# 操作系统实验报告

## 实验五 shell 程序

| 学号 | 姓名 |
| --- | --- |
| 10231016 | 童浩（组长） |
| 10231008 | 解佳琦 |
| 10231024 | 陈宇宁 |
| 38230213 | 邱伟豪 |

# 1 需求说明

## 1.1 基本需求

本程序提供一个命令指示符，格式为 xsh@*current_path*>（如“xsh@/home/tong/shell>”），表示等待用户的输入。用户输入后，如果命令合法，那么执行该命令并输出必要信息，然后再打印下一个命令提示符，如果命令不合法，那么会显示错误信息，并打印命令提示符。当用户没有输入时，xsh 一直处于随时等待输入状态，同时在屏幕上显示一些基本提示信息。

### 1.1.1 内部命令

本程序设计的内部命令如下：

1. exit
   结束所有的子进程并退出 shell。
2. jobs
   打印当前正在后台执行的作业和被挂起的作业信息。输出信息格式为
   INDEX PID STATE COMMAND
   jobs 自身是一条内部命令，所以不需要显示在输出上。
3. history
   列出用户最近输入过的 HISTORY_LEN 条命令，不论这个命令是否正确执行过。
4. fg %<int>
   把<int>所标识的作业放到前台运行。若这个作业原来已经挂起则让其继续运行。user-sh 应当在打印新的命令提示符之前等待前台运行的子进程结束。
5. bg %<int>
   把<int>所标识的已挂起的进程放在后台运行。

### 1.1.2 前台、后台作业切换

本程序能够执行前台和后台作业。两者的区别是：shell 在前台作业执行完之前要一直处于等待状态。而在开始执行后台作业时要立刻打印出提示符，让用户继续输入下一条命令。

执行前台作业即在提示符后输入一个可执行文件的路径（绝对路径）即可，执行后台作业则需在可执行文件路径后加上一个“&”符号。

并且可以通过处理组合键实现前/后台作业的切换：

Ctrl + Z

产生 SIGTSTP 信号，这个信号不会挂起 shell 程序，而是本 shell 程序挂起在前台运行的作业，如果没有任何前台作业，则该特殊键无效。

Ctrl + C

产生 SIGINT 信号，这个信号不终止 shell 程序，而是通过本程序发出信号杀死前台作业中的进程。如果没有任何前台作业，则该特殊键无效。

### 1.1.3 I/O 重定向

一个命令后面可能还跟有元字符“<”或“>”，它们表示执行输入或输出重定向，在重定向符号后面还跟着一个文件名。如果输出文件不存在，需要创建一个输出文件。如果输入文件不存在，则认为命令出现错误。

输入重定向符号：<

程序的输入被重定向到一个指定的文件中。

输出重定向符号：>

文件的输出被重定向到一个指定的文件中。

## 1.2 进阶需求

### 1.2.1 管道

当若干个命令被*元字符*“|”分隔开时，它们可以放在一条命令行当中，这个元字符代表管道符号。在这种情况下，user-sh 为每一个子命令都创建一个进程，并把它们的 I/O 用管道连接起来。由管道连接的多个进程多组成的作业只有当其所有的子进程都执行完毕时才算结束。

### 1.2.2 通配符

在本 shell 程序中，可以使用通配符‘*’和‘?’来匹配文件或文件夹。其中‘*’表示任意字符串（包括空串），‘?’表示任意一个字符。例如要删除以‘a’开头的所有文件，那么使用 rm a*这一命令即可。

## 1.3 自行改进

### 1.3.1 实现键盘的“↑”“↓”键切换语句功能

通过终端 I/O 编程，不显示“↑”“↓”键键盘码，通过“↑”（或“↓”）键来显示上一条（或下一条）命令，便于输入重复命令。

### 1.3.2 通过“Tab”键实现命令的补全

通过终端 I/O 编程，不显示“Tab”键键盘码。在输入了一条命令的前部之后，可以通过“Tab”键将其进行进行补全。如果以该字符串为前部的命令有不止一条，那么再按一下“Tab”可以显示出所有匹配的命令。

