Learn Linux, 101: Create and change hard and symbolic links

Use multiple names for the same file

Ian Shields
Senior Programmer
IBM

01 June 2010

Learn how to create and manage hard and symbolic links to files on your Linux® system. You can use the material in this article to study for the LPI 101 exam for Linux system administrator certification, or just to explore the differences between hard and soft, or symbolic, links and the best ways to link to files, as opposed to copying files.

View more content in this series

Overview

In this article, learn to create and manage hard and symbolic links. Learn to:

- Create hard or soft links
- Identify links and know their type
- Understand the difference between copying and linking files
- Use links for system administration tasks

This article helps you prepare for Objective 104.6 in Topic 104 of the Linux Professional Institute's Junior Level Administration (LPIC-1) exam 101. The objective has a weight of 2.

Prerequisites

About this series

This series of articles helps you learn Linux system administration tasks. You can also use the material in these articles to prepare for Linux Professional Institute Certification level 1 (LPIC-1) exams.

See our developerWorks roadmap for LPIC-1 for a description of and link to each article in this series. The roadmap is in progress and reflects the latest (April 2009) objectives for the LPIC-1 exams: as we complete articles, we add them to the roadmap. In the meantime, though, you can find earlier versions of similar material, supporting previous LPIC-1 objectives prior to April 2009, in our LPI certification exam prep tutorials.

To get the most from the articles in this series, you should have a basic knowledge of Linux and a working Linux system on which you can practice the commands covered in this article. Sometimes different versions of a program will format output differently, so your results may not always look exactly like the listings and figures shown here. In particular, much of the output we show is highly dependent on the packages that are already installed on our systems. Your own output may be quite different, but you should be able to recognize the important commonalities.

Introducing links

Connect with Ian

Ian is one of our most popular and prolific authors. Browse all of Ian's articles on developerWorks. Check out Ian's profile and connect with him, other authors, and fellow readers in My developerWorks.

On a storage device, a file or directory is contained in a collection of *blocks*. Information about a file is contained in an *inode*, which records information such as the owner, when the file was last accessed, how large it is, whether it is a directory or not, and who can read from or write to it. The inode number is also known as the *file serial number* and is unique within a particular filesystem. A *directory entry* contains a name for a file or directory and a pointer to the inode where the information about the file or directory is stored.

A *link* is simply an additional directory entry for a file or directory, allowing two or more names for the same thing. A *hard link* is a directory entry that points to an inode, while a *soft link* or *symbolic link* is a directory entry that points to an inode that provides the name of another directory entry. The exact mechanism for storing the second name may depend on both the file system and the length of the name. Symbolic links are also called *symlinks*.

You can create hard links only for files and not for directories. The exception is the special directory entries in a directory for the directory itself and for its parent (. and ..), which are hard links that maintain the count of the number of subdirectories. Because hard links point to an inode, and inodes are only unique within a particular file system, hard links cannot cross file systems. If a file has multiple hard links, the file is deleted only when the last link pointing to the inode is deleted and the link count goes to 0.

Soft links, or symlinks, merely point to another file or directory by name rather than by inode. Soft links can cross file system boundaries. Deleting a soft link does not delete the target file or directory, and deleting the target file or directory does not automatically remove any soft links.

Creating links

First let's look at how to create hard and soft links. Later in this article, we'll look at ways to identify and use the links we create here.

Hard links

You use the `ln` command to create additional hard links to an existing file (but not to a directory, even though the system sets up . and .. as hard links).

Listing 1 shows how to create a directory containing two files and a subdirectory with two hard links to file1, one in the same directory and one in the subdirectory. We have added a word to file1, and then another word to file3 and displayed the contents of the link in the subdirectory to show that all do indeed point to the same data.

Listing 1. Creating hard links

```
ian@attic4:~$ mkdir -p lpi104-6/subdir
ian@attic4:~$ touch lpi104-6/file1
ian@attic4:~$ touch lpi104-6/file2
ian@attic4:~$ ln lpi104-6/file1 lpi104-6/file3
ian@attic4:~$ ln lpi104-6/file1 lpi104-6/subdir/file3sub
ian@attic4:~$ echo "something" > lpi104-6/file1
ian@attic4:~$ echo "else" >> lpi104-6/file3
ian@attic4:~$ cat lpi104-6/subdir/file3sub
something
else
```

You will get an error if you attempt to create hard links that cross file systems or that are for directories. Listing 2 shows that my home and research directories are on different file systems and that an attempt to create a hard link across these fails, as does an attempt to create a hard link to the lpi104-6 directory.

