计算机学院 操作系统 课程实验报告

实验题目：		学号：202300130183
日期：2025/	班级： 23 级智能班	姓名：宋浩宇
Email：202300130183@mail.sdu.edu.cn		

实验方法介绍：
使用 Oracle Virtual Box 运行 Ubuntu24.04 虚拟环境来编写编译相应的代码。

实验过程描述：
首先编写指令解释器的部分。这包括了指令拆分，参数列表获取，重定向识别，后台运行识别，临时管道识别。然后是编写指令执行器的部分，当我们完成将指令拆解为程序可以理解的 token 之后，先建立指令对应的子进程，如果有临时管道则先建立管道，并建立多个子进程。再按照是否有临时管道、是否有重定向、是否在后台执行设置好子线程的各项参数,然后在使用系统调用 execvp 传入程序名称和参数列表。然后是进行历史记录处理，因为标准终端需要按下 enter 才会将内容输入到 stdin,因此我们需要先更改终端设置,让我们可以不使用 enter 就让程序读取到输入，并且我们还需要关闭回显，并维护一个命令行缓冲区，在这个缓冲区里根据按键加载内容，并显示在终端中。当我们按下 enter，缓冲区中的指令部分会被传给指令解释器,并记录到历史记录中,历史记录使我们程序维护的一个二维字符数组,当我们按下上下键时，历史记录里的内容会加载到行缓冲区中并显示在命令行中。还有我们通过绑定 sigint 的处理函数,并在这个处理函数里边判断是不是子进程来决定是否终止进程。用户输入的非法命令我们则使用 perror 调用系统错误信息。以上就是我们整个的解决方案，具体实现方式请看代码：

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <stdlib.h>
#include <signal.h>
#include <string.h>
#include <stdbool.h>
#include <fcntl.h>
#include <termios.h>
#include <ctype.h>
#define MAX_CMD_LEN 100
#define MAX_PIPE_CNT 100
int cpid = -1;
int background;
int running_at_background;
int as_son = 0;
```

```
#define HISTORY_SIZE 100
char history[HISTORY_SIZE][MAX_CMD_LEN];
int history_index = -1;
int history_cnt = 0;
char line_buffer[MAX_CMD_LEN];
int line_buffer_index = 0;
int line_buffer_cursor = 0;
void cmd_prompt()
{
    printf("\033[2K\rmyShellCommand> ");
    strcpy(line_buffer, "myShellCommand> ");
    line_buffer_index = strlen(line_buffer);
    line_buffer_cursor = line_buffer_index + 1;
    printf("\033[%dG", line_buffer_cursor);
    fflush(stdout);
}
void history_down()
{
    if (history_index == -1)
    {
        return;
    }
    else
    {
        history_index--;
    }
}
void history_up()
{
    if (history_index < history_cnt - 1)
    {
        history_index++;
        fflush(stdout);
    }
    else
    {
        return;
    }
}
void add_history(char* cmd)
{
    history_cnt++;
    // printf("\n%s", cmd);
    if (history_cnt > HISTORY_SIZE)
    {
        history_cnt = HISTORY_SIZE;
    }

    for (int i = history_cnt - 1; i > 0; i--)
    {
        strcpy(history[i], history[i - 1]);
    }
    strcpy(history[0], cmd);
}
void load_history()
{
    if (history_index == -1)
    {
        cmd_prompt();
        return;
    }
    cmd_prompt();
    printf("%s", history[history_index]);
    strcat(line_buffer, history[history_index]);
    line_buffer_index = strlen(line_buffer);
    line_buffer_cursor = line_buffer_index + 1;
    printf("\033[%dG", line_buffer_cursor);
    fflush(stdout);
}
static struct termios orig_termios;
void restore_terminal(void)
{
    tcsetattr(STDIN_FILENO, TCSAFLUSH, &orig_termios);
}
void enable_raw_mode(void)
{
    struct termios raw;
    tcgetattr(STDIN_FILENO, &orig_termios);
    atexit(restore_terminal);
    raw = orig_termios;
    raw.c_lflag &= ~(ICANON | ECHO);
    raw.c_cc[VMIN] = 0;
    raw.c_cc[VTIME] = 1;
    tcsetattr(STDIN_FILENO, TCSAFLUSH, &raw);
}
void sigint_handler(int sig)
{
    if (as_son == 1)
    {
        exit(1);
    }

    if (cpid == 0 && running_at_background == 0)
    {
        exit(1);
    }
    else
    {
        printf("\n");
        cmd_prompt();
    }
}
```

```
char* divide_cmd(char** cmd)
{
    if (**cmd == '\n' || **cmd == '\0' || **cmd == '|' || **cmd ==
'>' || **cmd == '<')
    {
        return NULL;
    }
    while (**cmd == ' ')
    {
        cmd++;
    }
    int len = 0;
    bool mark;
    while ((*cmd)[len] != ' ' && (*cmd)[len] != '\n' && (*cmd)[len] !=
'\0'&&(*cmd)[len]!='|'&&(*cmd)[len]!='>'&&(*cmd)[len]!='<')
    {
        len++;
    }
    char* cmd_ptr = malloc(len + 1);
    strncpy(cmd_ptr, *cmd, len);
    cmd_ptr[len] = '\0';
    while ((*cmd)[len] == ' ')
    {
        (*cmd) = (*cmd) + 1;
    }
    (*cmd) = (*cmd) + len;
    return cmd_ptr;
}
```

```c
char* divide_pipe(char** cmd)
{
    if (**cmd == '\n' || **cmd == '\0')
    {
        return NULL;
    }
    int len = 0;
    char* temp_char_ptr = *cmd;
    while ((*cmd)[len] != '|' && (*cmd)[len] != '\0' && (*cmd)[len] !=
'\n')
    {
        len++;
    }
    char* cmd_ptr = malloc(len + 1);
    strncpy(cmd_ptr, temp_char_ptr, len);
    cmd_ptr[len] = '\0';
    (*cmd) = (*cmd) + len;
    while (**cmd == '|' || **cmd == ' ')
    {
        (*cmd) = (*cmd) + 1;
    }
    return cmd_ptr;
}
char* divide_input(char** cmd)
{
    if (**cmd != '<')
    {
        return NULL;
    }
    while (**cmd == ' ' || **cmd =='<')
    {
        (*cmd) = (*cmd) + 1;
    }
    int len = 0;
    char* temp_char_ptr = *cmd;
    while ((*cmd)[len] != ' ' && (*cmd)[len] != '\0' && (*cmd)[len] !=
'\n')
    {
        len++;
    }
    char* cmd_ptr = malloc(len + 1);
    strncpy(cmd_ptr, temp_char_ptr, len);
    cmd_ptr[len] = '\0';
    return cmd_ptr;
}
char* divide_output(char** cmd)
{
    if (**cmd != '>')
    {
        return NULL;
    }
    while (**cmd == ' ' || **cmd =='>')
    {
        (*cmd) = (*cmd) + 1;
    }
    int len = 0;
    char* temp_char_ptr = *cmd;
    while ((*cmd)[len] != ' ' && (*cmd)[len] != '\0' && (*cmd)[len] !=
'\n')
    {
        len++;
    }
    char* cmd_ptr = malloc(len + 1);
    strncpy(cmd_ptr, temp_char_ptr, len);
    cmd_ptr[len] = '\0';
    return cmd_ptr;
}
int run_cmd(char* cmd)
{
    int status;
    char* cmd_ptr = cmd;
    if (!strcmp(cmd, "exit\n"))
    {
        restore_terminal();
        exit(0);
    }
    if (!strcmp(cmd, "exit"))
    {
        restore_terminal();
        exit(0);
    }

    if (!strcmp(cmd, "clear\n"))
    {
        printf("\033c");
        return 0;
    }
```

```c
    if (strchr(cmd, '&') != NULL)
    {
        background = 1;
    }
    char* input_ptr = strchr(cmd, '<');
    char* input_stream;
    char* output_ptr = strchr(cmd, '>');
    char* output_stream;
    char** argv;
    char* program;
    cmd_ptr = cmd;
    program = divide_cmd(&cmd_ptr);
    char* temp_char_ptr = cmd_ptr;
    int argv_cnt = 0;
    while (1)
    {
        char* tempp = divide_cmd(&cmd_ptr);
        if (tempp == NULL)
        {
            break;
        }
        argv_cnt++;
        free(tempp);
    }
    argv = malloc((argv_cnt + 2) * sizeof(char*));
    for (int i = 0; i < argv_cnt; i++)
    {
        argv[i + 1] = divide_cmd(&temp_char_ptr);
    }
    argv[0] = program;
    argv[argv_cnt + 1] = NULL;
    // printf("argv_cnt:%d\n", argv_cnt);
    // printf("program:%s\n", program);
    // printf("argv:\n");
    // for (int i = 0; i < argv_cnt; i++)
    // {
    //     printf("argv%d:%s\n", i, argv[i]);
    // }
    cpid = fork();
    if (cpid == 0)
    {
        as_son = 1;
        if (output_ptr)
        {
            output_stream = divide_output(&output_ptr);
            // printf("output_stream:%s\n", output_stream);
            fflush(stdout);
        }
        if (input_ptr)
        {
            input_stream = divide_input(&input_ptr);
            // printf("input_stream:%s\n", input_stream);
            fflush(stdout);
        }
        if (background == 1)
        {
            running_at_background = 1;
        }
        if (input_ptr)
        {
            int fd = open(input_stream, O_RDONLY);
            close(0);
            dup(fd);
            close(fd);
        }
        if (output_ptr)
        {
            int fd = open(output_stream, O_WRONLY | O_CREAT | O_TRUNC,
0666);
            close(1);
            dup(fd);
            close(fd);
        }
        execvp(program, argv);
        perror("");
        exit(0);
    }
    else
    {
        if (background == 0)
        {
            waitpid(cpid, &status, 0);
            for (int i = 0; i < argv_cnt; i++)
            {
                free(argv[i + 1]);
            }
            free(argv);
            free(program);
```

```c
            // printf("\n");
            fflush(stdout);
            if (as_son)
            {
                exit(0);
            }
        }
    }
    return 0;
}
int piped_run(char* cmd,int pipe_write,int pipe_read)
{
    int status;
    char* cmd_ptr = cmd;
    if (!strcmp(cmd, "exit\n"))
    {
        exit(0);
    }
    if (strchr(cmd, '&') != NULL)
    {
        background = 1;
    }
    char* input_ptr = strchr(cmd, '<');
    char* input_stream;
    char* output_ptr = strchr(cmd, '>');
    char* output_stream;
    char** argv;
    char* program;
    cmd_ptr = cmd;
    program = divide_cmd(&cmd_ptr);
    char* temp_char_ptr = cmd_ptr;
    int argv_cnt = 0;
    while (1)
    {
        char* tempp = divide_cmd(&cmd_ptr);
        if (tempp == NULL)
        {
            break;
        }
        argv_cnt++;
        free(tempp);
    }
    argv = malloc((argv_cnt + 2) * sizeof(char*));
    for (int i = 0; i < argv_cnt; i++)
    {
        argv[i + 1] = divide_cmd(&temp_char_ptr);
    }
    argv[0] = program;
    argv[argv_cnt + 1] = NULL;
    // printf("argv_cnt:%d\n", argv_cnt);
    // printf("program:%s\n", program);
    // printf("argv:\n");
    // for (int i = 0; i < argv_cnt; i++)
    // {
    //     printf("argv%d:%s\n", i, argv[i]);
    // }
    cpid = fork();
    if (cpid == 0)
    {
        as_son = 1;
        if (output_ptr)
        {
            output_stream = divide_output(&output_ptr);
            // printf("output_stream:%s\n", output_stream);
            fflush(stdout);
        }
        if (input_ptr)
        {
            input_stream = divide_input(&input_ptr);
            // printf("input_stream:%s\n", input_stream);
            fflush(stdout);
        }
        if (pipe_write != 0)
        {
            dup2(pipe_write, STDOUT_FILENO);
        }
        if (pipe_read != 0)
        {
            dup2(pipe_read, STDIN_FILENO);
        }
        if (background == 1)
        {
            running_at_background = 1;
        }
        if (input_ptr)
        {
            int fd = open(input_stream, O_RDONLY);
            close(0);
            dup(fd);
            close(fd);
        }
        if (output_ptr)
        {
            int fd = open(output_stream, O_WRONLY | O_CREAT | O_TRUNC,
0666);
            close(1);
            dup(fd);
            close(fd);
        }
        execvp(program, argv);
        perror("");

