touch -- -a 或 rm ./-a

int stat(const char \*pathname, struct stat \*statbuf);

int fstat(int fd, struct stat \*statbuf);

int lstat(const char \*pathname, struct stat \*statbuf);

statbuf是一个stat数据结构的地址，用于接受返回的文件信息;stat和lstat的第一个参数是字符串地址，lstat会显示链接的信息，fstat的第一个参数是文件描述符;

struct stat {

dev\_t st\_dev; /\* ID of device containing file \*/

ino\_t st\_ino; /\* Inode number \*/

mode\_t st\_mode; /\* File type and mode \*/

nlink\_t st\_nlink; /\* Number of hard links \*/

uid\_t st\_uid; /\* User ID of owner \*/

gid\_t st\_gid; /\* Group ID of owner \*/

dev\_t st\_rdev; /\* Device ID (if special file) \*/

off\_t st\_size; /\* Total size, in bytes \*/

blksize\_t st\_blksize; /\* Block size for filesystem I/O \*/

blkcnt\_t st\_blocks; /\* Number of 512B blocks allocated \*/

struct timespec定义了纳秒时间

struct timespec st\_atim; /\* Time of last access \*/

struct timespec st\_mtim; /\* Time of last modification \*/

struct timespec st\_ctim; /\* Time of last status change \*/

#define st\_atime st\_atim.tv\_sec /\* Backward compatibility \*/

#define st\_mtime st\_mtim.tv\_sec

#define st\_ctime st\_ctim.tv\_sec

};

文件类型

宏定义函数：

S\_ISREG(m) is it a regular file?

S\_ISDIR(m) directory?

S\_ISCHR(m) character device?

S\_ISBLK(m) block device?

S\_ISFIFO(m) FIFO (named pipe)?

S\_ISLNK(m) symbolic link? (Not in POSIX.1-1996.)

S\_ISSOCK(m) socket? (Not in POSIX.1-1996.)

位图：

S\_IFMT 0170000 bit mask for the file type bit field

S\_IFSOCK 0140000 socket

S\_IFLNK 0120000 symbolic link

S\_IFREG 0100000 regular file

S\_IFBLK 0060000 block device

S\_IFDIR 0040000 directory

S\_IFCHR 0020000 character device

S\_IFIFO 0010000 FIFO

umask 防止产生权限过松的文件

文件权限更改和管理

int chmod(const char \*pathname, mode\_t mode);

int fchmod(int fd, mode\_t mode);

mode\_t是一个16位的八进制数

粘着位

对目录设置粘着位，只有对该目录，具有写权限，的用户满足以下条件之一才能删除或重命名该目录下的文件：拥有此文件、拥有此目录、是超级用户

文件截断

int truncate(const char \*path, off\_t length);

int ftruncate(int fd, off\_t length);

将一个现有文件的长度截断为length，如果该文件长度大于length，则length以后数据不能再访问，若小于length则文件长度则增加;成功返回0,失败返回-1;

int link(const char \*oldpath, const char \*newpath);

创建一个新的目录项newpath，硬链接与目录项是同义词;成功返回0,失败返回-1;

int unlink(const char \*pathname);

删除目录项，并将由pathname所引用的文件的链接计数-1;

关闭一个文件时，内核首先检查打开文件的进程个数，如果计数个数达到0,内核再去检查链接计数，如果也是0,删除该文件内容;

unlink被用于即使程序崩溃时，创建的临时文件也不会保留下来;即进程创建一个文件后立即调用unlink;

如果pathname是符号链接，那么unlink删除该符号链接;

int remove(const char \*pathname);

remove是ISO C函数，对于文件等同于unlink，对于目录等同于rmdir;成功返回0,失败返回-1;

int rename(const char \*oldpath, const char \*newpath);

重命名文件，newpath已经存在且是文件或空目录，则将其删除再将oldname重命名为newname;处理符号文件本身;若成功返回0,失败返回-1;

int symlink(const char \*target, const char \*linkpath);

符号链接不占用block，所有内容保存在inode里面;传建议额指向target的新目录项linkpath，且不要求target存在;若成功返回0,失败返回-1;

int utime(const char \*filename, const struct utimbuf \*times);

int utimes(const char \*filename, const struct timeval times[2]);

成功返回0,失败返回-1;

struct utimbuf {

time\_t actime; /\* access time \*/

time\_t modtime; /\* modification time \*/

};

struct timeval {

long tv\_sec; /\* seconds \*/

long tv\_usec; /\* microseconds \*/

};

int mkdir(const char \*pathname, mode\_t mode);

创建一个新的空目录，其中.和..目录项是自动创建的;

int rmdir(const char \*pathname);

如果调用此函数使目录的链接计数成为0,且没有其他进程打开此目录，则释放此目录占用的空间;

int glob(const char \*pattern, int flags,

int (\*errfunc) (const char \*epath, int eerrno),

glob\_t \*pglob);

void globfree(glob\_t \*pglob);

对字符串的模式进行解析，可以解析通配符;比较重要的flag是GLOB\_APPEND可以不清空pglob;

DIR \*opendir(const char \*name);

DIR \*fdopendir(int fd);

成功返回指针，失败返回NULL;

int closedir(DIR \*dirp);

成功返回0，失败返回-1;

struct dirent \*readdir(DIR \*dirp);

成功返回指针，失败或到文件尾返回NULL;

struct dirent {

ino\_t d\_ino; /\* Inode number \*/

off\_t d\_off; /\* Not an offset; see below \*/

unsigned short d\_reclen; /\* Length of this record \*/

unsigned char d\_type; /\* Type of file; not supported

by all filesystem types \*/

char d\_name[256]; /\* Null-terminated filename \*/

};

void rewinddir(DIR \*dirp);

成功返回0,失败返回-1;

long telldir(DIR \*dirp);

返回目录中当前位置;

void seekdir(DIR \*dirp, long loc);

int chdir(const char \*path);

int fchdir(int fd);

修改当前工作目录;成功返回0,失败返回-1;

char \*getcwd(char \*buf, size\_t size);

返回当前工作路径;成功返回buf，失败返回NULL;