**Explaining Soft Link And Hard Link In Linux With Examples**

Written by [Sk](https://ostechnix.com/author/sk/) October 17, 2019 13641 Views

26 comments

***4***

**What is Soft Link And Hard Link In Linux?**

A **symbolic** or **soft link** is an actual link to the original file, whereas a **hard link** is a mirror copy of the original file. If you delete the original file, the soft link has no value, because it points to a non-existent file. But in the case of hard link, it is entirely opposite. Even if you delete the original file, the hard link will still has the data of the original file. Because hard link acts as a mirror copy of the original file.

In a nutshell, a soft link

* can cross the file system,
* allows you to link between directories,
* has different inode number and file permissions than original file,
* permissions will not be updated,
* has only the path of the original file, not the contents.

A hard Link

* can't cross the file system boundaries (i.e. A hardlink can only work on the same filesystem),
* can't link directories,
* has the same inode number and permissions of original file,
* permissions will be updated if we change the permissions of source file,
* has the actual contents of original file, so that you still can view the contents, even if the original file moved or removed.

Still don't get it? Well, allow me to show you some practical examples.

**Creating Soft Link or Symbolic Link**

Let us create an empty directory called **"test"**.

$ mkdir test

Change to the "test" directory:

$ cd test

Now, create a new file called **source.file** with some data as shown below.

$ echo "Welcome to OSTechNix" >source.file

Let us view the data of the source.file.

**$ cat source.file**

Welcome to OSTechNix

Well, the source.file has been created.

Now, create the a symbolic or soft link to the source.file.

To do so, run:

$ ln -s source.file softlink.file

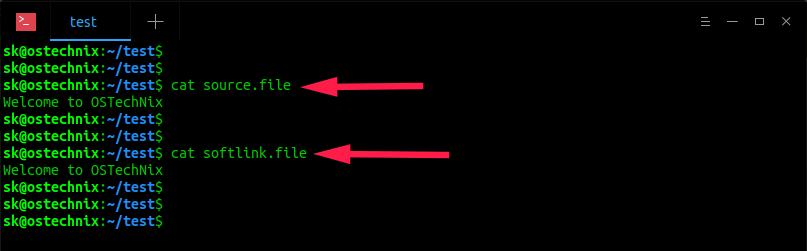
Let us compare the data of both source.file and softlink.file.

**$ cat source.file**

Welcome to OSTechNix

**$ cat softlink.file**

Welcome to OSTechNix

[](https://ostechnix.com/wp-content/uploads/2016/12/view-symlink-data.png)

As you see in the above output, softlink.file displays the same data as source.file.

Let us check the inodes and permissions of softlink.file and source.file.

$ ls -lia

**Sample output:**

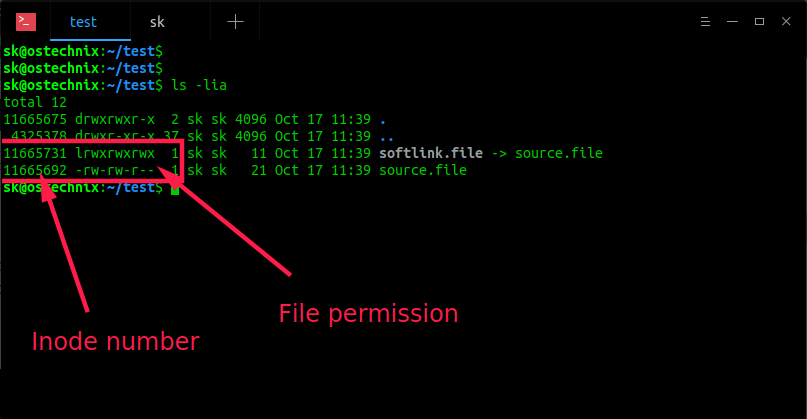
total 12

11665675 drwxrwxr-x 2 sk sk 4096 Oct 17 11:39 .

4325378 drwxr-xr-x 37 sk sk 4096 Oct 17 11:39 ..

