实验1 ls命令的实现

# 实验目的

Shell编程是linux编程的核心内容，Shell命令的理解和实现是理解Shell机制的最直接的方式，本实验要求用C语言编写和调试一个ls命令的实现程序。以达到理解Shell机制的目的。要求实现至少4个ls下的命令。

# 实验内容

通过阅读linux相关文档，以及使用man命令对ls的学习，了解到ls的一些基础命令，如 -a列出所有文件 -f不排序的-a -l显示所有信息，一行一个文件 -R 以递归的方式显示所有文件 -C按列排序显示 ``-x按行的方式输出 等等命令，并实现这些命令，只是用linux下文件操作如stat等函数的调用等。

# 实验步骤

* 首先了解每一个中命令的具体含义，分析每一个命令所管理的部分内容
* 分别编写每一个命令的代码，使用自定义的命令变量来确定执行的命令，依次来实现多个命令的同时执行效果
* 编写测试程序

## 任务分析

* 输入形式： ls [option]...[files]... 即ls命令后跟随具体的子命令和路径，不输入为默认当前目录并执行 ls -a ./
* 输出形式： 根据命令的不同，输出形式不同，如 ls -a 将输出所有的文件的文件名； ls -l 将输出每一个文件以及具体的信息，文件类型、权限、大小、修改时间、文件名等等内容； ls -R 将递归的输出每一个目录下的文件信息；此外多个命令的组合输出的内容也不同，如 ls -l -R ./ 将递归的输出当前目录下所有文件，每一个文件将输出所有信息。
* 程序达到的功能：基本实现ls命令的对应子命令

## 概要设计

### 主程序入口

main.h 将实现整个程序的入口，用来接受参数，并设置值，然后调用 myls方法

### 主要方法实现

* 首先要自定义一些命令的参数，用不同的位来表示当前接受到的子命令的情况，如 MYLS\_A 1 表示 myls -a 的情况等等
* 定义一个用于存储所有结果的结构体，方便后续排序输出：

// 为了能够实现对某些命令的排序功能，所以不能使用直接输出的方式，先要存下来  
typedef struct FILE\_NODE{  
 char file\_type; // 文件类型，包括普通文件、目录文件、链接文件等等  
 char file\_mode[9]; // 文件的各个权限  
 int st\_nlink; // 链接文件的个数  
 char uname[128]; // 用户名  
 char gname[128]; // 组名  
 int file\_size; // 文件大小  
 char date[128]; // 文件的修改日期  
 char file\_name[128]; // 文件名  
 bool islink; // 是否是链接文件  
 char file\_link[128]; // 链接文件指向的文件名  
}FILE\_NODE;

* 对于纯文件，使用 stat 函数来获取其所有的信息，并存储即可； 对于文件夹文件，使用 opendir 和 readdir 等函数的组合使用来对文件夹信息读取；
* 对于递归的读取，要使用一个递归函数来读取，当读到一个文件夹时，首先要保存文件夹的信息，然后等当前文件夹显示完毕后再对本层所有文件夹进行递归读取即可
* 格式化输出：针对不同的命令要根据上面的命令参数来格式化输出，确定输出的内容以及输出的格式（按列排序还是按行排序等）

## 详细设计

程序流程图：

## 调试分析、测试结果

一些命令的展示：

* 无参数时：

x31415@iZ2zea1nnstgr3h5ccfovoZ:~/linux/myls$ ./myls   
main.c main.o makefile myls myls.c myls.h myls.o

* ./myls --help ：

x31415@iZ2zea1nnstgr3h5ccfovoZ:~/linux/myls$ ./myls --help  
Here is the usage of 'myls':   
Usage: ls [OPTiON]...[FILE]...  
  
