

# 浙江大学实验报告

课程名称： 课程综合实践II 实验类型： 验证型

实验项目名称： Linux shell基本命令

学生姓名： 胡若凡 专业： 计算机科学与技术 学号： 3200102312

电子邮件地址： [2811668688@qq.com](mailto:2811668688@qq.com) 手机（可选）： 13913421107

实验日期： 2022 年 7 月 31 日

## 一、实验环境

内存： 7.5GiB

处理器： 11th Gen Intel(R) Core(TM) i5-11320H @ 3.20GHz 3.19 GHz

图形： NVIDIA Corporation GP108M [GeForce MX250] / GeForce MX250/PCIe/SSE2

系统名称： Ubuntu 20.04.2 LTS

操作系统类型： 64位

## 二、实验内容和结果及分析

### 题目一：

1. 在操作系统课程实验中，要用make工具编译内核，要掌握make和makefile。makfile文件中的每一行是描述文件间依赖关系的make规则。本实验是关于makefile内容的，您不需要在计算机上进行编程运行，只要书面回答下面这些问题。

对于下面的makefile:

```
`CC = gcc`

`OPTIONS = -O3 -o`

`OBJECTS = main.o stack.o misc.o`

`SOURCES = main.c stack.c misc.c`

`HEADERS = main.h stack.h misc.h`

`polish: main.c $(OBJECTS)`

`$(CC) $(OPTIONS) power $(OBJECTS) -lm`

`main.o: main.c main.h misc.h`

`stack.o: stack.c stack.h misc.h`
```

```
`misc.o: misc.c misc.h`
```

回答下列问题

- 所有宏定义的名字
- 所有目标文件的名字
- 每个目标的依赖文件
- 生成每个目标文件所需执行的命令
- 画出makefile对应的依赖关系树。
- 生成main.o stack.o和misc.o时会执行哪些命令，为什么？\*\*

回答：

a. 宏的名字为：CC, OPTIONS, OBJECTS, SOURCES, HEADERS

b. 所有目标文件的名字为：polish, main.o, stack.o, misc.o

c. polish的依赖文件：main.c, main.o, stack.o, misc.o;

main.o的依赖文件：main.c, main.h, misc.h;

stack.o的依赖文件：stack.c, stack.h, misc.h;

misc.o的依赖文件：misc.c, misc.h

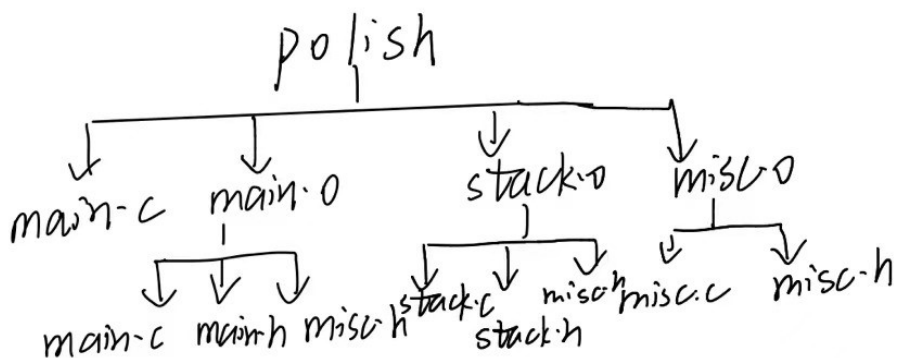
d. polish的命令：gcc -O3 -o power main.o stack.o misc.o -lm

main.o的命令：gcc -c main.c -o main.o

stack.o的命令：gcc -c stack.c -o stack.o

misc.o的命令：gcc -c misc.c -o misc.o

e. 如图所示



f. main.o的命令: gcc -c main.c -o main.o

stack.o的命令: gcc -c stack.c -o stack.o

misc.o的命令: gcc -c misc.c -o misc.o

因为这三个只是由单个的.c文件生成的.o文件。

## 题目二:

2.用编辑器创建main.c、compute.c、input.c、compute.h、input.h和main.h文件。下面是它们的内容。注意compute.h和input.h文件仅包含了compute和input函数的声明但没有定义。定义部分是在compute.c和input.c文件中。main.c包含的是两条显示给用户的提示信息。

```
$ cat compute.h
/* compute函数的声明原形 */
double compute(double, double);

$ cat input.h
/* input函数的声明原形 */
double input(char *);

$ cat main.h
/* 声明用户提示 */
#define PROMPT1 "请输入x的值: "
#define PROMPT2 "请输入y的值: "

$ cat compute.c
#include <math.h>
#include <stdio.h>
#include "compute.h"
double compute(double x, double y)
{
    return (pow ((double)x, (double)y));
}

$ cat input.c
#include <stdio.h>
#include "input.h"
double input(char *s)
{
    float x;
    printf("%s", s);
    scanf("%f", &x);
    return (x);
}

$ cat main.c
#include <stdio.h>
#include "main.h"
#include "compute.h"
#include "input.h"
main()
```

```

{
double x, y;
printf("本程序从标准输入获取x和y的值并显示x的y次方.\n");
x = input(PROMPT1);
y = input(PROMPT2);
printf("x的y次方是:%6.3f\n", compute(x,y));
}
$

```

为了得到可执行文件power，我们必须首先从三个源文件编译得到目标文件，并把它们连接在一起。下面的命令将完成这一任务。注意，在生成可执行代码时不要忘了连接上数学库。

```
$ gcc -c main.c input.c compute.c
```

```
$ gcc main.o input.o compute.o -o power -lm
```

(1) 创建上述三个源文件和相应头文件，用gcc编译器，生成power可执行文件，并运行power程序。给出完成上述工作的步骤和程序运行结果。

(2) 创建Makefile文件，使用make命令，生成power可执行文件，并运行power程序。给出完成上述工作的步骤和程序运行结果。

回答：

(1) 运行结果如下

```

huruofan@huruofan-virtual-machine:~$ touch computer.h
huruofan@huruofan-virtual-machine:~$ cat>>computer.h<<EOF
> /* compute函数的声明原形 */
> double compute(double, double);
> EOF
huruofan@huruofan-virtual-machine:~$ touch input.h
huruofan@huruofan-virtual-machine:~$ cat>>input.h<<EOF
> /* input函数的声明原形 */
> double input(char *);
> EOF
huruofan@huruofan-virtual-machine:~$ touch computer.c
huruofan@huruofan-virtual-machine:~$ cat>>computer.c<<EOF
> #include <math.h>
> #include <stdio.h>
> #include "compute.h"
> double compute(double x, double y)
> {
>     return (pow ((double)x, (double)y));
> }
> EOF
huruofan@huruofan-virtual-machine:~$ touch input.c
huruofan@huruofan-virtual-machine:~$ cat>>input.c<<EOF
> #include <stdio.h>
> #include "input.h"
> double input(char *s)
> {
>     float x;
>     printf("%s", s);
>     scanf("%f", &x);
>     return (x);
> }
> EOF
huruofan@huruofan-virtual-machine:~$ touch main.h
huruofan@huruofan-virtual-machine:~$ cat>>main.h<<EOF
> /* 声明用户提示 */
> #define PROMPT1 "请输入x的值: "
> #define PROMPT2 "请输入y的值: "
> EOF
huruofan@huruofan-virtual-machine:~$ touch main.c
huruofan@huruofan-virtual-machine:~$ cat>>main.c<<EOF

```

```

huruofan@huruofan-virtual-machine:~$ gcc -c main.c input.c compute.c
main.c:6:1: warning: return type defaults to 'int' [-Wimplicit-int]
main()
^~~~
huruofan@huruofan-virtual-machine:~$ gcc main.o input.o compute.o -o power -lm
huruofan@huruofan-virtual-machine:~$ ./power
本程序从标准输入获取x和y的值并显示x的y次方。
请输入x的值: 10
请输入y的值: 3
x的y次方是:1000.000

```

(2) 下载所需要的包，并且删除原来的执行文件，通过makefile 生成。

```
huruofan@huruofan-virtual-machine: ~
File Edit View Search Terminal Help
huruofan@huruofan-virtual-machine:~$ rm -f makefile
huruofan@huruofan-virtual-machine:~$ touch makefile
huruofan@huruofan-virtual-machine:~$ vim makefile
huruofan@huruofan-virtual-machine:~$ make
```

```
huruofan@huruofan-virtual-machine:~$ rm -f power
huruofan@huruofan-virtual-machine:~$ make
gcc compute.o input.o main.o -o power -lm
huruofan@huruofan-virtual-machine:~$ make
make: 'power' is up to date.
huruofan@huruofan-virtual-machine:~$ ./power
本程序从标准输入获取x和y的值并显示x的y次方。
请输入x的值：10
请输入y的值：3
x的y次方是：1000.000
huruofan@huruofan-virtual-machine:~$
```

### 题目三：

3. 编写shell 脚本，统计指定目录下的普通文件、子目录及可执行文件的数目，统计该目录下所有普通文件字节数总和，目录的路径名字由参数传入。（不能使用sed、awk等工具）

**思路：**首先是预判断参数，如果参数数目不对进行报错，其次就是对四种数目的文件进行统计，这个地方比较困难的地方是需要用ls 命令得到文件字节数进行累加，并且字节数存在于ls命令的第五个参数，所以取这条指令的第五个参数

**运行截图：**

```
huruofan@huruofan-virtual-machine:~$ ./Q3 ~/labs
bash: ./Q3: Permission denied
huruofan@huruofan-virtual-machine:~$ chmod Q3 777
chmod: invalid mode: 'Q3'
Try 'chmod --help' for more information.
huruofan@huruofan-virtual-machine:~$ chmod 777 Q3
huruofan@huruofan-virtual-machine:~$ ./Q3 ~/temp
"/home/huruofan/temp" has:
  7 ordinary file(s)
  6 directory(s)
  1 executable file(s)
 1964 byte(s) in ordinary files
huruofan@huruofan-virtual-machine:~$
```

**代码：**

```
#!/bin/bash

dd=0 ff=0 xx=0 bb=0//设置变量，初始化为0

if [ $# -gt 1 ] # 如果输入参数不为1个，则报错

then

    echo "Error"
```

```

exit 1

fi

if [ -d $1 ] #如果是目录文件

then

    PathName="$1" # 保存文件夹名字

    for file in $PathName #遍历

    do

        if [ -d "$file" ] # 如果是文件夹，文件夹计数+1

        then

            dd = dd + 1

            elif [ -x "$file" ] # 如果是可执行文件，可执行文件计数+1

            then

                xx = xx + 1

            elif [ -f "$file" ] # 如果是普通文件，普通文件计数+1并通过 ls 命令得到文件字节数，累加

            then

                ff = ff +1

                set -- `ls -l $file` #调出参数

                bb = bb + $5 #取出参数，作为字节数所需要的

            fi

        done

    elif

        echo "Error"

    exit 1

```

## 题目四：

4. 编写shel脚本，输入一个字符串，忽略（删除）非字母后，检测该字符串是否为回文(palindrome)。对于一个字符串，如果从前向后读和从后向前读都是同一个字符串，则称之为回文串。例如，单词“mom”，“dad”和“noon”都是回文串。（不能使用sed、awk、tr、rev等工具）

**思路：**回文判断，首先先对参数进行预判断，如果个数不为1的话报错；然后对于给定的字符串，去除所有不在a-z, A-Z之间的字符，然后开始做循环判断，判断前半部分的字母和对应的后半部分的字母是不是一样的，如果不是的话报错

**运行截图：**

```
huruofan@huruofan-virtual-machine:~$ ./Q4
./Q4: Input a string in one line:
ASADA1
./Q4: After deleting non-alpha characters: ASADA
./Q4: It is not palindrome.
huruofan@huruofan-virtual-machine:~$ ./Q4
./Q4: Input a string in one line:
abccba
./Q4: After deleting non-alpha characters: abccba
./Q4: It is palindrome.
```

**代码：**

```
#!/bin/bash

if [ $# -ne 1 ] # 如果输入参数不为1个，报错

then

    echo "Only one word please" # 提示正确用法

    exit 1

fi


str=${1//[!a-zA-Z]}      # 仅保留字母字符

Length=${#str}

for ((i=0; i<${Length/2}; i++))

do

    c1=${str:i:1}

    c2=${str:$(($Length-1-i)):1}

    if [ $c1 != $c2 ]

    then

        echo "$0: It is not palindrome."

        exit 0

    fi

done

echo "$0 : It is palindrome"
```

