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`exevp(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

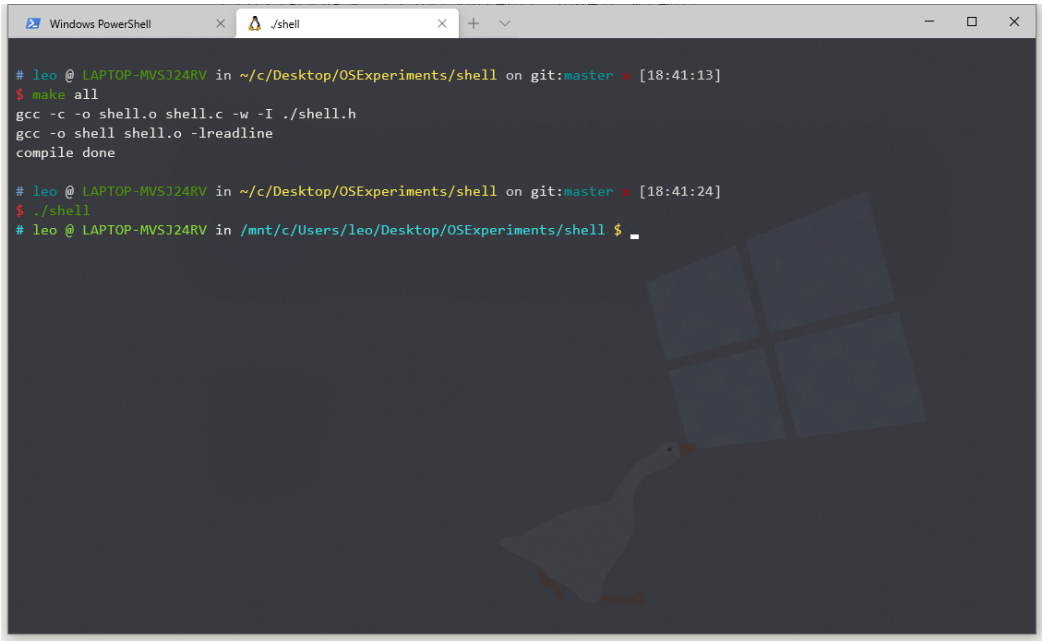
```
shell/
├── Makefile    #make文件
├── README.md   #本文件
├── shell       #编译成功的可执行文件
├── shell.c     #shell源代码
└── shell.h     #shell头文件
```

编译运行

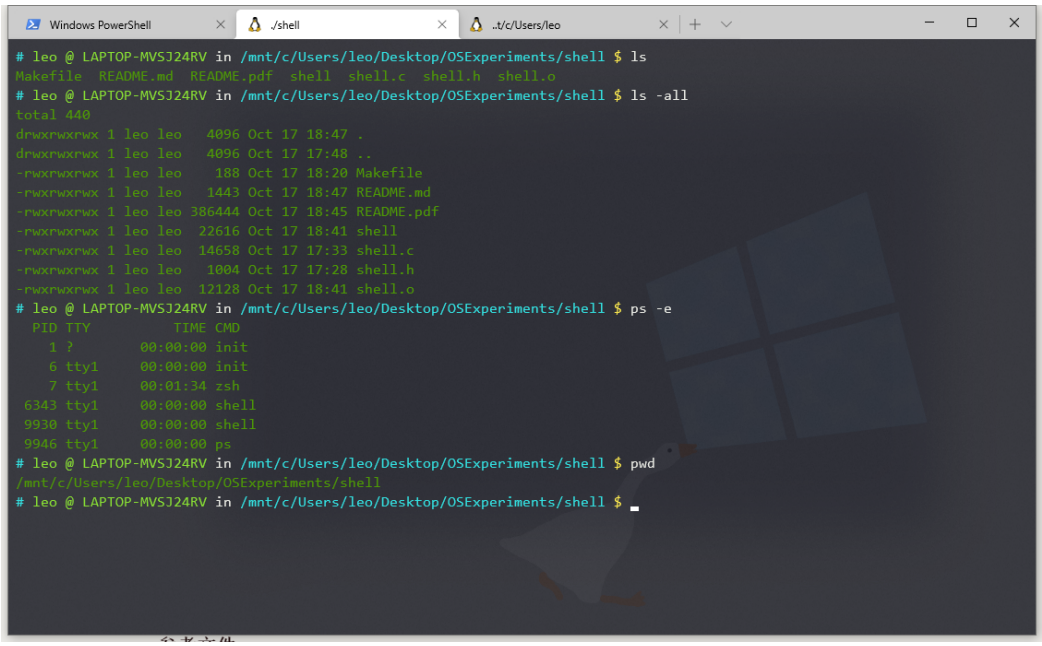
- `sudo apt-get install libreadline-dev` 安装`readline`库
- `make all` 编译
- `./shell` 运行
- `make clen` 清除编译产生的临时文件

