选课序号：80 学号：2220213301 姓名：邹康

实验二：Linux 命令环境下 C/C++语言实践

1. **实验目的** 

用 C/C++构造一个简单的 shell； 

理解 shell 程序的功能； 

学会 shell 的使用； 

掌握基本的 makefile 方法。

1. **实验内容 **

**基本任务 1：**用 C/C++编写一个简单的 shell 程序，实现以下基本的命令。

1) 浏览目录和文件的各种属性 ls（可以不支持参数）

2) 回显命令 echo

3) 显示文件内容 cat

4) 创建目录 mkdir

5) 删除文件 rm

6) 切换目录 cd

7) 显示当前目录 pwd

8) 文字统计 wc 

**基本任务 2：**每一条命令单独对应一个源程序文件，不允许所有命令一个源文件。

**基本任务 3：**写一个 makefile 来管理这些源文件。 

**基本任务 4**：写清楚 make 编译和运行的过程。 

1. **实验步骤**
2. 编写Shell主程序: 编写主程序，包括主要的Shell循环，用于读取用户输入和执行相应的命令。

void myshell\_loop()

{

    char \*line;

    char \*\*args;

    int status;

    printf("-----------------欢迎来到我的shell--------------\n ");

    printf("-------可以通过help命令获取帮助\n ");

    do

    {

        getcwd(curpath, sizeof(curpath));

        getlogin\_r(username, sizeof(username));

        gethostname(hostname, sizeof(hostname));

        printf("\e[32;1m%s@%s\e[0m:%s$ ", username, hostname,curpath); // 显示为绿色

        line = myshell\_readline();

        args = myshell\_splitline(line);

        status = mysh\_execute(args);

    } while (status);

}

int main(int argc, char \*argv[])

{

    myshell\_loop();

    return 0;

}

1. 实现命令解析: 编写代码来解析用户输入的命令，包括分解命令、参数和选项。

char \*\* myshell\_splitline(char \* line)

{

    int buffer\_size = MYSH\_BUFFER\_SIZE, position = 0;

    char \*\*tokens = malloc(buffer\_size \* sizeof(char \*));

    char \*token;

    token = strtok(line, MYSH\_TOK\_DELIM);

    while(token != NULL)

    {

        tokens[position++] = token;

        //传递NULL作为第一个参数，以表示它应该继续使用相同的字符串（`line`）来提取下一个令牌。

        token = strtok(NULL, MYSH\_TOK\_DELIM);

    }

    tokens[position] = NULL;

    return tokens;

}

1. 检查是否含有输出重定向，还是只有命令的执行

 // 检查是否有输出重定向符号 ">"

    for (i = 0; args[i] != NULL; i++) {

        if (strcmp(args[i], ">") == 0) {

            if (args[i + 1] == NULL) {

                perror("Missing output file name.\n");

                return 1;

            }

            output\_redirect = i;//第几个分词是重定向符号

            for ( int j = 0; j < output\_redirect; j++)

            {

                args\_a[j] = args[j];

            }

            output\_filename = args[i + 1]; // 下一个参数是输出文件名

            break;

        }

    }

1. 实现对命令的实现和执行：包括ls，cat，help，exit，echo，touch，rm，mkdir，rmdir等

//执行命令

int mysh\_without\_execute(char \*\*args)

{

    if (args[0] == NULL) {

    return 1;

} else if (strcmp(args[0], "help") == 0) {

    return mysh\_help(args);

} else if (strcmp(args[0], "exit") == 0) {

    return mysh\_exit(args);

} else if (strcmp(args[0], "ls") == 0) {

    return mysh\_ls(args);

} else if (strcmp(args[0], "cat") == 0) {

    return mysh\_cat(args);

} else if (strcmp(args[0], "echo") == 0) {

    return mysh\_echo(args);

} else if (strcmp(args[0], "mkdir") == 0) {

    return mysh\_mkdir(args);

} else if (strcmp(args[0], "rmdir") == 0) {

    return mysh\_rmdir(args);

} else if (strcmp(args[0], "touch") == 0) {

    return mysh\_touch(args);

} else if (strcmp(args[0], "rm") == 0) {

    return mysh\_rm(args);

} else if (strcmp(args[0], "pwd") == 0) {

    return mysh\_pwd(args);

} else if (strcmp(args[0], "wc") == 0) {

    return mysh\_wc(args);

} else if (strcmp(args[0], "cd") == 0) {

    return mysh\_cd(args);

