

shell的简单实现

实现的功能

- 执行系统内置程序 `/bin/`
- Tab自动补全，上下键切换命令
- shell内置命令: `exit history cd`
- 管道: `operationA | operationB`
- 输入输出重定向: `> >> <`

存在的问题

- 管道只能支持两个操作，管道还没有考虑参数只有一个的情况，当参数只有一个执行的是 `execvp(NULL, NULL)` 命令，不会向用户提示错误
- 输入输出重定向只能支持一个文件

文件目录

```
1 shell/
2 |— Makefile    #make文件
3 |— README.md   #本文件
4 |— shell       #编译成功的可执行文件
5 |— shell.c     #shell源代码
6 |— shell.h     #shell头文件
```

编译运行

- `sudo apt-get install libreadline-dev` 安装 `readline` 库
- `make all` 编译
- `./shell` 运行
- `make clen` 清除编译产生的临时文件

运行效果

编译

```
Windows PowerShell x ./.shell x + v - - - x
# leo @ LAPTOP-MVSJ24RV in ~/c/Desktop/OSExperiments/shell on git:master x [18:41:13]
$ make all
gcc -c -o shell.o shell.c -w -I ./shell.h
gcc -o shell shell.o -lreadline
compile done

# leo @ LAPTOP-MVSJ24RV in ~/c/Desktop/OSExperiments/shell on git:master x [18:41:24]
$ ./shell
# leo @ LAPTOP-MVSJ24RV in /mnt/c/Users/leo/Desktop/OSExperiments/shell $ _
```

🔧 系统功能

```
Windows PowerShell x ./.shell x .t/c/Users/leo x + v - - - x
# leo @ LAPTOP-MVSJ24RV in /mnt/c/Users/leo/Desktop/OSExperiments/shell $ ls
Makefile README.md README.pdf shell shell.c shell.h shell.o
# leo @ LAPTOP-MVSJ24RV in /mnt/c/Users/leo/Desktop/OSExperiments/shell $ ls -all
total 440
drwxrwxrwx 1 leo leo 4096 Oct 17 18:47 .
drwxrwxrwx 1 leo leo 4096 Oct 17 17:48 ..
-rwxrwxrwx 1 leo leo 188 Oct 17 18:20 Makefile
-rwxrwxrwx 1 leo leo 1443 Oct 17 18:47 README.md
-rwxrwxrwx 1 leo leo 386444 Oct 17 18:45 README.pdf
-rwxrwxrwx 1 leo leo 22616 Oct 17 18:41 shell
-rwxrwxrwx 1 leo leo 14658 Oct 17 17:33 shell.c
-rwxrwxrwx 1 leo leo 1004 Oct 17 17:28 shell.h
-rwxrwxrwx 1 leo leo 12128 Oct 17 18:41 shell.o
# leo @ LAPTOP-MVSJ24RV in /mnt/c/Users/leo/Desktop/OSExperiments/shell $ ps -e
PID TTY TIME CMD
1 ? 00:00:00 init
6 tty1 00:00:00 init
7 tty1 00:01:34 zsh
6343 tty1 00:00:00 shell
9930 tty1 00:00:00 shell
9946 tty1 00:00:00 ps
# leo @ LAPTOP-MVSJ24RV in /mnt/c/Users/leo/Desktop/OSExperiments/shell $ pwd
/mnt/c/Users/leo/Desktop/OSExperiments/shell
# leo @ LAPTOP-MVSJ24RV in /mnt/c/Users/leo/Desktop/OSExperiments/shell $ _
```

