操作系统整合性shell用户接口和历史纪录实践

201616070320 | **物联网1603** | 郭治洪 | 指导老师： 廖庆荣
Microsoft Office Visio 提供流程图支持

摘要:

shell在UNIX或Linux操作系统中是一种很重要的使用者接口，可以透过命令行的方式输入指令。这个项目希望可以实现shell的操作方式，并且可以将所有的执行记录下来，保存程历史纪录，这跟现行所有操作系统保存系统的log档案是一样的道理。在这个实践过程，能熟悉Linux的指令操作，同时也能以指令操作了解进程的产生、进程的管理，以及系统的调用等功能，充分了解操作系统的指令运行。

一、设计目的

1. 能孰悉Linux的shell指令
2. 能实践整合性的操作系统shell用户接口指令设计
3. 能实现操作系统的历史纪录
4. 能了解和实现进程如何产生以及运作
5. 能实现进程产生
6. 能实现操作系统功能调用

二、设计要求与内容

shell在UNIX或Linux操作系统中是一种很重要的使用者接口，可以透过命令行的方式输入指令。这个项目希望可以实现shell的操作方式，使用者可以在命令行的后方（>）输入指令，也可以加上（&）符号在背景执行，表示父进程（parent process）和子进程（child process）并行（concurrently）执行。例如编程结果可以执行如下：

```
osh>cat simple-shell.c
```

或是背景执行

```
osh>cat simple-shell.c &
```

例如下列五个指令也能在 `osh>` 底下运行：

exit：结束程序执行
history：显示历史纪录
ls：显示目录
mkdir：创建目录
rmdir：删除目录

并且可以将所有的执行记录下来，保存程历史纪录。这跟现行所有操作系统保存系统的log档案是一样的道理。

当完成这个项目之后，运行程序可以显示提示字符 `osh>` ，接着可以下达命令，例如下列所示history、!!、!N（N为数字，代表第几个曾经执行过的指令）等。

1. history：显示最近的十笔历史纪录，如果历史纪录超过十笔，则会显示最近的十笔，但是号码仍会显示出来。例如有三十笔纪录，会显示第21-30笔。如果没有历史纪录，则会显示没有历史纪录的讯息。
2. !!：執行最近運行過的指令，如果没有历史執行纪录，则会显示没有指令可以執行的讯息。
3. !3：執行第三笔历史纪录，如果历史纪录没有第三笔，则会显示错误讯息没有这一笔历史纪录可供再執行一次。
4. 程序中使用 `fork()` 可以来启动进程。
5. 如果下列15项在osh中运行不出来，则我们必须重新撰写shell指令，使其正确运行。
6. 这些shell指令主要是在UNIX或Linux 操作系统上运行，如果你使用 Windows的环境运行，除非Windows操作系统完整支持，否则可能会发生程序出错或是shell指令无法运行。
   以下这15项请各位同学填上指令功能说明请直接输入在程序中的批注说明。

这些 `man + command` 都可以在本地或者网上查到

```
    1.ps: report a snapshot of the current processes.
        报告当前执行程序的快照（状态）
    2.ps -ael 此 ps 指令有以下的参数:
          -a  Select all processes except both session leaders (see
              getsid(2)) and processes not associated with a terminal.
              显示现在终端下除了领导进程（session leaders）的所有进程
              领导进程可以在这里查到：（1）
https://unix.stackexchange.com/questions/18166/what-are-session-leaders-in-ps
                                        （2）
https://zh.wikipedia.org/wiki/%E8%A1%8C%E7%A8%8B%E7%BE%A4%E7%B5%84
                                        （3）https://en.wikipedia.org/wiki/Process_group
（英文版要比中文版全）
                wiki解释这些这些进程组跟信号传递有关，是为了方便系统对进程的管理而设立的，具体的我表示能力不足，时间不足，没法深入研究。
           -e Select all processes.  Identical to -A.
              选择所有的进程，与-A相同
           -l Long format.  The -y option is often useful with this.
              长格式，以详情信息显示
```

3.ps aux

在 man ps 中说:

```
 Note that "ps -aux" is distinct from "ps aux".  The POSIX and UNIX
    standards require that "ps -aux" print all processes owned by a user
    named "x", as well as printing all processes that would be selected
    by the -a option.  If the user named "x" does not exist, this ps may
    interpret the command as "ps aux" instead and print a warning.  This
    behavior is intended to aid in transitioning old scripts and habits.
    It is fragile, subject to change, and thus should not be relied upon.
```

请注意，“ps -aux”不同于“ps aux”。

POSIX和UNIX标准要求“ps -aux”打印用户拥有的所有进程命名为“x”，并打印所有将被选中的流程由-a选项。

如果名为“x”的用户不存在，则此ps可能请将该命令解释为“ps aux”，然后打印警告。

这个行为旨在帮助转换旧的脚本和习惯。它很脆弱，可能会发生变化，因此不应该依赖。

```
Commands options such as ps -aux are not recommended as it is a
    confusion of two different standards.  According to the POSIX and
    UNIX standards, the above command asks to display all processes with
    a TTY (generally the commands users are running) plus all processes
    owned by a user named "x".  If that user doesn't exist, then ps will
    assume you really meant "ps aux".
```

