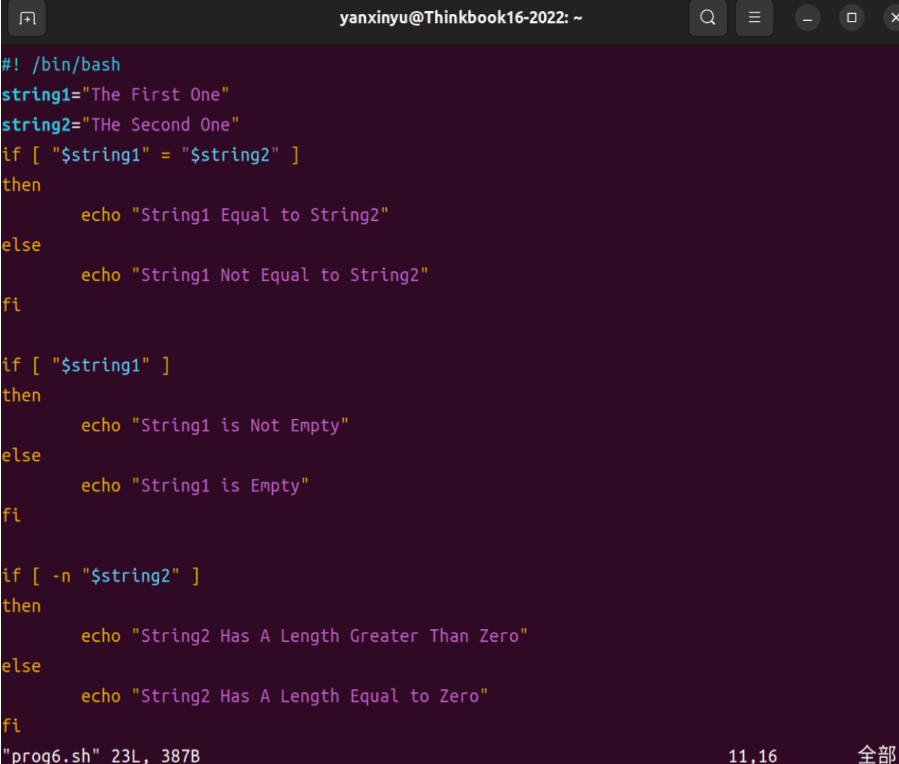
```
yanxinyu@Thinkbook16-2022:~$ vi prog5.sh
yanxinyu@Thinkbook16-2022:~$ ./prog5.sh
Please Enter Your Name:
Name Not Provided
yanxinyu@Thinkbook16-2022:~$ ./prog5.sh
Please Enter Your Name:Yan
Your Name is Yan
yanxinyu@Thinkbook16-2022:~$
```

## 操作过程 6:

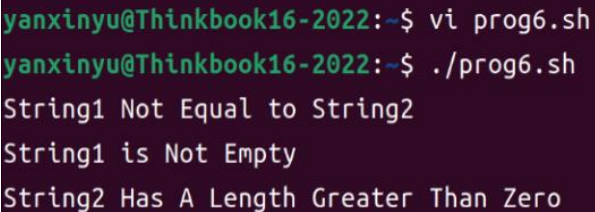
用 vi 编写《实验指导》“第四部分 Shell 程序设计”中的例 2、例 3，练习字符串比较运算符、数据比较运算符和文件运算符的用法，观察运行结果。

## 结果 6:

字符串比较运算符练习:



```
yanxinyu@Thinkbook16-2022: ~  
#!/bin/bash  
string1="The First One"  
string2="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 Greater Than Zero"  
else  
    echo "String2 Has A Length Equal to Zero"  
fi  
"prog6.sh" 23L, 387B 11,16 全部
```



```
yanxinyu@Thinkbook16-2022:~$ vi prog6.sh  
yanxinyu@Thinkbook16-2022:~$ ./prog6.sh  
String1 Not Equal to String2  
String1 is Not Empty  
String2 Has A Length Greater Than Zero
```

设当前目录下有一个文件 filea，控制权为(~ r-xr-xr-x 即 0751),有一个子目录 cppdir，控制权为(drwx rwx rwx)