        exit(0);
    }
    else
    {
        if (background == 0)
        {
            waitpid(cpid, &status, 0);
            if (pipe_write != 0)
            {
                close(pipe_write);
            }
            if (pipe_read != 0)
            {
                close(pipe_read);
            }
            free(argv);
            free(program);
            // printf("\n");
            if (as_son)
            {
                exit(0);
            }
            fflush(stdout);
        }
    }
    // for (int i = 0; i < argv_cnt; i++)
    // {
    //     free(argv[i + 1]);
    // }
    return 0;
```

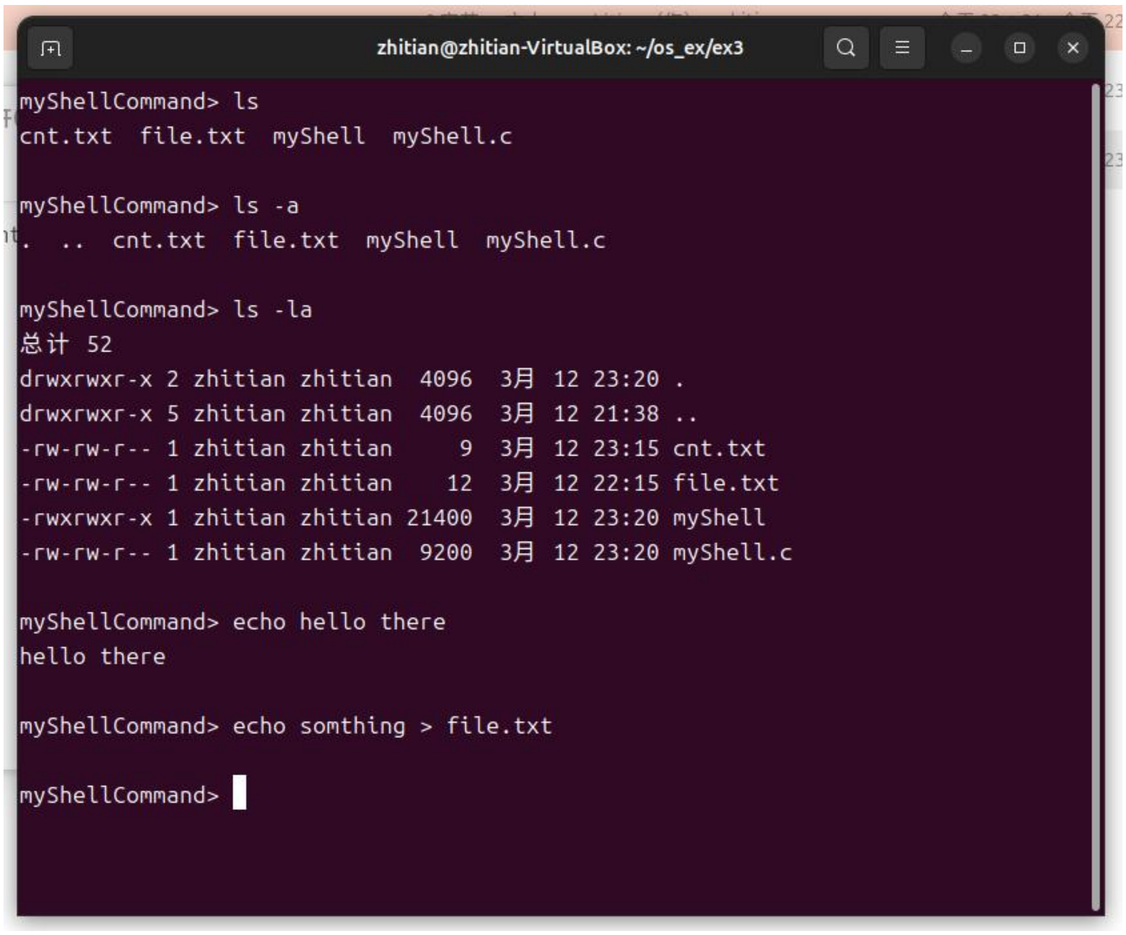
```
}
int when_need_pipe(char* cmd)
{
    int cmd_cnt = 0;
    char* cmd_ptr = cmd;
    char** cmds = malloc(MAX_PIPE_CNT * sizeof(char*));
    if (strchr(cmd, '|') != NULL)
    {
        while (1)
        {
            char* temp = divide_pipe(&cmd_ptr);
            if (temp == NULL)
            {
                break;
            }
            cmds[cmd_cnt++] = temp;
        }
        // printf("cmds:\n");
        // for (int j = 0; j < cmd_cnt; j++)
        // {
        //     printf("cmd%d:%s\n", j, cmds[j]);
        // }
        fflush(stdout);
    }
```

```
    int** pipe_fd = malloc(cmd_cnt * sizeof(int*));
    for (int i = 0; i < cmd_cnt; i++)
    {
        pipe_fd[i] = malloc(2 * sizeof(int));
        pipe(pipe_fd[i]);
    }
    for (int i = 0; i < cmd_cnt; i++)
    {
        if (i == 0)
        {
            piped_run(cmds[i], pipe_fd[i][1], 0);
        }
        else if (i < cmd_cnt - 1)
        {
            piped_run(cmds[i], pipe_fd[i][1], pipe_fd[i - 1][0]);
        }
        else if (i == cmd_cnt - 1)
        {
            piped_run(cmds[i], 0, pipe_fd[i - 1][0]);
        }
    }
}
int run()
{
    struct sigaction SIGINT_ACT;
    SIGINT_ACT.sa_handler = sigint_handler;
    sigemptyset(&SIGINT_ACT.sa_mask);
    SIGINT_ACT.sa_flags = 0;
    sigaction(SIGINT, &SIGINT_ACT, NULL);
    printf("\033c");
    enable_raw_mode();
    cmd_prompt();
    while (1)
    {
        char c = 0;
        // enable_raw_mode();
        if (read(STDIN_FILENO, &c, 1) == -1)
        {
            continue;
        }
        if (c == '\x1B')
        {
            char seq[2];
            if (read(STDIN_FILENO, &seq[0], 1) != 1)
            {
                continue;
            }
            if (read(STDIN_FILENO, &seq[1], 1) != 1)
            {
                continue;
            }
            if (seq[0] == '[')
            {
                switch (seq[1])
                {
                case 'A':
                    history_up();
                    load_history();
                    break;
                case 'B':
                    history_down();
                    load_history();
                    break;
                }
            }
        }
        else if (c == '\n')
        {
            history_index = -1;
            restore_terminal();
            char cmd[MAX_CMD_LEN];
            strcpy(cmd, line_buffer + 16);
            line_buffer_index = 0;
            line_buffer_cursor = 0;
            add_history(cmd);
            background = 0;
            printf("\n");
            fflush(stdout);
            if (strchr(cmd, '|'))
            {
                when_need_pipe(cmd);
            }
            else
            {
                run_cmd(cmd);
            }
            enable_raw_mode();
            cmd_prompt();
            if (fileno(stdin) != 0)
            {
                printf("%d",fileno(stdin));
                exit(EXIT_SUCCESS);
            }
        }
        else if (c >= 32 && c <= 126)
        {
            history_index = -1;
            putc(c, stdout);
            line_buffer[line_buffer_index++] = c;
            line_buffer[line_buffer_index] = '\0';
            line_buffer_cursor = line_buffer_index + 1;
            if (line_buffer_index == MAX_CMD_LEN - 1)
            {
                line_buffer_index = 0;
            }
            fflush(stdout);
        }
        else if (c == 127 || c == 8)
        {
            if (line_buffer_cursor > strlen("myShellCommand> ") + 1)
            {
                line_buffer_cursor--;
                line_buffer[line_buffer_cursor] = '\0';
                line_buffer_index--;
                printf("\010 \010");
                fflush(stdout);
                history_index = -1;
            }
        }
        else if (c == -1)
        {
            exit(0);
        }
    }
}
int main(void)
{
    run();
    return 0;
}
```

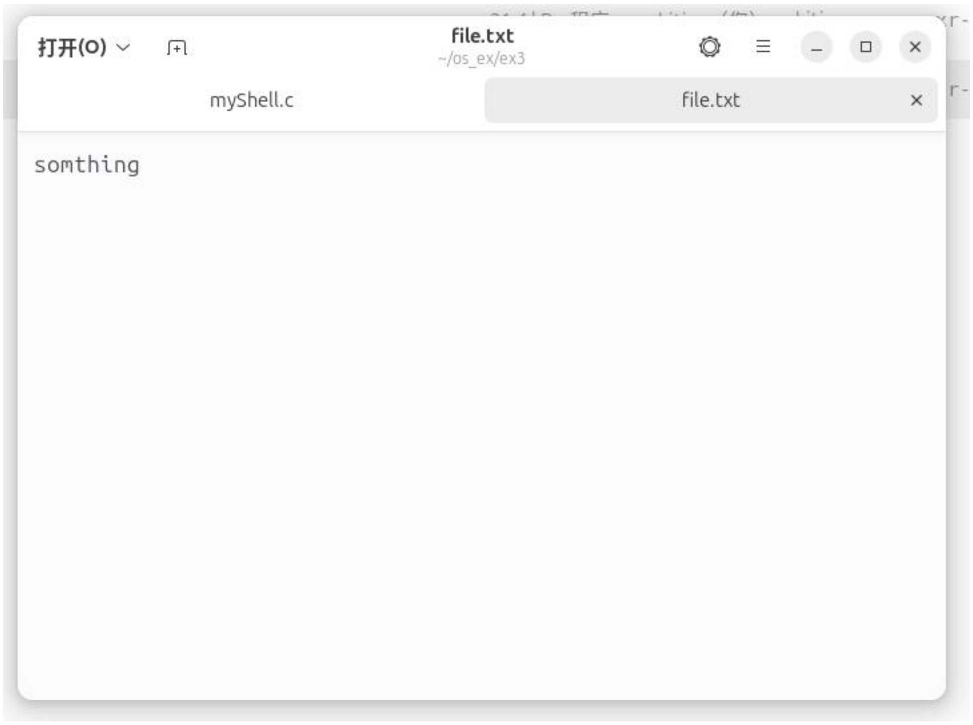
结论分析：

首先是几个实验指导书给的测试命令。

命令 clear 清屏

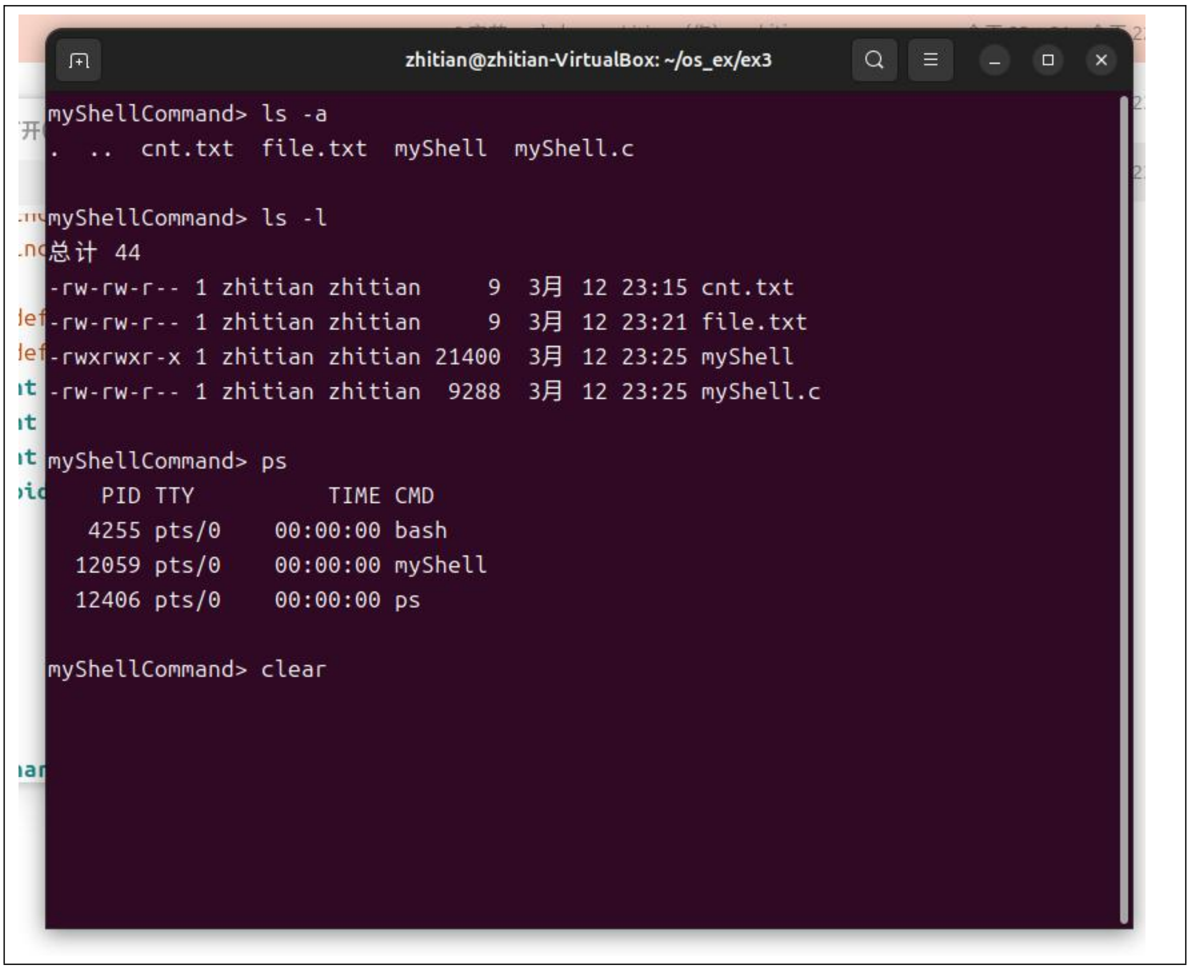