**11665731 lrwxrwxrwx** 1 sk sk 11 Oct 17 11:39 **softlink.file** -> source.file

**11665692 -rw-rw-r--** 1 sk sk 21 Oct 17 11:39 **source.file**

[](https://ostechnix.com/wp-content/uploads/2016/12/check-the-inodes-and-permissions-of-symbolic-link.png)

As we see in the above screenshot, the **inode number (11665731**vs**11665692)** and **file permissions (lrwxrwxrwx**vs**-rw-r--r--)** are **different**, even though the softlink.file has same contents as source.file, . Hence, it is proved that soft link don't share the same inode number and permissions of original file.

Now, remove the original file (i.e source.file) and see what happens.

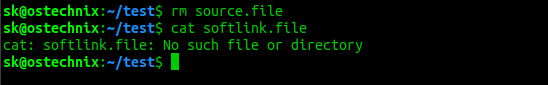
$ rm source.file

Check contents of the softlink.file using command:

$ cat softlink.file

**Sample output:**

cat: softlink.file: No such file or directory

[](https://ostechnix.com/wp-content/uploads/2016/12/check-symbolic-link-contents.png)

As you see above, there is no such file or directory called softlink.file after we removed the original file (i.e source.file). So, now we understand that soft link is just a link that points to the original file. The softlink is like a shortcut to a file. If you remove the file, the shortcut is useless.

As you already know, if you remove the soft link, the original file will still be available.

**Suggested read:**

* [**How To List Symlinks On Linux**](https://ostechnix.com/quick-tip-how-to-list-symlinks-on-linux/)
* [**How To Find Broken Symlinks And Delete Them On Linux**](https://ostechnix.com/how-to-find-broken-symlinks-and-delete-them-on-linux/)

**Creating Hard Link**

Create a file called **source.file** with some contents as shown below.

$ echo "Welcome to OSTechNix" >source.file

Let us verify the contents of the file.

**$ cat source.file**

Welcome to OSTechNix

source.file has been created now.

Now, let us create the hard link to the source.file as shown below.

$ ln source.file hardlink.file

[create hard link](https://ostechnix.com/wp-content/uploads/2016/12/create-hard-link.png)

Check the contents of hardlink.file.

**$ cat hardlink.file**

Welcome to OSTechNix

You see the hardlink.file displays the same data as source.file.

Let us check the inode and permissions of hardlink.file and source.file.

$ ls -lia

**Sample output:**

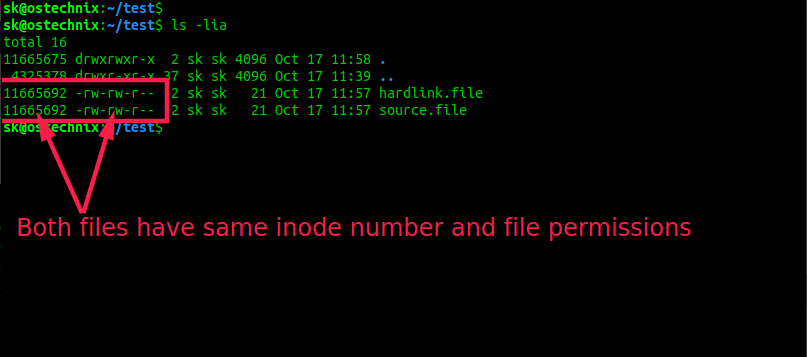
total 16

11665675 drwxrwxr-x 2 sk sk 4096 Oct 17 11:58 .

4325378 drwxr-xr-x 37 sk sk 4096 Oct 17 11:39 ..

**11665692 -rw-rw-r--** 2 sk sk 21 Oct 17 11:57 **hardlink.file**

**11665692 -rw-rw-r--** 2 sk sk 21 Oct 17 11:57 **source.file**

[](https://ostechnix.com/wp-content/uploads/2016/12/check-the-inodes-and-permissions-of-hard-link.png)

Now, we see that both hardlink.file and source.file have the same the **inodes number (11665692)** and **file permissions (-rw-r--r--)**. Hence, it is proved that hard link file shares the same inodes number and permissions of original file.

**Note:** If we change the permissions on source.file, the same permission will be applied to the hardlink.file as well.

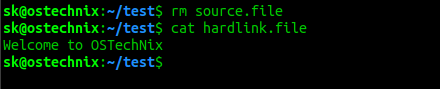
Now, remove the original file (i.e source.file) and see what happens.