<-l>: --list use a long listing format 输出文件的所有信息  
<-a>: --all do not ignore entries starting with. 不忽略隐藏文件  
<-f>: do not sort 不排序  
<-R>: --recursive list subdirectories recursively 以递归的方式显示所有文件  
<-C>: list entries by columns 按列的方式输出（默认）  
<-x>: list entries by lines instead of by columns 按行的方式输出

* myls -a

x31415@iZ2zea1nnstgr3h5ccfovoZ:~/linux/myls$ ./myls -a  
. .. main.c main.o makefile myls myls.c myls.h myls.o

* myls -l

x31415@iZ2zea1nnstgr3h5ccfovoZ:~/linux/myls$ ./myls -l  
-rwxrw-r-- 1 x31415 x31415 1334 Jun 11 13:29 main.c   
-rwxrw-r-- 1 x31415 x31415 4208 Jun 11 13:29 main.o   
-rwxrw-r-- 1 x31415 x31415 328 Jun 07 19:52 makefile   
-rwxrwxr-x 1 x31415 x31415 22624 Jun 11 13:29 myls   
-rwxrw-r-- 1 x31415 x31415 9913 Jun 11 13:38 myls.c   
-rwxrw-r-- 1 x31415 x31415 2824 Jun 10 19:58 myls.h   
-rwxrw-r-- 1 x31415 x31415 16480 Jun 10 22:32 myls.o

* myls -R

x31415@iZ2zea1nnstgr3h5ccfovoZ:~/linux/myls$ ./myls -R ../  
/../:  
ipc myls pipe socket   
  
/../ipc/:  
Semaphore msg sharecache socket   
  
/../ipc/Semaphore/:  
main main.c sharecache socket   
  
/../ipc/msg/:  
msgrecive msgrecive.c msgsend msgsend.c   
  
/../ipc/sharecache/:  
shmrecive shmrecive.c shmsend shmsend.c   
  
/../myls/:  
main.c main.o makefile myls myls.c myls.h myls.o   
  
/../pipe/:  
A A.c B B.c C C.c pipe pipe.c   
  
/../socket/:  
client server B B.c C C.c pipe pipe.c   
  
/../socket/client/:  
client client.c client.o makefile C C.c pipe pipe.c   
  
/../socket/server/:  
makefile server server.c server.o C C.c pipe pipe.c

* myls -R -l ../

x31415@iZ2zea1nnstgr3h5ccfovoZ:~/linux/myls$ ./myls -R -l ../  
/../:  
drwxrwxr-x 5 x31415 x31415 4096 Jun 21 20:26 ipc   
drwxrwxrwx 2 x31415 x31415 4096 Jun 21 21:50 myls   
drwxrwxr-x 2 x31415 x31415 4096 Jun 18 16:42 pipe   
drwxrwxr-x 4 x31415 x31415 4096 Jun 11 18:26 socket   
  
/../ipc/:  
drwxrwxr-x 2 x31415 x31415 4096 Jun 21 20:22 Semaphore   
drwxrwxr-x 2 x31415 x31415 4096 Jun 21 20:25 msg   
drwxrwxr-x 2 x31415 x31415 4096 Jun 21 20:57 sharecache   
  
/../ipc/Semaphore/:  
-rwxrwxr-x 1 x31415 x31415 13192 Jun 21 20:22 main   
-rwxrw-r-- 1 x31415 x31415 2257 Jun 21 20:22 main.c   
  
/../ipc/msg/:  
-rwxrwxr-x 1 x31415 x31415 8704 Jun 21 19:49 msgrecive   
-rwxrw-r-- 1 x31415 x31415 742 Jun 21 19:50 msgrecive.c   
-rwxrwxr-x 1 x31415 x31415 8792 Jun 21 19:50 msgsend   
-rwxrw-r-- 1 x31415 x31415 790 Jun 21 19:50 msgsend.c   
  
/../ipc/sharecache/:  
-rwxrwxr-x 1 x31415 x31415 12944 Jun 21 20:57 shmrecive   
-rwxrw-r-- 1 x31415 x31415 1283 Jun 21 20:57 shmrecive.c   
-rwxrwxr-x 1 x31415 x31415 12984 Jun 21 20:51 shmsend   
-rwxrw-r-- 1 x31415 x31415 1200 Jun 21 20:48 shmsend.c   
  