} else {

    // 处理未知命令或错误

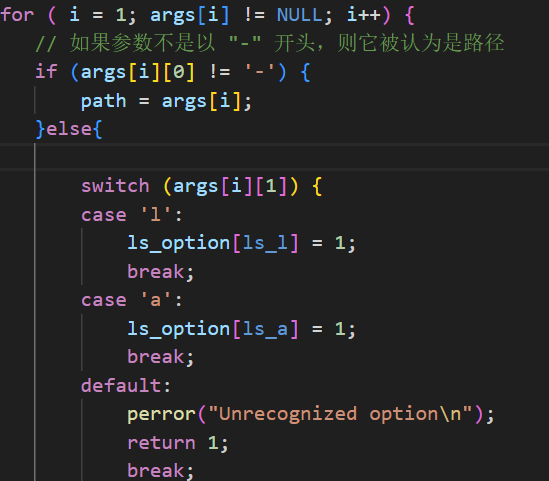
    perror("Unknown command\n");

    return 1;

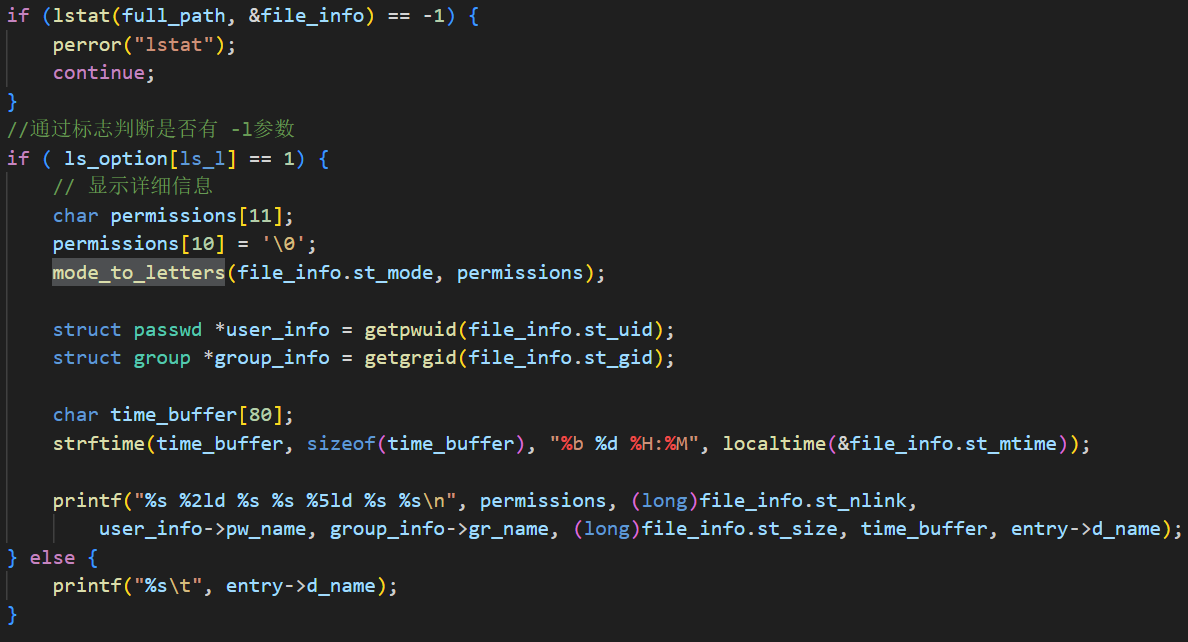
}

1. 各命令实现
2. Ls（-a，-l）

初始化ls\_option数组，用于表示-l和-a选项是否被指定。默认情况下，它们都被设置为0。

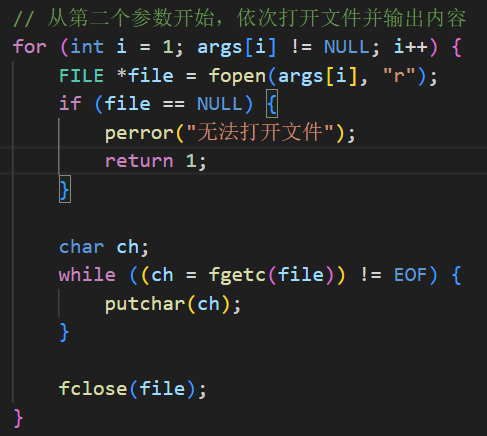


调用list\_files函数，传递path和ls\_option数组作为参数，以列出文件和目录使用opendir函数打开指定路径，如果路径不存在，使用perror函数输出错误信息并返回。使用readdir函数遍历目录中的文件和子目录。  


