- **umask**
- **chmod**
- **mkdir**
- **link**
- **unlink**
- **rename**
- **symlink**
- **readlink**
- **utime**

Source: http://web.eecs.utk.edu/~plank/plank/classes/cs360/

umask

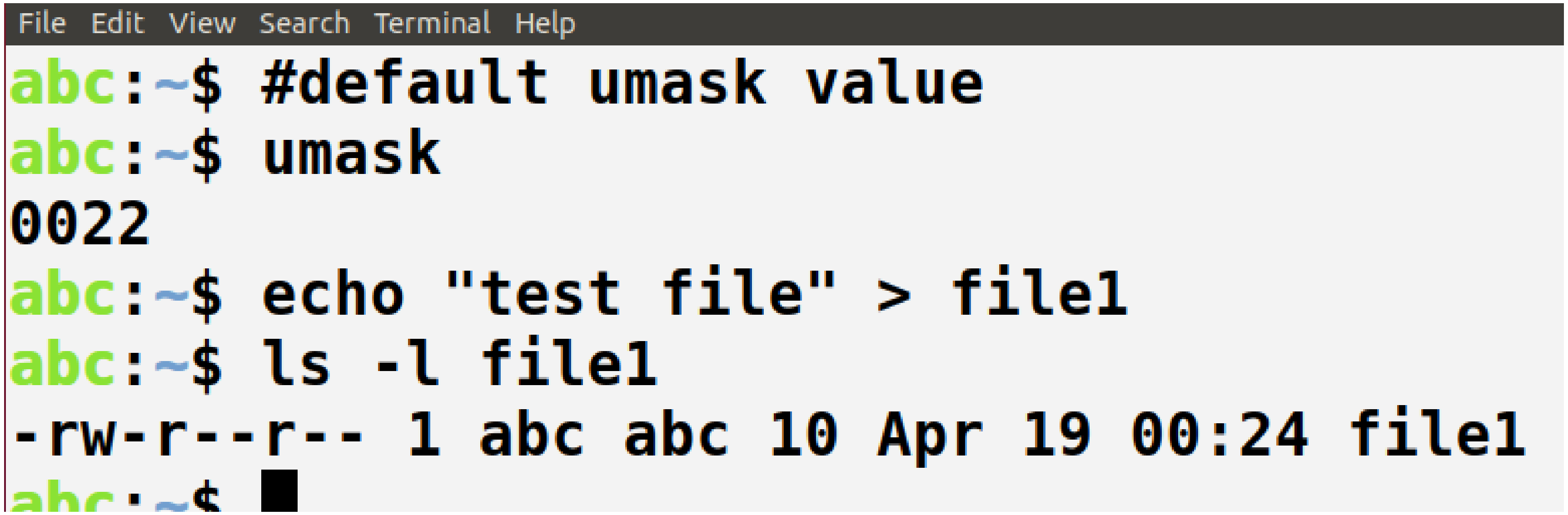
- **umask** is a system call that handles the "file mode creation mask" .
- "file creation mask" (which I will call the "umask" out of habit) is a nine-bit number. If a bit in the umask is set, then whenever you make a system call that creates a file, that bit in the protection mode will be turned off.
- Formally, when you specify a mode when you open a file, the real protection mode will be:

- (mode & ~umask)

umask

- the umask "turns off" protection bits.
- The point of the umask is to allow programs to create files with the following protection modes:
  - Regular text and data files may be opened with the mode 0666.
  - Directories and executable files may be opened with the mode 0777.

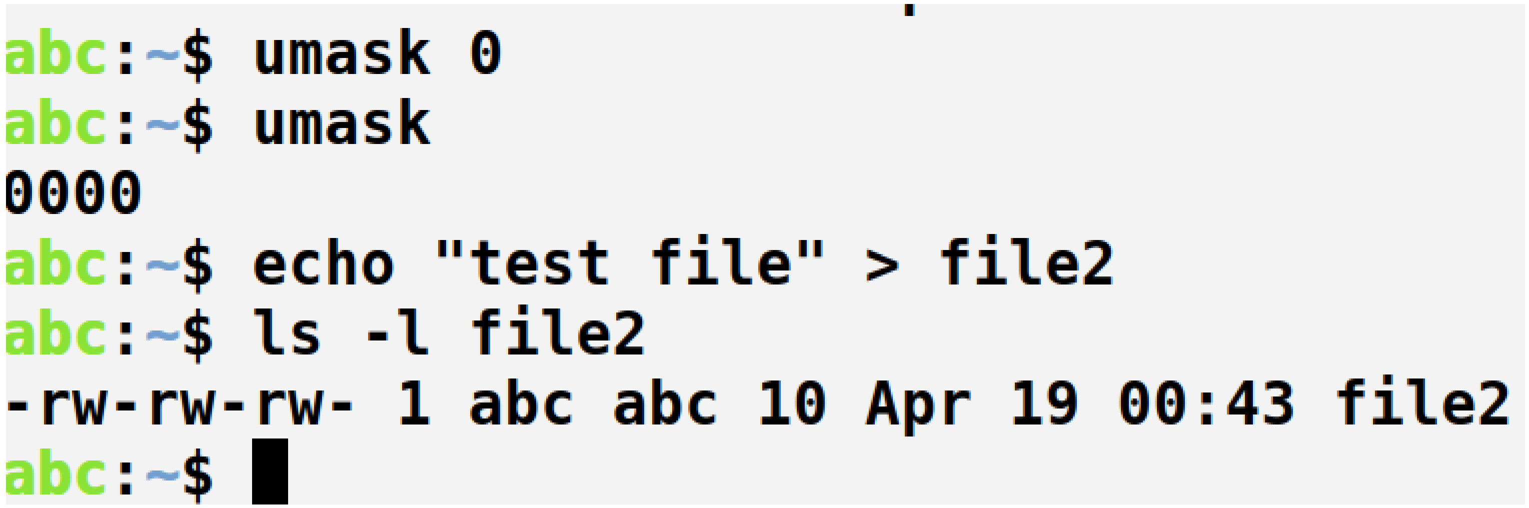
umask



```
abc:~$ #default umask value
abc:~$ umask
0022
abc:~$ echo "test file" > file1
abc:~$ ls -l file1
-rw-r--r-- 1 abc abc 10 Apr 19 00:24 file1
abc:~$ 
```

- ***Regular text and data files may be opened with the mode 0666.***
- rwxrwxrwx
- Umask = 0022 =**000010010**
- ~Umask = 0755 =**111101101**
- Mode = 0666 =110110110
- mode&~umask =110100100 =0644=**rw-r--r--**