程序 compare2.h 中有如下内容:

chmod 777 cppdir

chmod 751 filea

则结果如下:

```
yanxinyu@Thinkbook16-2022: ~  
#!/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 permission"  
else  
    echo "filea does not have read permission"  
fi  
if [ -w filea ]  
then  
    echo "filea has write permission"  
else  
    echo "filea does not have write permission"  
fi  
if [ -x cppdir ]  
then  
    echo "cppdir has execute permission"  
else  
    echo "cppdir does not have execute permission"  
fi  
~
```

1,8 全部

```
yanxinyu@Thinkbook16-2022:~$ vi prog6-1.sh  
yanxinyu@Thinkbook16-2022:~$ chmod 751 filea  
yanxinyu@Thinkbook16-2022:~$ ./prog6-1.sh  
cppdir is a directory  
filea is a regular file  
filea has read permission  
filea has write permission  
cppdir has execute permission  
yanxinyu@Thinkbook16-2022:~$ chmod 111 filea  
yanxinyu@Thinkbook16-2022:~$ ./prog6-1.sh  
cppdir is a directory  
filea is a regular file  
filea does not have read permission  
filea does not have write permission  
cppdir has execute permission  
yanxinyu@Thinkbook16-2022:~$
```



## 操作过程 7:

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。

## 结果 7:

```
yanxinyu@Thinkbook16-2022: ~  
#!/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" 21L, 422B 5,2 全部
```

```
yanxinyu@Thinkbook16-2022:~$ vi prog7.sh  
yanxinyu@Thinkbook16-2022:~$ ./prog7.sh  
Please Input String1:Test123  
Please Input String2:  
String1 Not Equal to String2  
String1 is Not Empty  
String2 Has A Length Equal to Zero  
yanxinyu@Thinkbook16-2022:~$ ./prog7.sh  
Please Input String1:123  
Please Input String2:123  
String1 Equal to String2  
String1 is Not Empty  
String2 Has A Length Greater Than Zero  
yanxinyu@Thinkbook16-2022:~$ ./prog7.sh  
Please Input String1:  
Please Input String2:  
String1 Equal to String2  
String1 is Empty  
String2 Has A Length Equal to Zero
```

## 操作过程 8:

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。

## 结果 8:

```
yanxinyu@Thinkbook16-2022: ~  
#!/bin/bash  
read -p "please enter 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 "$file" ]  
then  
    echo "file has read permission"  
else  
    echo "file does not have read permission"  
fi  
if [ -w "$file" ]  
then  
    echo "file has write permission"  
else  
    echo "file does not have write permission"  
fi  
if [ -x "$file" ]  
then  
    echo "file has execute permission"  
else  
    echo "file does not have execute permission"  
fi
```

32,2 全部

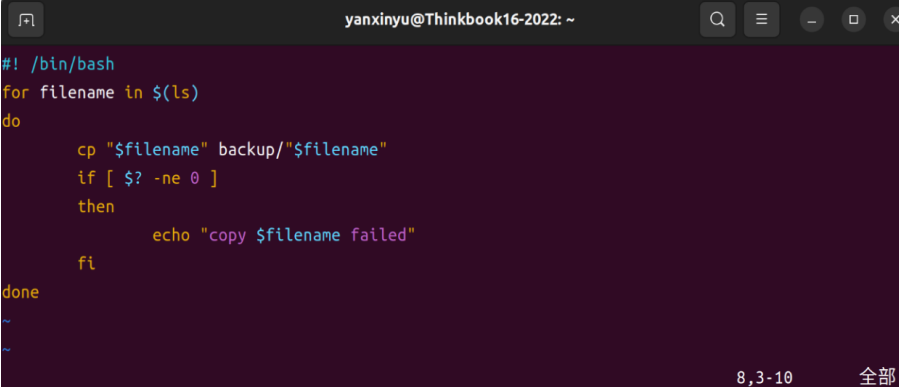
```
yanxinyu@Thinkbook16-2022:~$ vi prog8.sh  
yanxinyu@Thinkbook16-2022:~$ ./prog8.sh  
please enter file path:who.log  
file is not a directory  
file is a regular file  
file has read permission  
file has write permission  
file does not have execute permission  
yanxinyu@Thinkbook16-2022:~$
```

## 操作过程 9:

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。

## 结果 9:

### 例 4



```
yanxinyu@Thinkbook16-2022: ~  
#!/bin/bash  
for filename in $(ls)  
do  
    cp "$filename" backup/"$filename"  
    if [ $? -ne 0 ]  
    then  
        echo "copy $filename failed"  
    fi  
done
```