### 1.3.3 实现退格键功能

通过终端 I/O 编程，实现 BackSpace 的删除功能，不显示键盘码。

### 1.3.4 改进文法

改进文法提供对通配符和有管道的复杂命令的支持

### 1.3.5 后台命令执行完成给出结果

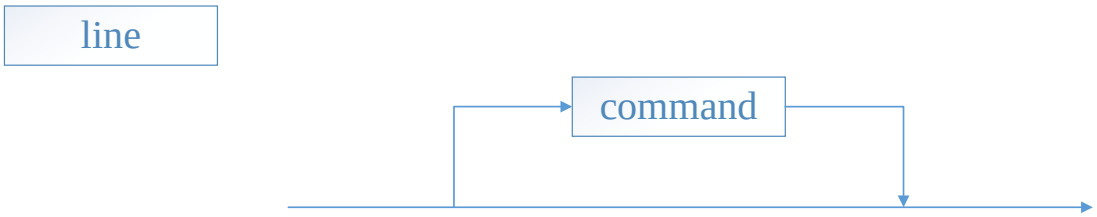
后台命令在完成后，在按下一个回车后给出命令完成的提示。

# 2 设计说明

## 2.1 结构设计

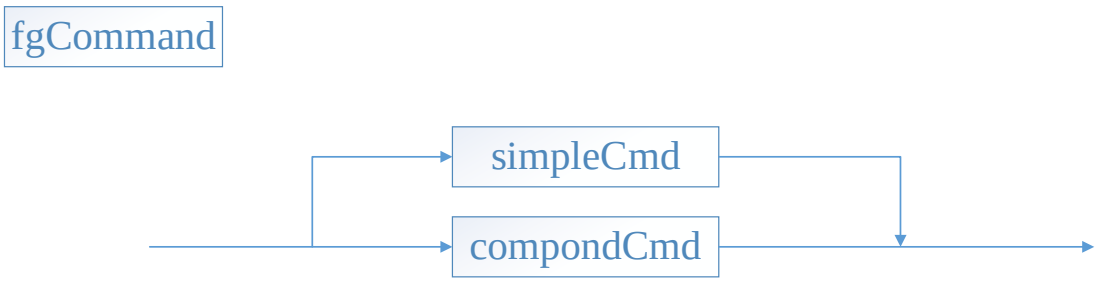
### 2.1.1 语法状态图

1. 命令行 line




2. 命令 command

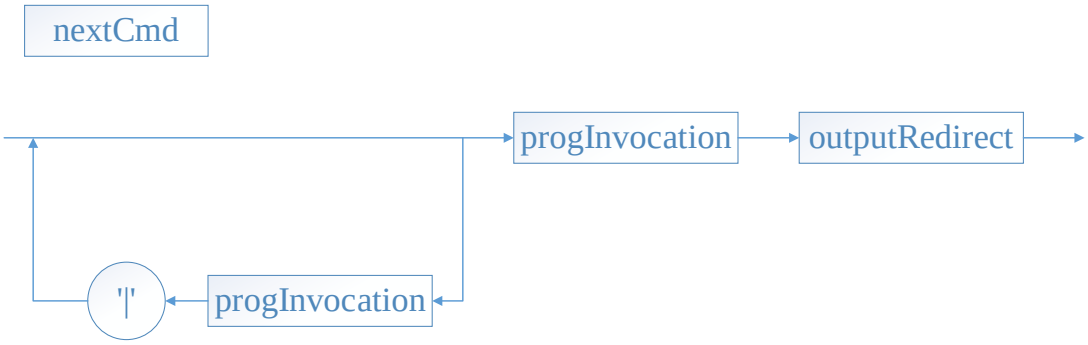

3. 前台命令 fgCommand



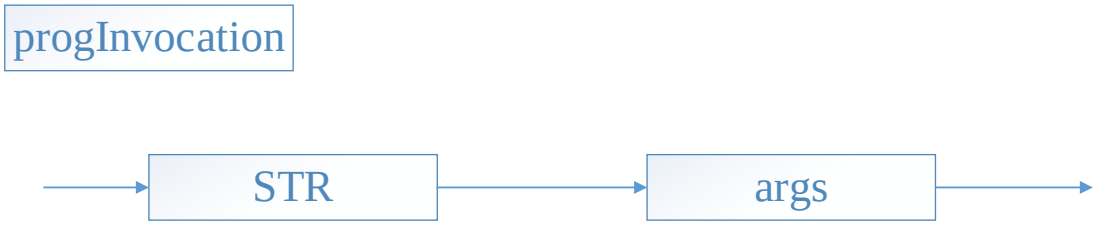

4. 简单命令 simpleCmd


5. 复合命令 compondCmd

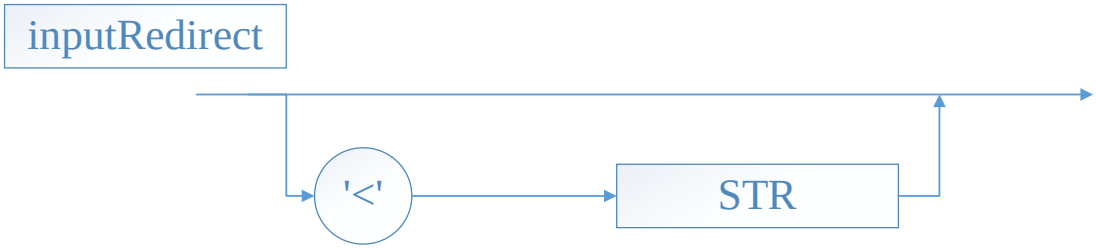

6. 下一条命令 nextCmd



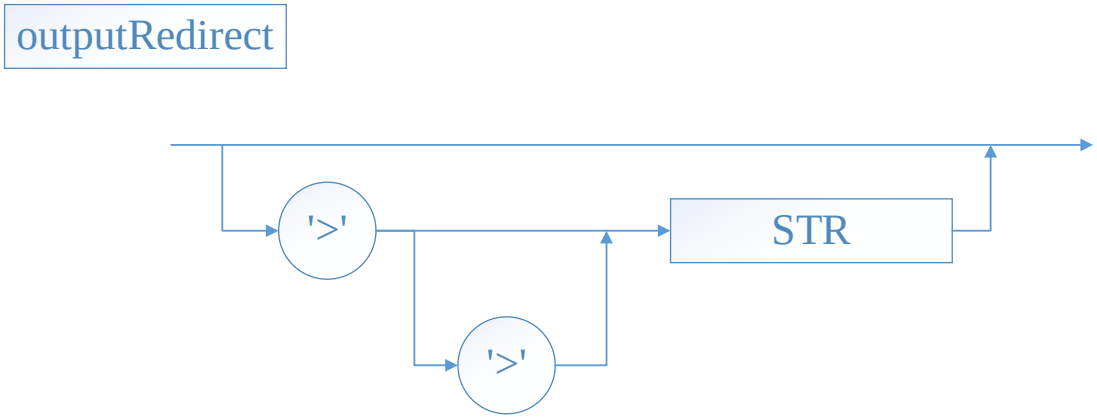

7. 程序调用 progInvocation



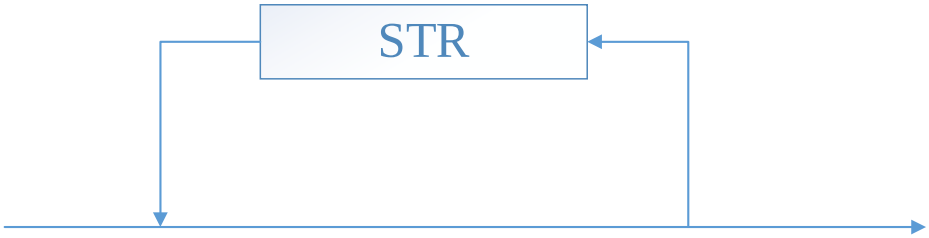

