Links in Unix

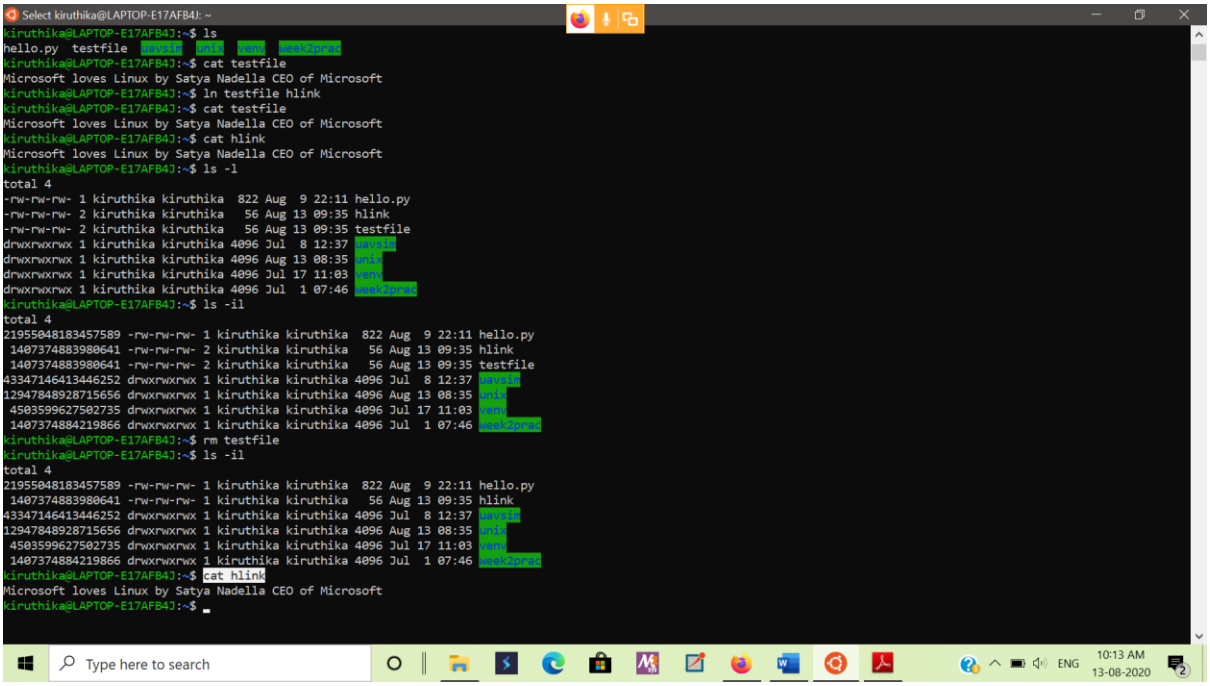
A link in UNIX is like a pointer to a file(inode). Like pointers in any programming languages, links in UNIX are pointers pointing to a file or a directory. Creating links is a kind of shortcuts to access a file. Links allow more than one file name to refer to the same file, elsewhere.

Links are used in many instances:

- To create a convenient path to a directory buried deep within the file hierarchy;
- Linking libraries
- Making sure files are in constant locations (without having to move the original)
- Keeping a “copy” of a single file in multiple locations

There are two different types of links:

- **Hard link**
  - $ ln sourcefile hardlinkname



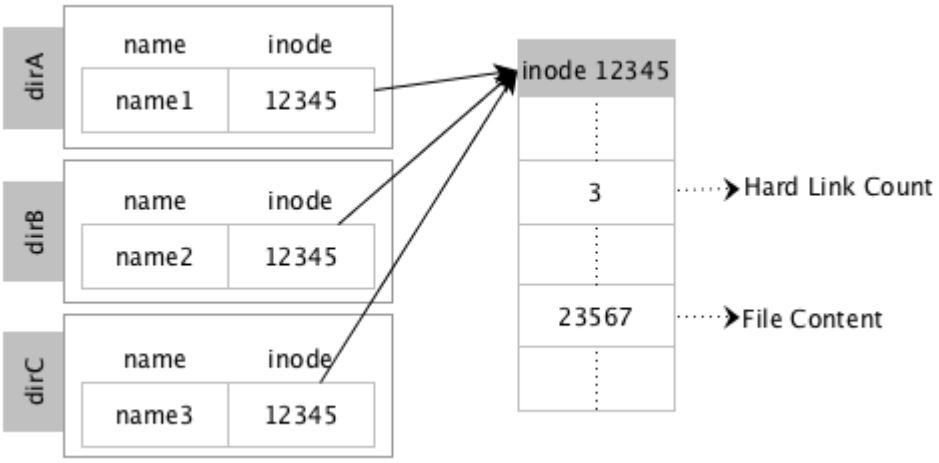
A link is just a pointer to an inode and also each filename in a directory is just a link to an inode.