运行效果

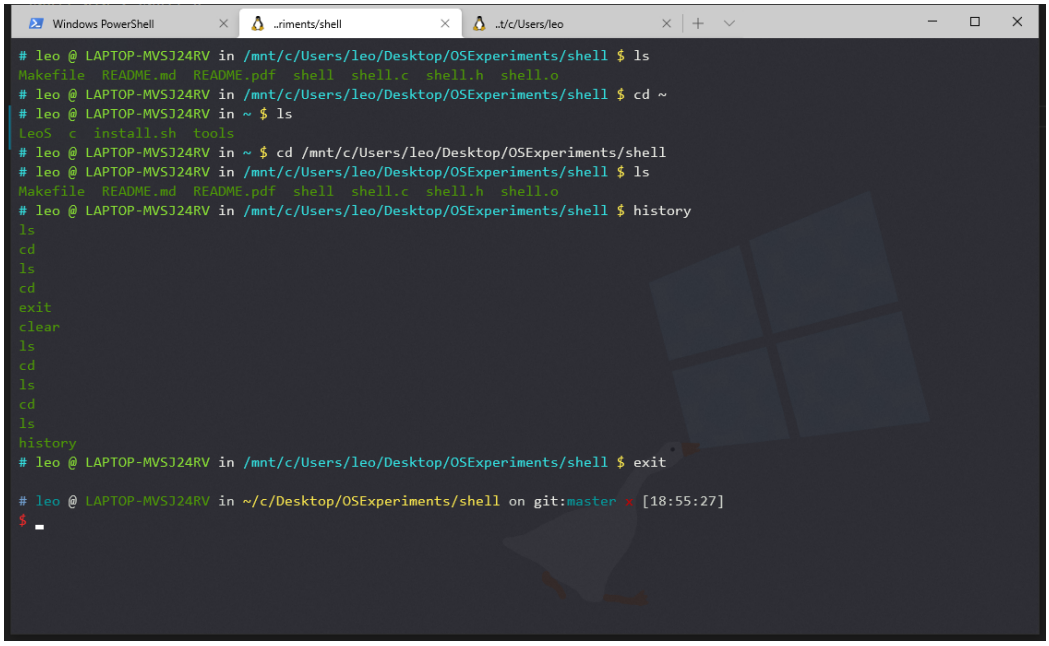
编译



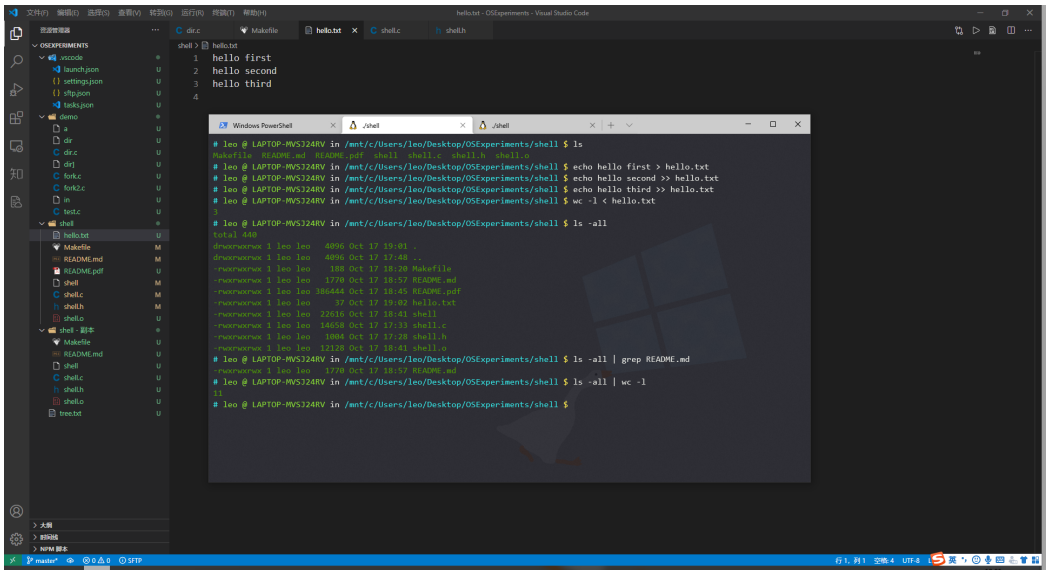
系统功能



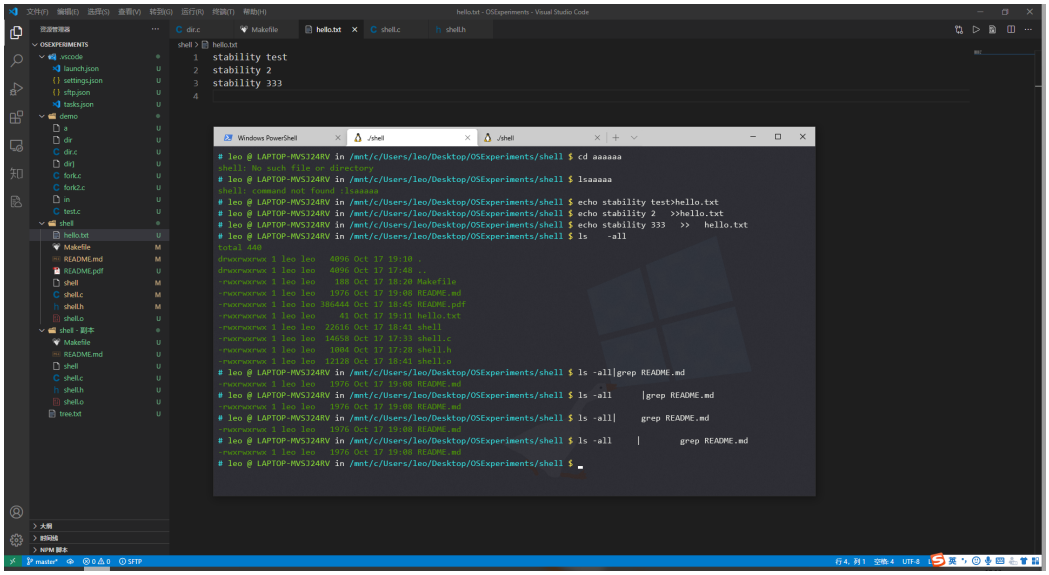
shell内置命令



管道 输入输出重定向



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
```

```
#define INSTRUCTION_SYS_EXE 4

#define SYSTEM_EXE 1
#define NOT_SYSTEM_EXE 0

#define REDIRECTION_IN 1
#define REDIRECTION_OUT 2
#define REDIRECTION_OUT_ADD 3
#define REDIRECTION_PIPE 4
#define REDIRECTION_NONE 0

#define STDIN 0
#define STDOUT 1
#define STDERR 2

char ** getParameter(char* args);
int shell();
void trim(char* args);
int cd(char** args);
char * getUserName();
int systemExe(char** args);
int whichOperation(char**args);
int redirectionType(char** args);
char* getRedirectionFile(char** args, int type);
char* cmdPreprocess(char* cmd);
#endif //
```

shell.c

