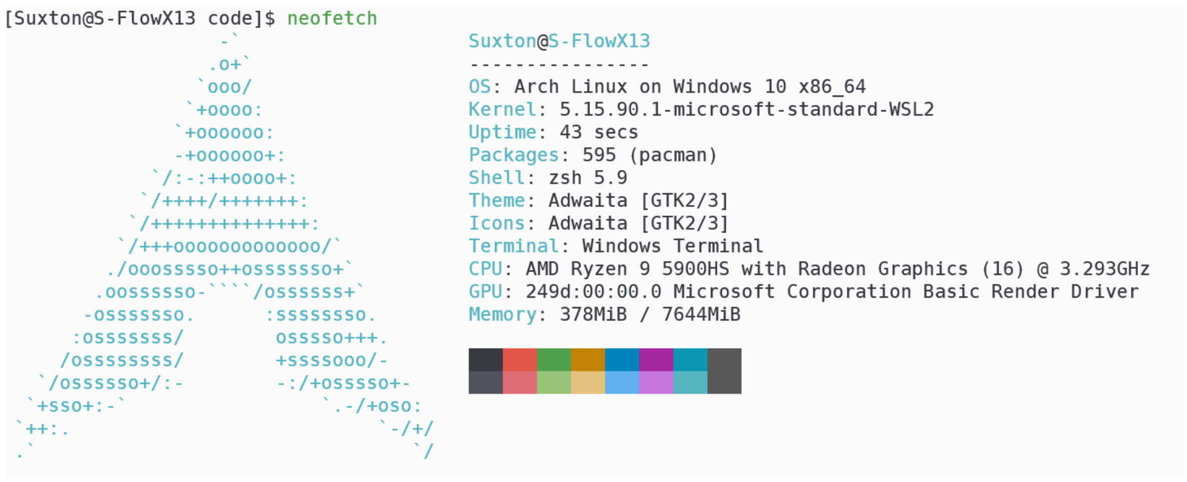
实验二 DEVELOPING A SHELL

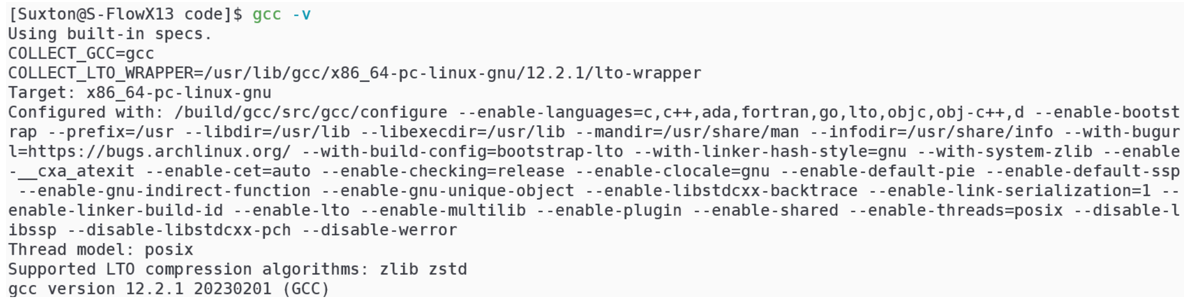
构建程序

1. 程序的测试运行环境为 Arch Linux，内核版本5.15



```
[Suxton@S-FlowX13 code]$ neofetch
                   -`                    Suxton@S-FlowX13
                  .o+`                   ----------------
                 `ooo/                   OS: Arch Linux on Windows 10 x86_64
                `+oooo:                  Kernel: 5.15.90.1-microsoft-standard-WSL2
               `+oooooo:                 Uptime: 43 secs
               -+oooooo+:                Packages: 595 (pacman)
             `/:-:++oooo+:               Shell: zsh 5.9
            `/++++/+++++++:              Theme: Adwaita [GTK2/3]
           `/++++++++++++++:             Icons: Adwaita [GTK2/3]
          `/+++ooooooooooooo/`           Terminal: Windows Terminal
         ./ooosssso++osssssso+`          CPU: AMD Ryzen 9 5900HS with Radeon Graphics (16) @ 3.293GHz
        .oossssso-````/ossssss+`         GPU: 249d:00:00.0 Microsoft Corporation Basic Render Driver
       -osssssso.      :ssssssso.        Memory: 378MiB / 7644MiB
      :osssssss/        osssso+++.
     /ossssssss/        +ssssooo/-
   `/ossssso+/:-        -:/+osssso+-
  `+sso+:-`                 `.-/+oso:
 `++:.                           `-/+/
 .`                                 `/
```

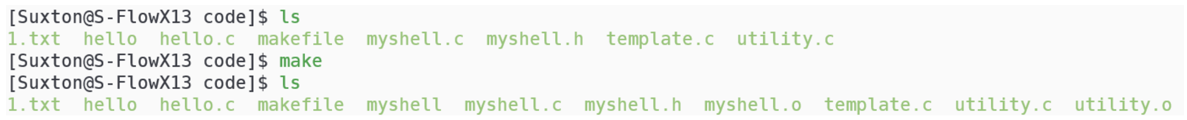
2. gcc版本12.2.1



```
[Suxton@S-FlowX13 code]$ gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/lib/gcc/x86_64-pc-linux-gnu/12.2.1/lto-wrapper
Target: x86_64-pc-linux-gnu
Configured with: /build/gcc/src/gcc/configure --enable-languages=c,c++,ada,fortran,go,lto,objc,obj-c++,d --enable-bootst
rap --prefix=/usr --libdir=/usr/lib --libexecdir=/usr/lib --mandir=/usr/share/man --infodir=/usr/share/info --with-bugur
l=https://bugs.archlinux.org/ --with-build-config=bootstrap-lto --with-linker-hash-style=gnu --with-system-zlib --enable
-__cxa_atexit --enable-cet=auto --enable-checking=release --enable-clocale=gnu --enable-default-pie --enable-default-ssp
 --enable-gnu-indirect-function --enable-gnu-unique-object --enable-libstdcxx-backtrace --enable-link-serialization=1 --
enable-linker-build-id --enable-lto --enable-multilib --enable-plugin --enable-shared --enable-threads=posix --disable-l
ibssp --disable-libstdcxx-pch --disable-werror
Thread model: posix
Supported LTO compression algorithms: zlib zstd
gcc version 12.2.1 20230201 (GCC)
```

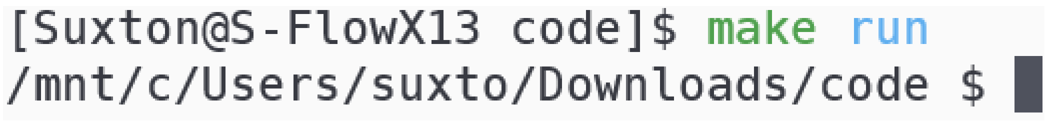
3. makefile种定义了两种构建方式，默认是只构建。如果加 run 的话就是编译并运行。

   仅编译



```
[Suxton@S-FlowX13 code]$ ls
1.txt  hello  hello.c  makefile  myshell.c  myshell.h  template.c  utility.c
[Suxton@S-FlowX13 code]$ make
[Suxton@S-FlowX13 code]$ ls
1.txt  hello  hello.c  makefile  myshell  myshell.c  myshell.h  myshell.o  template.c  utility.c  utility.o
```