使用lstat函数获取文件的详细信息,并编写一个函数mode\_to\_letters将用户权限转化为字符串，如果不需要显示详细信息，仅输出文件或目录的名称。  


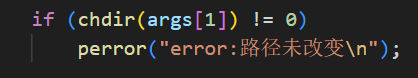
1. cat

使用fopen函数打开文件,使用fgetc函数逐个字符读取文件内容，直到遇到文件末尾（EOF）,通过循环可以读取多个文件。但是不能不输入参数，等待用户输入。



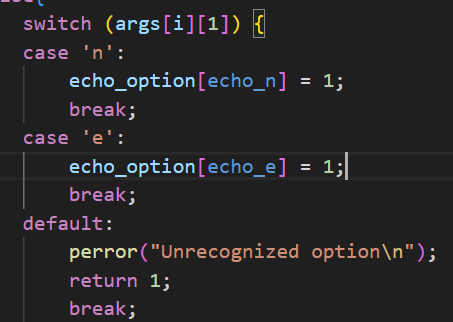
1. cd

使用了chdir函数来尝试切换目录

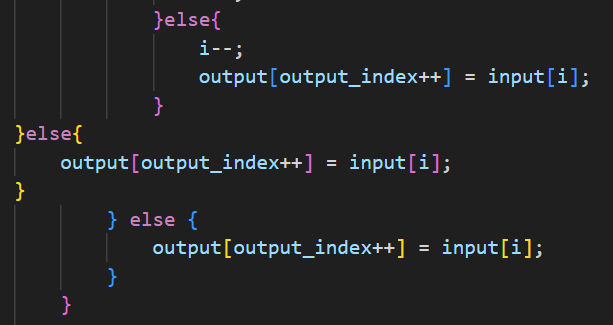
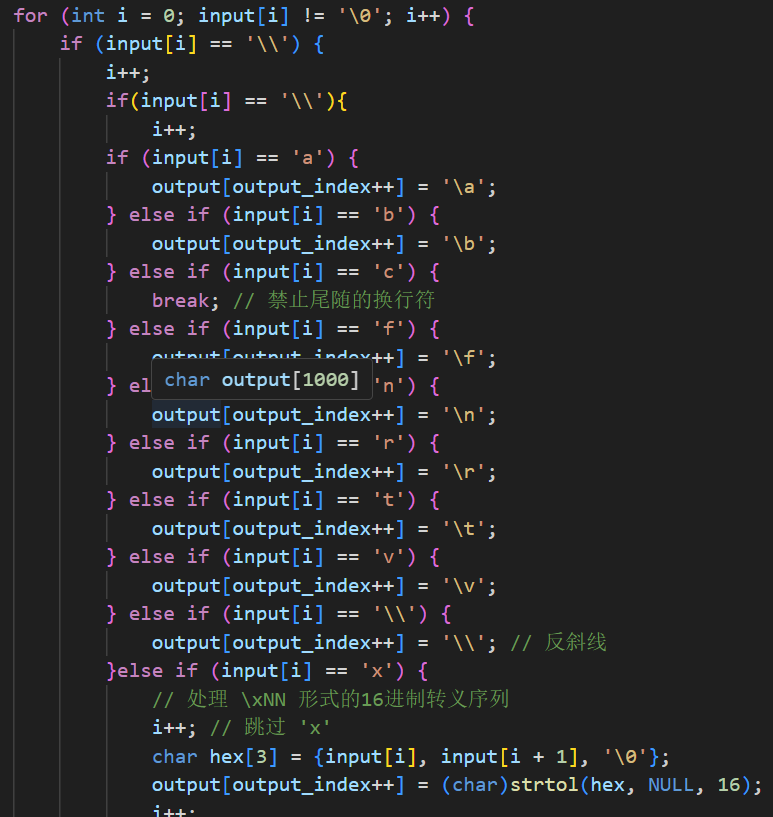


1. echo

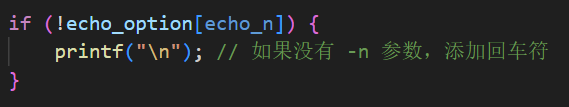
初始化echo\_option数组，用于表示-n和-e选项是否被指定。默认情况下，它们都被设置为0。



编写解释和打印包含转义字符的字符串的函数

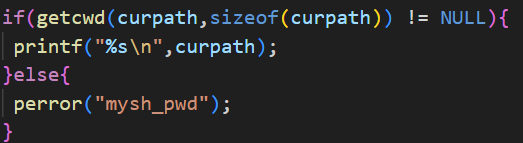


如果没有-n选项，输出换行符以结束行

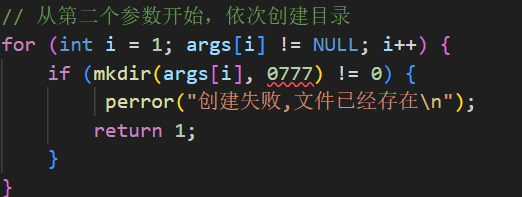


1. pwd

使用getcwd函数来获取当前工作目录的路径,获取成功printf函数来输出当前工作目录的路径。

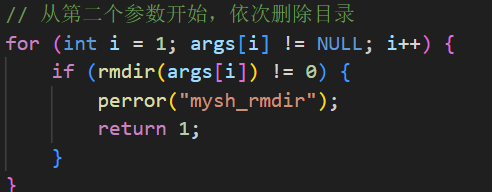


1. mkdir  
   使用mkdir函数来尝试创建目录。默认情况下，使用0777权限，可以创建多级目录。



1. rmdir

使用rmdir函数来尝试删除目录



1. touch

使用open函数来尝试创建文件，给默认权限0666.



