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

Use multiple names for the same file

Ian Shields

January 27, 2016
(First published June 01, 2010)

Learn how to create and manage hard and symbolic links to files on your Linux® system. You can use the material in this tutorial 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 tutorial, 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 tutorial helps you prepare for Objective 104.6 in Topic 104 of the Linux Server Professional (LPIC-1) exam 101. The objective has a weight of 2.

Introducing links

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.

About this series

This series of tutorials helps you learn Linux system administration tasks. You can also use the material in these tutorials to prepare for the Linux Professional Institute's LPIC-1: Linux Server Professional Certification exams.

See "*Learn Linux, 101*: A roadmap for LPIC-1" for a description of and link to each tutorial in this series. The roadmap is in progress and reflects the version 4.0 objectives of the LPIC-1 exams as updated April 15th, 2015. As tutorials are completed, they will be added to the roadmap.

A *link* is simply an additional directory entry for a file or directory, allowing two or more names for the same thing.

Prerequisites

To get the most from the tutorials 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 tutorial. Unless otherwise noted, the examples in this tutorial use CentOS 6 with a 2.6.32-504 kernel.

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.

Creating links

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 filesystem 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.

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

Hard links

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 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@atticf22 ~]$ mkdir -p lpi104-6/subdir
[ian@atticf22 ~]$ touch lpi104-6/file1
[ian@atticf22 ~]$ touch lpi104-6/file2
[ian@atticf22 ~]$ ln lpi104-6/file1 lpi104-6/file3
[ian@atticf22 ~]$ ln lpi104-6/file1 lpi104-6/subdir/file3sub
[ian@atticf22 ~]$ echo "something" > lpi104-6/file1
[ian@atticf22 ~]$ echo "else" >> lpi104-6/file3
[ian@atticf22 ~]$ cat lpi104-6/subdir/file3sub
something
else
```

You will get an error if you attempt to create hard links that cross filesystems or that are for directories. Listing 2 shows that my home and research directories are on different filesystems 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@atticf22 ~]$ df . research
Filesystem     1K-blocks     Used Available Use% Mounted on
/dev/sda5       71168700 31642752  35887712  47% /
/dev/sdb3       60326992 30677592  26578276  54% /home/ian/research
[ian@atticf22 ~]$ mkdir -p research/lpi104-6/
[ian@atticf22 ~]$ ln lpi104-6/file1 research/lpi104-6/file3
ln: failed to create hard link ‘research/lpi104-6/file3’ => ‘lpi104-6/file1’: Invalid cross-device link
[ian@atticf22 ~]$ 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@atticf22 ~]$ # Create symlink using absolute paths
[ian@atticf22 ~]$ ln -s ~/lpi104-6/file1 ~/lpi104-6/file4
[ian@atticf22 ~]$ # Create symlink using relative paths
[ian@atticf22 ~]$ cd lpi104-6/
[ian@atticf22 lpi104-6]$ ln -s file1 file5
[ian@atticf22 lpi104-6]$ cd ..
[ian@atticf22 ~]$ # Create symlink across file systems
[ian@atticf22 ~]$ mkdir -p ~ian/research/lpi104-6
[ian@atticf22 ~]$ ln -s ~/lpi104-6/file1 ~ian/research/lpi104-6/file4
[ian@atticf22 ~]$ # Create symlink for directory
[ian@atticf22 ~]$ 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@atticf22 ~]$ echo "another line" >> ~ian/research/lpi104-6/file
[ian@atticf22 ~]$ # cat a symlink
[ian@atticf22 ~]$ cat lpi104-6/file5
something
else
[ian@atticf22 ~]$ # cat a hard link
[ian@atticf22 ~]$ cat lpi104-6/file1
something
else
[ian@atticf22 ~]$ # display directory contents using symlink
[ian@atticf22 ~]$ 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@atticf22 ~]$ ln -s lpi104-6/file1 lpi104-6/file6
```

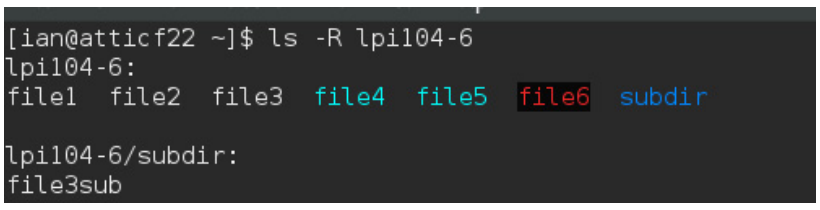
Identifying links

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, the `ls` command is aliased to `ls --color=auto`, which prints different types of filesystem objects in different colors. If you use this option, symlinks might show up with cyan text, as illustrated in Figure 1.