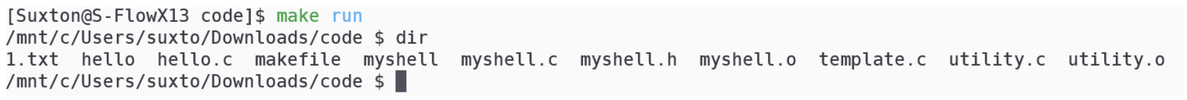
   编译运行



```
[Suxton@S-FlowX13 code]$ make run
/mnt/c/Users/suxto/Downloads/code $
```

4. 界面说明：

   在 $ 前面的是当前的工作目录，后面是输入的命令

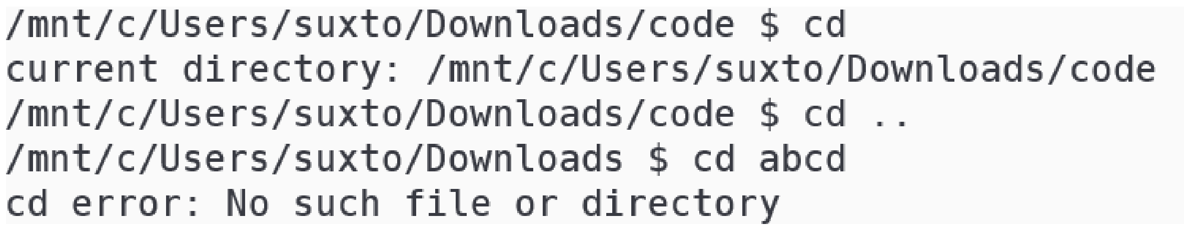


```
[Suxton@S-FlowX13 code]$ make run
/mnt/c/Users/suxto/Downloads/code $ dir
1.txt  hello  hello.c  makefile  myshell  myshell.c  myshell.h  myshell.o  template.c  utility.c  utility.o
/mnt/c/Users/suxto/Downloads/code $
```

内置命令

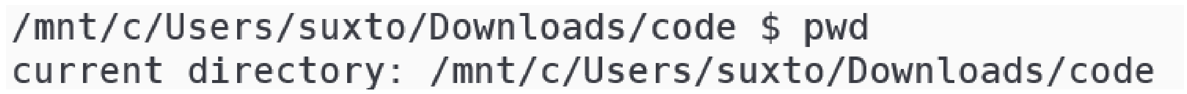
1. cd

   在没有参数的时候会输出当前的目录，有参数的时候会选择目录或者输出错误信息。



```
/mnt/c/Users/suxto/Downloads/code $ cd
current directory: /mnt/c/Users/suxto/Downloads/code
/mnt/c/Users/suxto/Downloads/code $ cd ..
/mnt/c/Users/suxto/Downloads $ cd abcd
cd error: No such file or directory
```

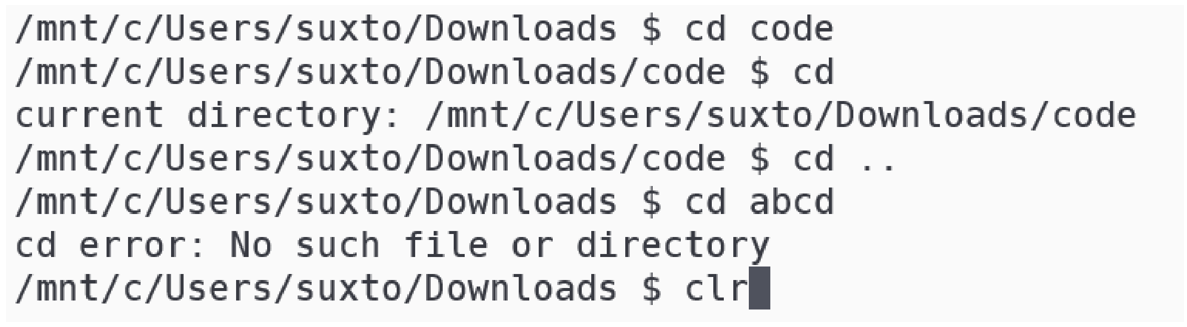
   而且pwd命令被无参数的cd替换



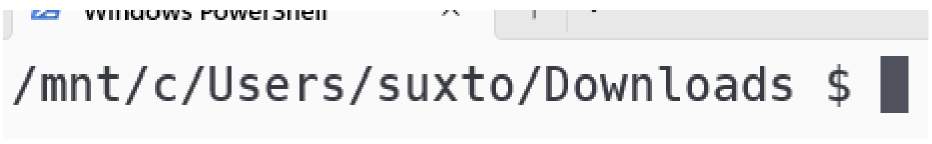
```
/mnt/c/Users/suxto/Downloads/code $ pwd
current directory: /mnt/c/Users/suxto/Downloads/code
```

2. clr

   清楚屏幕信息

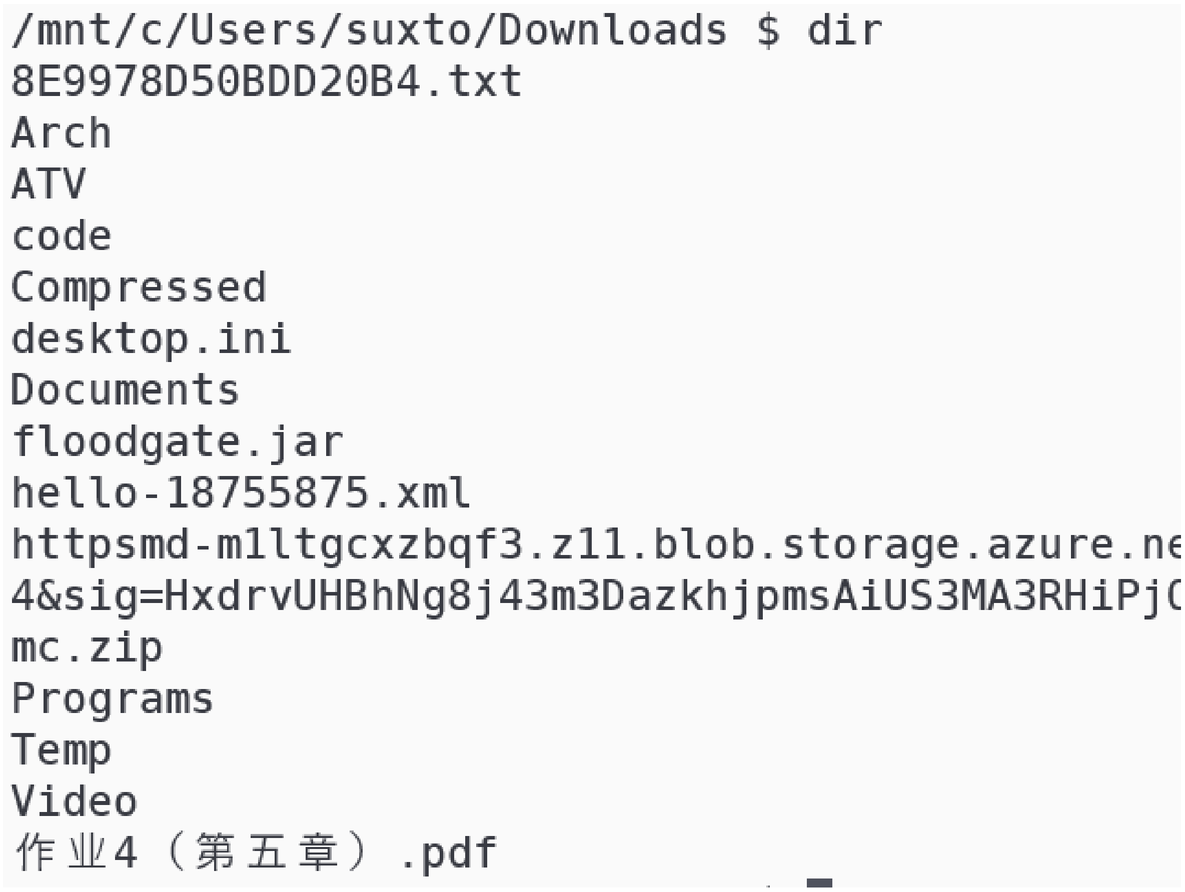


```
/mnt/c/Users/suxto/Downloads $ cd code
/mnt/c/Users/suxto/Downloads/code $ cd
current directory: /mnt/c/Users/suxto/Downloads/code
/mnt/c/Users/suxto/Downloads/code $ cd ..
/mnt/c/Users/suxto/Downloads $ cd abcd
cd error: No such file or directory
/mnt/c/Users/suxto/Downloads $ clr
```



```
/mnt/c/Users/suxto/Downloads $
```