/../myls/:  
-rwxrw-r-- 1 x31415 x31415 1334 Jun 11 13:29 main.c   
-rwxrw-r-- 1 x31415 x31415 4208 Jun 11 13:29 main.o   
-rwxrw-r-- 1 x31415 x31415 328 Jun 07 19:52 makefile   
-rwxrwxr-x 1 x31415 x31415 22624 Jun 21 21:50 myls   
-rwxrw-r-- 1 x31415 x31415 9913 Jun 11 13:38 myls.c   
-rwxrw-r-- 1 x31415 x31415 2824 Jun 10 19:58 myls.h   
-rwxrw-r-- 1 x31415 x31415 16480 Jun 21 21:50 myls.o   
  
/../pipe/:  
-rwxrwxr-x 1 x31415 x31415 8408 Jun 18 01:15 A   
-rwxrw-r-- 1 x31415 x31415 262 Jun 18 01:15 A.c   
-rwxrwxr-x 1 x31415 x31415 8480 Jun 18 16:39 B   
-rwxrw-r-- 1 x31415 x31415 463 Jun 18 16:40 B.c   
-rwxrwxr-x 1 x31415 x31415 8600 Jun 18 15:48 C   
-rwxrw-r-- 1 x31415 x31415 431 Jun 18 15:48 C.c   
-rwxrwxr-x 1 x31415 x31415 12824 Jun 18 16:42 pipe   
-rwxrw-r-- 1 x31415 x31415 2046 Jun 18 16:42 pipe.c   
  
/../socket/:  
drwxrwxr-x 2 x31415 x31415 4096 Jun 11 19:36 client   
drwxrwxr-x 2 x31415 x31415 4096 Jun 11 19:38 server   
  
/../socket/client/:  
-rwxrwxr-x 1 x31415 x31415 13208 Jun 11 19:36 client   
-rwxrw-r-- 1 x31415 x31415 1315 Jun 11 19:36 client.c   
-rwxrw-r-- 1 x31415 x31415 3456 Jun 11 19:36 client.o   
-rwxrw-r-- 1 x31415 x31415 330 Jun 11 18:27 makefile   
  
/../socket/server/:  
-rwxrw-r-- 1 x31415 x31415 331 Jun 11 18:28 makefile   
-rwxrwxr-x 1 x31415 x31415 13224 Jun 11 19:38 server   
-rwxrw-r-- 1 x31415 x31415 1948 Jun 11 19:38 server.c   
-rwxrw-r-- 1 x31415 x31415 3712 Jun 11 19:38 server.o

* myls -X

x31415@iZ2zea1nnstgr3h5ccfovoZ:~/linux/myls$ ./myls -X ./  
main.c main.o makefile myls myls.c myls.h myls.o

## 使用说明

在linux系统中，进入程序文件夹，可以直接使用 ./myls 来使用，也可以编译源文件生成可执行文件，编译的方法使用 make 即可。

# 实验总结

这次实验使用linux中的一些系统提供的函数来实现对文件信息的读取，主要是 stat opendir 和 readdir 等函数的使用，对于其他的操作主要是逻辑的实现上代码和格式化输出的编写。这次实验充分的理解了linux下c对文件系统的操作模式，同时观看源码和别人的实现也从中学习到了很多操作的方法，比如c中对结构体的排序方法，文件操作中格式化的优化过程等等。

# 代码

## main.c

#include"myls.h"  
  
int main(int avgc, char\*\* avgs){  
   
 init();  
 char path[1024];  
 strcpy(path, "./"); // 默认执行路径为当前路径  
 for(int i = 1; i <= avgc - 1; ++i){  
 if(!strcmp(avgs[i], "-a"))setCommandIsA();  
 else if(!strcmp(avgs[i], "-f"))setCommandIsF();  
 else if(!strcmp(avgs[i], "-l"))setCommandIsL();  
 else if(!strcmp(avgs[i], "-R"))setCommandIsR();  
 else if(!strcmp(avgs[i], "-C"))setCommandIsC();  
 else if(!strcmp(avgs[i], "-x"))setCommandISX();  
 else if(!strcmp(avgs[i], "--help")){  
 puts("Here is the usage of 'myls': ");  
 puts("Usage: ls [OPTiON]...[FILE]...");  
 puts("");  
 puts("<-l>:\t--list\tuse a long listing format\t输出文件的所有信息");  
 puts("<-a>:\t--all\tdo not ignore entries starting with.\t不忽略隐藏文件");  
 puts("<-f>:\t\tdo not sort\t不排序");  
 puts("<-R>:\t--recursive\tlist subdirectories recursively\t以递归的方式显示所有文件");  
 puts("<-C>:\t\tlist entries by columns\t按列的方式输出（默认）");  
 puts("<-x>:\t\tlist entries by lines instead of by columns\t按行的方式输出");  
 return 0;  
 }  
 else strcpy(path, avgs[i]);  
 }  
  