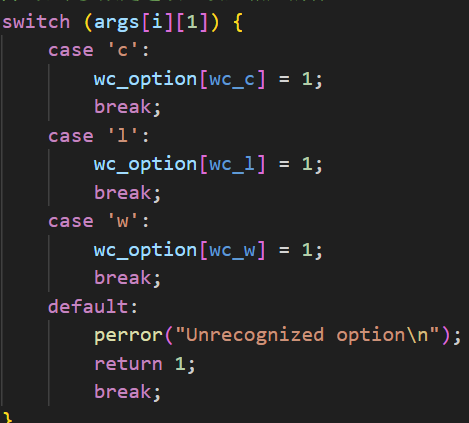
1. rm

使用unlink函数来尝试删除文件

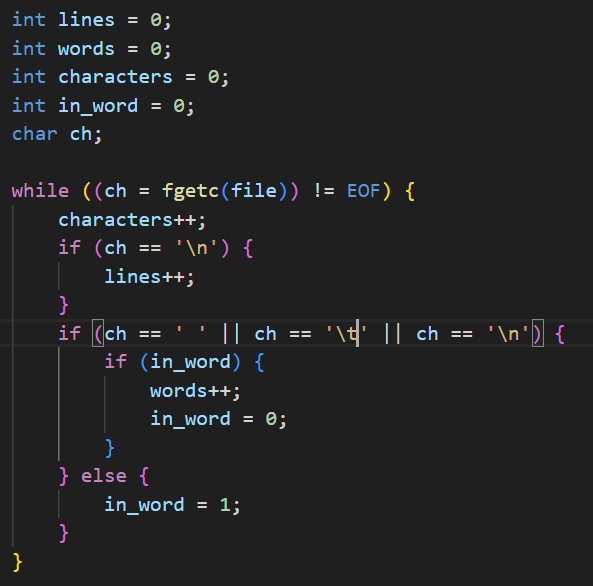


1. wc（-c ,-l ,-w）

检查命令行参数是否以-开头，如果是，则根据参数设置相应的标志给wc\_option。



对文件内容进行处理，使用fopen打开文件，初始化字符数、行数、单词数和一个标志变量in\_word，以便跟踪当前是否在一个单词内。在循环中，逐字符读取文件内容并统计字符数、行数和单词数。最后，关闭文件，并在根据选项打印统计结果



1. 编写帮助文档: 创建帮助文档，以解释每个命令的使用方法和选项，以及每个命令的注意事项。

  printf("Welcome to Help of My Shell !\n");

       printf("-------------------\n");

       printf("Command List as Following\n");

       printf("1)  查看命令使用 help [commond]\n"

        "2) 回显命令 echo\n"

        "3) 显示文件内容 cat\n"

        "4) 创建目录 mkdir\n"

        "5) 删除目录 rmdir \n"

        "6) 创建文件 touch\n"

        "7) 删除文件 rm\n"

        "8) 切换目录 cd\n"

        "9) 显示当前目录 pwd\n"

        "10) 文字统计 wc\n"

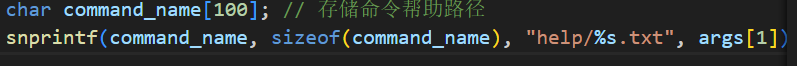
        "11) 结束程序 exit\n"

        "12)浏览目录和文件的各种属性 ls\n"

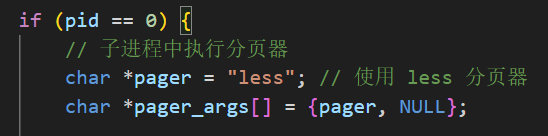
        "13) 输出重定向的使用 cdx\n");

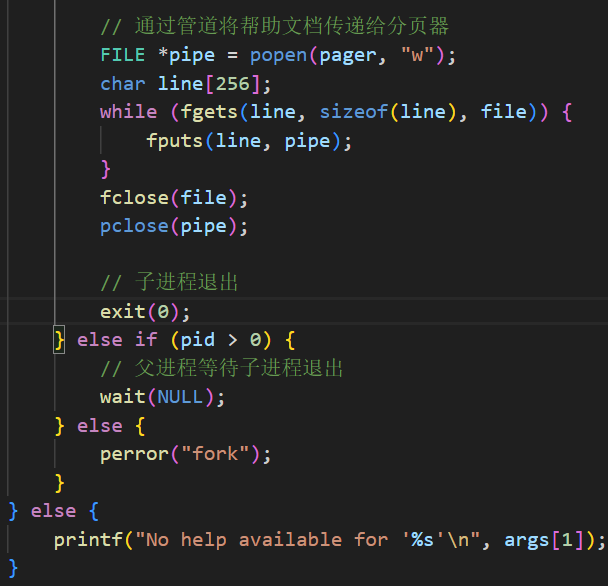
当输入是命令帮助，就要显示相关命令文档

使用snprintf函数构建帮助文档的路径。帮助文档以help/[command\_name].txt的形式存储



使用fork函数创建一个子进程。在子进程中，将执行分页器来显示帮助文档。分页器使用less。使用popen函数创建一个管道，以便将帮助文档传递给分页器。从帮助文档文件中逐行读取内容，并将每行内容写入管道，以便在分页器中显示。





