Unix/Linux File System – directories, inodes, hard links

Ian! D. Allen – idallen@idallen.ca – www.idallen.com

Winter 2013 - January to April 2013 - Updated 2019-03-01 04:02 EST

This Term Is Finished - Material may not be updated - See current term for Updates

Updated: 2019-03-01 04:02 EST

Table of Contents

This Term Is Finished - Material may not be updated - See current term for updates

THIS TERM IS FINISHED | CONTENT MAY N

1 File systems contain names of directories and files

Index Top

Unix/Linux has hierarchical file systems consisting of directories, sub-directories, and data files. Each thing has a name in the file system tree.

1.1 Files are a list of bytes

Index Top

In Unix, a **file** is a sequence of bytes without structure. Any necessary structure (e.g. for a database) is added by the programs that manipulate the data in the file. Linux itself doesn't know about the internal structure of a database file – all it does is return bytes.

> *Most Unix books say "everything is a file", and they loosely use the word "file" to refer to anything in the file system, including directories, symbolic links, devices, etc. The manual page for the `find` command says that it can `search for files`, but it really means that it can search for any kind of thing, not just strictly a "file".*

1.2 Even hardware devices have file names

Index Top

Unix tries its best to treat every device attached to it as if it were a list of bytes. Therefore, everything, including network cards, hard drives, partitions, keyboards, printers, and plain files are treated as file-

- Your computer memory is `/dev/mem`.
- Your first hard disk is `/dev/sda`.
- A terminal (keyboard and screen) is `/dev/tty1`.
- Etc.

```
$ ls -li /dev/mem /dev/sda /dev/tty1
5792 crw-r----- 1 root kmem 1, 1 Oct 13 02:30 /dev/mem
 888 brw-rw---- 1 root disk 8, 0 Oct 13 02:30 /dev/sda
5808 crw-rw---- 1 root tty  4, 1 Oct 13 02:31 /dev/tty1
```

Most input and output devices and directories are treated as files in Linux. If you have sufficient permissions, you can directly read all these devices using their file system names. Recent versions of Unix have evolved directories into non-readable (non-file) objects.

2 Things are stored in Index Nodes = Inodes

Index Top

As with most things computer-related, things in the file system are not stored on disk by name, they are stored using a numbered data structure called an index number or **inode**.

Everything in a Unix file system has a unique inode number that manages the storage and attributes for that thing: every file, directory, special file, etc. Files and directories are both managed with inodes.

2.1 Directories map names to inode numbers

Index Top

Directories map file system names to inode numbers for you. Each inode is identified by a unique inode number that can be shown using the `-i` option to the `ls` command:

```
$ ls -l -i /usr/bin/perl*
266327 -rwxr-xr-x 2 root root 10376 Mar 18  2013 /usr/bin/perl
266327 -rwxr-xr-x 2 root root 10376 Mar 18  2013 /usr/bin/perl5.14.2
266331 -rwxr-xr-x 2 root root 45183 Mar 18  2013 /usr/bin/perlbug
266328 -rwxr-xr-x 1 root root   224 Mar 18  2013 /usr/bin/perldoc
266329 -rwxr-xr-x 1 root root   125 Mar 18  2013 /usr/bin/perldoc.stub
266330 -rwxr-xr-x 1 root root 12318 Mar 18  2013 /usr/bin/perlivp
266331 -rwxr-xr-x 2 root root 45183 Mar 18  2013 /usr/bin/perlthanks
```

The program `/usr/bin/perl`, above, is not stored on disk in the same place as its name `perl`; it is stored in an inode somewhere else, under inode number `266327`.

Unix **directories** are what map file system names (e.g. `perl`) to inode numbers (e.g. `266327`) that contain the actual data.

In the example above, you can see that file name `/usr/bin/perl` really leads to inode number `266327` and that another name `perl5.14.2` leads to the same inode number. The names are separate from the things they name.

When you access the `perl` program by name, the system finds the `perl` name in a directory, paired with the inode number `266327` that holds the actual data, and then the system has to go elsewhere on disk to that inode number `266327` to access the data for the `perl` program.

File and directory data is actually stored under inode numbers, not under names. Directories map the names to the inode numbers for you.

2.2 One inode, many names

Index Top

As you can see above, a name `perl` in a directory leads to a single inode number `266327`, but the inode `266327` may have several names that lead to it, e.g. both `perl` and `perl5.14.2`.

