...making Linux just a little more fun!

<-- prev | next -->

Q & A: The difference between hard and soft links

By Lew Pitcher

I participate in about 30 usenet newsgroups, and in a virtual LUG, and a number of questions keep coming up. I've answered a few of these questions often enough to have 'canned' an answer, which I modify, depending on the circumstances.

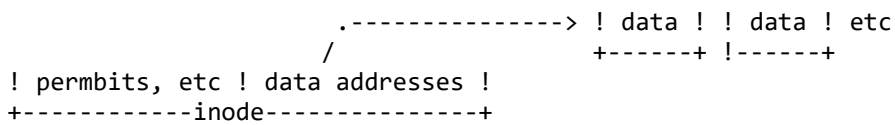
Here's one, now...

Q: Can someone give me a simple explanation of the difference between a soft link and a hard link? The documentation I've read mention these links but make no strong explanations of their meaning and how/when to use them. Thanks!

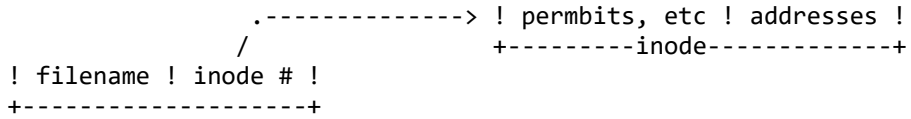
A: OK, I'll give it a try...

Unix files consist of two parts: the data part and the filename part.

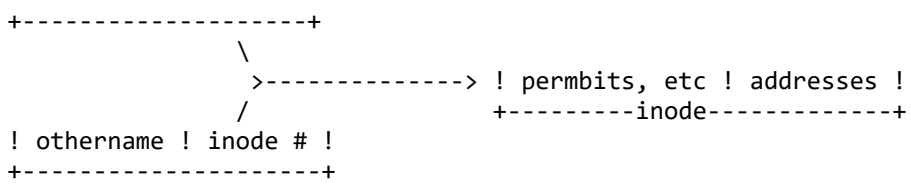
The data part is associated with something called an 'inode'. The inode carries the map of where the data is, the file permissions, etc. for the data.



The filename part carries a name and an associated inode number.



More than one filename can reference the same inode number; these files are said to be 'hard linked' together.



On the other hand, there's a special file type whose data part carries a path to another file. Since it is a special file, the OS recognizes the data as a path, and redirects opens, reads, and writes so that, instead of accessing the

data within the special file, they access the data in the file named by the data in the special file. This special file is called a 'soft link' or a 'symbolic link' (aka a 'symlink').

Now, the filename part of the file is stored in a special file of its own along with the filename parts of other files; this special file is called a directory. The directory, as a file, is just an array of filename parts of other files.

When a directory is built, it is initially populated with the filename parts of two special files: the '.' and '..' files. The filename part for the '.' file is populated with the inode# of the directory file in which the entry has been made; '.' is a hardlink to the file that implements the current directory.

The filename part for the '..' file is populated with the inode# of the directory file that contains the filename part of the current directory file. '..' is a hardlink to the file that implements the immediate parent of the current directory.

The 'ln' command knows how to build hardlinks and softlinks; the 'mkdir' command knows how to build directories (the OS takes care of the above hardlinks).

There are restrictions on what can be hardlinked (both links must reside on the same filesystem, the source file must exist, etc.) that are not applicable to softlinks (source and target can be on seperate file systems, source does not have to exist, etc.). OTOH, softlinks have other restrictions not shared by hardlinks (additional I/O necessary to complete file access, additional storage taken up by softlink file's data, etc.)

In other words, there's tradeoffs with each.

Now, let's demonstrate some of this...

ln in action

Let's start off with an empty directory, and create a file in it

```
~/directory $ ls -lia
total 3
  73477 drwxr-xr-x   2 lpitcher users        1024 Mar 11 20:16 .
  91804 drwxr-xr-x  29 lpitcher users        2048 Mar 11 20:16 ..

~/directory $ echo "This is a file" >basic.file

~/directory $ ls -lia
total 4
  73477 drwxr-xr-x   2 lpitcher users        1024 Mar 11 20:17 .
  91804 drwxr-xr-x  29 lpitcher users        2048 Mar 11 20:16 ..
  73478 -rw-r--r--   1 lpitcher users          15 Mar 11 20:17 basic.file

~/directory $ cat basic.file
This is a file
```

Now, let's make a hardlink to the file

```
~/directory $ ln basic.file hardlink.file

~/directory $ ls -lia
total 5
  73477 drwxr-xr-x   2 lpitcher users        1024 Mar 11 20:20 .
  91804 drwxr-xr-x  29 lpitcher users        2048 Mar 11 20:18 ..
  73478 -rw-r--r--   2 lpitcher users          15 Mar 11 20:17 basic.file
  73478 -rw-r--r--   2 lpitcher users          15 Mar 11 20:17 hardlink.file

~/directory $ cat hardlink.file
This is a file
```