```
myShellCommand> ls
cnt.txt  file.txt  myShell  myShell.c

myShellCommand> ls -a
.  ..  cnt.txt  file.txt  myShell  myShell.c

myShellCommand> ls -la
总计 52
drwxrwxr-x 2 zhitian zhitian  4096  3月  12 23:20 .
drwxrwxr-x 5 zhitian zhitian  4096  3月  12 21:38 ..
-rw-rw-r-- 1 zhitian zhitian     9  3月  12 23:15 cnt.txt
-rw-rw-r-- 1 zhitian zhitian    12  3月  12 22:15 file.txt
-rwxrwxr-x 1 zhitian zhitian 21400  3月  12 23:20 myShell
-rw-rw-r-- 1 zhitian zhitian  9200  3月  12 23:20 myShell.c

myShellCommand> echo hello there
hello there

myShellCommand> echo somthing > file.txt

myShellCommand> 
```

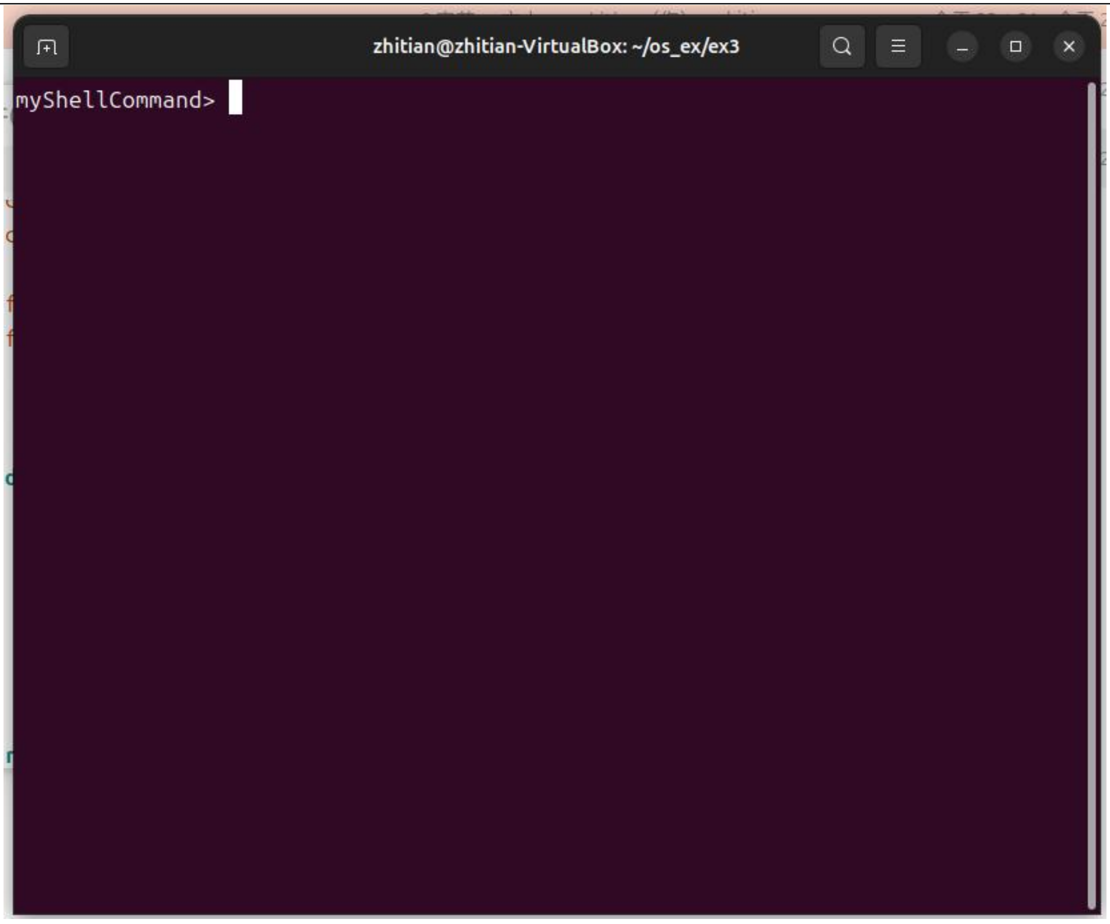


```
somthing
```

命令 clear 清屏

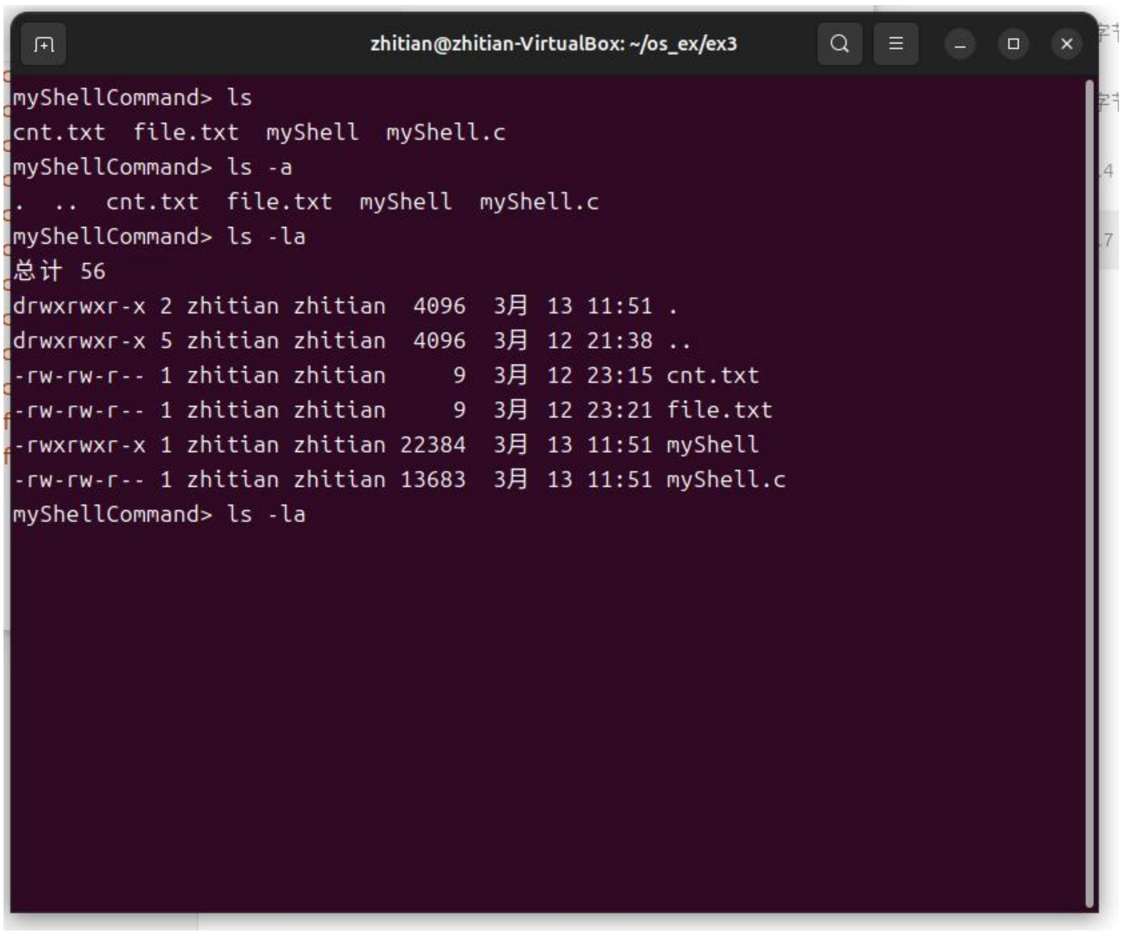


```
myShellCommand> ls -a
.  ..  cnt.txt  file.txt  myShell  myShell.c

myShellCommand> ls -l
总计 44
-rw-rw-r-- 1 zhitian zhitian     9  3月 12 23:15 cnt.txt
-rw-rw-r-- 1 zhitian zhitian     9  3月 12 23:21 file.txt
-rwxrwxr-x 1 zhitian zhitian 21400  3月 12 23:25 myShell
-rw-rw-r-- 1 zhitian zhitian  9288  3月 12 23:25 myShell.c

myShellCommand> ps
    PID TTY          TIME CMD
   4255 pts/0    00:00:00 bash
  12059 pts/0    00:00:00 myShell
  12406 pts/0    00:00:00 ps