8. 输入重定向 inputRedirect



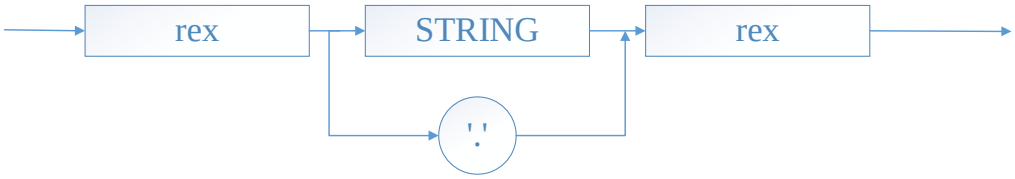

9. 输出重定向 outputRedirect



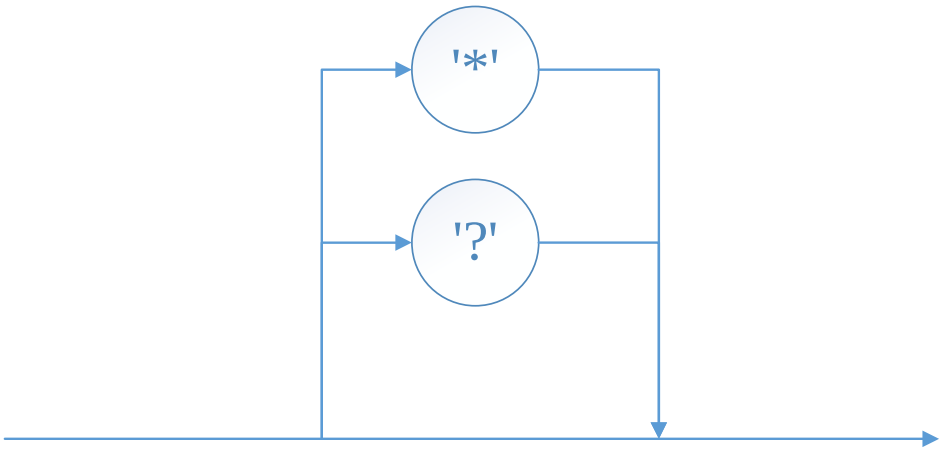
10. 命令参数 args

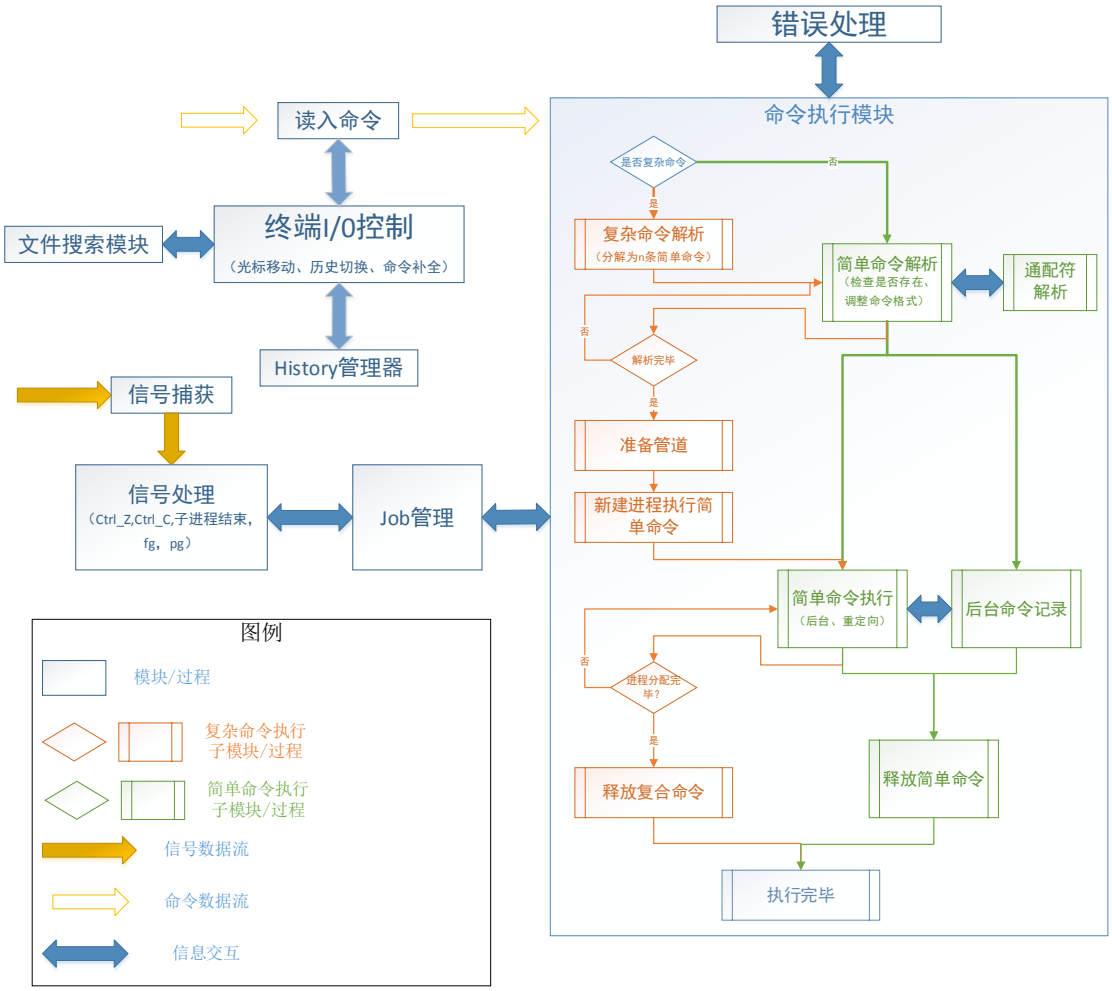

11. 串 STR

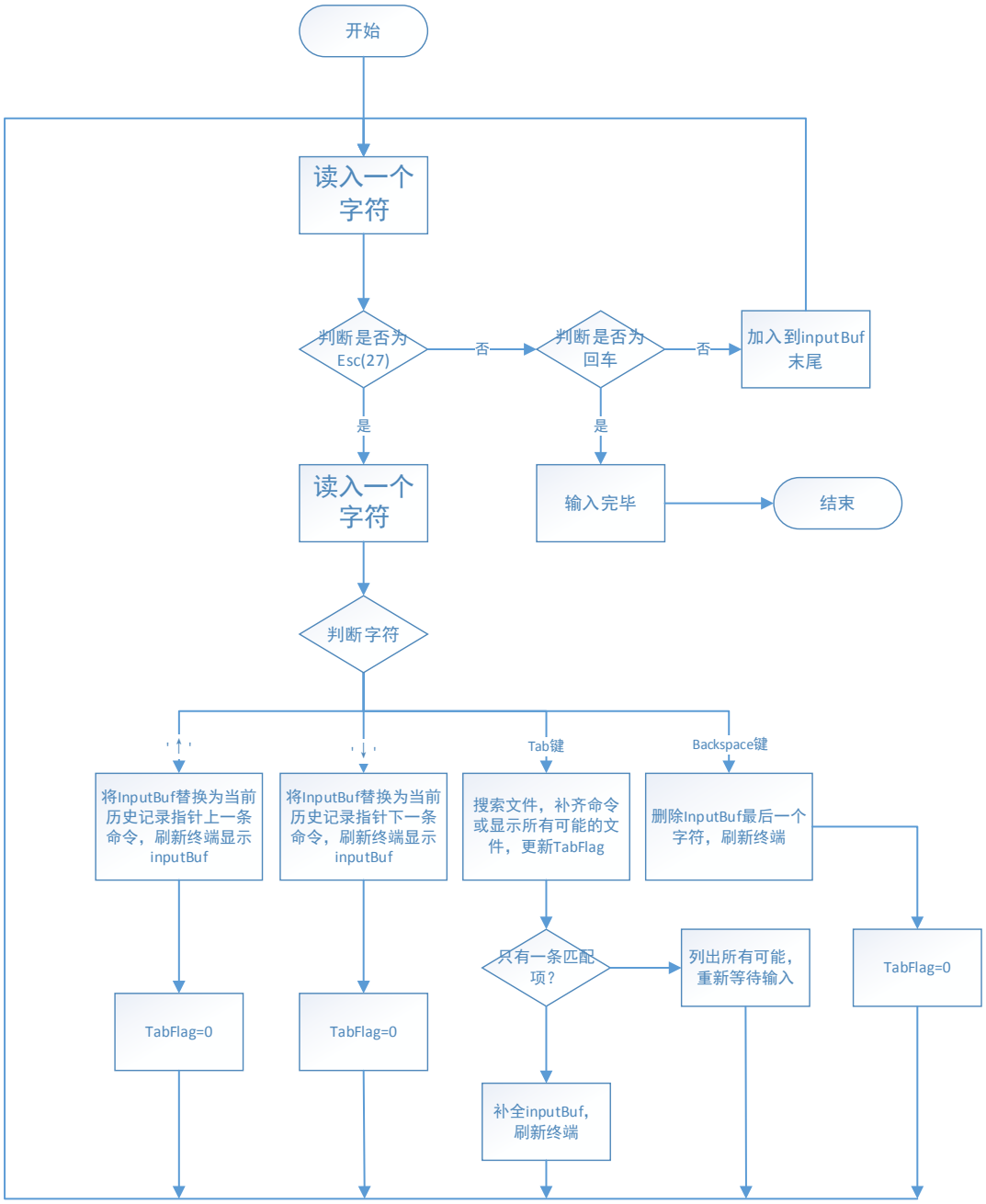

12. 通配符 rex

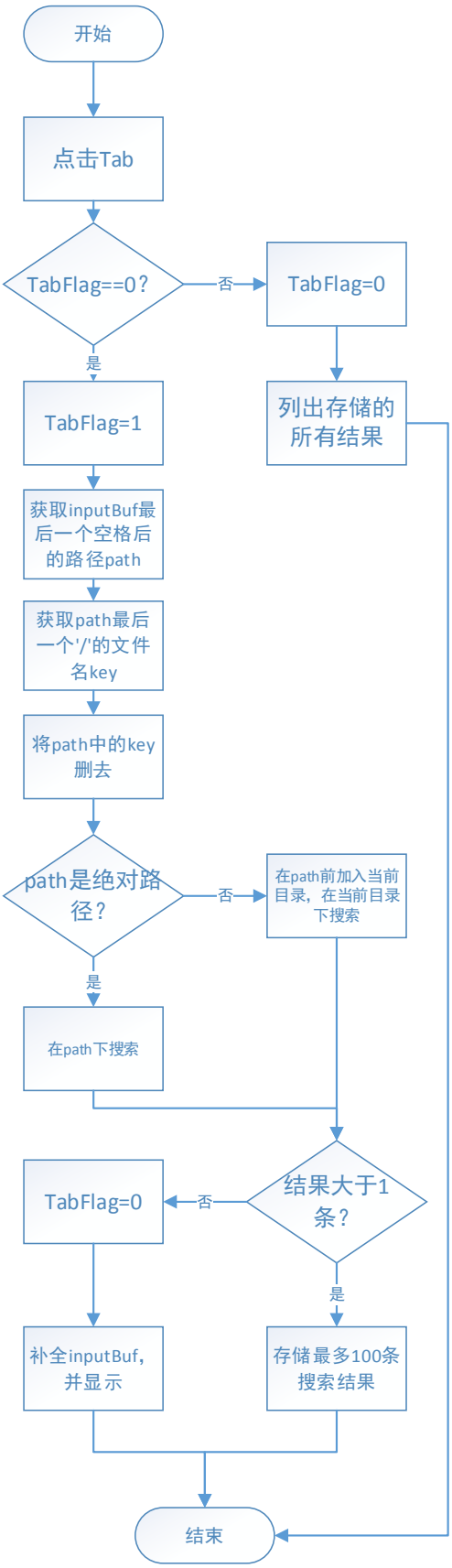
## 2.1.2 程序结构图


错误处理
读入命令
命令执行模块
是否复杂命令
否
是
复杂命令解析
（分解为n条简单命令）
简单命令解析
（检查是否存在、
调整命令格式）
通配符
解析
文件搜索模块
终端I/O控制
（光标移动、历史切换、命令补全）
否
解析完毕
是
History管理器
信号捕获
准备管道
信号处理
（Ctrl_Z,Ctrl_C,子进程结束，
fg，pg）
Job管理
新建进程执行简
单命令
简单命令执行
（后台、重定向）
后台命令记录
否
进程分配完
毕？
是
释放复合命令
释放简单命令
执行完毕
图例
模块/过程
复杂命令执行
子模块/过程
简单命令执行
子模块/过程
信号数据流
命令数据流
信息交互