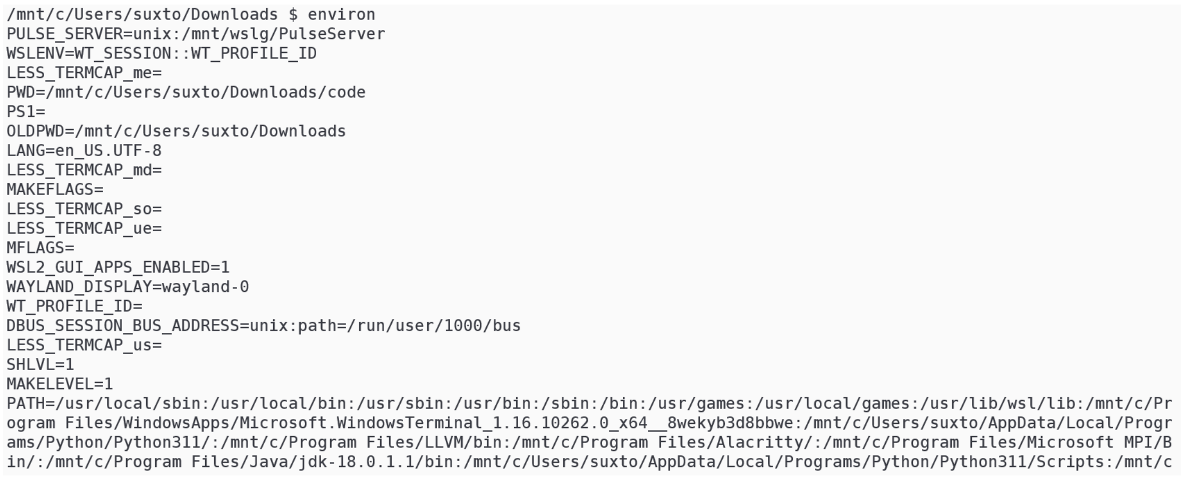
3. dir



```
/mnt/c/Users/suxto/Downloads $ dir
8E9978D50BDD20B4.txt
Arch
ATV
code
Compressed
desktop.ini
Documents
floodgate.jar
hello-18755875.xml
httpsmd-m1ltgcxzbqf3.z11.blob.storage.azure.ne
4&sig=HxdrvUHBhNg8j43m3DazkhjpmsAiUS3MA3RHiPjC
mc.zip
Programs
Temp
Video
作业4（第五章）.pdf
```

4. environ

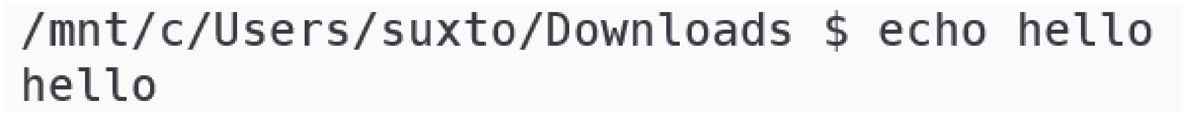
   显示所有环境变量



```
/mnt/c/Users/suxto/Downloads $ environ
PULSE_SERVER=unix:/mnt/wslg/PulseServer
WSLENV=WT_SESSION::WT_PROFILE_ID
LESS_TERMCAP_me=
PWD=/mnt/c/Users/suxto/Downloads/code
PS1=
OLDPWD=/mnt/c/Users/suxto/Downloads
LANG=en_US.UTF-8
LESS_TERMCAP_md=
MAKEFLAGS=
LESS_TERMCAP_so=
LESS_TERMCAP_ue=
MFLAGS=
WSL2_GUI_APPS_ENABLED=1
WAYLAND_DISPLAY=wayland-0
WT_PROFILE_ID=
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus
LESS_TERMCAP_us=
SHLVL=1
MAKELEVEL=1
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/usr/lib/wsl/lib:/mnt/c/Pr
ogram Files/WindowsApps/Microsoft.WindowsTerminal_1.16.10262.0_x64__8wekyb3d8bbwe:/mnt/c/Users/suxto/AppData/Local/Progr
ams/Python/Python311/:/mnt/c/Program Files/LLVM/bin:/mnt/c/Program Files/Alacritty/:/mnt/c/Program Files/Microsoft MPI/B
in/:/mnt/c/Program Files/Java/jdk-18.0.1.1/bin:/mnt/c/Users/suxto/AppData/Local/Programs/Python/Python311/Scripts:/mnt/c
```

5. echo

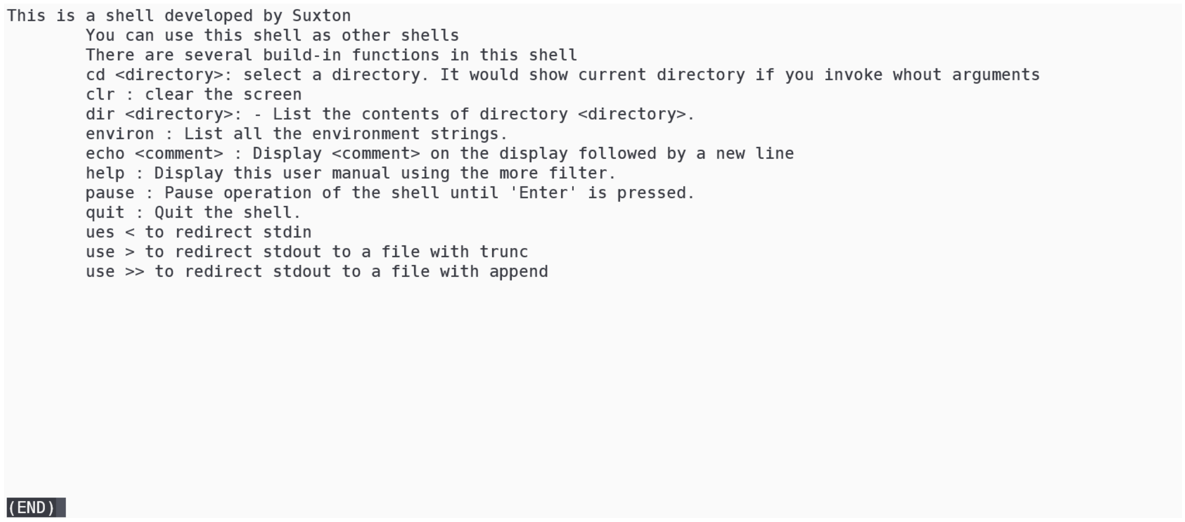
   输出后面的参数



```
/mnt/c/Users/suxto/Downloads $ echo hello
hello
```