Listing 2. Failures with hard link creation

```
ian@attic4:~$ df . research
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/sda7             71205436   9355052  58233352  14% /
/dev/sdb3            137856204  27688208 103165264  22% /home/ian/ian-research
ian@attic4:~$ ln lpi104-6/file1 research/lpi104-6/file3
ln: creating hard link `research/lpi104-6/file3' => `lpi104-6/file1': No such file or dir
ectory
ian@attic4:~$ ln lpi104-6 lpidir104-6
ln: `lpi104-6': hard link not allowed for directory
```

Soft links

You use the `ln` command with the `-s` option to create soft links. Soft links use file or directory names, which may be relative or absolute. If you are using relative names, you will usually want the current working directory to be the directory where you are creating the link; otherwise, the link you create will be relative to another point in the file system. Listing 3 shows you two ways to create a soft link for the file1 that we just created, and also how to create soft links instead of the two hard links that failed in Listing 2.

Listing 3. Creating soft links

```
ian@attic4:~$ # Create symlink using absolute paths
ian@attic4:~$ ln -s ~/lpi104-6/file1 ~/lpi104-6/file4
ian@attic4:~$ # Create symlink using relative paths
ian@attic4:~$ cd lpi104-6/
ian@attic4:~/lpi104-6$ ln -s file1 file5
ian@attic4:~/lpi104-6$ cd ..
ian@attic4:~$ # Create symlink across file systems
ian@attic4:~$ mkdir ~ian/research/lpi104-6
ian@attic4:~$ ln -s ~/lpi104-6/file1 ~ian/research/lpi104-6/file4
ian@attic4:~$ # Create symlink for directory
ian@attic4:~$ ln -s lpi104-6 lpidir104-6
```

As before, you can use any of the links or the target file name to reference the file or directory. Listing 4 shows some examples.

Listing 4. Using soft links

```
ian@attic4:~$ echo "another line" >> ~ian/research/lpi104-6/file
ian@attic4:~$ # cat a symlink
ian@attic4:~$ cat lpi104-6/file5
something
else
another line
ian@attic4:~$ # cat a hard link
ian@attic4:~$ cat lpi104-6/file1
something
else
another line
ian@attic4:~$ # display directory contents using symlink
ian@attic4:~$ ls lpidir104-6
file1  file2  file3  file4  file5  subdir
```

While we're creating links, let's create a link using relative paths when our working directory is **not** the directory where we want the link. We'll look at what this does in the next section.

Listing 5. Creating a bad soft link

```
ian@attic4:~$ ln -s lpi104-6/file1 lpi104-6/file6
```

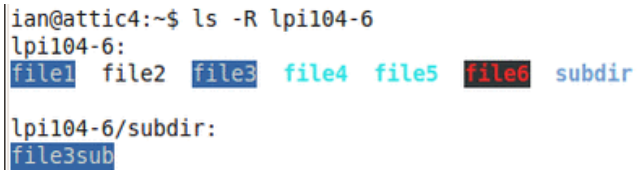
Identifying links

In the previous section, you saw how to create links, but not how to distinguish the links you created. Let's look at that now.

Finding information

On many systems today, the `ls` command is aliased to `ls --color=auto`, which prints different types of file system objects in different colors. The colors are configurable. If you use this option, hard links might show up with a dark blue background, and symlinks with cyan text, as illustrated in Figure 1.

Figure 1. Using the --colors option of ls to identify links



While color might be convenient for sighted people who can distinguish them, they are not much use to others, and certainly not much use to shell scripts or programs. Without color, you need more information, such as that provided by a long listing using `ls -l`. In Listing 6 we explicitly disable color output, but you could also explicitly call the `/bin/ls` command.

Listing 6. Identifying links

```
ian@attic4:~$ ls --color=none -lR lpi104-6
lpi104-6:
total 12
-rw-r--r-- 3 ian ian   28 2010-05-27 17:17 file1
-rw-r--r-- 1 ian ian    0 2010-05-26 14:11 file2
-rw-r--r-- 3 ian ian   28 2010-05-27 17:17 file3
lrwxrwxrwx 1 ian ian   24 2010-05-27 17:15 file4 -> /home/ian/lpi104-6/file1
lrwxrwxrwx 1 ian ian    5 2010-05-27 17:15 file5 -> file1
lrwxrwxrwx 1 ian ian   14 2010-05-27 17:37 file6 -> lpi104-6/file1
drwxr-xr-x 2 ian ian 4096 2010-05-26 14:11 subdir

lpi104-6/subdir:
total 4
-rw-r--r-- 3 ian ian 28 2010-05-27 17:17 file3sub
ian@attic4:~$ /bin/ls -l ~ian/research/lpi104-6/file4
lrwxrwxrwx 1 ian ian 24 2010-05-25 11:51 /home/ian/research/lpi104-6/file4 -> /home/ian/
lpi104-6/file1
ian@attic4:~$ /bin/ls -l lpidir104-6
lrwxrwxrwx 1 ian ian 8 2010-05-27 17:16 lpidir104-6 -> lpi104-6
```