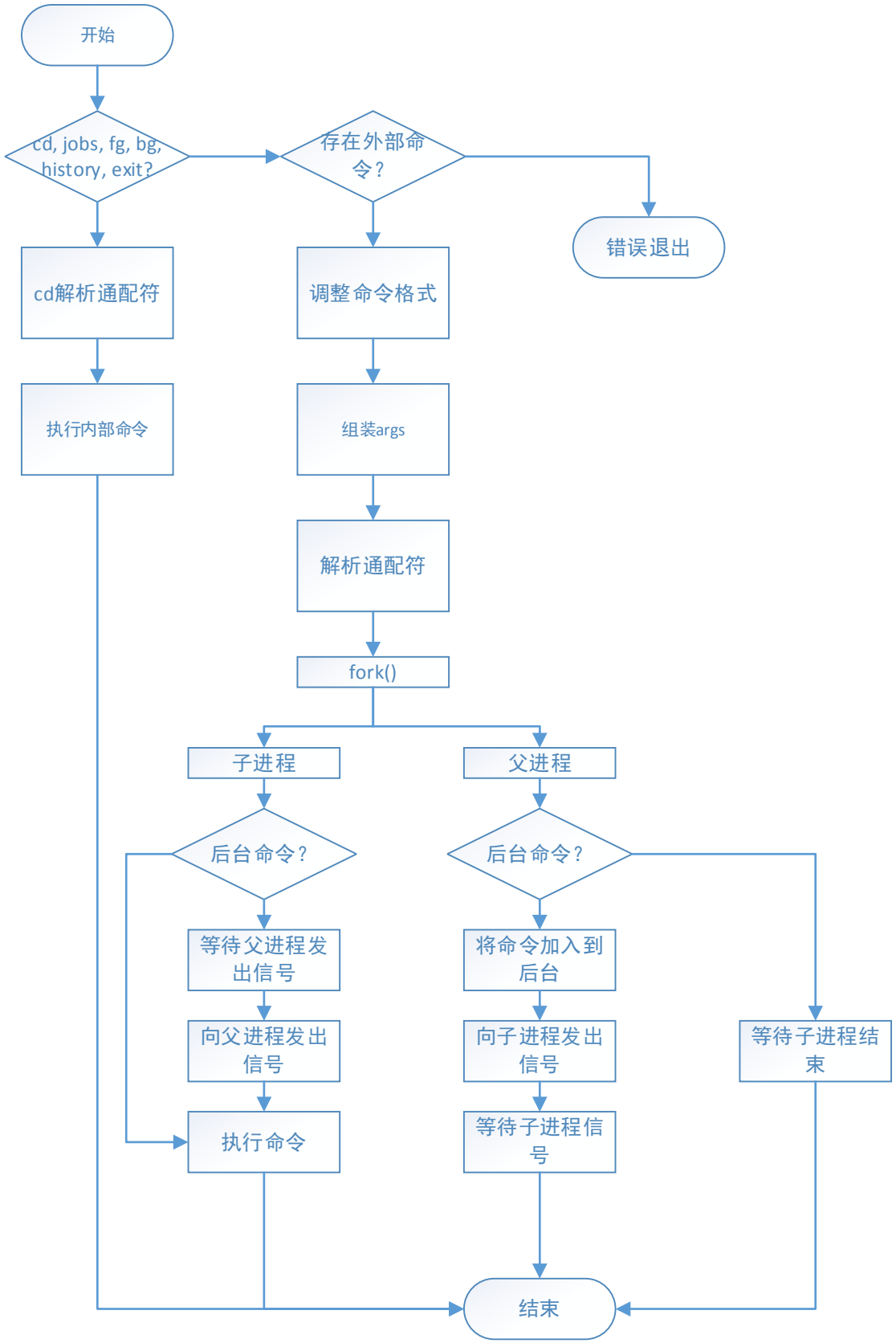
### 2.1.3 终端控制流程图


开始
读入一个字符
判断是否为Esc(27)
否
判断是否为回车
否
加入到inputBuf末尾
是
是
读入一个字符
输入完毕
结束
判断字符
Tab键
Backspace键
将InputBuf替换为当前历史记录指针上一条命令，刷新终端显示inputBuf
将InputBuf替换为当前历史记录指针下一条命令，刷新终端显示inputBuf
搜索文件，补齐命令或显示所有可能的文件，更新TabFlag
删除InputBuf最后一个字符，刷新终端
只有一条匹配项?
列出所有可能，重新等待输入
TabFlag=0
TabFlag=0
TabFlag=0
补全inputBuf，刷新终端