## 题目五：

5.编写一个实现文件备份和同步的shell脚本程序dirsync。程序的参数是两个需要备份同步的目录，如：

dirsync ~\dir1 ~\dir2 # ~\dir1为源目录，~\dir2为目标目录

dirsync程序实现两个目录内的所有文件和子目录（递归所有的子目录）内容保持一致。程序基本功能如下。

(1) 备份功能：目标目录将使用来自源目录的最新文件，新文件和新子目录进行升级，源目录将保持不变。dirsync程序能够实现增量备份。

(2) 同步功能：两个方向上的旧文件都将被最新文件替换，新文件都将被双向复制。源目录被删除的文件和子目录，目标目录也要对应删除。

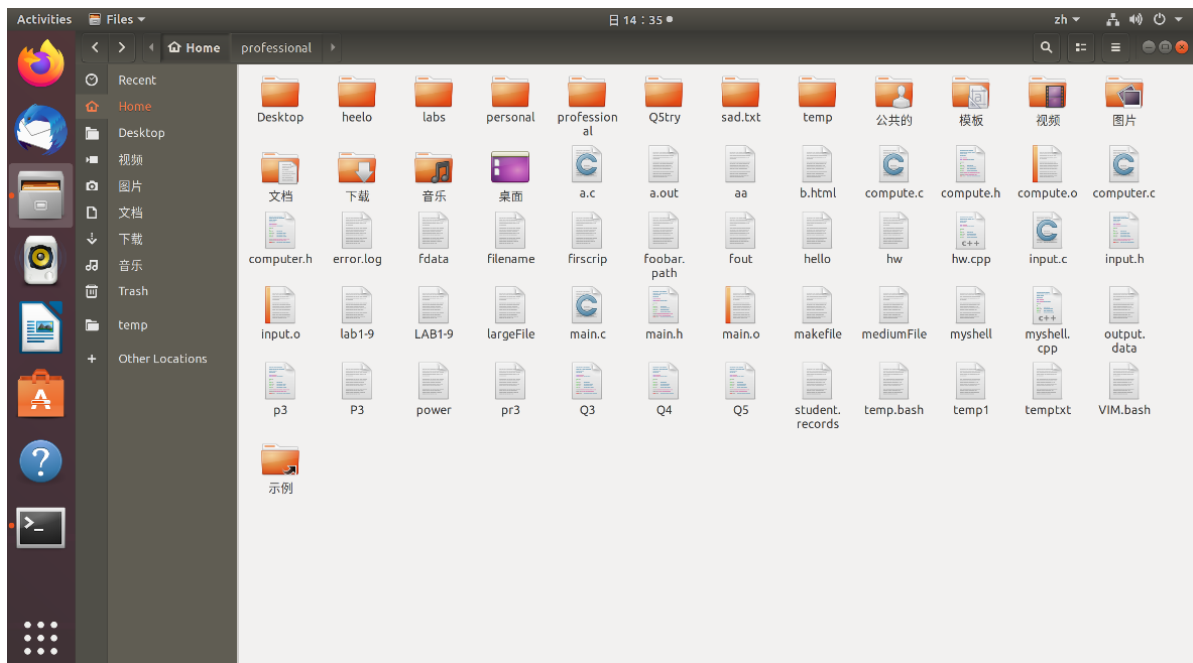
(3) 其它功能自行添加设计。

本题要求：不能使用现有的备份或同步程序，如：/usr/bin/rsync

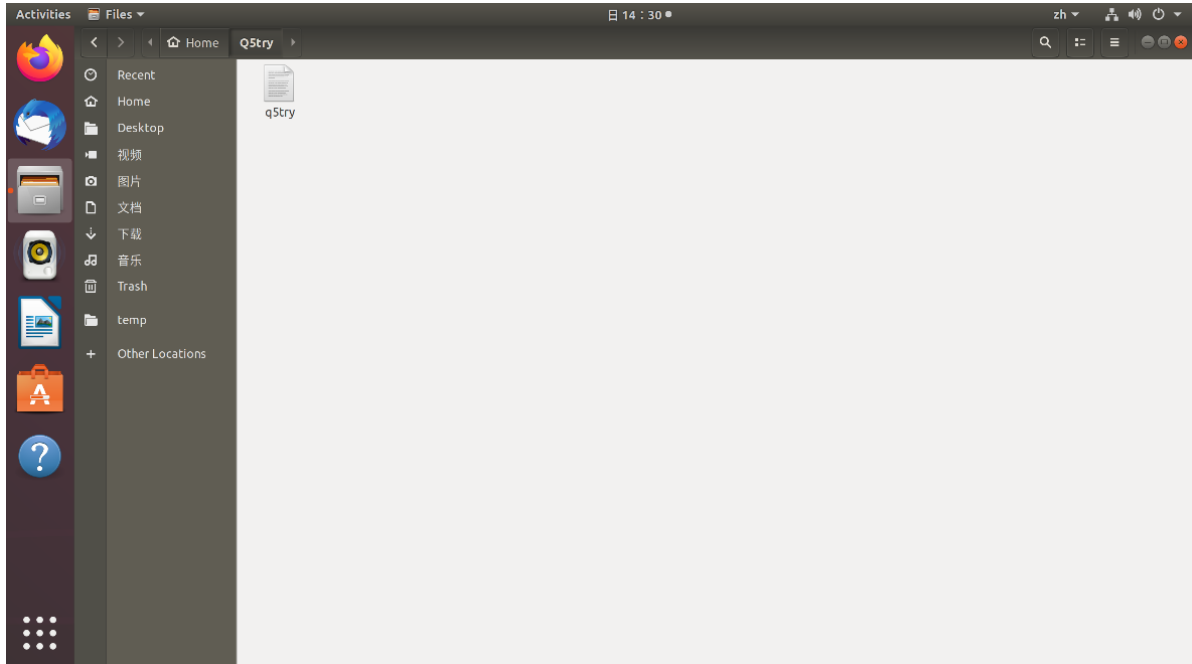
**思路：**首先依旧是对参数进行预判断，如果参数不对的话进行相应的报错；其次是备份，分成删除与复制两步，删除时把目标文件有而备份没的或类型不对的删除，再把目标目录有的给复制替换上去；最后是同步，备份后可以确保的是源目录中没有的文件在目标目录中一定也没有；目标目录中有的文件在源目录中一定有，于是只要把源目录有目标目录没有的给进行补全就可以。

