Symbolic links and hard links: creating, updating, deleting and all that

Filed under: Webdev (" attr(href) ")—**Tags**: bash (" attr(href) "), shell (" attr(href) ")

Symbolic link grokking post.

Join me on a miraculous journey to unlock the ancient mystery that is the difference between soft link and a hard link. Plus a ton of examples.

Terminology

Symlink, symbolic link, or soft link:

> In computing, a symbolic link (also symlink or soft link) is a special type of file that contains a reference to another file or directory in the form of an absolute or relative path and that affects pathname resolution. —Wikipedia (" attr(href) ")

Hard link:

> In computing, a hard link is a directory entry that associates a name with a file on a file system. —Wikipedia (" attr(href) ")

hmm

Let's look closer.

The gist of links

Link is to have a directory or file to be in two places at the same time.

It's hard to pull analogies from real life, but lets try; it's like having a binder in a shelf **A**, and you'd take a wormhole gun, set it to make a wormhole to shelf **B**, point it at the binder in shelf **A** and pull the trigger. Now the binder exists in two places at the same time, and if you remove something from either one of the binders, the change affects both binders, cause they're the same binder. Deep.

Syntax

Hard link:

```
┌── link, ln - makes links
│
│                          ┌── the path to the intended link
│                          │   can use . or ~ or other relative paths
│                    ┌─────┴──────┐
ln /path/to/original /path/to/link
   └───────┬───────┘
           └── the path to the original file/folder
               can use . or ~ or other relative paths
```

Symlink:

```
┌── link, ln - makes links
│   ┌── Create a symbolic link
│   │                         ┌── the path to the intended symlink
│   │                         │   can use . or ~ or other relative paths
│   │                   ┌─────┴───────┐
ln -s /path/to/original /path/to/symlink
      └───────┬───────┘
              └── the path to the original file/folder
                  can use . or ~ or other relative paths
```

This super clear explanation is thanks to this (" attr(href) ") answer. More link syntax examples below.

The difference between soft link and a hard link

Like the name suggest, **soft link** is more fragile than **hard link**.

Here's stuff one can do with **hard links** that would break a soft link:

- Both ends of hard link can be renamed and the link would persist
- Both ends can be moved around and the link persists
- The source directory of hard link can be deleted and the target link would persist

Stuff that soft links do that hard links can't:

- Reach from a filesystem to another, e.g. from your host machine to a dev VM

Maybe there's more that I just don't know, drop a comment if you know.

Technical explanation

There are these thing called **inodes** (Index Nodes) inside your computer (might as well call it black magic), which represent files and directories. A file is really a link to an inode, hard link creates another file with a link to the same inode.

> An inode is a data structure on a traditional Unix-style file system such as UFS or ext3. An inode stores basic information about a regular file, directory, or other file system object.

Make some test files with dummy content:

```
$ mkdir link-test \
&& cd link-test \
&& echo Banana > File_A \
&& echo Apple > File_B \
&& echo Orange > File_C \
&& mkdir Dir_A \
&& mv File_C Dir_A/File_C
```

That gives you :

```
├── Dir_A
│   └── File_C
├── File_A
└── File_B
```

You may see the inode number of `File_A` :

```
$ ls -i File_A
51882811 File_A
```

`stat` command shows all the attributes stored into the inode. So basically, that's the inode, as far as us mere mortals are concerned:

```
$ stat -x File_A
  File: "File_A"
  Size: 7            FileType: Regular File
  Mode: (0644/-rw-r--r--)         Uid: (  501/   hilja)  Gid: (   20/   staff)
Device: 1,4   Inode: 51882811    Links: 1
Access: Tue Jan 20 09:14:04 2015
Modify: Tue Jan 20 09:14:04 2015
Change: Tue Jan 20 09:14:04 2015
```

This is on OS X, the output of `stat` might look a bit different in Linux. See about the OS X stat here (" attr(href) ").