1. 通过makefile管理源文件

CC = gcc：这行代码定义了CC变量，指定了使用的C编译器。

target = 3301\_mysh：这行定义了target变量，指定了要生成的可执行文件的名称3301\_mysh。

src = $(wildcard \*.c)：这行使用wildcard函数查找当前目录中的所有.C文件，并将它们赋值给src变量。

objects = $(patsubst %.c,%.o,$(src))：这行生成了一个对象文件列表，通过将源文件的扩展名从.c替换为.o来实现。这些对象文件被赋值给objects变量。

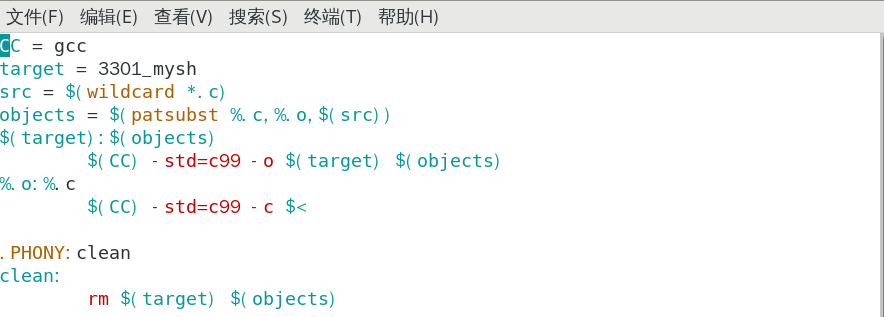
$(target): $(objects)：这行规定target目标文件依赖于objects（.c文件）

$(CC) -std=c99 -o $(target) $(objects)：这行是构建命令，使用CC变量（GCC）编译objects中的C源文件，使用C99标准生成target可执行文件。

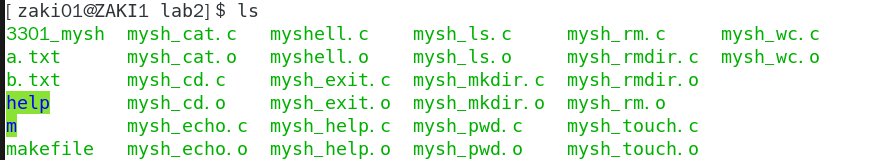
%.o: %.c：C源文件（%.c）编译为对象文件（%.o）。它使用CC变量和C99标准来编译每个源文件为一个对象文件。

.PHONY: clean：这行规定了一个"phony"目标，名为clean，用于执行清理操作。在这种情况下，它用于清理生成的可执行文件和对象文件。

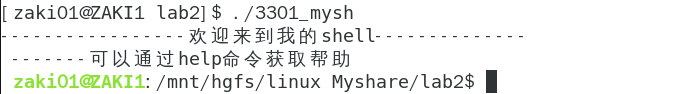
clean：此规则指定了删除target和所有objects文件的命令，用于清理项目。



1. make编译，生成可执行文件3301\_mysh



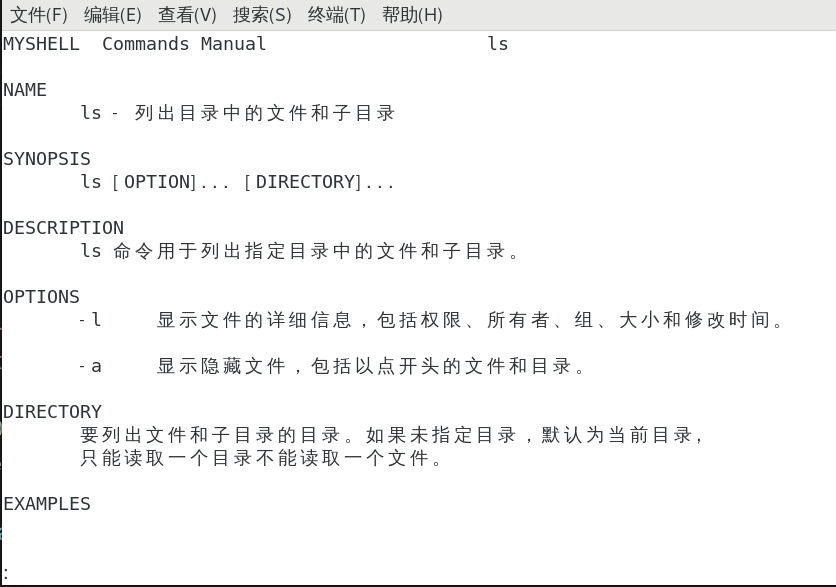
1. **实验结果：**
2. 运行make进行编译 --> ./3301\_mysh