### 运行截图：

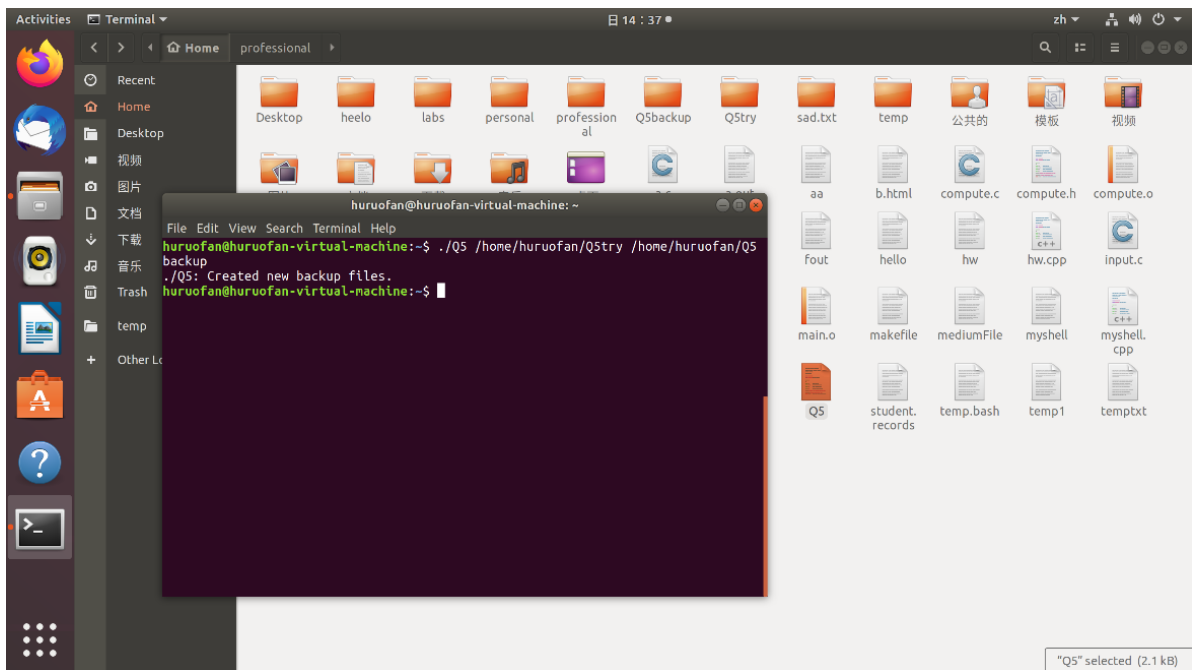
首先，创建Q5try文件夹，并且在其中创建一个q5try的文件

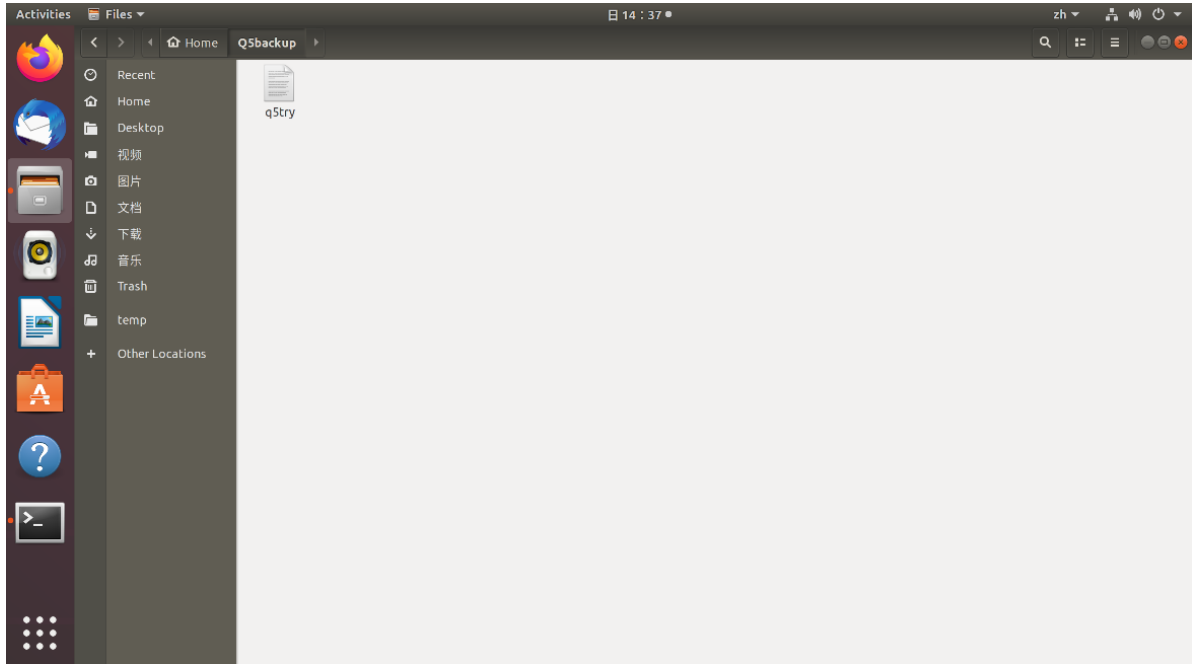




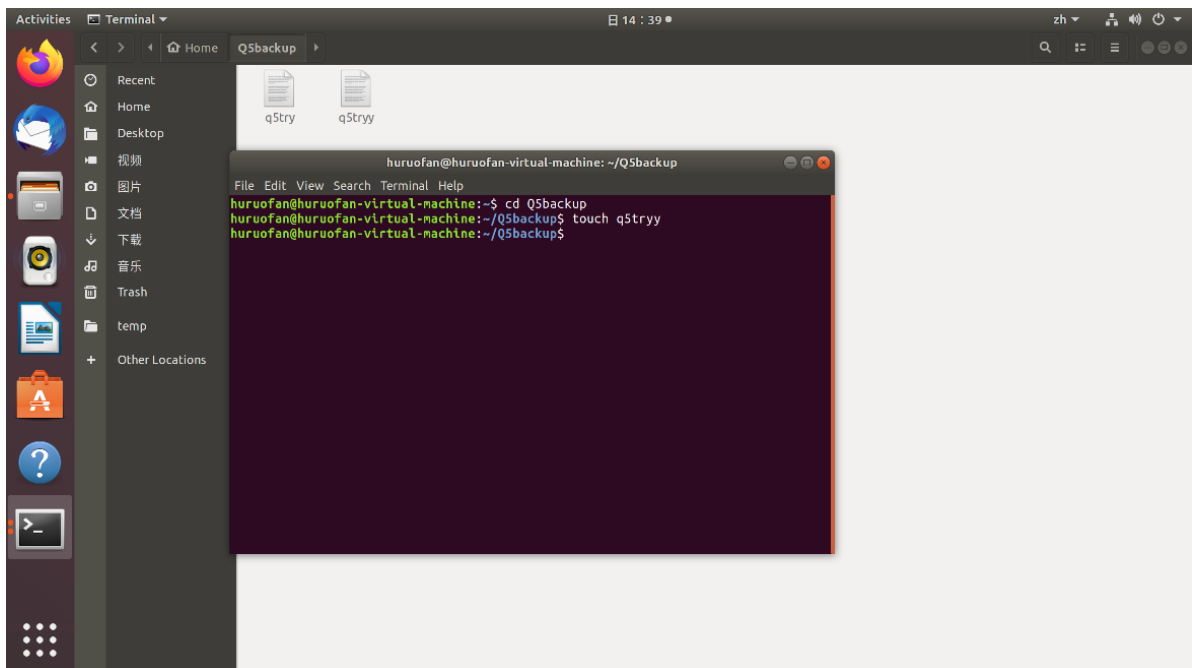


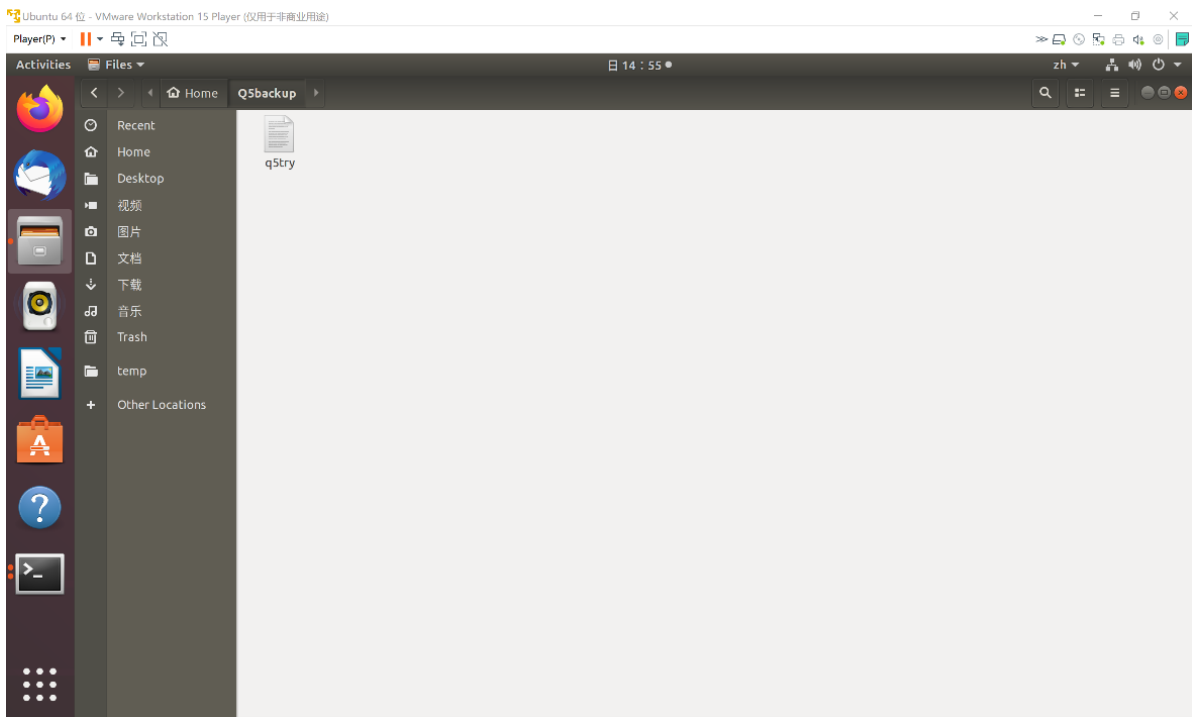
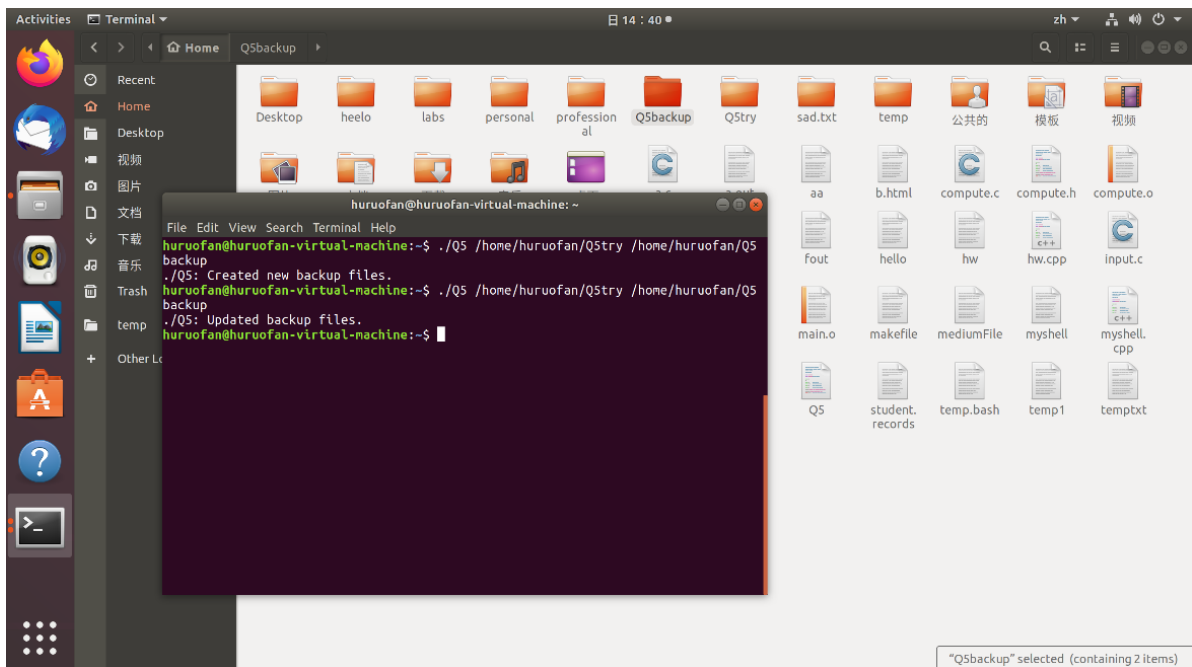
当程序实现备份功能后，出现Q5backup，并且Q5backup中出现相应的q5try文件





如果在Q5backup中创建一个额外的文件，在运行程序，那么再打开后，发现Q5backup中的这个额外的文件消失，实现了同步功能





代码:

```
#!/bin/bash
#备份功能实现
function Backward(){
    BackupPos=$2/"*"
    for file2 in $BackupPos
    do
        #根据目标目录中的file2，生成源目录的文件路径file1
        file1="${file2/#$2/$1}"
        #如果类型不对，删除
        if [[ -d "$file2" ]]; then
            if [[ !(-d "$file1") ]]; then
                rm -rf $file2
            else
            
```

```

        Backward "$file1" "$file2" #递归实现操作
    fi
else
    #如果类型不对或者不存在，删除
    if [[ !(-e "$file1") || (-d "$file1") ]]; then
        rm -rf $file2
    #不存在的要复制补全
    elif [[ $file1 -ot $file2 ]]; then
        cp -fp $file2 $file1
    fi
fi
done
}

function Forward(){
    SourcePos=$1/"*"
    for file1 in $SourcePos
    do
        #根据源目录中的file1，生成目标目录的文件路径file2
        file2="{file1/#$1/$2}"
        if [[ -d "$file1" ]]; then #类型正确
            if [[ !(-d "$file2") ]]; then
                cp -rfp $file1 $file2 #进行拷贝
            else
                Forward "$file1" "$file2"
            fi
        else
            #如果源目录有文件而目标目录没有，则备份一份到目标目录
            if [[ !(-e "$file2") || (-d "$file2") ]]; then
                cp -fp $file1 $file2
            elif [[ $file2 -ot $file1 ]]; then
                cp -fp $file1 $file2
            fi
        fi
    done
}

set `echo $1 $2`
if [ $# -ne 3 ] ; then # 如果输入参数不为3个，报错
    if [ $# -lt 3 ] ; then # 如果输入的参数小于3个的话，报错
        echo -n "Missing argument(s)! "
    elif [ $# -gt 3 ] ; then # 如果输入的参数大于3个的话，报错
        echo -n "Too many arguments! "
    fi
    usage
    exit 1
fi

if [[ $# -ne 2 || !(-d $1) ]]; then
    #参数个数必须为2，且第一个参数必须为目录文件
    echo "Usage: Input two parameters as two directories."
    exit 1
fi

```

```

if [[ !(-d $2) ]]; then
    #如果目标目录不存在，则直接整个拷贝
    if [[ -e $2 ]]; then
        rm -rf $2 #移除
    fi
    cp -rp $1 $2
    echo "$0: Created new backup files." #提示，创建了新的
    exit 0
fi

#先Backward再Forward
Backward $1 $2
Forward $1 $2
echo "$0: Updated backup files."
exit 0

```

## 题目六：

6.使用bash，编写一个命令行编辑器程序，编辑器的功能参考vim/vi，实现vim部分功能即可。不能直接调用现成的编辑器。

要求：不能使用开源代码，自己编写所有的代码。鼓励使用图形界面。

本实验题要求提供以下文档：

- 有功能描述文档。
- 设计文档，包括设计思想、功能模块、数据结构、算法等。
- 源程序。有详细的注释和良好编程风格。

### 6-1 功能描述

本程序支持功能如下：

- 可以切换两种模式，一种为控制模式，一种为编辑模式，可以自动创建文件。
- 控制模式下，通过输入的不同指令，可以对文档的整体内容进行一个整体的编辑，包括整体保存退出wq，整体首字母转换wa，整体小写往大写转换wb，整体大写往小写转换wc，整体删除特定带有特定内容的控制命令wo与wz。
- 编辑模式下，通过输入，可以保存每一行的内容，并且按原格式保存至文档之中。

### 6-2 设计思想

本程序设计思路如下：

1. 通过对输入的文件名，**if**判断需不需要新建一个文件。
2. 如果文件原存在，那么将所有文件的内容**while**逐行读入并输出。
3. 开始控制模式与编辑模式的转换，切换键为**esc**与**A/a**键，当按下**A**键时，会进入编辑模式，按下**esc**时，会进入控制模式
4. 编辑模式中，接收每一行的读入，逐行读入，并且保存至一个变量中，当编辑结束，将此变量内容加到原文件末尾
5. 控制模式中，接收读到的命令控制符，并且利用**sed**语句实现文档的修改，然后清屏将修改后的内容显示到屏幕上。

## 6-3 源代码

```
#!/bin/bash
#首先检查是否存在文件
#clear对整个终端命令进行翻页，更清楚
clear
#read -p可以显示vim命令
read -p "vim" file_name
#默认路径在桌面
FILE="/home/huruofan/"${file_name}
#如果存在文件，check设为1
if [ -e "$FILE" ]; then
    check=1
else
    #不存在的话创建普通文件
    > $FILE
    check=0
fi

#文件判断后，如正常的vim编辑器，再次翻页开始编辑
clear
Const=1
word=""
a=1
temp1=""
#如果原文件时存在的
if [[ $check==$Const ]]
then
    #逐行读入并且进行换行符的添加
    while read line
    do
        #将此行显示
        echo $line
        content+=$line
        content+="\n"
    done < $file_name
else
    content=""
fi

#如果原文件存在，最后增添一个换行符，以便后续读入
if [[ $content != $temp1 ]]
then
    content+="\n"
```

```

fi

word=""
Stop="stop"
temp="\n"
AAA="a"
FI="a"
SE=""

#wq为保存内容
WQ="wq"
#wo为整体删除带oo的
WO="wo"
#wz为整体删除带zz的
WZ="wz"
#w为替换所有单词的首字母为大写
WA="wa"
#wb为把所有的小写变成大写
WB="wb"
#wc为把所有的大写变成小写
WC="wc"


#这里一共分为控制模式和写入模式
#控制模式下我设置了六种模式，在上有关描述
#如果控制模式下不为保存一直继续
while [[ $tt != $WQ ]]
do
    #文档可能修改过，与当前目录内容不同，因此需要将content设为""
    content1=$temp1
    if [[ $FI != $SE ]]
    then
        FI=$SE
    else
        #清屏，把进行过控制操作的新文档内容给输出到终端上
        clear
        content=""
        #提示不是保存操作，将会继续进行两种模式的循环
        echo "not wq operation,repeat"
        while read line
        do
            #读文档和逐行输出
            echo $line
            content+=$line
            content+="\n"
        done < $file_name

    fi

    #模式转换，如果不是读入单个字符A无法继续
    #同vim，这里设置了静默读入和不显示在屏幕上，因此不会出现问题
    stty -echo
    read -s -n 1 CH
    while [[ $CH != $AAA ]]
    do
        read -s -n 1 CH
    done
    #恢复显示

```

```

stty echo
#设置结束符，为读到esc符号时
cESC=`echo -ne "\033"`
read word

#如果没有读到esc的话，一直做
while [[ ${word} != ${cESC} ]]
do
    content2 = content1
    #如果不是空，要加换行符
    if [[ $content1 != $temp1 ]]
    then content1+= $temp
    fi
    content1+= $word
    read word
done

#最后一次读入要解决换行问题
content1=$content2
content+= $content1
#把写入的内容同步更新到文档里
echo -e "$content" > $file_name、
#控制模式写入
read -p ":" -t 5 tt

#两个删除
if [ $tt == $WO ];
then
    nl $file_name | sed -i '/oo/p'
fi

if [ $tt == $WZ ];
then
    nl $file_name | sed -i '/zz/p'
fi

#转换首字母
if [ $tt == $WA ];
then
    sed -i 's/\b[a-z]/\u&/g' $file_name
fi

#整体小写大写转换
if [ $tt == $WB ];
then
    sed -i 's/[a-z]/\u&/g' $file_name
fi

#整体大写小写转换
if [ $tt == $WC ];
then
    sed -i 's/[A-Z]/\l&/g' $file_name
fi
done