Each file name is mapped to only one single inode number, but one file inode number may have many names that map to it.

2.3 Inodes contain pointers to disk blocks

Index Top

A Unix inode manages the disk storage space for a file or a directory. The inode contains a list of pointers to the disk blocks that belong to that file or directory.

The disk blocks store the data for the inode. The larger the file or directory, the more disk block pointers it needs in the inode.

2.4 Inodes contain attributes (owners, permissions, times, etc.)

Index Top

Also stored in the inode are the attributes of the file or directory: permissions, owner, group, size, access/modify times, etc.

The attributes of a file are stored in the inode for the file. They attributes not stored with the name of the file, which is up in some directory.

The attributes of a directory are also stored in the inode for the directory. The attributes of the things named *in* the directory – names of files or sub-directories – are *not* stored in the directory; they are stored in the individual inodes of those things.

The attributes of a directory inode apply only to that directory itself, not to the things named *in* the directory, which have their own inodes.

The name of the file or directory is *not* stored in its own inode. Inodes have only numbers, attributes, and disk blocks – an inode does not contain its own name. The names are kept separately, in directories.

2.5 Inodes are unique inside a file system

Index Top

Inode numbers are specific to a **file system** inside a **disk partition**. Each file system has its own set of inode numbers.

Numbering is done separately for each file system, so different disk partitions may have file system objects with the same inode numbers.

2.6 Inodes are a fixed resource

Index Top

Every Linux file system is created new with a large set of available inodes. You can list the free inodes using `df -i`. Some types of Unix file systems can never make more inodes, even if there is lots of disk space available; when all the inodes are used up, the file system can create no more files until some files are deleted to free some inodes.

3 File System Diagrams are Wrong

Index Top

Most diagrams showing file systems and links in Unix texts are wrong and range from confusing to seriously misleading. Here's the truth, complete with an ASCII-art file system diagram below.

This Term Is Finished - Material may not be updated - See current term for updates

The names for inodes (names for files, directories, devices, etc.) are stored on disk in directories. Only the *names* and the associated *inode numbers* are stored in the directory; the actual *disk space* for whatever data is being named is stored in the numbered inode, not in the directory. The names and numbers are kept in the directory; the names are *not* kept with the data, which is in the inode.

In the directory, beside each name, is the index number (inode number) indicating where to find the disk space used to actually store the thing being named. You can see this name-inode pairing using `ls -i`:

```
$ ls -i /usr/bin/perl*
266327 /usr/bin/perl        266329 /usr/bin/perldoc.stub
266327 /usr/bin/perl5.14.2  266330 /usr/bin/perlivp
266331 /usr/bin/perlbug     266331 /usr/bin/perlthanks
266328 /usr/bin/perldoc
```

The crucial thing to know is that the names and the actual storage for the things being named are in *separate places*. Most texts make the error of writing Unix file system diagrams that put the names right *on* the things that are being named. That is misleading and the cause of many misunderstandings about Unix files and directories.

Names exist one level *above* (separate from) the items that they name:

```
WRONG - names on things      RIGHT - names above things
=======================      ==========================

    R O O T            --->         [etc,bin,home]   <-- ROOT directory
   /   |   \                         /    |      \
etc   bin   home       --->  [passwd]  [ls,rm]  [abcd0001]
 |   /   \    \                  |      /    \       |
 |  ls   rm  abcd0001  --->      |  <data>  <data>  [.bashrc]
 |              |                |                   |
passwd       .bashrc   --->  <data>               <data>
```

Directories are lists of names and inode numbers, as shown by the square-bracketed lists in the diagram on the right, above. (The actual inode numbers are omitted from this small diagram.)

The name of each thing (file, directory, special file, etc.) is kept in a directory, separate from the storage space for the thing it names. This allows inodes to have multiple names and to have names in multiple directories; all the names can refer to the same storage space by simply using the same inode number.

In the correct diagram on the right, the directories are lists of names in square brackets. Directories give names to the objects below them in the tree. The top directory on the right is the ROOT directory inode, containing the list of names under it: `etc`, `bin`, `home`, and others.

The line leading downwards from the name `bin` in the ROOT directory indicates that the name `bin` is paired with an inode number that is another directory inode containing the list of names in the `bin` directory, including names `ls` and `rm` and others. The line leading down from `ls` in the `bin` directory inode leads to the data inode for the file `/bin/ls`. There is no name kept with the data inode – the name is up in the directory above it.