6. help

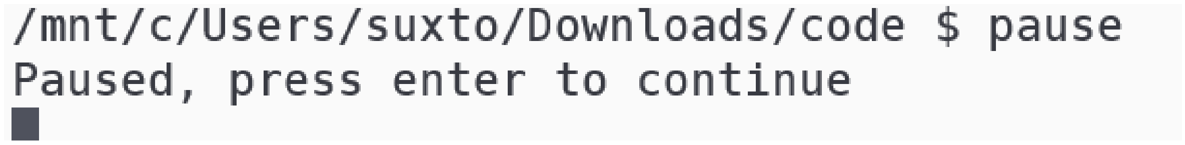
   显示帮助手册，使用more输出，这样当终端空间不够的时候可以上下翻页



```
This is a shell developed by Suxton
        You can use this shell as other shells
        There are several build-in functions in this shell
        cd <directory>: select a directory. It would show current directory if you invoke whout arguments
        clr : clear the screen
        dir <directory>: - List the contents of directory <directory>.
        environ : List all the environment strings.
        echo <comment> : Display <comment> on the display followed by a new line
        help : Display this user manual using the more filter.
        pause : Pause operation of the shell until 'Enter' is pressed.
        quit : Quit the shell.
        ues < to redirect stdin
        use > to redirect stdout to a file with trunc
        use >> to redirect stdout to a file with append







(END)
```

7. pause

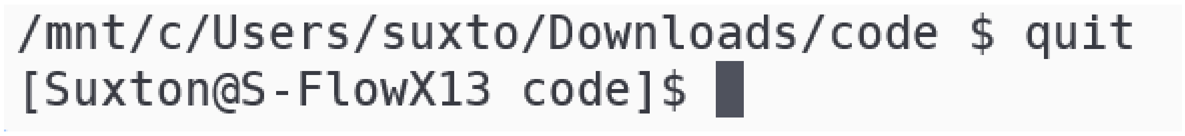
   暂停shell，直到空格按下



```
/mnt/c/Users/suxto/Downloads/code $ pause
Paused, press enter to continue
```

8. quit

   退出myshell

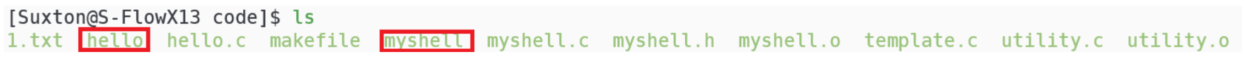


```
/mnt/c/Users/suxto/Downloads/code $ quit
[Suxton@S-FlowX13 code]$
```

myshell环境

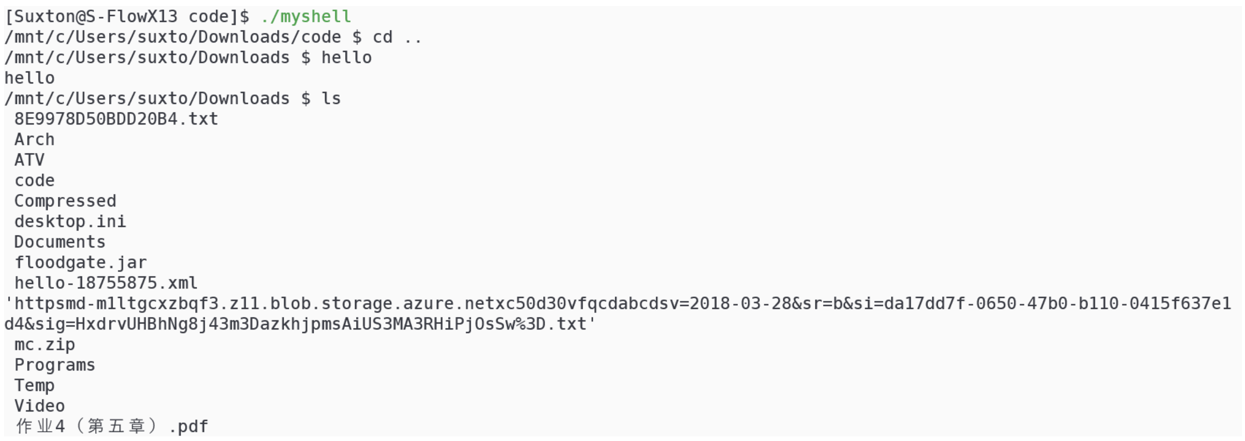
我有一个hello程序，可以输出一段hello，和myshell处于同一个目录下。

红色框中的是可执行文件



```
[Suxton@S-FlowX13 code]$ ls
1.txt  hello  hello.c  makefile  myshell  myshell.c  myshell.h  myshell.o  template.c  utility.c  utility.o
```

现在进入myshell，选择其他的目录，运行hello，成功执行。而且很显然在 当前目录下并没有hello的可执行程序。所以，只要将可执行文件放在myshell目录下就能在任何地方执行。

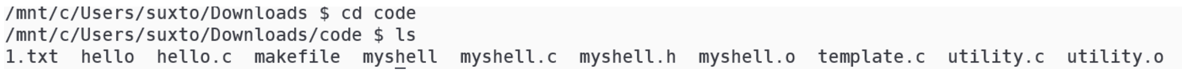


```
[Suxton@S-FlowX13 code]$ ./myshell
/mnt/c/Users/suxto/Downloads/code $ cd ..
/mnt/c/Users/suxto/Downloads $ hello
hello
/mnt/c/Users/suxto/Downloads $ ls
 8E9978D50BDD20B4.txt
 Arch
 ATV
 code
 Compressed
 desktop.ini
 Documents
 floodgate.jar
 hello-18755875.xml
'httpsmd-m1ltgcxzbqf3.z11.blob.storage.azure.netxc50d30vfqcdabcdsv=2018-03-28&sr=b&si=da17dd7f-0650-47b0-b110-0415f637e1
d4&sig=HxdrvUHBhNg8j43m3DazkhjpmsAiUS3MA3RHiPjOsSw%3D.txt'
 mc.zip
 Programs
 Temp
 Video
 作 业4（第 五 章）.pdf
```

外部命令

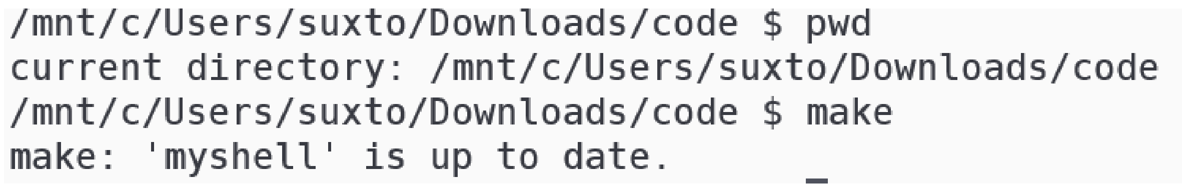
外部命令太多了，这里测试几个有代表性的。

1. ls



```
/mnt/c/Users/suxto/Downloads $ cd code
/mnt/c/Users/suxto/Downloads/code $ ls
1.txt  hello  hello.c  makefile  myshell  myshell.c  myshell.h  myshell.o  template.c  utility.c  utility.o
```