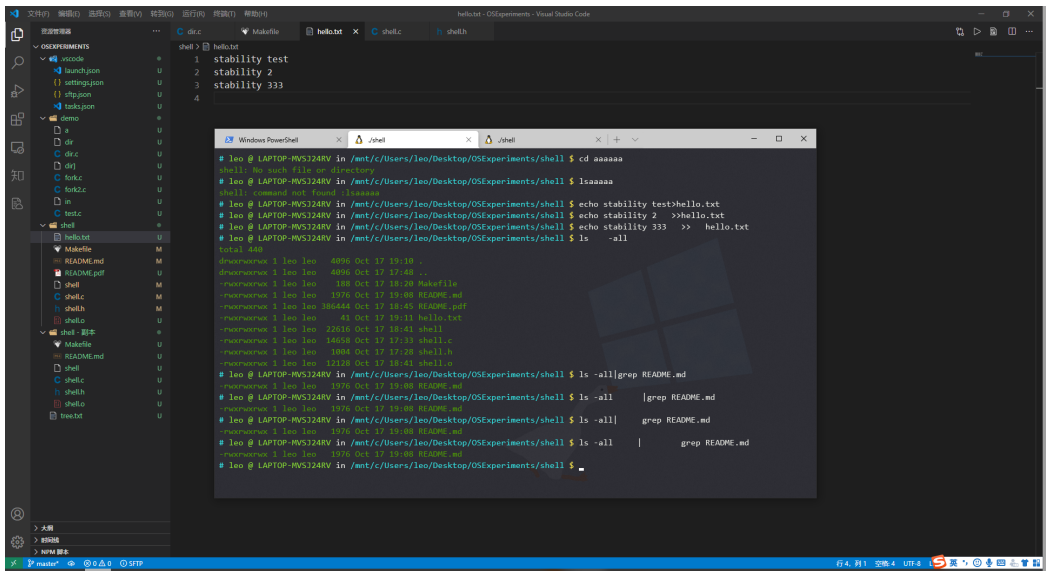
🔧 shell内置命令

```
Windows PowerShell x .riments/shell x .t/c/Users/leo x + v - □ x
# leo @ LAPTOP-MV5J24RV in /mnt/c/Users/leo/Desktop/OSExperiments/shell $ ls
Makefile README.md README.pdf shell shell.c shell.h shell.o
# leo @ LAPTOP-MV5J24RV in /mnt/c/Users/leo/Desktop/OSExperiments/shell $ cd ~
# leo @ LAPTOP-MV5J24RV in ~ $ ls
leo5 c install.sh tools
# leo @ LAPTOP-MV5J24RV in ~ $ cd /mnt/c/Users/leo/Desktop/OSExperiments/shell
# leo @ LAPTOP-MV5J24RV in /mnt/c/Users/leo/Desktop/OSExperiments/shell $ ls
Makefile README.md README.pdf shell shell.c shell.h shell.o
# leo @ LAPTOP-MV5J24RV in /mnt/c/Users/leo/Desktop/OSExperiments/shell $ history
ls
cd
ls
cd
exit
clear
ls
cd
ls
cd
ls
history
# leo @ LAPTOP-MV5J24RV in /mnt/c/Users/leo/Desktop/OSExperiments/shell $ exit
# leo @ LAPTOP-MV5J24RV in ~\c\Desktop\OSExperiments\shell on git:master x [18:55:27]
$
```

管道 输入输出重定向

```
Makefile x shell x shell.c x shell.h
shell > | cat
1 hello first
2 hello second
3 hello third
4
# leo @ LAPTOP-MV5J24RV in /mnt/c/Users/leo/Desktop/OSExperiments/shell $ ls
# leo @ LAPTOP-MV5J24RV in /mnt/c/Users/leo/Desktop/OSExperiments/shell $ echo hello first > hello.txt
# leo @ LAPTOP-MV5J24RV in /mnt/c/Users/leo/Desktop/OSExperiments/shell $ echo hello second >> hello.txt
# leo @ LAPTOP-MV5J24RV in /mnt/c/Users/leo/Desktop/OSExperiments/shell $ echo hello third >> hello.txt
# leo @ LAPTOP-MV5J24RV in /mnt/c/Users/leo/Desktop/OSExperiments/shell $ wc -l < hello.txt
4
# leo @ LAPTOP-MV5J24RV in /mnt/c/Users/leo/Desktop/OSExperiments/shell $ ls -all
total 440
drwxrwxr-x 1 leo leo 4096 Oct 17 18:01 .
drwxrwxr-x 1 leo leo 4096 Oct 17 17:48 ..
-rwxrwxr-x 1 leo leo 180 Oct 17 18:20 Makefile
-rwxrwxr-x 1 leo leo 1770 Oct 17 18:57 README.md
-rwxrwxr-x 1 leo leo 208444 Oct 17 18:45 README.pdf
-rwxrwxr-x 1 leo leo 32 Oct 17 19:02 hello.txt
-rwxrwxr-x 1 leo leo 22016 Oct 17 18:41 shell
-rwxrwxr-x 1 leo leo 14008 Oct 17 17:33 shell.c
-rwxrwxr-x 1 leo leo 1084 Oct 17 17:28 shell.h
-rwxrwxr-x 1 leo leo 12120 Oct 17 18:41 shell.o
# leo @ LAPTOP-MV5J24RV in /mnt/c/Users/leo/Desktop/OSExperiments/shell $ ls -all | grep README.md
-rwxrwxr-x 1 leo leo 1770 Oct 17 18:57 README.md
# leo @ LAPTOP-MV5J24RV in /mnt/c/Users/leo/Desktop/OSExperiments/shell $ ls -all | wc -l
11
# leo @ LAPTOP-MV5J24RV in /mnt/c/Users/leo/Desktop/OSExperiments/shell $
```