myShellCommand> clear
```



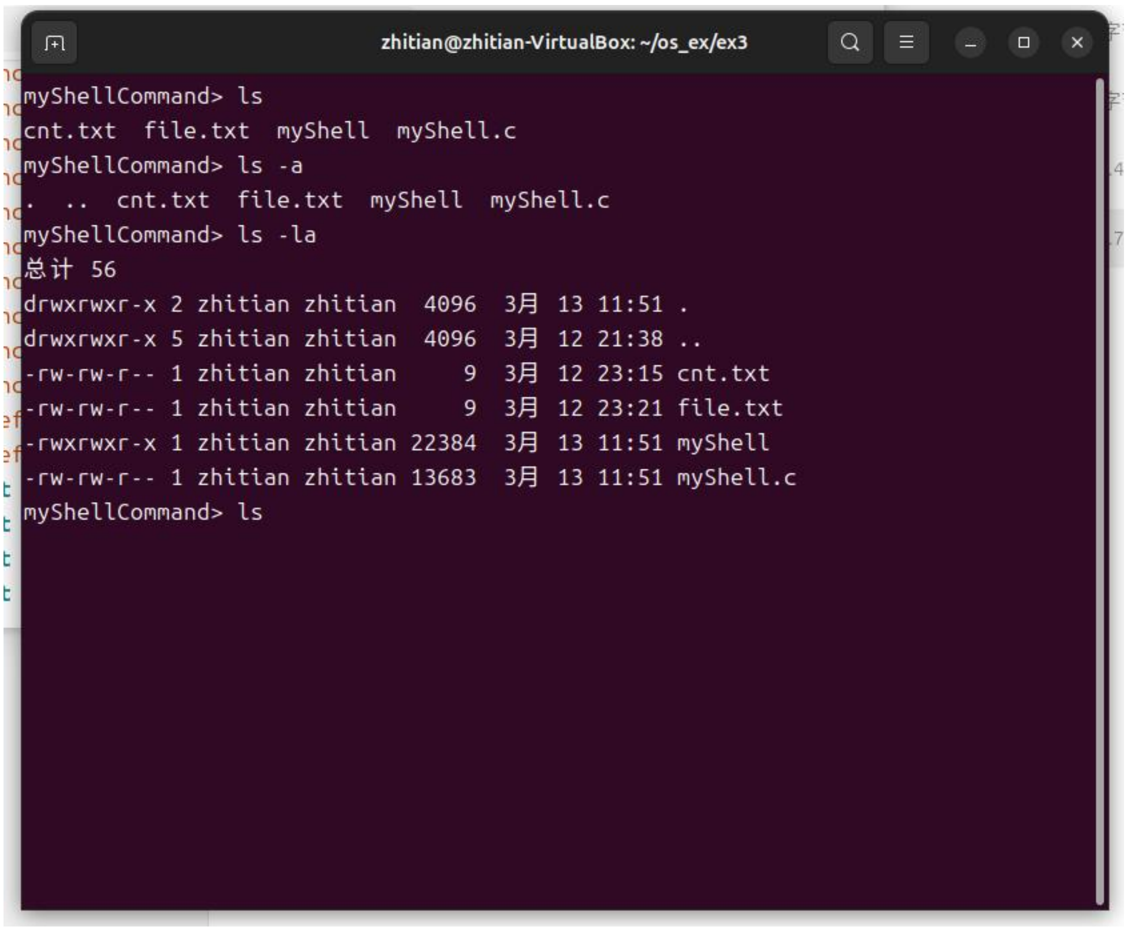
```
myShellCommand> 
```

然后是历史记录部分
按一次上键之后：



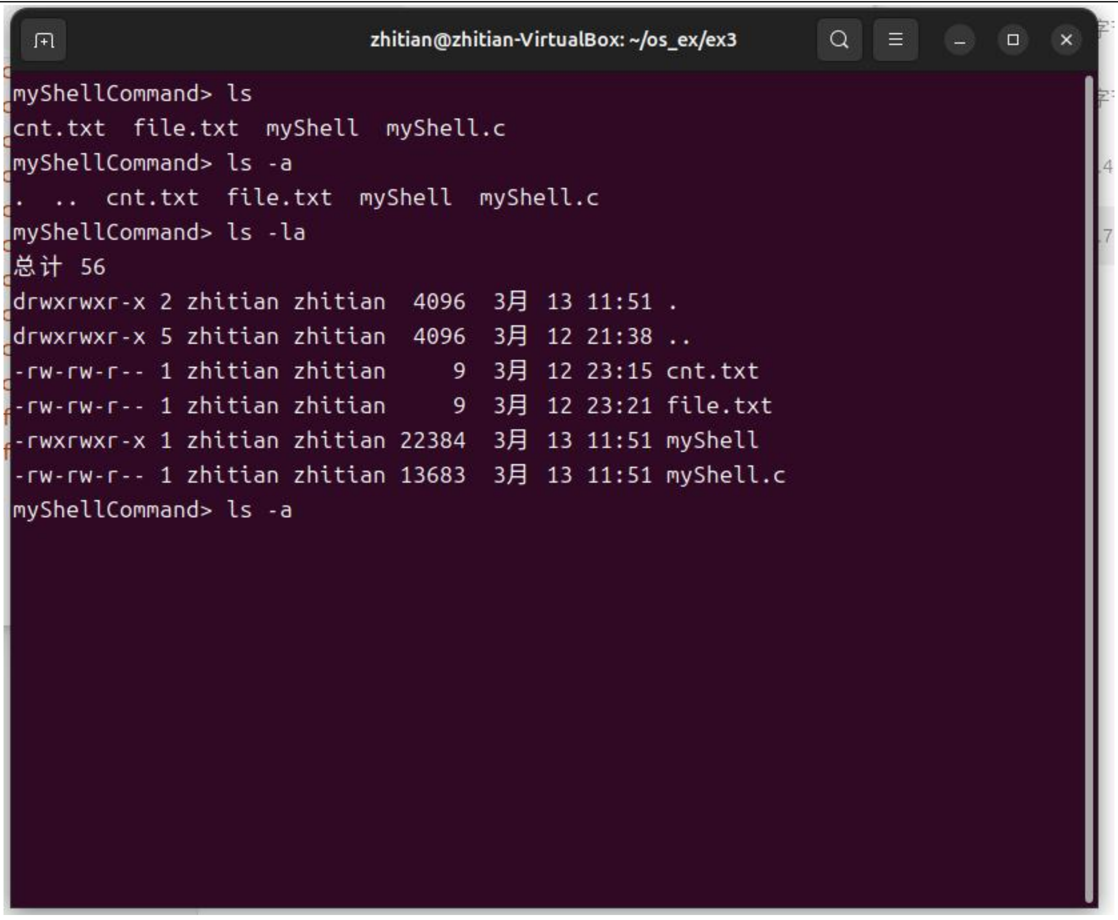
```
myShellCommand> ls
cnt.txt  file.txt  myShell  myShell.c
myShellCommand> ls -a
.  ..  cnt.txt  file.txt  myShell  myShell.c
myShellCommand> ls -la
总计 56
drwxrwxr-x 2 zhitian zhitian  4096  3月  13 11:51 .
drwxrwxr-x 5 zhitian zhitian  4096  3月  12 21:38 ..
-rw-rw-r-- 1 zhitian zhitian     9  3月  12 23:15 cnt.txt
-rw-rw-r-- 1 zhitian zhitian     9  3月  12 23:21 file.txt
-rwxrwxr-x 1 zhitian zhitian 22384  3月  13 11:51 myShell
-rw-rw-r-- 1 zhitian zhitian 13683  3月  13 11:51 myShell.c
myShellCommand> ls -la
```

再按两次上键之后



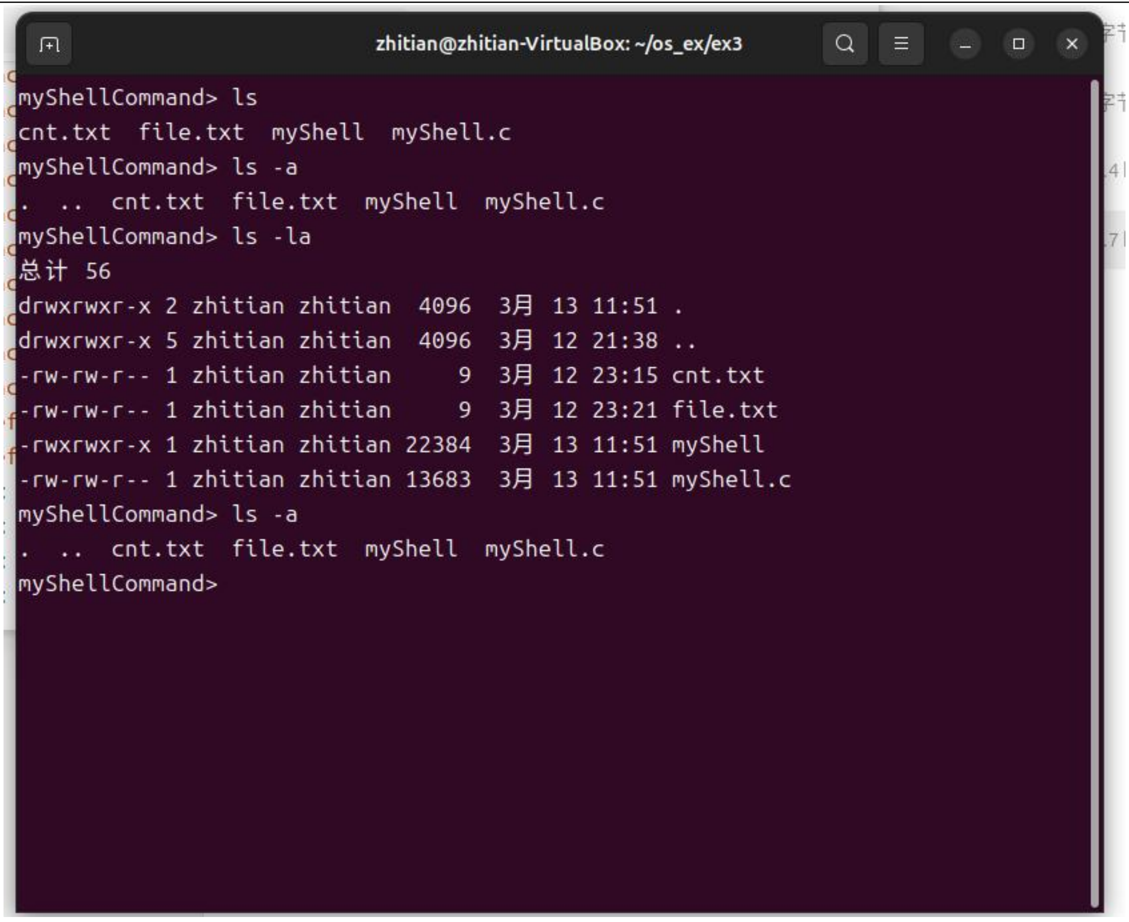
```
myShellCommand> ls
cnt.txt  file.txt  myShell  myShell.c
myShellCommand> ls -a
.  ..  cnt.txt  file.txt  myShell  myShell.c
myShellCommand> ls -la
总计 56
drwxrwxr-x 2 zhitian zhitian  4096  3月  13 11:51 .
drwxrwxr-x 5 zhitian zhitian  4096  3月  12 21:38 ..
-rw-rw-r-- 1 zhitian zhitian     9  3月  12 23:15 cnt.txt
-rw-rw-r-- 1 zhitian zhitian     9  3月  12 23:21 file.txt
-rwxrwxr-x 1 zhitian zhitian 22384  3月  13 11:51 myShell
-rw-rw-r-- 1 zhitian zhitian 13683  3月  13 11:51 myShell.c
myShellCommand> ls
```

再按一次下键



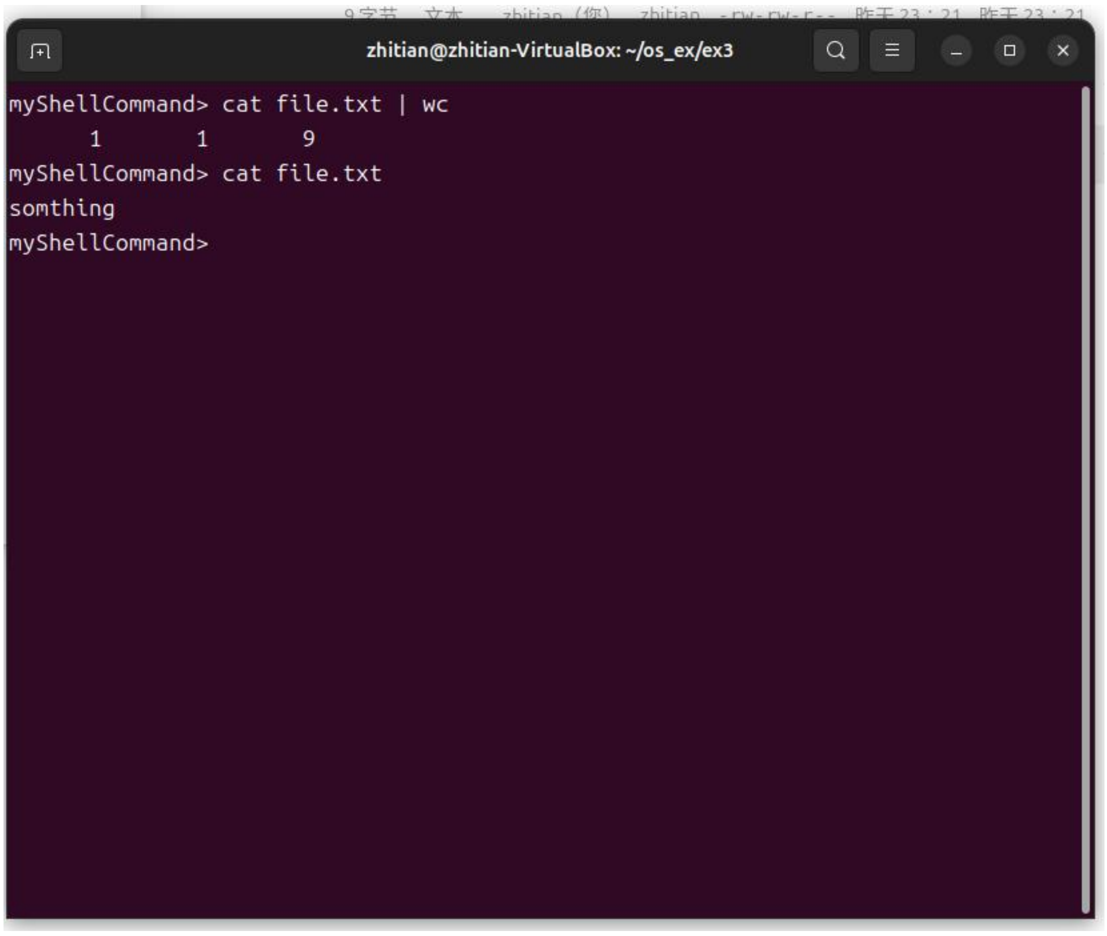
```
myShellCommand> ls
cnt.txt  file.txt  myShell  myShell.c
myShellCommand> ls -a
.  ..  cnt.txt  file.txt  myShell  myShell.c
myShellCommand> ls -la
总计 56
drwxrwxr-x 2 zhitian zhitian  4096  3月  13 11:51 .
drwxrwxr-x 5 zhitian zhitian  4096  3月  12 21:38 ..
-rw-rw-r-- 1 zhitian zhitian     9  3月  12 23:15 cnt.txt
-rw-rw-r-- 1 zhitian zhitian     9  3月  12 23:21 file.txt
-rwxrwxr-x 1 zhitian zhitian 22384  3月  13 11:51 myShell
-rw-rw-r-- 1 zhitian zhitian 13683  3月  13 11:51 myShell.c
myShellCommand> ls -a
```

能够正常执行历史命令


zhitian@zhitian-VirtualBox: ~/os_ex/ex3

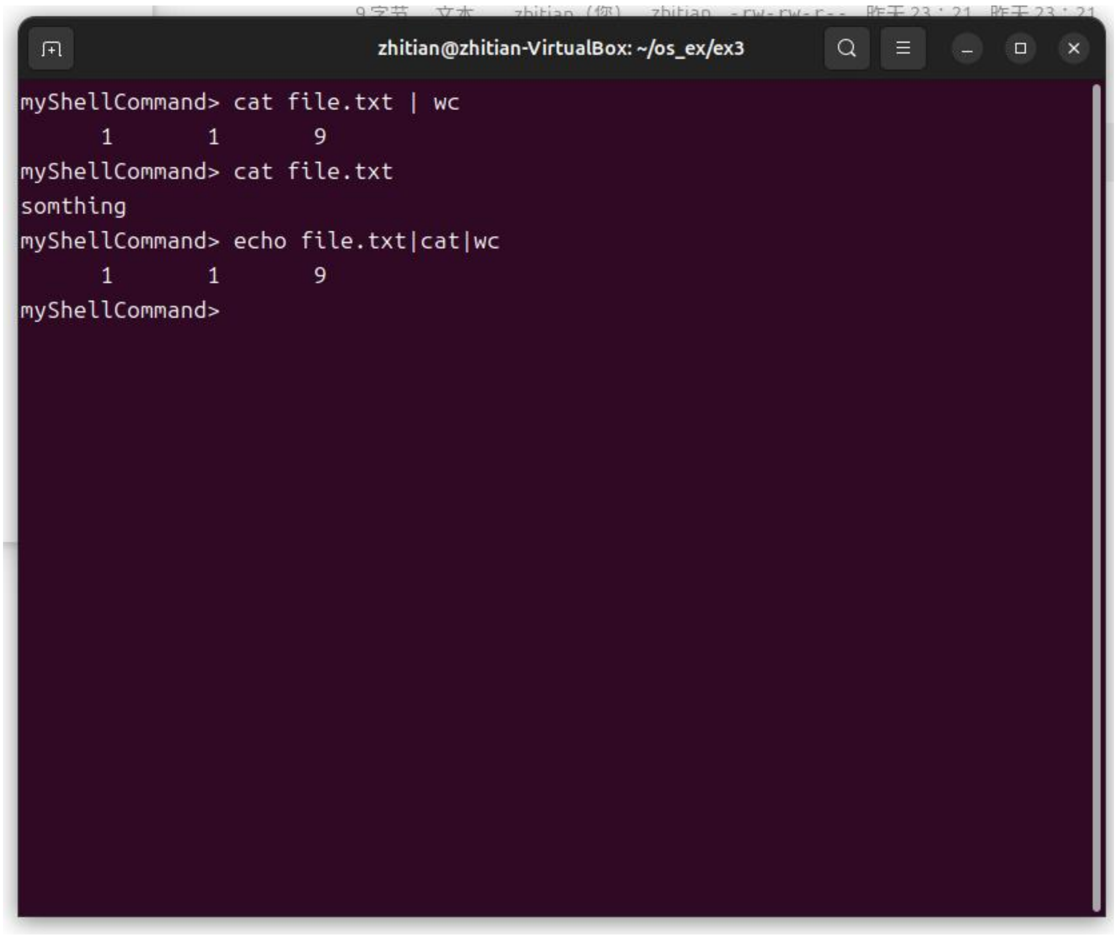
```
myShellCommand> ls
cnt.txt  file.txt  myShell  myShell.c
myShellCommand> ls -a
.  ..  cnt.txt  file.txt  myShell  myShell.c
myShellCommand> ls -la
总计 56
drwxrwxr-x 2 zhitian zhitian  4096  3月 13 11:51 .
drwxrwxr-x 5 zhitian zhitian  4096  3月 12 21:38 ..
-rw-rw-r-- 1 zhitian zhitian     9  3月 12 23:15 cnt.txt
-rw-rw-r-- 1 zhitian zhitian     9  3月 12 23:21 file.txt
-rwxrwxr-x 1 zhitian zhitian 22384  3月 13 11:51 myShell
-rw-rw-r-- 1 zhitian zhitian 13683  3月 13 11:51 myShell.c
myShellCommand> ls -a
.  ..  cnt.txt  file.txt  myShell  myShell.c
myShellCommand>
```

管道示例（请忽略拼写错误）：



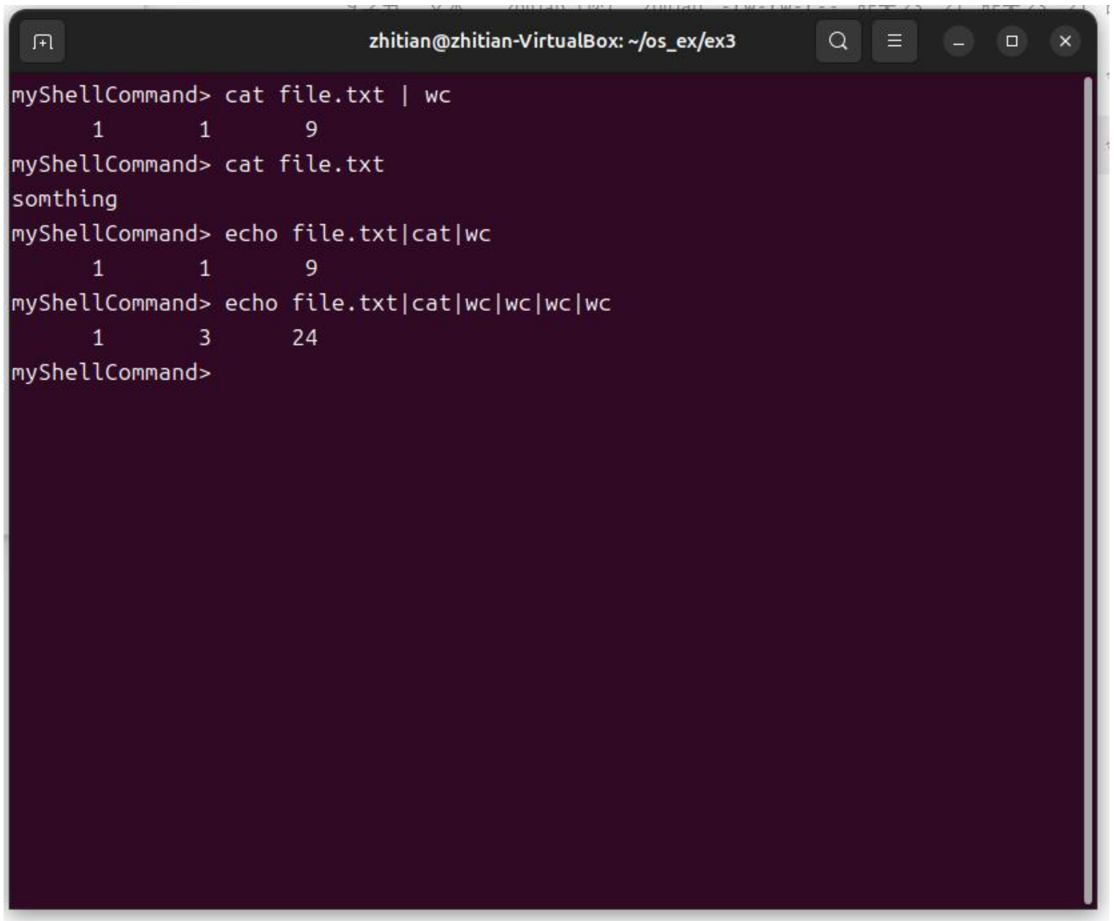
```
myShellCommand> cat file.txt | wc
      1       1       9
myShellCommand> cat file.txt
somthing
myShellCommand>
```

我们还可以有多重管道：



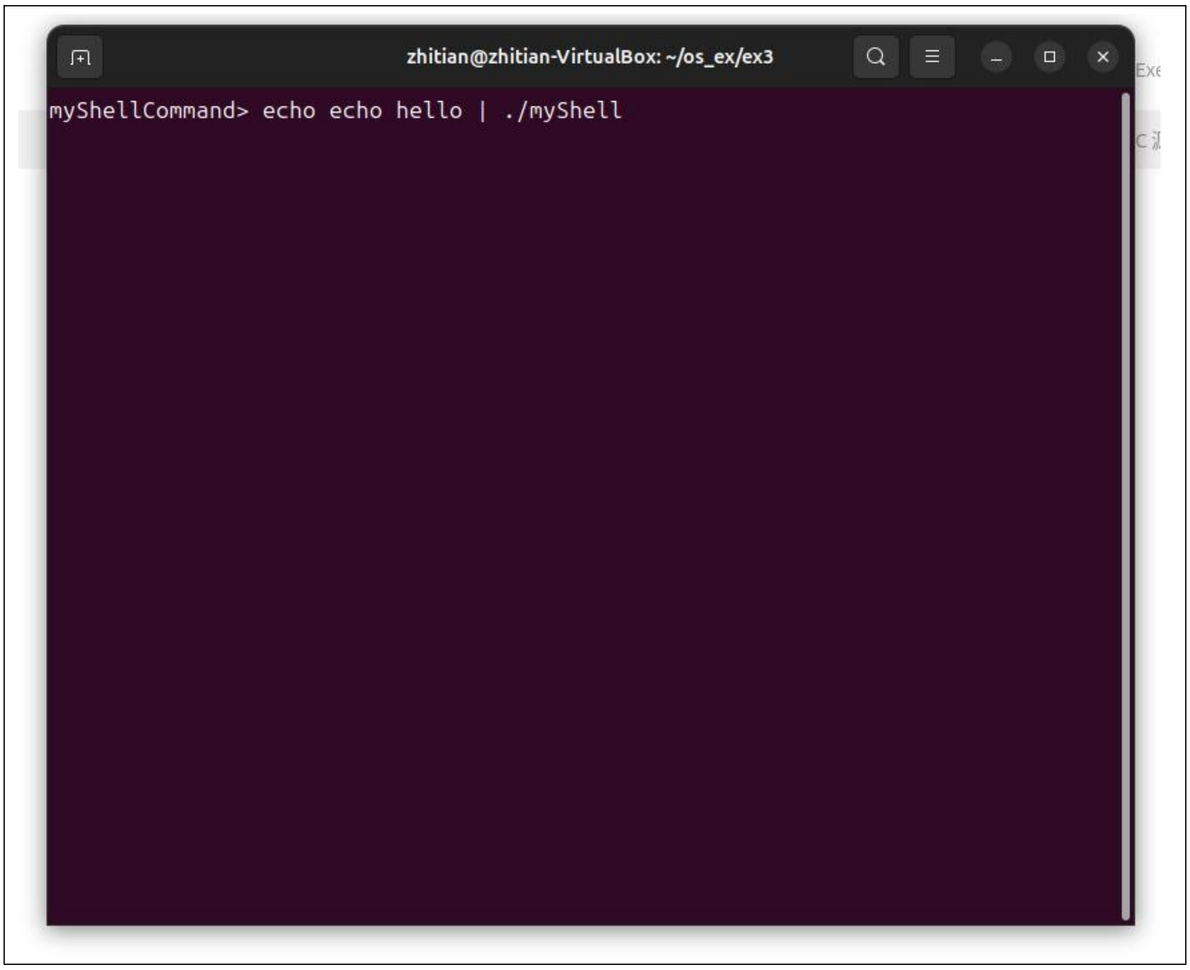
```
myShellCommand> cat file.txt | wc
      1       1       9
myShellCommand> cat file.txt
somthing
myShellCommand> echo file.txt|cat|wc
      1       1       9
myShellCommand>
```

甚至更多

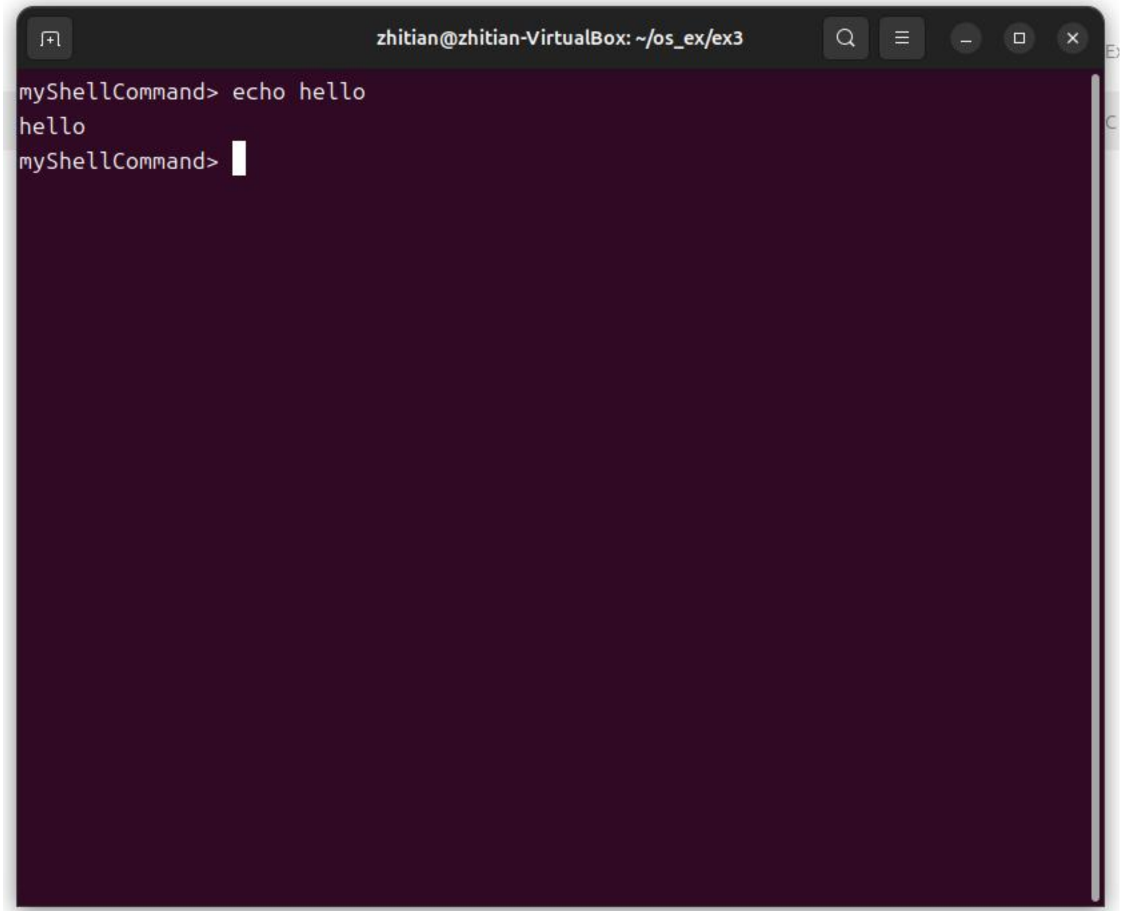