```



####

## 6-4 运行截图

首先在终端内运行VIM.bash。

```
huruofan@huruofan-virtual-machine:~$ bash VIM.bash
```

其次，对一个全新的文件开始编辑。

```
huruofan@huruofan-virtual-machine: ~
File Edit View Search Terminal Help
vim try
```

直到静默按下A，不然无法进入编辑模式。由于A是不显示的，所以按键时不会显示在屏幕上。

```
huruofan@huruofan-virtual-machine: ~
File Edit View Search Terminal Help

```

进行随意的编辑，并在按下esc键后，进入控制模式，这里按下wq，进行保存。

```
huruofan@huruofan-virtual-machine: ~
File Edit View Search Terminal Help
a
b
c
^[
:wq
```

重新用cat try命令，打开try文件查看，发现已为编辑输入的内容。

```
huruofan@huruofan-virtual-machine: ~
File Edit View Search Terminal Help
huruofan@huruofan-virtual-machine:~$ cat try
a
b
c
```

再次进行编辑，发现从前的内容目前已存在。

```
huruofan@huruofan-virtual-machine: ~
File Edit View Search Terminal Help
a
b
c
```

再次编辑，并且保存后，发现换页并且显示出了从前的内容和现在的内容合并的文件内容。

```
huruofan@huruofan-virtual-machine: ~
File Edit View Search Terminal Help
a
b
c
d
e
f
^[
:wq
```

```
huruofan@huruofan-virtual-machine: ~
File Edit View Search Terminal Help
a
b
c
d
e
f
```

进行控制模式，这里显示小大写转换，当按下wb时，将会集体进行小写往大写的转换，当按下wc时，将会集体进行大写往小写的转换。此后按下A后将会继续进入编辑模式。

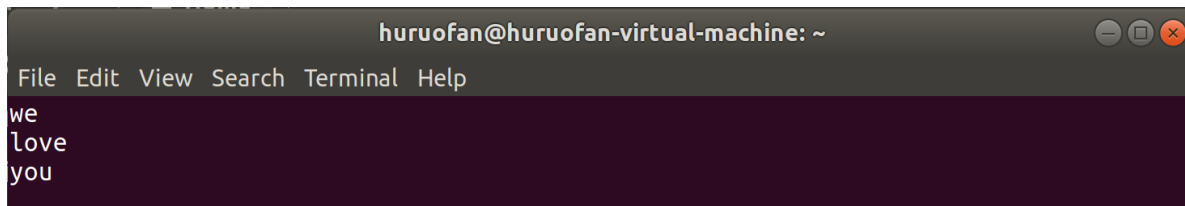
```
huruofan@huruofan-virtual-machine: ~
File Edit View Search Terminal Help
not wq operation,repeat
A
B
C
D
E
F
```

```
huruofan@huruofan-virtual-machine: ~
File Edit View Search Terminal Help
not wq operation,repeat
a
b
c
d
e
f
```

重新编辑文档，使得内容为

```
huruofan@huruofan-virtual-machine: ~
File Edit View Search Terminal Help
woo
we
love
you
```

控制模式下，按下wo，发现对应含特定字符oo的行已删除



```
huruofan@huruofan-virtual-machine: ~  
File Edit View Search Terminal Help  
we  
love  
you
```

## 题目七:

7.使用任何一种程序设计语言实现一个shell 程序的基本功能。

### 7-1 任务要求

(一) 这个shell 程序必须支持以下内部命令：bg、cd、clr、dir、echo、exec、exit、fg、help、jobs、pwd、set、test、time、umask。部分命令解释如下：

(1) cd ——把当前默认目录改变为。如果没有参数，则显示当前目录。如该目录不存在，会出现合适的错误信息。这个命令也可以改变PWD 环境变量。

(2) pwd ——显示当前目录。

(3) time ——显示当前时间

(4) clr ——清屏。

(5) dir ——列出目录的内容。

(6) set ——列出所有的环境变量。

(7) echo ——在屏幕上显示并换行（多个空格和制表符可能被缩减为一个空格）。

(8) help ——显示用户手册，并且使用more 命令过滤。

(9) exit ——退出shell。

(10) shell 的环境变量应该包含shell=/myshell，其中/myshell 是可执行程序shell 的完整路径（不是你的目录下的路径，而是它执行程序的路径）。

(二) 其他的命令行输入被解释为程序调用，shell 创建并执行这个程序，并作为自己的子进程。程序的执行的环境变量包含一下条目：

parent=/myshell。

(三) shell 必须能够从文件中提取命令行输入，例如shell 使用以下命令行被调用：

myshell batchfile

这个批处理文件应该包含一组命令集，当到达文件结尾时shell 退出。很明显，如果shell 被调用时没有使用参数，它会在屏幕上显示提示符请求用户输入。

(四) shell 必须支持I/O 重定向，stdin 和stdout，或者其中之一，例如命令行为：

programname arg1 arg2 < inputfile > outputfile

使用arg1 和arg2 执行程序programname，输入文件流被替换为inputfile，输出文件流被替换为outputfile。

stdout 重定向应该支持以下内部命令：dir、environ、echo、help。

使用输出重定向时，如果重定向字符是>，则创建输出文件，如果存在则覆盖之；如果重定向字符为>>，也会创建输出文件，如果存在则添加到文件尾。

(五) shell 必须支持后台程序执行。如果在命令行后添加&字符，在加载完程序后需要立刻返回命令行提示符。

(六) 必须支持管道("|") 操作。

(七) 命令行提示符必须包含当前路径。

#### **提示：**

你可以假定所有命令行参数（包括重定向字符<、>、>>和后台执行字符&）和其他命令行参数用空白空间分开，空白空间可以为一个或多个空格或制表符。

#### **项目要求：**

(1) 设计一个简单的全新命令行shell，至少满足上面的要求并且在指定的Linux 平台上执行。拒绝使用已有的shell程序的任何环境及功能。严禁使用开源代码。

(2) 写一个关于如何使用shell 的简单的用户手册，用户手册应该包含足够的细节以方便Linux初学者使用。例如：你应该解释I/O 重定向、管道、程序环境和后台程序执行。

(3) 源代码必须有很详细的注释，并且有很好的组织结构以方便别人阅读和维护；否则会扣除客观的分。结构和注释好的程序更加易于理解，并且可以保证批改你作业的人不用很费劲地去读你的代码。

#### **实验报告内容包括：**

源代码文件、必要设计文档和用户手册，所有提交的文件以文本形式复制到报告内；还有myshell上测试各种命令的运行结果截图。

## **7-2 设计思路**

### **单条命令：**

#### **cd**

- (1) 如果参数数量大于1个，报错.
- (2) 特殊情况：cd ~ 或者无参，回到原先记录好的主目录
- (3) 一般情况：cd+地址，先用chdir判断是否存在，不存在此地址报错，否则转移地址并修改PWD值为当前更新的地址

#### **clr**

- (1) 如果有参数，报错
- (2) 如果无参数，实现清屏功能，调用系统自带的函数system("clear")

#### **dir**

- (1) 如果参数大于等于2，报错
- (2) 确定目录，如果无参数，目录为当前地址的目录，否则为第一个参数指定的目录
- (3) 记录当前目录下的文件名到数组，并进行排序，整理答案到待输出的参数中

## **echo**

- (1) 对有双引号的，脱双引号
- (2) 对于带\$的，去找参数
- (3) 对于不特殊的，正常输出

## **exec**

- (1) 参数信息太少，直接报错
- (2) 参数信息足够，调用execvp

## **set**

- (1) 判断参数，数量不对时报错
- (2) 遍历environ，对环境变量的值进行记录并准备输出

## **exit**

- (1) 如果有参数，报错
- (2) 调用系统自带的\_exit(0)函数，终止进程并进行输出

## **pwd**

- (1) 如果有参数，报错
- (2) 将记录好的PWD输出

## **help**

- (1) 如果有参数，报错
- (2) 如果无参数，打开已经编辑好的help文档，逐行把文件内容记录到待输出的变量之中

## **time**

- (1) 如果有参数，报错
- (2) 声明time\_t timer，并且调用localtime，得到当前的时间

## **unset**

- (1) 判断参数个数是否为1，报错
- (2) 通过unsetenv实现对给定参数的调用

## umask

- (1) 参数大于1, 报错
- (2) umask无参时, 表示输出当前的umask值。
- (3) 当数值不足3位时, 默认向右对齐(高位补0), 设置新的umask值。

## shift

- (1) shift如果大于等于2, 报错
- (2) 无参时, 左移1位, 否则移动相应的位数, 如果位数过多, 置1

## test

- (1) 判断参数个数是否为2或3, 否则报错
- (2) 如果参数个数为2时, 支持对文件类型的判断
- (3) 如果参数个数为3时, 支持文件类型比较, 支持数字大小的比较

## 重定向:

(1) 重定向任务中, 需要实现把原本从标准输入(stdin)和标准输出(stdout)实现I/O重定向。这里具体实现的任务是输入重定向<, 输出重定向>>(追加)和>(覆盖)的重定向

(2) 重定向实现中, 具体要借助的是dup2函数。操作中, 如果命令中检查到'<', '>'或者'>>'三者其中之一时, 将重定向的目标文件名保存到inputfile或outputfile中, 用open()函数用相应的方式打开文件(只读、覆盖写、追加写)。之后, 只需要用dup2()函数用打开的文件流替换相应的标准输出或输入流便实现了重定向操作。完成了重定向操作后, 需要注意将标准输入输出流复原。

## 进程控制:

(1) 进程控制任务中, 主要需要实现的为以下几个分任务:

在指令后面加上&符号, 指令就在后台运行。MyShell无需等待这条指令执行完成, 可以立刻输出终端提示符并接收用户的下一条指令。

fg指令, 将被挂起或后台运行的作业转移至前台。

bg指令, 将被挂起的作业转移至后台运行。

jobs指令, 查看当前所有被挂起或后台运行的作业。

(2) 进程任务实现中, 具体实现思路如下:

对于需要后台运行的命令: 如果命令以'&'结尾, 需要把工作交给子进程。也就是说, 对于需要后台执行的命令, 在主进程先用fork创建子进程, 但是主进程中使用waitpid()函数, 但使用其中的非阻塞选项WNOHANG。这样就可以不阻塞主进程的进程, 实现命令的后台执行。

对于命令fg: 让MyShell向指定的进程发送SIGCONT信号即可, 然后MyShell一直等待直至子进程结束。

对于命令bg：让Myshell向指定进程发送SIGCONT信号，但是Myshell不需要等待子进程。

对于命令jobs：循环遍历，判断进程是否在运行，然后打印进程。

### 管道符：

(1) 管道符操作任务中，主要解决的前一个指令的输出是下一个指令的输入；第一条指令的输入是标准输入；最后一条指令的输出是标准输出的问题。为了简化任务，这里我假设管道符号最多只有一个。

(2) 管道操作的实现中，在命令读取的过程中检查到'|'时，进入管道符运算函数。首先调用pipe()函数创建无名管道，管道两端的文件描述符分别保存在pipeFd[0]和pipeFd[1]中。（《UNIX环境高级编程》：管道的作用可以类比为**一个共享文件，一个进程将信息写到管道内，另一个进程再从管道内读取信息，就完成了两个进程之间的通信**）对于管道，利用fork()函数先创建一个子进程pid1，在pid1中，将标准输出重定向到管道的读入端，并执行命令，命令的输出将输入管道文件。在主进程中，首先用waitpid()函数阻塞主进程，等待子进程pid1返回，待pid1返回后，再次利用fork()函数创建一个子进程pid2，在pid2中，将标准输入重定向到管道的输出端，并执行命令，命令的输入将从管道文件中读取。在主进程中，用waitpid()函数阻塞主进程，等待子进程pid2返回，待pid2也返回后，利用close()函数关闭管道两端即可。

### 外部命令：

(1) 外键命令任务中，除了内建命令之外，myshell需要能够自动查找并执行外部命令。

(2) 外键命令实现中，如果命令行输入被解释为程序调用，通过fork()创建子进程，然后在子进程中调用execvp()函数来查找并执行这个程序，如果没有找到则会报错。

### 脚本文件：

(1) 脚本文件任务中，myshell能够从脚本文件中提取命令行输入，在调用myshell时，如果不加参数则进入命令行输入模式，如果加上一个脚本文件的参数，则会从参数代表的文件中提取命令并执行。

(2) 脚本任务实现中，在检查到命令行参数时，myshell将打开参数代表的文件，之后读取命令时，会先从文件流中读取内容到buf，而不是从标准输入流中读取。同样的，如果打开文件失败则会进行报错。

### 程序环境：

(1) 程序环境任务中，myshell能显示路径。

(2) 程序环境实现中，记录下用户名，主机名以及当前路径PWD，按格式输出，即可完成此任务。

## 7-3 用户手册

### cd

参数个数：无参数或一个参数

作用：无参数回到主目录，否则根据参数跳转目录

使用示例：cd ~

## **clr**

参数个数：无参数

作用：清屏

使用示例：clr

## **dir**

参数个数：无参数或一个参数

作用：显示目录下的内容，无参数为当前目录，一个参数为参数指定的目录

使用示例：dir

## **echo**

参数个数：无限制

作用：显示指定的内容，无参数不显示，有参数显示参数指定的内容语句

使用示例：echo "we are famliy"