The ROOT directory inode has no name because there is no directory above it to give it one! Every other directory except ROOT has a name because there is a directory inode above it that contains its name.

4 Directories hold only names and inode numbers

Index Top

To make a hierarchical file system, file system names are stored in directories.

Each Unix directory is itself an inode. Like all inodes, directory inodes contain pointers to disk blocks and attribute information about the inode (permissions, owner, etc.), but what is stored in the disk blocks of a directory inode is not file data but directory data. That directly data is simply a list of names and inode numbers.

The directory inode contains attribute information about the *directory*, itself, not about the things named *in* the directory. (Use `ls -ld` to see the attributes of the directory inode itself.)

On Unix, for each thing in the file system, only the name and the inode number of the thing is kept in the directory. No data or attributes about the thing are kept in the directory, only the name of the thing and its inode number.

The *name* of a file is kept in a directory, paired with its inode number. The file's actual attributes and pointers to disk blocks are kept elsewhere, in the inode for the file.

This means that names are not kept in the same inodes with the things that they name.

Directories are what give names to inodes on Unix. Directories can be thought of as "files containing lists of names and inode numbers". Files have disk blocks containing file data; directories also have disk blocks; but, the blocks contain lists of names and inode numbers.

If a directory is damaged in Unix, only the names are lost, not any of the file data blocks or the file attributes.

4.1 Attributes are stored with the inode, not the name

Index Top

A Unix directory is only a list of pairs of names and associated inode numbers. The attribute information about an item named *in* a directory – the type, permissions, owner, etc. of the thing – is kept with the inode associated with the thing, not in the directory.

Reading a Unix directory tells you only some names and inode numbers; you know nothing about the types, sizes, owners, or modify times of those inodes unless you actually go out to each separate inode on disk and access it to read its attributes. Without actually accessing the inode, you can't know the attributes of the inode; you can't even know if the inode is a file inode or a directory inode. (Some modern Unix file systems also cache a second copy of the inode type in the directory to speed up common file system browsing operations.)

To find out attribute information of some file system object, that information is stored in the inode of the object, not in the directory. You must first use the inode number associated with the object to find the inode of the item and look at the item's attributes. This is why `ls` or `ls -i` are much faster than `ls -l` on a huge directory:

a. `ls` or `ls -i` only need to read the names and inode numbers from the directory – no additional inode access is needed because no other attributes are being queried. Reading the one directory inode is sufficient.
b. `ls -l` has to display attribute information for every object named in the directory, so it has to do a separate inode lookup to find out the inode attribute information for every inode in the directory. A directory with 1000 names in it requires 1000 separate inode lookups to fetch the attributes!

No attribute information about the things named in the directory is kept in the directory (except on those modern file systems where caching of inode type is enabled). The directory only contains pairs of names and inode numbers.

To find a thing by name, the system goes to a directory inode, looks up the name in the disk space out to the disk a

second time and finds that inode on the disk. If that inode is another directory, the process repeats from left-to-right along the pathname until the inode of the last pathname component (on the far right in the pathname) is found. Then the disk block pointers of that last inode can be used to find the data contents of the last pathname component.

4.2 Damaged directories create orphans

Index Top

The name and inode number pairing in a Unix directory is the only connection between a name and the thing it names on disk. The name is kept separate from the data belonging to the thing it names (the actual inode on disk).

If a disk error damages a directory inode or the directory disk blocks, file data is not usually lost; since, the actual data for the things named in the directory are stored in inodes separate from the directory itself.

If a directory is damaged, only the names of the things are lost and the inodes become "orphan" inodes without names. The storage used for the things themselves is elsewhere on disk and may be undamaged. You can run a file system recovery program such as `fsck` to recover the data (but not the names).

The name of an item (file, directory, etc.) and its inode number are the only things kept in a directory. The directory storage for that name and number is managed by its own inode that is separate from the inode of each thing in the directory. The name and number are stored in the directory inode; the data for the item named is stored in its own inode somewhere else.

5 Multiple names – hard links

Index Top

Because (1) data in a file is managed by an inode with a unique number, (2) the name of the file is not kept in that inode, and (3) directories pair names with inode numbers, a Unix file (inode) can be given multiple names by having multiple name-and-inode pairs in one or more directories.