命令选项如ps -aux不推荐，因为它是一个两种不同标准的混淆。

根据POSIX和UNIX标准，上述命令要求显示所有进程TTY（通常是用户正在运行的命令）以及所有进程由名为“x”的用户拥有。

如果该用户不存在，那么ps会假设你的意思是“ps aux”。

```
 a = show processes for all users
 u = display the process's user/owner
 x = also show processes not attached to a terminal
 a =显示所有用户的进程
 u =显示进程的用户/所有者
 x =还显示未连接到终端的进程
 ## 1.2.3参考 (1) http://man7.org/linux/man-pages/man1/ps.1.html
              (2) http://man.linuxde.net/ps
    3.参考 https://unix.stackexchange.com/questions/106847/what-does-aux-mean-in-ps-aux
```

4.ls - list directory contents

列出列出目录内容

-l use a long listing format

使用长列表格式

这是linux相当常用一个指令

```
# 参考 (1) http://man7.org/linux/man-pages/man1/ls.1.html
       (2) https://zh.wikipedia.org/wiki/Ls
       (3) https://en.wikipedia.org/wiki/Ls
       (4) http://man.linuxde.net/ls
```

5.ls -l | more

这个指令是不能在这个shell执行的，因为这个简单shell脚本没有解析管道（pipe）的功能

但是可以在linux下的shell执行

其实我可以了解下的，但是时间不允许，而且现在实力还不足，因此先放弃跳过吧

```
# 管道的我参考了: (1) https://ryanstutorials.net/linuxtutorial/piping.php
                  (2) https://swcarpentry.github.io/shell-novice/04-pipefilter/
                  (3) https://en.wikipedia.org/wiki/Pipeline_(Unix)
```

(4) https://zh.wikipedia.org/wiki/%E7%AE%A1%E9%81%93_(Unix)