8,3-10 全部



```
yanxinyu@Thinkbook16-2022:~$ vi prog9-1.sh  
yanxinyu@Thinkbook16-2022:~$ ./prog9-1.sh  
cp: 未指定 -r; 略过目录'公共的'  
copy 公共的 failed  
cp: 未指定 -r; 略过目录'模板'  
copy 模板 failed  
cp: 未指定 -r; 略过目录'视频'  
copy 视频 failed  
cp: 未指定 -r; 略过目录'图片'  
copy 图片 failed  
cp: 未指定 -r; 略过目录'文档'  
copy 文档 failed  
cp: 未指定 -r; 略过目录'下载'  
copy 下载 failed  
cp: 未指定 -r; 略过目录'音乐'  
copy 音乐 failed  
cp: 未指定 -r; 略过目录'桌面'  
copy 桌面 failed  
cp: 未指定 -r; 略过目录'cppdir'  
copy cppdir failed  
cp: 无法创建普通文件'backup/f': 没有那个文件或目录  
copy f failed  
cp: 无法创建普通文件'backup/f5': 没有那个文件或目录  
copy f5 failed  
cp: 无法打开'filea' 读取数据: 权限不够  
copy filea failed
```

双小括号 (( )) :

① 整数扩展。这种扩展计算是整数型的计算，不支持浮点型。`((exp))`结构扩展并计算一个算术表达式的值，如果表达式的结果为 0，那么返回的退出状态码为 1，或者是"假"，而一个非零值的表达式所返回的退出状态码将为 0，或者是"true"。若是逻辑判断，表达式 `exp` 为真则为 1，假则为 0。

② 只要括号中的运算符、表达式符合 C 语言运算规则，都可用在 `$(exp)` 中，甚至是三目运算符。作不同进位(如二进制、八进制、十六进制)运算时，输出结果全都自动转化成了十进制。如：`echo $(16#5f)` 结果为 95(16 进位转十进制)

③ 单纯用 (( )) 也可重定义变量值, 如 `a=5; ((a++))` 可将 \$a 重定义为 6

④ 常用于算术运算比较，双括号中的变量可以不使用\$符号前缀。括号内支持多个表达式用逗号分开，只要括号中的表达式符合 C 语言运算规则。如可以直接使用 `for((i=0;i<5;i++))`，如果不使用双括号，则为 `for i in `seq 0 4``或者 `for i in {0..4}`。再如可以直接使用 `if (($i<5))`，如果不使用双括号，则为 `if [ $i -lt 5 ]`。

### 例 5

```
loopcount=0
result=0
while [ $loopcount -lt 10 ]
do
    loopcount=$((expr $loopcount + 1 ))
    result=$((result+($loopcount*2)))
done
echo "Result is $result"
```

```
yanxinyu@Thinkbook16-2022:~$ vi prog9-2.sh
yanxinyu@Thinkbook16-2022:~$ ./prog9-2.sh
Result is 110
```

exper :

① `expr` 命令可以实现数值运算、数值或字符串比较、字符串匹配、字符串提取、字符串长度计算等功能。它还具有几个特殊功能，判断变量或参数是否为整数、是否为空、是否为 0 等。

② 'expr'支持普通的算术操作，算术表达式优先级低于字符串表达式，高于逻辑关系表达式。'+-'加减运算。两端参数会转换为整数，如果转换失败则报错。'\* / %'，即乘，除，取模运算。两端参数会转换为整数，如果转换失败则报错。

### 例 6

```
yanxinyu@Thinkbook16-2022: ~  
#!/bin/bash  
loopcount=0  
result=0  
until [ $loopcount -ge 10 ]  
do  
    loopcount=$((expr $loopcount + 1))  
    result=$((($result + ($loopcount*2)))  
done  
echo "Result is "$result
```

"prog9-3.sh" 9L, 167B

```
yanxinyu@Thinkbook16-2022:~$ vi prog9-3.sh
yanxinyu@Thinkbook16-2022:~$ ./prog9-3.sh
Result is 110
yanxinyu@Thinkbook16-2022:~$
```

### 例 7

```
yanxinyu@Thinkbook16-2022: ~  
#!/bin/bash  
select item in continue finish  
do  
    if [ $item == "finish" ]  
    then  
        break  
    fi  
done
```