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

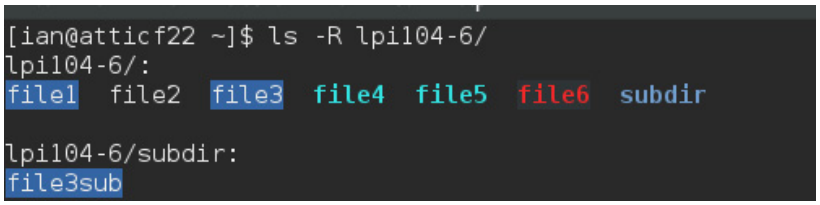


On older configurations hard links might show up with a dark blue background. The colors are configurable using the `dircolors` program. If you customize your terminal appearance, you will probably want to change colors for some of the output of `ls`. Listing 6 shows one way of getting a blue background for hard links on our Fedora 22 system, and Figure 2 shows the result. Use the man pages to further understand the example.

Listing 6. Using dircolors to set a blue background for hard links

```
[ian@atticf22 ~]$ # Save a copy of dircolors defaults
[ian@atticf22 ~]$ dircolors -p > dircolors-defaults
[ian@atticf22 ~]$ grep MULTI dircolors-defaults
MULTIHARDLINK 00 # regular file with more than one link
[ian@atticf22 ~]$ # Change MULTIHARDLINK to blue background
[ian@atticf22 ~]$ sed -e'/MULTI/s/00/00;44/' dircolors-defaults > dircolors-new
[ian@atticf22 ~]$ grep MULTI dircolors-new
MULTIHARDLINK 00;44 # regular file with more than one link
[ian@atticf22 ~]$ # Set the new colors for the current terminal session
[ian@atticf22 ~]$ eval $(dircolors dircolors-new )
```

Figure 2. Identifying hard links with a blue background



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 7, we explicitly disable color output for the first example, but you could also explicitly call the `/bin/ls` command as we have done for the other two examples.

Listing 7. Identifying links

```
[ian@atticf22 ~]$ ls --color=none -lR lpi104-6
lpi104-6:
total 12
-rw-rw-r--. 3 ian ian   15 Aug  9 14:19 file1
-rw-rw-r--. 1 ian ian    0 Aug  9 14:19 file2
-rw-rw-r--. 3 ian ian   15 Aug  9 14:19 file3
lrwxrwxrwx. 1 ian ian   24 Aug  9 14:26 file4 -> /home/ian/lpi104-6/file1
lrwxrwxrwx. 1 ian ian    5 Aug  9 14:26 file5 -> file1
lrwxrwxrwx. 1 ian ian   14 Aug  9 14:34 file6 -> lpi104-6/file1
drwxrwxr-x. 2 ian ian 4096 Aug  9 14:19 subdir

lpi104-6/subdir:
total 4
-rw-rw-r--. 3 ian ian 15 Aug  9 14:19 file3sub
[ian@atticf22 ~]$ /bin/ls -l ~ian/research/lpi104-6/file4
lrwxrwxrwx. 1 ian ian 24 Aug  9 14:27 /home/ian/research/lpi104-6/file4 -> /home/ian/lpi104-6/file1
[ian@atticf22 ~]$ /bin/ls -l lpidir104-6
lrwxrwxrwx. 1 ian ian 8 Aug  9 14:27 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. We still don't 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 tip off 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 8 shows both short and long output for our lpi104-6 directory.

Listing 8. Displaying inode information

```
[ian@atticf22 ~]$ ls -i lpi104-6
1988884 file1  1988884 file3  1988892 file5  1988605 subdir
1988886 file2  1988885 file4  1988891 file6
[ian@atticf22 ~]$ ls -il lpi104-6
total 12
1988884 -rw-rw-r--. 3 ian ian   15 Aug  9 14:19 file1
1988886 -rw-rw-r--. 1 ian ian    0 Aug  9 14:19 file2
1988884 -rw-rw-r--. 3 ian ian   15 Aug  9 14:19 file3
1988885 lrwxrwxrwx. 1 ian ian   24 Aug  9 14:26 file4 -> /home/ian/lpi104-6/file1
1988892 lrwxrwxrwx. 1 ian ian    5 Aug  9 14:26 file5 -> file1
1988891 lrwxrwxrwx. 1 ian ian   14 Aug  9 14:34 file6 -> lpi104-6/file1
1988605 drwxrwxr-x. 2 ian ian 4096 Aug  9 14:19 subdir
```

You can also use the `find` command to search for symbolic links using the `-type l` find expression as shown in Listing 9.

Listing 9. Using find to locate symlinks