...Google一下有很多，我就不贴了

管道的意思就是把前一个命令结果作为参数给接下来命令执行。

一般适合进程间的通信，另外进程间通信有很多方式，如socket等等

这个命令是把 当前目录以长格式列出来给more命令

more - file perusal filter for crt viewing

用于crt查看的文件读取过滤器

more：http://man7.org/linux/man-pages/man1/more.1.html

6. top - display Linux processes

   显示linux的进程

   # 参考 http://man7.org/linux/man-pages/man1/top.1.html

7. cal - display a calendar

   显示日历

   # 参考：http://man7.org/linux/man-pages/man1/cal.1.html

8. who - show who is logged on

   显示谁在登陆

   # 参考：https://linux.die.net/man/1/who

9. date - print or set the system date and time

   显示或者设置系统时间

   # 参考：https://linux.die.net/man/1/date

10. pwd - print name of current/working directory

    打印当前工作目录名称

    # 参考：https://linux.die.net/man/1/pwd

11. mv - move (rename) files

    移动（重命名）文件

    两个还能同时干

    # 参考：https://linux.die.net/man/1/mv

12. cp - copy files and directories

    复制目录和文件

    # 参考：http://man7.org/linux/man-pages/man1/cp.1.html

13. file - determine file type

    确定文件类型

    # 参考：https://linux.die.net/man/1/file

14. cat - concatenate files and print on the standard output

    和文件进行交互，和标准输出进行输出

    # 参考：http://man7.org/linux/man-pages/man1/cat.1.html

    标准输出可以参考上面的管道里面的输出重定向等概念

15. rm - remove files or directories

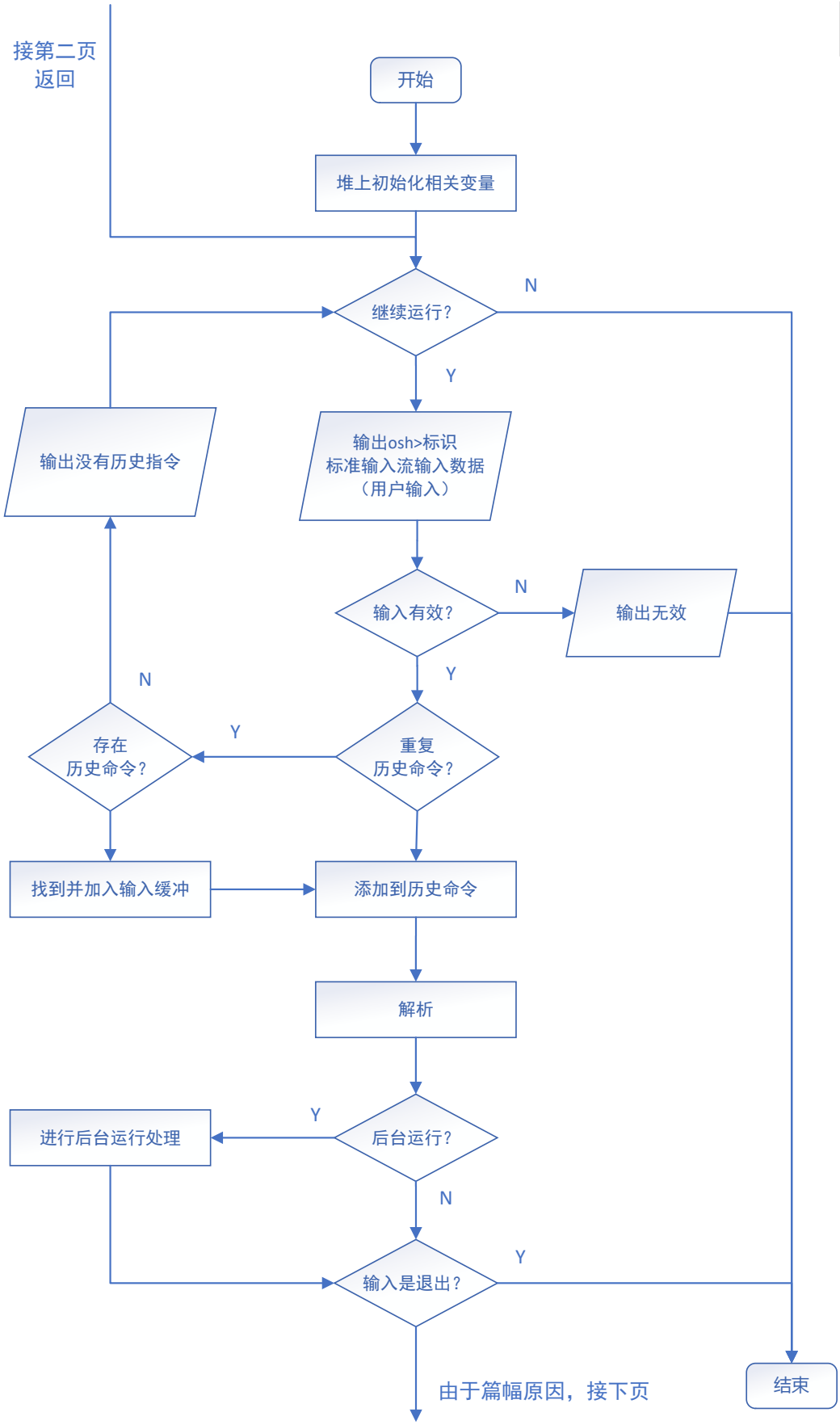
    删除文件和目录

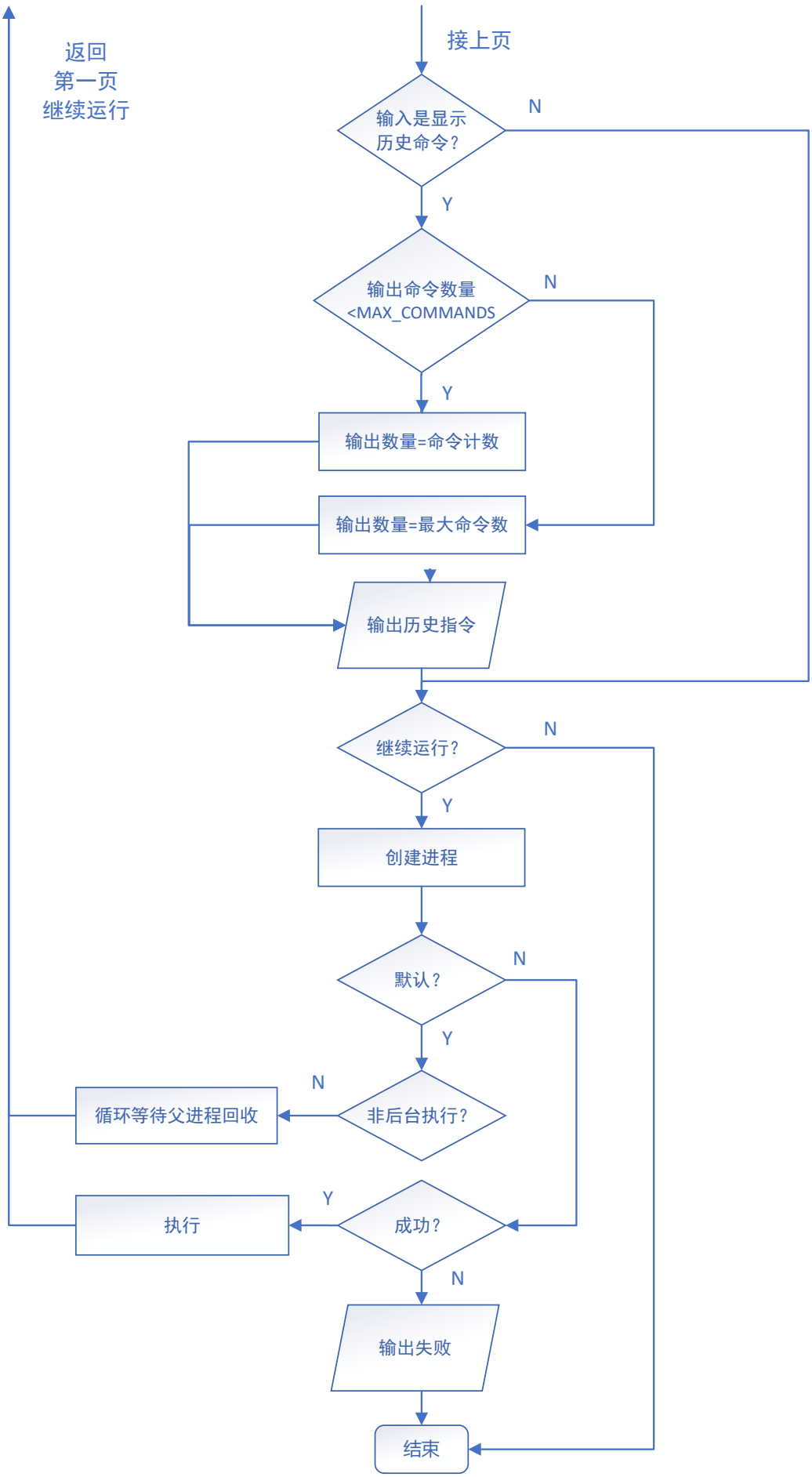
    # 参考：http://man7.org/linux/man-pages/man1/rm.1.html