 myls(path);  
  
 return 0;  
}

## myls.h

#ifndef myls\_h  
#define myls\_h  
#include<stdio.h>  
#include<stdlib.h>  
#include<string.h>  
  
  
// stat  
#include<sys/types.h>  
#include<sys/stat.h>  
#include<unistd.h>   
  
// dir  
// #include<sys/types.h>  
#include<dirent.h>  
  
// pwd  
#include<pwd.h>  
  
// time  
#include<time.h>  
  
// gcc不支持bool  
#include<stdbool.h>  
  
// 为了输出的美观，某些命令需要得知当前窗口的宽度  
#include<sys/ioctl.h>  
  
void init();  
void myls(const char\* path);  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 文件处理部分 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
// 为了能够实现对某些命令的排序功能，所以不能使用直接输出的方式，先要存下来  
typedef struct FILE\_NODE{  
 char file\_type; // 文件类型，包括普通文件、目录文件、链接文件等等  
 char file\_mode[9]; // 文件的各个权限  
 int st\_nlink; // 链接文件的个数  
 char uname[128]; // 用户名  
 char gname[128]; // 组名  
 int file\_size; // 文件大小  
 char date[128]; // 文件的修改日期  
 char file\_name[128]; // 文件名  
 bool islink; // 是否是链接文件  
 char file\_link[128]; // 链接文件指向的文件名  
}FILE\_NODE;  
FILE\_NODE file\_node[128];  
int file\_node\_num; // 文件数  
  
void error\_print(const char\* e); // 错误输出  
void display(); // 将处理后的按一定的规则输出  
void file\_print(const char\* filename, const struct stat\* st); // 处理单个文件  
void dir\_print(const char\* pathname); // 处理一个目录  
void file\_type\_print(const struct stat\* st); // 处理类型，目录或者文件等  
void file\_mode\_print(const struct stat\* st); // 处理权限  
void file\_nlink\_print(const struct stat\* st); // 处理硬链接数量  
void file\_user\_group\_print(const struct stat\* st); // 处理用户及组  
void file\_size\_print(const struct stat\* st); // 处理文件大小  
void file\_lastchange\_time\_print(const struct stat\* st); // 处理文件最后修改时间  
void file\_name\_print(const char\* filename, const struct stat\* st); // 处理文件名，如果是链接文件，追加对应链接的文件路径  
void file\_link\_print(const char\* filename); // 处理链接的文件  
  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 命令处理部分 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
#define MYLS\_A 1  
#define MYLS\_L 2  
#define MYLS\_F 4  
#define MYLS\_R 8  
#define MYLS\_X 16  
#define MYLS\_C 32  
unsigned int COMMANDFLAG;  
void setCommandIsA(); // 设置 -a 显示文件名，按列适当的以字典序排序  
void setCommandIsL(); // 设置 -l 显示文件的所有信息，每行一个，默认以文件名的字典序排序  
void setCommandIsF(); // 设置 -f 与 -a 类似，但不排序  
void setCommandIsR(); // 对文件夹进行递归显示  
void setCommandISX(); // 对文件按列输出，横向排序  
void setCommandIsC(); // 对文件按列输出，纵向排序  
#endif

## myls.c

#include "myls.h"  
  
  
void dir\_print\_dfs();  
void init(){  
 COMMANDFLAG = 0;  
}  
void myls(const char\* path){  
 struct stat st = {};  
 if(!~lstat(path, &st))error\_print("lstat...");  
  