A hard link is just an extra directory entry pointing to that inode. When you ls -l, the number after the permissions is the named link count. Most regular files will have one link. Creating a new hard link to a file will make both filenames point to the same inode.

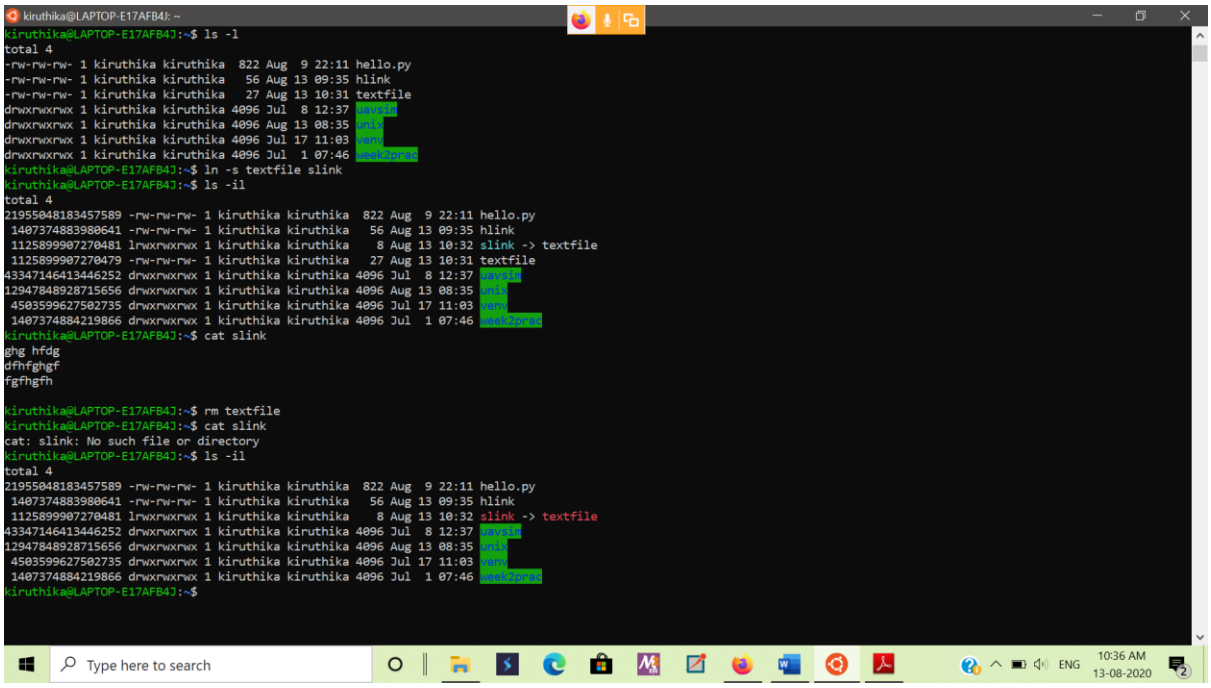
A hard link is the same as a regular name. In the above example, **testfile or hlink**, which is the original file and which is the hard link? By the end, you can't really tell (even by timestamps in which they were created, modified or accessed) because both names point to the same contents, the same inode.

The following diagram illustrates the hard link semantics. Here, directory entries in dirA, dirB, and dirC for file names name1, name2, and name3 respectively all point to

the inode 12345, which contains metadata including the text in the file and a count of the number of hard links to the file (or physical data).

