5.1（30分）**使用任何一种程序设计语言实现一个shell 程序的基本功能。**

shell 或者命令行解释器是操作系统中最基本的用户接口。写一个简单的shell 程序——**myshell**，它具有以下属性：

(一) 这个shell 程序必须支持以下内部命令：bg、cd 、clr、dir、echo 、exec 、exit 、environ、fg 、help、jobs 、pwd 、quit、set 、shift 、test 、time 、umask、unset。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) environ ——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) quit ——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：

1） 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（见上面（四） 中的命令行）。

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数。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。

myshell设计文档

一、设计思想：

在准备myshell的设计之前，我首先详细研究了bash的命令结构，观察分析了bash相关内建命令的执行结果。首要任务是选择一门编程语言，我在查阅了资料发现，对于进程控制，目录跳转等操作，有很多Linux下的C语言系统调用函数可以直接进行处理，方便我们对myshell的设计，再加上C语言是我最早接触的编程语言，于是我选择了用C语言进行本次myshell的设计。

在确定编程语言之后，仿照bash对于命令的处理过程，我将myshell的主程序设计为一个无限循环，循环的第一步是打印命令提示符，第二步是读取用户的输入，第三步是对用户的输入进行拆分保存，最后一步是执行相应的命令。这个思路流程还是比较清晰的。

在主程序的框架设计完成之后，着重需要完成的就是执行命令的模块，对于这个模块的实现也分步骤进行。首先对需要实现的内建命令进行一一实现并测试，这当中借助许多系统调用函数可以简化命令的实现，例如使用getcwd()函数可以直接获取当前目录，使用chdir()函数可以进行目录间的切换，这两个函数正好可以用在cd命令的实现中。

在内建命令实现并调试完成后，开始对外部命令的执行进行实现。我的思路是先检查用户输入是否是内建命令，如果不是则默认为外部命令。对于外部命令的执行，利用fork()创建一个新的子进程，在子进程中利用execvp()函数查找并执行相应的外部命令，在主进程中使用waitpid()函数阻塞主进程，等待子进程的退出，即可实现外部命令的执行。

接着考虑myshell对于读取脚本文件并执行其中命令的支持，我的想法也很简单，执行myshell时如果不带参数，则打印命令提示符，且读取输入从标准输入流中读取；而执行myshell时如果带上了一个参数，这个参数为脚本文件的路径，则不打印命令提示符，先尝试打开该脚本文件，如果成功打开，则读取输入从脚本文件中读取，后面的输入拆分和命令执行过程都不变。

然后考虑myshell对于后台命令的实现，这个实现也比较容易，只需要定义一个标志是否为后台命令的变量，如果命令以‘&’结尾，就把这个标志置1。对于需要后台执行的命令，在主进程中仍然使用waitpid()函数，但使用其中的非阻塞选项，这样就可以不阻塞主进程的进行，实现命令的后台执行了。

最后需要重点考虑的是重定向和管道操作的实现，在查阅资料后发现，重定向和管道操作的核心实现原理其实十分类似，管道操作其实也是一种重定向，是将标准输入输出重定向到管道文件的两端，而重定向操作是将标准输入输出重定向到指定的文件，这个操作都可以通过dup2()函数来实现。

对于管道操作的实现，在命令读取的过程中检查到‘|’时，将预先定义的表示是否为管道操作的标志置1。在标志为1的情况下，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，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。之后利用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，命令的输出将输进管道文件。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。

对于重定向的实现，在命令读取的过程中检查到‘<’，‘>’或者‘>>’时，将预先定义的表示是否有重定向的标志置1。在标志为1的情况下，将重定向的目标文件名保存到infile_path或outfile_path中，之后用open()函数用相应的方式打开文件(只读、覆盖写、追加写)，再用dup2()函数用打开的文件流替换相应的标准输出或输入流，即完成了重定向操作，最后还需要将标准输入输出流复原。

按照以上思路，我完成了本次myshell的设计。

二、功能模块：

整个系统主要分为五个模块：信号处理模块、打印命令提示符模块、读取输入模块、分割输入模块、执行命令模块。

其中，信号处理模块主要负责对于子进程返回产生的SIGCHLD信号以及按下ctrl-Z之后产生的SIGTSTP信号进行处理，捕获SIGCHLD信号是为了更新后台进程表，而捕获SIGTSTP信号是为了将当前的进程挂起。

打印命令提示符模块主要负责打印命令提示符，其中要包含用户名，主机名以及当前的路径名。

读取输入模块主要负责读取用户的输入并存储到缓冲区buf中，等待分割输入模块的进一步处理。用户的输入可以通过键盘输入，也可以选择通过脚本文件输入。

分割输入模块主要负责对用户的输入进行拆分，由于规定了不同参数之间需要以空格分隔，所以可以以空格为分隔符对输入进行拆分，在拆分过程中遇到‘>’、‘<’、‘|’等标准性符号时，需要设置相应的重定向标志或者管道标志的值为1，便于执行命令模块的执行。

执行命令模块主要负责对拆分后的命令进行执行，也是myshell的核心模块。该模块根据分割输入模块设置好的一些标志，如是否为管道命令、是否含有输入，输出重定向、是否为后台执行的命令，来分别执行相应的处理。内建命令的执行之间调用相应的函数，外部命令的执行则使用execvp()函数来查找并执行。

三、数据结构与算法：

本次myshell设计中，命令的保存使用了指针数组，数组中的每一项存储以空格隔开的单个项的信息，例如输入ls -l，则数组第一项保存“ls”，第二项保存“-l”。并且将该指针数组定义为全局变量，方便其他函数进行调用

另外，还定义了job结构，如下所示，用来存储任务信息，方便jobs命令的使用。定义了全局变量job* jobs，并且利用shmget()函数创建共享内存，共享内存的地址赋给了jobs，从而实现了jobs在所有进程中都可以访问和修改。

```
//定义结构job，用来存储任务信息
typedef struct job
{
  pid_t pid;
  char jobname[MAX_NAME_LENGTH];
  int type;
  int status;
} job;
```