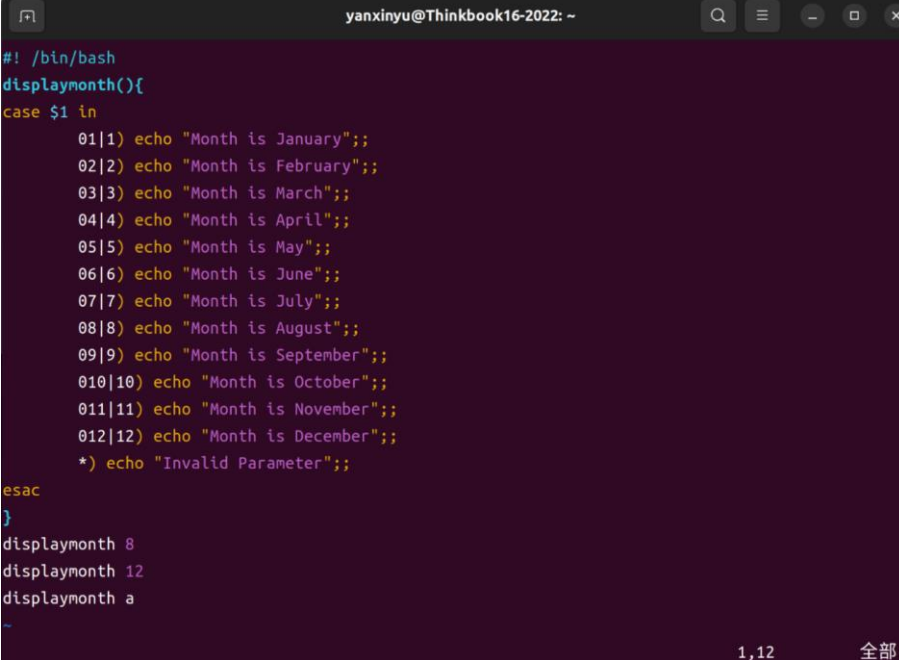
"prog9-4.sh" 8L, 96B 8,4 全部

```
yanxinyu@Thinkbook16-2022:~$ vi prog9-4.sh
yanxinyu@Thinkbook16-2022:~$ ./prog9-4.sh
1) continue
2) finish
#? 1
#? 2
yanxinyu@Thinkbook16-2022:~$
```

## 操作过程 10:

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。

## 结果 10:



```
yanxinyu@Thinkbook16-2022: ~  
#!/bin/bash  
displaymonth(){  
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  
displaymonth 12  
displaymonth a  
~  
1,12 全部
```

```
yanxinyu@Thinkbook16-2022:~$ vi prog10.sh  
yanxinyu@Thinkbook16-2022:~$ ./prog10.sh  
Month is August  
Month is December  
Invalid Parameter  
yanxinyu@Thinkbook16-2022:~$
```

### 操作过程 11:

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。

## 结果 11:

The screenshot shows a terminal window titled "yanxinyu@Thinkbook16-2022: ~". The terminal displays the following commands and output:

```

echo "Home is "$HOME
echo "Path is "$PATH
if [ $# -ne 0 ]
then
    ls -ld
fi

```

At the bottom of the terminal, it says "prog11.sh" 6L, 74B. On the right side, there are icons for search, menu, and window management.

---

A second terminal window snippet shows the execution of the script:

```

yanxinyu@Thinkbook16-2022:~$ vi prog11.sh
yanxinyu@Thinkbook16-2022:~$ ./prog11.sh
Home is /home/yanxinyu
Path is /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:/snap/bin
yanxinyu@Thinkbook16-2022:~$ ./prog11.sh Pulic
Home is /home/yanxinyu
Path is /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:/snap/bin
drwxr-x--- 17 yanxinyu yanxinyu 4096 11月 12 00:01 .
yanxinyu@Thinkbook16-2022:~$

```

**思考：**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：**① 做个批处理程序，体会一下批处理概念

参考例 9，编写函数，将其修改为读入用户输入的数字，随后输出对应的月份，或者提示错误信息。

```
yanxinyu@Thinkbook16-2022: ~  
#!/bin/bash  
read param  
displaymonth(){  
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 $param  
~  
~  
~  
"prog12.sh" 20L, 501B 1,1 全部
```

```
yanxinyu@Thinkbook16-2022:~$ vi prog12.sh  
yanxinyu@Thinkbook16-2022:~$ ./prog12.sh  
4  
Month is April  
yanxinyu@Thinkbook16-2022:~$ ./prog12.sh  
  
Invalid Parameter  
yanxinyu@Thinkbook16-2022:~$
```