2. make



```
/mnt/c/Users/suxto/Downloads/code $ pwd
current directory: /mnt/c/Users/suxto/Downloads/code
/mnt/c/Users/suxto/Downloads/code $ make
make: 'myshell' is up to date.
```

后台运行

在结尾加上 & 就能后台执行，后台执行就是不等命令执行完成就回到shell。

下面使用ping命令作为对比

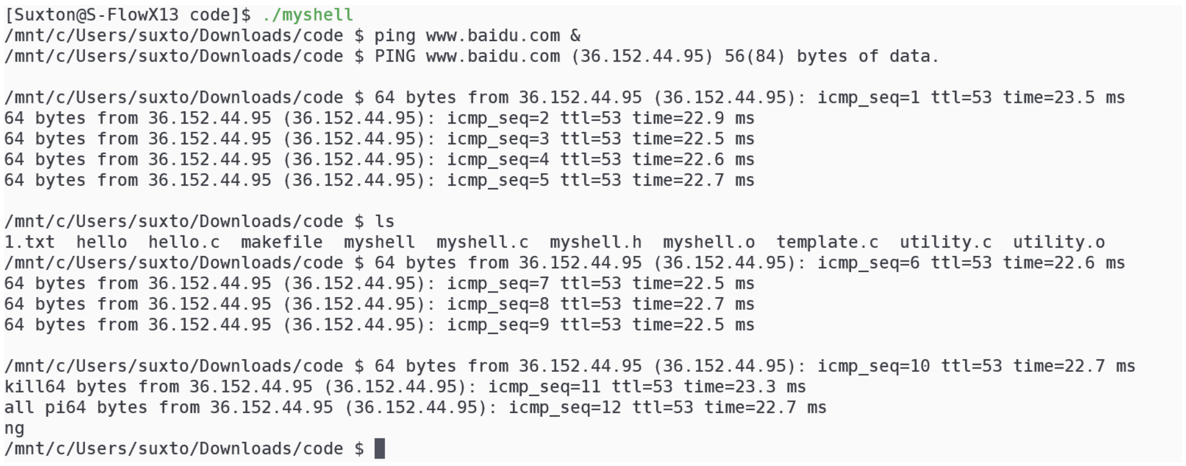
1. 不加 &

   一直没有回到myshell，输入的命令没有任何反应。

```
/mnt/c/Users/suxto/Downloads/code $ ping www.baidu.com
PING www.baidu.com (36.152.44.95) 56(84) bytes of data.
64 bytes from 36.152.44.95 (36.152.44.95): icmp_seq=1 ttl=53 time=23.6 ms
64 bytes from 36.152.44.95 (36.152.44.95): icmp_seq=2 ttl=53 time=22.5 ms


64 bytes from 36.152.44.95 (36.152.44.95): icmp_seq=3 ttl=53 time=22.7 ms


s64 bytes from 36.152.44.95 (36.152.44.95): icmp_seq=4 ttl=53 time=22.5 ms
dadasd
64 bytes from 36.152.44.95 (36.152.44.95): icmp_seq=5 ttl=53 time=24.4 ms
ls
64 bytes from 36.152.44.95 (36.152.44.95): icmp_seq=6 ttl=53 time=24.8 ms
64 bytes from 36.152.44.95 (36.152.44.95): icmp_seq=7 ttl=53 time=22.7 ms
64 bytes from 36.152.44.95 (36.152.44.95): icmp_seq=8 ttl=53 time=22.7 ms
64 bytes from 36.152.44.95 (36.152.44.95): icmp_seq=9 ttl=53 time=22.7 ms
^C64 bytes from 36.152.44.95: icmp_seq=10 ttl=53 time=22.7 ms
```

2. 加上 &

   开始执行之后马上回到 myshell， 而且可以对我输入的其他命令做出响应（图中我演示了 ls 和 killall）。我在 ping 执行的时候使用 killall 命令结束了 ping 的进程。



```
[Suxton@S-FlowX13 code]$ ./myshell
/mnt/c/Users/suxto/Downloads/code $ ping www.baidu.com &
/mnt/c/Users/suxto/Downloads/code $ PING www.baidu.com (36.152.44.95) 56(84) bytes of data.

/mnt/c/Users/suxto/Downloads/code $ 64 bytes from 36.152.44.95 (36.152.44.95): icmp_seq=1 ttl=53 time=23.5 ms
64 bytes from 36.152.44.95 (36.152.44.95): icmp_seq=2 ttl=53 time=22.9 ms
64 bytes from 36.152.44.95 (36.152.44.95): icmp_seq=3 ttl=53 time=22.5 ms
64 bytes from 36.152.44.95 (36.152.44.95): icmp_seq=4 ttl=53 time=22.6 ms
64 bytes from 36.152.44.95 (36.152.44.95): icmp_seq=5 ttl=53 time=22.7 ms

/mnt/c/Users/suxto/Downloads/code $ ls
1.txt  hello  hello.c  makefile  myshell  myshell.c  myshell.h  myshell.o  template.c  utility.c  utility.o
/mnt/c/Users/suxto/Downloads/code $ 64 bytes from 36.152.44.95 (36.152.44.95): icmp_seq=6 ttl=53 time=22.6 ms
64 bytes from 36.152.44.95 (36.152.44.95): icmp_seq=7 ttl=53 time=22.5 ms
64 bytes from 36.152.44.95 (36.152.44.95): icmp_seq=8 ttl=53 time=22.7 ms
64 bytes from 36.152.44.95 (36.152.44.95): icmp_seq=9 ttl=53 time=22.5 ms

/mnt/c/Users/suxto/Downloads/code $ 64 bytes from 36.152.44.95 (36.152.44.95): icmp_seq=10 ttl=53 time=22.7 ms
kill64 bytes from 36.152.44.95 (36.152.44.95): icmp_seq=11 ttl=53 time=23.3 ms
all pi64 bytes from 36.152.44.95 (36.152.44.95): icmp_seq=12 ttl=53 time=22.7 ms
ng
/mnt/c/Users/suxto/Downloads/code $ █
```

重定向输入输出

输入

使用 < 符号来指定重定向输入的文件， 此处使用cat命令测试

```
/mnt/c/Users/suxto/Downloads/code $ cat < 1.txt
echo hello
hi
ls
dir
```

输出

1. 增加文本

   使用 >> 符号指定输出的文件，输出方式为增加。

   使用内置命令echo

   ```
   /mnt/c/Users/suxto/Downloads/code $ cat 1.txt
   echo hello
   hi
   ls
   dir
   /mnt/c/Users/suxto/Downloads/code $ echo hello >> 1.txt
   /mnt/c/Users/suxto/Downloads/code $ cat 1.txt
   echo hello
   hi
   ls
   dir
   hello
   ```