 // 递归显示  
 if(COMMANDFLAG & MYLS\_R){  
 dir\_print\_dfs(path);  
 return;  
 }  
  
 file\_node\_num = -1;  
 if(S\_ISDIR(st.st\_mode))dir\_print(path);  
 else file\_print(path, &st);  
 display();  
 return;  
}  
  
void error\_print(const char\* e){  
 perror("Error: ");  
 perror(e);  
 exit(EXIT\_FAILURE);  
 // exit(1);  
}  
  
int cmp\_string(const void\* a, const void\* b){  
 // 结构体字符串排序  
 return strcmp((\*(FILE\_NODE\*)a).file\_name, (\*(FILE\_NODE\*)b).file\_name);  
}  
int get\_max\_len(bool sizeOrname){  
 // 获取当前文件组中大小最大的那一个的长度  
 int len = 0;  
 for(int i = 0; i <= file\_node\_num; ++i){  
 len = len < file\_node[i].file\_size ? file\_node[i].file\_size : len;  
 }  
 int ans = 0;  
 while(len){  
 ++ans;  
 len /= 10;  
 }  
 return ans;  
}  
int get\_winsize\_col(){  
 // 获取当前控制台的宽度  
 struct winsize size;  
 ioctl(STDIN\_FILENO, TIOCGWINSZ, &size);  
 return size.ws\_col;  
}  
void display(){  
 // 默认按照文件名的字典序进行输出  
 if(!(COMMANDFLAG & MYLS\_F))qsort(file\_node, file\_node\_num + 1, sizeof(file\_node[0]), cmp\_string);  
   
 if(COMMANDFLAG & MYLS\_L){  
 int max\_size\_len = get\_max\_len(true); // 按文件大小最大的格式化输出  
 int max\_uname\_len = 0;  
 int max\_gnmame\_len = 0;  
 for(int i = 0; i <= file\_node\_num; ++i){  
 max\_uname\_len = max\_uname\_len < strlen(file\_node[i].uname) ? strlen(file\_node[i].uname) : max\_uname\_len;  
 max\_gnmame\_len = max\_gnmame\_len < strlen(file\_node[i].gname) ? strlen(file\_node[i].gname) : max\_gnmame\_len;  
 }  
 for(int i = 0; i <= file\_node\_num; ++i){  
 printf("%c", file\_node[i].file\_type);  
 printf("%s ", file\_node[i].file\_mode);  
 printf("%d ", file\_node[i].st\_nlink);  
 printf("%\*s ", max\_uname\_len, file\_node[i].uname);  
 printf("%\*s ", max\_gnmame\_len, file\_node[i].gname);  
 printf("%\*d ", max\_size\_len, file\_node[i].file\_size);  
 printf("%s ", file\_node[i].date);  
 printf("%-20s", file\_node[i].file\_name);  
 if(file\_node[i].islink)printf("%s", file\_node[i].file\_link);  
 puts("");  
 }  
 }  
 else{  
 // 确定每一行的缩进长度，以及最后的行数  
 int col = get\_winsize\_col();  
 int max\_col = 0;  
 for(int i = 0; i <= file\_node\_num; ++i)max\_col = max\_col < strlen(file\_node[i].file\_name) ? strlen(file\_node[i].file\_name) : max\_col;  
 max\_col += 1;  
 max\_col = col / max\_col;  
 int max\_row = (file\_node\_num + max\_col - 1) / max\_col;  
 int cols[10];  
 memset(cols, 0, sizeof(cols));  
 // 获取每一行的最长值  
 for(int i = 0; i < max\_row; ++i){  
 for(int j = 0; j < max\_col; ++j){  
 if(COMMANDFLAG & MYLS\_X){  
 cols[j] = cols[j] < strlen(file\_node[i \* max\_col + j].file\_name) ? strlen(file\_node[i \* max\_col + j].file\_name) : cols[j];  
 }  
 else{  
 cols[j] = cols[j] < strlen(file\_node[i + j \* max\_row].file\_name) ? strlen(file\_node[i + j \* max\_row].file\_name) : cols[j];  
 }  
 }  
 }  
 int flag = 100;  
 int x;  
 for(int i = 0; i < max\_row; ++i){  
 for(int j = 0; j < max\_col; ++j){  
 if(COMMANDFLAG & MYLS\_X){  
 x = i \* max\_col + j;  
 }  
 else{  
 x = i + j \* max\_row;  
 }  
 printf("%-\*s ", cols[j], file\_node[x].file\_name);  
 --flag;  
 if(flag <= 0)return;  
 }  
 puts("");  
 }  
 }  
 return;  
}  
void file\_print(const char\* filename, const struct stat\* st){  
 ++file\_node\_num;  
 file\_type\_print(st);  
 file\_mode\_print(st);  
 file\_nlink\_print(st);  
 file\_user\_group\_print(st);  
 file\_size\_print(st);  
 file\_lastchange\_time\_print(st);  
 file\_name\_print(filename, st);  
}  
  
