Symlinks vs Hardlinks and how to create them

n article that explains the differences between symlinks and ardlinks and how to create them

ritten by Benjamin Cane (https://twitter.com/madflojo) on 2013-10-10 00:08 | 4 min read

a previous article I covered a little bit about Symlinks and Hardlinks but I ver really explained what they are or how to create them. Today I am going cover how to create both Symlinks and Hardlinks and what the difference is between the two.

What are Symlinks and Hardlinks

Hard Links

In Linux when you perform an listing in a directory the listing is actually is a list of references that map to an inode. When you create a hard link the hard link is yet another reference to the same inode as the original file. A hard link allows a user to create two exact files without having to duplicate the data on disk. However unlike creating a copy, if you modify the hard link you are in turn modifying the original file as well as they both reference the same inode.

Hard links have some limitations however, in most **(but not all)** Unix/Linux distributions hard links cannot be created for directories. Hard links are also not allowed to cross file systems. Inodes are unique to each file system, because a hard link is a second reference to the same inode they cannot be used across multiple file systems.

ymbolic Links (aka Symlinks)

ymbolic link is similar to a hard link in that it is used to link to an already sting file, however it is very different in its implementation. A symbolic k is not a reference to an inode but rather an pointer that redirects to other file or directory.

mlinks do not have the same restrictions as hardlinks, a symlink can be ed to reference a file in another file system. Symlinks can also be used to int to a directory.

hen to use a Symbolic link or a Hard Link

here are many cases when you would want to use a symlink and really only a few where you would want to use a hard link. Below are a few use cases for both that will help explain when and how to use them.

Hard Link Use Case:

I have a system that runs multiple applications, one application to receive files, and two more applications to process the files. In my case I want all of my files to land in one directory, lets say `/data/repository/` and stay there until they are done processing. By having the files always residing in the repository directory I am able to tell how many files I have pending processing.

While I want all of my files to stay in the repository I also don't want both applications to process every file. I want to control which application can process which files. To do this I can create two more directories `/data/app1`

and `/data/app2`. When I receive a file into the repository I can simply create a hard link in `/data/app1` or `/data/app2` and this will allow those applications to process the file without having to create a copy of the file. Once the application has finished processing the file it can remove the hard link leaving the original file in place.

e benefits of using hard links in this scenario are:

- I can have a file in two places without having to duplicate data on disk
- Each Hard Link does not increase the number of inodes, this will help if this system processes a large amount of files
- I can use the stat or find commands to identify the current number of hard links to a file to identify which files have finished processing

theory I could have also used a symbolic link for this use case however if I l I would lose the 2nd and 3rd benefits.

mbolic Link Use Case:

n my system I have multiple application running and all of those applications have a directory within the applications home directory called **logs**. However as a best practice in my environment all application logs should be stored in a `/logs` filesystem. This provides quick and easy access to look through application logs without having to search for them.

The applications cannot be configured to write their log files inside of `/logs`; however I can simply remove the applications logs directory and replace it with a symlink to `/logs/<application name>`. By doing this I am ensuring that all of my application logs are in `/logs` without much hassle.

Creating a Symbolic Link

Now that we have covered when and why to use hard links and symlinks lets cover how to create them.

Symlinking a Directory

Creating a symlink is easy, simply run `ln -s` and specify the **source (directory you want to symlink to)** and the **linkname (the symlinks name)**.

Format:


```
ln -s <source> <linkname>
```

In the below example I will be creating a symlink named dir2 that links to dir1.

Example:

```
ln -s /var/tmp/testarea/dir1 dir2

ls -l
total 4
drwxrwxr-x 2 madflojo madflojo 4096 Oct 8 06:38 dir1
lrwxrwxrwx 1 madflojo madflojo 4 Oct 8 06:38 dir2 -> dir1
```

As you can see from the output of the ls command the link looks very different from the directory. The format **linkname -> target** is how all symlinks show, whether they are linking to a file or a directory. You may also notice that the file type shows as **l** for symbolic link.

Symlinking a File

Creating a symlink for a file is the same process as creating a symlink for a directory.

```
$ cat file.real
This is file.real

$ ln -s file.real file.sym

$ ls -l
total 4
-rw-rw-r-- 1 madflojo madflojo 18 Oct 9 18:49 file.real
lrwxrwxrwx 1 madflojo madflojo 9 Oct 9 18:49 file.sym -> file.real

$ cat file.sym
is is file.real
```


you can see the symlink for a file appears the same as a symlink for a ectory. This is due to the fact that a symbolic link is the same whether it ints to a file or a directory.

reating Hard Links

eating a hard link can be done via the `ln` command as well. With hard ks however we will not be using the `-s` flag.

rmat:

```
$ ln <source> <linkname>
```

Example:

```
$ ln file.real file.hard

$ ls -l
total 8
-rw-rw-r-- 2 madflojo madflojo 18 Oct 9 18:49 file.hard
-rw-rw-r-- 2 madflojo madflojo 18 Oct 9 18:49 file.real
lrwxrwxrwx 1 madflojo madflojo 9 Oct 9 18:49 file.sym -> file.real

$ cat file.hard
This is file.real
```

As you can see the hard link will also allow you to link to the contents of **file.real** but the output of ls is very different. As far as the file system is concerned there is no difference between **file.real** and **file.hard** as they are both references to the same **inode (10097087)**.

The only way to even show that **file.hard** is a hard link is to look at the links unt in the inode. You can do this with the stat command.


```
stat file.hard
ile: file.hard
ize: 18 Blocks: 8 IO Block: 4096 regular file
vice: fc00h/64512d Inode: 10097087 Links: 2
cess: (0664/-rw-rw-r--) Uid: ( 1001/madflojo) Gid: ( 1001/madflojo)
cess: 2013-10-09 18:56:36.079158024 -0700
dify: 2013-10-09 18:49:46.477126917 -0700
ange: 2013-10-09 18:56:29.903127399 -0700
irth: -
```

hile this shows that the inode for **file.hard** has 2 links it does not cessarily show which is the original file and which is the hard link.

ith hard links you can delete the original file

Since a hard link is simply a reference to an inode, you can actually delete the original file and the data will still be available via the hard link. Where as with a symlink if the target is deleted the symlink will no longer be able to provide data.

```
$ rm file.real

$ ls -l
total 4
-rw-rw-r-- 1 madflojo madflojo 18 Oct 9 18:49 file.hard
lrwxrwxrwx 1 madflojo madflojo 9 Oct 9 18:49 file.sym -> file.real

$ cat file.hard
This is file.real

$ cat file.sym
cat: file.sym: No such file or directory
```

The symlink is now a "broken" link where as the hard link is now indistinguishable from any other file.