```
[ian@atticf22 ~]$ find lpi104-6 research/lpi104-6 -type l
lpi104-6/file4
lpi104-6/file6
lpi104-6/file5
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 10 illustrates the difference in the output from these two options.

Listing 10. Dereferencing links with ls -H and ls -L

```
[ian@atticf22 ~]$ /bin/ls -lH lpidir104-6
total 12
-rw-rw-r--. 3 ian ian   15 Aug  9 14:19 file1
-rw-rw-r--. 1 ian ian    0 Aug  9 14:19 file2
-rw-rw-r--. 3 ian ian   15 Aug  9 14:19 file3
lrwxrwxrwx. 1 ian ian   24 Aug  9 14:26 file4 -> /home/ian/lpi104-6/file1
lrwxrwxrwx. 1 ian ian    5 Aug  9 14:26 file5 -> file1
lrwxrwxrwx. 1 ian ian   14 Aug  9 14:34 file6 -> lpi104-6/file1
drwxrwxr-x. 2 ian ian 4096 Aug  9 14:19 subdir
[ian@atticf22 ~]$ /bin/ls -lL lpidir104-6
/bin/ls: cannot access lpidir104-6/file6: No such file or directory
total 20
-rw-rw-r--. 3 ian ian   15 Aug  9 14:19 file1
-rw-rw-r--. 1 ian ian    0 Aug  9 14:19 file2
-rw-rw-r--. 3 ian ian   15 Aug  9 14:19 file3
-rw-rw-r--. 3 ian ian   15 Aug  9 14:19 file4
-rw-rw-r--. 3 ian ian   15 Aug  9 14:19 file5
l????????? ? ?   ?      ?            ? file6
drwxrwxr-x. 2 ian ian 4096 Aug  9 14:19 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 11. Note that we need to create the lpi104-6/lpi104-6 before we can write the file.

Listing 11. Reading from and writing to a broken symlink

```
[ian@atticf22 ~]$ cat lpi104-6/file6
cat: lpi104-6/file6: No such file or directory
[ian@atticf22 ~]$ echo "Testing file6" > lpi104-6/file6
bash: lpi104-6/file6: No such file or directory
[ian@atticf22 ~]$ mkdir lpi104-6/lpi104-6
[ian@atticf22 ~]$ cat lpi104-6/file6
cat: lpi104-6/file6: No such file or directory
[ian@atticf22 ~]$ echo "Testing file6" > lpi104-6/file6
[ian@atticf22 ~]$ cat lpi104-6/file6
Testing file6
[ian@atticf22 ~]$ ls lpi104-6/lpi104-6
file1
[ian@atticf22 ~]$ ls -l lpi104-6/file6
lrwxrwxrwx. 1 ian ian 14 Aug  9 14:34 lpi104-6/file6 -> 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 file name or the `-inum` option with an inode number, as shown in Listing 12.

Listing 12. Finding hard links to the same file

```
[ian@atticf22 ~]$ find lpi104-6 -samefile lpi104-6/file1
lpi104-6/file1
lpi104-6/file3
lpi104-6/subdir/file3sub
[ian@atticf22 ~]$ ls -i lpi104-6/file1
1988884 lpi104-6/file1
[ian@atticf22 ~]$ find lpi104-6 -inum 1988884
lpi104-6/file1
lpi104-6/file3
lpi104-6/subdir/file3sub
```

To find which files link symbolically to a particular file, you can use the `find` command and the `-lname` option with a file name, as Listing 13 illustrates. Links may use a relative or absolute path, so you probably want a leading asterisk in the name to find all matches.