## **exec**

参数个数：一个参数

作用：使用参数代表的命令替换当前进程

示例：exec ls

## **set**

参数个数：无参数

作用：无参数时，显示所有环境变量

示例：set

## **exit**

参数个数：无参数

作用：退出程序

示例：exit

## **pwd**

参数个数：无参数



作用：显示当前的路径

示例：pwd

## **help**

参数个数：无参数

作用：显示帮助文档

示例：help

## **time**

参数个数：无参数

作用：显示当前时间

示例：time

## **unset**

参数个数：一个参数

作用：将参数所指的环境变量的值取消

示例：unset USER

## **umask**

参数个数：无参数或一个参数

作用：无参数时，显示当前掩码；有1个参数时，将当前掩码修改为参数的值

示例：umask 0777

## **shift**

参数个数：无参数或两个参数

作用：从标准输入读入参数，左移后输出，左移的位数由shift命令后跟的参数决定，无参数则默认左移一位，有1个参数则左移参数代表的位数

示例：shift 1

## **test**

参数个数：两个参数或三个参数

作用：进行文件类型的确定，整型的比较

示例：test 2 -ne 2

test -w File

## **jobs**

参数个数：无参数

作用：显示当前存在的进程

示例：jobs

## **fg**

参数个数：无参数

作用：将最近的一个后台任务转到前台执行

示例：fg

## **bg**

参数个数：无参数

作用：将最近一个挂起的进程转为后台执行

示例：bg

## **重定向**

作用：输入输出的重定向,<为输入重定向,>为覆盖的输出重定向,>>为追加的输出重定向

示例：cat < test1.txt >> test2.txt

## **管道符**

作用：在符号'|'左边命令的输出将成为右边命令的输入

示例：cat hello | sort

## **进程控制**

作用：在输入命令后空格并输入字符'&'，即可使得该条命令在后台执行而不阻塞主进程

示例：echo &

## **脚本文件**

作用：myshell能够从脚本文件中提取命令行输入

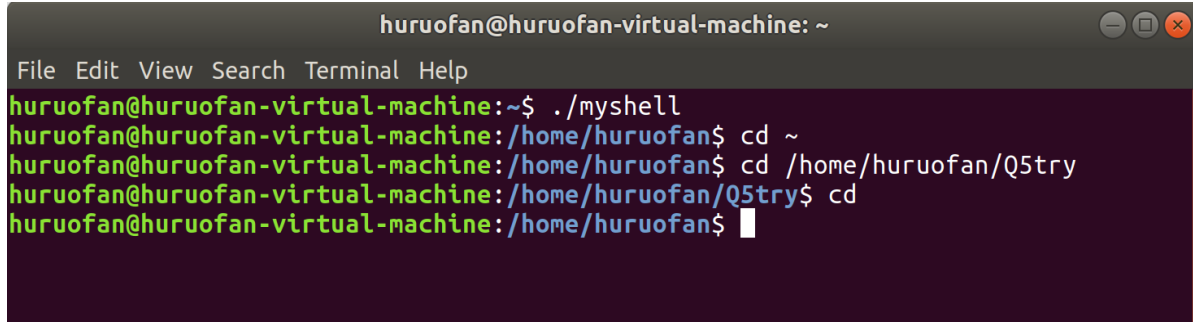
示例：myshell batchfile

## 7-4 运行截图

1. cd命令：转换当前的目录位置

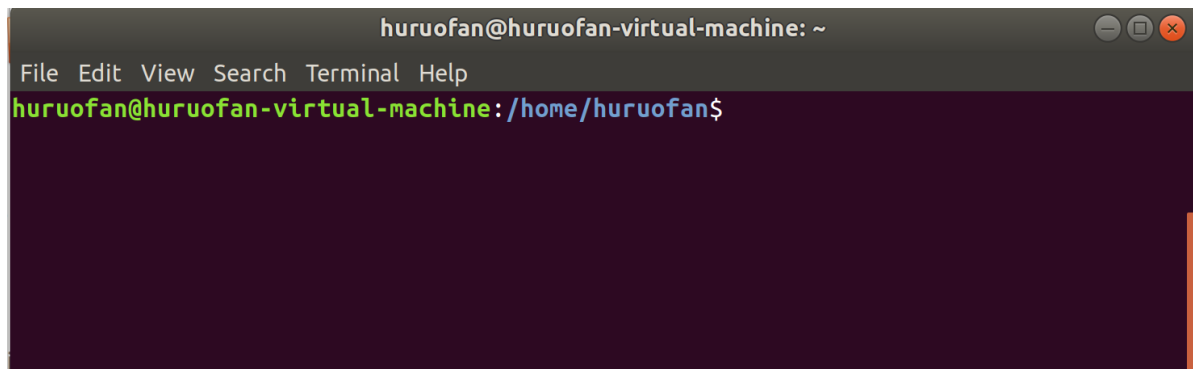
```
huruofan@huruofan-virtual-machine:/home/huruofan$ cd ~  
huruofan@huruofan-virtual-machine:/home/huruofan$ cd /home/huruofan/Q5try  
huruofan@huruofan-virtual-machine:/home/huruofan/Q5try$ cd
```

2. clr命令：清屏，在实行clr后，由上图的命令转到下图的清屏



A terminal window titled 'huruofan@huruofan-virtual-machine: ~' with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows the following commands and output:

```
huruofan@huruofan-virtual-machine:~$ ./myshell  
huruofan@huruofan-virtual-machine:/home/huruofan$ cd ~  
huruofan@huruofan-virtual-machine:/home/huruofan$ cd /home/huruofan/Q5try  
huruofan@huruofan-virtual-machine:/home/huruofan/Q5try$ cd  
huruofan@huruofan-virtual-machine:/home/huruofan$
```



The same terminal window as above, but after the 'clr' command has been executed. The screen is now clear, showing only the prompt 'huruofan@huruofan-virtual-machine:/home/huruofan\$'.

3. dir命令：显示目录下的内容，展示无参与有参的情况

```
huruofan@huruofan-virtual-machine: ~
File Edit View Search Terminal Help
huruofan@huruofan-virtual-machine:~/home/huruofan$ dir
.ICEauthority
.LAB1-9.swo
.LAB1-9.swp
.Q5.swo
.Q5.swp
.VIM.bash.swn
.VIM.bash.swo
.VIM.bash.swp
.bash_history
.bash_logout
.bashrc
.cache
.config
.firscrip.swo
.firscrip.swp
.gnupg
.lab1-9.swp
.local
.mozilla
.pam_environment
.pr3.swp
.profile
.ssh

huruofan@huruofan-virtual-machine:~/home/huruofan$ dir /home/huruofan/Q5try
q5try
```

4. 展现echo命令，如果不带参数将会输出空白，带参数输出参数的内容

```
huruofan@huruofan-virtual-machine:~/home/huruofan$ echo

huruofan@huruofan-virtual-machine:~/home/huruofan$ echo we love you
we love you
```

5.展现exec命令，执行外部命令ls

```
huruofan@huruofan-virtual-machine:~/home/huruofan$ exec ls
aa          examples.desktop  input.h          myshell          Q5          公共的
a.c          fdata             input.o          myshell.cpp       Q5backup    模板
a.out        filename          lab1-9           output.data       Q5try       视频
b.html       firscrip          LAB1-9           p3                sad.txt     图片
compute.c    foobar.path       labs             P3                student.records 文档
compute.h    fout             largeFile        personal          temp        下载
compute.o    heelo            main.c           power             temp1       音乐
computer.c   hello            main.h           pr3               temp.bash   桌面
computer.h   hw               main.o           professional      temptxt
Desktop      hw.cpp           makefile         Q3                try
error.log    input.c          mediumFile       Q4                VIM.bash
```

6.展现help命令：显示帮助手册（这里为了测试，help我只是简要定义）

```
huruofan@huruofan-virtual-machine:~/home/huruofan$ help
This is the help file
```

7.展示test命令：能够测试文档和整数，对未知命令能实行报错

```

huruofan@huruofan-virtual-machine:/home/huruofan$ test 2 ge 3
test: Unknown command 2.
huruofan@huruofan-virtual-machine:/home/huruofan$ test 2 -ge 3
false
huruofan@huruofan-virtual-machine:/home/huruofan$ test -r hello
false
huruofan@huruofan-virtual-machine:/home/huruofan$ test 4 -gt 2
true

```

8. 展示pwd命令：能够显示当前路径

```

huruofan@huruofan-virtual-machine:/home/huruofan$ pwd
/home/huruofan
huruofan@huruofan-virtual-machine:/home/huruofan$ cd Q5try
huruofan@huruofan-virtual-machine:/home/huruofan/Q5try$ pwd
/home/huruofan/Q5try

```

9. set命令，输出环境变量

```

huruofan@huruofan-virtual-machine:/home/huruofan/Q5try$ set
CLUTTER_IM_MODULE=xim
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd
=40;33;01:or=40;31;01:mi=00:su=37;41:sg=30;43:ca=30;41:tw=30;42:ow=34;42:st=37;4
4:ex=01;32:*.tar=01;31:*.tgz=01;31:*.arc=01;31:*.arj=01;31:*.taz=01;31:*.lha=01;
31:*.lz4=01;31:*.lzh=01;31:*.lzma=01;31:*.tlz=01;31:*.txz=01;31:*.tzo=01;31:*.t7
z=01;31:*.zip=01;31:*.z=01;31:*.Z=01;31:*.dz=01;31:*.gz=01;31:*.lrz=01;31:*.lz=0
1;31:*.lzo=01;31:*.xz=01;31:*.zst=01;31:*.tzst=01;31:*.bz2=01;31:*.bz=01;31:*.tb
z=01;31:*.tbz2=01;31:*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01;31:
*.ear=01;31:*.sar=01;31:*.rar=01;31:*.alz=01;31:*.ace=01;31:*.zoo=01;31:*.cpio=0
1;31:*.7z=01;31:*.rz=01;31:*.cab=01;31:*.wim=01;31:*.swm=01;31:*.dwm=01;31:*.esd
=01;31:*.jpg=01;35:*.jpeg=01;35:*.mjpg=01;35:*.mjpeg=01;35:*.gif=01;35:*.bmp=01;
35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;35:*.tga=01;35:*.xbm=01;35:*.xpm=01;35:*.tif
=01;35:*.tiff=01;35:*.png=01;35:*.svg=01;35:*.svgz=01;35:*.mng=01;35:*.pcx=01;35
:*.mov=01;35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01;35:*.mkv=01;35:*.webm=01;35:*.ogm
=01;35:*.mp4=01;35:*.m4v=01;35:*.mp4v=01;35:*.vob=01;35:*.qt=01;35:*.nuv=01;35:*.
wmv=01;35:*.asf=01;35:*.rm=01;35:*.rmvb=01;35:*.flc=01;35:*.avi=01;35:*.fli=01;
35:*.flv=01;35:*.gl=01;35:*.dl=01;35:*.xcf=01;35:*.xwd=01;35:*.yuv=01;35:*.cgm=0
1;35:*.emf=01;35:*.ogv=01;35:*.ogx=01;35:*.aac=00;36:*.au=00;36:*.flac=00;36:*.m
4a=00;36:*.mid=00;36:*.midi=00;36:*.mka=00;36:*.mp3=00;36:*.mpc=00;36:*.ogg=00;3
6:*.ra=00;36:*.wav=00;36:*.oga=00;36:*.opus=00;36:*.spx=00;36:*.xspf=00;36:
LC_MEASUREMENT=zh_CN.UTF-8
LESSCLOSE=/usr/bin/lesspipe %s %s
LC_PAPER=zh_CN.UTF-8

```

10. shift命令，移动，当只输入shift不加参数，移动1次。

```

aaa bbb ccc
aaa bbb

```

11. time命令，输出时间

```

huruofan@huruofan-virtual-machine:/home/huruofan$ time
2022.8.8. Monday 20:13:6

```

12. umask命令，实现命令的补全和修改

```

huruofan@huruofan-virtual-machine:/home/huruofan$ umask
0002
huruofan@huruofan-virtual-machine:/home/huruofan$ umask 77
huruofan@huruofan-virtual-machine:/home/huruofan$ umask
0077

```

13.unset命令，将参数所指的环境变量的值取消，这里取消HOME

```

huruofan@huruofan-virtual-machine:/home/huruofan$ echo $HOME
/home/huruofan
huruofan@huruofan-virtual-machine:/home/huruofan$ unset HOME
huruofan@huruofan-virtual-machine:/home/huruofan$ echo $HOME
huruofan@huruofan-virtual-machine:/home/huruofan$

```

14.exit命令，退出程序

```

huruofan@huruofan-virtual-machine:/home/huruofan$ exit
huruofan@huruofan-virtual-machine:~$

```

15.调用外部程序，这里编了一个打印hello, word的程序，可以看到内外输出一致

```

huruofan@huruofan-virtual-machine: ~
File Edit View Search Terminal Help
huruofan@huruofan-virtual-machine:~$ g++ outfile.cpp -o outfile
huruofan@huruofan-virtual-machine:~$ ./outfile
hello,world
huruofan@huruofan-virtual-machine:~$ ./myshell
huruofan@huruofan-virtual-machine:/home/huruofan$ ./outfile
hello,world