```
myShellCommand> cat file.txt | wc
      1       1       9
myShellCommand> cat file.txt
somthing
myShellCommand> echo file.txt|cat|wc
      1       1       9
myShellCommand> echo file.txt|cat|wc|wc|wc|wc
      1       3      24
myShellCommand>
```

还有给的测试中的自调用（这里是因为我们的 shell 在启动时会清一次屏幕）



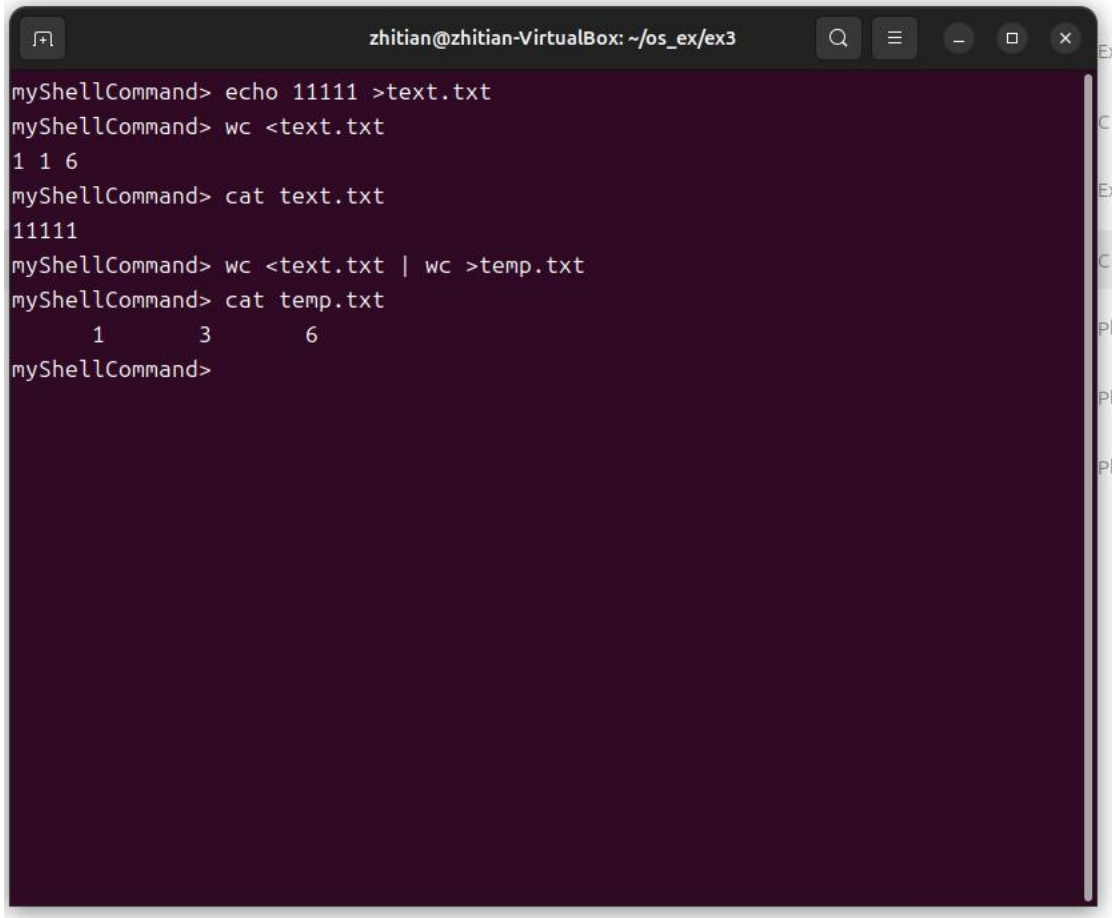
```
myShellCommand> echo echo hello | ./myShell
```



zhitian@zhitian-VirtualBox: ~/os_ex/ex3

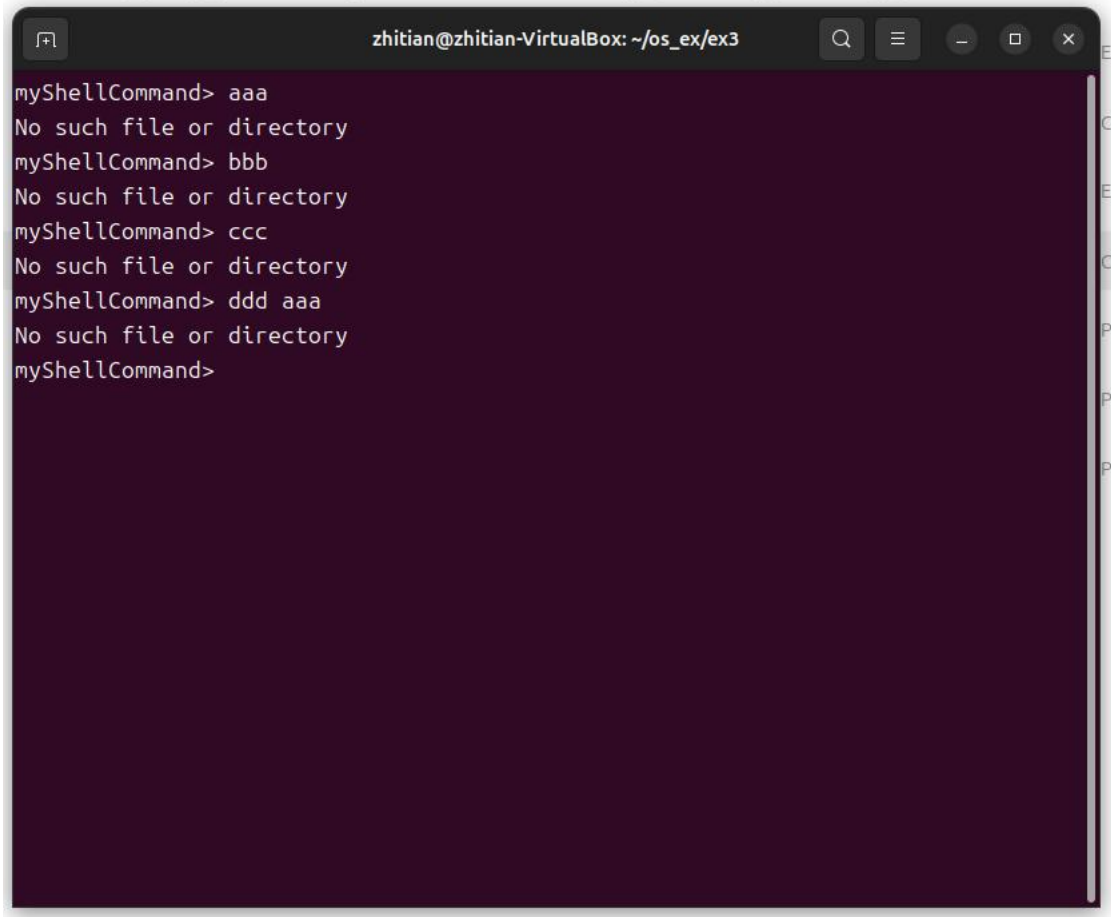
```
myShellCommand> echo hello
hello
myShellCommand> 
```

关于重定向的详细展示（包括了在使用临时管道的情况下的重定向）：

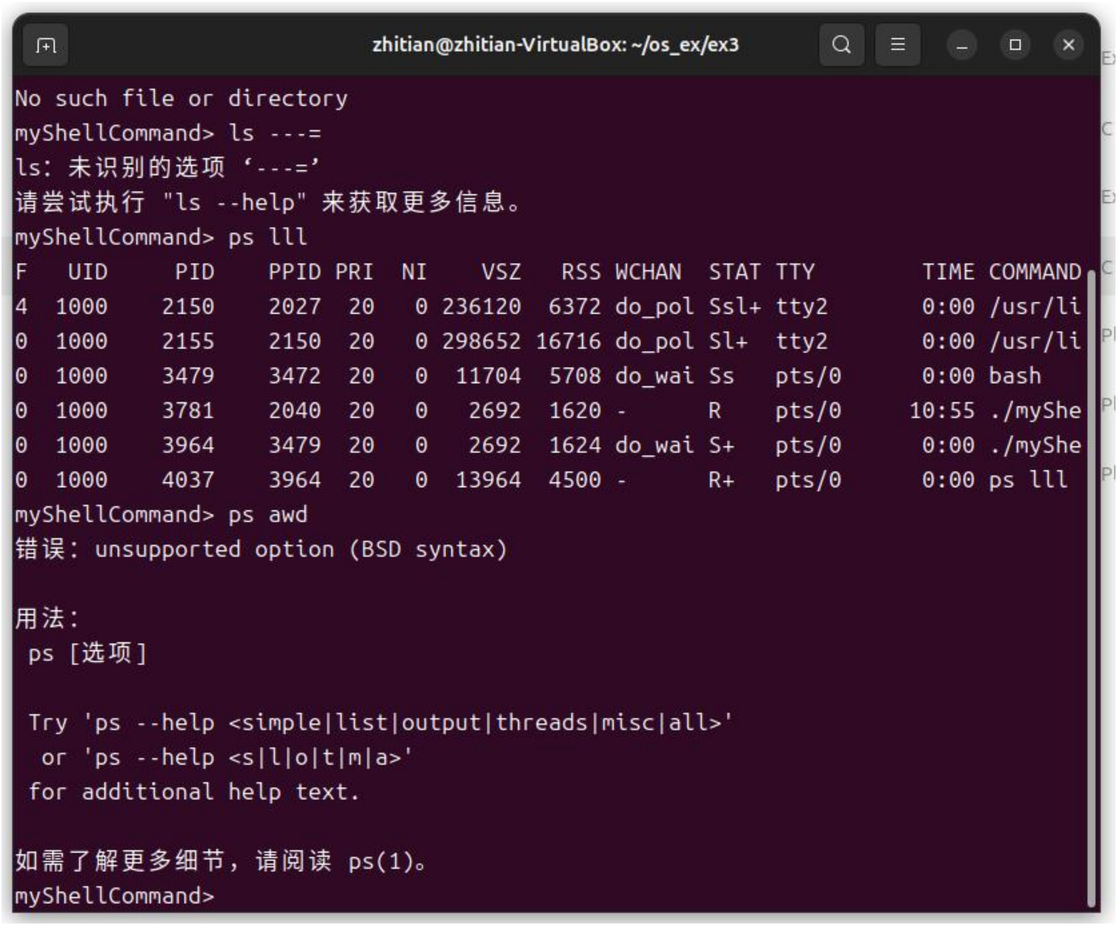