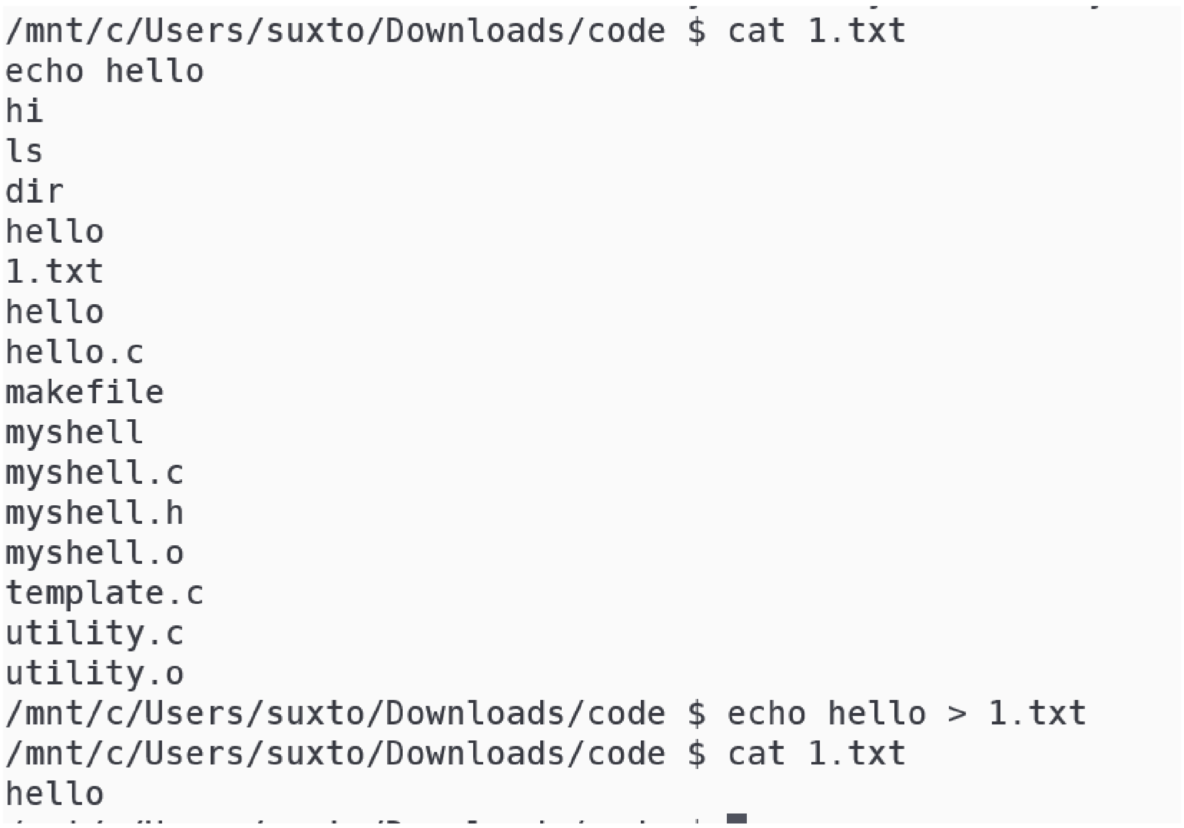
   使用外部命令ls

```
/mnt/c/Users/suxto/Downloads/code $ cat 1.txt
echo hello
hi
ls
dir
hello
/mnt/c/Users/suxto/Downloads/code $ ls >> 1.txt
/mnt/c/Users/suxto/Downloads/code $ cat 1.txt
echo hello
hi
ls
dir
hello
1.txt
hello
hello.c
makefile
myshell
myshell.c
myshell.h
myshell.o
template.c
utility.c
utility.o
```

2. 截断文本

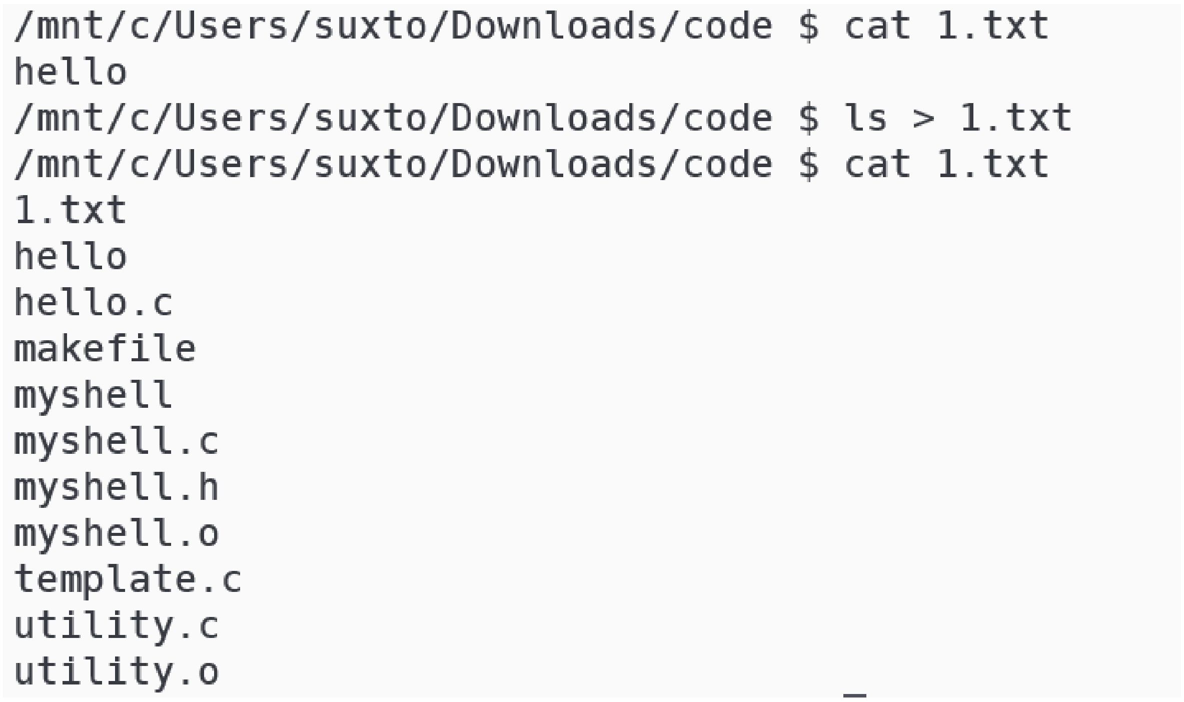
   使用 > 符号指定输出的文件，输出方式为截断。

   使用内置命令echo



```
/mnt/c/Users/suxto/Downloads/code $ cat 1.txt
echo hello
hi
ls
dir
hello
1.txt
hello
hello.c
makefile
myshell
myshell.c
myshell.h
myshell.o
template.c
utility.c
utility.o
/mnt/c/Users/suxto/Downloads/code $ echo hello > 1.txt
/mnt/c/Users/suxto/Downloads/code $ cat 1.txt
hello
```