Inode `123` may be paired with the name `cat` in one directory and the same `123` may be paired with the name `x` in the same or a different directory. Either name leads to the same file inode and the

e directory, the

only thing different between the two is the name – both names lead to the same inode and therefore to the same data and attributes (permissions, owner, etc.).

> *You can use `ls -i` to see the inode numbers paired with each name, and the `find` command has a useful `-inum` expression operator.*

5.1 Link counts count names; `ln` creates, `rm` removes only a name

Index Top

Multiple names for the same inode are called "hard links". The system keeps a "link count" in each inode that counts the number of names each inode has been given. The `ln` command can create a new name (a new hard link) in a directory for an existing file inode, increasing the file's inode link count. The `rm` command removes a name (a hard link) from a directory, decreasing the file's inode link count.

When the link count for an inode goes to zero, the inode has no names and the inode is freed and recycled and all the storage and data used by the item is released.

The `rm` command does not remove *files*; it removes *names* for files. When all the names for a file inode are removed, the system removes the inode itself and releases all the disk space.

As long as an inode has at least one name is some directory (a non-zero link count), it cannot be freed up and released.

6 Tracing Inodes in Pathnames – ASCII Art

Index Top

Below is an ASCII Art diagram of some directories and a hard-linked file with two names.

6.1 Slashes separate names in pathnames

Index Top

When you look at a Unix pathname, remember that that the slashes separate the names of the pathname components. All the components to the left of the rightmost slash must be directories, including the "empty" ROOT directory name to the left of the leftmost slash. For example:

```
/home/alex/foobar
```

In the above example, there are three slashes and therefore four pathname components:

1. The nameless ROOT directory is the start of this absolute pathname.
2. Inside the above ROOT directory is the name of the `home` directory.
3. Inside the above `home` directory is the name of the `alex` directory.
4. Inside the above `alex` directory is the name of the `foobar` file.

> *The "empty" name in front of the first slash is the name of the ROOT directory. The ROOT directory doesn't have a name. (Some books get around this by calling the ROOT directory "slash" or `/`. That is wrong. ROOT doesn't have a name – slashes separate names.)*

The last (rightmost) component of a pathname can be a file or a directory (or any other thing, such as a symbolic link); for this example, let's assume `foobar` is a name for a file inode.

6.2 Names reside above the things they name

Index Top

Below is a file system diagram written correctly, with the names for things shown in the directory one level *above* the things to which the names actually refer. Each box represents an inode; the inode numbers for the box are given beside the box, on the left.