  
  
void dir\_print(const char\* pathname){  
 DIR\* dir = opendir(pathname); // 打开目录  
  
 if(dir == NULL)error\_print("opendir..."); // 判断该目录是否能打开  
  
 if(!~chdir(pathname))error\_print("chdir..."); // 改变工作区，方便stat使用  
  
 struct dirent\* dir\_next = NULL; // 目录列表文件的指针  
  
 struct stat st = {}; // 文件或目录的stat  
 while(dir\_next = readdir(dir)){  
 char\* filename = dir\_next->d\_name; // 文件或目录名字  
 if(!~lstat(filename, &st))error\_print(filename);  
 file\_print(filename, &st);  
 }  
 return;  
}  
  
char paths[100][128];  
int path\_n;  
void dir\_print\_dfs(const char\* path){  
 file\_node\_num = -1;  
 strcpy(paths[++path\_n], path);  
 for(int i = 0; i <= path\_n; ++i)printf("%s%c", paths[i], (paths[i][strlen(paths[i]) - 1] == '/' ? : '/'));  
 printf(":\n");  
 dir\_print(path);  
 display();  
 puts("");  
 // char pathname[1024];  
 // strcpy(pathname, sta[top]);  
 // char buf[1024];  
 // getcwd(buf, 1024);  
 // printf("%s\n", buf);  
 // printf("%s\n", sta[top]);  
 // if(!~chdir(sta[top]))error\_print("chdir...");  
 // getcwd(buf, 1024);  
 // printf("%s\n", buf);  
 struct stat st = {};  
 char tmp[100][128];  
 int num = 0;  
 for(int i = 0; i <= file\_node\_num; ++i){  
 if(!~lstat(file\_node[i].file\_name, &st))error\_print(file\_node[i].file\_name);  
 if(S\_ISDIR(st.st\_mode)){  
 strcpy(tmp[num++], file\_node[i].file\_name);  
 }  
 }  
 for(int i = 0; i < num; ++i){  
 dir\_print\_dfs(tmp[i]);  
 }  
 if(!~chdir("../"))error\_print("chdir...");  
 --path\_n;  
}  
  
void file\_type\_print(const struct stat\* st){  
 // mode\_t mode = (\*st).st\_mode;  
 mode\_t mode = st->st\_mode;  
   
 if(S\_ISREG(mode))file\_node[file\_node\_num].file\_type = '-'; // 普通文件  
 else if(S\_ISDIR(mode))file\_node[file\_node\_num].file\_type = 'd'; // 目录文件  
 else if(S\_ISLNK(mode))file\_node[file\_node\_num].file\_type = 'l'; // 链接文件  
 else if(S\_ISCHR(mode))file\_node[file\_node\_num].file\_type = 'c'; // 字符设备文件  
 else if(S\_ISFIFO(mode))file\_node[file\_node\_num].file\_type = 'p'; // 管道文件  
 else if(S\_ISBLK(mode))file\_node[file\_node\_num].file\_type = 'b'; // 块设备文件  
}  
  
void file\_mode\_print(const struct stat\* st){  
 mode\_t mode = st->st\_mode;  
  
 file\_node[file\_node\_num].file\_mode[0] = (mode & S\_IRUSR ? 'r' : '-');  
 file\_node[file\_node\_num].file\_mode[1] = (mode & S\_IWUSR ? 'w' : '-');  
 file\_node[file\_node\_num].file\_mode[2] = (mode & S\_IWUSR ? 'x' : '-');  
  