The second column of output is a *link count* showing the number of hard links to this file, so we know that file1, file3, and file3sub all have multiple hard links pointing to the object they represent, although we do not yet have enough information to know they all represent the same object. If you delete a file that has a link count greater than 1, the link count in the inode is reduced by 1, but the file is not deleted until the count goes to 0. All other hard links to the same file will show a link count that is now reduced by 1.

In the first column of output, you see the first character is an 'l' (lower-case L) for symbolic links. You also see the *target* of the link displayed after the -> characters. For example file4 -> /home/ian/lpi104-6/file1. Another tipoff is that the size is the number of characters in the link target's name. Note that the link counts in the directory listing are not updated for symbolic links. Deleting the link does not affect the target file. Symlinks do not prevent a file from being deleted; if the target file is moved or deleted, then the symlink will be broken. For this reason, many systems use colors in directory listings, often pale blue for a good link and red for a broken one.

You can use the `-i` option of the `ls` command to display inode numbers for file and directory entries. Listing 7 shows both short and long output for our lpi104-6 directory.

Listing 7. Displaying inode information

```
ian@attic4:~$ ls -i lpi104-6
1680103 file1  1680103 file3  1680107 file5  1680101 subdir
1680104 file2  1680108 file4  1680110 file6
ian@attic4:~$ ls -il lpi104-6
total 12
1680103 -rw-r--r-- 3 ian ian   28 2010-05-27 17:17 file1
1680104 -rw-r--r-- 1 ian ian    0 2010-05-26 14:11 file2
1680103 -rw-r--r-- 3 ian ian   28 2010-05-27 17:17 file3
1680108 lrwxrwxrwx 1 ian ian   24 2010-05-27 17:15 file4 -> /home/ian/lpi104-6/file1
1680107 lrwxrwxrwx 1 ian ian    5 2010-05-27 17:15 file5 -> file1
1680110 lrwxrwxrwx 1 ian ian   14 2010-05-27 17:37 file6 -> lpi104-6/file1
1680101 drwxr-xr-x 2 ian ian 4096 2010-05-26 14:11 subdir
```

You can also use the `find` command to search for symbolic links using the `-type l` find expression as shown in Listing 8.

Listing 8. Using find to locate symlinks

```
ian@attic4:~$ find lpi104-6 research/lpi104-6 -type l
lpi104-6/file6
lpi104-6/file5
lpi104-6/file4
research/lpi104-6/file4
```

Broken symlinks

In Listing 5, we claimed to create a bad soft link. This is one example of a broken symlink. Since hard links always point to an inode that represents a file, they are always valid. However, symlinks can be broken for many reasons, including:

- Either the original file or the target of the link did not exist when the link was created (as in Listing 5).
- The target of a link is deleted or renamed.
- Some element in the path to the target is removed or renamed.

None of these conditions raises an error, so you need to think carefully about what might happen to your symlinks as you create them. In particular, your choice of absolute or relative paths is likely to be influenced by what you expect to happen to the objects you are linking over the life of the link.

If you are using colored output, broken symlinks are likely to show up as red text on a black background, as is the case for file6 in Figure 1. Otherwise, you will need to use either the `-H` or `-L` options of `ls` to dereference the link and give you information about the target. The `-H` option dereferences links on the command line and the `-L` option dereferences those plus links that are part of the display. Listing 9 illustrates the difference in the output from these two options.