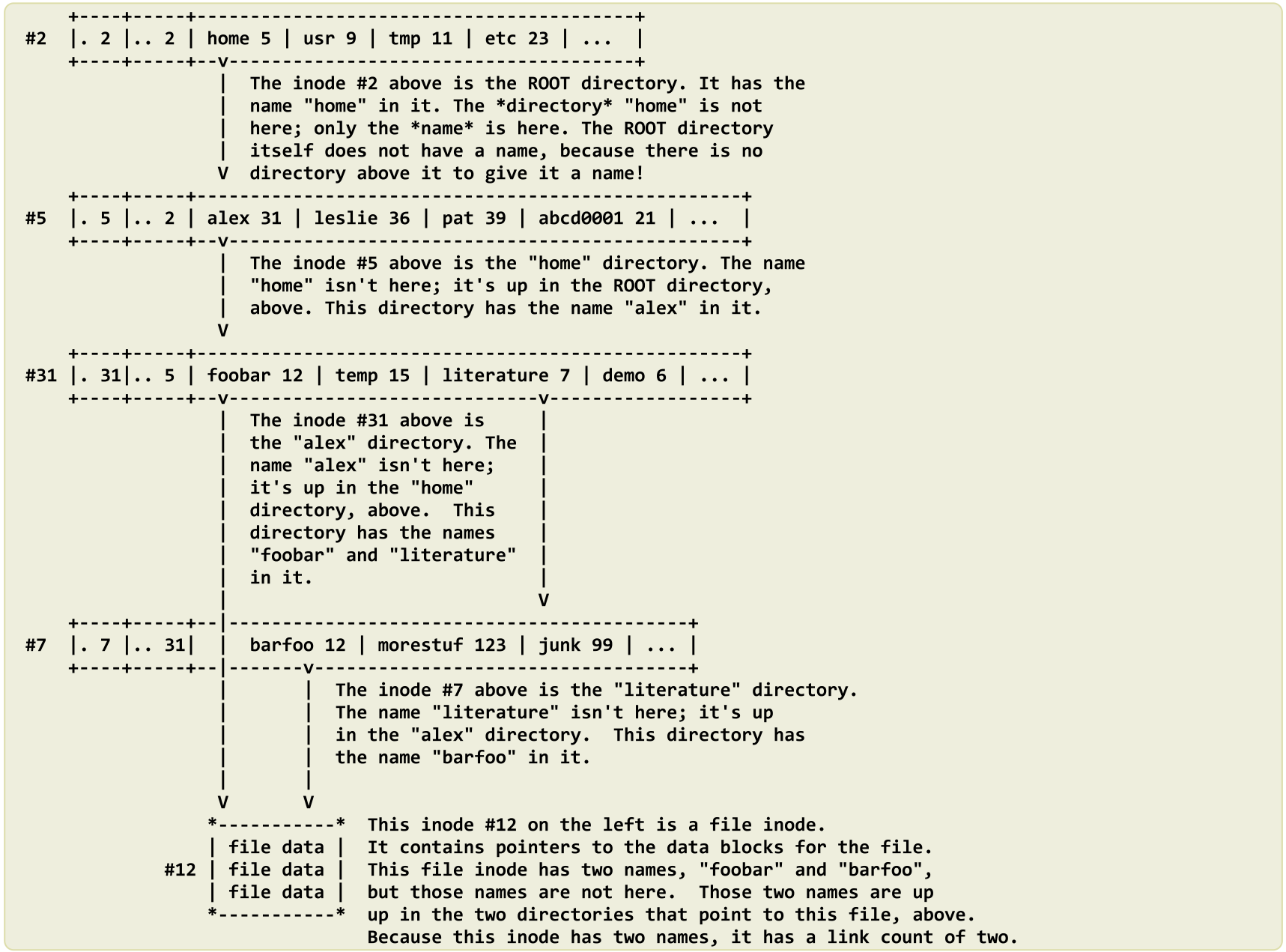
Inside the directory inodes you can see the pairing of names and inode numbers. (These inode numbers are made up – see your actual Unix system for some real inode numbers.)

One of the inodes below, `#12`, is not a directory; it is an inode for a file and the inode contains the file data.

The downward arrows in the diagram trace two paths (hard links) to the same `#12` file data, `/home/alex/foobar` and `/home/alex/literature/barfoo`.

We will trace the inodes for two pathnames in the diagram below:

1. `/home/alex/foobar`
2. `/home/alex/literature/barfoo`



The absolute pathname `/home/alex/foobar` starts at the nameless ROOT directory, inode `#2`. It travels through two more directory inodes and stops at file inode `#12`. Using all four inode numbers,

The absolute pathname `/home/alex/literature/barfoo` starts at the ROOT inode and travels through three more directory inodes. It stops at the same `#12` file inode as `/home/alex/foobar`. Using all five inode numbers, `/home/alex/literature/barfoo` could be written as `#2->#5->#31->#7->#12`.

Thus, `/home/alex/foobar` and `/home/alex/literature/barfoo` are two absolute pathnames leading to the same inode `#12` file data. The names `foobar` and `barfoo` are two names for the same file and are called "hard links". Because the file inode `#12` has two names, it has a "link count" of two.

Let's examine each of the two pathnames and their inodes in more detail.

6.3 Tracing Pathname 1: `/home/alex/foobar`

Index Top

> *Remember: Directories are chunks of storage that pair names with inode numbers. That is all that is in a directory: names and inode numbers.*

The box below represents the layout of names and inode numbers inside the actual disk space given to the nameless ROOT directory, inode `#2`:

```
    +----+-----+---------------------------------------+
#2  |. 2 |.. 2 | home 5 | usr 9 | tmp 11 | etc 23 | ...  |
    +----+-----+---------------------------------------+
```

If you look at the ROOT directory above, you will see that both the name `.` and the name `..` in this ROOT directory are paired with inode `#2`, the inode number of the ROOT directory itself.

Following either name `.` or `..` will lead to inode `#2` and right back to this same ROOT inode. The ROOT directory is the only directory that is its own parent.

The above ROOT directory has the name `home` in it, paired with inode `#5`. The actual disk *space* of the directory `home` is not here; only the *name* `home` is here, along with its own inode number `#5`. To read the actual contents of the `home` directory, find the disk space managed by inode `#5` somewhere else on disk and look there. (In fact, until we look up inode `#5` and find out that it is a directory inode, we have no way of even knowing that the name `home` is a name of a directory!)