定义了类型为struct sigaction的结构变量old_action和new_action，如下图所示，用于信号处理。

```
//定义用于信号处理的结构
struct sigaction old_action;
struct sigaction new_action;
```

myshell用户手册

一、内建命令的使用：

1. bg

命令作用： 将最近一个挂起的进程转为后台执行。

使用示例： bg

参数个数： 无参数

2. cd

命令作用： 无参数则显示当前目录，有参数则改变当前目录为参数内容

使用示例： cd

参数个数： 无参数或1个参数

3. clr

命令作用： 清空当前屏幕内容

使用示例： clr

参数个数： 无参数

4. dir

命令作用： 无参数则显示当前目录下的内容，有参数则显示参数所指目录下的内容

使用示例： dir

使用示例： dir /

参数个数： 无参数或1个参数

5. echo

命令作用： 无参数则显示空内容，有参数则显示参数内容

使用示例： echo

使用示例： echo 1 22 oop

参数个数： 无参数或任意多参数

6. exec

命令作用： 使用参数代表的命令替换当前进程

使用示例： exec ls

参数个数： 1个参数

7. exit

命令作用： 退出当前进程

使用示例： exit

参数个数： 无参数

8. environ

命令作用： 显示所有的环境变量

使用示例： environ

参数个数： 无参数

9. fg

命令作用： 将最近的一个后台任务转到前台执行

使用示例： fg

参数个数： 无参数

10. help

命令作用： 显示用户手册

使用示例： help

参数个数： 无参数

11. jobs

命令作用： 显示所有的后台进程

使用示例： jobs

参数个数： 无参数

12. pwd

命令作用： 显示当前路径

使用示例： pwd

参数个数： 无参数

13. quit

命令作用： 退出myshell

使用示例： quit

参数个数： 无参数

14. set

命令作用： 无参数时，显示所有环境变量；有2个参数时，设置第1个参数代表的环境变量的值为第2个参数

使用示例： set

使用示例： set USER Wang

参数个数： 无参数或2个参数

15. shift

命令作用： 从标准输入读入参数(以空格分隔)，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

使用示例： shift

使用示例： shift 2

参数个数： 无参数或1个参数

16. test

命令作用： 可以进行一些字符串、数字的比较，包括两字符串是否相等，两数字之间的大小关系是否成立(相等，不相等，大于，小于，大于等于，小于等于)

使用示例： test abc = abc

使用示例： test abc != abc

使用示例： test 2 -eq 2

使用示例： test 2 -ne 2

使用示例： test 2 -gt 2

使用示例： test 2 -ge 2

使用示例： test 2 -lt 2

使用示例： test 2 -le 2

参数个数： 3个参数

17. time

命令作用： 显示当前时间

使用示例： time

参数个数： 无参数

18. umask

命令作用： 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

使用示例： umask

使用示例： umask 0222

参数个数： 无参数或1个参数

19. unset

命令作用： 将参数所指的环境变量的值取消

使用示例： unset USER

参数个数： 1个参数

二、外部命令的执行：

简单描述： 除了内建命令之外，myshell还能够自动查找并执行外部命令

实现原理： 其他的命令行输入被解释为程序调用，myshell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，如果没有找到则会输出相应的错误提示信息

使用示例： ls -l

使用示例： gedit test.txt

三、脚本文件的执行：

简单描述： 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

实现原理： 在检查到命令行参数时，myshell将打开参数代表的文件，之后用Read_command()函数读取命令时，从文件流中读取内容到buf，而不是从标准输入流中读取，如果打开文件失败则输出相应的错误提示信息

使用示例： myshell test.sh

四、I/O重定向：**

简单描述： myshell能够支持I/O重定向，在输入要执行的命令后，输入‘<’，再接输入重定向到的文件inputfile，myshell在执行命令时就会从inputfile中读取而非从标准输入中读取；输入‘>’或者‘>>’再接输出重定向到的文件outputfile，myshell就会将命令执行的结果输出到outputfile中而非输出到屏幕上，其中‘>’表示覆盖写，‘>>’表示追加写

实现原理： 在命令读取的过程中检查到‘<’，‘>’或者‘>>’时，将预先定义的表示是否有重定向的标志置1。在标志为1的情况下，将重定向的目标文件名保存到infile_path或outfile_path中，之后用open()函数用相应的方式打开文件(只读、覆盖写、追加写)，再用dup2()函数用打开的文件流替换相应的标准输出或输入流，即完成了重定向操作，最后还需要将标准输入输出流复原

使用示例： wc < test1.txt >> test2.txt

五、后台程序执行：

简单描述： myshell能够支持后台程序执行，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

实现原理： 在命令读取的过程中检查到‘&’时，将预先定义的表示是否为后台执行的标志置1。在标志为1的情况下，利用fork()函数创建子进程来执行命令，但在主进程中，使用waitpid()函数的WNOHANG选项，不阻塞主进程，这样就实现了命令在后台执行，主进程仍然可以显示命令提示符，进行其他操作

使用示例： sleep 5 &

六、管道操作：

简单描述： myshell能够支持管道操作，在符号‘|’左边命令的输出将成为右边命令的输入

实现原理： 在命令读取的过程中检查到‘|’时，将预先定义的表示是否为管道操作的标志置1。在标志为1的情况下，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，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。之后利用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，命令的输出将输进管道文件。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

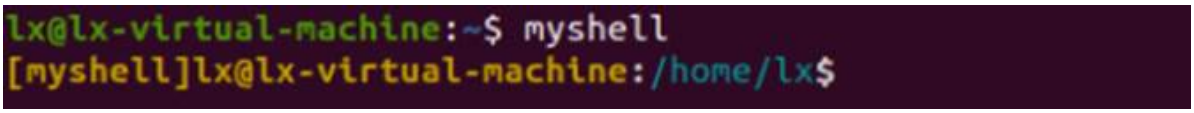
使用示例： ls | wc

七、程序环境：

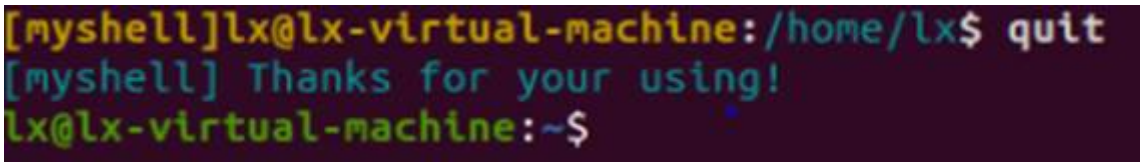
简单描述： myshell利用extern直接使用Linux系统保存的环境变量environ，要显示所有的环境变量只需要循环打印即可，修改环境变量则使用getenv()，setenv()等函数即可实现。此外还在程序开头利用shmget()函数创建了一段共享内存用于存储后台进程表的相关信息，共享内存在连接之后可以被所有进程访问到，所以使用共享内存存储后台进程表是较合理的选择