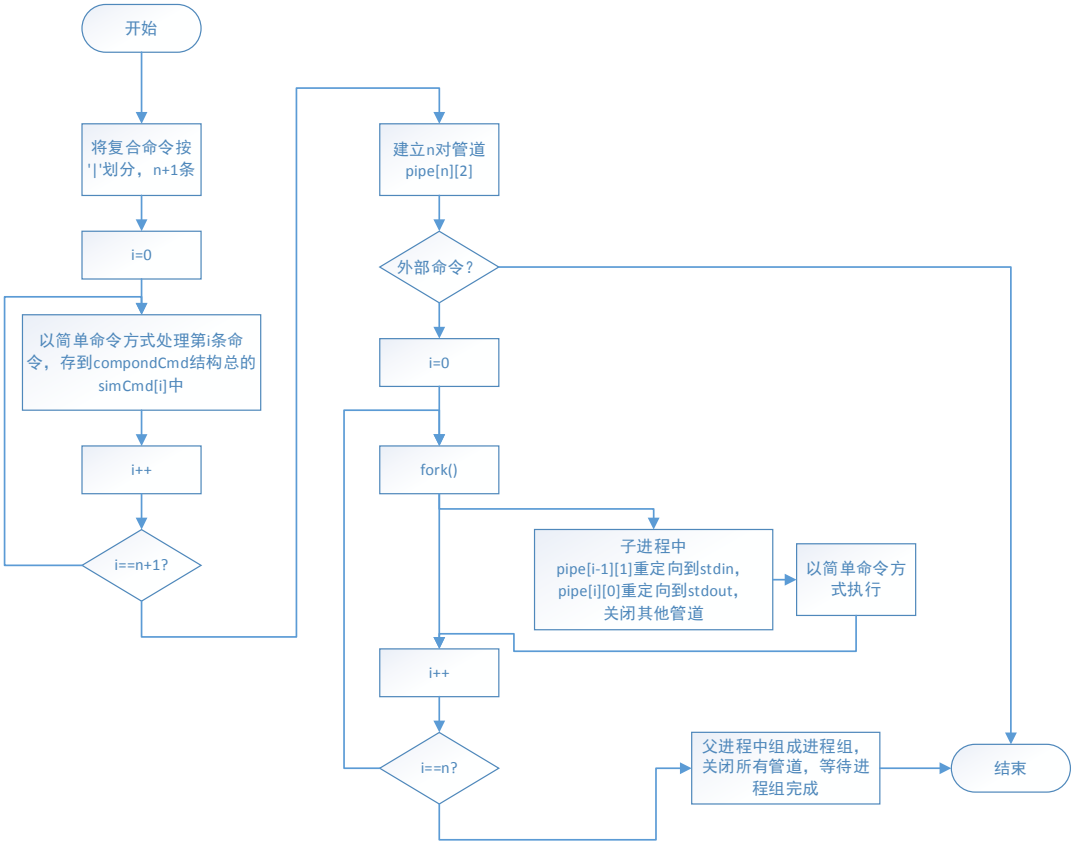
### 2.1.4 Tab 键补全流程图



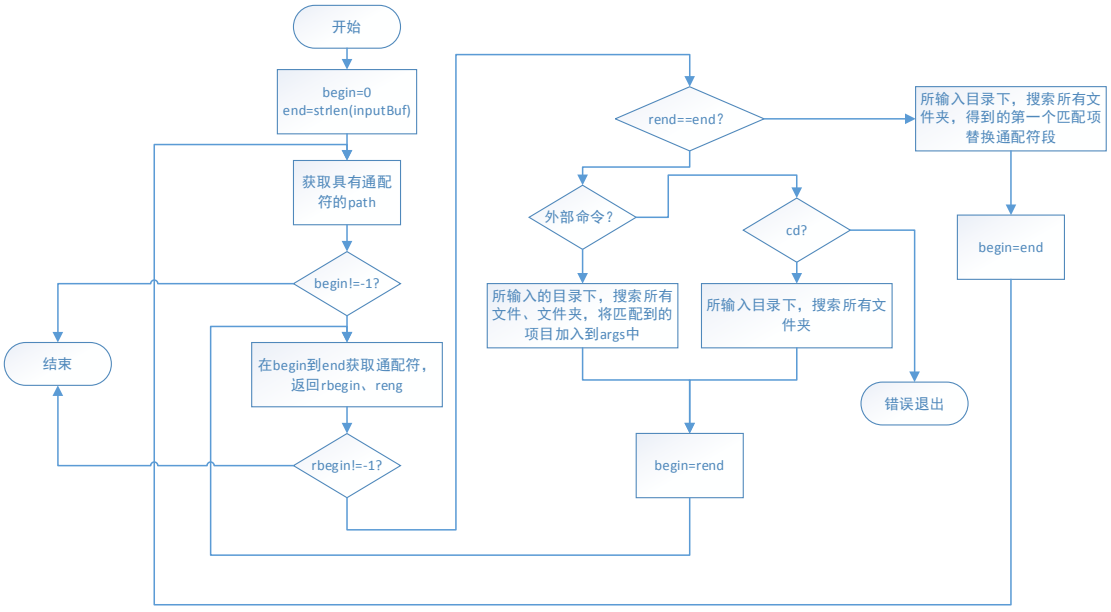
### 2.1.5 简单命令执行流程图



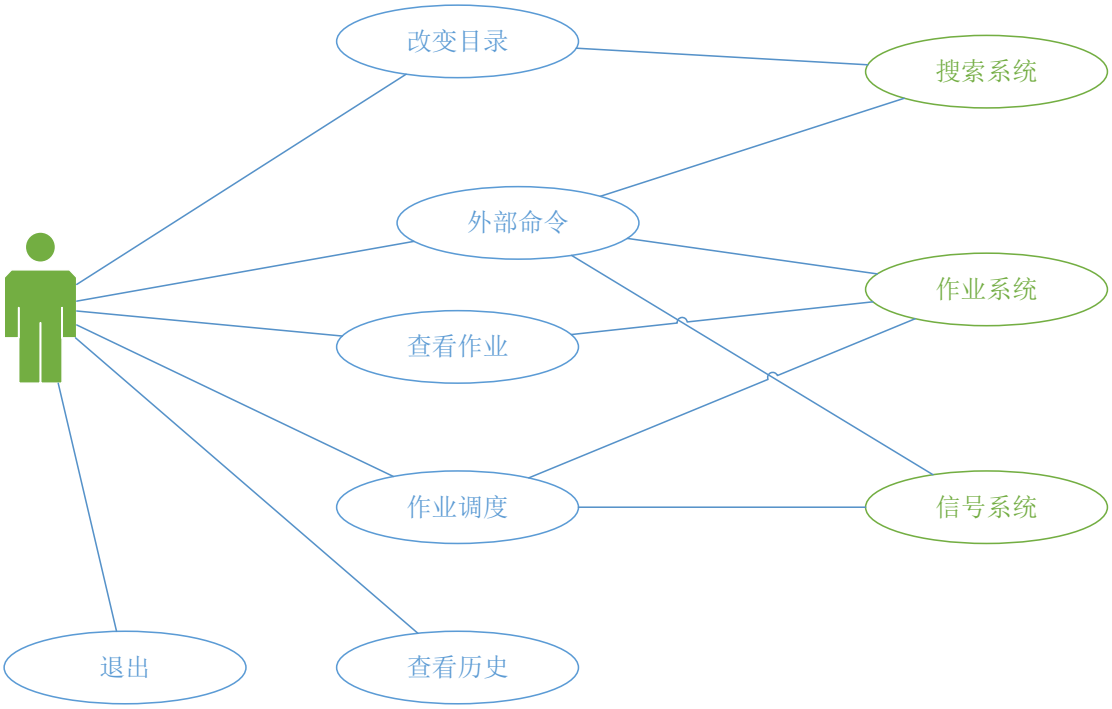
## 2.1.6 复合命令执行流程图




## 2.1.7 通配符流程图



### 2.1.8 用例图




## 2.2 功能设计

这里给出各功能模块的设计方案或实现方法。

### 2.2.1 重要的数据结构设计

#### 2.2.1.1 简单命令

```
typedef struct SimpleCmd {
        int isBack;     // 是否后台运行
        char **args;    // 命令及参数
        char *input;    // 输入重定向
        char *output;   // 输出重定向
    } SimpleCmd;
```

简单命令由四部分组成，其中 isBack 为 1 时表示执行该命令的进程在后台执行，为 0 时在前台执行；**args 为命令及参数，*args[0]为命令名，后面的字符串为它的参数
*input 为输入重定向的文件名，当没有输入重定向时，这个值为 NULL
*output 为输出重定向的文件名，当没有输出重定向时，这个值为 NULL

#### 2.2.1.2 复合命令

```
typedef struct CompondCmd{
        int isBack;
        SimpleCmd **simCmd;
    } CompondCmd;
```

复合命令由两部分组成，其中 isBack 表示执行该命令的进程是否在后台执行，为 1 时在后台执行，为 0 时在前台执行；**simCmd 为一系列简单命令，这些简单命令一起构成了这个复合命令。

#### 2.2.1.3 历史命令

```
typedef struct History {
        int start;
        int end;
        int cur;
        char cmds[HISTORY_LEN][100];
    } History;
```

历史命令由四部分组成，前三部分均为整数，分别代表历史命令的起始下标、终止下标和当前下标，cmds[HISTORY_LEN][100]为储存历史命令的数组，一共可以储存 HISTORY_LEN(设定为 100)条命令。

#### 2.2.1.4 作业

```
typedef struct Job {
        int pid;
        char cmd[100];
        char state[10];
        struct Job *next;
    } Job;
```