The above ROOT directory pairing of `home` with inode `#5` is what gives the `home` directory inode its name. The name `home` is separate from the disk space for `home`. The ROOT directory itself does not have a

Let us move to the storage space for the `home` directory at inode `#5`.

The box below represents the layout of names and inode numbers inside the actual disk space given to the `home` directory, inode `#5`:

```
    +----+-----+-------------------------------------------------+
#5  |. 5 |.. 2 | alex 31 | leslie 36 | pat 39 | abcd0001 21 | ...  |
    +----+-----+-------------------------------------------------+
```

The name `home` for this inode `#5` isn't found in this inode; the name `home` is given to inode `#5` up in the ROOT directory. Names are separate from the things they name.

We see that the name `.` above leads back to this same `#5` inode, which is why `/home` and `/home/.` lead to the same `#5` inode.

We see that the name `..` above leads up to the parent `#2` inode (the ROOT inode), which is why `/home/..` leads us to `/` the ROOT.

The above `home` directory has the name `alex` in it, paired with inode `#31`. The actual disk *space* of the directory `alex` is not here; only the *name* `alex` is here, along with its own inode number `#31`. To read the actual contents of the `alex` directory, find the disk space manged by inode `#31` somewhere on disk and look there. (In fact, until we look up inode `#31` and find out that it is a directory inode, we have no way of even knowing that the name `alex` is a name of a directory!)

The above `home` directory pairing of `alex` with inode `#31` is what gives the `alex` directory inode its name. The name `alex` is separate from the disk space for `alex`.

Let us move to the storage space for the `alex` directory at inode `#31`.

The box below represents the layout of names and inode numbers inside the actual disk space given to the `alex` directory, inode `#31`:

```
    +----+-----+-------------------------------------------------+
#31 |. 31|.. 5 | foobar 12 | temp 15 | literature 7 | demo 6 | ... |
    +----+-----+-------------------------------------------------+
```

The name `alex` for this inode isn't in this inode; the name `alex` is given to inode `#31` up in the `home` directory. Names are separate from the things they name.

We see that the name `.` above leads back to this same `#31` inode, which is why `/home/alex` and `/home/alex/.` lead to the same `#31` inode.

We see that the name `..` above leads up to the parent `#5` inode (the `/home` inode), which is why `/home/alex/..` leads us to `/home`.

The above `alex` directory has the name `foobar` in it, paired with inode `#12`. The actual disk *space* of the file `foobar` is not here; only the *name* `foobar` is here, along with its own inode number `#12`. To read the actual data of the file `foobar`, find the disk space managed by inode `#12` somewhere on disk and look there. (In fact, until we look up inode `#12` and find out that it is a plain file inode, we no way of even knowing that the name `foobar` is a name of a plain file!)

The above `alex` directory pairing of `foobar` with inode `#12` is what gives the `foobar` file inode one of its two names. The name `foobar` is separate from the disk space for `foobar`.

Let us move to the storage space for the `foobar` file at inode `#12`.

The box below represents the actual disk space given to the `foobar` file, inode `#12`:

```
    *-----------*
#12 | file data |
    *-----------*
```

The name `foobar` for this inode isn't in this inode; the name `foobar` is up in the `alex` directory. Names are separate from the things they name.

> *This `foobar` inode is a file inode, not a directory inode, and the attributes of this inode will indicate that. All the attributes of an inode – type, permissions, owner, group, modify date, etc. – are stored in the inode itself. The only thing not stored in the inode is the name of the inode, which is always stored in the directory above the inode (the parent directory of the inode).*

The inode for a file contains pointers to disk blocks that contain file data, not directory data. There are no special directory names `.` and `..` in files. There are no names here at all; the disk block pointers in this inode point to just file data (whatever is in the file).

This completes the inode trace for `/home/alex/foobar`: `#2->#5->#31->#12`

6.4 Tracing Pathname 2: /home/alex/literature/barfoo

Index Top

Remember: Directories are chunks of storage that pair names with inode numbers. That is all that is in a directory: names and inode numbers.

Let's now trace the inode path for the name `/home/alex/literature/barfoo`. This pathname is a "hard link" to `/home/alex/foobar`; both the `foobar` and `barfoo` names point to the same inode number. Let's see how this is possible.

The trace from ROOT through `/home/alex` is the same as before. Things change in our second trace because of `/home/alex/literature`.