$ rm source.file

Check contents of hardlink.file using command:

$ cat hardlink.file

**Sample output:**

[](https://ostechnix.com/wp-content/uploads/2016/12/check-hard-link-contents.png)

As you see above, even if I deleted the source file, I can view contents of the hardlink.file. Hence, it is proved that Hard link shares the same inode number, the permissions and data of the original file.

**So, what is the difference between Hard link and the normal copied file?**

You might be wondering why would we create a hard link while we can easily copy/paste the original file? Creating a hard link to a file is different than copying it.

If you copy a file, it will just duplicate the content. So if you modify the content of a one file (either original or hard link), it has no effect on the other one. However if you create a hard link to a file and change the content of either of the files, the change will be be seen on both.

Let us have a look at the source.file.

$ cat source.file

Welcome to OSTechNix

The source file has a single line that says - Welcome to OSTechNix.

Append a new line, for example "Welcome to Linux" in source.file or hardlink.file.

$ echo "Welcome to Linux" >>source.file

Now check contents of both files.

$ cat hardlink.file

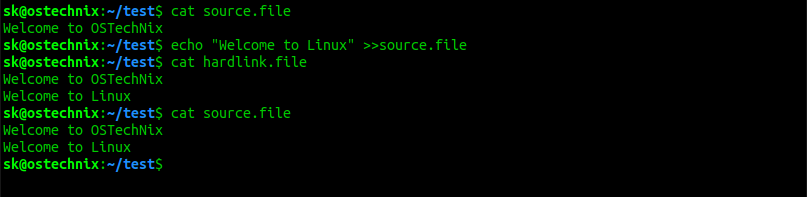
Welcome to OSTechNix

Welcome to Linux

$ cat source.file

Welcome to OSTechNix

Welcome to Linux

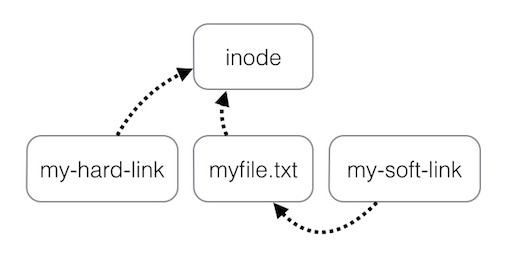
[](https://ostechnix.com/wp-content/uploads/2019/10/Update-hard-link.png)

See? The changes we just made on source.file are updated in both files. Meaning - both files (source and hard link) synchronizes. Whatever changes you do in any file will be reflected on other one. If you normally copy/paste the file, you will not see any new changes in other file.

For more details, check the man pages.

$ man ln

That's all for now. Hope you got a basic idea about symbolic or soft link and hard link.



Visualized path difference between hard link and symbolic link references

Hard links and symbolic links are two different methods to refer to a file in the hard drive. These methods are part of the filesystem that organizes what file is what and where. A hard link is essentially a synced carbon copy of a file that refers directly to the inode of a file. Symbolic links on the other hand refer directly to the file which refers to the inode, a shortcut. In order to understand how symbolic and hard links work, we will need to go over what are inodes.

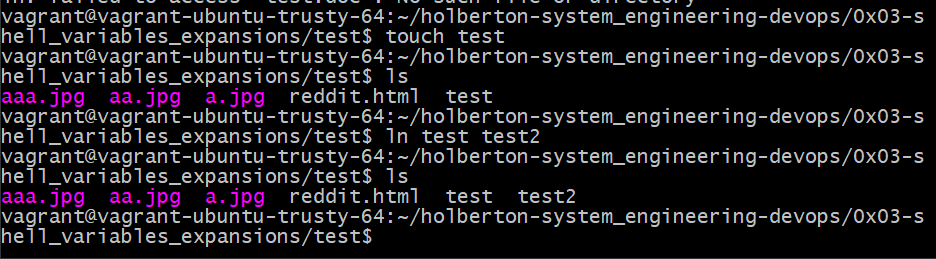
**What is an inode?**

The inode is a database that describes the file/directory attributes such as metadata and the physical location on the hard drive. They are essentially the numerical equivalent of a full address. With an inode, the OS can retrieve information about the file such as permission privileges and the physical location of the data on the hard drive to access the file. Should a file be moved from one folder to another, the file will be moved to a different location on the hard drive and its inode value will change with it automatically. This will be important for hard links. Speaking of hard links….