作业链表的每一项有四部分，pid 为进程号，cmd[100]为表示该命令的字符串，state[10]为作业状态，*next 为下一作业。

### 2.2.2 主要函数或接口设计

这里给出主要函数或接口的功能说明、实现方法和调用关系。

### 2.2.2.1 函数功能说明

**1. 文件 bison.tab.c 中**

函数格式：`int main(int argc, char** argv)`
函数功能：主函数，整个程序的入口。
参数说明：main 函数参数

函数格式：`int yylex()`
函数功能：词法分析函数
参数说明：无参数

函数格式：`void yyerror(const char *s)`
函数功能：词法错误显示
参数说明：　const char　　*s 错误命令字符串

文件 init.c 中：
函数格式：`extern void getEnvPath(int len, char *buf)`
函数功能：通过路径文件获取环境路径
参数说明：　int len　　数组buff的长度
　　char *buff　　文件路径

函数格式：`extern void init()`
函数功能：初始化操作
参数说明：　无参数

**2. 文件 execute.c 中**

函数格式：`int exists(char *cmdFile)`
函数功能：判断命令是否存在
参数说明：　char *cmdFile　要进行判断的命令字符串

函数格式：`int str2Pid(char *str, int start, int end)`
函数功能：将字符串转换为整形的 Pid
参数说明：　char *str　　要进行转换的字符串
　　int　start　　起始下标
　　int　end　　结束下标

函数格式：`void justArgs(char *str)`
函数功能：调整部分外部命令的格式
参数说明：　char *str　　命令字符串

函数格式：`void release()`
函数功能：释放环境变量空间
参数说明：　无参数

函数格式：void addHistory(char *cmd)
函数功能：添加历史记录
参数说明：　char *cmd　　要添加的命令字符串

函数格式：void freeSimpleCmd(SimpleCmd *scmd)
函数功能：释放简单命令所占空间
参数说明：SimpleCmd *scmd　　目标命令

函数格式：void freeCompondCmd(CompondCmd *ccmd)
函数功能：释放复合命令所占空间
参数说明：CompondCmd *ccmd　　目标命令

函数格式：SimpleCmd* handleSimpleCmdStr(int begin, int end)
函数功能：对简单命令进行解析
参数说明：　int begin　　起始下标
　　int end　结束下标

函数格式：CompondCmd* handleCompondCmdStr(int begin, int end)
函数功能：对复合命令进行解析
参数说明：　int begin　　起始下标
　　int end　结束下标

函数格式：int prepareOutCmd(SimpleCmd *cmd)
函数功能：为执行外部命令做准备
参数说明：　SimpleCmd *cmd　　目标简单命令指针

函数格式：int execOuterCmd(SimpleCmd *cmd)
函数功能：执行外部命令
参数说明：SimpleCmd *cmd　　目标命令指针

函数格式：void execSimpleCmd(SimpleCmd *cmd)
函数功能：执行简单命令
参数说明：SimpleCmd *cmd　　目标命令指针

函数格式：int execCompondCmd(CompondCmd *cmd)
函数功能：执行复合命令
参数说明：CompondCmd *cmd 目标命令指针

函数格式：void execute(int isSimple)
函数功能：执行命令
参数说明：　int isSimple 表示是否是简单命令

**3. 文件 handleInBuff.c 中**

函数格式：int inHistory(int cur)
函数功能：查看指定下标是否在历史列表中
参数说明：int　　cur　当前命令的下标

函数格式：extern int handleInBuff(void)
函数功能：对输入缓冲区进行处理，监听‘↑’、‘↓’和‘Tab’
参数说明：　无参数

**4. 文件 handleJob.c 中**

函数格式：extern Job* addJob(pid_t pid)
函数功能：添加新的作业
参数说明：　pid_t　　pid　进程号

函数格式：extern void rmJob(int sig, siginfo_t *sip, void* noused)
函数功能：移除一个作业
参数说明：　int sig　信号量
　　siginfo_t　　*sip 信号信息

**5. 文件 handleSignal.c 中**

函数格式：extern void ctrl_Z()
函数功能：组合键Ctrl+Z功能的实现
参数说明：　无参数

函数格式：extern void ctrl_C()
函数功能：组合键Ctrl+C功能的实现
参数说明：　无参数

函数格式：extern void fg_exec(int pid)
函数功能：fg 命令的实现
参数说明：int　　pid　进程号

函数格式：extern void bg_exec(int pid)
函数功能：bg 命令的实现
参数说明：int　　pid　进程号

**6. 文件 search.c 中**

函数格式：void cleanTab()
函数功能：清楚 Tab 功能的缓冲区
参数说明：　无参数

函数格式：void sortrecommend()
函数功能：对匹配命令进行排序
参数说明：　无参数

函数格式：extern int tabFile(void)
函数功能：Tab 键功能的实现
参数说明：　无参数

函数格式：int WildCharMatch(char *src, char *pattern, int ignore_case)
函数功能：带有通配符的字符串比较
参数说明：　char *src 原字符串
　　char *pattern　带通配符的字符串
　　int　ignore_case　是否区分大小写

函数格式：extern void getRegex(int *b, int *e, char *path)
函数功能：得到 b,e 之间有通配符最短的不含空格的路径串
参数说明：　int *b 要在 path 当中寻找的开始位置
　　int *e 要在 path 当中寻找的结束位置
　　char *path 要寻找通配符的字符串

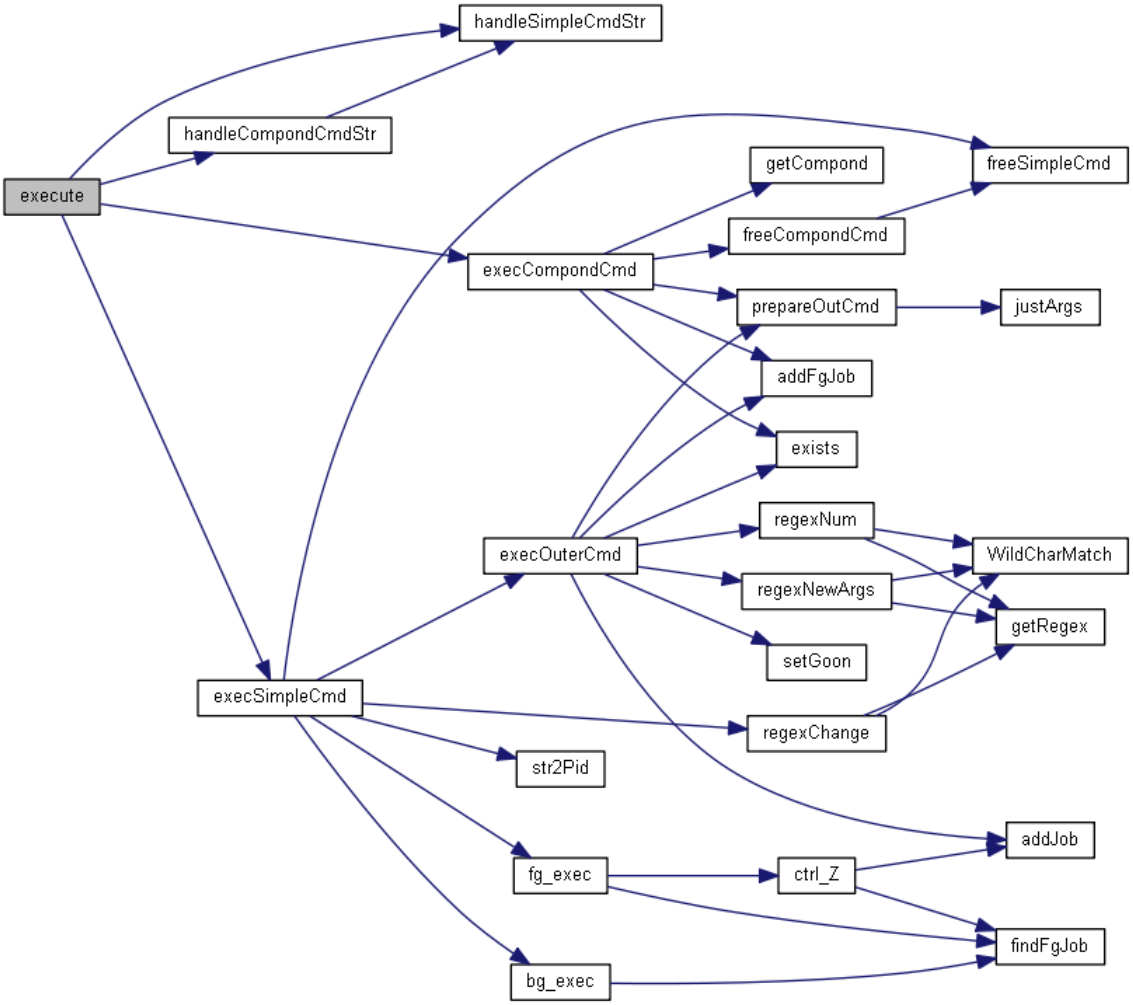
函数格式：extern void regexChange(char *origin, char *newarg, unsigned char d_type)
函数功能：用找到的匹配替换原来的参数
参数说明：　char *origin　原参数字符串
　　char *newarg 替换后的参数字符串
　　unsigned char d_type　匹配文件的类型