If we look at the `alex` directory inode `#31` again, we see that the name `literature` is paired with inode `#7`:

```
    +----+-----+-------------------------------------------------+
#31 |. 31|.. 5 | foobar 12 | temp 15 | literature 7 | demo 6 | ... |
    +----+-----+-------------------------------------------------+
```

The above `alex` directory has the name `literature` in it, paired with inode `#7`. The actual disk *space* of the directory `literature` is not here; only the *name* `literature` is here, along with its own inode number `#7`. To read the actual contents of the directory `literature`, find the disk space managed by inode `#7` somewhere else on disk and look there. (In fact, until we look up inode `#7` and find out that it is a directory inode, we no way of even knowing that the name `literature` is a name of a directory!)

Let us move to the storage space for the `literature` directory at inode `#7`.

The box below represents the layout of names and inode numbers inside the actual disk space given to the `literature` directory, inode `#7`:

```
    +----+-----+-----------------------------------------+
#7  |. 7 |.. 31|    barfoo 12 | morestuf 123 | junk 99 | ... |
    +----+-----+-----------------------------------------+
```

The name `literature` for this inode isn't in this inode; the name `literature` is given to inode `#7` up in the `alex` directory. Names are separate from the things they name.

We ... /literature and

We see that the name `..` above leads up to the parent `#31` inode (the `/home/alex` inode), which is why `/home/alex/literature/..` leads us to `/home/alex`.

The above `literature` directory has the name `barfoo` in it, paired with inode `#12`. The actual disk *space* of the file `barfoo` is not here; only the *name* `barfoo` is here, along with its own inode number `#12`. To read the actual data of the file `barfoo`, find the disk space managed by inode `#12` somewhere on disk and look there. (In fact, until we look up inode `#12` and find out that it is a plain file inode, we no way of even knowing that the name `barfoo` is a name of a plain file!)

The above `literature` directory pairing of `barfoo` with inode `#12` is what gives the `barfoo` file inode the other one of its two names. The name `barfoo` is separate from the disk space for `barfoo`.

You will recall that we have seen inode `#12` in the previous trace. Above, in the `alex` directory (inode `#31`), inode `#12` was also paired with the name `foobar`. In the `literature` directory (inode `#7`), inode `#12` is paired with the name `barfoo`. Inode `#12` therefore has two different names; both names `foobar` and `barfoo` are both hard links to the same inode `#12`, and the `ls` command can prove this:

```
$ ls -i /home/alex/foobar /home/alex/literature/barfoo
12 /home/alex/foobar   12 /home/alex/literature/barfoo
```

Having two names means the "link count" of inode `#12` is set to "`2`". Both names lead to the same `#12` inode and thus to the same data and same attributes. This is *one* single file with *two* names. A change to the file data using the name `foobar` changes the data in inode `#12`. That changes file data for the name `barfoo` too; because, `foobar` and `barfoo` are two names for the same `#12` inode storage – they are two names that point to the same storage inode.

All the inode attributes – everything about data inode `#12` except its name – is kept with the inode. The only thing different in a long listing of `foobar` and `barfoo` will be the names; everything else (file type, permissions, owner, group, link count, size, modification times, etc.) is part of inode `#12` and must therefore be identical for the two names. Neither name is more "original" than the other; both names have equal status. To release the `#12` inode storage, you have to delete both names so that the link count of inode `#12` drops to zero.

6.5 Summary Tracing Pathname 2: `/home/alex/literature/barfoo`

Index Top

Let's summarize the inodes used in this pathname:

Start on the left and walk the tree of names and inodes left to right. To be a valid Unix path, everything to the left of the rightmost slash must be a directory. (Thus, ROOT, `home`, `alex`, and `literature` must be directories, if this is a valid pathname.)

Start with the nameless ROOT directory in front of the first slash (ROOT doesn't have a name, since it does not appear in any parent directory) and look for the first pathname component (`home`) inside that ROOT directory (inside inode `#2`).

Let's trace the pathname:

Look in the ROOT directory (located in inode `#2`) for the name of the first pathname component: `home`. We find the name `home` inside the ROOT directory, paired with inode `#5`. Go back out to the disk to find inode `#5` that is the actual `home` directory.