```

16.进程控制，这里用echo来展示，发现确实在后台运行

```

huruofan@huruofan-virtual-machine:/home/huruofan$ echo love &
[1] 3537 Running echo love
huruofan@huruofan-virtual-machine:/home/huruofan$ love

```

17.jobs，显示进程

```

huruofan@huruofan-virtual-machine:/home/huruofan$ sleep 10 &
[1] 3676 Running sleep 10
huruofan@huruofan-virtual-machine:/home/huruofan$ jobs
[1] 3676 Running sleep 10

```

18.fg，将最近的一个后台任务转到前台执行

```

huruofan@huruofan-virtual-machine: ~
File Edit View Search Terminal Help
huruofan@huruofan-virtual-machine:~$ sleep 20 &
[1] 3631
huruofan@huruofan-virtual-machine:~$ jobs
[1]+  Running sleep 20 &
huruofan@huruofan-virtual-machine:~$ fg
sleep 20
huruofan@huruofan-virtual-machine:~$ jobs

```

## 19.管道

```
huruofan@huruofan-virtual-machine:~/myshell$ cat outfile.cpp | sort
{
}
    cout<<"hello,world"<<endl;
#include<iostream>
int main()
    return 0;
using namespace std;
```

## 20.重定向

```
huruofan@huruofan-virtual-machine:/home/huruofan$ cat < outfile.cpp > help
huruofan@huruofan-virtual-machine:/home/huruofan$ cat help
#include<iostream>

using namespace std;
int main()
{
    cout<<"hello,world"<<endl;
    return 0;
}
```

## 21.执行命令

```
huruofan@huruofan-virtual-machine:~$ ./myshell Q7.sh
2022.8.8. Monday 20:42:42
/home/huruofan
```

## 7-5 源代码

```
#include <iostream>
#include<stdlib.h>
#include<algorithm>
#include <string>
#include <sstream>
#include <ctime>
#include <stddef.h>
#include <time.h>
#include <unistd.h>
#include <dirent.h>
#include <pwd.h>
#include <sys/wait.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <fcntl.h>
#include <fstream>
using namespace std;

string UserName,HomeDir,PWD;//用户名,主目录地址,当前地址
string Message;//用来报错等
string Answer;//用来输出所需的信息
```

```
string argv[1024]; //用来作为全局参数供调用
string CMDInfo[1024]; //子进程执行的命令
string HostName; //主机名
string ShellPath; //用来记录主路径
string CurrentCMD; //用来记录当前命令
bool InIsTerminal; //输入来自终端。只有为真时，才会输出子进程表和提示字符串
bool OutIsTerminal; //输出送至终端。
int TerminalIn; //终端输入的文件描述符，未必等于STDIN_FILENO，因为标准输入有可能被重定向
int TerminalOut; //终端输出的文件描述符，未必等于STDOUT_FILENO，因为标准输出有可能被重定向
char buffer[1024]; //用来存储当前指令
int States[1024]; //设置三种状态，0为空，1为后台运行，2为被挂起
int Jobs[1024]; //子进程的pid
int argc; //参数的总个数
int PID; //主进程号
int SubPID; //子进程号
int Front, Rear; //作为进程表的头尾
int State; //0表示错误，1表示正确

//用来更改路径目录
void cd(string cmd[], int ParaNum);

//用来清屏
void clr(string cmd[], int ParaNum);

//用来显示目录
void dir(string cmd[], int ParaNum);

//用来显示文本
void echo(string cmd[], int ParaNum);

//用来执行命令
void exec(string cmd[], int ParaNum);

//用来列出所有环境变量
void set(string cmd[], int ParaNum);

//用来退出系统
void exit(string cmd[], int ParaNum);

//用来显示路径
void pwd(string cmd[], int ParaNum);

//用来打开帮助手册
void help(string cmd[], int ParaNum);

//用来删除环境变量
void unset(string cmd[], int ParaNum);

//用来显示时间
void time(string cmd[], int ParaNum);

//用来输出屏蔽字或更改屏蔽字的值
void umask(string cmd[], int ParaNum);

//用来测试得出结果
```



```

void test(string cmd[], int ParaNum);

//
void bg(string cmd[], int ParaNum);

//将被挂起或后台运行的作业转移至前台
void fg(string cmd[], int ParaNum);

//输出所有进程
void jobs(string cmd[], int ParaNum);

//移位
void shift(string cmd[], int ParaNum);

//用来记录都需要的全局变量值
void Initialize(int Argc,char * Argv[]);

void ExecSingleCMD(string cmd[], int ParaNum, bool WithFork);

void Exec(string CMD);

void ExecMultipCMD(string cmd[], int ParaNum, bool WithFork);

string JobString(int JobIndex, int Finished);

int main(int Argc, char * Argv[])
{
    Initialize(Argc,Argv);
    //设置为无限循环地去执行
    for (;)
    {
        if (InIsTerminal){
            //这里需要打印信息，格式设置参考了开源代码
            sprintf(buffer, "\e[1;32m%s@\s\e[0m:\e[1;34m%s\e[0m$ ",
                UserName.c_str(),
                HostName.c_str(),
                PWD.c_str());
        }
        string tempinformation = buffer;
        write(Terminalout, buffer, tempinformation.length());//把一些符号输出到定义
        好的终端里
        for (int i = 0; ; i++){
            if (read(STDIN_FILENO, buffer + i, 1) <= 0){//循环寻找EOF
                buffer[i] = '\0';
                break;
            }
            else if (buffer[i] == '\n'){
                buffer[i] = '\0';
                break;
            }
        }
        //执行指令
        Exec(buffer);
    }
}

```

```

}

void Initialize(int Argc, char * Argv[])
{
    char temp[1024]={0};
    argc = Argc;
    //将两个值赋好
    for (int i = 0; i < argc; i++)
        argv[i] = Argv[i];
    //得到进程号
    PID = getpid();
    //初始先把子进程设置为不存在
    SubPID = -1;
    //得到用户名
    UserName = getenv("USERNAME");
    //获取主目录地址
    HomeDir = getenv("HOME");
    //获取当前地址
    PWD = getenv("PWD");
    //得到当前主机名
    gethostname(temp, 1024);
    HostName = temp;
    //将参数path的符号连接内容到参数buf所指的内存空间，返回的内容不是以NULL作字符串结尾，但会将字符串的字符数返回。
    //若参数bufsiz小于符号连接的内容长度，过长的内容会被截断
    int len = readlink("/proc/self/exe", temp, 1024);
    temp[len] = '\0';
    ShellPath = temp;
    setenv("SHELL", ShellPath.c_str(), 1);
}

//第一层解析指令：处理"&"后台运行符号
void Exec(string CMD){
    //执行之前，先扫描Jobs表，输出已完成的进程，并更新Jobs表
    for (int i = Front; i < Rear; i++){
        if (States[i] && waitpid(Jobs[i], NULL, WNOHANG) == Jobs[i]) {
            if (InIsTerminal){//只有输入来自终端的时候才要打印子进程表
                string str = JobString(i, 1);
                write(TerminalOut, str.c_str(), str.length());//如果是在终端，要做输出
            }
            States[i] = 0;
            if (Front == i) Front++;
        }
    }
    if (Front == Rear) Front = Rear = 0;
    //切割字符串
    stringstream sstm;
    int ParaNum = 0;//用空白字符分隔后得到的参数个数
    string cmd[1024]; //切割的结果
    sstm << CMD;
    while(1){
        cmd[ParaNum] = "";
        sstm >> cmd[ParaNum];
    }
}

```

```

        if (cmd[ParaNum] == "") break;
        ParaNum++;
    }
    //指令串分隔完毕。检查末尾是否是&, 表示后台运行
    if (ParaNum > 0 && cmd[ParaNum - 1] == "&") {
        ParaNum--;
        int pid = fork();//调用fork创建一个新的进程
        if (pid){
            //父进程, 存储子进程的id, 指令等信息
            //因为带了&, 交给子进程做
            Jobs[Rear] = pid;
            States[Rear] = 1;
            CMDInfo[Rear] = "";
            for (int i = 0; i < ParaNum; i++) CMDInfo[Rear] += cmd[i] + " ";
            CurrentCMD = CMDInfo[Rear];
            Rear++;
            if (InIsTerminal){//只有输入来自终端的时候才要打印子进程表
                string str = JobString(Rear - 1, 0);
                write(TerminalOut, str.c_str(), str.length());//如果是在终端, 要做输出
            }
        }else{//子进程, 执行指令
            setpgid(0, 0);//使子进程单独成为一个进程组。
            ExecMultipCMD(cmd, ParaNum, false);//在该子进程组里运行
            exit(0);
        }
    }else{//没有&符号, 前台运行
        CurrentCMD = CMD;
        ExecMultipCMD(cmd, ParaNum, true);
    }
}

```

//第二层解析指令: 执行多条用管道符分隔的指令。单条指令交给ExecSingleCMD函数

```

void ExecMultipCMD(string cmd[], int ParaNum, bool WithFork){
    //如果整个指令串都没有管道符, 则直接在当前进程中执行
    bool PipeFlag = false;
    for (int i = 0; i < ParaNum; i++){
        if (cmd[i] == "|"){
            PipeFlag = true;
            break;
        }//寻找是不是被管道符给分开的
    }
    if (PipeFlag == false){
        ExecSingleCMD(cmd, ParaNum, WithFork);
        return;
    }
    //否则, 生成一个子进程, 执行用管道符连接的多条指令
    if (WithFork) SubPID = fork();
    if (WithFork && SubPID){//父进程
        while (SubPID != -1 && !waitpid(SubPID, NULL, WNOHANG));//先让子进程做完
        SubPID = -1;
    }else{//子进程
        //标记上一个管道符的位置
        int LastPipe = -1;
        //管道符

```

```

    int PipeFD[2];
    const int MaxPipe = 1024;
    int PipePid[MaxPipe];
    int PipeCNT;
    //在末尾临时添加一个管道符
    cmd[ParaNum++] = "|";
    //扫描，找出所有的管道符
    PipeCNT = 0;
    for (int i = 0; i < ParaNum; i++)
        if (cmd[i] == "|"){//遇到管道符，把LastPipe + 1到i - 1之间的指令提取出来，
//作为子进程执行

            //创建管道
            if (LastPipe == -1) { //第一个遇到的管道符
                PipeFD[0] = STDIN_FILENO;
                PipeFD[1] = -1;
                pipe(PipeFD);
            }
            //拷贝进程
            PipePid[0] = fork();
            if (PipePid[0] == 0){ //子进程
                //重定向
                dup2(PipeFD[0], STDIN_FILENO);
                dup2(PipeFD[1], STDOUT_FILENO);
                close(PipeFD[1]); close(PipeFD[0]);
                //执行指令
                ExecSingleCMD(cmd , i , false);
                exit(0);
            }
        }
    close(PipeFD[0]);
    //等待所有子进程完成
    waitpid(PipePid[0], NULL, 0);
    exit(0);
}
}

```

//第三层解析指令：执行单条指令，可能包含重定向

```

void ExecSingleCMD(string cmd[], int ParaNum, bool WithFork){
    //备份三个标准输入输出
    int InputFD = dup(STDIN_FILENO), OutputFD = dup(STDOUT_FILENO), ErrorFD =
dup(STDERR_FILENO);
    int InputFDNew = -1, OutputFDNew = -1, ErrorFDNew = -1;
    //搜索重定向符号
    string InputFile = "", OutputFile = "", ErrorFile = "";
    for (int i = ParaNum - 2; i >= 0; i--)
        { //从可能出现的地方开始
            //输入重定向
            if (cmd[i] == "<"){
                if (InputFile != ""){
                    Message = "MyShell: Expected at most 1 input redirection.\n";
                    break;
                }
                InputFile = cmd[i + 1];
                //将重定向的目标文件名保存到inputfile，并用open以O_RDONLY方式打开文件
                InputFDNew = open(InputFile.c_str(), O_RDONLY);
            }
        }
}

```

```

        if(InputFDNew < 0){
            Message = "MyShell: Unable to open " + InputFile + ".\n";
            break;
        }
        dup2(InputFDNew, STDIN_FILENO); //把这个文件描述符，用标准输入代替一下
        close(InputFDNew); //关闭
        ParaNum = i;
    }
    //输出重定向，追加模式
    else if (cmd[i] == ">>"){
        if (OutputFile != ""){
            Message = "MyShell: Expected at most 1 output redirection.\n";
            break;
        }
        OutputFile = cmd[i + 1];
        //将重定向的目标文件名保存到outputfile，并用open以追加模式
        OutputFDNew = open(OutputFile.c_str(), O_WRONLY | O_CREAT |
O_APPEND, 0666);
        if(OutputFDNew < 0){
            Message = "MyShell: Unable to open " + OutputFile + ".\n";
            break;
        }
        dup2(OutputFDNew, STDOUT_FILENO);
        close(OutputFDNew);
        ParaNum = i;
    }
    //输出重定向，覆盖模式
    else if (cmd[i] == ">"){
        if (OutputFile != ""){
            Message = "MyShell: Expected at most 1 output redirection.\n";
            break;
        }
        OutputFile = cmd[i + 1];
        //将重定向的目标文件名保存到outputfile，并用open以覆盖模式
        OutputFDNew = open(OutputFile.c_str(), O_WRONLY | O_CREAT | O_TRUNC,
0666);
        if(OutputFDNew < 0){
            Message = "MyShell: Unable to open " + OutputFile + ".\n";
            break;
        }
        dup2(OutputFDNew, STDOUT_FILENO);
        close(OutputFDNew);
        ParaNum = i;
    }
}
if (!State){ //如果前面步骤没有出错->state=0,那么说明没有问题，可以继续做
    //解析指令
    if (ParaNum == 0) {
        Answer = "";
        Message = "";
        State = 0;
    }else if (cmd[0] == "cd") {
        cd(cmd + 1, ParaNum - 1);
    }else if (cmd[0] == "clr") {
        clr(cmd + 1, ParaNum - 1);
    }
}