Listing 9. Dereferencing links with ls -H and ls -L

```
ian@attic4:~$ /bin/ls -lH lpidir104-6
total 12
-rw-r--r-- 3 ian ian   28 2010-05-27 17:17 file1
-rw-r--r-- 1 ian ian    0 2010-05-26 14:11 file2
-rw-r--r-- 3 ian ian   28 2010-05-27 17:17 file3
lrwxrwxrwx 1 ian ian   24 2010-05-27 17:15 file4 -> /home/ian/lpi104-6/file1
lrwxrwxrwx 1 ian ian    5 2010-05-27 17:15 file5 -> file1
lrwxrwxrwx 1 ian ian   14 2010-05-27 17:37 file6 -> lpi104-6/file1
drwxr-xr-x 2 ian ian 4096 2010-05-26 14:11 subdir
ian@attic4:~$ /bin/ls -lL lpidir104-6
/bin/ls: cannot access lpidir104-6/file6: No such file or directory
total 20
-rw-r--r-- 3 ian ian   28 2010-05-27 17:17 file1
-rw-r--r-- 1 ian ian    0 2010-05-26 14:11 file2
-rw-r--r-- 3 ian ian   28 2010-05-27 17:17 file3
-rw-r--r-- 3 ian ian   28 2010-05-27 17:17 file4
-rw-r--r-- 3 ian ian   28 2010-05-27 17:17 file5
l????????? ? ?   ?      ?                ? file6
drwxr-xr-x 2 ian ian 4096 2010-05-26 14:11 subdir
```

Note the error message indicating that file6 does not exist and also the output for it with all the '?' characters, again indicating that the file is not found.

One final point on our broken symbolic link. Attempts to read the file will fail as it does not exist. However, attempts to write it will work if you have the appropriate permission on the target file, as shown in Listing 10. Note that we need to create the lpi104-6/lpi104-6 before we can write the file.

Listing 10. Reading from and writing to a broken symlink

```
ian@attic4:~$ cat lpi104-6/file6
cat: lpi104-6/file6: No such file or directory
ian@attic4:~$ echo "Testing file6" > lpi104-6/file6
bash: lpi104-6/file6: No such file or directory
ian@attic4:~$ mkdir lpi104-6/lpi104-6
ian@attic4:~$ cat lpi104-6/file6
cat: lpi104-6/file6: No such file or directory
ian@attic4:~$ echo "Testing file6" > lpi104-6/file6
ian@attic4:~$ cat lpi104-6/file6
Testing file6
ian@attic4:~$ ls lpi104-6/lpi104-6
file1
```

Who links to me?

To find which files are hard links to a particular inode, you can use the `find` command and the `-samefile` option with a filename or the `-inum` option with an inode number, as shown in Listing 11.

Listing 11. Finding hard links to the same file

```
ian@attic4:~$ find lpi104-6 -samefile lpi104-6/file1
lpi104-6/subdir/file3sub
lpi104-6/file3
lpi104-6/file1
ian@attic4:~$ ls -i lpi104-6/file1
1680103 lpi104-6/file1
ian@attic4:~$ find lpi104-6 -inum 1680103
lpi104-6/subdir/file3sub
lpi104-6/file3
lpi104-6/file1
```

To find which files link symbolically to a particular file, you can use the `find` command and the `-lname` option with a filename, as illustrated in Listing 12. Links may use a relative or absolute path, so you probably want a leading asterisk in the name to find all matches.

Listing 12. Finding symbolic links to a file or directory

```
ian@attic4:~$ find lpi104-6 research/lpi104-6 -lname "*file1"
lpi104-6/file6
lpi104-6/file5
lpi104-6/file4
research/lpi104-6/file4
```

Copying versus linking

Depending on what you want to accomplish, sometimes you will use links and sometimes it may be better to make a copy of a file. The major difference is that links provide multiple names for a single file, while a copy creates two sets of identical data under two different names. You would certainly use copies for backup and also for test purposes where you want to try out a new program without putting your operational data at risk. You use links when you need an alias for a

file (or directory), possibly to provide a more convenient or shorter path. In the next section, we'll look at some other uses for links.

As you have seen, when you update a file, all the links to it see the update, which is not the case if you copy a file. You have also seen that symbolic links can be broken but that subsequent write operations may create a new file. Use links with care.

Links and system administration

Links, especially symbolic links, are frequently used in Linux system administration. Commands are often aliased so the user does not have to know a version number for the current command, but can access other versions by longer names if necessary. As shown in Listing 13, the gcc command is a symlink and there are three different names for it on my system.

Listing 13. Aliasing commands to a particular version

```
ian@attic4:~$ which gcc
/usr/bin/gcc
ian@attic4:~$ ls -l /usr/bin/gcc
lrwxrwxrwx 1 root root 7 2009-12-28 23:17 /usr/bin/gcc -> gcc-4.4
ian@attic4:~$ find /usr/bin -lname "*gcc-4.4"
/usr/bin/x86_64-linux-gnu-gcc-4.4
/usr/bin/gcc
/usr/bin/x86_64-linux-gnu-gcc
```

Other uses come into play when multiple command names use the same underlying code, such as the various commands for stopping and for restarting a system. Sometimes, a new command name, such as genisofs, will replace an older command name, but the old name (mkisofs) is kept as a link to the new command. And the alternatives facility uses links extensively so you can choose which among several alternatives to use for a command such as `java`. Listing 14 shows some examples.