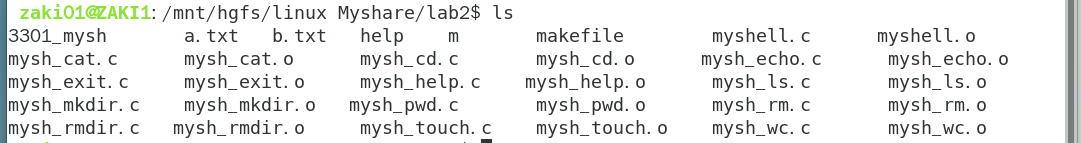
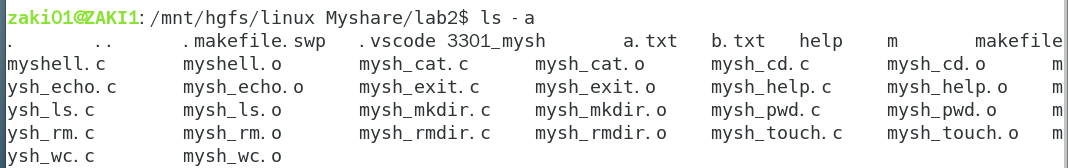
1. 查看帮助文档help（用户手册），里面包含用户注意事项和命令的使用方法：

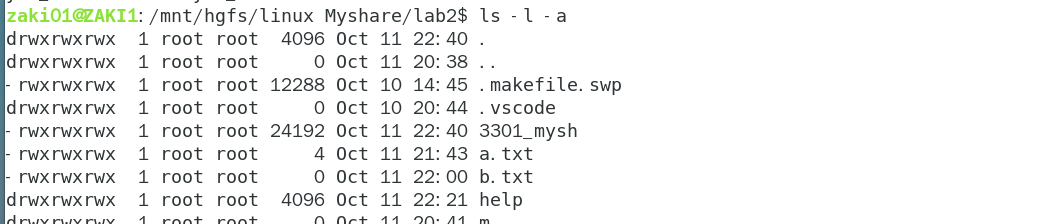
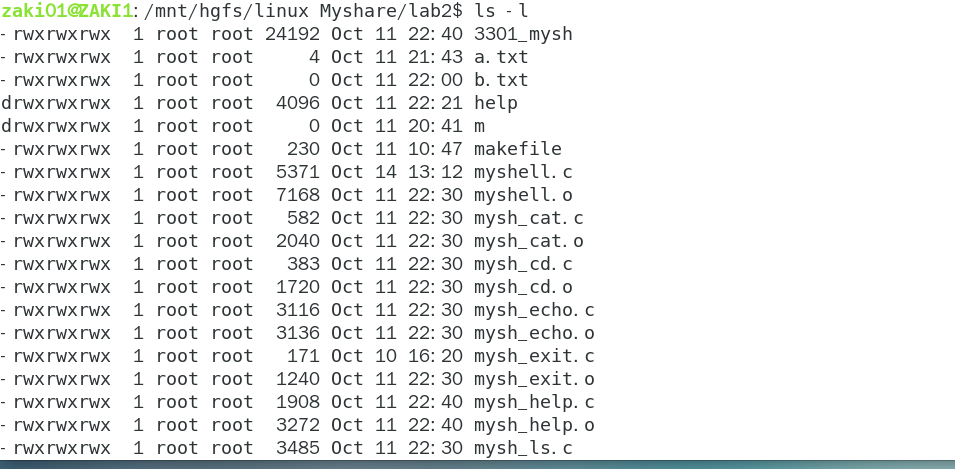