健壮性测试



代码

Makefile

```
1
2  shell.o : shell.c
3      gcc -c -o shell.o shell.c -w -I ./shell.h
4  shell.bin : shell.o
5      gcc -o shell shell.o -lreadline
6  all : shell.bin
7      @echo "compile done"
8  clean :
9      rm *.o shell
```

shell.h

```
1  #ifndef __SHELL_H__
2  #define __SHELL_H__
3
4  #include <stdio.h>
5  #include <sys/types.h>
6  #include <unistd.h>
7  #include <pwd.h>
8  #include <readline/readline.h>
9  #include <readline/history.h>
10 #include <fcntl.h>
11 #include <dirent.h>
12 #include <string.h>
13 #include <stdlib.h>
14 #include <unistd.h>
15
16 #define INSTRUCTION_WRONG 0
17 #define INSTRUCTION_CD 1
18 #define INSTRUCTION_EXIT 2
19 #define INSTRUCTION_HISTORY 3
```

```

20 #define INSTRUCTION_SYS_EXE 4
21
22 #define SYSTEM_EXE 1
23 #define NOT_SYSTEM_EXE 0
24
25 #define REDIRECTION_IN 1
26 #define REDIRECTION_OUT 2
27 #define REDIRECTION_OUT_ADD 3
28 #define REDIRECTION_PIPE 4
29 #define REDIRECTION_NONE 0
30
31 #define STDIN 0
32 #define STDOUT 1
33 #define STDERR 2
34
35 char ** getParameter(char* args);
36 int shell();
37 void trim(char* args);
38 int cd(char** args);
39 char * getUserName();
40 int systemExe(char** args);
41 int whichOperation(char** args);
42 int redirectionType(char** args);
43 char* getRedirectionFile(char** args, int type);
44 char* cmdPreprocess(char* cmd);
45 #endif //

```

shell.c

```

1 #include "shell.h"
2 #define EXIT_STATUS_S 1
3
4
5 extern void add_history PARAMS((const char *));
6
7 /* Add the contents of FILENAME to the history list, a line
   at a time.
8   If FILENAME is NULL, then read from ~/.history. Returns 0
   if
9   successful, or errno if not. */
10 extern int read_history PARAMS((const char *));
11
12 /* Write the current history to FILENAME. If FILENAME is
   NULL,
13   then write the history list to ~/.history. values
   returned
14   are as in read_history (). */
15 extern int write_history PARAMS((const char *));
16
17 /* Return a NULL terminated array of HIST_ENTRY which is the
   current input history.

```

```

18 Element 0 of this list is the beginning of time. If there is
   no history, return NULL.*/
19 extern HIST_ENTRY **history_list PARAMS((void));
20
21 /**
22  * @brief shell程序
23  * @note
24  * @retval
25  */
26 int shell() {
27     pid_t pid,pid_pipe;
28     int fd,fd_pipe; //文件描述符
29     int flag=0,j=0; //管道判断参数是否完整
30
31     char*cmd; //读取用户输入
32     char*prompt;
33     char hostname[40]={" @ LAPTOP-MVSJ24RV "};
34     char** args;
35     char* username;
36     int printfReturn=0;
37     char userhome[32]={"/home/"};
38     char tempCmd[40];
39
40     char* argsFirst[10];
41     char* argsSecond[10];
42
43     char file_path_getcwd[80];
44
45     /*输出配色方案*/
46     char colorUser[50] = {"\001\033[1;32m\002"}; //绿色高亮显示
   用户名字 hostname
47     char colorDirectory[50] = {"\001\033[1;36m\002"}; //紫色显示
   当前文件目录
48     char colorReset[50] = {"\001\033[0m\002"}; //显示恢复默认值
49     char colorShow[50] = {"\033[5;32m"};
50
51     /*从~/.history读出，之前每次放在循环里面，
52     因此每次都会让history的list长度加倍*/
53     read_history(NULL);
54     do{
55         /*获取当前路径*/
56         getcwd(file_path_getcwd,80);
57
58         /*获取当前用户*/
59         username = getUsername();
60
61         //当前路径是用户目录，显示~
62         strcat(userhome,username);
63         if(strcmp(file_path_getcwd,userhome)==0) {
64             strcpy(file_path_getcwd,"~");
65         }
66         memset(userhome,0,sizeof(userhome));