**What is a hard link?**

A hard link is a direct reference to a file via its inode. You can also only hardlink files and not directories. By using a hardlink, you can change the original file’s contents or location and the hardlink will still point to the original file because its inode is still pointing to that file. There is no referencing to the original file. In addition, hardlinks can only refer to files within the same volume otherwise symbolic links will be needed. To make a hard link of a file, you will require the ln command and refer to the source file before naming what the hard link will be named. Here is an example of how a hard link named test 2 will be made.

Image for post



First made the test file before making hard link test2

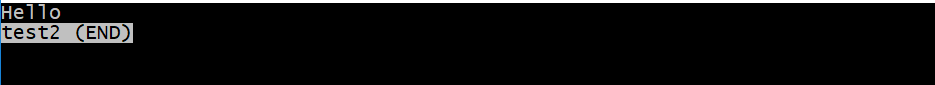
The file test should be completely empty and I will add “Hello” to it via the hard link.

Image for post

Image for post

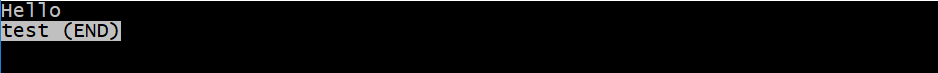
Typing in Hello into the file test via test2

Image for post



Opening test 2 with Hello in it

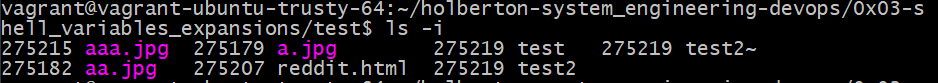
Image for post



The change from the hardlink is reflected in the original file

As seen in the photos above, I have changed the original file via the hard link by adding “Hello”. By opening the original file, the word “Hello” is already there. We can further make sure the files are referring to the same inode by using the ls -i command.

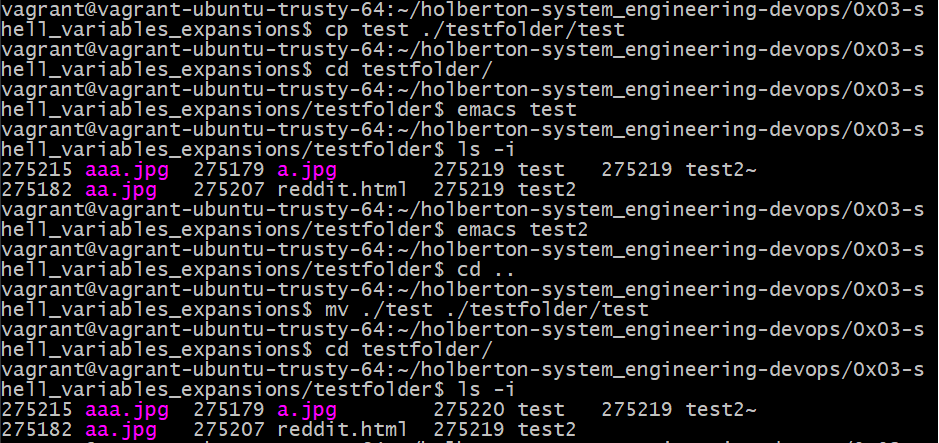
Image for post



test and test2 have the exact same inode.

Now what will happen if we copy over a similar file called test from a different folder into this folder? For this experiment we will change the folder name from ‘test’ to ‘test folder’.