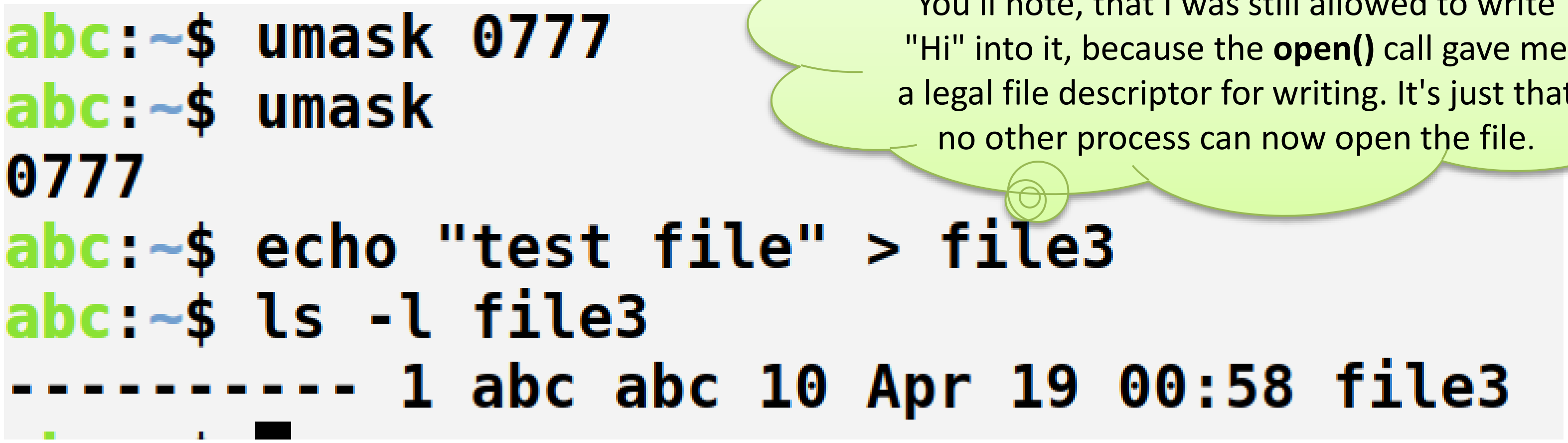
umask



```
abc:~$ umask 0
abc:~$ umask
0000
abc:~$ echo "test file" > file2
abc:~$ ls -l file2
-rw-rw-rw- 1 abc abc 10 Apr 19 00:43 file2
abc:~$
```

- ***Regular text and data files may be opened with the mode 0666.***
- rwxrwxrwx
- Umask = 0000 =000000000
- ~Umask = 0777 =111111111
- Mode = 0666 =110110110
- mode&~umask =110110110 =0666=rw-rw-rw-

umask

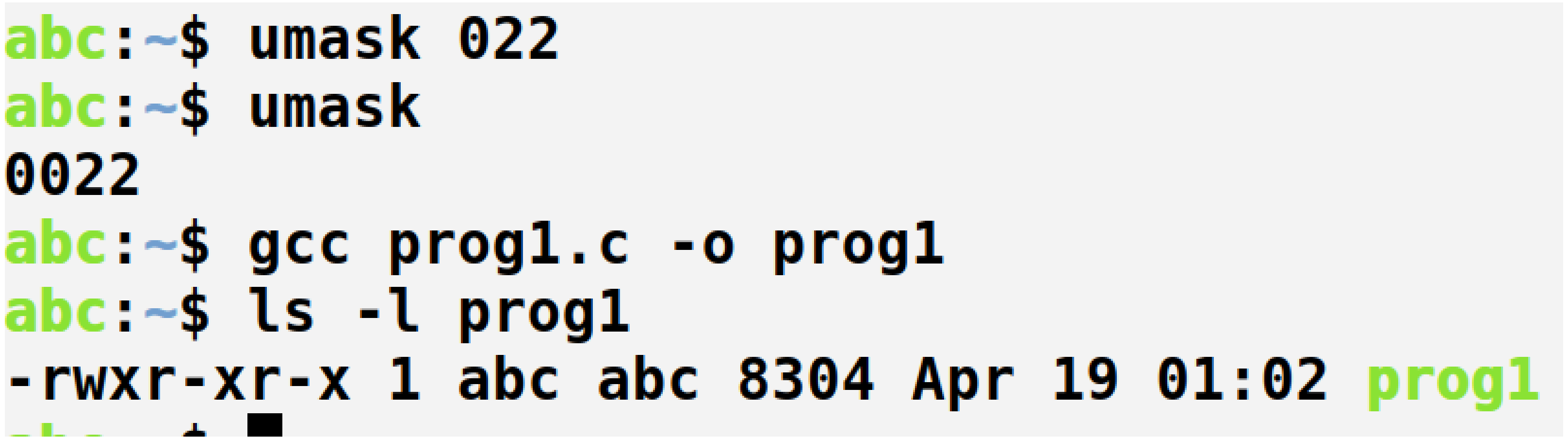


```
abc:~$ umask 0777
abc:~$ umask
0777
abc:~$ echo "test file" > file3
abc:~$ ls -l file3
---------- 1 abc abc 10 Apr 19 00:58 file3
```

When my umask is 0777, then all nine bits, so the file has a protection mode of 0000. You'll note, that I was still allowed to write "Hi" into it, because the **open()** call gave me a legal file descriptor for writing. It's just that no other process can now open the file.

- ***Regular text and data files may be opened with the mode 0666.***
- rwxrwxrwx
- Umask = 0777 =111111111
- ~Umask = 0000 =000000000
- Mode = 0666 =110110110
- mode&~umask =000000000 =0000=---------

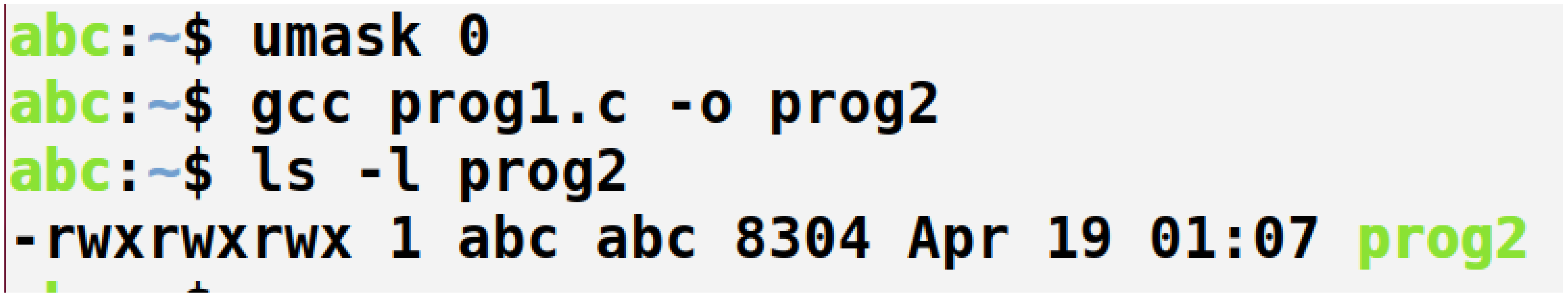
umask



```
abc:~$ umask 022
abc:~$ umask
0022
abc:~$ gcc prog1.c -o prog1
abc:~$ ls -l prog1
-rwxr-xr-x 1 abc abc 8304 Apr 19 01:02 prog1
```

