Hard links and Soft links

As was mentioned in the tutorial on file system structure, every file has a data structure (record) known as an i-node that stores information about the file, and the filename is simply used as a reference to that data structure. A link is simply a way to refer to the contents of a file. There are two types of links:

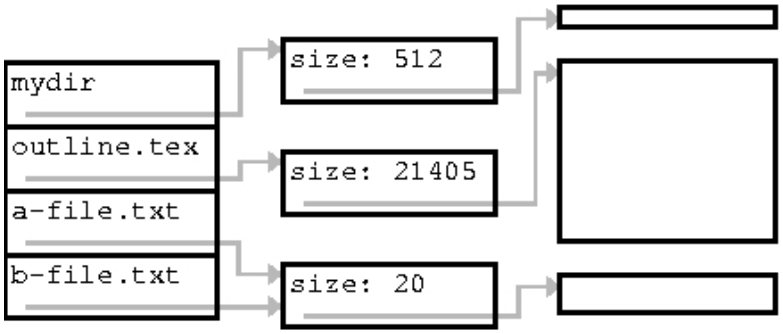
Hard links

A hard link is a pointer to the file's i-node. For example, suppose that we have a file a-file.txt that contains the string "The file a-file.txt":

```
% cat a-file.txt
The file a-file.txt
```

Now we use the ln command to create a link to a-file.txt called b-file.txt:

```
% ls
./  ../  a-file.txt
% ln a-file.txt b-file.txt
% ls
./  ../  a-file.txt  b-file.txt
```



The two names a-file.txt and b-file.txt now refer to the same data:

```
% cat b-file.txt
The file a-file.txt
```

If we modify the contents of file b-file.txt, then we also modify the contents of file a-file.txt:

```
% vi b-file.txt
...
% cat b-file.txt
The file a-file.txt has been modified.
```

```
% cat a-file.txt
The file a-file.txt has been modified.
```

and vice versa:

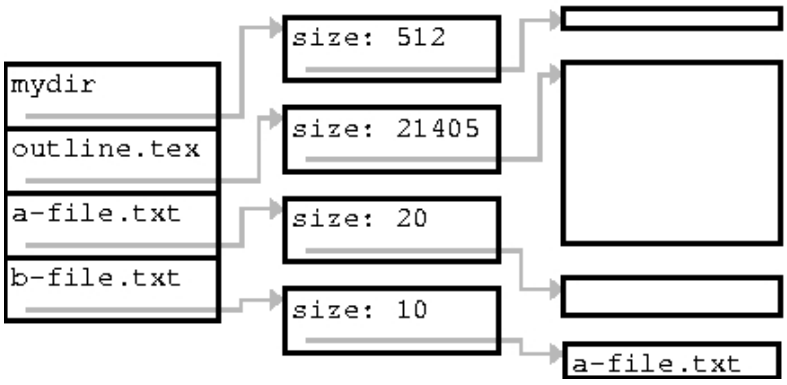
```
% vi a-file.txt
...
% cat a-file.txt
The file a-file.txt has been modified again!
% cat b-file.txt
The file a-file.txt has been modified again!
```

Soft links (symbolic links)

A soft link, also called symbolic link, is a file that contains the *name* of another file. We can then access the contents of the other file through that name. That is, a symbolic link is like a pointer to the pointer to the file's contents. For instance, supposed that in the previous example, we had used the -s option of the ln to create a soft link:

```
% ln -s a-file.txt b-file.txt
```

On disk, the file system would look like the following picture:



But what are the differences between the two types of links, in practice? Let us look at an example that highlights these differences. The directory currently looks like this (let us assume that a-file.txt b-file.txt are both hard links to the same file):

```
% ls
./  ../  a-file.txt  b-file.txt
```

Let us first add another symbolic link using the -s option:

```
% ln -s a-file.txt Symbolicb-file.txt
% ls -F
./  ../  a-file.txt  b-file.txt  Symbolicb-file.txt@
```

A symbolic link, that ls -F displays with a @ symbol, has been added to the directory. Let us examine the contents of the file:

```
% cat Symbolicb-file.txt
The file a-file.txt has been modified again!
```