Image for post

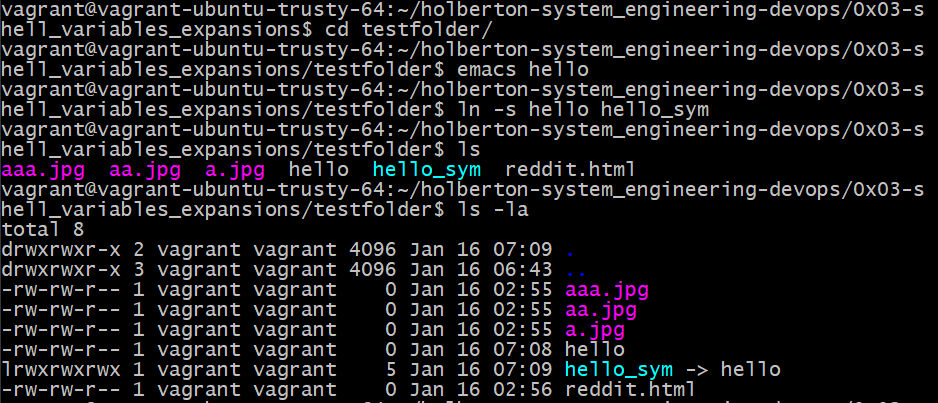


Here we can see that the cp command does not change the inode value of the original value but mv does. We have copied over a file from the parent directory into ‘testfolder’ and the inode value has not changed. It is only when you move over a file and replace the file that the inode value changes.

**What are symbolic links?**

Symbolic links are essentially shortcuts that reference to a file instead of its inode value. This method can be applied to directories and can reference across different hard disks/volumes. Since the symbolic link is referring to the original file and not its inode value, then replacing the original file into a different folder will break the symbolic link, or create a dangling link.

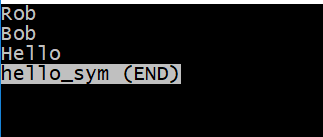
Image for post



Making a symbolic link. Note the link has an arrow pointing to the original file in its filename

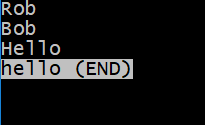
Since the symbolic link is a link that directs to the original file, changing the symboliclink should change the original file.

Image for post



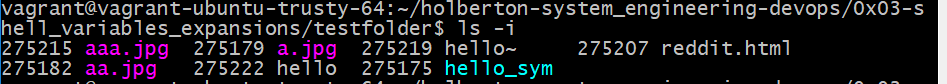
Changed the link hello\_sym

Image for post



The change is reflected in the original file

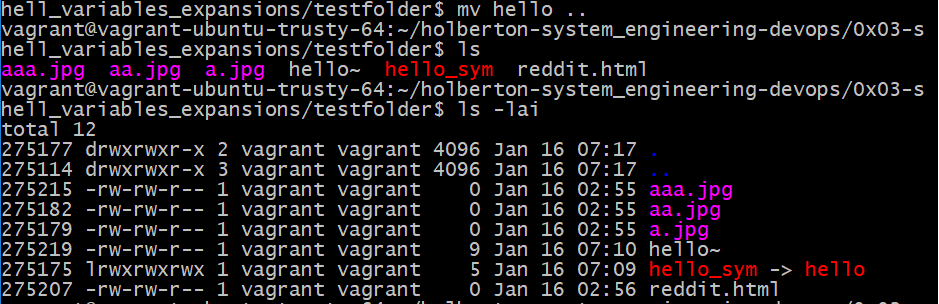
Image for post



A quick ls -i check shows they have different inodes

What will break a symbolic link is when the original file is moved to a different file or deleted.

Image for post



Moving the original file to a different folder broke the link

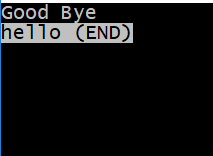
Image for post

Image for post

Opening the link shows the link is broken

So symbolic links can be seen as a static link to the last known location of the original file. The link should work even if you replace the original file with a different file with the same name.

Image for post



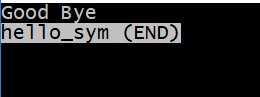
Made a new file hello with new contents

Image for post

Image for post

The link is now working again

Image for post



Contents of the link shows the contents of the new file

Hello there again. In this introduction to hard links and symbolic links, we will first try to learn and define what each one means. Then we will discuss what the differences between the two are.

So what is a hard link and what exactly does it do in the shell?

