

# File System, Part 3: Permissions

goldcase edited this page 12 days ago · 17 revisions

## Does a directory have an inode too?

Yes! Though a better way to think about this, is that a directory (like a file) *is* an inode (with some data - the directory name and inode contents). It just happens to be a special kind of inode.

From [Wikipedia](#):

Unix directories are lists of association structures, each of which contains one filename and one inode number.

Remember, inodes don't contain filenames--only other file metadata.

## How can I have the same file appear in two different places in my file system?

First remember that a file name != the file. Think of the inode as 'the file' and a directory as just a list of names with each name mapped to an inode number. Some of those inodes may be regular file inodes, others may be directory inodes.

If we already have a file on a file system we can create another link to the same inode using the 'ln' command

```
$ ln file1.txt blip.txt
```

However blip.txt *is* the same file; if I edit blip I'm editing the same file as 'file1.txt!' We can prove this by showing that both file names refer to the same inode:

```
$ ls -li file1.txt blip.txt
134235 file1.txt
134235 blip.txt
```

These kinds of links (aka directory entries) are called 'hard links'

The equivalent C call is `link`

```
link(const char *path1, const char *path2);

link("file1.txt", "blip.txt");
```

For simplicity the above examples made hard links inside the same directory however

Edit

New Page

▼ Pages 51

Home

#Example Markdown

#Informal Glossary

#Piazza: When And How to Ask For Help

C Programming, Part 1: Introduction

C Programming, Part 2: Text Input And Output

C Programming, Part 3: Common Gotchas

C Programming, Part 4: Debugging

Deadlock, Part 1: Resource Allocation Graph

Deadlock, Part 2: Deadlock Conditions

File System, Part 1: Introduction

File System, Part 2: Files are inodes (everything else is just data...)


File System, Part 3: Permissions

File System, Part 4: Working with directories

File System, Part 5: Virtual file systems

Show 36 more pages...

Clone this wiki locally

 Clone in Desktop

hard links can be created anywhere inside the same filesystem.

## What happens when I `rm` (remove) a file?

When you remove a file (using `rm` or `unlink` ) you are removing an inode reference from a directory. However the inode may still be referenced from other directories. In order to determine if the contents of the file are still required, each inode keeps a reference count that is updated whenever a new link is created or destroyed.

## Case study: Back up software that minimizes file duplication

An example use of hard-links is to efficiently create multiple archives of a file system at different points in time. Once the archive area has a copy of a particular file, then future archives can re-use these archive files rather than creating a duplicate file. Apple's "Time Machine" software does this.

## Can I create hard links to directories as well as regular files?

No. Well yes. Not really... Actually you didn't really want to do this, did you? The POSIX standard says no you may not! The `ln` command will only allow root to do this and only if you provide the `-d` option. However even root may not be able to perform this because most filesystems prevent it!

Why? The integrity of the file system assumes the directory structure (excluding softlinks which we will talk about later) is a non-cyclic tree that is reachable from the root directory. It becomes expensive to enforce or verify this constraint if directory linking is allowed. Breaking these assumptions can cause file integrity tools to not be able to repair the file system. Recursive searches potentially never terminate and directories can have more than one parent but `..` can only refer to a single parent. All in all, a bad idea.

## How do I change the permissions on a file?

Use `chmod` (short for "change the file mode bits")

There is a system call, `int chmod(const char *path, mode_t mode);` but we will concentrate on the shell command. There's two common ways to use `chmod` ; with an octal value or with a symbolic string:

```
$ chmod 644 file1
$ chmod 755 file2
$ chmod 700 file3
$ chmod ugo-w file4
$ chmod o-rx file4
```

The base-8 ('octal') digits describe the permissions for each role: The user who owns the file, the group and everyone else. The octal number is the sum of three values given to the three types of permission: read(4), write(2), execute(1)

Example: `chmod 755 myfile`

- `r + w + x = digit`
- user has 4+2+1, full permission
- group has 4+0+1, read and execute permission
- all users have 4+0+1, read and execute permission

## How do I read the permission string from ls?

Use ``ls -l'`. Note that the permissions will output in the format `'drwxrwxrwx'`. The first character indicates the type of file type. Possible values for the first character:

- `(-)` regular file
- `(d)` directory
- `(c)` character device file\
- `(l)` symbolic link
- `(p)` pipe
- `(b)` block device
- `(s)` socket

## What is sudo?

Use `sudo` to become the admin on the machine. e.g. Normally (unless explicitly specified in the `'/etc/fstab'` file, you need root access to mount a filesystem). `sudo` can be used to temporarily run a command as root (provided the user has sudo privileges)

```
$ sudo mount /dev/sda2 /stuff/mydisk
$ sudo adduser fred
```

## How do I change ownership of a file?

Use `chown username filename`

## How do I set permissions from code?

```
chmod(const char *path, mode_t mode);
```

## Why are some files 'setuid'? What does this mean?

The set-user-ID-on-execution bit changes the user associated with the process when the file is run. This is typically used for commands that need to run as root but are executed by non-root users. An example of this is `sudo`

The set-group-ID-on-execution changes the group under which the process is run.

## Why are they useful?

The most common usecase is so that the user can have root(admin) access for the duration of the program.

## What permissions does sudo run as ?

```
$ ls -l /usr/bin/sudo
-r-s--x--x  1 root  wheel  327920 Oct 24 09:04 /usr/bin/sudo
```

The 's' bit means execute and set-uid; the effective userid of the process will be different from the parent process. In this example it will be root

## What's the difference between getuid() and geteuid()?

- `getuid` returns the real user id (zero if logged in as root)
- `geteuid` returns the effective userid (zero if acting as root, e.g. due to the setuid flag set on a program)

## How do I ensure only privileged users can run my code?

- Check the effective permissions of the user by calling `geteuid()` . A return value of zero means the program is running effectively as root.

[Go to File System: Part 4](#)

Legal and Licensing information: Unless otherwise specified, submitted content to the wiki must be original work (including text, java code, and media) and you provide this material under a [Creative Commons License](#). If you are not the copyright holder, please give proper attribution and credit to existing content and ensure that you have license to include the materials.