```
#include "shell.h"
#define EXIT_STATUS_S 1


extern void add_history PARAMS((const char *));

/* Add the contents of FILENAME to the history list, a line at a time.
   If FILENAME is NULL, then read from ~/.history.  Returns 0 if
   successful, or errno if not. */
extern int read_history PARAMS((const char *));

/* Write the current history to FILENAME.  If FILENAME is NULL,
   then write the history list to ~/.history.  Values returned
   are as in read_history ().  */
extern int write_history PARAMS((const char *));

/* Return a NULL terminated array of HIST_ENTRY which is the current input history.
```

```
Element 0 of this list is the beginning of time.  If there is no history, return NULL.*/
extern HIST_ENTRY **history_list PARAMS((void));

/**
 * @brief  shell程序
 * @note
 * @retval
 */
int shell() {
    pid_t pid,pid_pipe;
    int fd,fd_pipe; //文件描述符
    int flag=0,j=0; //管道判断参数是否完整

    char*cmd; //读取用户输入
    char*prompt;
    char hostname[40]={" @ LAPTOP-MVSJ24RV "};
    char** args;
    char* username;
    int printfReturn=0;
    char userhome[32]={"/home/"};
    char tempCmd[40];

    char* argsFirst[10];
    char* argsSecond[10];

    char file_path_getcwd[80];

    /*输出配色方案*/
    char colorUser[50] = {"\001\033[1;32m\002"}; //绿色高亮显示用户名字 hostname
    char colorDirectory[50] = {"\001\033[1;36m\002"};//紫色显示当前文件目录
    char colorReset[50] = {"\001\033[0m\002"}; //显示恢复默认值
    char colorShow[50] = {"\033[5;32m"};

    /*从~/.history读出，之前每次放在循环里面，
    因此每次都会让history的list长度加倍*/
    read_history(NULL);
    do{
        /*获取当前路径*/
        getcwd(file_path_getcwd,80);

        /*获取当前用户*/
        username = getUserName();

        //当前路径是用户目录，显示~
        strcat(userhome,username);
        if(strcmp(file_path_getcwd,userhome)==0) {
            strcpy(file_path_getcwd,"~");
        }
        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
```

```
                                    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
                                    dup2(fd_pipe,1);

 execvp(argsFirst[0],argsFirst); //执行管道的前半部分

                                }else if(pid_pipe > 0) {
                                    wait();
                                }
                                close(fd_pipe);
                                break;
                            default:
                                break;
                        }
                    }
                    if(type == REDIRECTION_PIPE) { //如果是管道的代码执行的部分不同
                        wait(pid_pipe);
                        fd_pipe = open("/tmp/tempfile",O_RDONLY);
                        dup2(fd_pipe,0);
                        execvp(argsSecond[0],argsSecond);
                        if(remove("/tmp/tempfile"))
                            printf("remove error\n");
                        close(fd_pipe);
                    }else {
                        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
                        if(type) {
                            close(fd);
                        }
                    }
                }else if(pid > 0) {
                    wait(pid);
                }else {
                    printf("创建线程失败！\n");
                }
                break;
            default:
                break;
        }
        free(prompt);
        free(cmd);
    }while(1);
}

/**
 * @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
 * @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
 * @param  cmd:
```