测试截图

无参数运行myshell后，进入myshell程序，输出命令行提示符等待输入，如下图所示。可以看到命令行提示符包含了当前路径。



首先测试内建命令的执行，输入quit并回车，结果如下图所示，显示了提示信息且返回了bash，符合预期。



输入clr并回车，结果如下图所示，完成了清屏操作，符合预期。

输入environ并回车，结果如下图所示(只截取部分显示结果)，显示了所有环境变量的值，可以看到myshell的环境变量包含了shell=/myshell，符合要求，再将其他环境变量的值与bash的结果进行比较，符合预期。

```
[myshell]lx@lx-virtual-machine:/home/lx$ environ
SHELL=/home/lx/myshell
SESSION_MANAGER=local/lx-virtual-machine:@/tmp/.ICE-unix/1253,unix/lx-virtual-m
achine:/tmp/.ICE-unix/1253
QT_ACCESSIBILITY=1
COLORTERM=truecolor
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/etc/xdg
XDG_MENU_PREFIX=gnome-
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
GTK_IM_MODULE=ibus
LANGUAGE=zh_CN:zh
```

```
lx@lx-virtual-machine:~$ env
SHELL=/bin/bash
SESSION_MANAGER=local/lx-virtual-machine:@/tmp/.ICE-unix/1253,unix/lx-virtual-m
achine:/tmp/.ICE-unix/1253
QT_ACCESSIBILITY=1
COLORTERM=truecolor
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/etc/xdg
XDG_MENU_PREFIX=gnome-
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
GTK_IM_MODULE=ibus
LANGUAGE=zh_CN:zh
```

输入cd并回车，结果如下图所示，显示了当前路径名，再用ls查看当前目录下的内容，再用environ命令查看当前的环境变量，可以看到PWD环境变量的值为/home/lx，和当前路径相同；之后输入cd / 并回车，结果如下图所示，切换到了根目录，再用ls查看当前目录下的内容，与之前的查看结果不同，说明切换目录成功，再用environ命令查看当前的环境变量，可以看到PWD环境变量的值为/，说明切换目录的同时PWD环境变量的值也完成了相应的更改。

```
[myshell]lx@lx-virtual-machine:/home/lx$ cd
/home/lx
[myshell]lx@lx-virtual-machine:/home/lx$ ls
11.html  视频  a.htm       fdata        largeFile    personal      test.sh
1.c      图片  b.htm       firscrip     mediumFile   professional  test.txt
1.txt    文档  cid.sh      foobar       myourt.txt   result.txt
22.html  下载  cid.txt     foobar.path  myshell      smallFile
2.c      音乐  d2          fout         myshell.c    snap
公共的   桌面  dirsync.sh  homework     output.data  temp
模板     aaa   error.log   l2.c         p01.sh       test1
[myshell]lx@lx-virtual-machine:/home/lx$ environ
```

```
PWD=/home/lx
LOGNAME=lx
XDG_SESSION_DESKTOP=ubuntu
```

```
[myshell]lx@lx-virtual-machine:/home/lx$ cd /
[myshell]lx@lx-virtual-machine:/$ ls
bin    dev   lib    libx32      mnt   root  snap      sys   usr
boot   etc   lib32  lost+found  opt   run   srv       temp  var
cdrom  home  lib64  media       proc  sbin  swapfile  tmp
[myshell]lx@lx-virtual-machine:/$ environ
```

```
PWD=/
LOGNAME=lx
XDG_SESSION_DESKTOP=ubuntu
```

cd命令后的目录名如果不存在，会输出相应的错误提示信息，结果如下图所示，符合预期。

```
[myshell]lx@lx-virtual-machine:/home/lx$ cd ??
[myshell] Error: Cannot find directory named ??
```

输入pwd并回车，结果如下图所示，显示了当前路径名，符合预期。

```
[myshell]lx@lx-virtual-machine:/home/lx$ pwd
/home/lx
```

输入exit并回车，结果如下图所示，命令提示符变为了bash的，说明成功退出myshell，符合预期。

```
[myshell]lx@lx-virtual-machine:/home/lx$ exit
lx@lx-virtual-machine:~$
```

输入time并回车，结果如下图所示，显示的当前时间与系统时间相同，符合预期。

```
[myshell]lx@lx-virtual-machine:/home/lx$ time
2020/8/9 Sun 00:29:21
```

8月9日 星期日，00：29

输入umask并回车，结果如下图所示，显示了当前的掩码，符合预期；之后输入umask 222 并回车，再输入umask查看当前掩码，可以看到被修改成了0222，符合预期。

```
[myshell]lx@lx-virtual-machine:/home/lx$ umask
0002
[myshell]lx@lx-virtual-machine:/home/lx$ umask 222
[myshell]lx@lx-virtual-machine:/home/lx$ umask
0222
```