```

```

}else if (cmd[0] == "dir") {
    dir(cmd + 1, ParaNum - 1);
}else if (cmd[0] == "echo") {
    echo(cmd + 1, ParaNum - 1);
}else if (cmd[0] == "exec") {
    exec(cmd + 1, ParaNum - 1);
}else if (cmd[0] == "exit") {
    exit(cmd + 1, ParaNum - 1);
}else if (cmd[0] == "help") {
    help(cmd + 1, ParaNum - 1);
}else if (cmd[0] == "pwd") {
    pwd(cmd + 1, ParaNum - 1);
}else if (cmd[0] == "set") {
    set(cmd + 1, ParaNum - 1);
}else if (cmd[0] == "shift") {
    shift(cmd + 1, ParaNum - 1);
}else if (cmd[0] == "test") {
    test(cmd + 1, ParaNum - 1);
}else if (cmd[0] == "umask") {
    umask(cmd + 1, ParaNum - 1);
}else if (cmd[0] == "unset") {
    unset(cmd + 1, ParaNum - 1);
}else if (cmd[0] == "exit") {
    exit(cmd + 1, ParaNum - 1);
}else if (cmd[0] == "time") {
    time(cmd + 1, ParaNum - 1);
}else if (cmd[0] == "bg") {
    bg(cmd + 1, ParaNum - 1);
}else if (cmd[0] == "fg") {
    fg(cmd + 1, ParaNum - 1);
}else if (cmd[0] == "jobs") {
    jobs(cmd + 1, ParaNum - 1);
}else{//其他命令，表示程序调用
    Message = "";
    Answer = "";
    State = 0;
    if (WithFork){
        //fork将父进程拷贝一份变成子进程
        SubPID = fork();
        if (SubPID == 0){//子进程
            //设置PARENT环境变量
            setenv("PARENT", ShellPath.c_str(), 1);
            exec(cmd, ParaNum);
        }
        //父进程等待子进程完成
        while (SubPID != -1 && !waitpid(SubPID, NULL, WNOHANG));
        SubPID = -1;
    }else{
        //设置PARENT环境变量
        setenv("PARENT", ShellPath.c_str(), 1);
        exec(cmd, ParaNum);
    }
}
}
}
//输出结果

```

```

write(STDOUT_FILENO, Answer.c_str(), Answer.length());
write(STDERR_FILENO, Message.c_str(), Message.length());
//恢复到标准输入和标准输出
dup2(InputFD, STDIN_FILENO);
close(InputFD);
dup2(OutputFD, STDOUT_FILENO);
close(OutputFD);
}

```

```

void cd(string cmd[], int ParaNum){
    Message = Answer = "";
    //如果参数过多
    if (ParaNum > 1)
    {
        State = 0;
        Message = "cd : too many parameters";
        return;
    }
    //如果没有参数，或者参数是特殊的~, 那么要做的是回到主目录
    if (ParaNum == 0 || (ParaNum == 1 && cmd[0] == "~"))
    {
        State = 1;
        //首先是改变目录
        chdir(HomDir.c_str());
        //更新PWD环境变量
        PWD = HomDir;
        //设置PWD值为现在的PWD值
        setenv("PWD", PWD.c_str(), 1);
    }
    else if (ParaNum == 1)
    { //参数个数为1
        if (chdir(cmd[0].c_str())){
            State = 0;
            Message = "cd: can't find the path\n";
        }else{//chdir调用成功
            //绝对路径的获取就是直接看cmd就行
            State = 1;
            PWD = cmd[0];
            setenv("PWD", PWD.c_str(), 1);
        }
    }
}
}

```

```

void clr(string cmd[], int ParaNum){
    Message = Answer = "";
    //如果在不该有参数时有了参数
    if (ParaNum > 0)
    {
        State = 0;
        Message = "clr : too many parameters\n";
        return;
    }
    //实现清屏功能

```

```

        else system("clear");
        State = 1;
    }

void dir(string cmd[], int ParaNum){
    Message = Answer = "";
    if (ParaNum >= 2){//参数过多, 报错
        State = 0;
        Message = "dir: Too many parameters.\n";
    } else {
        //当前的工作目录
        string Dictionary;
        if (ParaNum == 0) Dictionary = PWD;
        else Dictionary = cmd[0];
        DIR *pDir;
        struct dirent* ptr;
        //打开目录
        if(!(pDir = opendir(Dictionary.c_str()))){
            //错误信息
            State = 0;
            Message = "dir: Folder " + cmd[0] + " doesn't Exist.\n";
            return;
        }
        string Name[1024];
        int Cnt = 0;
        //读取文件信息
        while((ptr = readdir(pDir)) != NULL)
            Name[Cnt++] = ptr->d_name;
        //关闭文件
        closedir(pDir);
        //对找到的文件名进行排序
        sort(Name+0, Name+Cnt);
        State = 1;
        for (int i = 0; i < Cnt; i++)
            Answer += Name[i]+"\n";
    }
}

void echo(string cmd[], int ParaNum){
    Message = Answer = "";
    //遍历每个参数
    for (int i = 0; i < ParaNum; i++){
        //设置一个value, 要承接改变
        string value = cmd[i];
        //value前有双引号, 要一直去找到下一个双引号, 并且不做输出
        if (cmd[i][0]==''){
            while (cmd[i][cmd[i].length()-1]!='' && i<ParaNum)
                value += cmd[i++];
            //如果找不到双引号, 报错
            if (i == ParaNum) {
                State = 0;
                Message = "echo : wrong for less character";
                return;
            }
            //下面是已经找到配套的了

```



```

        value += cmd[i];
        //处理value, 去掉前后的两个双引号
        value = value.substr(1,value.length()-2);
    }
    //下面再考虑一种用$开头的特殊情况
    else if (cmd[i][0] == '$') {
        value = cmd[i][0];
        //截断
        value = value.substr(1,value.length()-1);
        int temp = 0;
        //去取出这个整数
        for (int k = 0; k < value.length(); i++)
            temp = temp*10 + value[i] - '0';
        value = argv[temp];
    }
    Answer += value;
}
State = 1;
Answer += "\n";
}

void exec(string cmd[], int ParaNum){
    Message = Answer = "";
    //如果参数个数不够
    if (ParaNum == 0)
    {
        State = 0;
        Message = "exec : no parameter";
        return;
    }
    //参数构造, 一个指针数组 p200所传授
    char * arg[ParaNum + 1];
    for (int i = 0; i < ParaNum; i++)
        //使用强制转换
        arg[i] = const_cast<char *>(cmd[i].c_str());
    arg[ParaNum] = NULL;
    execvp(cmd[0].c_str(), arg);
    State = 0;
    Message = "exec : fail to execute";
}

void exit(string cmd[], int ParaNum){
    Message = Answer = "";
    //判断参数数量
    if (ParaNum != 0)
    {
        State = 0;
        Message = "exit : too many parameter";
        return;
    }
    State = 1;
    //终止所有, 释放内存
    _exit(0);
}

```

```

void pwd(string cmd[], int ParaNum){
    Message = Answer = "";
    if (ParaNum != 0)
    {
        State = 0;
        Message = "pwd : too many parameter";
        return;
    }
    char buf[1024];
    //取出地址,判断是否可行
    if(!getcwd(buf, 1024))
    {
        State = 0;
        Message = "pwd : wrong";
        return;
    }
    //可行时更改需要输出的内容
    PWD = buf;
    Answer = buf;
    State = 1;
}

void help(string cmd[], int ParaNum){
    Message = Answer = "";
    //如果参数过多,报错
    if (ParaNum != 0)
    {
        State = 0;
        Message = "help : too many parameter";
        return;
    }
    //先打开help.txt
    ifstream infile;
    infile.open("help.txt", ios::in);
    //打开失败
    if (!infile.is_open())
    {
        State = 0;
        Message = "help : can't open";
        return;
    }
    string buf;
    //打开成功时,放到Answer中
    State = 1;
    while (getline(infile,buf))
        Answer += buf + '\n';
}

void set(string cmd[], int ParaNum){
    Message = Answer = "";
    extern char** environ;//环境变量表
    //输出所有环境变量
    for(int i = 0; environ[i] != NULL; i++)
        Answer = Answer + environ[i] + "\n";
}

```

```

}

void shift(string cmd[], int ParaNum){
    Message = Answer = "";
    //如果参数过多, 报错
    if (ParaNum > 1)
    {
        State = 0;
        Message = "shift : too many parameter";
        return;
    }
    int cnt = 0;
    //确定移动的到底是多少位
    if (ParaNum == 0) cnt = 1;
    else {
        string value = cmd[0];
        for (int i = 0; i < value.length(); i++)
            cnt = cnt*10 + value[i]-'0';
    }
    //开始移动,先确定个数
    argc = argc-cnt <= 0 ? 1 : argc-cnt;
    //把后面的都移过来
    for (int i = 1; i < argc; i++)
        argv[i] = argv[i + cnt];
    State = 1;
}

void unset(string cmd[], int ParaNum){
    Message = Answer = "";
    //判断参数个数是否正确
    if (ParaNum > 1 || ParaNum == 0){
        State = 0;
        Message = "unset : wrong";
        return;
    }
    State = 1;
    unsetenv(cmd[0].c_str());
}

void time(string cmd[], int ParaNum){
    Message = Answer = "";
    if (ParaNum > 0)
    {
        State = 0;
        Message = "time : wrong";
        return;
    }
    State = 1;
    time_t timer;//time_t就是long int 类型
    struct tm *tblock;
    timer = time(NULL);//这一句也可以改成time(&timer);
    tblock = localtime(&timer);
    Answer = asctime(tblock);
}

```

```

void umask(string cmd[], int ParaNum){
    Message = Answer = "";
    State = 1;
    if (ParaNum > 1){//参数过多, 报错
        State = 0;
        Message = "umask: Too many parameters.\n";
        return;
    } //无参, 表示显示umask的值
    if (ParaNum == 1){
        //提取这个参数准备操作
        string value = cmd[0];
        int num = 0;
        //先判断是否这个数字位数不对
        if (value.length() > 4)
        {
            State = 0;
            Message = "umask: wrong";
            return;
        }
        //判断最高位是否在范围
        if (value.length() == 4)
        {
            if (value[3] != '0')
            {
                State = 0;
                Message = "umask :wrong";
                return;
            }
        }
        //判断其他位是否在范围
        for (int i = 0; i < value.length() && i<3; i++)
        {
            if (value[i] < '0' || value[i] > '7')
            {
                State = 0;
                Message = "umask : wrong";
                return;
            }
        }
        while (value.length() < 4) value = "0" + value;//补齐到恰好4位
        //化成八进制
        num = ((value[0] - '0') << 9) | ((value[1] - '0') << 6) | ((value[2]
- '0') << 3) | (value[3] - '0');
        umask(num);
    }
    else{
        mode_t currentmode = umask(0);
        umask(currentmode);
        Answer = to_string((currentmode >> 9) & 7) + to_string((currentmode >> 6)
& 7)
                + to_string((currentmode >> 3) & 7) + to_string(currentmode &
7) + "\n";
    }
}

```

```

}

void test(string cmd[], int ParaNum){
    Message = Answer = "";
    State = 1;
    //判断参数个数是否在合理范围内
    if (ParaNum <= 1){
        State = 0;
        Message = "test: Too few parameters.\n";
        return;
    }
    if (ParaNum >= 4){
        State = 0;
        Message = "test: Too many parameters.\n";
        return;
    }
    if (ParaNum == 2){
        //下面对二元运算符，利用课本提供的函数进行判断
        struct stat buf;
        char *ptr;
        if (!stat(const_cast<char *>(cmd[1].c_str()), &buf) < 0){
            State = 0;
            Message = "test : wrong";
            return;
        }
        if (cmd[0] == "-f"){//文件为普通文件
            if (S_ISREG(buf.st_mode))
                Answer = "true";
            else Answer = "false";
        }
        else if (cmd[0] == "-d"){//文件为目录
            if (S_ISDIR (buf.st_mode))
                Answer = "true";
            else Answer = "false";
        }
        else if (cmd[0] == "-c"){//文件为字符型特殊文件
            if (S_ISCHR (buf.st_mode))
                Answer = "true";
            else Answer = "false";
        }
        else if (cmd[0] == "-b"){//文件为块特殊文件
            if (S_ISBLK (buf.st_mode))
                Answer = "true";
            else Answer = "false";
        }
        else if (cmd[0] == "-h" || cmd[0] == "-L"){//文件为符号链接
            if (S_ISLNK (buf.st_mode))
                Answer = "true";
            else Answer = "false";
        }
        else if (cmd[0] == "-p"){//文件为命名管道
            if (S_ISSOCK(buf.st_mode))
                Answer = "true";
            else Answer = "false";
        }
    }
}