使用外部命令ls

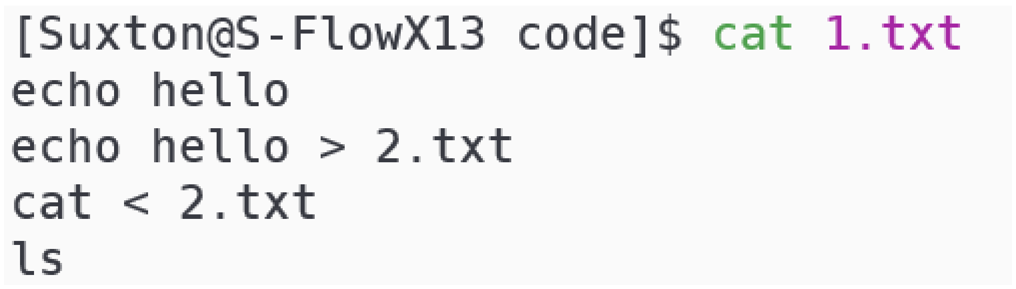


```
/mnt/c/Users/suxto/Downloads/code $ cat 1.txt
hello
/mnt/c/Users/suxto/Downloads/code $ ls > 1.txt
/mnt/c/Users/suxto/Downloads/code $ cat 1.txt
1.txt
hello
hello.c
makefile
myshell
myshell.c
myshell.h
myshell.o
template.c
utility.c
utility.o
```

执行批处理脚本

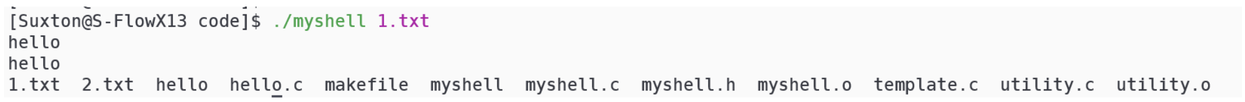
在myshell后面的参数中传入批处理文件就可以执行。

演示的脚本为1.txt



```
[Suxton@S-FlowX13 code]$ cat 1.txt
echo hello
echo hello > 2.txt
cat < 2.txt
ls
```

结果：



```
[Suxton@S-FlowX13 code]$ ./myshell 1.txt
hello
hello
1.txt  2.txt  hello  hello.c  makefile  myshell  myshell.c  myshell.h  myshell.o  template.c  utility.c  utility.o
```

代码

1. `makefile`

```
.PHONY: clear

CC = gcc
RM = rm
EXE = myshell
OBJS = myshell.o utility.o

$(EXE): $(OBJS)
	@$(CC) -o $(EXE) $(OBJS)
run: $(EXE)# clean
	@./$(EXE)
myshell.o: myshell.c myshell.h
	@$(CC) -o myshell.o -c myshell.c
utility.o: utility.c myshell.h
	@$(CC) -o utility.o -c utility.c
clear:
    @$(RM) $(EXE) $(OBJS)
clean:
    @$(RM) $(OBJS)
```

2. `myshell.c`

```
#include "myshell.h"
char elf_path[BUF_SIZE] = ":";//myshell可执行文件路径
char env[BUF_SIZE];
char path[BUF_SIZE];//当前工作路径
char cmd[BUF_SIZE];//当前命令

int main(int argc, char** argv) {
    int count = readlink("/proc/self/exe", elf_path + 1, BUF_SIZE - 1);//得到myshell的绝对路径
    elf_path[count - 6] = '\0';
    setenv("PATH", strcat(getenv("PATH"), elf_path), 1);//将文件目录加入环境变量
    // printf("export PATH=\"$PATH:%s\"", elf_path);
    // strcat(env, elf_path);
    // system(env);
    getcwd(path, BUF_SIZE);//得到当前工作路径

    if (argc == 1) {//没有参数，交互模式
        while (1) {
            printf("%s $ ", path);//展示当前路径
            fgets(cmd, BUF_SIZE, stdin);//读取输入
            if (cmd_analyzer(cmd) == 0)//分析命令行
                return 0;
        }
    }
    else {//有参数，执行批操作文件
        int fd = open(argv[1], O_RDONLY | O_CREAT, 0666);//重定向输入
        // close(fileno(stdin));
        close(STDIN_FILENO);
        // dup2(fd, fileno(stdin));
        dup2(fd, STDIN_FILENO);//将文件 重定向到stdin
        close(fd);
        while (1) {
            // printf("%s $ ", path);//展示当前路径
            if(fgets(cmd, BUF_SIZE, stdin)==NULL)break;//读取输入,遇到结尾退出
            if (cmd_analyzer(cmd) == 0)//分析命令行
                return 0;
        }
    }
    return 0;
}
```