- ***Directories and executable files may be opened with the mode 0777.***
- **rwxrwxrwx**
- Umask = 0022 =**000010010**
- ~Umask = 0755 =**111101101**
- Mode = 0777 =111111111
- mode&~umask =111101101 =0755=**rwxr-xr-x**

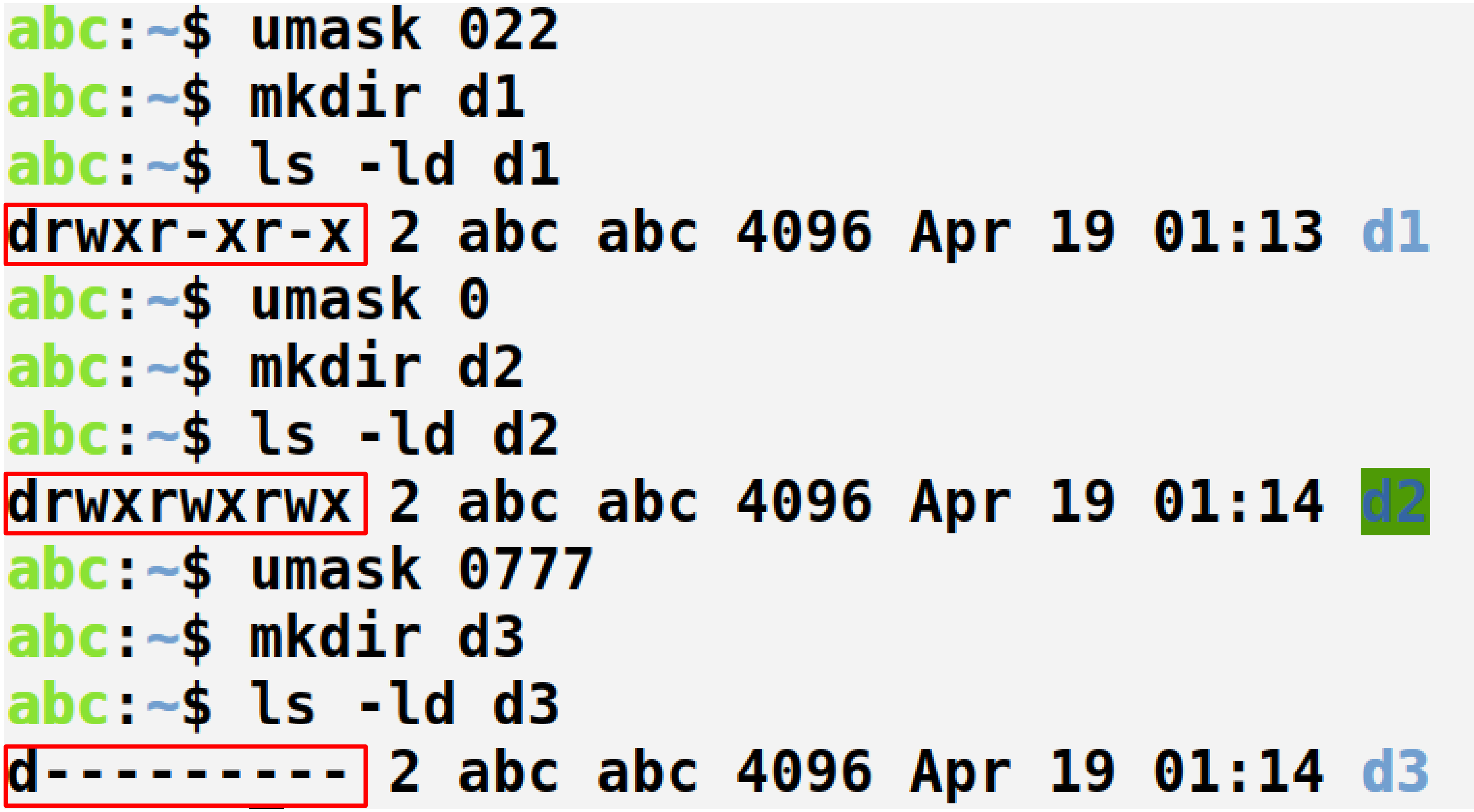
umask



```
abc:~$ umask 0
abc:~$ gcc prog1.c -o prog2
abc:~$ ls -l prog2
-rwxrwxrwx 1 abc abc 8304 Apr 19 01:07 prog2
```

- ***Directories and executable files may be opened with the mode 0777.***
- **rwxrwxrwx**
- Umask = 0000 =**000000000**
- ~Umask = 0777 =**111111111**
- Mode = 0777 =111111111
- mode&~umask =111111111 =0777=**rwxrwxrwx**

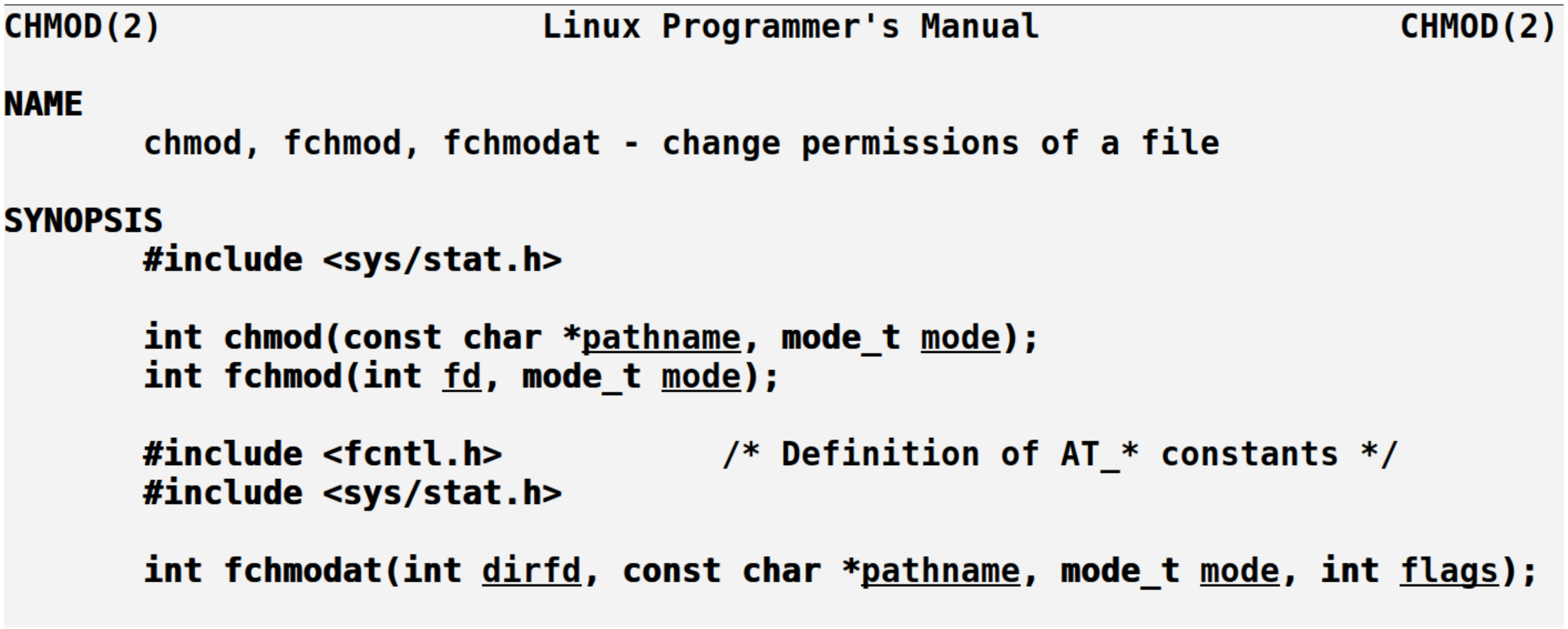
umask



```
abc:~$ umask 022
abc:~$ mkdir d1
abc:~$ ls -ld d1
drwxr-xr-x 2 abc abc 4096 Apr 19 01:13 d1
abc:~$ umask 0
abc:~$ mkdir d2
abc:~$ ls -ld d2
drwxrwxrwx 2 abc abc 4096 Apr 19 01:14 d2
abc:~$ umask 0777
abc:~$ mkdir d3
abc:~$ ls -ld d3
d--------- 2 abc abc 4096 Apr 19 01:14 d3
```