```
 * @retval
 */
char* cmdPreprocess(char* cmd) {
    int j=0,i=0;
    char* temp = (char*)malloc(sizeof(char)*strlen(cmd)*2); //原来的命令空间加倍
    memset(temp,0,strlen(cmd)*2);
    temp[0]='\0';
    strcat(temp,cmd);
    free(cmd);
    cmd = temp;
    for(i=0;cmd[i]!='\0';i++) {
        if(cmd[i] == '<'||cmd[i] == '|') { //输入重定向只有一种情况
            if(i>0 && cmd[i-1]!=' ') { //定向符号前面没有空格，增加一个空格
                for(j=strlen(cmd)+1;j>i;j--) {
                    cmd[j] = cmd[j-1];
                }
                cmd[i] = ' ';
                i++;
            }
            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
                for(j=strlen(cmd)+1;j>i+1;j--) {
                    cmd[j] = cmd[j-1];
                }
                cmd[i+1] = ' ';
            }
        }else if(cmd[i] == '>') { //输出重新定向有两种情况
            if(cmd[i-1]!=' '&&cmd[i-1]!='>') {//> >>两者前面非空格处理方式相同
                for(j=strlen(cmd)+1;j>i;j--) {
                    cmd[j] = cmd[j-1];
                }
                cmd[i] = ' ';
            }
            if(cmd[i+1]!=' '&&cmd[i+1]!='>') {
                for(j=strlen(cmd)+1;j>i+1;j--) {
                    cmd[j] = cmd[j-1];
                }
                cmd[i+1] = ' ';
            }
        }
    }
    return cmd;
}

/**
 * @brief  得到对应命令的INSTRUCTION_*数字，用来switch对应的操作
 * @note
 * @retval
```

```
 */
int whichOperation(char**args) {
    int redirection = redirectionType(args);
    if(redirection)
        return INSTRUCTION_SYS_EXE;
    else if(strcmp(args[0],"cd")==0)
        return INSTRUCTION_CD;
    else if(strcmp(args[0],"exit")==0)
        return INSTRUCTION_EXIT;
    else if(strcmp(args[0],"history")==0)
        return INSTRUCTION_HISTORY;
    //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
    else if(systemExe(args)==SYSTEM_EXE)
        return INSTRUCTION_SYS_EXE;
    else if(args[0][0]=='.' && args[0][1]=='/')
        return INSTRUCTION_SYS_EXE;
    else return INSTRUCTION_WRONG;
}

/**
 * @brief  删除命令中多余的空格
 * @note
 * @param  args: 输入命令字符串
 * @retval None
 */
void trim(char* args) {
    int i,j=0;
    for(i=0;args[i]!='\0';i++){
        if (args[i]!=' '){
            args[j++]=args[i];
            if (args[i+1]==' ')
                args[j++] = args[++i];
        }
    }
    if (args[j-1]==' ')
        args[j-1]='\0';
    args[j] = '\0';
}

/**
 * @brief  将输入的以空格分隔的字符串存放到数组中
 * @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
 * @param  args: 空格分隔的字符串
 * @retval 二级指针，存放每一个字符串的指针
 */
char** getParameter(char* args) {
    int count=0;
    char* tempargs = args;
    char* temp[10];
    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  */
```

```c
char * getUserName() {
    uid_t uid = getuid();
    struct passwd *pw = getpwuid (uid);
    if(pw) {
        return pw->pw_name;
    }
    return NULL;
}

/**
 * @brief  判断输入的指令是否为系统可执行的命令
 * @note
 * @param  args: 输入信息转换为的二级指针
 * @retval 1:命令为系统可执行命令 0:系统不能执行的命令
 */
int systemExe(char** args) {
    DIR *d;
    struct dirent *dir;
    d = opendir("/bin/");
    if (d) {
        while ((dir = readdir(d)) != NULL) {
            if(strcmp(args[0],dir->d_name)==0){ //待执行命令为系统命令
                closedir(d);
                return SYSTEM_EXE;
            }
        }
        closedir(d);
    }
    return NOT_SYSTEM_EXE;
}

/**
 * @brief  通过比较每一个参数，确定重定向的类型
 * @note
 * @param  args:
 * @retval
 */
int redirectionType(char** args) {
    for(int i=0;args[i]!=NULL;i++) {
        for(int j=0;args[i][j]!='\0';j++){
            switch (args[i][j]){
                case '<':
                    return REDIRECTION_IN;
                    break;
                case '>':
                    if(args[i][j+1]=='>')
                        return REDIRECTION_OUT_ADD;
                    return REDIRECTION_OUT;
                case '|':
                    return REDIRECTION_PIPE;
            }
```

```
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
    ||strcmp(args[i],">")==0||strcmp(args[i],">>")==0) {
414             args[i]=NULL;
415             return args[i+1];
416         }
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