Files can be deleted based on their inode number:

```
$ find . -inum 51882811 -exec rm -i {} ;
```

More here (" attr(href) ").

Visual explanation

In hard links the link sits kind of between the inode and the file:

```
$ ln File_A File_B

┌──────────┐     ┌──────────┐
|  File_A  |     |  File_B  |
└─────┬────┘     └─────┬────┘
      |                |
      └─────┐  ┌─link──┘
            |  |
      ┌─────┴──┴─────┐
      |    inode     |
      | representing |
      |    File_A    |
      └──────────────┘
```

Where as soft link has nothing to do with the inode, it links between the files like so:

```
$ ln -s File_A File_B

┌──────────┐          ┌──────────┐
|  File_A  ├───link───|  File_B  |
└─────┬────┘          └──────────┘
      |
      └─────┐
            |
      ┌─────┴────────┐
      |    inode     |
      | representing |
      |    File_A    |
      └──────────────┘
```

I hope I'm making sense here, please correct me in the comments if you have a better explanation.

More link syntax examples

If link target file name is not specified, the originals name will be used:

```
$ cd link-test
$ ln -s /path/to/File_B .
```

That just grabs the `File_B` from an absolute path and jugs the link into the current directory.

Any sort of a relative path can be used with links if needed:

```
$ ln -s ../../File_B .
```

View link

In our test directory (that we made earlier), let’s make some links to test. First, symlink:

```
$ cd link-test
$ ln -s File_A File_A_link
```

And a hard link:

```
$ ln File_B File_B_link
```

Now listing with the `-l` flag show the symlinks:

```
$ ls -l
-rw-r--r--  1 hilja  staff  7 20 Jan 09:14 File_A
lrwxr-xr-x  1 hilja  staff  6 21 Jan 10:32 File_A_link -> File_A
-rw-r--r--  2 hilja  staff  6 20 Jan 09:14 File_B
-rw-r--r--  2 hilja  staff  6 20 Jan 09:14 File_B_link
```

Or with `readlink` , this might come handy if writing a larger script:

```
$ readlink File_A_link
File_A
```

Either one of these won’t show the hard link. But notice the number `2` before the user name, that reads: the file has two names. Meaning the original and the link. We can view the duplicate files:

```
$ find . -xdev -samefile File_B
./File_B
./File_B_link
```

Here’s the breakdown of the command:

- `.` – Where to search, current directory in this case
- `-xdev` – “Don’t descend directories on other filesystems.”
- `-samefile name` – “True if the file is a hard link to name. If the command option -L is speci-fied, it is also true if the file is a symbolic link and points to name.”
- `File_B` – What to search

Remove hard link

```
$ rm File_B
$ ls
File_A    File_A_link    File_B_link
```

The `File_B_link` persists, essentially it’s now the original `File_B` .

Remove soft link

Either, remove the link:

```
$ rm File_A_link
```

Or use `unlink` :

```
$ unlink File_A_link
```

Remove soft link from a directory

There is a small gotcha here, link the directory and try to remove it, and bash will nag you:

```
$ ln -s Dir_A Dir_A_link
$ unlink Dir_A_link/
unlink: Dir_A_link/: is a directory
```

The gotcha being the forward slash after the directory name, leave out and it won’t nag you anything.

```
$ unlink Dir_A_link
```

Update a soft link

Now the `File_A_link` links to `File_A` :

```
$ ls -l File_A_link
lrwxr-xr-x  1 hilja  staff  6 21 Jan 15:09 File_A_link -> File_A
```

Let’s update the link to point to `Dir_A/File_C` :

```
ln -nsf Dir_A/File_C File_A_link
```

If symbolic link

Symlink can be tested with an if statement. The syntax is:

`[ -h FILE ]` True if FILE exists and is a symbolic link.