Listing 14. Command alias examples

```
ian@attic4:~$ find /sbin -lname "initctl"
/sbin/restart
/sbin/start
/sbin/stop
/sbin/status
/sbin/reload
ian@attic4:~$ ls -l $(which mkisofs)
lrwxrwxrwx 1 root root 11 2009-12-28 23:17 /usr/bin/mkisofs -> genisoimage
ian@attic4:~$ ls -l $(which java)
lrwxrwxrwx 1 root root 22 2010-01-17 15:16 /usr/bin/java -> /etc/alternatives/java
```

Library names are also managed extensively using symlinks, either to allow programs to link to a general name while getting the current version, or to manage systems such as 64-bit systems that are capable of running 32-bit programs. Some examples are shown in Listing 15. Notice that some use absolute paths, while some use relative paths.

Listing 15. Library links

```
ian@attic4:~$ ls -l /usr/lib/libm.so
lrwxrwxrwx 1 root root 14 2010-05-27 11:23 /usr/lib/libm.so -> /lib/libm.so.6
ian@attic4:~$ find  /usr/lib/ -lname "*libstdc++*"
/usr/lib/gcc/x86_64-linux-gnu/4.4/libstdc++.so
/usr/lib/libstdc++.so.6
ian@attic4:~$ ls -l /usr/lib/gcc/x86_64-linux-gnu/4.4/libstdc++.so
lrwxrwxrwx 1 root root 23 2010-01-19 08:49 /usr/lib/gcc/x86_64-linux-gnu/4.4/libstdc++.s
o -> ../../../libstdc++.so.6
```

For more information about linking, consult the man pages for `ln` and the other commands you have seen in this article.

Resources

Learn

- Develop and deploy your next app on the IBM Bluemix cloud platform.
- Use the developerWorks roadmap for LPIC-1 to find the developerWorks articles to help you study for LPIC-1 certification based on the April 2009 objectives.
- At the LPIC Program site, find detailed objectives, task lists, and sample questions for the three levels of the Linux Professional Institute's Linux system administration certification. In particular, see their April 2009 objectives for LPI exam 101 and LPI exam 102. Always refer to the LPIC Program site for the latest objectives.
- Review the entire LPI exam prep series on developerWorks to learn Linux fundamentals and prepare for system administrator certification based on earlier LPI exam objectives prior to April 2009.
- For additional examples of how to use symbolic links in Linux system administration, see:
  - "Learn Linux, 101: Manage shared libraries" (developerWorks, March 2010)
  - "Dissecting shared libraries" (developerWorks, January 2005)
  - "Boot Linux faster" (developerWorks, September 2003)
- In the developerWorks Linux zone, find hundreds of how-to articles and tutorials, as well as downloads, discussion forums, and a wealth other resources for Linux developers and administrators.
- Stay current with developerWorks technical events and webcasts focused on a variety of IBM products and IT industry topics.
- Attend a free developerWorks Live! briefing to get up-to-speed quickly on IBM products and tools as well as IT industry trends.
- Watch developerWorks on-demand demos ranging from product installation and setup demos for beginners, to advanced functionality for experienced developers.
- Follow developerWorks on Twitter, or subscribe to a feed of Linux tweets on developerWorks.

Get products and technologies

- Evaluate IBM products in the way that suits you best: Download a product trial, try a product online, use a product in a cloud environment, or spend a few hours in the SOA Sandbox learning how to implement Service Oriented Architecture efficiently.

Discuss

- Participate in the discussion forum for this content.
- Get involved in the My developerWorks community. Connect with other developerWorks users while exploring the developer-driven blogs, forums, groups, and wikis.

About the author

Ian Shields

Ian Shields works on a multitude of Linux projects for the developerWorks Linux zone. He is a Senior Programmer at IBM at the Research Triangle Park, NC. He joined IBM in Canberra, Australia, as a Systems Engineer in 1973, and has since worked on communications systems and pervasive computing in Montreal, Canada, and RTP, NC. He has several patents and has published several papers. His undergraduate degree is in pure mathematics and philosophy from the Australian National University. He has an M.S. and Ph.D. in computer science from North Carolina State University. Learn more about Ian in Ian's profile on developerWorks Community.