1. 查看命令使用帮助 help ls/.... （按q退出查看）

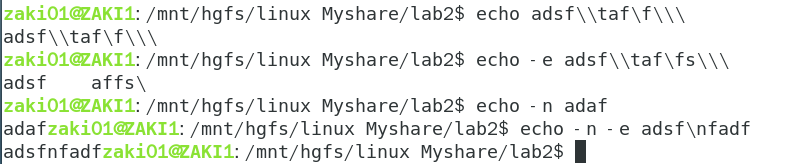


1. 执行ls 命令（ls，ls-l，ls-a）

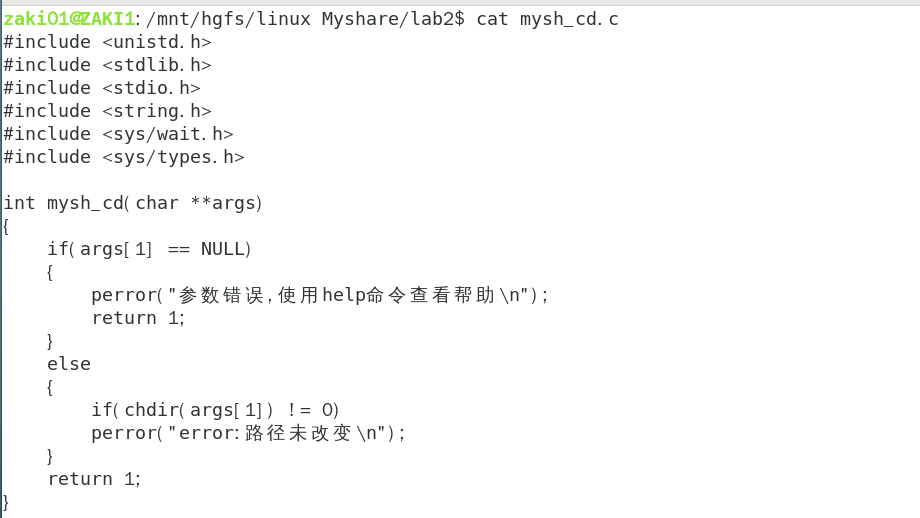
  


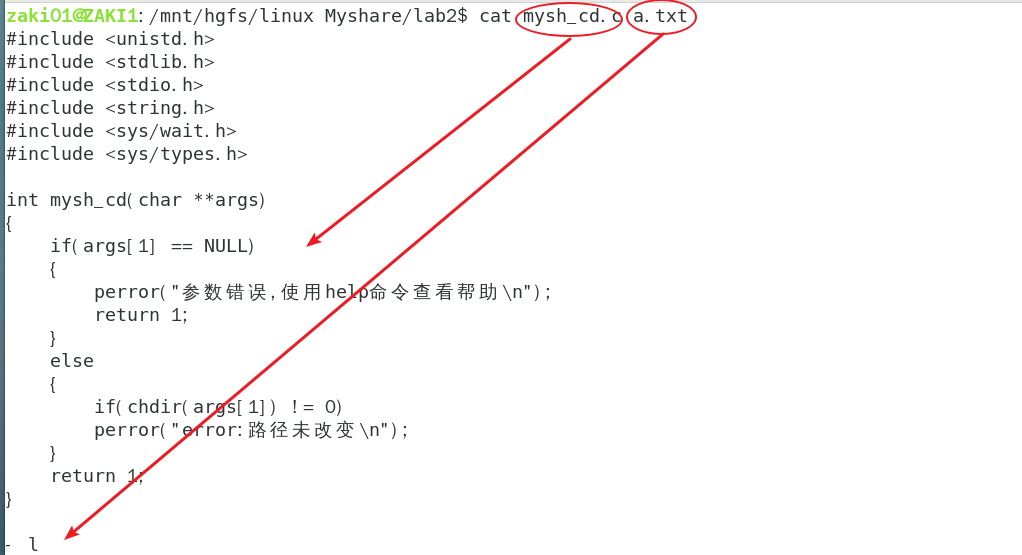


4.echo命令(echo -n /-e)：

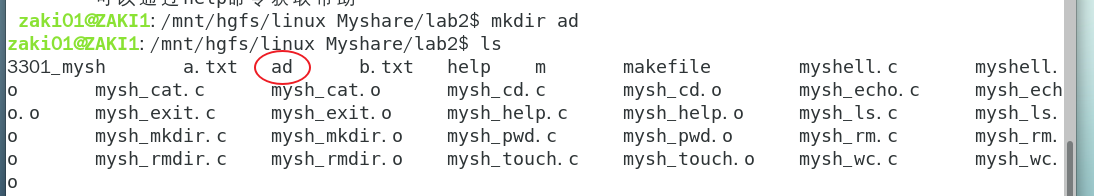


5.cat命令：

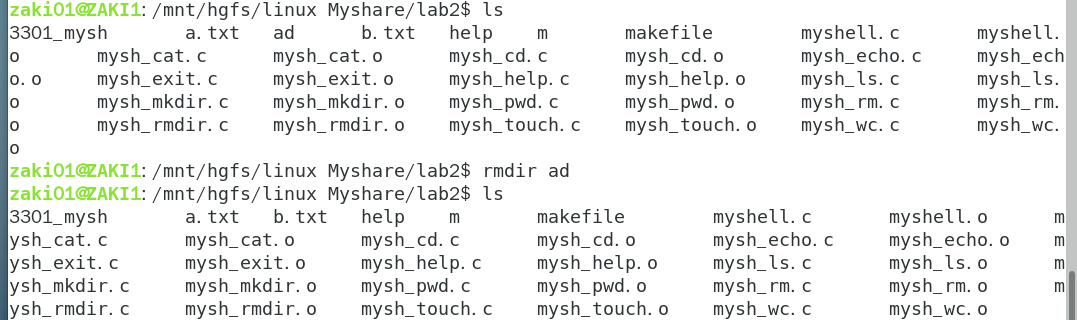




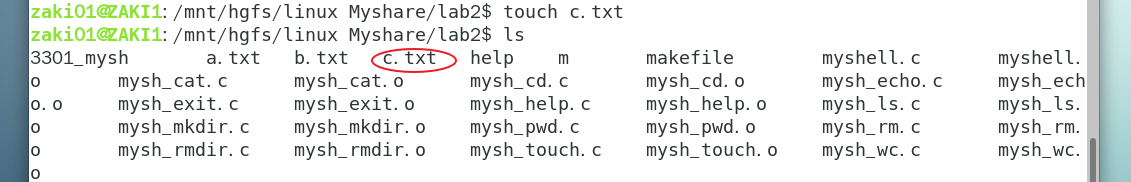
6.mkdir命令：



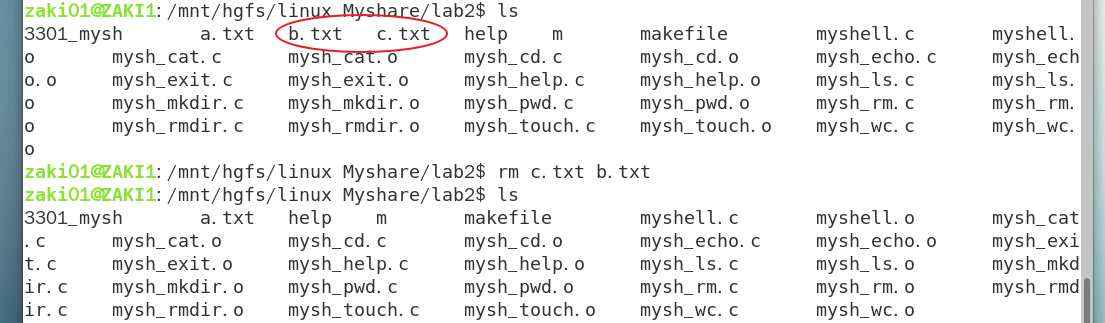
7.rmdir命令：



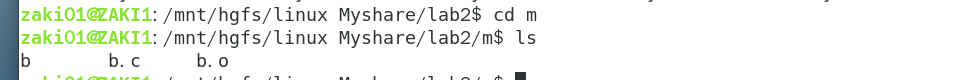
8.touch命令：



9.rm命令：



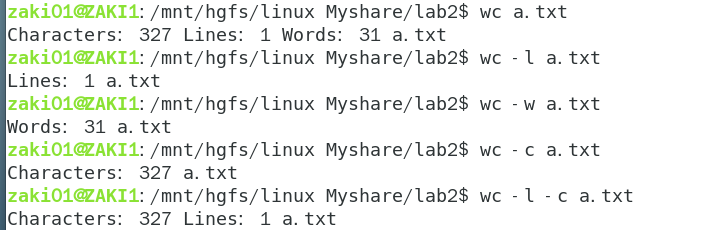