函数格式：extern int regexNum(char *origin)
函数功能：得到通配符字符串的个数
参数说明：　char *origin　原参数字符串

函数格式：extern int regexNewArgs(char *origin, char **args, int *index){
函数功能：找到匹配，更换原来的参数
参数说明：　char *origin 原参数字符串
　　char **args 存储新参数
　　int *index 第 index 个参数

#### 2.2.2.2 函数调用关系

这里只给出最主要的函数 execute()调用关系图




# 3 测试和使用说明

## 3.1 使用说明

列出程序的开发环境，如操作系统、使用的编程语言、开发工具和组件等。

列出程序的运行环境，如操作系统、必要的运行库等。

开发环境：

操作系统：Ubuntu12.04

编程语言：C 语言

开发工具：CodeBlocks

运行环境：

操作系统：Ubuntu 12.04

运行库：Bison、Flex、Gcc

## 3.2 测试说明

程序启动：

```
highestop@HighestopUbuntu:~$ cd /home/highestop/Documents/Linux_01/shell/
highestop@HighestopUbuntu:~/Documents/Linux_01/shell$ ./xsh
```

（1） ls 命令显示文件目录

输入：ls [–color=auto]

输出：该目录下的所有文件和文件夹

```
xsh@/home/highestop/Documents/Linux_01/shell>ls --color=auto
a     bison.output  execute.c       handleSignal.o  search.o      temp5
a1    bison.tab.c   execute.o       init.c          shell.cbp     ttt
a2    bison.tab.h   global.h        init.o          shell.depend  tttt
a3    bison.tab.o   grep            lex.l           shell.layout  xsh
abc   bison.y       handleInBuff.c  lex.yy.c        temp          ysh.conf
a.c   c             handleInBuff.o  main.c          temp1
b     c.c           handleJob.c     makefile        temp2
b.c   daf           handleJob.o     obj             temp3
bin   dsfadf        handleSignal.c  search.c        temp4
```

（2） cd 命令退出和进入文件夹目录

输入：cd abc（abc 为运行目录下的一文件夹）

输出：切换到下层目录

```
xsh@/home/highestop/Documents/Linux_01/shell>cd abc
xsh@/home/highestop/Documents/Linux_01/shell/abc>
```

输入：cd ..

输出：切换到上层目录

```
xsh@/home/highestop/Documents/Linux_01/shell/abc>cd a3
xsh@/home/highestop/Documents/Linux_01/shell/abc/a3>cd ..
xsh@/home/highestop/Documents/Linux_01/shell/abc>cd ..
xsh@/home/highestop/Documents/Linux_01/shell>
```

（3） 运行程序并进行前后台切换

输入：./a

输出：30s 后自动结束程序

```
xsh@/home/highestop/Documents/Linux_01/shell>./a
30
29
28
27
26
```

输入：Ctrl+C

输出：终止进程

```
xsh@/home/highestop/Documents/Linux_01/shell>./a
30
29
28
27
26
25
^C
xsh@/home/highestop/Documents/Linux_01/shell>jobs
There is no jobs.
```

输入：Ctrl+Z

输出：终止进程

```
xsh@/home/highestop/Documents/Linux_01/shell>./a
30
29
28
27
^Z
[3585]  STOPPED         ./a&
```

输入：jobs（查看挂起的进程以及 PID）

输出：显示挂起的进程的信息

```
xsh@/home/highestop/Documents/Linux_01/shell>jobs
INDEX   PID     STATE           COMMAND
1       3585    STOPPED         ./a&
```

输入：fg %3600（此时其它命令无效）

输出：前台继续执行该进程

```
xsh@/home/highestop/Documents/Linux_01/shell>fg %3600
[3600]  RUNNING         ./a&
xsh@/home/highestop/Documents/Linux_01/shell>26
25
24
23
```

输入：bg %3585（此时可运行其它命令）及 ls 命令

输出：后台继续执行该进程

```
xsh@/home/highestop/Documents/Linux_01/shell>bg %3585
[3585]  RUNNING         ./a&
xsh@/home/highestop/Documents/Linux_01/shell>26
25
24
23
ls22

a     bison.output  execute.c       handleSignal.o  search.o      temp5
a1    bison.tab.c   execute.o       init.c          shell.cbp     ttt
a2    bison.tab.h   global.h        init.o          shell.depend  tttt
a3    bison.tab.o   grep            lex.l           shell.layout  xsh
abc   bison.y       handleInBuff.c  lex.yy.c        temp          ysh.conf
a.c   c             handleInBuff.o  main.c          temp1
b     c.c           handleJob.c     makefile        temp2
b.c   daf           handleJob.o     obj             temp3
bin   dsfadf        handleSignal.c  search.c        temp4
xsh@/home/highestop/Documents/Linux_01/shell>21
20
19
```

输入：./a&（直接后台运行，可同时执行其它命令）及 ls 命令

输出：给出进程信息，运行程序，并可同时执行 ls 命令

```
xsh@/home/highestop/Documents/Linux_01/shell>./a&
[3358]  RUNNING         ./a&
xsh@/home/highestop/Documents/Linux_01/shell>30
29
28
27
26
25
ls
a     bison.output  execute.c       handleSignal.o  search.o      temp5
a1    bison.tab.c   execute.o       init.c          shell.cbp     ttt
a2    bison.tab.h   global.h        init.o          shell.depend  tttt
a3    bison.tab.o   grep            lex.l           shell.layout  xsh
abc   bison.y       handleInBuff.c  lex.yy.c        temp          ysh.conf
a.c   c             handleInBuff.o  main.c          temp1
b     c.c           handleJob.c     makefile        temp2
b.c   daf           handleJob.o     obj             temp3
bin   dsfadf        handleSignal.c  search.c        temp4
xsh@/home/highestop/Documents/Linux_01/shell>24
23
22
21
```

（4） 重定向输入和输出

输出重定向：

输入：ls –n > temp（重定向输出）以及 cat temp（查看文件内容）

输出：ls –n 的输出内容被存到 temp 文件中

```
xsh@/home/highestop/Documents/Linux_01/shell>ls -n > temp
xsh@/home/highestop/Documents/Linux_01/shell>cat temp
total 428
-rwxrwxr-x 1 1000 1000  7411  4月 10 20:18 a
-rw------- 1 1000 1000   104  4月 10 20:19 a1
-rw------- 1 1000 1000   104  4月 10 20:20 a2
-rw------- 1 1000 1000    44  4月 10 20:16 a3
drwxrwxr-x 4 1000 1000  4096  4月 17 23:09 abc
-rwxrw-rw- 1 1000 1000   145  4月 10 20:18 a.c
-rwxrw-rw- 1 1000 1000  7373  4月  9 13:53 b
-rwxrw-rw- 1 1000 1000    78  4月  9 01:27 b.c
drwxrwxr-x 4 1000 1000  4096  4月 17 14:10 bin
-rwxrw-rw- 1 1000 1000  7041  4月  8 22:20 bison.output
-rwxrw-rw- 1 1000 1000 48868  4月 17 11:07 bison.tab.c
-rwxrw-rw- 1 1000 1000  1978  4月 17 11:07 bison.tab.h
-rw-rw-r-- 1 1000 1000 15132  4月 16 15:47 bison.tab.o
-rwxrw-rw- 1 1000 1000  3550  4月 17 11:07 bison.y
-rwxrw-rw- 1 1000 1000  7337  4月  9 13:53 c
```