chmod

- Works just like chmod when executed from the shell
- **chmod("f1", 0600)** will set the protection of file **f1** to be "rw-" for you, and "---" for everyone else

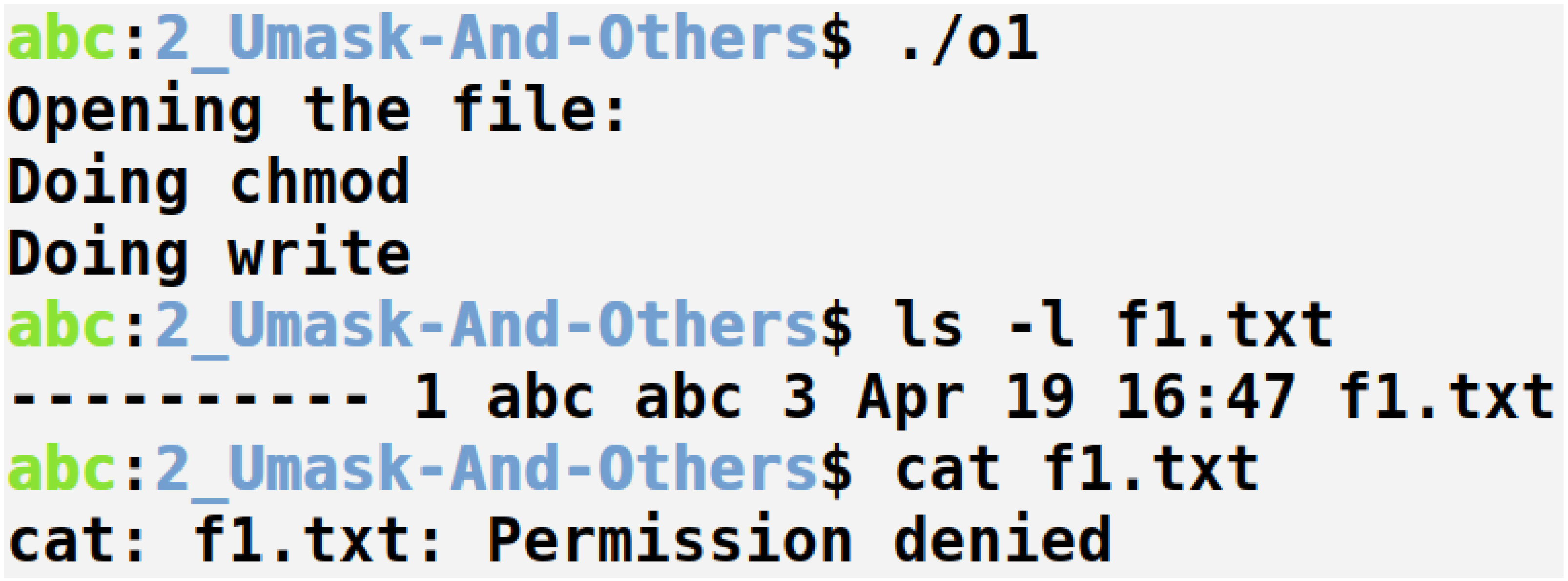


```
CHMOD(2)                  Linux Programmer's Manual                  CHMOD(2)

NAME
       chmod, fchmod, fchmodat - change permissions of a file

SYNOPSIS
       #include <sys/stat.h>

       int chmod(const char *pathname, mode_t mode);
       int fchmod(int fd, mode_t mode);

       #include <fcntl.h>           /* Definition of AT_* constants */
       #include <sys/stat.h>

       int fchmodat(int dirfd, const char *pathname, mode_t mode, int flags);
```

chmod

```
int main()
{
  int fd;

  printf("Opening the file:\n");
  fd = open("f1.txt", O_WRONLY | O_CREAT | O_TRUNC);
  sleep(1);

  printf("Doing chmod\n");
  chmod("f1.txt", 0000);
  sleep(1);

  printf("Doing write\n");
  write(fd, "Hi\n", 3);

  return 0;
}
```

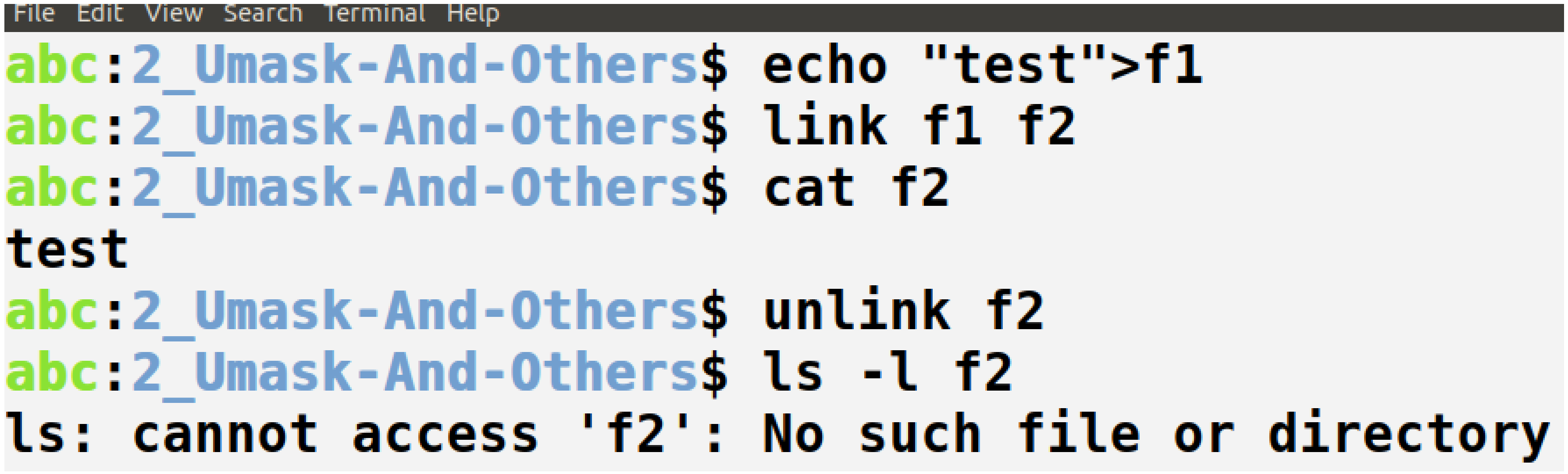
chmod



```
abc:2_Umask-And-Others$ ./o1
Opening the file:
Doing chmod
Doing write
abc:2_Umask-And-Others$ ls -l f1.txt
---------- 1 abc abc 3 Apr 19 16:47 f1.txt
abc:2_Umask-And-Others$ cat f1.txt
cat: f1.txt: Permission denied
```