3. `utility.c`

```
#include "myshell.h"
extern char path[BUF_SIZE];//当前工作路径
extern char elf_path[BUF_SIZE];//myshell可执行文
件路径
// FILE* myin;
// FILE* myout;


char* cmd_list[] = {//命令列表
    "exit",
    "quit",
    "cd",
    "pwd",
    "echo",
    "clr",
    "pause",
    "dir",
    "help",
    "environ"
};
int (*func_list[]) (char**) = {//命令函数列表
    &myexit,
    &myexit,
    &mycd,
    &mycd,
    &myecho,
    &myclr,
    &mypause,
    &mydir,
    &myhelp,
    &myenviron
};

int cmd_analyzer(char* cmd) {//指令分析
    char** divs = (char**)malloc(BUF_SIZE *
sizeof(char*));//建立指针数组，存入多个字符串
    char* word = strtok(cmd, DIV_CHARS);//得到第
一个字符
    char* in, * out;//重定向文件路径e
    in = out = NULL;
    // myin = stdin;//如果没有重定向默认为stdin
    // myout = stdout;//如果没有重定向默认为stdout
    int pos = 0;
    int singal = 0;
    int fd = STDOUT_FILENO;
    int fd_old = fd;
    while (word != NULL) {
        if (strcmp(word, "&") == 0) {//判断是否需
要后台运行
            singal |= BACKGROUND;
            break;
        }
        // else if (strcmp(word, "<") == 0) {//
判断是否需要重定向输入
        //     char* pre = word;
        //     word = strtok(NULL, DIV_CHARS);
        //     if (word == NULL) {
        //         divs[pos++] = pre;
        //         break;
        //     }
        //     else {
        //         int fd_in = open(word,
O_RDONLY | O_CREAT, 0666);
        //         int fd_in_old =
dup(STDIN_FILENO);
        //         dup2(fd_in, STDIN_FILENO);
        //         divs[pos] =
(char*)malloc(BUF_SIZE * sizeof(char));
        //         while (scanf("%s",
divs[pos]) != EOF) {
        //             pos++;
        //             divs[pos] =
(char*)malloc(BUF_SIZE * sizeof(char));
        //         }
        //         free(divs[pos]);
        //         dup2(fd_in_old,
STDIN_FILENO);
        //         close(fd_in);
        //         divs[pos] = NULL;
        //         word = strtok(NULL,
DIV_CHARS);
        //         continue;
        //     }
        // }
        divs[pos++] = word;
        word = strtok(NULL, DIV_CHARS);
    }
    divs[pos] = NULL;
    for (int i = 1;i < pos;i++) {//检查是否需要重
定向
        if (strcmp(divs[i], "<") == 0 && divs[i
+ 1] != NULL) {//重定向输入
            singal |= REIN;
            in = divs[i + 1];
        }
        else if (strcmp(divs[i], ">") == 0 &&
divs[i + 1] != NULL) {//重定向截断输出
            singal |= REOUT_TRUNC;
            out = divs[i + 1];
            // fd = open(divs[i + 1], O_WRONLY
| O_CREAT | O_TRUNC, 0666);
            // fd_old = dup(STDOUT_FILENO);
            // dup2(fd, STDOUT_FILENO);
        }
        else if (strcmp(divs[i], ">>") == 0 &&
divs[i + 1] != NULL) {//重定向增加输出
            singal |= REOUT_APPEND;
            out = divs[i + 1];
            // fd = open(divs[i + 1], O_WRONLY
| O_CREAT | O_APPEND, 0666);
            // fd_old = dup(STDOUT_FILENO);
            // dup2(fd, STDOUT_FILENO);
        }
        else continue;
        pos = i < pos ? i : pos;//重定向的内容不用
传参
        i++;
    }
    divs[pos] = NULL;//标记参数列表结尾
    if (divs[0] == NULL) {//输入为空
        free(divs);
        return 1;
    }
    for (int i = 0;i < CMD_NUM;i++) {//分析是第几
个内置函数
        if (!strcmp(divs[0], cmd_list[i])) {
            int status;//保存执行结果
            if (singal & REOUT_TRUNC) {//重定向
截断输出
                fd = open(out, O_WRONLY |
O_TRUNC | O_CREAT, 0666);
                fd_old = dup(STDOUT_FILENO);
                close(fileno(stdout));
                dup2(fd, fileno(stdout));
                close(fd);
            }
            else if (singal & REOUT_APPEND) {//
重定向增加输出
                fd = open(out, O_WRONLY |
O_APPEND | O_CREAT, 0666);
                fd_old = dup(STDOUT_FILENO);
                close(fileno(stdout));
                dup2(fd, fileno(stdout));
                close(fd);
            }
            status = ex_cmd(i, divs);
            free(divs);
            if (fd_old != STDOUT_FILENO) {//如果
重定向，输出恢复到stdio
                dup2(fd_old, STDOUT_FILENO);
                close(fd);
            }
            return status;
        }
    }
    not_buildin_cmd(singal, divs, in, out);
    free(divs);//释放内存
    return 1;
}
int ex_cmd(int n, char** args) {//前台执行命令
    return (*func_list[n])(args);
}

int not_buildin_cmd(int singal, char** args,
char* in, char* out) {//外部命令
    pid_t pid = fork();
    int statue = 1;
    if (pid == 0) {//子进程
        if (singal & REOUT_TRUNC) {//重定向截断输
出
            int fd = open(out, O_WRONLY |
O_TRUNC | O_CREAT, 0666);
            // close(fileno(stdout));
            close(STDOUT_FILENO);
            // dup2(fd, fileno(stdout));
            dup2(fd, STDOUT_FILENO);
            close(fd);
        }
        else if (singal & REOUT_APPEND) {//重定
向增加输出
            int fd = open(out, O_WRONLY |
O_APPEND | O_CREAT, 0666);
            // close(fileno(stdout));
            close(STDOUT_FILENO);
            // dup2(fd, fileno(stdout));
            dup2(fd, STDOUT_FILENO);
            close(fd);
        }
        if (singal & REIN) {//重定向输入
            int fd = open(in, O_RDONLY |
O_CREAT, 0666);
            // close(fileno(stdin));
            close(STDIN_FILENO);
            // dup2(fd, fileno(stdin));
            dup2(fd, STDIN_FILENO);
            close(fd);
        }
        execvp(args[0], args);
        // char** new_env = { __environ };
        // execve(args[0], args, __environ);
        // execv(args[0], args);
        perror("execvp error");
        exit(EXIT_FAILURE);
    }
    else if (pid < 0) {
        perror("fork error");
    }
    else {
        if (singal & BACKGROUND) return 1;//后台执行
        else waitpid(pid, &statue, 0);//前台执行，等待子进程完成
    }
    // return statue;
    return 1;
}

int mycd(char** args) {//cd命令
    if (args[1] == NULL) {//没有参数，就输出当前目录
        printf("current directory: %s\n", path);
    }
    else {//有参数选择目录
        if (chdir(args[1]) != 0) perror("cd error");//打开失败，显示错误信息
        else getcwd(path, BUF_SIZE);//打开成功，更新path
    }
    return 1;
}

int myexit(char** args) {//返回0表示退出
    return 0;
}

int myecho(char** args) {//echo命令
```

```
    int pos = 1;
    while (args[pos] != NULL) {//按序输出每个参数
        printf("%s ", args[pos++]);
    }
    printf("\n");
    return 1;
}

int myclr(char** args) {//清楚屏幕
    system("clear");
    return 1;
}

int mypause(char** args) {//暂停命令
    puts("Paused, press enter to continue");
    while (getchar() != '\n');
    return 1;
}


int mydir(char** args) {//dir命令
    return not_buildin_cmd(0, args, NULL,
NULL);
}

int myhelp(char** args) {//help命令
    char* helps = "This is a shell developed by
Suxton\n\
        You can use this shell as other
shells\n\
        There are several build-in functions in
this shell\n\
        cd <directory>: select a directory. It
would show current directory if you invoke
whout arguments\n\
        clr : clear the screen\n\
        dir <directory>: - List the contents of
directory <directory>.\n\
        environ : List all the environment
strings.\n\
        echo <comment> : Display <comment> on
the display followed by a new line\n\
        help : Display this user manual using
the more filter.\n\
        pause : Pause operation of the shell
until 'Enter' is pressed.\n\
        quit : Quit the shell.\n\
        ues < to redirect stdin\n\
        use > to redirect stdout to a file with
trunc\n\
        use >> to redirect stdout to a file
with append\n";//我的终端没有中文字体，所以都是英文
    FILE* pipe = popen("more", "w");//用管道传给
more
    fprintf(pipe, "%s", helps);
    pclose(pipe);
    return 1;
}

int check_char(char ch) {//排除变色控制字符
```

```
    return (isalpha(ch) || isalnum(ch) || ch ==
':' || ch == '/' ||
        ch == '_' || ch == '=' || ch == '-' ||
        ch == '.' || ch == '*' || ch == ';' ||
        ch == '%' || ch == '\t' || ch == '\n'
||
        ch == ' ' || ch == '\r'); //return 1;
    // return 0;
}
int myenviron(char** args) {//environ命令
    int cnt = 0;
    while (__environ[cnt] != NULL) {
        // int bound = strlen()
        for (int i = 0;;i++) {
            if (check_char(__environ[cnt][i]))
                putchar(__environ[cnt][i]);
            else break;
        }
        cnt++;
        putchar('\n');
    }
    return 1;
}
```

4. `myshell.h`

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

```c
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <ctype.h>
#include <sys/types.h>
#include <sys/wait.h>
//常量
#define CMD_NUM 10 //含有的命令
#define BUF_SIZE 256//缓冲区大小
#define DIV_CHARS " \n\t\r"

//运行参数
//后台运行
#define BACKGROUND  1
//重定向输入
#define  REIN 2
//重定向截断输出
#define  REOUT_TRUNC  4
//重定向增加输出
#define  REOUT_APPEND 8

//工具
int cmd_analyzer(char*);//命令分析
int ex_cmd(int, char**);//命令执行
int not_buildin_cmd(int, char**, char*, char*);
//外部命令

//内建命令
int myexit(char**);
int mycd(char**);
int myecho(char**);
int myclr(char**);
int mypause(char**);
int mydir(char**);
int myhelp(char**);
int myenviron(char**);
```