三、设计原理

通过解析用户输入的命令，然后通过系统调用 `fork()` 和 `execvp()` 进行进程的创建和执行，实现一个简要的shell脚本，除此之外用C的字符数组储存历史数组和显示，以及做了一些容错处理。更多的信息详见我详细的注释。

四、流程圖


接第二页
返回
开始
堆上初始化相关变量
继续运行?
N
Y
输出没有历史指令
输出osh>标识
标准输入流输入数据
（用户输入）
输入有效?
N
输出无效
Y
N
存在
历史命令?
Y
重复
历史命令?
找到并加入输入缓冲
添加到历史命令
解析
进行后台运行处理
Y
后台运行?
N
输入是退出?
Y
由于篇幅原因，接下页
结束


返回
第一页
继续运行
接上页
输入是显示
历史命令?
N
Y
输出命令数量
<MAX_COMMANDS
N
Y
输出数量=命令计数
输出数量=最大命令数
输出历史指令
继续运行?
N
Y
创建进程
默认?
N
Y
非后台执行?
N
循环等待父进程回收
执行
成功?
Y
N
输出失败
结束

五、测试结果與說明

```
[root@localhost ~]# ./shell
osh>ps
   PID TTY          TIME CMD
  2487 pts/0    00:00:00 bash
  2515 pts/0    00:00:00 shell
  2516 pts/0    00:00:00 ps
osh>ps -ael
F S   UID     PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S     0       1       0  2  80   0 - 31339 ep_pol ?        00:00:01 systemd
1 S     0       2       0  0  80   0 -     0 kthrea ?        00:00:00 kthreadd
1 S     0       3       2  0  80   0 -     0 smpboo ?        00:00:00 ksoftirqd/0
1 S     0       4       2  0  80   0 -     0 worker ?        00:00:00 kworker/0:0
1 S     0       5       2  0  60 -20 -     0 worker ?        00:00:00 kworker/0:0H
1 S     0       6       2  0  80   0 -     0 worker ?        00:00:00 kworker/u256:0
```

- 测试 `ps` 指令，报告当前执行程序的快照（状态）
- 测试 `ps -ael` 指令，以长格式显示所有进程

```
osh>ps uax
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root          1  1.0  0.2 125356  3840 ?        Ss   05:26   0:01 /usr/lib/systemd/systemd --switched-root --system --deserialize 22
root          2  0.0  0.0      0     0 ?        S    05:26   0:00 [kthreadd]
root          3  0.0  0.0      0     0 ?        S    05:26   0:00 [ksoftirqd/0]
root          4  0.0  0.0      0     0 ?        S    05:26   0:00 [kworker/0:0]
root          5  0.0  0.0      0     0 ?        S<   05:26   0:00 [kworker/0:0H]
root          6  0.0  0.0      0     0 ?        S    05:26   0:00 [kworker/u256:0]
root          7  0.0  0.0      0     0 ?        S    05:26   0:00 [migration/0]
root          8  0.0  0.0      0     0 ?        S    05:26   0:00 [rcu_bh]
root          9  0.2  0.0      0     0 ?        R    05:26   0:00 [rcu_sched]
root         10  0.0  0.0      0     0 ?        S<   05:26   0:00 [lru-add-drain]
root         11  0.0  0.0      0     0 ?        S    05:26   0:00 [watchdog/0]
root         13  0.0  0.0      0     0 ?        S    05:26   0:00 [kdevtmpfs]
root         14  0.0  0.0      0     0 ?        S<   05:26   0:00 [netns]
```