```

```

else if (cmd[0] == "-e"){//文件存在
    struct stat buf;
    int ret = lstat(cmd[1].c_str(), &buf);
    if (ret == 0) Answer = "true\n";
    else Answer = "false\n";
}else if (cmd[0] == "-r"){//文件可读
    struct stat buf;
    int ret = lstat(cmd[1].c_str(), &buf);
    if (ret == 0 && access(cmd[1].c_str(), R_OK)) Answer = "true\n";
    else Answer = "false\n";
}else if (cmd[0] == "-w"){//文件可写
    struct stat buf;
    int ret = lstat(cmd[1].c_str(), &buf);
    if (ret == 0 && access(cmd[1].c_str(), W_OK)) Answer = "true\n";
    else Answer = "false\n";
}else if (cmd[0] == "-x"){//文件可执行
    struct stat buf;
    int ret = lstat(cmd[1].c_str(), &buf);
    if (ret == 0 && access(cmd[1].c_str(), X_OK)) Answer = "true\n";
    else Answer = "false\n";
}
}

else if (ParaNum == 3){//二元运算符，有且仅有3个参数
    string valueStr1 = cmd[0];
    string valueStr2 = cmd[2];
    if (cmd[1] == "-ef"){//文件1和文件2的设备和inode相同
        struct stat buf1, buf2;
        int ret1 = lstat(valueStr1.c_str(), &buf1);
        int ret2 = lstat(valueStr2.c_str(), &buf2);
        if (ret1 == 0 && ret2 == 0 && buf1.st_dev == buf2.st_dev &&
buf1.st_ino == buf2.st_ino) Answer = "true\n";
        else Answer = "false\n";
    }else if (cmd[1] == "-nt"){//文件1比文件2更新
        struct stat buf1, buf2;
        int ret1 = lstat(valueStr1.c_str(), &buf1);
        int ret2 = lstat(valueStr2.c_str(), &buf2);
        if (ret1 == 0 && ret2 == 0 && (buf1.st_mtim.tv_sec>
buf2.st_mtim.tv_sec || buf1.st_mtim.tv_nsec> buf2.st_mtim.tv_nsec))
            Answer = "true\n";
        else Answer = "false\n";
    }else if (cmd[1] == "-ot"){//文件1比文件2更旧
        struct stat buf1, buf2;
        int ret1 = lstat(valueStr1.c_str(), &buf1);
        int ret2 = lstat(valueStr2.c_str(), &buf2);
        if (ret1 == 0 && ret2 == 0 && (buf1.st_mtim.tv_sec<
buf2.st_mtim.tv_sec || buf1.st_mtim.tv_nsec< buf2.st_mtim.tv_nsec)) Answer =
"true\n";
        else Answer = "false\n";
    }else if (cmd[1] == "-eq" || cmd[1] == "-ge" || cmd[1] == "-gt" ||
cmd[1] == "-le" || cmd[1] == "-lt" || cmd[1] == "-ne"){
        int x1 = 0, x2 = 0;
        //先判断要转换的数字是不是整数
        for (int i = 0; i < valueStr1.length(); i++)
            if (valueStr1[i] < '0' || valueStr1[i] > '9')
                {

```

```

        State = 0;
        Message = "Wrong for not integer";
        return;
    }
    for (int i = 0; i < valueStr2.length(); i++)
        if (valueStr2[i] < '0' || valueStr2[i] > '9')
        {
            State = 0;
            Message = "Wrong for not integer";
            return;
        }
    //如果到这，证明是两个整数，可以进行后续判断的
    for (int i = 0; i < valueStr1.length(); i++)
        x1 = x1 * 10 + valueStr1[i] - '0' ;
    for (int i = 0; i < valueStr2.length(); i++)
        x2 = x2 * 10 + valueStr2[i] - '0' ;
    //switch是不能对string类型做的
    if (cmd[1] == "-eq"){
        if (x1 == x2) Answer = "true\n";
        else Answer = "false\n";
    }
    if (cmd[1] == "-ge"){//整数>=
        if (x1 >= x2) Answer = "true\n";
        else Answer = "false\n";
    }
    else if (cmd[1] == "-gt"){//整数>
        if (x1 > x2) Answer = "true\n";
        else Answer = "false\n";
    }
    else if (cmd[1] == "-le"){//整数<=
        if (x1 <= x2) Answer = "true\n";
        else Answer = "false\n";
    }
    else if (cmd[1] == "-lt"){//整数<
        if (x1 < x2) Answer = "true\n";
        else Answer = "false\n";
    }
    else if (cmd[1] == "-ne"){//整数!=
        if (x1 == x2) Answer = "true\n";
        else Answer = "false\n";
    }
}
else {
    State = 0;
    Message = "test: Unknown command " + cmd[0] + ".\n";
}
}
}
}

```

//用于转换成可以输出的格式

```

string JobString(int JobIndex, int Finished){
    if (JobIndex < Front || JobIndex >= Rear || States[JobIndex] != 1 &&
        States[JobIndex] != 2) return "";
    return "[" + to_string(JobIndex + 1) + "]\t" + to_string(Jobs[JobIndex]) +
        "\t\t" + ((Finished) ? "Finish" : (States[JobIndex] == 1) ? "Running" :
        "Hanging") + "\t\t" + CMDInfo[JobIndex] + "\n";
}

```

```

void bg(string cmd[], int ParaNum){
    Message = Answer = "";
    State = 1;
}

```

```

//无参，列出所有正在后台运行的进程
if (ParaNum == 0){
    for (int i = Front; i < Rear; i++){
        if (States[i] == 1)
            Message += JobString(i,0);
        if (Message == "") {
            State = 0;
            Message = "bg: No mission is running at the background.\n";
        }
        return;
    }
}
//根据参数列表，把对应的任务移至后台运行
for (int i = 0; i < ParaNum; i++){
    int workID = 0;
    //将字符串转换为整数
    for (int j = 0; j < cmd[i].length(); j++){
        if (cmd[i][j] >= '0' && cmd[i][j] <='9'){
            workID = workID * 10 + cmd[i][j] - '0';
        } else{
            State = 0;
            Message = "bg : not integer";
            return;
        }
    }
    //若不存在指定作业号的任务
    if (workID > Rear || workID <= Front || States[workID - 1] == 0){
        State = 0;
        Message += "bg: " + cmd[i] + ": No mission.\n";
        continue;
    }
    //若已经在后台运行
    if (States[workID - 1] == 1){
        State = 0;
        Message += "bg: " + cmd[i] + ": Already running at the
background.\n";
        continue;
    }
    //更新Jobs表并发送信号
    States[workID - 1] = 1;
    kill(Jobs[workID - 1], SIGCONT);
}
}

void fg(string cmd[], int ParaNum){
    Message = Answer = "";
    State = 1;
    if (ParaNum == 0 && ParaNum >= 2){
        State = 0;
        Message = "fg: wrong input.\n";
        return;
    }
    //根据参数列表，把对应的任务移至前台运行
    int workID = 0;
    //将字符串转换为整数
    for (int j = 0; j < cmd[0].length(); j++){
        if (cmd[0][j] >= '0' && cmd[0][j] <='9'){

```



```

        workID = workID * 10 + cmd[0][j] - '0';
    } else{
        State = 0;
        Message = "bg : not integer";
        return;
    }
    if (workID > Rear || workID <= Front || States[workID - 1] == 0){
        State = 0;
        Message = "fg: " + cmd[0] + ": No mission.\n";
        return;
    }
    CurrentCMD = CMDInfo[workID - 1];
    if (InIsTerminal){//把指令内容输出到屏幕, 告知用户前台作业的信息
        write(TerminalOut, CMDInfo[workID - 1].c_str(), CMDInfo[workID - 1].length());
        write(TerminalOut, "\n", 1);
    }
    //更新Jobs表
    States[workID - 1] = 0;
    SubPID = Jobs[workID - 1];
    if (Rear == workID && Front == workID - 1) Front = Rear = 0;//复原
    else if (Front == workID - 1) Front++;//到下一个
    else if (Rear == workID) Rear--;//到前一个
    //发送信号
    setpgid(SubPID, getpid());//设置进程组, 使子进程进入前台进程组
    kill(SubPID, SIGCONT);
    //等待子进程完成
    while (SubPID != -1 && !waitpid(SubPID, NULL, WNOHANG));
    SubPID = -1;
}

//打印进程
void jobs(string cmd[], int ParaNum){
    Message = Answer = "";
    State = 1;
    if (ParaNum > 0){
        State = 0;
        Message = "jobs: Too many parameters.\n";
        return;
    }
    for (int i = Front; i < Rear; i++)
        Answer += JobString(i, 0);
}

```

## 八.实验心得

在本次实验中, 我把对每道题自己做题时候的想法, 还有自己认为的不足总结如下:

首先, 第一道题, 这道题是关于makefile内容的, 由于在面向对象的程序设计里, 进行多个文件编程的时候有时需要自己编写简单的makefile, 所以对语法有一定的基础了解, 在解答这道题的时候回答问题没有特别大的难度。

第二道题，首先碰到的一个语法小报错出在main上，这个地方改成正常的int main()应该就能解决问题，第二问要注意先删除之前生成的文件，然后要注意链接上数学库，才能正常运行，在做完这道题后我试验了一下，发现连上需要特别注意的是题目中说的-lm。

第三道题，思路其实是比较直观的，就是统计个数，只要打开路径逐个统计就好。这里的三个注意点是，一个是路径名称，我这里其实是默认了搜索的就是主目录下的文件了，一个是要给程序授予权限777，否则无法操作，最后是bcount是根据ls命令得到的，这点我一开始没想到，本来以为也是简单的同格式的命令，结果发现没有这样的命令，和同学在学校里讨论了才想出的。

第四道题，这个题我觉得也是让我们熟悉一下shell，为后面做准备。然后这道题的算法思路其实很清晰了，就是做一个回文判断，但是还要去除非字母，我就直接用了自带的一个操作str=\${1//[a-zA-Z]}，然后进行了一个遍历，感觉不是很困难。

第五道题，这是第一个让我感觉十分头疼的问题，主要是vim编辑器的功能确实很复杂，我对这个语言的掌握程度也不高，本来想做一个尽可能像vim的，但是在如何修改已有内容中停留了很久也没找到方法。我退而求其次，想能不能对已有内容不实现删除修改，只是对每一行进行至少一个增加内容的修改。但是由于我的算法设计的是，每次从文件内读，这样想读一行改一行，碰到了一个问题就是想用read -p "\$line" new，把读入的行输出再从命令端读一个新的字符串，但是发现这样的read还是会自动去找文件读，最后只能放弃。

最后我实现的就只剩下：两种模式，一种编辑模式一种控制模式，编辑模式能自由增添新的内容，控制模式用暑假搜到的sed实现了一下基本的控制，但是其中我设置了A和esc的转换，感觉这个还是很有意思的，特别是对A的静默读和对esc这个特殊符号的判断处理。此外，由于我是把编辑模式中新读入的内容给加到文本中，再进控制模式，所以对于重定向也相当于了解深了一些吧。最后，还是很希望到时候可以看到一些同学的思路，是如何编辑已有内容的，网上搜索不到一些相关内容，希望开学能明白。

第六道题，最后一道题我还是选择了第一个比较大的myshell，主要是第二道题想了想，主要就是用UNIX那本书里，然后字符串判断下参数是哪个调用一些函数操作，然后就想试一下第一题，也是对linux的控制语句有个更好的了解。

我尽力去实现了我认为能够理解的部分，然后在这里必须指出我阅读了一些开源代码，我学习了他的代码框架结构，他的实现中是把一条语句分为了从&命令->|命令->重定向命令->单条命令，这样的一个顺序去实现的，我沿用了这个思路，因为在我的思考中，笼统的概括这里就是要让命令在有前提的情况下，先让前提实现完，再进行。但是由于我还没有接触过进程的概念，所以这里我只做了最基础的一个管道的情况，这样就只要参考UNIX书给出的提示，就可以完成代码了。

另外对于单条命令的实现里，我结合了UNIX书和网上的教学代码，基本属于摸着石头过河的状态，去思考其中需要我们实现的状态，所以我其实做了一些简化，比如test我只是挑选了比较好实现的功能去实现，对于set只考虑了无参数，对于shift其实有一定的bug还没能调完。但是总体来说，我还是结合了课本知识，对这些单条命令有了一个很好的理解。

我觉得这个代码，能让我对于shell实现的流程有一个深入的理解，虽然我做的很多都是简化版本，但是对于命令究竟实现之前要完成什么，我更明白了。另外这个代码其实是我在数据库系统做的cmu15545后第二个很大的程序，所以在其中构建框架，再一点点实现，还是很有成就感的。这里额外值得记录的是，我是先写了单条命令，然后单条命令自己写了一个简化代码去测试，感觉这个及时确定自己是否正确是一个很好的习惯。