输入dir并回车，结果如下图所示，显示了当前目录的内容；之后输入dir /home 并回车，显示了/home目录下的内容。与bash下执行相同的命令进行比较，可以说明myshell的执行结果符合预期。

```
[myshell]lx@lx-virtual-machine:/home/lx$ dir
1.txt  error.log  视频  11.html  temp  cid.sh  myshell.c  .bashrc  1.c  .ssh  f
data  test.txt  myourt.txt  桌面  下载  .cache  myshell  .gnupg  .t.swp  .mozil
la  cid.txt  d2  p01.sh  .config  2.c  smallFile  result.txt  l2.c  .profile  d
irsync.sh  公共的  foobar  音乐  homework  firscrip  .local  a.htm  professiona
l  fout  personal  b.htm  .bash_history  aaa  test1  snap  .bash_logout  largeF
ile  模板  test.sh  文档  22.html  .sudo_as_admin_successful  output.data  foob
ar.path  图片  mediumFile
[myshell]lx@lx-virtual-machine:/home/lx$ dir /home
p3.sh  lx  aaa
```

```
lx@lx-virtual-machine:~$ dir
11.html  视频  a.htm       fdata        largeFile    personal      test.sh
1.c      图片  b.htm       firscrip     mediumFile   professional  test.txt
1.txt    文档  cid.sh      foobar       myourt.txt   result.txt
22.html  下载  cid.txt     foobar.path  myshell      smallFile
2.c      音乐  d2          fout         myshell.c    snap
公共的   桌面  dirsync.sh  homework     output.data  temp
模板     aaa   error.log   l2.c         p01.sh       test1
lx@lx-virtual-machine:~$ dir /home
aaa  lx  p3.sh
```

输入echo并回车，结果如下图所示，显示了空行；之后输入echo 11 pp ccc 并回车，显示了刚刚输入的内容。结果均符合预期。

```
[myshell]lx@lx-virtual-machine:/home/lx$ echo

[myshell]lx@lx-virtual-machine:/home/lx$ echo 11 pp ccc
11 pp ccc
```

输入exec ls并回车，结果如下图所示，执行了ls命令并回到了bash，结果符合预期。

```
[myshell]lx@lx-virtual-machine:/home/lx$ exec ls
11.html  视频  a.htm       fdata        largeFile    personal      test.sh
1.c      图片  b.htm       firscrip     mediumFile   professional  test.txt
1.txt    文档  cid.sh      foobar       myourt.txt   result.txt
22.html  下载  cid.txt     foobar.path  myshell      smallFile
2.c      音乐  d2          fout         myshell.c    snap
公共的   桌面  dirsync.sh  homework     output.data  temp
模板     aaa   error.log   l2.c         p01.sh       test1
lx@lx-virtual-machine:~$
```

输入set并回车，结果如下图所示，显示了所有的环境变量，符合预期。接着输入set SHELL testshell，之后再次输入set查看环境变量，可以看到SHELL的值成功修改，符合预期。

```
[myshell]lx@lx-virtual-machine:/home/lx$ set
SHELL=/home/lx/myshell
SESSION_MANAGER=local/lx-virtual-machine:@/tmp/.ICE-unix/1253,unix/lx-virtual-m
achine:/tmp/.ICE-unix/1253
QT_ACCESSIBILITY=1
COLORTERM=truecolor
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/etc/xdg
XDG_MENU_PREFIX=gnome-
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
GTK_IM_MODULE=ibus
LANGUAGE=zh_CN:zh
```

```
[myshell]lx@lx-virtual-machine:/home/lx$ set SHELL testshell
[myshell]lx@lx-virtual-machine:/home/lx$ set
SHELL=testshell
SESSION_MANAGER=local/lx-virtual-machine:@/tmp/.ICE-unix/1253,unix/lx-virtual-m
achine:/tmp/.ICE-unix/1253
QT_ACCESSIBILITY=1
```

输入unset SHELL并回车，之后再次输入set查看环境变量，结果如下图所示，可以看到SHELL的值变为了空字符串，符合预期。

```
[myshell]lx@lx-virtual-machine:/home/lx$ unset SHELL
[myshell]lx@lx-virtual-machine:/home/lx$ set
SHELL=
SESSION_MANAGER=local/lx-virtual-machine:@/tmp/.ICE-unix/1253,unix/lx-virtual-m
achine:/tmp/.ICE-unix/1253
QT_ACCESSIBILITY=1
```

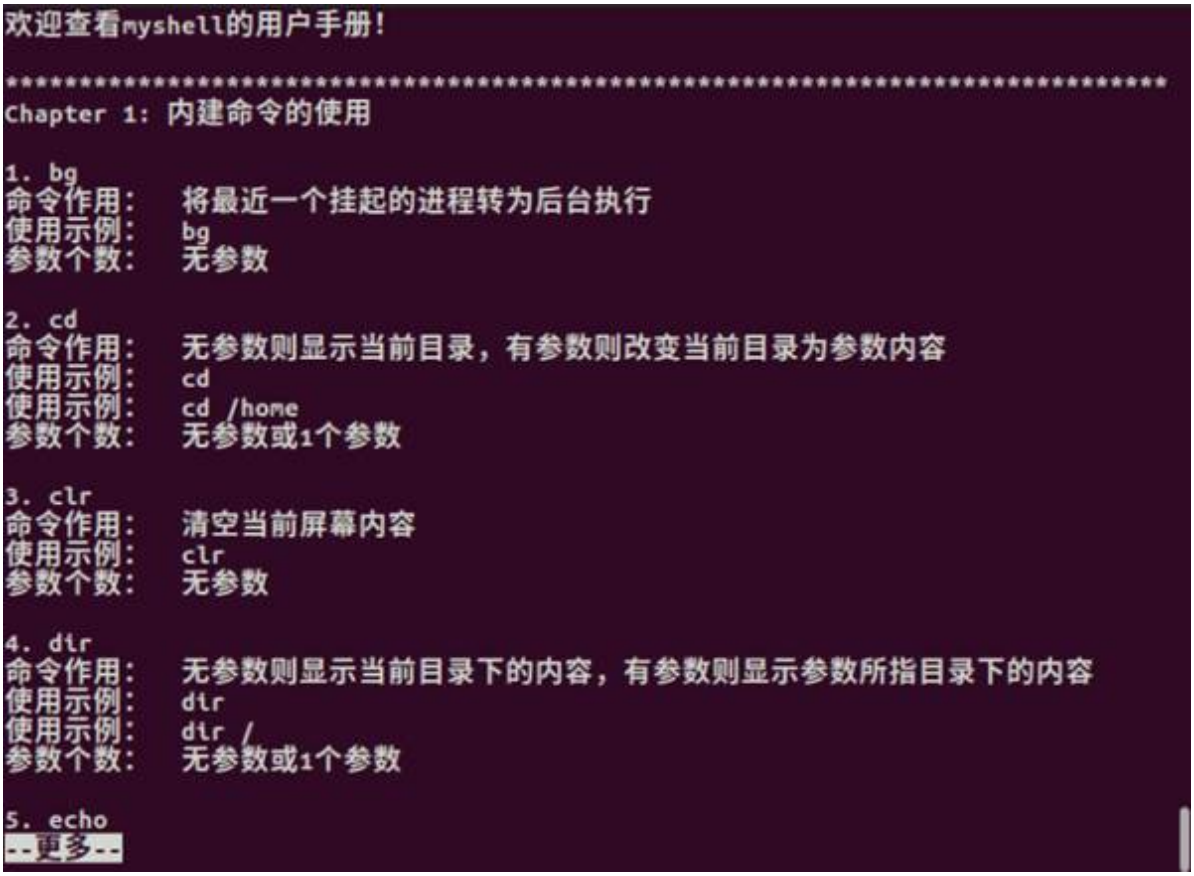
测试test命令的各个功能（判断字符串是否相等，判断两数直接的大小关系是否成立），结果如下图所示，符合预期。（-eq -ne -gt -ge -lt -le 选项分别代表相等、不相等、大于、大于等于、小于、小于等于）

```
[myshell]lx@lx-virtual-machine:/home/lx$ test aaa = bbb
False
[myshell]lx@lx-virtual-machine:/home/lx$ test aaa != bbb
True
[myshell]lx@lx-virtual-machine:/home/lx$ test 2 -eq 2
True
[myshell]lx@lx-virtual-machine:/home/lx$ test 2 -ne 2
False
[myshell]lx@lx-virtual-machine:/home/lx$ test 2 -gt 1
True
[myshell]lx@lx-virtual-machine:/home/lx$ test 2 -ge 2
True
[myshell]lx@lx-virtual-machine:/home/lx$ test 2 -lt 2
False
[myshell]lx@lx-virtual-machine:/home/lx$ test 2 -le 2
True
```