- 测试 `ps aux` 指令，显示所有用户进程并显示标记及显示未连接终端的进程

```
osh>ls -l
total 311712
-rw-------.  1 root root      1434 Jun 30  2017 anaconda-ks.cfg
-rwxr-xr-x   1 root root      6019 Jan 18 20:28 backup.sh
drwxr-xr-x   5 root root       119 Jan 21 23:26 bash-ini-parser
-rw-r--r--   1 root root      1657 Mar 24  2016 CentOS7-Base-zzu.repo
drwxr-xr-x   2 root root         6 Jan 16 19:20 dev
drwxr-xr-x   4 root root        43 Jan 26 13:54 gai
drwxr-xr-x   6 root root       256 Jan 25 10:29 git-ftp
-rwxr-xr-x   1 root root      8728 Mar  9 19:38 hello
```

- 测试 `ls -l` 指令，以长格式输出当前目录下文件

```
osh>ls -l|more
ls: invalid option -- '|'
Try 'ls --help' for more information.
```

- 测试带管道的 `ls -l|more` 指令，由于这个shell没有写pipe解析，因此没法用

```
top - 05:30:04 up 4 min,  1 user,  load average: 0.05, 0.19, 0.11
Tasks: 131 total,   1 running, 130 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  1865284 total,  1134960 free,   330356 used,   399968 buff/cache
KiB Swap:  2097148 total,  2097148 free,        0 used.  1332636 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
    1 root      20   0  125356   3840   2556 S  0.0  0.2   0:01.41 systemd
    2 root      20   0       0      0      0 S  0.0  0.0   0:00.00 kthreadd
    3 root      20   0       0      0      0 S  0.0  0.0   0:00.07 ksoftirqd/0
    4 root      20   0       0      0      0 S  0.0  0.0   0:00.00 kworker/0:0
    5 root       0 -20       0      0      0 S  0.0  0.0   0:00.00 kworker/0:0H
    6 root      20   0       0      0      0 S  0.0  0.0   0:00.00 kworker/u256:0
    7 root      rt   0       0      0      0 S  0.0  0.0   0:00.00 migration/0
    8 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcu_bh
```

- 测试 `top` 指令显示进程

```
[root@localhost ~]# ./shell
osh>cal
      May 2018
Su Mo Tu We Th Fr Sa
       1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31

osh>who
root     pts/0        2018-05-24 05:26 (192.168.159.1)
osh>date
Thu May 24 05:30:24 CST 2018
osh>pwd
/root
osh>mv t test
mv: cannot stat ‘t’: No such file or directory
osh>cp t test
cp: cannot stat ‘t’: No such file or directory
osh>file shell.c
shell.c: C source, UTF-8 Unicode text, with CRLF line terminators
osh>cat shell.c
#include <stdio.h>
#include <unistd.h>
#include <errno.h>
#include <stdlib.h>
#include <sys/types.h>
#include <string.h>
#define MAX_LINE 80
```

- 测试 `cal` 输出日历指令
- 测试 `who` 显示谁在登陆，
- 测试 `date` 输出日期
- 测试 `pwd` 输出当前工作目录
- 测试 `mv` 和 `cp` 指令，由于我本地没有，当然会报错，假如成功是不回显的
- 测试 `file` 输出文件信息指令，测试 `cat` 指令查看文件内容

```
osh>rm test
rm: cannot remove ‘test’: Is a directory
```

- 测试 `rm` 指令，我本地有一个 `test` 文件夹，要想删除必须用 `-r` 参数，不加会报错

具体可以参照上面或者代码中的说明。

六、收获、体会和建议

最后感谢老师的认真教导和帮助，以及二班童沪琨同学的帮助。

可能不太好，但是我尽力完成了。

我会继续加油的。

另外 word 排版太难受了，我用了 MarkDown 排的版，长代码部分为了适应做了自动截取，但是有显示不好的地方，希望不要介意。

附录

1.源碼

```
/**
 * Solution to the shell interface program.
 *  操作系统整合性 shell 用户接口和历史纪录实践
 *  物联网1603 郭治洪 201616070320
 *  指导老师：廖庆荣
 */

 /*
    这些 man + command 都可以在本地或者网上查到

    1.ps：report a snapshot of the current processes.
        报告当前执行程序的快照（状态）
    2.ps -ael 此 ps 指令有以下的参数：
          -a  Select all processes except both session leaders (see
              getsid(2)) and processes not associated with a terminal.
              显示现在终端下除了领导进程（session leaders）的所有进程
              领导进程可以在这里查到：（1）
https://unix.stackexchange.com/questions/18166/what-are-session-leaders-in-ps
                                        （2）
https://zh.wikipedia.org/wiki/%E8%A1%8C%E7%A8%8B%E7%BE%A4%E7%B5%84
                                        （3）https://en.wikipedia.org/wiki/Process_group
 （英文版要比中文版全）
                wiki解释这些这些进程组跟信号传递有关，是为了方便系统对进程的管理而设立的，具体的我表
```

示能力不足，时间不足，没法深入研究。

-e Select all processes. Identical to -A.

选择所有的进程，与-A相同

-l Long format. The -y option is often useful with this.