We see that:

a. `hardlink.file` shares the same inode (73478) as basic.file
b. `hardlink.file` shares the same data as basic.file

If we change the permissions on `basic.file`:

```
~/directory $ chmod a+w basic.file

~/directory $ ls -lia
total 5
  73477 drwxr-xr-x   2 lpitcher users        1024 Mar 11 20:20 .
  91804 drwxr-xr-x  29 lpitcher users        2048 Mar 11 20:18 ..
  73478 -rw-rw-rw-   2 lpitcher users          15 Mar 11 20:17 basic.file
  73478 -rw-rw-rw-   2 lpitcher users          15 Mar 11 20:17 hardlink.file
```

then the same permissions change on `hardlink.file`.

The two files (`basic.file` and `hardlink.file`) share the same inode and data, but have different file names.

Let's now make a softlink to the original file:

```
~/directory $ ln -s basic.file softlink.file

~/directory $ ls -lia
total 5
  73477 drwxr-xr-x   2 lpitcher users        1024 Mar 11 20:24 .
  91804 drwxr-xr-x  29 lpitcher users        2048 Mar 11 20:18 ..
  73478 -rw-rw-rw-   2 lpitcher users          15 Mar 11 20:17 basic.file
  73478 -rw-rw-rw-   2 lpitcher users          15 Mar 11 20:17 hardlink.file
  73479 lrwxrwxrwx   1 lpitcher users          10 Mar 11 20:24 softlink.file -> basic.file
```

```
~/directory $ cat softlink.file
This is a file
```

Here, we see that although `softlink.file` accesses the same data as `basic.file` and `hardlink.file`, it does not share the same inode (73479 vs 73478), nor does it exhibit the same file permissions. It does show a new permission bit: the 'l' (softlink) bit.

If we delete `basic.file`:

```
~/directory $ rm basic.file

~/directory $ ls -lia
total 4
  73477 drwxr-xr-x   2 lpitcher users        1024 Mar 11 20:27 .
  91804 drwxr-xr-x  29 lpitcher users        2048 Mar 11 20:18 ..
  73478 -rw-rw-rw-   1 lpitcher users          15 Mar 11 20:17 hardlink.file
  73479 lrwxrwxrwx   1 lpitcher users          10 Mar 11 20:24 softlink.file -> basic.file
```

then we lose the ability to access the linked data through the softlink:

```
~/directory $ cat softlink.file
cat: softlink.file: No such file or directory
```

However, we still have access to the original data through the hardlink:

```
~/directory $ cat hardlink.file
This is a file
```

You will notice that when we deleted the original file, the hardlink didn't vanish. Similarly, if we had deleted the softlink, the original file wouldn't have vanished.

A further note with respect to hardlink files

When deleting files, the data part isn't disposed of until all the filename parts have been deleted. There's a count in the inode that indicates how many filenames point to this file, and that count is decremented by 1 each time one of those filenames is deleted. When the count makes it to zero, the inode and its associated data are deleted.

By the way, the count also reflects how many times the file has been opened without being closed (in other words, how many references to the file are still active). This has some ramifications which aren't obvious at first: you can delete a file so that no "filename" part points to the inode, without releasing the space for the data part of the file, because the file is still open.

Have you ever found yourself in this position: you notice that `/var/log/messages` (or some other syslog-owned file) has grown too big, and you

```
rm /var/log/messages
touch /var/log/messages
```

to reclaim the space, but the used space doesn't reappear? This is because, although you've deleted the filename part, there's a process that's got the data part open still (syslogd), and the OS won't release the space for the data until the process closes it. In order to complete your space reclamation, you have to

```
kill -SIGHUP `cat /var/run/syslogd.pid`
```

to get syslogd to close and reopen the file.

You can use this to your advantage in programs: have you ever wondered how you could hide a temporary file? Well, you could do the following:

```
{
   FILE *fp;

   fp = fopen("some.hidden.file","w");
   unlink("some.hidden.file"); /* deletes the filename part */

   /* some.hidden.file no longer has a filename and is truely hidden */
   fprintf(fp,"This data won't be found\n"); /* access the data part */
   /*etc*/
   fclose(fp); /* finally release the data part */
}
```

Canadian by birth, and living in Brampton, Ontario, I am a career techie working at a major Canadian bank. For over 25 years, I've programmed on all sorts of systems, from Z80 CP/M up to OS/390. Primarily, I develop OS/390 MVS applications for banking services, and have incorporated Linux into my development environment.

Published in Issue 105 of Linux Gazette, August 2004