 file\_node[file\_node\_num].file\_mode[3] = (mode & S\_IRGRP ? 'r' : '-');  
 file\_node[file\_node\_num].file\_mode[4] = (mode & S\_IWGRP ? 'w' : '-');  
 file\_node[file\_node\_num].file\_mode[5] = (mode & S\_IXGRP ? 'x' : '-');  
  
 file\_node[file\_node\_num].file\_mode[6] = (mode & S\_IROTH ? 'r' : '-');  
 file\_node[file\_node\_num].file\_mode[7] = (mode & S\_IWOTH ? 'w' : '-');  
 file\_node[file\_node\_num].file\_mode[8] = (mode & S\_IXOTH ? 'x' : '-');  
  
 return;  
}  
  
void file\_nlink\_print(const struct stat\* st){  
 file\_node[file\_node\_num].st\_nlink = st->st\_nlink;  
 return;  
}  
  
void file\_user\_group\_print(const struct stat\* st){  
 memset(file\_node[file\_node\_num].uname, '\0', sizeof(file\_node[file\_node\_num].uname));  
 strcpy(file\_node[file\_node\_num].uname, getpwuid(st->st\_uid)->pw\_name); // 根据文件的用户id获取用户名  
  
 memset(file\_node[file\_node\_num].gname, '\0', sizeof(file\_node[file\_node\_num].gname));  
 strcpy(file\_node[file\_node\_num].gname, getpwuid(st->st\_gid)->pw\_name); // 根据文件的用户id获取组名  
 return;  
}  
  
void file\_size\_print(const struct stat\* st){  
 file\_node[file\_node\_num].file\_size = st->st\_size;  
 return;  
}  
  
void file\_lastchange\_time\_print(const struct stat\* st){  
 char\* mon[] = {"Jan", "Feb", "Mar", "Apr", "May", "Jun",  
 "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"};  
   
 struct tm\* file\_time = localtime(&(st->st\_mtime));  
 if(file\_time == NULL)error\_print("localtime...");  
 memset(file\_node[file\_node\_num].date, '\0', sizeof(file\_node[file\_node\_num].date));  
 char \_mon[6], \_day[4], \_time[7];  
 sprintf(\_mon, "%s ", mon[file\_time->tm\_mon]);  
 sprintf(\_day, "%02d ", file\_time->tm\_mday + 1);  
 sprintf(\_time, "%02d:%02d", file\_time->tm\_hour, file\_time->tm\_min);  
 strcat(file\_node[file\_node\_num].date, \_mon);  
 strcat(file\_node[file\_node\_num].date, \_day);  
 strcat(file\_node[file\_node\_num].date, \_time);  
 return;  
}  
  
void file\_name\_print(const char\* filename, const struct stat\* st){  
 if(filename[0] == '.'){  
 if(!(COMMANDFLAG & MYLS\_A) && !(COMMANDFLAG & MYLS\_F)){  
 --file\_node\_num;  
 return;  
 }  
 }  
 memset(file\_node[file\_node\_num].file\_name, '\0', sizeof(file\_node[file\_node\_num].file\_name));  
 sprintf(file\_node[file\_node\_num].file\_name, "%s", filename);  
 if(S\_ISLNK(st->st\_mode))file\_link\_print(filename);  
 return;  
}  
  
void file\_link\_print(const char\* filename){  
 char buf[1024];  
 if(0 == readlink(filename, buf, sizeof buf))error\_print("readlink...");  
 memset(file\_node[file\_node\_num].file\_link, '\0', sizeof(file\_node[file\_node\_num].file\_link));  
 sprintf(file\_node[file\_node\_num].file\_link, "-> %s", buf);  
 return;  
}  
  
  
  
void setCommandIsA(){  
 COMMANDFLAG |= MYLS\_A;  
 return;  
}  
void setCommandIsL(){  
 COMMANDFLAG |= MYLS\_L;  
 return;  
}  
void setCommandIsF(){  
 COMMANDFLAG |= MYLS\_F;  
 return;  
}  
void setCommandIsR(){  
 COMMANDFLAG |= MYLS\_R;  
 return;  
}  
void setCommandISX(){  
 COMMANDFLAG |= MYLS\_X;  
 return;  
}  
void setCommandIsC(){  
 COMMANDFLAG |= MYLS\_C;  
 return;  
}