长格式，以详情信息显示

3.ps aux

在 man ps 中说：

Note that "ps -aux" is distinct from "ps aux". The POSIX and UNIX standards require that "ps -aux" print all processes owned by a user named "x", as well as printing all processes that would be selected by the -a option. If the user named "x" does not exist, this ps may interpret the command as "ps aux" instead and print a warning. This behavior is intended to aid in transitioning old scripts and habits. It is fragile, subject to change, and thus should not be relied upon.

请注意，“ps -aux”不同于“ps aux”。

POSIX和UNIX标准要求“ps -aux”打印用户拥有的所有进程命名为“x”，并打印所有将被选中的流程由-a选项。

如果名为“x”的用户不存在，则此ps可能请将该命令解释为“ps aux”，然后打印警告。

这个行为旨在帮助转换旧的脚本和习惯。它很脆弱，可能会发生变化，因此不应该依赖。

Commands options such as ps -aux are not recommended as it is a confusion of two different standards. According to the POSIX and UNIX standards, the above command asks to display all processes with a TTY (generally the commands users are running) plus all processes owned by a user named "x". If that user doesn't exist, then ps will assume you really meant "ps aux".

命令选项如ps -aux不推荐，因为它是一个两种不同标准的混淆。

根据POSIX和UNIX标准，上述命令要求显示所有进程TTY（通常是用户正在运行的命令）以及所有进程由名为“x”的用户拥有。

如果该用户不存在，那么ps会假设你的意思是“ps aux”。

a = show processes for all users

u = display the process's user/owner

x = also show processes not attached to a terminal

a =显示所有用户的进程

u =显示进程的用户/所有者

x =还显示未连接到终端的进程

1.2.3参考 (1) http://man7.org/linux/man-pages/man1/ps.1.html

(2) http://man.linuxde.net/ps

3.参考 https://unix.stackexchange.com/questions/106847/what-does-aux-mean-in-ps-aux