1. cd命令：



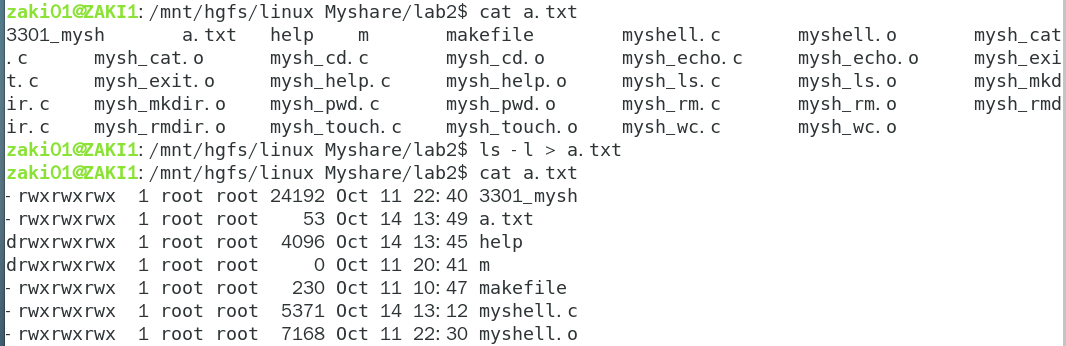
1. pwd命令：



1. wc命令(wc -c /-l /-w)：



1. 实现输出重定向>：



1. 退出shell:



15.使用make clean 清除.o文件和目标文件

1. **实验分析**

本次实验的主要目标是通过编程方式深入理解基本命令的实现原理和工作机制。通过实验，我成功创建了一个包含多个基本命令的自定义Shell程序，并编写了相关代码以支持这些命令的功能。这包括内置命令（如`cd`，`echo`，`exit`，`help`）以及外部命令的执行。

通过使用Makefile，我能够有效地管理多个源文件并将它们编译成一个可执行的Shell程序。这提高了我对Linux编译和构建管理的理解，并让我能够更好地组织和维护我的代码。

总的来说，这次实验增强了我对Linux Shell命令和编程的理解，通过自己编写Shell程序，我深入了解了各种命令的功能和使用方法。这将有助于我更好地理解和利用Linux系统中的命令行工具，并提升了我的编程技能和Linux系统管理能力。