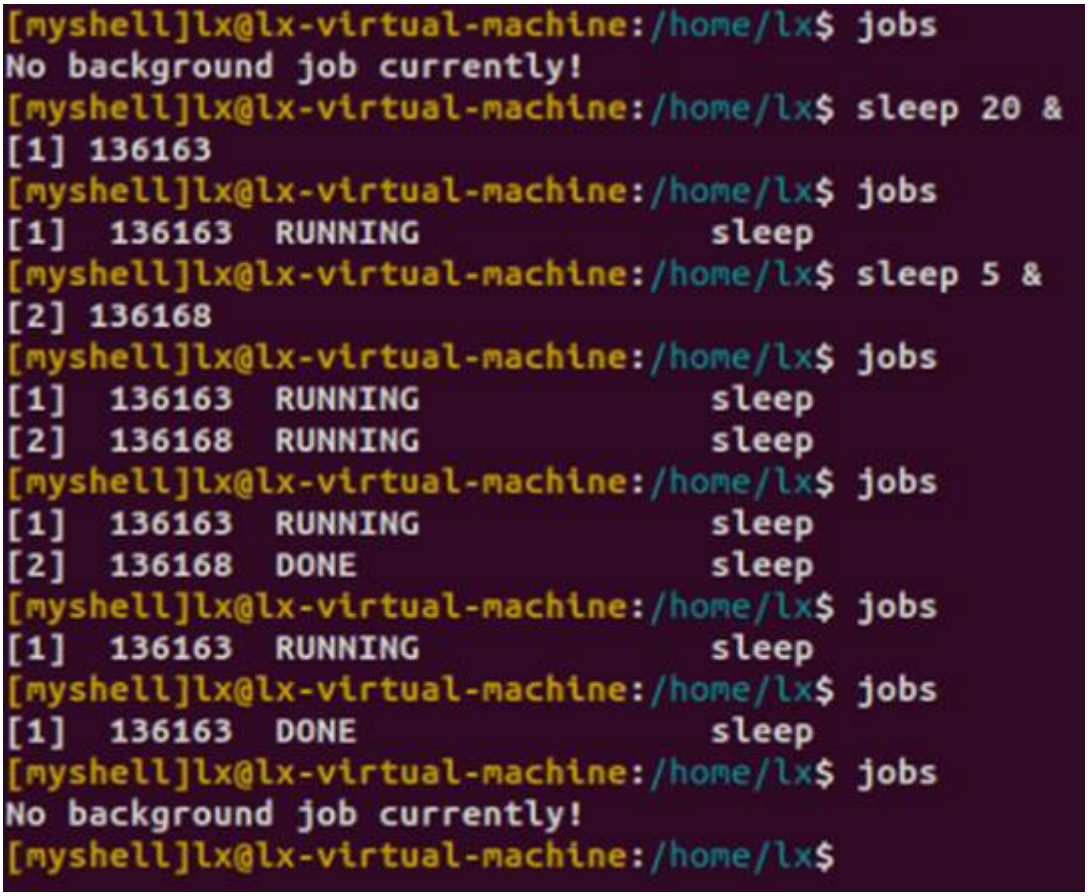
输入shift并回车，之后输入aaa bbb ccc，结果如下图所示，可以看到刚刚的输入左移一位后被输出。输入shift 2并回车，之后输入aaa bbb ccc，结果如下图所示，可以看到刚刚的输入左移两位后被输出。结果均符合预期

```
[myshell]lx@lx-virtual-machine:/home/lx$ shift
aaa bbb ccc
bbb ccc
[myshell]lx@lx-virtual-machine:/home/lx$ shift 2
aaa bbb ccc
ccc
```