② 做个菜单，显示系统环境参数。将此程序设置为人人可用。

```
select item in env finish
do
    if [ $item == "finish" ]
    then
        echo "finish"
        break
    else
        $item
    fi
done
```

```
yanxinyu@Thinkbook16-2022:~$ vi prog13.sh
yanxinyu@Thinkbook16-2022:~$ ./prog13.sh
1) env
2) finish
#? 1
SHELL=/bin/bash
SESSION_MANAGER=local/Thinkbook16-2022:@/tmp/.ICE-unix/1761,unix/Thinkbook16-2022:/tmp/.ICE-unix/1761
QT_ACCESSIBILITY=1
COLORTERM=truecolor
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/etc/xdg
SSH_AGENT_LAUNCHER=gnome-keyring
XDG_MENU_PREFIX=gnome-
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
LANGUAGE=zh_CN:en
LC_ADDRESS=zh_CN.UTF-8
GNOME_SHELL_SESSION_MODE=ubuntu
LC_NAME=zh_CN.UTF-8
SSH_AUTH_SOCK=/run/user/1000/keyring/ssh
XMODIFIERS=@im=ibus
DESKTOP_SESSION=ubuntu
LC_MONETARY=zh_CN.UTF-8
GTK_MODULES=gail:atk-bridge
PWD=/home/yanxinyu
XDG_SESSION_DESKTOP=ubuntu
LOGNAME=yanxinyu
XDG_SESSION_TYPE=wayland
SYSTEMD_EXEC_PID=1794
XAUTHORITY=/run/user/1000/.mutter-Xwaylandauth.BQCXU1
HOME=/home/yanxinyu
USERNAME=yanxinyu
IM_CONFIG_PHASE=1
LANG=zh_CN.UTF-8
LC_PAPER=zh_CN.UTF-8

XDG_DATA_DIRS=/usr/share/ubuntu:/usr/local/share:/usr/share:/var/lib/snapd/desktop
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:/nap/bin
GDMSESSION=ubuntu
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus
LC_NUMERIC=zh_CN.UTF-8
_=/usr/bin/env
#? 2
finish
yanxinyu@Thinkbook16-2022:~$
```

# 讨论

## 1. Linux 的 Shell 有什么特点？

**答：**Shell 的概念最初是在 Unix 操作系统中形成和得到广泛应用的。Unix 的 Shell 有很多种类，Linux 系统继承了 Unix 系统中 Shell 的全部功能，现在默认使用的是 bash。

Shell 具有如下突出特点：

- 把已有命令进行适当组合构成新的命令。
- 提供了文件名扩展字符（通配符，如\*、?、[ ]），使得用单一的字符串可以匹配多个文件名，省去键入一长串文件名的麻烦。
- 可以直接使用 Shell 的内置命令，而不需创建新的进程，如 Shell 中提供的 cd、echo、exit、pwd、kill 等命令。为防止因某些 Shell 不支持这类命令而出现麻烦，许多命令都提供了对应的二进制代码，从而也可以在新进程中运行。
- Shell 允许灵活地使用数据流，提供通配符、输入/输出重定向、管道线等机制，方便了模式匹配、I/O 处理和数据传输。
- 结构化的程序模块，提供了顺序流程控制、条件控制、循环控制等。
- Shell 提供了在后台执行命令的能力。
- Shell 提供了可配置的环境，允许创建和修改命令、命令提示符和其它的系统行为。
- Shell 提供了一个高级的命令语言，能够创建从简单到复杂的程序。这些 Shell 程序称为 Shell 脚本，利用 Shell 脚本，可把用户编写的可执行程序与 Unix 命令结合在一起，当作新的命令使用，从而便于用户开发新的命令。

## 2. 怎样进行 Shell 编程？如何运行？有什么条件？

**答：**Shell 程序就是一个由 Shell 和 Linux 命令组成的文件。可以使用 vim 或 gedit 编辑器编辑 Shell 程序的代码，然后用 chmod 将文件的权限设置为可执行模式。例如：

\$ 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)

## 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。

**答：**vi 编辑程序有三种工作方式，

### ① 命令行模式

在 shell 环境（提示符为\$）下输入启动 Vi 命令，进入编辑器时，就是处于该模式下。在该模式下，用户可以输入各种合法的 Vi 命令，用于管理自己的文档。此时从键盘上输入的任何字符都被当做编辑命令来解释，若输入的字符是合法的 vi 命令，则 vi 在接受用户命令之后完成相应的动作。