**Hard Link Definition:**

A *hard link* is merely an additional name for an existing [file](http://www.linfo.org/file.html) on [Linux](http://www.linfo.org/linuxdef.html) or other [Unix-like](http://www.linfo.org/unix-like.html) [operating systems](http://www.linfo.org/operating_system.html).

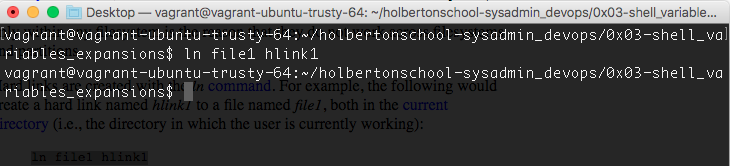
Any number of hard links, and thus any number of names, can be created for any file. Hard links can also be created to other hard links. However, they cannot be created for [directories](http://www.linfo.org/directory.html), and they cannot cross [filesystem](http://www.linfo.org/filesystem.html) boundaries or span across [partitions](http://www.linfo.org/partition.html).

Perhaps the most useful application for hard links is to allow files, [programs](http://www.linfo.org/program.html), and scripts (i.e. short programs) to be easily accessed in a different directory from the original file or [executable file](http://www.linfo.org/executable.html) (i.e., the ready-to-run version of a program). Typing the name of the hard link will cause the program or script to be executed in the same way as using its original name. [(http://www.linfo.org/hard\_link.html](http://www.linfo.org/hard_link.html))

Want to read this story later? Save it in [Journal](https://usejournal.com/?utm_source=medium.com&utm_medium=blog&utm_campaign=noteworthy&utm_content=eid7).

So what does this definition really mean? Well, you can create a hard link to an existing file by using the command ln file\_name hardlink. I have provided an example below of creating a hard link in action. In the example below I created a hardlink aka a shortcut to the file named file1 with the hardlink named hlink1.

Image for post

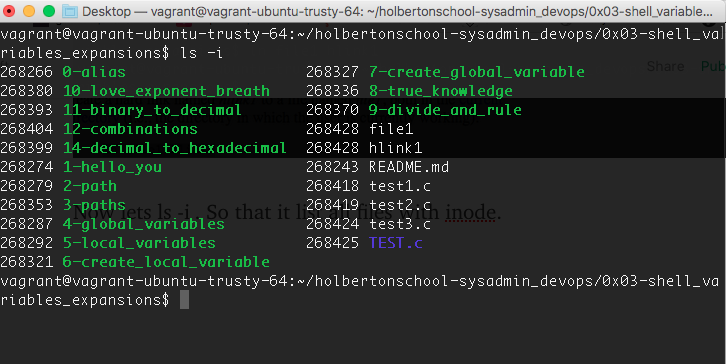


Now lets ls -i. So that it lists all files with inode.

INODE Definition: The **inode** is a [data structure](https://en.wikipedia.org/wiki/Data_structure) in a [Unix-style file system](https://en.wikipedia.org/wiki/Unix_filesystem) which describes a [filesystem](https://en.wikipedia.org/wiki/Filesystem) object such as a [file](https://en.wikipedia.org/wiki/Computer_file) or a [directory](https://en.wikipedia.org/wiki/Directory_%28computing%29). Each inode stores the attributes and disk block location(s) of the object’s data.[[1]](https://en.wikipedia.org/wiki/Inode#cite_note-1) Filesystem object attributes may include [metadata](https://en.wikipedia.org/wiki/Metadata) (times of last change,[[2]](https://en.wikipedia.org/wiki/Inode#cite_note-2) access, modification), as well as owner and [permission](https://en.wikipedia.org/wiki/File_system_permissions) data.[[3]](https://en.wikipedia.org/wiki/Inode#cite_note-3)

Directories are lists of names assigned to inodes. A directory contains an entry for itself, its parent, and each of its children. (<https://en.wikipedia.org/wiki/Inode>)

Image for post



If you look at the file1 and the hlink1 you can view the inode on the left side is the same. In essence, hardlinks act as a shortcut to that file that is hard linked.