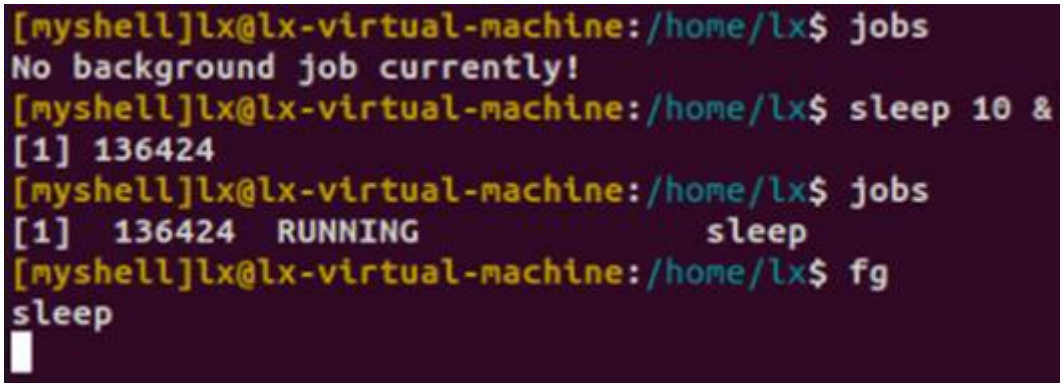
输入help并回车，结果如下图所示，可以看到显示了myshell的用户手册，从左下角的“更多”也可以看出成功使用了more命令对其进行了过滤。

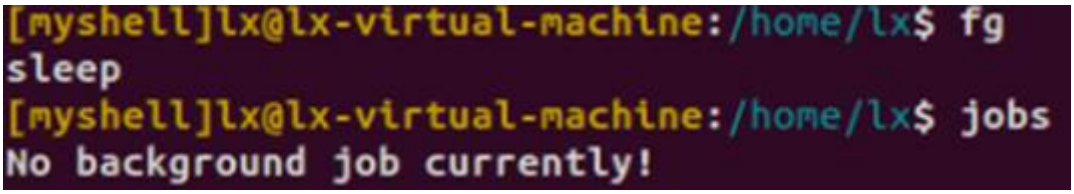


测试jobs命令的执行，测试过程与结果如下图所示，可以看到起初没有后台命令，执行了sleep 20 &后，再利用jobs查看，显示了一条后台命令的信息；继续执行sleep 5 &后，再利用jobs查看，显示了两条后台命令的信息。等待sleep 5 &执行完用jobs查看，可以看到其执行状态变为了DONE，再次输入jobs查看时就只剩下一条记录了。等待sleep 20 &也执行完后利用jobs查看，可以看到其执行状态变为了DONE，再次输入jobs查看则显示没有后台进程了。测试结果与预期相符。

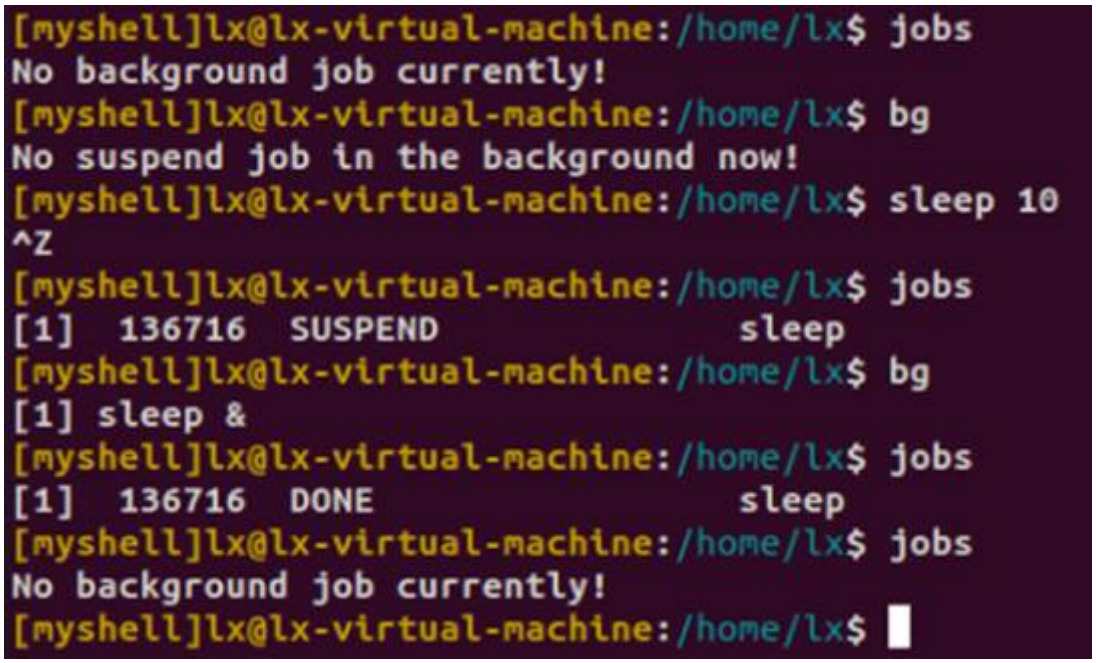


测试fg命令的执行，首先利用jobs命令查看，可以看到暂时没有后台进程；之后执行sleep 10 &，再利用jobs命令查看，可以看到sleep命令再后台运行中；再输入fg命令将后台命令转到前台，需要等待sleep 10 &执行完才能进行接下来的操作。等待其执行完后，命令提示符重新显示，输入jobs命令查看，可以看到已经没有后台进程了。综上可以看出fg命令的执行结果符合预期。