link and unlink

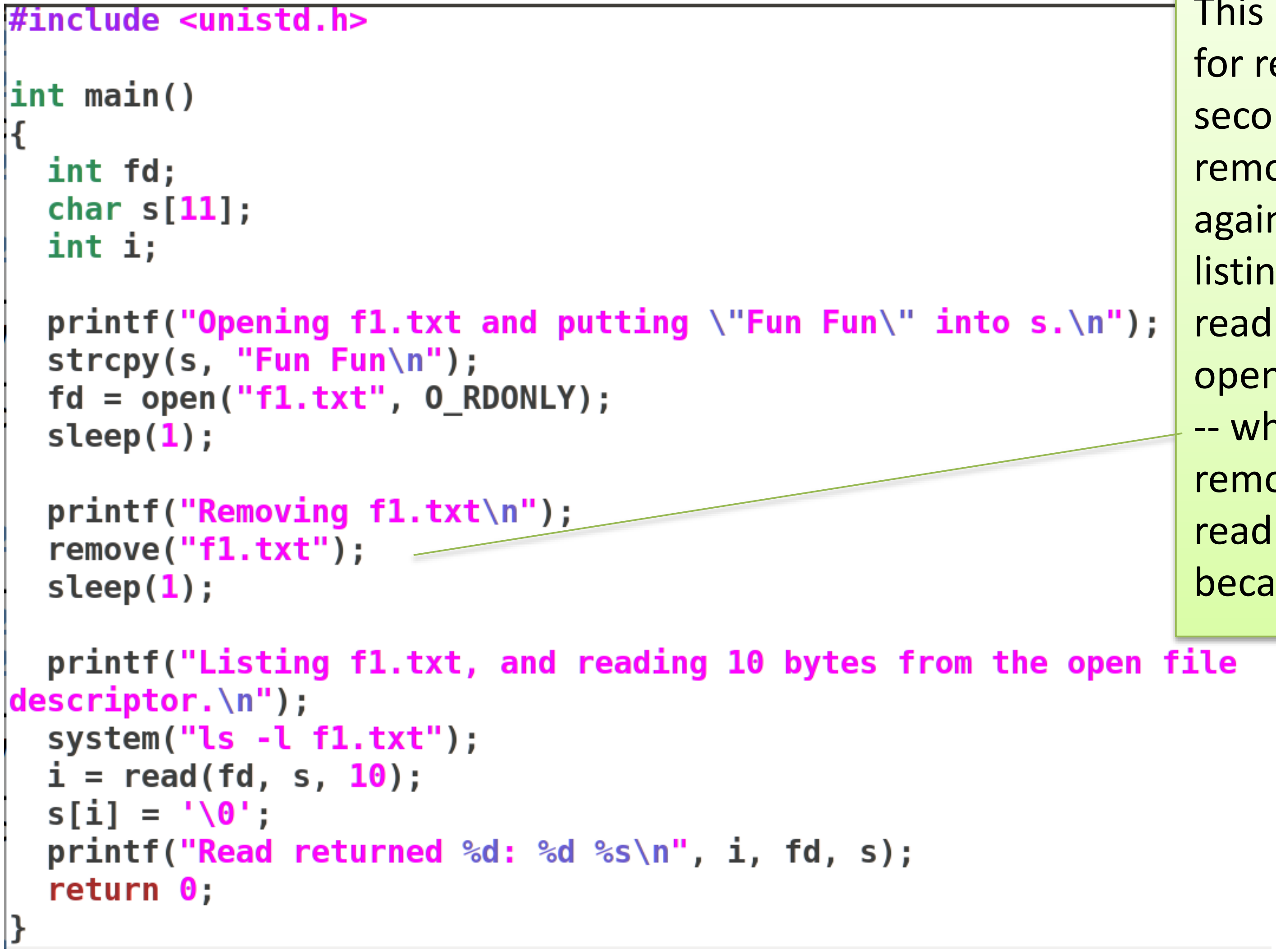
- link(char *f1, char *f2) works just like:
- ln f1 f2
- f2 has to be a file -- it cannot be a directory.
- unlink(char *f1) works like:
- rm f1



```
abc:2_Umask-And-Others$ echo "test">f1
abc:2_Umask-And-Others$ link f1 f2
abc:2_Umask-And-Others$ cat f2
test
abc:2_Umask-And-Others$ unlink f2
abc:2_Umask-And-Others$ ls -l f2
ls: cannot access 'f2': No such file or directory
```

remove

- remove(char *f1) works like unlink(), but it also works for (empty) directories. Unlink() fails on directories.