### ② 文本输入模式

在命令模式下输入插入命令 i、附加命令 a、打开命令 o、修改命令 c、取代命令 r 或替换命令 s 都可以进入文本输入模式。在该模式下，用户输入的任何字符都被 vi 当做文件内容保存起来，并将其显示在屏幕上。在文本输入过程中，若想回到命令模式下，按 Esc 键即可。

### ③ 末行模式

vi 有一个专门的“转义”命令，可访问很多面向行的 Ex 命令。在命令模式下，用户按“:”键即可进入末行模式下，此时 Vi 会在显示窗口的最后一行(通常也是屏幕的最后一行)显示一个“:”作为末行模式的提示符，等待用户输入命令。多数文件管理命令都是在此模式下执行的(如把编辑缓冲区的内容写到文件中等)。末行命令执行完后，vi 自动回到命令模式。

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`

答:

### Shell 程序:

```
#!/bin/bash
ls >> filedir.txt
mkdir testdir2
for filename in *.c
do
    cp "$filename" testdir2/"$filename"
    chmod 111 testdir2/"$filename"
done
ls testdir2 >> filedir.txt
who >> filedir.txt
more filedir.txt

"prog14.sh" 11L, 215B 1,1 全部
```

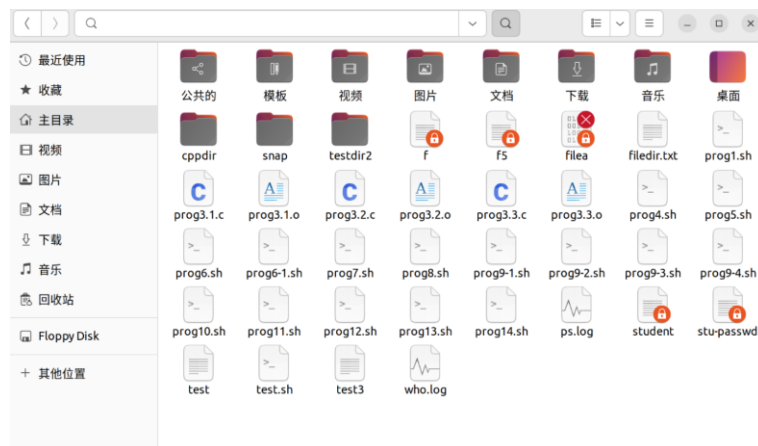
运行程序:

```
yanxinyu@Thinkbook16-2022:~$ vi prog14.sh
yanxinyu@Thinkbook16-2022:~$ ./prog14.sh
```

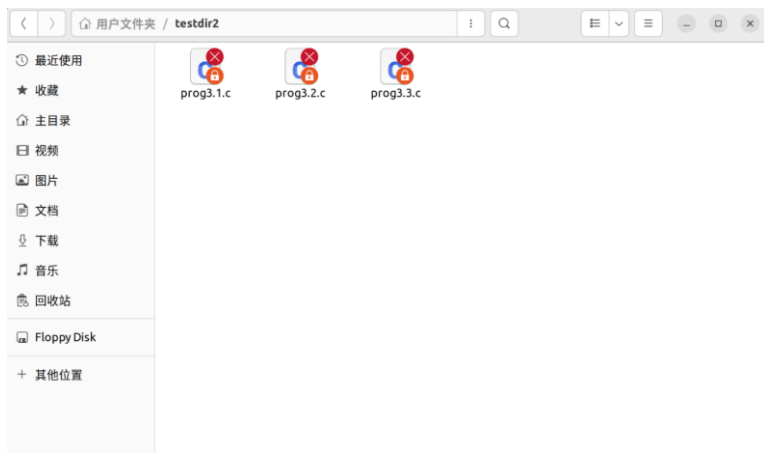
运行结果:

```
yanxinyu@Thinkbook16-2022: ~  
yanxinyu@Thinkbook16-2022: ~$ ./prog14.sh  
用户id=1000(yanxinyu) 组id=1000(yanxinyu) 组=1000(yanxinyu),1002(myusers)  
公共的  
模板  
视频  
图片  
文档  
下载  
音乐  
桌面  
cppdir  
f  
fs  
filea  
filedir.txt  
prog10.sh  
prog11.sh  
prog12.sh  
prog13.sh  
prog14.sh  
prog1.sh  
prog3.1.c  
prog3.1.o  
prog3.2.c  
prog3.2.o  
prog3.3.c  
prog3.3.o  
prog4.sh  
prog5.sh  
  
prog6-1.sh  
prog6.sh  
prog7.sh  
prog8.sh  
prog9-1.sh  
prog9-2.sh  
prog9-3.sh  
prog9-4.sh  
ps.log  
snap  
student  
stu-passwd  
test  
test3  
testdir2  
test.sh  
who.log  
prog3.1.c  
prog3.2.c  
prog3.3.c  
yanxinyu tty2 2022-11-11 17:03 (tty2)
```