```
myShellCommand> echo 11111 >text.txt
myShellCommand> wc <text.txt
1 1 6
myShellCommand> cat text.txt
11111
myShellCommand> wc <text.txt | wc >temp.txt
myShellCommand> cat temp.txt
      1       3       6
myShellCommand>
```

错误检测（使用自带的 perror）

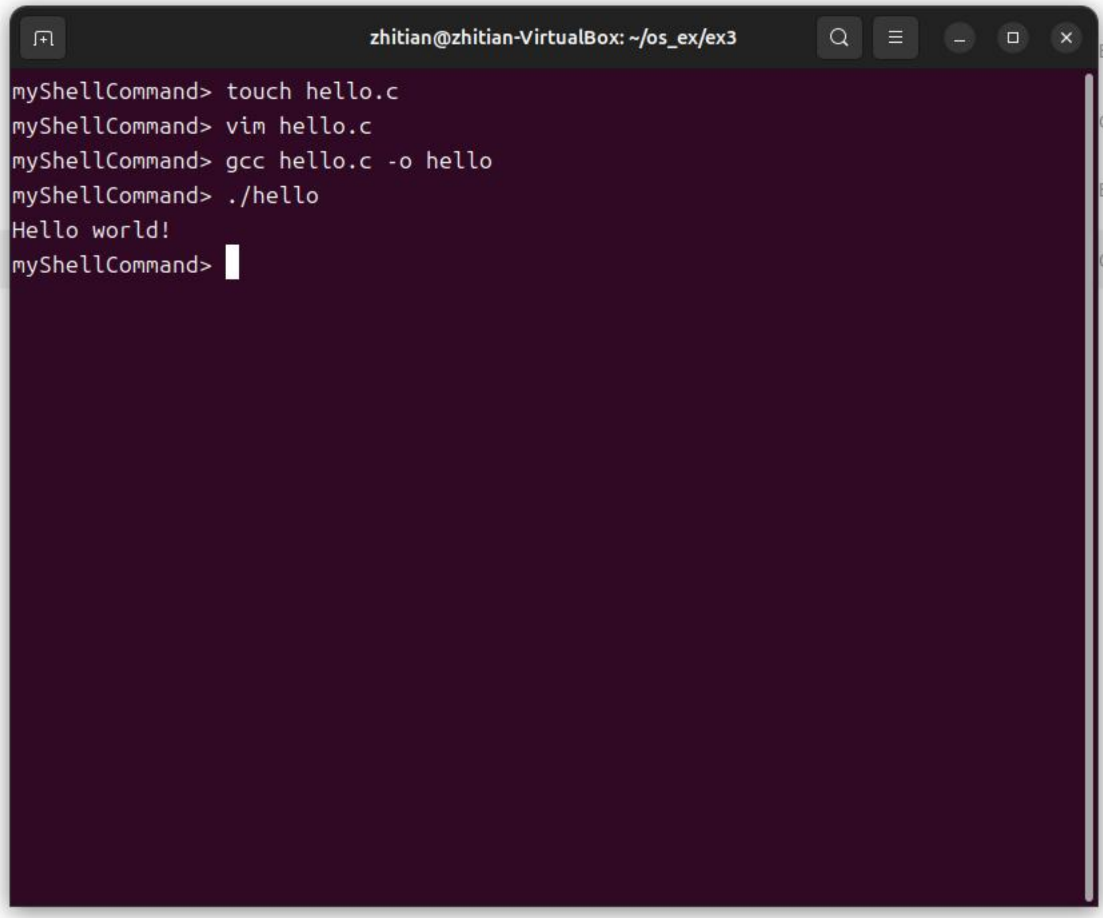