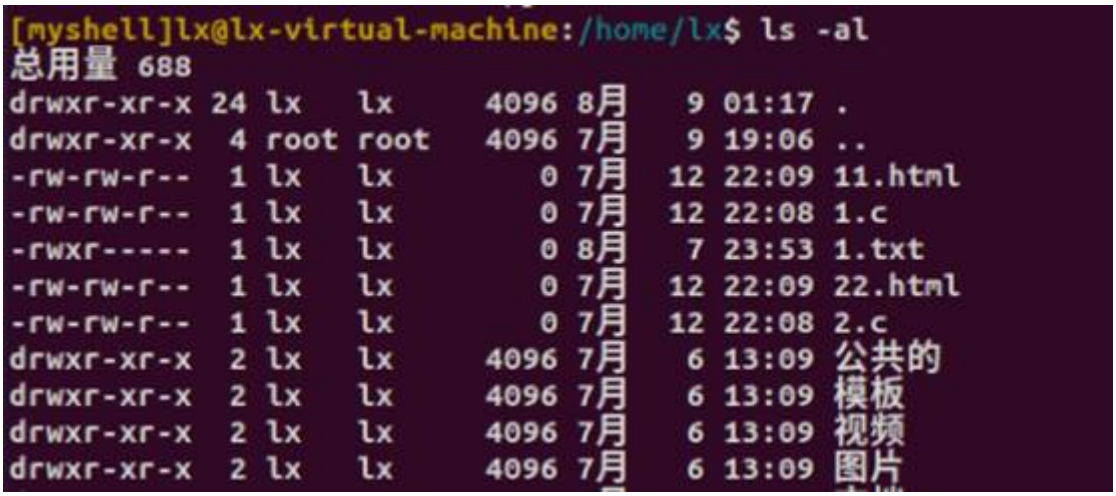


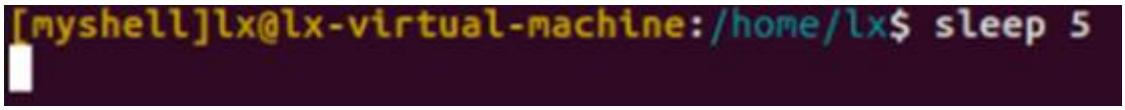


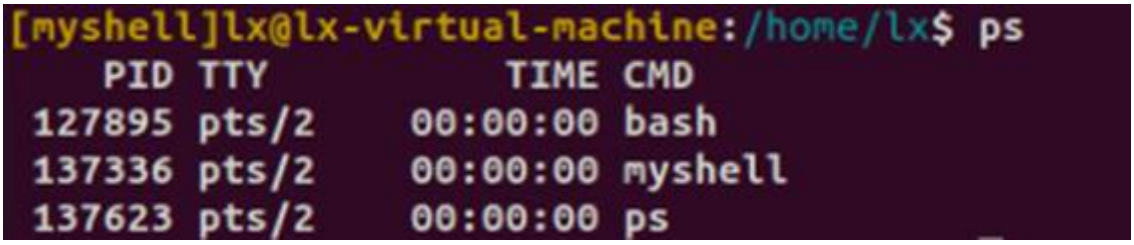
测试bg命令的执行，首先利用jobs命令查看，可以看到暂时没有后台进程；然后直接输入bg命令，提示暂时没有挂起的后台进程。之后执行sleep 10，再在执行过程中按下ctrl-z，这时可以看到命令提示符立即出现，说明回到了主进程。利用jobs命令查看，可以看到sleep命令在后台中，且状态为SUSPEND（停止中）；再输入bg命令将挂起的后台命令转为在后台继续执行，可以看到显示了[1] sleep &，说明该任务已转为后台执行，在其执行结束后使用jobs命令查看，可以看到该进程已经执行完毕，状态为DONE（已完成），之后再利用jobs命令查看，则显示当前没有后台进程。综上可以看出bg命令的执行结果符合预期。

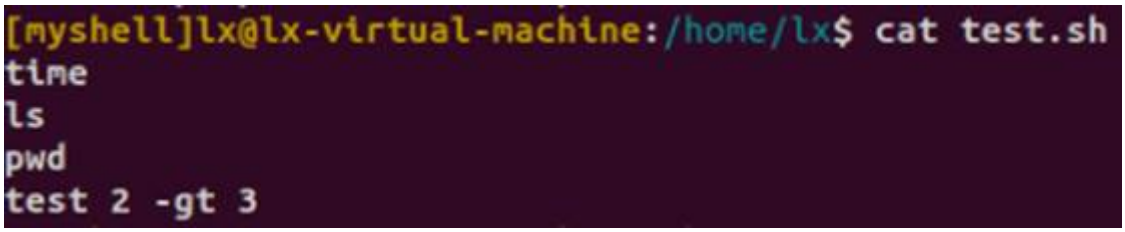


测试外部命令的执行，测试内容与结果如下图所示，可以看到myshell能够正常查找并执行外部命令。

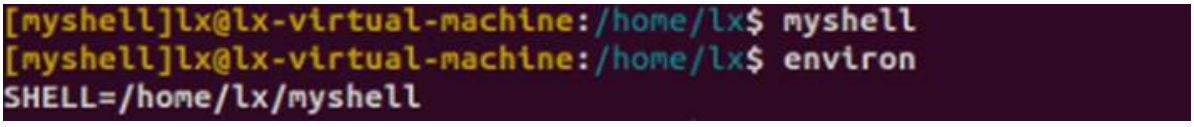








在myshell中执行environ，可以看到最后一个环境变量是OLDPWD。之后在myshell中再次执行myshell，之后执行environ，可以看到最后一个环境变量变为了PARENT=/home/lx/myshell，实现了题目要求中子进程执行的环境变量包含以下条目：parent=/myshell



测试利用myshell读取脚本文件并执行，首先在bash中查看test.sh脚本文件的内容，之后输入myshell test.sh，通过myshell执行脚本文件test.sh，执行的结果如下图所示。可以看到文件的执行结果与脚本文件的内容相吻合。

```
lx@lx-virtual-machine:~$ cat test.sh
time
ls
pwd
test 2 -gt 3
lx@lx-virtual-machine:~$ myshell test.sh
2020/8/9 Sun 02:00:44
11.html  视频  a.htm       fdata        largeFile    personal      test.sh
1.c      图片  b.htm       firscrip     mediumFile   professional  test.txt
1.txt    文档  cid.sh      foobar       myourt.txt   result.txt
22.html  下载  cid.txt     foobar.path  myshell      smallFile
2.c      音乐  d2          fout         myshell.c    snap
公共的   桌面  dirsync.sh  homework     output.data  temp
模板     aaa   error.log   l2.c         p01.sh       test1
/home/lx
False
```

测试重定向操作的执行，测试过程与结构如下图所示。使用wc命令进行测试，标准输入重定向到test1.txt中，标准输出重定向到test2.txt中，比较myshell和bash的运行结果，可以看到结果完全一致，说明重定向操作成功。