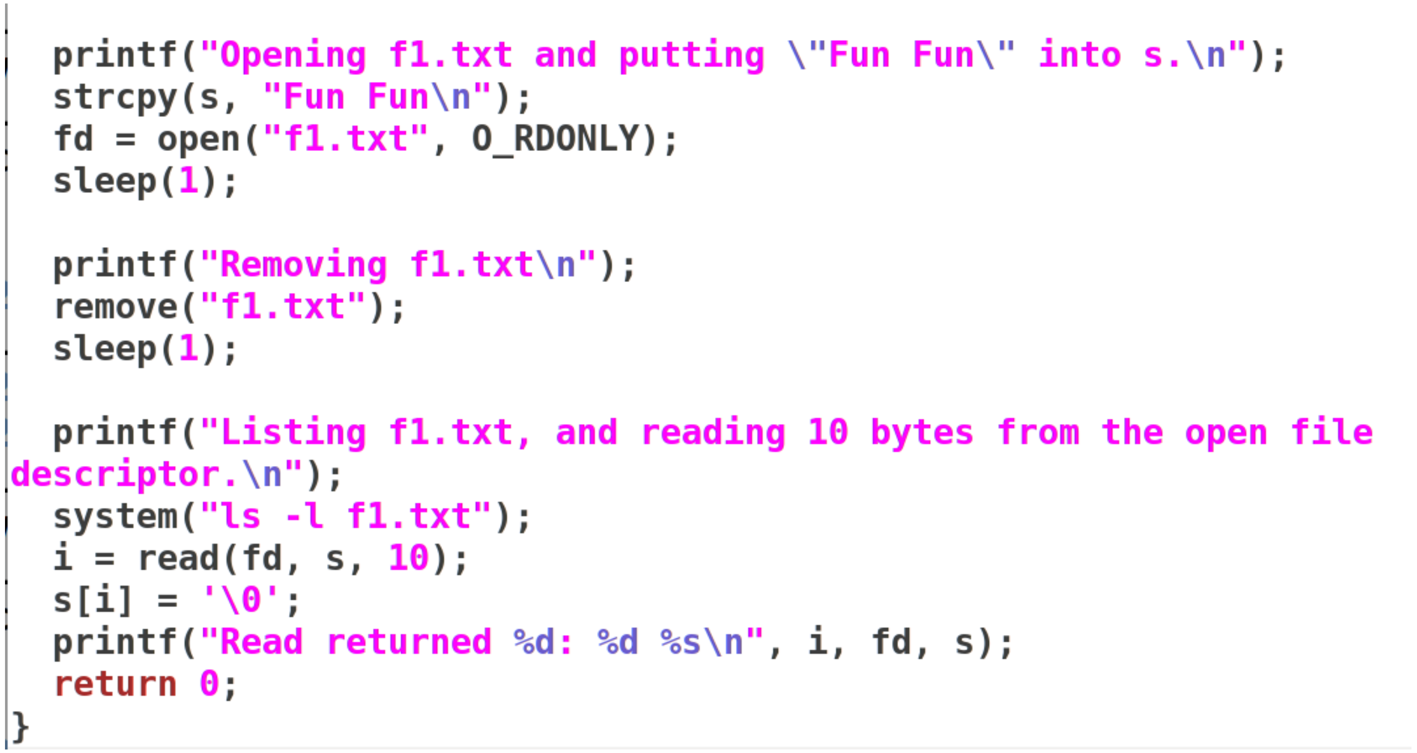


```
#include <unistd.h>

int main()
{
  int fd;
  char s[11];
  int i;

  printf("Opening f1.txt and putting \"Fun Fun\" into s.\n");
  strcpy(s, "Fun Fun\n");
  fd = open("f1.txt", O_RDONLY);
  sleep(1);

  printf("Removing f1.txt\n");
  remove("f1.txt");
  sleep(1);

  printf("Listing f1.txt, and reading 10 bytes from the open file
descriptor.\n");
  system("ls -l f1.txt");
  i = read(fd, s, 10);
  s[i] = '\0';
  printf("Read returned %d: %d %s\n", i, fd, s);
  return 0;
}
```

This program opens **f1.txt** for reading, sleeps a second, and then removes **f1.txt**. It sleeps again, performs a long listing and then tries to read 10 bytes from the open file. The question is -- what happens when we remove **f1.txt**? Will the read call succeed, or fail because the file is gone?

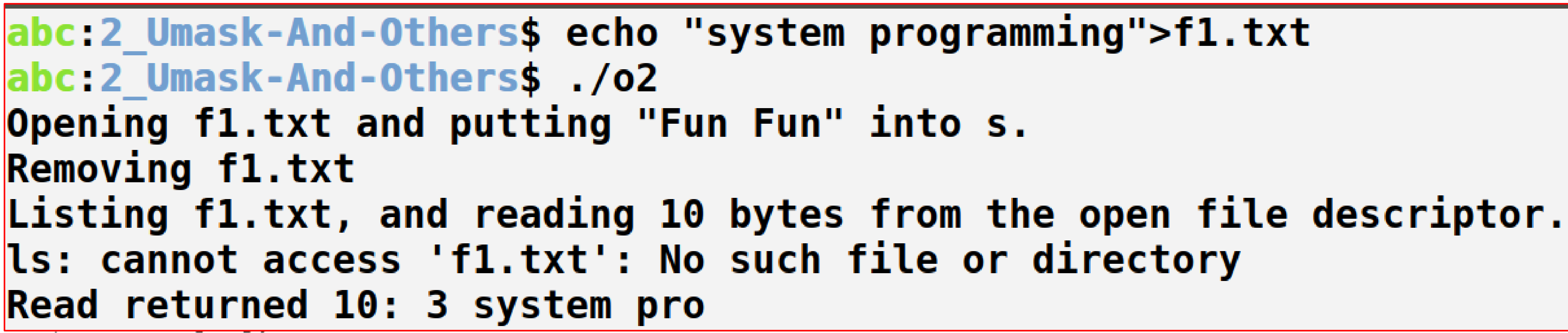
remove

- The **ls** command shows that **f1.txt** is indeed gone after the **remove()** call.
- However, the operating system does not delete the file until the last file descriptor to it is closed. For that reason, the **read()** call succeed

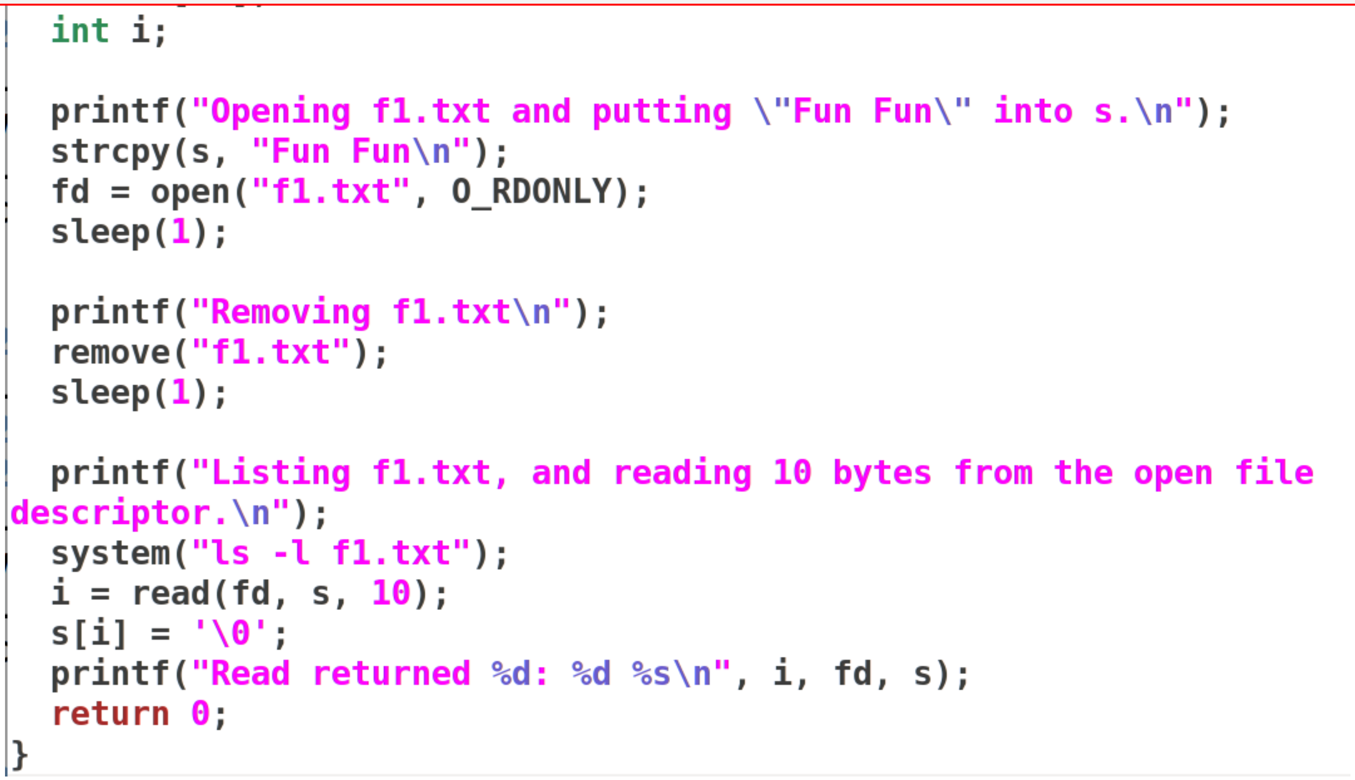


```
  printf("Opening f1.txt and putting \"Fun Fun\" into s.\n");
  strcpy(s, "Fun Fun\n");
  fd = open("f1.txt", O_RDONLY);
  sleep(1);

  printf("Removing f1.txt\n");
  remove("f1.txt");
  sleep(1);

  printf("Listing f1.txt, and reading 10 bytes from the open file
descriptor.\n");
  system("ls -l f1.txt");
  i = read(fd, s, 10);
  s[i] = '\0';
  printf("Read returned %d: %d %s\n", i, fd, s);
  return 0;
}
```