```
myShellCommand> aaa
No such file or directory
myShellCommand> bbb
No such file or directory
myShellCommand> ccc
No such file or directory
myShellCommand> ddd aaa
No such file or directory
myShellCommand>
```



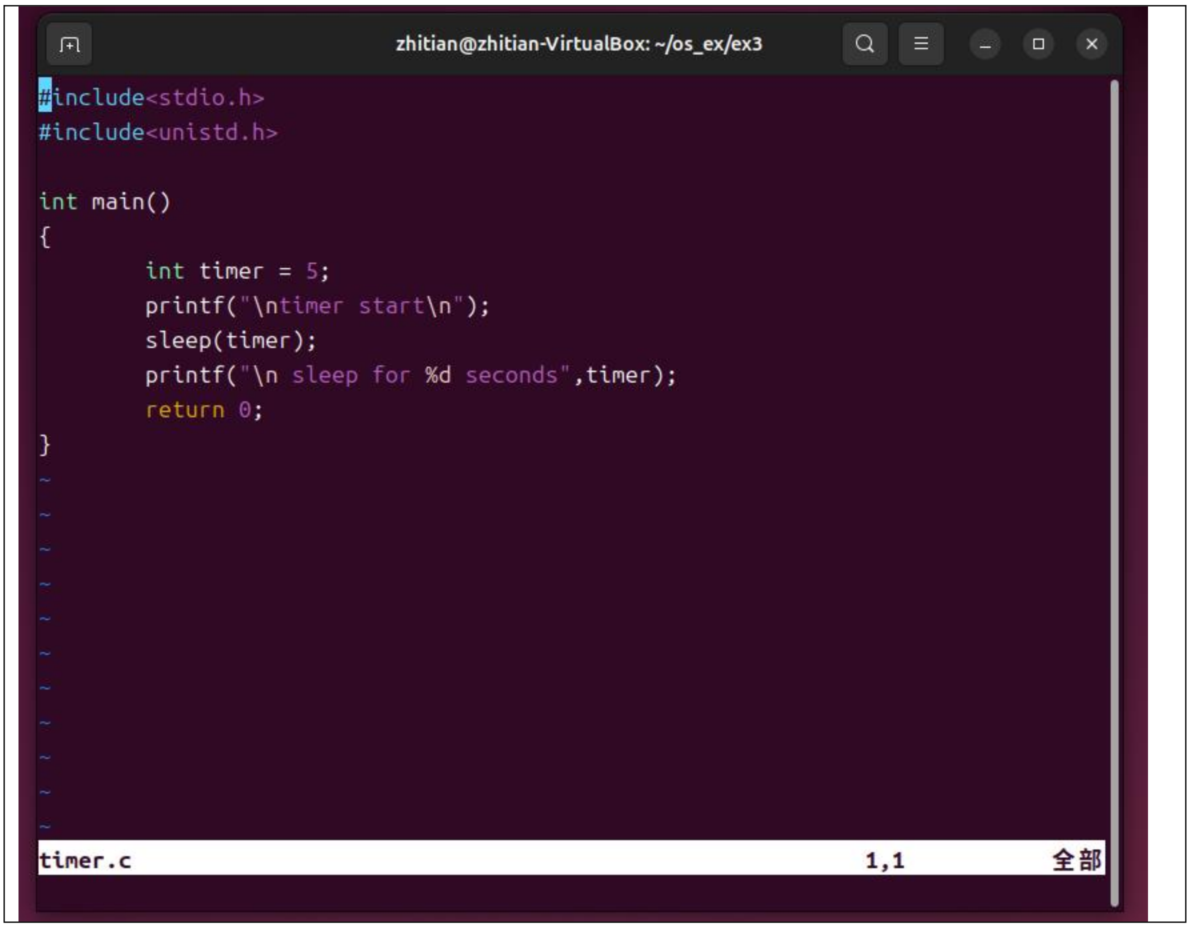
```
No such file or directory
myShellCommand> ls ---=
ls: 未识别的选项 ‘---=’
请尝试执行 "ls --help" 来获取更多信息。
myShellCommand> ps lll
F   UID     PID    PPID PRI  NI    VSZ   RSS WCHAN  STAT TTY        TIME COMMAND
4  1000    2150    2027  20   0 236120  6372 do_pol Ssl+ tty2       0:00 /usr/li
0  1000    2155    2150  20   0 298652 16716 do_pol Sl+  tty2       0:00 /usr/li
0  1000    3479    3472  20   0  11704  5708 do_wai Ss   pts/0      0:00 bash
0  1000    3781    2040  20   0   2692  1620 -      R    pts/0     10:55 ./myShe
0  1000    3964    3479  20   0   2692  1624 do_wai S+   pts/0      0:00 ./myShe
0  1000    4037    3964  20   0  13964  4500 -      R+   pts/0      0:00 ps lll
myShellCommand> ps awd
错误: unsupported option (BSD syntax)

用法:
 ps [选项]

 Try 'ps --help <simple|list|output|threads|misc|all>'
  or 'ps --help <s|l|o|t|m|a>'
 for additional help text.