```

```

67     strcpy(userhome, "/home/");
68
69     /*提示信息拼接*/
70     prompt = (char*)malloc(sizeof(char)*128);
71     memset(prompt, 0, sizeof(char)*128);
72     strcat(prompt, "\001\033[1;36m# \033[0m\002");
73     strcat(prompt, colorUser);
74     strcat(prompt, username);
75     strcat(prompt, hostname);
76     strcat(prompt, "\001\033[0m\002in ");
77     strcat(prompt, colorDirectory);
78     strcat(prompt, file_path_getcwd);
79     strcat(prompt, "\001\033[1;33m\002 $ ");
80     strcat(prompt, colorReset);
81
82     fflush(stdin);
83     cmd = readline(prompt);
84     strcpy(tempCmd, cmd);
85     fflush(stdin);
86     printf("%s", colorShow); //更改输出的颜色
87     fflush(stdout);
88     cmd = cmdPreprocess(cmd);
89
90     /*去除命令中多余的空格*/
91     trim(cmd);
92
93     /*将字符串中命令存放到二级指针，空格分隔，每一个存放到一个数组元
    素中*/
94     args = getParameter(cmd); /*通过一个二级指针返回，不能确定
    二级指针的长度，二级指针的最后一个元素是NULL*/
95
96     int type = redirectionType(args);
97
98     /*执行指定的命令*/
99     int operationCode = whichOperation(args);
100    if(operationCode != INSTRUCTION_WRONG)
101        add_history(tempCmd); //有效命令才能加入到history文件
    里面
102    switch (operationCode) {
103        case INSTRUCTION_WRONG:
104            printf("shell: command not found
    :%s\n", args[0]);
105            break;
106        case INSTRUCTION_CD:
107            cd(args);
108            break;
109        case INSTRUCTION_EXIT:
110            write_history(NULL);
111            return EXIT_STATUS_S;
112        case INSTRUCTION_HISTORY:
113            history();
114            break;

```

```

115         case INSTRUCTION_SYS_EXE:
116             pid = fork();
117             if(pid == 0) { /*命令执行子线程，必须要等待子线程执
行完毕父线程才退出*/
118                 if(type) { //重定向代码
119                     char *file = getRedirectionFile(args,
type);
120                     switch (type){
121                         case REDIRECTION_IN:
122                             fd = open(file,O_RDONLY);
123                             dup2(fd,STDIN);
124                             break;
125                         case REDIRECTION_OUT:
126                             fd = open(file,O_RDWR |
O_CREAT | O_TRUNC,0644);
127                             dup2(fd,STDOUT);
128                             break;
129                         case REDIRECTION_OUT_ADD:
130                             fd = open(file,O_RDWR |
O_CREAT | O_APPEND,0644); //O_TRUNC 和 O_APPEND 两种方式不能同时
使用
131                             dup2(fd,STDOUT);
132                             break;
133                         case REDIRECTION_PIPE:
134                             /*管道命令，先执行左边，将结果存放到临时
存放区，然后读取临时文件，执行右边*/
135                             flag=0,j=0;
136                             for(int i=0;flag!=2;i++) { //
这里没有错误判断，如果管道只输入一个变量会出现问题
137                                 if(flag == 0) {
138                                     argsFirst[j++] =
args[i];
139                                     if(args[i] == NULL) {
140                                         flag++;
141                                         j=0;
142                                     }
143                                 }else{
144                                     argsSecond[j++] =
args[i];
145                                     if(args[i] == NULL) {
146                                         flag++;
147                                     }
148                                 }
149                             }
150
151                             pid_pipe = fork();
152                             if(pid_pipe < 0) {
153                                 printf("failed to fork
new process!\n");
154                             }else if(pid_pipe == 0) { //子
进程执行左边部分