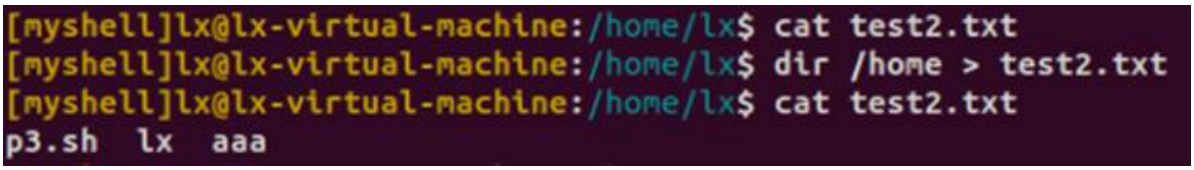
```
[myshell]lx@lx-virtual-machine:/home/lx$ cat test1.txt
This is a test!
[myshell]lx@lx-virtual-machine:/home/lx$ cat test2.txt
xxxxxxxx
[myshell]lx@lx-virtual-machine:/home/lx$ wc -m < test1.txt > test2.txt
[myshell]lx@lx-virtual-machine:/home/lx$ cat test2.txt
16
```

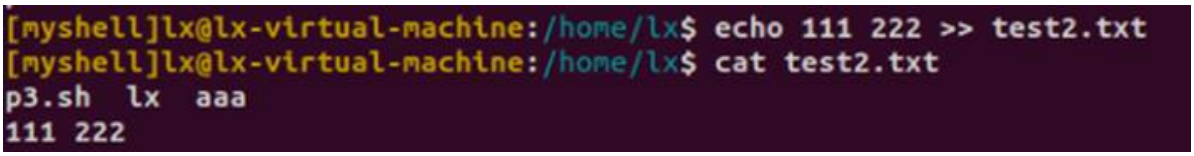
```
lx@lx-virtual-machine:~$ gedit test2.txt
lx@lx-virtual-machine:~$ cat test1.txt
This is a test!
lx@lx-virtual-machine:~$ cat test2.txt
xxxxxxxx
lx@lx-virtual-machine:~$ wc -m < test1.txt > test2.txt
lx@lx-virtual-machine:~$ cat test2.txt
16
```

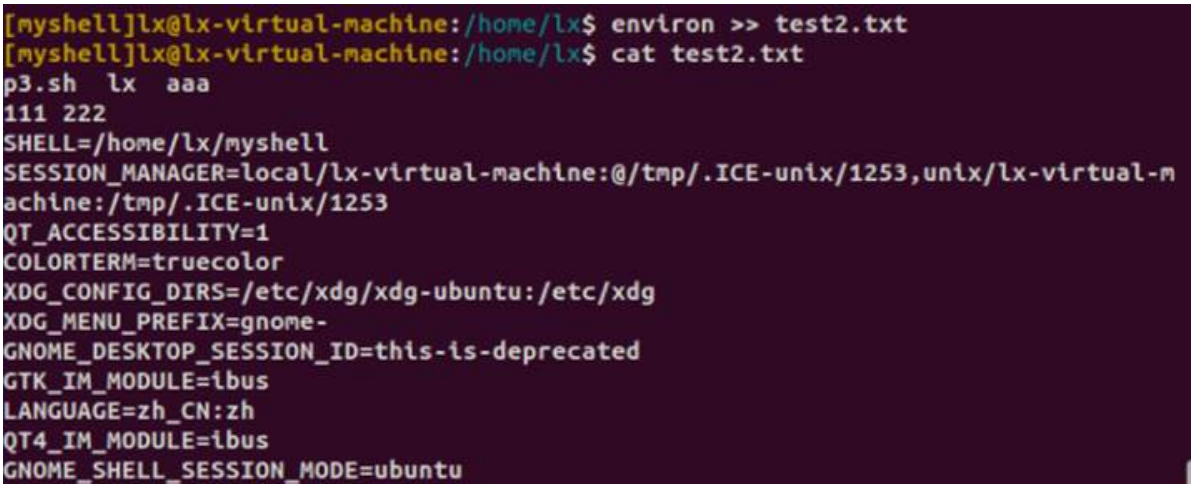
```
[myshell]lx@lx-virtual-machine:/home/lx$ cat test1.txt
This is a test!
[myshell]lx@lx-virtual-machine:/home/lx$ cat test2.txt
16
[myshell]lx@lx-virtual-machine:/home/lx$ wc < test1.txt >> test2.txt
[myshell]lx@lx-virtual-machine:/home/lx$ cat test2.txt
16
 1  4 16
```

```
lx@lx-virtual-machine:~$ cat test1.txt
This is a test!
lx@lx-virtual-machine:~$ cat test2.txt
16
lx@lx-virtual-machine:~$ wc < test1.txt >> test2.txt
lx@lx-virtual-machine:~$ cat test2.txt
16
 1  4 16
```

另外，myshell的输出重定向也支持所有的内部命令，这里以dir，environ，echo，help为例。（environ命令和help命令的结果只截取了部分内容展示）







测试后台程序执行，在myshell中输入sleep 10 &后，输出了该任务的序号以及pid，并立刻返回了命令行提示符，主程序可以进行其他操作而不被阻塞。多次输入jobs查看后台任务，前面几次显示后台任务sleep正在执行中，在其执行完后输入jobs查看则显示后台任务sleep已经执行完毕。可以说明后台程序执行功能符合预期。

```
[myshell]lx@lx-virtual-machine:/home/lx$ sleep 10 &
[1] 139479
[myshell]lx@lx-virtual-machine:/home/lx$ jobs
[1]  139479  RUNNING                 sleep
[myshell]lx@lx-virtual-machine:/home/lx$ jobs
[1]  139479  RUNNING                 sleep
[myshell]lx@lx-virtual-machine:/home/lx$ jobs
[1]  139479  DONE                    sleep
[myshell]lx@lx-virtual-machine:/home/lx$ jobs
No background job currently!
```

测试管道操作，测试过程和结果如下图所示。可以看到，myshell执行管道操作ls -l | wc 和 ls | grep 1 和bash执行这两个操作的结果是一样的，可以说明myshell的管道操作可以正常执行。

```
[myshell]lx@lx-virtual-machine:/home/lx$ ls -l | wc
     47     416    2262
[myshell]lx@lx-virtual-machine:/home/lx$ ls | grep 1
11.html
1.c
1.txt
p01.sh
test1
test1.txt
```

```
lx@lx-virtual-machine:~$ ls -l | wc
     47     416    2262
lx@lx-virtual-machine:~$ ls | grep 1
11.html
1.c
1.txt
p01.sh
test1
test1.txt
```