If we change the file Symbolicb-file.txt, then the file a-file.txt is also modified.

```
% vi Symbolicb-file.txt
...
% cat Symbolicb-file.txt
The file a-file.txt has been modified a third time!
% cat a-file.txt
The file a-file.txt has been modified a third time!
% cat b-file.txt
The file a-file.txt has been modified a third time!
```

If we remove the file a-file.txt, we can no longer access the data through the symbolic link Symbolicb-file.txt:

```
% ls -F
./  ../  a-file.txt  b-file.txt  Symbolicb-file.txt@
% rm a-file.txt
rm: remove `a-file.txt'? y
% ls -F
./  ../  b-file.txt  Symbolicb-file.txt@
% cat Symbolicb-file.txt
cat: Symbolicb-file.txt: No such file or directory
```

The link Symbolicb-file.txt contains the name a-file.txt, and there no longer is a file with that name. On the other hand, b-file.txt has its own pointer to the contents of the file we called a-file.txt, and hence we can still use it to access the data.

```
% cat b-file.txt
The file a-file.txt has been modified a third time!
```

Although it may seem like symbolic links are not particularly useful, hard links have their drawbacks. The most significant drawback is that hard links cannot be created to link a file from one file system to another file on another file system. A Unix file structure hierarchy can consist of several different file systems (possibly on several physical disks). Each file system maintains its own information regarding the internal structure of the system and the individual files on the system. Hard links only know this system-specific information, which make hard links unable to span file systems. Soft links, on the other hand, know the name of the file, which is more general, and are able to span file systems.

For a concrete analogy, suppose that our friend Joel User is a student at both SFU and UBC. Both universities assign him a student number. If he tries to use his UBC student number at SFU,

he will not meet with any success. He will also fail if he tries to use his SFU student number at UBC. But if he uses his legal name, *Joel User*, he will probably be successful. The student numbers are system-specific (like hard links), while his legal name spans both of the systems (like soft links).

Here is an example that demonstrates a situation where a hard link cannot be used and a symbolic link is needed. Suppose that we try to create a hard link from the current working directory to the C header stdio.h, which is located on a physically different file system from the current directory:

```
% ln /usr/include/stdio.h stdio.h
ln: creating hard link `stdio.h' to `/usr/include/stdio.h':
Invalid cross-device link
```

The ln command fails because stdio.h is stored on a different file system. If we want to create a link to it, we will have to use a symbolic link:

```
% ln -s /usr/include/stdio.h stdio.h
% ls -l
lrwxrwxrwx    1 a1a1 guest          20 Apr 20 11:58 stdio.h -> /
usr/include/stdio.h
% ls
./  ../  stdio.h@
```

Now we can view the file stdio.h just as if it was located in the working directory. For example:

```
% cat stdio.h
/* Copyright (c) 1988 AT&T */
/* All Rights Reserved */

/* THIS IS UNPUBLISHED PROPRIETARY SOURCE CODE OF AT&T */
/* The copyright notice above does not evidence any */
/* actual or intended publication of such source code. */

/*
* User-visible pieces of the ANSI C standard I/O package.
*/

#ifndef _STDIO_H
#define _STDIO_H
...
```

The entire output of the cat command was not included to save space.

Note that the long listing (ls -l) of a soft link does not accurately reflect its associated permissions. To view the permissions of the file or directory that the symbolic link references, the -L options of the ls command can be used. For example:

```
% ln -s /usr/include/stdio.h stdio.h

% ls -l stdio.h
lrwxrwxrwx   1 a1a1     undergrad     20 May 10 15:13 stdio.h -
> /usr/include/stdio.h

% ls -l /usr/include/stdio.h
-rw-r--r--   1 root     bin        11066 Jan  5  2000 /usr/
include/stdio.h

% ls -lL stdio.h
-rw-r--r--   1 root     bin        11066 Jan  5  2000 stdio.h
```

Knowledge of links becomes doubly important when working as part of a software development team, and even more so when using a source code management tool such as RCS. The man page for the ln command contains more information on Unix links and the ln command.