输入重定向：

输入：cat -n < a.c

输出：把 a.c 的内容加上行号显示在屏幕上

```
xsh@/home/tong/shell>cat -n < a.c
     1  #include<stdio.h>
     2
     3  main(){
     4  int i;
     5  for(i=30;i>0;i--){
     6  sleep(1);
     7  printf("%d\n",i);
     8  }
     9  printf("ready.....\n");
    10  sleep(1);
    11  printf("boooooooom!\n");
    12
    13  }
```

（5） TAB 键补全及通配符

输入：a+TAB

输出：a1/ a2/ a3/

```
xsh@/home/highestop/Documents/Linux_01/shell>ls
a    a.c          bison.tab.c  c.c        global.h       handleJob.o     lex.l     search.c      temp   temp5
a1   b            bison.tab.h  daf        grep           handleSignal.c  lex.yy.c  search.o      temp1  ttt
a2   b.c          bison.tab.o  dsfadf     handleInBuff.c handleSignal.o  main.c    shell.cbp     temp2  tttt
a3   bin          bison.y      execute.c  handleInBuff.o init.c          makefile  shell.depend  temp3  xsh
abc  bison.output c            execute.o  handleJob.c    init.o          obj       shell.layout  temp4  ysh.conf
xsh@/home/highestop/Documents/Linux_01/shell>cd abc
xsh@/home/highestop/Documents/Linux_01/shell/abc>ls
a1  a2  a3  defg
xsh@/home/highestop/Documents/Linux_01/shell/abc>a
a1/     a2/     a3/
xsh@/home/highestop/Documents/Linux_01/shell/abc>a
```

输入：cd ab*/a?（或 rm –f ab*/a?）

输出：abc/a1（假如 abc/目录下有 a1，a2 和 a3 三个目录，匹配到之后会选择第 1 个符合要求的进入）

```
xsh@/home/highestop/Documents/Linux_01/shell>cd ab*/a?
xsh@/home/highestop/Documents/Linux_01/shell/abc/a1
```

（6） 管道命令

输入：ls | cat -n | cat -n

输出：显示当前目录下所有文件名并在前面加入两个行号

```
xsh@/home/tong/shell>ls | cat -n | cat -n
     1        1  a
     2        2  a1
     3        3  a111
     4        4  a2
     5        5  a3
     6        6  aaaaaa
     7        7  ab
     8        8  abc
     9        9  a.c
    10       10  b
    11       11  b.c
    12       12  bin
    13       13  bison.output
    14       14  bison.tab.c
    15       15  bison.tab.h
    16       16  bison.tab.o
    17       17  bison.y
    18       18  c
    19       19  c.c
    20       20  daf
    21       21  dsfadf
```

管道与重定向组合：

输入：cat -n < b.c | cat -n > b.c.c.c

cat b.c.c.c

输出：首先第一条命令将 b.c 中的内容加上了两个行号输出到了 b.c.c.c 中

第二条命令为显示 b.c.c.c 的内容

```
xsh@/home/tong/shell>cat -n < b.c | cat -n > b.c.c.c
xsh@/home/tong/shell>cat b.c.c.c
     1        1  #include <stdio.h>
     2        2
     3        3  int main(){
     4        4          sleep(3);
     5        5          printf("OKKKK@!\n");
     6        6          return 0;
     7        7  }
```

（7） history 历史记录以及上下键

输入：history

输出：所有历史记录

```
xsh@/home/highestop/Documents/Linux_01/shell>history
history
cd abc
ls
cd a1
ls
cd ..
cd ..
rm -f abc/a1
rm -f abc/a1/aaa
rm abc/a1
```

# 4 会议记录

**（1） 第一次会议**

| 组员 | 前一阶段任务 | 是否完成 | 后一阶段任务 |
|---|---|---|---|
| 童浩 | 确定组员分工，学习 Linux 进程编程相关的知识 | 是 | 程序整体框架的设计及改进文法 |
| 解佳琦 | | 是 | 对键盘输入进行监听，并调整输入字符串 |
| 陈宇宁 | | 是 | 实现输入输出重定向的功能 |
| 邱伟豪 | | 是 | 实现前台、后台任务等基本功能 |

**（2） 第二次会议**

| 组员 | 前一阶段任务 | 是否完成 | 后一阶段任务 |
|---|---|---|---|
| 童浩 | 程序整体框架的设计及改进文法 | 是 | 实现管道功能 |
| 解佳琦 | 对键盘输入进行监听，并调整输入字符串 | 是 | 实现‘Tab’键的功能 |
| 陈宇宁 | 实现输入输出重定向的功能 | 是 | 实现查看历史命令和‘↑’‘↓’键功能 |
| 邱伟豪 | 实现前台、后台任务等基本功能 | 是 | 整理全局代码保证功能实现 |

**（3） 第三次会议**

| 组员 | 前一阶段任务 | 是否完成 | 后一阶段任务 |
|---|---|---|---|
| 童浩 | 实现管道功能 | 是 | 对管道与其他命令整合调试并完善代码 |
| 解佳琦 | 实现‘Tab’键的功能 | 是 | 在命令中加入通配符匹配的功能 |
| 陈宇宁 | 实现查看历史命令和‘↑’‘↓’键功能 | 是 | 开始进行文档的编写 |
| 邱伟豪 | 整理全局代码保证功能实现 | 是 | 程序功能的测试 |

**（4） 第四次会议**

| 组员 | 前一阶段任务 | 是否完成 | 后一阶段任务 |
|---|---|---|---|
| 童浩 | 对管道与其他命令整合调试并完善代码 | 是 | 编写自己负责的部分的文档 |
| 解佳琦 | 完成 TAB 键匹配及通配符代码 | 是 | |
| 陈宇宁 | 开始进行文档的编写 | 是 | |
| 邱伟豪 | 程序功能的测试 | 是 | |

# 5 其它说明

| 学号 | 姓名 | 分工情况 | 工作量比例 |
|---|---|---|---|
| 10231016 | 童浩（组长） | 总体程序框架<br>对文法进行改进<br>管道的实现 | 29% |
| 10231008 | 解佳琦 | 键盘监听并调整输入字符串<br>TAB 键及通配符匹配功能 | 26% |
| 10231024 | 陈宇宁 | 重定向输入输出<br>查看历史命令及‘↑’‘↓’键功能 | 23% |

| | | | |
|---|---|---|---|
| 38230213 | 邱伟豪 | 基本命令的实现<br>程序测试 | 22% |

# 6 程序清单

## 6.1 源代码

（路径：~\源代码\shell）

### 6.1.1shell 程序

bison.y
execute.c
handleInBuff.c
handleJob.c
handleSignal.c
init.c
search.c
global.h
ysh.conf
makefile
可执行程序：
xsh

### 6.2.1 测试用程序

a.c　b.c　c.c

## 6.2 可执行程序

（路径：~\可执行程序\shell）

### 6.1.1shell 程序

xsh

### 6.2.1 测试用程序

a：等待 30 秒，程序结束。显示倒计时。
b：等待 3 秒，程序结束。
c：等待 5 秒，程序结束。

## 6.2 其他目录

ab、abc 文件夹及其以下目录，测试 cd、通配符等所用文件夹和文件。