Example script:

```
is_link() {
    local file=$1
    [[ -h $file ]]
}

if is_link path/to/file; then
    # Do something...
fi
```

That’s the basics of links. Please leave a comment if you have something to add :)

Related posts

- Git: rebasing workflow and resolving merge conflicts (" attr(href) ")
- Git: move commits between branches with cherry-pick (" attr(href) ")
- Git: checkout files or directories from another branch (" attr(href) ")
- Git: handy branching commands (" attr(href) ")
- Copy files from local to remote, and tab complete in the remote server (" attr(href) ")
- Basics of Bash programming, aka shell scripting (" attr(href) ")
- Replace strings in files with sed, the bash command (" attr(href) ")
- Upgrade to bash 4 in Mac OS X (" attr(href) ")

Comments

bajar musica (" attr(href) ") says: 27/05/2017 at 01:55 (" attr(href) ")

You’re so awesome! I do not suppose I’ve read through something
like this before. So wonderful to discover another person with some
original thoughts on this topic. Seriously.. thanks for starting
this up. This web site is something that is required on
the internet, someone with some originality!

Vivek Singh says: 07/06/2017 at 17:23 (" attr(href) ")

I have a question :

I have 2 unix shell scripts : A.ksh and B.ksh

In current scenario, A.ksh is running in production server.
B.ksh has been created from A.ksh after removing some of its phases.

Let’s say I want to run B.ksh for a month and then get back to running A.ksh after this period.

Now , what I want is to link A.ksh to B.ksh, so that even on running A.ksh, B.ksh gets called.

And after a month period, I have to remove this link, so that I can run A.ksh again with its original content.

Problem is that after linking A.ksh to B.ksh, contents of A.ksh are getting overwritten with that of B.ksh.

What I want is to just link these 2 scripts and after unlinking, there should not be any content change in either of them.

Thanks a lot in advance!!!

Merlin Beedell says: 30/06/2017 at 11:59 (" attr(href) ")

Is it possible to view the original directory content that is ‘masked’ when that same directory is soft-linked to a different path?

Comment

Anything goes, just be nice

Name

Slartibartfast

Email

Will not be published

Website

Your potential website

Send it →

Club-Mate, the beverage → club-mate.fi (" attr(href) ")

- A blog of Hiljá Studio (" attr(href) "). Lolling since 2007!
- @hiljaa (" attr(href) ")
- And then she was like, all rights reserved and yadda yadda © 2017
- I can write pretty much anything here, no one will notice...

Subscribe & Follow

- clubmate.fi/feed (" attr(href) ")
- @hiljaa (" attr(href) ")

Tag blob

.htaccess (" attr(href) "), animate (" attr(href) "), array (" attr(href) "), bash (" attr(href) "), beveled (" attr(href) "), bower (" attr(href) "), compass (" attr(href) "), conditional (" attr(href) "), cookie (" attr(href) "), custom fields (" attr(href) "), custom taxonomy (" attr(href) "), database (" attr(href) "), functions.php (" attr(href) "), git (" attr(href) "), grunt (" attr(href) "), gulp (" attr(href) "), homebrew (" attr(href) "), javascript (" attr(href) "), jquery (" attr(href) "), layout (" attr(href) "), loop (" attr(href) "), media queries (" attr(href) "), meta box (" attr(href) "), mixin (" attr(href) "), nginx (" attr(href) "), npm (" attr(href) "), object (" attr(href) "), pagespeed (" attr(href) "), php (" attr(href) "), plugins (" attr(href) "), preg_match (" attr(href) "), regular expression (" attr(href) "), RequireJS (" attr(href) "), rsync (" attr(href) "), sass (" attr(href) "), scss (" attr(href) "), server (" attr(href) "), shell (" attr(href) "), snippets (" attr(href) "), ssh (" attr(href) "), str_replace (" attr(href) "), sublime text (" attr(href) "), term (" attr(href) "), text editor (" attr(href) "), title (" attr(href) ")