remove



```
abc:2_Umask-And-Others$ echo "system programming">f1.txt
abc:2_Umask-And-Others$ ./o2
Opening f1.txt and putting "Fun Fun" into s.
Removing f1.txt
Listing f1.txt, and reading 10 bytes from the open file descriptor.
ls: cannot access 'f1.txt': No such file or directory
Read returned 10: 3 system pro
```



```
   int i;

   printf("Opening f1.txt and putting \"Fun Fun\" into s.\n");
   strcpy(s, "Fun Fun\n");
   fd = open("f1.txt", O_RDONLY);
   sleep(1);

   printf("Removing f1.txt\n");
   remove("f1.txt");
   sleep(1);

   printf("Listing f1.txt, and reading 10 bytes from the open file
descriptor.\n");
   system("ls -l f1.txt");
   i = read(fd, s, 10);
   s[i] = '\0';
   printf("Read returned %d: %d %s\n", i, fd, s);
   return 0;
}
```

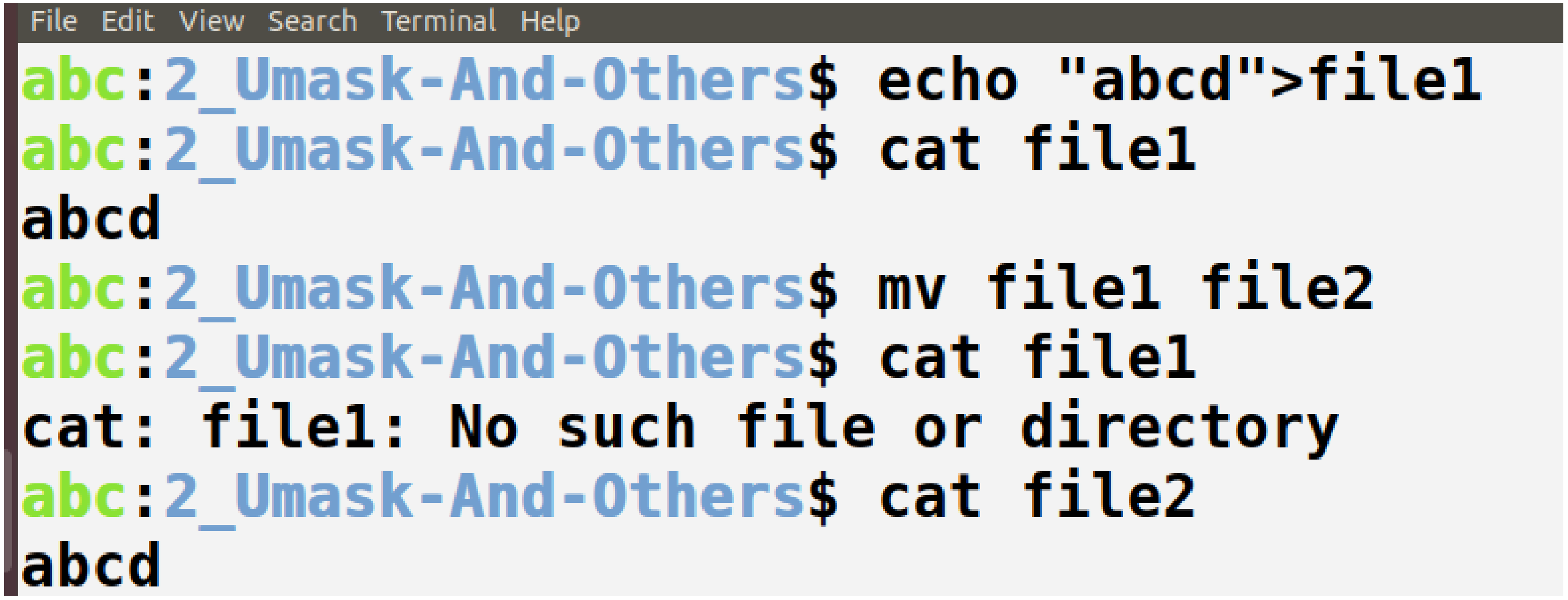
remove

```
abc:2_Umask-And-Others$ ./o2
Opening f1.txt and putting "Fun Fun" into s.
Removing f1.txt
Listing f1.txt, and reading 10 bytes from the open file descriptor.
ls: cannot access 'f1.txt': No such file or directory
Read returned -1: -1 Fun Fun
```

- What happened?
- First, the **open()** call failed and returned -1. Thus, the **read()** call also failed and returned -1.
- Since the **read** call failed, the bytes of **s** were never overwritten - thus when we printed them out, we got "Fun Fun."
- Make sure you understand this code and its output.
- It is deterministic -- we are not getting segmentation violations or random behavior with these calls -- we are simply getting well-defined errors in our system calls.