Listing 13. Finding symbolic links to a file or directory

```
[ian@atticf22 ~]$ find lpi104-6 research/lpi104-6 -lname "*file1"
lpi104-6/file4
lpi104-6/file6
lpi104-6/file5
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 14, the `python` command is a symlink to `python2`, `python2` is itself a symlink to the 2.7 version `python2.7`.

Listing 14. Aliasing commands to a particular version

```
[ian@atticf22 ~]$ which python
/usr/bin/python
[ian@atticf22 ~]$ ls -l /usr/bin/python
lrwxrwxrwx. 1 root root 7 May 27 14:12 /usr/bin/python -> python2
[ian@atticf22 ~]$ ls -l /usr/bin/python2
lrwxrwxrwx. 1 root root 9 May 27 14:12 /usr/bin/python2 -> python2.7
[ian@atticf22 ~]$ ls -l /usr/bin/python2.7
-rwxr-xr-x. 1 root root 7120 May 27 14:12 /usr/bin/python2.7
```

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 `genisoimage`, 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 15 shows some examples.

Listing 15. Command alias examples

```
[ian@atticf22 ~]$ which halt
/usr/sbin/halt
[ian@atticf22 ~]$ ls -l /usr/sbin/halt
lrwxrwxrwx. 1 root root 16 Jun  9 09:16 /usr/sbin/halt -> ../bin/systemctl
[ian@atticf22 ~]$ find /usr/sbin /usr/bin -lname "*/systemctl"
/usr/sbin/halt
/usr/sbin/telinit
/usr/sbin/shutdown
/usr/sbin/runlevel
/usr/sbin/poweroff
/usr/sbin/reboot
[ian@atticf22 ~]$ which mkisofs
/usr/bin/mkisofs
[ian@atticf22 ~]$ ls -l /usr/bin/mkisofs
lrwxrwxrwx. 1 root root 25 Jun 15 08:02 /usr/bin/mkisofs -> /etc/alternatives/mkisofs
[ian@atticf22 ~]$ alternatives --display mkisofs
mkisofs - status is auto.
 link currently points to /usr/bin/genisoimage
/usr/bin/genisoimage - priority 50
 slave mkisofs-mkhybrid: /usr/bin/genisoimage
 slave mkisofs-mkhybridman: /usr/share/man/man1/genisoimage.1.gz
 slave mkisofs-mkisofsman: /usr/share/man/man1/genisoimage.1.gz
Current `best' version is /usr/bin/genisoimage.
```

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. Listing 16 shows some examples. Notice that some use absolute paths, while some use relative paths.

Listing 16. Library links

```
[ian@atticf22 ~]$ ls -l /usr/lib*/libm.so*
lrwxrwxrwx. 1 root root 21 Feb 23 10:31 /usr/lib64/libm.so -> ../../lib64/libm.so.6
lrwxrwxrwx. 1 root root 12 Feb 23 10:33 /usr/lib64/libm.so.6 -> libm-2.21.so
lrwxrwxrwx. 1 root root 12 Feb 23 10:35 /usr/lib/libm.so.6 -> libm-2.21.so
[ian@atticf22 ~]$ find  /usr/lib/ /usr/lib64/ -lname "*libstdc++*"
/usr/lib/gcc/x86_64-redhat-linux/5.1.1/libstdc++.so
/usr/lib/gcc/x86_64-redhat-linux/5.1.1/32/libstdc++.so
/usr/lib/gcc/x86_64-redhat-linux/5.1.1/32/libstdc++.a
/usr/lib64/libstdc++.so.6
[ian@atticf22 ~]$ ls -l /usr/lib64/libstdc++.so.6
lrwxrwxrwx. 1 root root 19 Jun 18 06:52 /usr/lib64/libstdc++.so.6 -> libstdc++.so.6.0.21
[ian@atticf22 ~]$ ls -l /usr/lib64/libwbclient*
lrwxrwxrwx. 1 root root 19 Jul  1 10:37 /usr/lib64/libwbclient.so.0 -> libwbclient.so.0.12
lrwxrwxrwx. 1 root root 40 Jul  1 10:37 /usr/lib64/libwbclient.so.0.12 ->
/etc/alternatives/libwbclient.so.0.12-64
```

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

Related topics

- Use the developerWorks roadmap for LPIC-1 to find the developerWorks tutorials to help you study for LPIC-1 certification based on the LPI Version 4.0 April 2015 objectives.
- At the Linux Professional Institute website, find detailed objectives, task lists, and sample questions for the certifications. In particular, see:
  - The LPIC-1: Linux Server Professional Certification program details
  - LPIC-1 exam 101 objectives
  - LPIC-1 exam 102 objectives

  Always refer to the Linux Professional Institute website for the latest objectives.
- 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)