4.ls - list directory contents

列出列出目录内容

-l use a long listing format

使用长列表格式

这是linux相当常用一个指令

参考 (1) http://man7.org/linux/man-pages/man1/ls.1.html

(2) https://zh.wikipedia.org/wiki/Ls

(3) https://en.wikipedia.org/wiki/Ls

(4) http://man.linuxde.net/ls

5.ls -l | more

这个指令是不能在这个shell执行的，因为这个简单shell脚本没有解析管道（pipe）的功能

但是可以在linux下的shell执行
其实我可以了解下的，但是时间不允许，而且现在实力还不足，因此先放弃跳过吧
管道的我参考了：（1）https://ryanstutorials.net/linuxtutorial/piping.php
（2）https://swcarpentry.github.io/shell-novice/04-pipefilter/
（3）https://en.wikipedia.org/wiki/Pipeline_(Unix)
（4）https://zh.wikipedia.org/wiki/%E7%AE%A1%E9%81%93_(Unix)
...Google一下有很多，我就不贴了
管道的意思就是把前一个命令结果作为参数给接下来命令执行。
一般适合进程间的通信，另外进程间通信有很多方式，如socket等等
这个命令是把 当前目录以长格式列出来给more命令
more - file perusal filter for crt viewing
用于crt查看的文件读取过滤器
more：http://man7.org/linux/man-pages/man1/more.1.html
6. top - display Linux processes
显示linux的进程
参考 http://man7.org/linux/man-pages/man1/top.1.html
7. cal - display a calendar
显示日历
参考：http://man7.org/linux/man-pages/man1/cal.1.html
8. who - show who is logged on
显示谁在登陆
参考：https://linux.die.net/man/1/who
9. date - print or set the system date and time
显示或者设置系统时间
参考：https://linux.die.net/man/1/date
10.pwd - print name of current/working directory
打印当前工作目录名称
参考：https://linux.die.net/man/1/pwd
11.mv - move (rename) files
移动（重命名）文件
两个还能同时干
参考：https://linux.die.net/man/1/mv
12.cp - copy files and directories
复制目录和文件
参考：http://man7.org/linux/man-pages/man1/cp.1.html
13.file - determine file type
确定文件类型
参考：https://linux.die.net/man/1/file
14.cat - concatenate files and print on the standard output
和文件进行交互，和标准输出进行输出
参考：http://man7.org/linux/man-pages/man1/cat.1.html
标准输出可以参考上面的管道里面的输出重定向等概念
15.rm - remove files or directories
删除文件和目录
参考：http://man7.org/linux/man-pages/man1/rm.1.html
END
最后感谢老师的认真教导和帮助，以及二班童沪琨同学的帮助。
可能不太好，但是我尽力完成了。
我会继续加油的。

```
 */


#include <stdio.h>
#include <unistd.h>
#include <errno.h>
#include <stdlib.h>
#include <sys/types.h>
#include <string.h>


#define MAX_LINE        80
/* 80 chars per line, per command, should be enough. */
#define MAX_COMMANDS    5
/* size of history */

char history[MAX_COMMANDS][MAX_LINE]; //记录命令的字符数组
char display_history [MAX_COMMANDS][MAX_LINE]; //要显示命令的字符数组

//注意这里命令字符数组和显示命令字符数组不太一样


int command_count = 0; //命令计数

/**
 * Add the most recent command to the history.
 */

void addtohistory(char inputBuffer[])
{
    int i = 0;

    // add the command to history
    strcpy(history[command_count % MAX_COMMANDS], inputBuffer);
    //将输入缓冲数组复制到记录命令的字符数组

    // add the display-style command to history
    while (inputBuffer[i] != '\n' && inputBuffer[i] != '\0')
    {
        //复制要显示命令的字符数组
        display_history[command_count % MAX_COMMANDS][i] = inputBuffer[i];
        i++;
    }
    display_history[command_count % MAX_COMMANDS][i] = '\0';
    //添加截止符号
    ++command_count; //命令计数+1

    return;
}
```

```
/**
 * The setup function below will not return any value, but it will just: read
 * in the next command line; separate it into distinct arguments (using blanks as
 * delimiters), and set the args array entries to point to the beginning of what
 * will become null-terminated, C-style strings.
 */

int setup(char inputBuffer[], char *args[],int *background)
{
    int length,         /* # of characters in the command line */
    //命令长度
    i,                 /* loop index for accessing inputBuffer array */
    //输入缓冲数组索引下标
    start,             /* index where beginning of next command parameter is */
    //下一个命令参数的索引下标
    ct,                 /* index of where to place the next parameter into args[] */
    //在参数数组放置参数的索引下标
    command_number;     /* index of requested command number */
    //历史命令的索引
    ct = 0;

    /* read what the user enters on the command line */
    do {
        printf("osh>");
        fflush(stdout); //使用fflush(stdout)及时把数据输出
        length = read(STDIN_FILENO,inputBuffer,MAX_LINE);
        //从标准输入流输入数据
    }
    while (inputBuffer[0] == '\n'); /* swallow newline characters */
    //以回车为结尾
    /**
     *  0 is the system predefined file descriptor for stdin (standard input),
     *  which is the user's screen in this case. inputBuffer by itself is the
     *  same as &inputBuffer[0], i.e. the starting address of where to store
     *  the command that is read, and length holds the number of characters
     *  read in. inputBuffer is not a null terminated C-string.
     */

    start = -1;
    if (length == 0)
        //按下CTRL+D
        exit(0);            /* ^d was entered, end of user command stream */

    /**
     * the <control><d> signal interrupted the read system call
     * if the process is in the read() system call, read returns -1
       * However, if this occurs, errno is set to EINTR. We can check this  value
       * and disregard the -1 value
```

```
 */
if ( (length < 0) && (errno != EINTR) )
{
    perror("error reading the command");
    //处理输入错误的情况
    //error返回错误
    exit(-1);           /* terminate with error code of -1 */
}

/**
 * Check if they are using history
 */

if (inputBuffer[0] == '!') {
    //如果输入的第一位是!
    if (command_count == 0) {
        //如果命令计数0，输出没有命令
        printf("No history\n");
        return 1;
    }
    else if (inputBuffer[1] == '!')
    {
        // restore the previous command
        // !! 还原之前的命令
        strcpy(inputBuffer,history[(command_count - 1) % MAX_COMMANDS]);
        length = strlen(inputBuffer) + 1;
        //长度等于输入缓冲数组+1
    }
    else if (isdigit(inputBuffer[1]))
    { /* retrieve the nth command */
        // !n 还原第n条命令
        command_number = atoi(&inputBuffer[1]);
        //转为整型变量
        strcpy(inputBuffer,history[command_number]);
        length = strlen(inputBuffer) + 1;
    }
}


/**
 * Add the command to the history
 */
//添加到历史命令中
addtohistory(inputBuffer);

/**
 * Parse the contents of inputBuffer
  解析输入缓冲数组的命令
 */
```

```
for (i=0;i<length;i++)
{
    //遍历每一个元素
    /* examine every character in the inputBuffer */
    /*
        command: `cp xxx yyy &`