Now moving on to “Soft Links”

Soft links is a special kind of file that points to another file, much like a shortcut. Unlike a hard link, **a symbolic link does not contain the data in the target file.** It simply points to another entry somewhere in the file system. This difference gives symbolic links certain qualities that hard links do not have, such as the ability to link to directories, or to files on remote computers networked through [NFS](https://kb.iu.edu/d/adux). Also, when you delete a target file, symbolic links to that file become unusable, whereas **hard links preserve the contents of the file.**(<https://kb.iu.edu/d/abbe>)

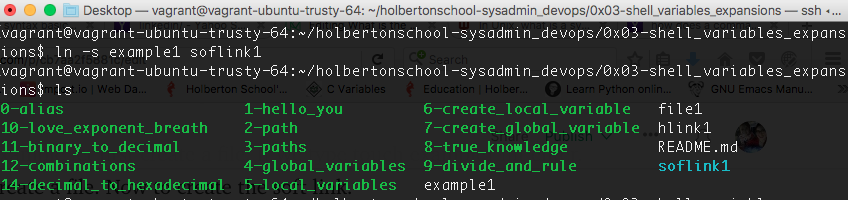
Now let's create a soft link to compare with the hard link we created above.

So let's open up a terminal. Go to the finder or search and input terminal it should pop up click on it and open.

For our example let us first create a file. If we type touch example1 hit enter. We can create a file. Now to create the soft link.

If we type in the shell ln -s example1 softlink1 we create a soft link between the files. See the example below.

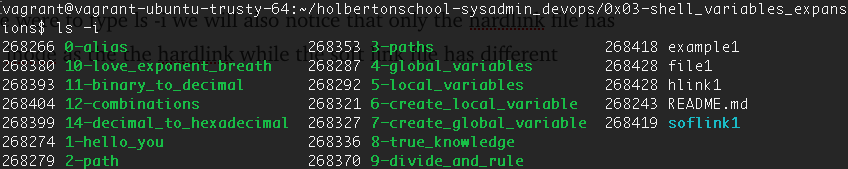
Image for post



softlink

Now if we were to type ls -i we will also notice that only the hardlink file has the same inode as the hardlink while the soft link file has different inodes.

Image for post



**what are the differences between hard and soft links?**

**A soft link does not contain the data in the target file.**

**A soft link points to another entry somewhere in the file system.**

**A soft link has the ability to link to directories, or to files on remote computers networked through**[**NFS**](https://kb.iu.edu/d/adux)**.**

**Deleting a target file for a symbolic link makes that link useless.**

**A hard link** **preserves the contents of the file.**

**A hard link cannot be created for**[**directories**](http://www.linfo.org/directory.html)**, and they cannot cross**[**filesystem**](http://www.linfo.org/filesystem.html)**boundaries or span across**[**partitions**](http://www.linfo.org/partition.html)**.**

**In a hardlink you can use any of the hardlink names created to execute a program or script in the same manner as the original name given.**

In essence:

“Underneath the file system files are represented by inodes

A file in the file system is basically a link to an inode.  
A hard link then just creates another file with a link to the same underlying inode.

When you delete a file it removes one link to the underlying inode. The inode is only deleted (or deletable/over-writable) when all links to the inode have been deleted.

A symbolic link is a link to another name in the file system.

Once a hard link has been made the link is to the inode. deleting renaming or moving the original file will not affect the hard link as it links to the underlying inode. Any changes to the data on the inode is reflected in all files that refer to that inode.

Note: Hard links are only valid within the same File System. Symbolic links can span file systems as they are simply the name of another file.”

(<http://stackoverflow.com/questions/185899/what-is-the-difference-between-a-symbolic-link-and-a-hard-link>)

**Examining the Shell and how it executes a command**

* The shell reads its input from a file, from a string or from the user’s terminal.
* Input is broken up into words and operators, obeying the quoting rules. These tokens are separated by *metacharacters*. Alias expansion is performed.
* The shell *parses* (analyzes and substitutes) the tokens into simple and compound commands.
* The shell performs various shell expansions, breaking the expanded tokens into lists of filenames and commands and arguments.
* Redirection is performed if necessary, redirection operators and their operands are removed from the argument list.
* Commands are executed.
* Optionally the shell waits for the command to complete and collects its exit status.

Image for post