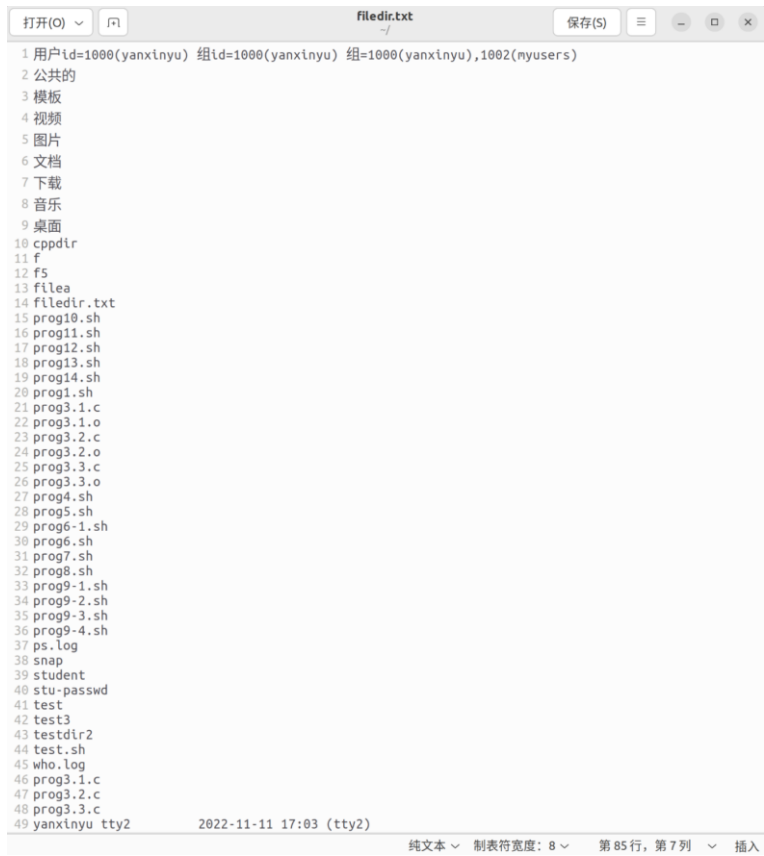
运行后目录:



testdir2 文件夹中的内容:



filedir.txt 中的内容:



## 实验体会

本次实验以 Shell 编程为主，通过实验，我学习了许多 Shell 编程的基础语法。在自己动手编写程序后，我感到 Shell 编程时对于格式要求的严格性极高，并不像我们平时使用 IDE 编写高级语言时，可以对一些不太重要的格式规范放宽要求，如等号前后的空格等，我正因这些问题，导致在尝试 Shell 编程遇到困难。

当然，通过本实验，我也体会到了 Shell 命令的强大之处。Shell 就是命令行工具的胶水，没有任何语言能像 Shell 一样方便地将一大堆命令行工具组合起来。原则上来说，Shell 做什么都可以，但显然它最适合的是自动化，因为只需要将原来手动敲的命令都复制到一个文件里面就行了。

Shell 跟标准的编程语言区别很大，它基本上是一个面向字符串的编程语言，组合用好 awk/sed/grep，偶尔配合 eval，有时候会发挥奇效，但也有可能原地爆炸(笑)。它可以跟 Python 之类的其他语言配合起来，比如某个复杂的功能使用一个 Python 脚本来实现，然后在 Shell 中调用这个脚本实现较复杂的功能；或者反过来，在 Python 脚本中调用外部的 Shell 脚本来提高自动化的效率，也是可以的。若是后续有条件能够进行更深一步的学习，可以编写脚本实现更强大的功能，帮助进行开发或是日常使用。