```



```

155         fd_pipe =
open("/tmp/tempfile",O_WRONLY|O_CREAT|O_TRUNC,0644); //创建临
时文件保存管道符前面的操作
156         dup2(fd_pipe,1);
157
execvp(argsFirst[0],argsFirst); //执行管道的前半部分
158
159         }else if(pid_pipe > 0) {
160             wait();
161         }
162         close(fd_pipe);
163         break;
164         default:
165             break;
166     }
167 }
168 if(type == REDIRECTION_PIPE) { //如果是管道
的代码执行的部分不同
169     wait(pid_pipe);
170     fd_pipe =
open("/tmp/tempfile",O_RDONLY);
171     dup2(fd_pipe,0);
172     execvp(argsSecond[0],argsSecond);
173     if(remove("/tmp/tempfile"))
174         printf("remove error\n");
175     close(fd_pipe);
176 }else {
177     execvp(args[0],args); //最后加一个NULL
作为参数的终结符号，参数就是完整指令，不能去掉开始的命令
178     if(type) {
179         close(fd);
180     }
181 }
182 }else if(pid > 0) {
183     wait(pid);
184 }else {
185     printf("创建线程失败! \n");
186 }
187 break;
188 default:
189     break;
190 }
191 free(prompt);
192 free(cmd);
193 }while(1);
194 }
195
196 /**
197  * @brief 输入命令预处理，定向符号和管道符号没有用空格分隔在对应符号前
后增减空格
198  * @note 还需要判断是不是出现多个重新定向的问题，双指针实现
199  * @param cmd:

```

```

200  * @retval
201  */
202  char* cmdPreprocess(char* cmd) {
203      int j=0,i=0;
204      char* temp = (char*)malloc(sizeof(char)*strlen(cmd)*2);
//原来的命令空间加倍
205      memset(temp,0,strlen(cmd)*2);
206      temp[0]='\0';
207      strcat(temp,cmd);
208      free(cmd);
209      cmd = temp;
210      for(i=0;cmd[i]!='\0';i++) {
211          if(cmd[i] == '<' || cmd[i] == '|') { //输入重定向只有一种情
况
212              if(i>0 && cmd[i-1]!=' ') { //定向符号前面没有空格，增
加一个空格
213                  for(j=strlen(cmd)+1;j>i;j--) {
214                      cmd[j] = cmd[j-1];
215                  }
216                  cmd[i] = ' ';
217                  i++;
218              }
219              if(cmd[i+1]!=' ') { //定向符号后面没有空格，增加一个空
格，如果是在字符串末尾也增加一个空格，在后面的trim函数中处理掉即可
220                  for(j=strlen(cmd)+1;j>i+1;j--) {
221                      cmd[j] = cmd[j-1];
222                  }
223                  cmd[i+1] = ' ';
224              }
225          }else if(cmd[i] == '>') { //输出重新定向有两种情况
226              if(cmd[i-1]!=' ' && cmd[i-1]!='>') { //>>两者前面非
空格处理方式相同
227                  for(j=strlen(cmd)+1;j>i;j--) {
228                      cmd[j] = cmd[j-1];
229                  }
230                  cmd[i] = ' ';
231              }
232              if(cmd[i+1]!=' ' && cmd[i+1]!='>') {
233                  for(j=strlen(cmd)+1;j>i+1;j--) {
234                      cmd[j] = cmd[j-1];
235                  }
236                  cmd[i+1] = ' ';
237              }
238          }
239      }
240      return cmd;
241  }
242
243  /**
244   * @brief 得到对应命令的INSTRUCTION_*数字，用来switch对应的操作
245   * @note
246   * @retval

```

```

247  */
248  int whichoperation(char**args) {
249      int redirection = redirectionType(args);
250      if(redirection)
251          return INSTRUCTION_SYS_EXE;
252      else if(strcmp(args[0], "cd")==0)
253          return INSTRUCTION_CD;
254      else if(strcmp(args[0], "exit")==0)
255          return INSTRUCTION_EXIT;
256      else if(strcmp(args[0], "history")==0)
257          return INSTRUCTION_HISTORY;
258      //这里判断指定的命令是否在/bin/目录下, 需要考虑一个问题: ./命令执行程序
259      else if(systemExe(args)==SYSTEM_EXE)
260          return INSTRUCTION_SYS_EXE;
261      else if(args[0][0]=='.' && args[0][1]=='/')
262          return INSTRUCTION_SYS_EXE;
263      else return INSTRUCTION_WRONG;
264  }
265
266  /**
267   * @brief 删除命令中多余的空格
268   * @note
269   * @param args: 输入命令字符串
270   * @retval None
271   */
272  void trim(char* args) {
273      int i,j=0;
274      for(i=0;args[i]!='\0';i++){
275          if (args[i]!=' '){
276              args[j++]=args[i];
277              if (args[i+1]==' ')
278                  args[j++] = args[++i];
279          }
280      }
281      if (args[j-1]==' ')
282          args[j-1]='\0';
283      args[j] = '\0';
284  }
285
286  /**
287   * @brief 将输入的以空格分隔的字符串存放到数组中
288   * @note 由于增加输入输出重定向的命令, 字符串分割需要增加一个定向符的判断
289   * @param args: 空格分隔的字符串
290   * @retval 二级指针, 存放每一个字符串的指针
291   */
292  char** getParameter(char* args) {
293      int count=0;
294      char* tempargs = args;
295      char* temp[10];
296      while (tempargs!=NULL)