如需了解更多细节，请阅读 ps(1)。
myShellCommand>
```

也可以做到像普通 shell 一样编写程序编译程序：

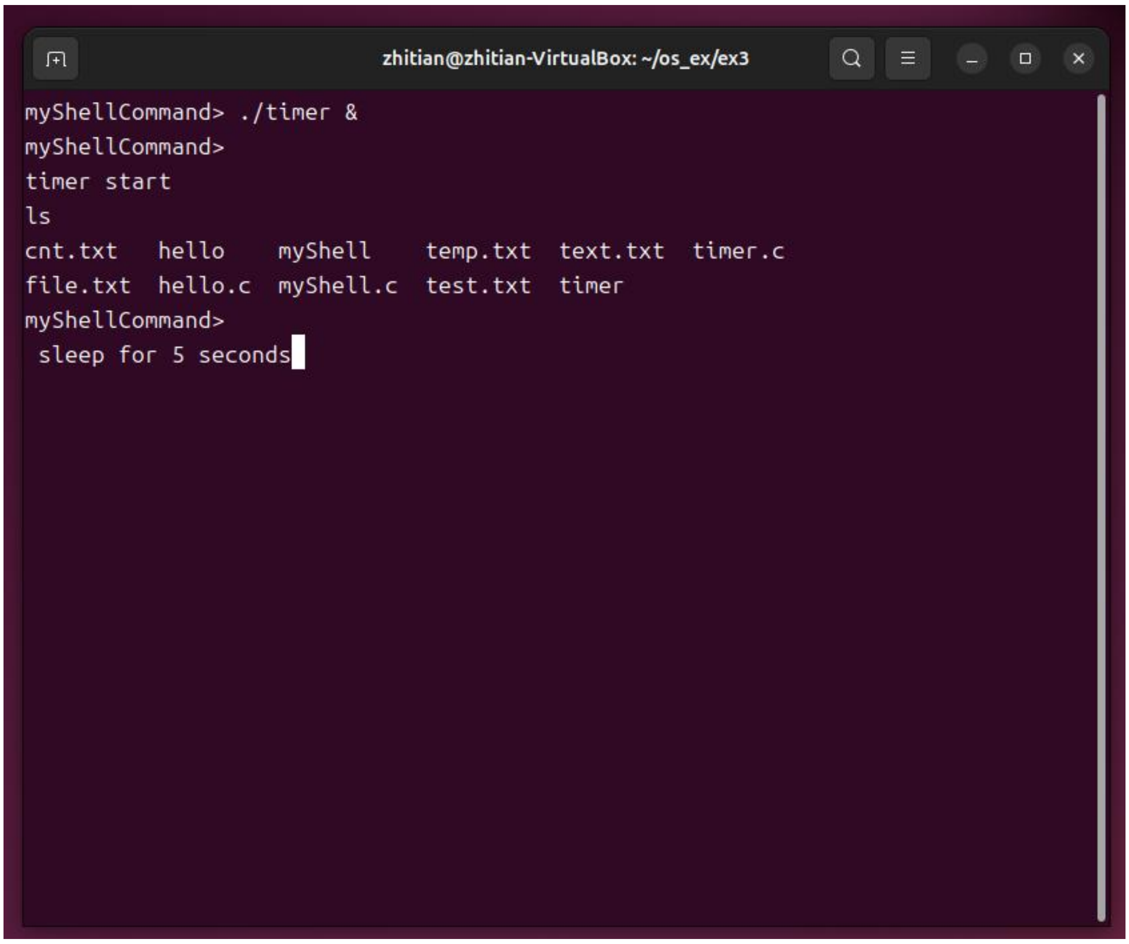


```
myShellCommand> touch hello.c
myShellCommand> vim hello.c
myShellCommand> gcc hello.c -o hello
myShellCommand> ./hello
Hello world!
myShellCommand> 
```

除此之外还有后台运行的功能，为了演示这个功能我们使用如下测试程序：

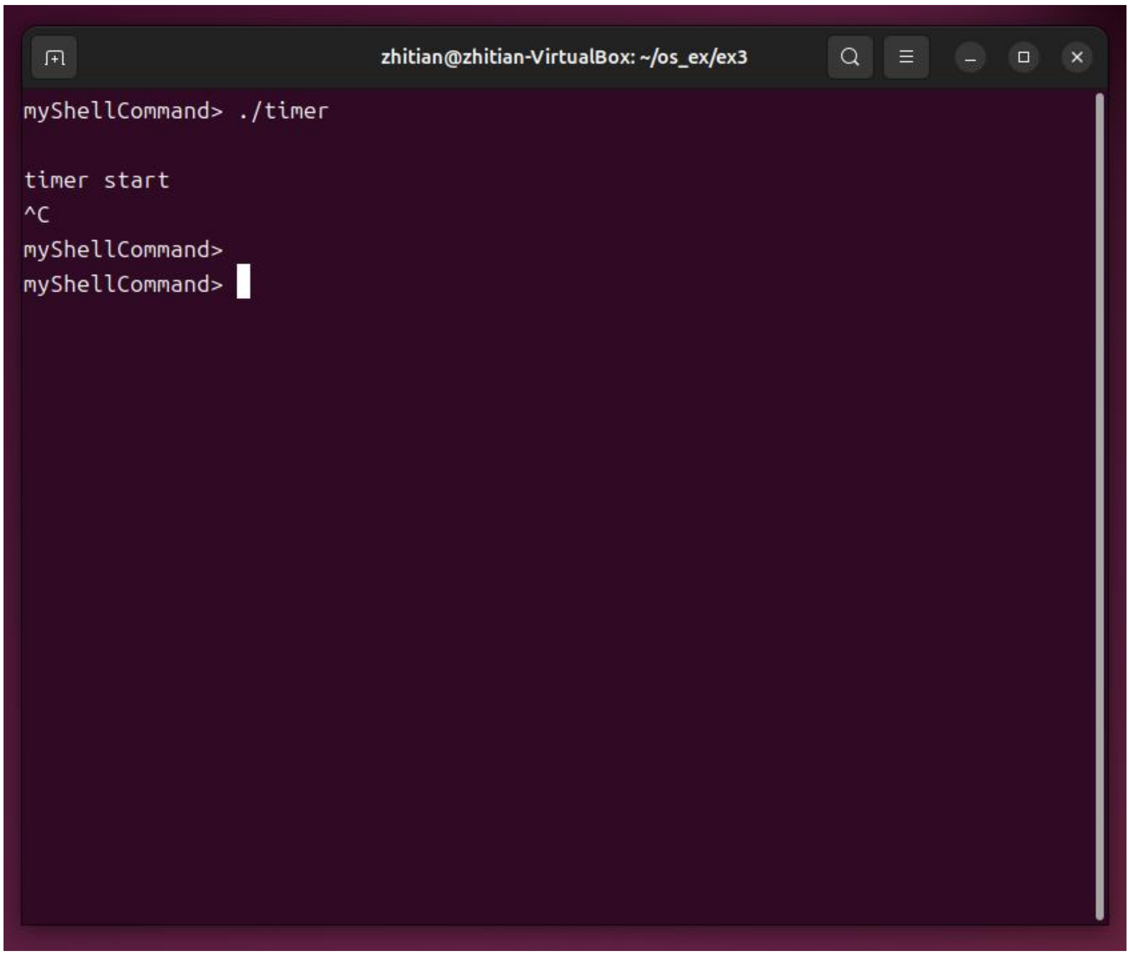


```
#include<stdio.h>
#include<unistd.h>

int main()
{
        int timer = 5;
        printf("\ntimer start\n");
        sleep(timer);
        printf("\n sleep for %d seconds",timer);
        return 0;
}
~
~
~
~
~
~
~
~
~
~
~
timer.c                                                    1,1          全部
```



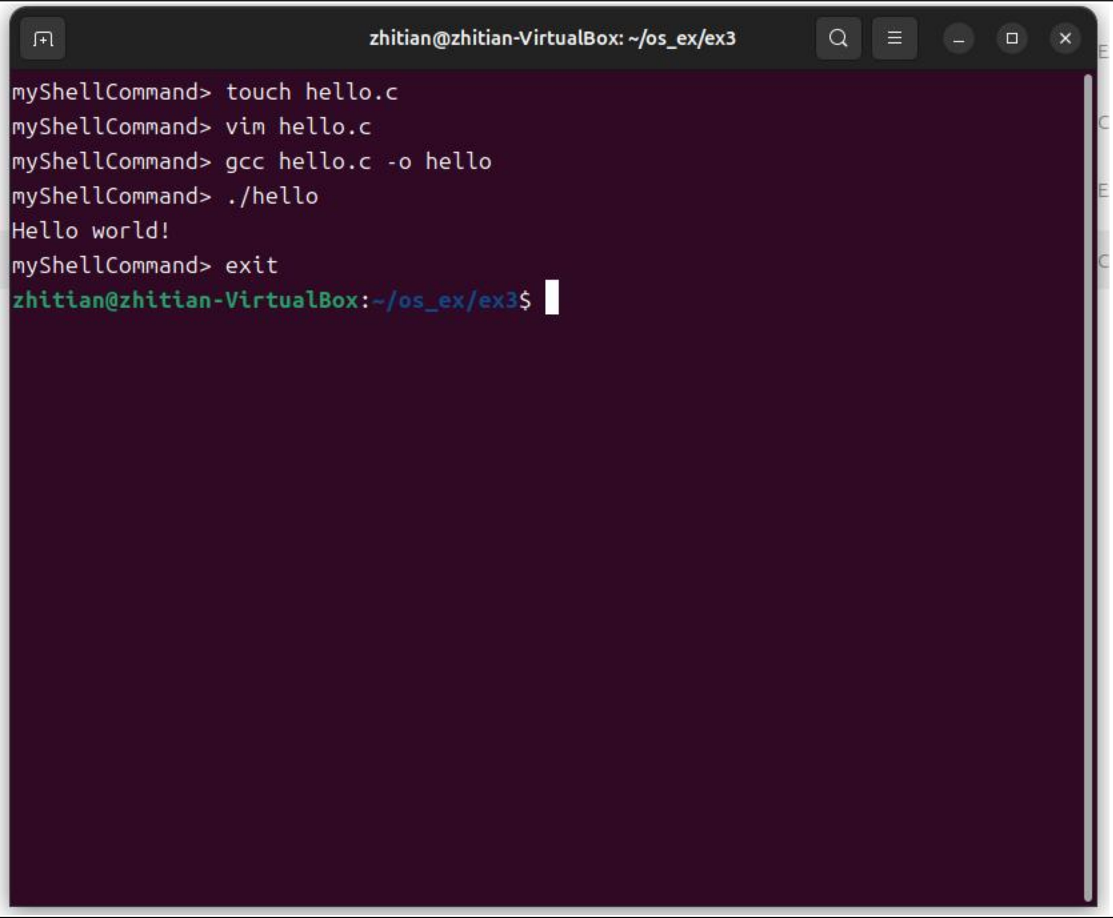
```
myShellCommand> ./timer &
myShellCommand>
timer start
ls
cnt.txt   hello    myShell    temp.txt  text.txt  timer.c
file.txt  hello.c  myShell.c  test.txt  timer
myShellCommand>
 sleep for 5 seconds
```

这里在这中间我们又输入了一次 ls 命令，因为系统调度顺序的问题这里显示的文本顺序可能有些问题，但是不影响我们 shell 的使用。
而使用^C 关闭正在运行的程序我们同样使用这个 timer



```
myShellCommand> ./timer

timer start
^C
myShellCommand>
myShellCommand>
```

下方那一行使我们又键入了一次^C，shell 不会退出
exit 命令退出 shell



```
myShellCommand> touch hello.c
myShellCommand> vim hello.c
myShellCommand> gcc hello.c -o hello
myShellCommand> ./hello
Hello world!
myShellCommand> exit
zhitian@zhitian-VirtualBox:~/os_ex/ex3$
```

结论：

与一个标准的 shell 比较，myShell 没有更换工作目录，没有环境变量传递，重定向不够完善，不兼容<<和>>，错误提示不够完善。没有字符高亮，没有用户显示，没有 tab 补全，无法将双引号括住的部分作为一个整体的字符串处理。还有很多我想不到的缺陷。关于解决方案，更换工作目录可以先看看系统有没有更换进程工作目录的函数，如果没有也可以手动存储，将工作目录与程序启动目录的相对路径在指令解释的时候作为前缀。环境变量可以多加一个环境变量解析器，并在 exec 调用时作为参数传入。完善重定向只需要更改重定向解析器的 open 的参数即可。错误提示可以手动加入一些，比如没有对应的程序的时候加一个提示 sudo apt install，暂时想不出更多提示的形式。字符高亮可以用转义字实现，用户显示可以获取后作为命令提示符的一部分一起输出出来，tab 补全可以让程序在检测到 tab 键入后使用 ls 获取所有文件和目录的文件名,然后比较已输入的字符串找到第一个可以进行补全的字符串。引号识别就需要大改解释器里的分词器了，相对会更麻烦一些。