        将上面命令解析成下面的形式:

        args[0]="cp";
        args[1]="xxx";
        args[2]="yyy";
        args[3]=NULL;
        args是char ** 数组

        start是下一个命令参数的位置
        在发现空格后我们认为前面的是一个命令参数
        然后就会直接将前面命令参数截止掉并且取地址给命令参数数组
        使输入缓冲数组每次都是0下标开始，start结束，start起始位置因此是-1
        ct是参数数组的索引下标每次完成后+1
        由于后台执行会忽略输入的 & 因此这个不是参数
        所以会将它的这位变成char *的NULL指针
        由于execvp要求参数数组最后必须是char *的NULL指针
        因此需要特别注意
    */
    switch (inputBuffer[i])
    {
        //处理空格
        case ' ':
        case '\t' :               /* argument separators */
            if(start != -1)
            {

                  args[ct] = &inputBuffer[start];    /* set up pointer */
                ct++;
            }
            inputBuffer[i] = '\0'; /* add a null char; make a C string */
            start = -1;
            break;
        //处理回车
        case '\n':                 /* should be the final char examined */
            if (start != -1)
            {
                args[ct] = &inputBuffer[start];
                ct++;
            }
            inputBuffer[i] = '\0';
            //当前位截止
            args[ct] = NULL; /* no more arguments to this command */
```

```
                //要求参数数组最后必须是char *的NULL指针
                break;
            //处理其他字符
            default :             /* some other character */
                if (start == -1)
                    start = i;
                  if (inputBuffer[i] == '&')
                {
                    *background  = 1; //在后台运行的情况
                       inputBuffer[i-1] = '\0';
                    //在前一位就截止
                    //忽略这个 & 后台运行标志
                }
        } /* end of switch */
    }    /* end of for */

    /**
     * If we get &, don't enter it in the args array
     */

    //要求参数数组最后必须是char *的NULL指针

    if (*background)
        args[--ct] = NULL;

    args[ct] = NULL; /* just in case the input line was > 80 */

    return 1;

} /* end of setup routine */


int main(void)
{
    char inputBuffer[MAX_LINE];     /* buffer to hold the command entered */
    //输入缓冲数组
    int background;                 /* equals 1 if a command is followed by '&' */
    /*是否后台运行变量
      Linux 后台运行使用&
      如 command & */
    char *args[MAX_LINE/2 + 1];    /* command line (of 80) has max of 40 arguments */
    /*命令最多有一半的参数
      这是参数字符指针数组
    */
    pid_t child;                    /* process id of the child process */
    //child 进程的进程id
    int status;                   /* result from execvp system call*/
    //系统execvp()函数返回状态
    int shouldrun = 1;
```

```
    int i, upper;

    while (shouldrun)
    {                    /* Program terminates normally inside setup */
        background = 0;

        shouldrun = setup(inputBuffer,args,&background);       /* get next command */

        if (strncmp(inputBuffer, "exit", 4) == 0)
            return 0; //输入退出
        else if (strncmp(inputBuffer,"history", 7) == 0)
        {
            //输入history输出显示数组里的命令
            //upper为输出历史指令数量
            if (command_count < MAX_COMMANDS)
             //没有达到最大命令数量
                upper = command_count;
            else
                upper = MAX_COMMANDS;

            for (i = 0; i < upper; i++)
            {//输出历史指令
                printf("%d \t %s\n", i, display_history[i]);
            }

            continue;
        }


        if (shouldrun) {
            child = fork();          /* creates a duplicate process! */
            /*创建一个新的进程，根据资料linux每次创建进程都是fork一个父进程变成child进程，Linux的第一个进程是init
                因此init也是Linux的根进程
                参考:
                    1.http://www.cnblogs.com/vamei/archive/2012/09/20/2694466.html?from=timeline&isappinstalled=1
                    2.http://www.cnblogs.com/vamei/archive/2012/09/19/2692452.html?from=timeline&isappinstalled=1
                    3.https://unix.stackexchange.com/questions/18166/what-are-session-leaders-in-ps
             */
            switch (child)
            {
                case -1:
                    //创建失败的情况
                    perror("could not fork the process");
                    break; /* perror is a library routine that displays a system
```

```
                        error message, according to the value of the system
                        variable "errno" which will be set during a function
                        (like fork) that was unable to successfully
                        complete its task. */
                        /* error返回系统调用最后一个错误
                          参考资料: http://man7.org/linux/man-pages/man3/errno.3.html
                        */

                case 0: /* this is the child process */
                    //成功情况
                    status = execvp(args[0],args);
                    //execvp()会从PATH 环境变量所指的目录中查找符合参数file 的文件名，找到后便执行该文件，然后将第二个参数argv 传给该欲执行的文件。
                    /*
                        参考资料:
                            1.https://linux.die.net/man/3/execvp

2.http://www.csl.mtu.edu/cs4411.ck/www/NOTES/process/fork/exec.html
                        说明:
                            execvp的第一个参数是执行命令的文件名，而第二个参数是执行命令的参数，其中包括args[0]，也就刚这个执行命令文件名
                            且最后以char * 的NULL指针结尾
                    */
                    if (status != 0){
                        //执行失败情况
                        perror("error in execvp");
                        exit(-2); /* terminate this process with error code -2 */
                    }
                    break;

                default :  /* this is the parent */
                    if (background == 0) /* handle parent,wait for child */
                        //如果不在后台运行就回收子进程
                        while (child != wait(NULL)); //父进程回收子进程
                        //不成功就一直循环
            }
        }
    }

    return 0;
}
```

主要参考文献

1. 汤小丹、梁红兵、哲凤屏、汤子瀛（2014）。计算机操作系统 第四版，西安电子科技大学出版社。

2. 梁红兵、汤小丹、汤子瀛（2014）。计算机操作系统 第四版 学习指导与题解，西安电子科技大学出版社。

3. A. Silberschatz, G. Gagne, P. B. Galvin (2013). Operating system concepts. John Wiley & Sons.