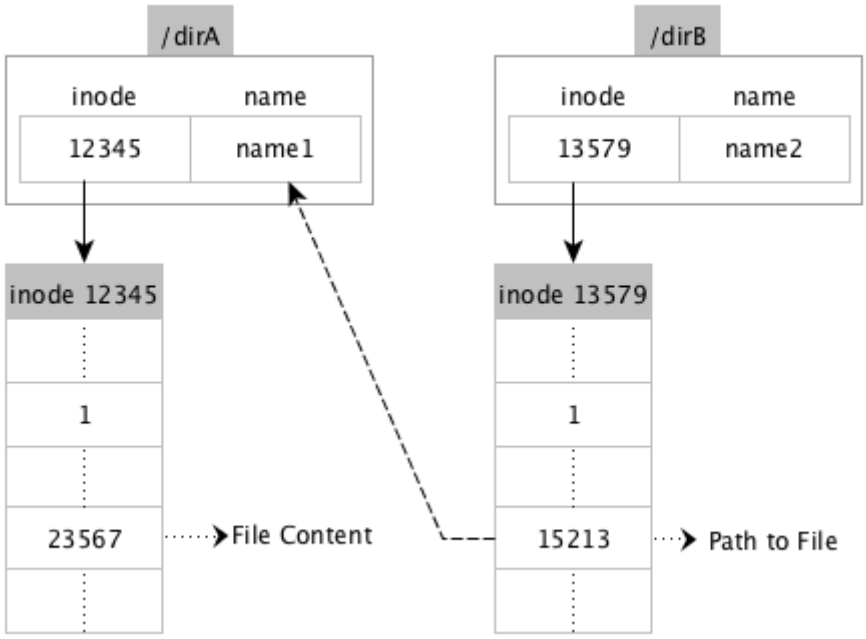


- **Symbolic link or soft link**
  - $ ln -s sourcefile softlinkname



Symbolic or soft links they point by file/directory name, and not directly to an inode.

When a symbolic link is created, a new inode is created and the text part of the inode (associated with the symlink file) contains the path to the actual file. The following diagram illustrates the symbolic link semantics. Here, the directory entry in dirA for file name, name1, is associated with inode 12345, which contains the text in the file. The directory entry in dirB for symbolic link file, name2, is associated with inode 13579, which contains the path to file in dirA (/dirA/name1). Deleting the file, name1, in dirA will result in the symlink file in dirB, name2, pointing to stale content, which in turn will return errors when accessed.



The difference between the two links are significant.

With hard links,

- you can only link to files (and not directories);
    - Allowing hard links to directories would break the directed acyclic graph structure of the filesystem, possibly creating directory loops and dangling directory subtrees, if a subdir could point back to its grandparent using a hardlink and both **the directory and its hardlink will have same inode number**, thus creating a loop.
    - Why is this loop a concern? Because when you are traversing, there is no way to detect you are looping (without keeping track of inode numbers as you traverse). Imagine you are writing the **du** command, which needs to recurse through subdirs to find out about **disk usage**. How would du know when it hit a loop? It is error prone and a lot of bookkeeping that du would have to do, just to pull off this simple task.
- With hardlinks, you cannot create reference(link) to a file on a different disk or volume
    - Because a single **inode** number must be used to represent file in each file system. And one file system may not know how data is organized in another file system. As the **inode** is a data structure used to represent a file-system object, it's internal to the File system, and you can't point to an **inode** of another file system. If allowed discrepancies might occur, that are difficult to solve.
- Hardlinks refer to the same inode as the original source file.
- A hard link will continue to remain usable, even if the original file is removed.

Symbolic (soft) links, on the other hand,

- can link to directories,

    - why can du command deal with symlinks easily and not hard links? We were able to see above that hard links are indistinguishable from normal directory entries. Symlinks, however, are special, detectable, and skippable! du command notices that the symlink is a symlink, and skips it completely, thus avoiding loop , so to allow softlinks for directories

- Soft links can refer a file/folder on a different disk or volume,
    - Because they point to the actual pathname of the file and not the inode number
- Soft links will be broken (unusable) link if the original file is deleted,
- Soft links reference abstract filenames and directories (as opposed to physical locations),
- Soft links are given their own, unique inode.

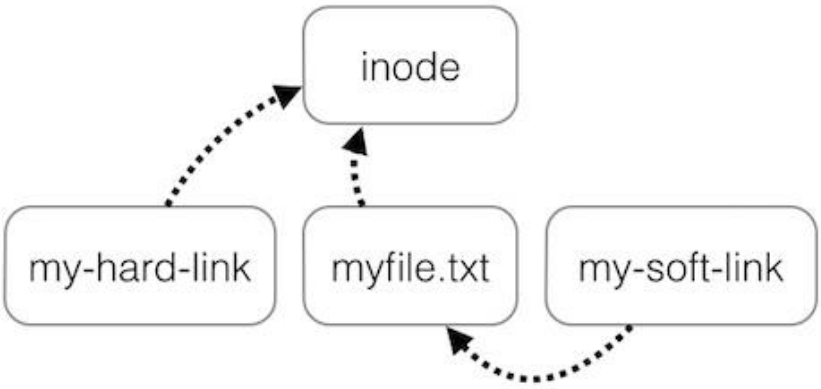


Diagram illustrating difference in hardlink and softlink