```

```

297     temp[count++] = strsep(&tempargs, " ");
298     char**parameter = (char**) malloc(sizeof(char*)*
(count+1));
299     for(int i=0; i<count; i++) {
300         parameter[i] = temp[i];
301     }
302     parameter[count]=NULL;
303     return parameter;
304 }
305
306 /**
307  * @brief  shell cd命令
308  * @note
309  * @param  args: "cd path"命令解析的二级指针
310  * @retval 成功返回 0
311  */
312 int cd(char** args) {
313     char userhome[32]={"/home/"};
314     if(args[1] == NULL) {
315         fprintf(stderr, "sh: expected argument to \"cd\"\n");
316     }else {
317         if(strcmp(args[1], "~")==0) {
318             strcat(userhome, getUserName());
319             chdir(userhome);
320         }
321         else {
322             if(chdir(args[1])!=0 || args[2]!=NULL)
323                 perror("shell");
324         }
325     }
326     return 0;
327 }
328
329 /**
330  * @brief  shell内置命令，输出history信息
331  * @note
332  * @retval 正常退出返回0
333  */
334 int history() {
335     HIST_ENTRY ** history;
336     history=history_list();
337     for(int i=0; history[i]!=NULL; i++) {
338         printf("%s\n", history[i]->line);
339     }
340     return 0;
341 }
342
343 /**
344  * @brief  获取用户名字
345  * @note
346  * @retval
347  */

```

```

348 char * getUsername() {
349     uid_t uid = getuid();
350     struct passwd *pw = getpwuid (uid);
351     if(pw) {
352         return pw->pw_name;
353     }
354     return NULL;
355 }
356
357 /**
358  * @brief 判断输入的指令是否为系统可执行的命令
359  * @note
360  * @param args: 输入信息转换为的二级指针
361  * @retval 1:命令为系统可执行命令 0:系统不能执行的命令
362  */
363 int systemExe(char** args) {
364     DIR *d;
365     struct dirent *dir;
366     d = opendir("/bin/");
367     if (d) {
368         while ((dir = readdir(d)) != NULL) {
369             if(strcmp(args[0],dir->d_name)==0){ //待执行命令为系
系统命令
370                 closedir(d);
371                 return SYSTEM_EXE;
372             }
373         }
374         closedir(d);
375     }
376     return NOT_SYSTEM_EXE;
377 }
378
379 /**
380  * @brief 通过比较每一个参数，确定重定向的类型
381  * @note
382  * @param args:
383  * @retval
384  */
385 int redirectionType(char** args) {
386     for(int i=0;args[i]!=NULL;i++) {
387         for(int j=0;args[i][j]!='\0';j++){
388             switch (args[i][j]){
389                 case '<':
390                     return REDIRECTION_IN;
391                     break;
392                 case '>':
393                     if(args[i][j+1]=='>')
394                         return REDIRECTION_OUT_ADD;
395                     return REDIRECTION_OUT;
396                 case '|':
397                     return REDIRECTION_PIPE;
398             }

```

```

399     }
400 }
401     return REDIRECTION_NONE;
402 }
403
404 /**
405  * @brief 返回重定向的文件名
406  * @note 代码可以优化，直接比较字符串，不用单个字符比较，懒得改了
407  * @param args:
408  * @param type:
409  * @retval
410  */
411 char* getRedirectionFile(char** args, int type) {
412     for(int i=0;args[i]!=NULL;i++) {
413         if(strcmp(args[i],"<")==0||strcmp(args[i],"|")==0
414         ||strcmp(args[i],">")==0||strcmp(args[i],">>")==0) {
415             args[i]=NULL;
416             return args[i+1];
417         }
418     }
419 }
420 /**
421  * @brief 主函数
422  * @note
423  * @retval
424  */
425 int main() {
426     shell();
427     return 0;
428 }

```

参考文件

- [readline库的简单使用](#)
- [shell输出改变字体颜色](#)
- [shell的自己实现](#)