rename

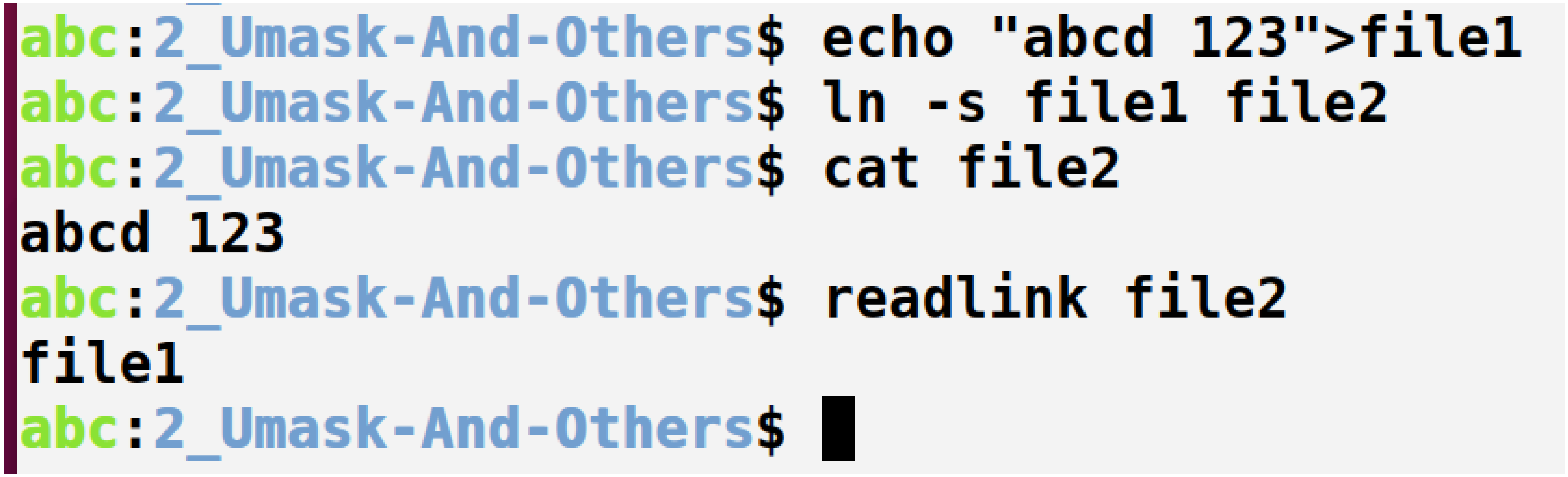
- rename() renames a file moving it between directories if required.
- Any other hardlinks to the file are unaffected.
- rename(char *f1, char *f2) works just like:
- mv f1 f2



```
abc:2_Umask-And-Others$ echo "abcd">file1
abc:2_Umask-And-Others$ cat file1
abcd
abc:2_Umask-And-Others$ mv file1 file2
abc:2_Umask-And-Others$ cat file1
cat: file1: No such file or directory
abc:2_Umask-And-Others$ cat file2
abcd
```

symlink and readlink

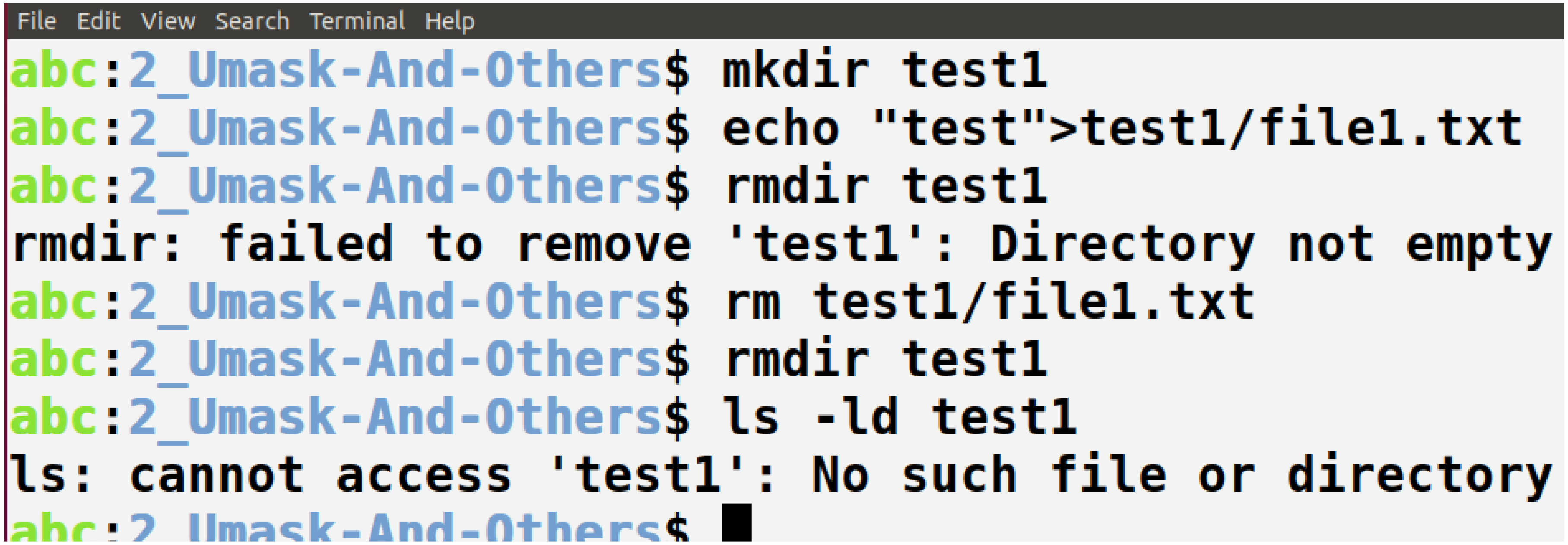
- Symlink() creates a symbolic link
- Readlink() reads the symbolic link pathname



```
abc:2_Umask-And-Others$ echo "abcd 123">file1
abc:2_Umask-And-Others$ ln -s file1 file2
abc:2_Umask-And-Others$ cat file2
abcd 123
abc:2_Umask-And-Others$ readlink file2
file1
abc:2_Umask-And-Others$ 
```

mkdir and rmdir

- Mkdir attempts to create a directory
- Rmdir deletes a directory, which must be empty



```
abc:2_Umask-And-Others$ mkdir test1
abc:2_Umask-And-Others$ echo "test">test1/file1.txt
abc:2_Umask-And-Others$ rmdir test1
rmdir: failed to remove 'test1': Directory not empty
abc:2_Umask-And-Others$ rm test1/file1.txt
abc:2_Umask-And-Others$ rmdir test1
abc:2_Umask-And-Others$ ls -ld test1
ls: cannot access 'test1': No such file or directory
abc:2_Umask-And-Others$
```

utime

- Change last access and modification times
- This system call lets you change the time fields of a file's inode.
- It looks like it should be illegal (for example, one could write a program to make it look like one has finished his homework on time...)

utime

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

```
struct utimbuf {
        time_t actime;      /* access time */
        time_t modtime;     /* modification time */
    };
```

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

```
struct timeval {
        long tv_sec;      /* seconds */
        long tv_usec;     /* microseconds */
    };
```

utime

```
struct tm {
    int tm_sec;         /* seconds */
    int tm_min;         /* minutes */
    int tm_hour;        /* hours */
    int tm_mday;        /* day of the month */
    int tm_mon;         /* month */
    int tm_year;        /* year */
    int tm_wday;        /* day of the week */
    int tm_yday;        /* day in the year */
    int tm_isdst;       /* daylight saving time */
};
```

chdir and getcwd:

- **chdir, getcwd**: These are like the shell commands **cd** and **pwd**.