> *Note how the names are separate from the things they name. The actual directory inode `#5` of the `home` directory is not the same as the inode `#2` of the ROOT directory that contains the directory name `home`. The name is stored in a different place (`#2`) than the thing it names (`#5`).*

In inode `#5`, the directory inode that has the name `home`, look for the name `alex`. We find `alex` paired with inode `#31`. Go back out to the disk to find inode `#31` that is the actual `alex` directory inode. Again, the name `alex` is contained in directory inode `#5` (`home`) and that name is stored separately from inode `#31` that is the actual `alex` directory itself.

In inode `#31`, the directory inode that has the name `alex`, look for the name `literature`. We find `literature` paired with inode `#7`. Go back out to the disk to find inode `#7` that is the actual `literature` directory inode. Again, the name `literature` is contained in directory inode `#31` (`alex`) and that name is stored separately from the inode `#7` that is the actual `literature` directory itself.

In inode `#7`, the directory inode that has the name `literature`, look for the name `barfoo`. We find `barfoo` paired with inode `#12`. Go back out to the disk to find inode `#12` that is the actual data of the file `barfoo`. Again, the name `barfoo` is contained in directory inode `#7` (`literature`) and that name is stored separately from the inode `#12` that is the actual data of the file. The name of a file is not part of the inode that makes up the actual file data.

We have found the inode that is the file data: inode `#12`. The name of this file, `barfoo`, is stored up in

6.6 Every inode has a link count: a count of names

Index Top

Every **directory inode** contains a name `..` (dot dot) paired with the number of the inode that is the unique parent directory of the inode. That unique parent directory is the only one containing the name of this inode. Because a directory can have only one parent, hard links are not permitted for directories. (The name `..` can only link to one parent directory.)

Directory inode `#5` above contains the name `..` paired with inode `#2` (the ROOT directory), and it is in that `#2` inode directory that we see that inode `#5` is paired with the name `home`. The parent directory of inode `#5` is the directory that contains the name `home`.

Unlike directory inodes, **file inodes** contain no record of which parent directories give it its names. The only thing that is recorded in the file inode is the number of names the inode has: the link count. File inode `#12` has two names, so it has a link count of two. The inode has no information about in which directories the two names are located.

There is no easy way to know which directories give a file inode its one or more names. In a file inode, there is no name `..` to point to a parent directory, because a file inode might have hundreds or thousands of parent directories (thousands of names).

If you have a file inode with multiple names (a link count larger than one) and you want to find the other names for the inode, you have to do a brute-force search in every directory on the file system to see which directories might have names paired with this inode number.

Usually, the multiple names for an inode are in the same directory or in directories that are closely related to each other (parent directories, sub-directories, or sibling directories), but that isn't always the case. In the worst case, finding all the names for a file inode may require searching for that inode number in *every* directory in the whole file system, something that could take hours on a very large file system.

A file is deleted from disk only when its link count goes to zero, i.e. when all the names for the inode are removed. Then the disk blocks for the inode are returned to the system for use by other files, and the inode (with a zero link count) is returned to the pool of free and available inodes.

6.7 Permissions on data vs. permissions on directories

Index Top

Each Unix inode has a set of permissions that govern what a process can do to that inode. Since names may have

permissions to change the name of a thing in a directory inode without having permissions to change the data in the inode that is the thing itself, or vice-versa.

- If a process has permission to change a directory inode, it can change the names of things in that directory inode, including adding names (e.g. `ln`), removing names (e.g. `rm`), or renaming names (e.g. `mv`). These are operations on directory inodes.
- If a process has permission to change a file inode, it can erase, append to, or change the content of the file itself. The file inode is different from the inode containing the name of the file.

Names are stored in directory inodes, separate from the things they name, so a process may have permissions to change the content of a file (the file inode) without having permissions to change the name of the file (the directory inode containing the name), or vice-versa.

If file data inode `#12` above has appropriate permission attributes, a process could read or write the data in that file inode. It is the permission attributes on the inode `#12` containing the file *data* that govern what a process can do with the *data* in the file.

The two names of the file, either `foobar` or `barfoo`, are stored up in directory inodes separate from the inode `#12`. The permissions on the inodes of those two parent directories containing the *names* of the file do not control whether a process can modify the inode containing the *data* of the file. It is the inode that contains the data (`#12`) that controls whether a process can read or write the data in that inode.

Directory inodes have permissions that control whether a process is allowed to pass through the directory to access the things named in the directory. This is called *access* or *search* permission (`x`).

If the any of the inodes of the directories containing the names leading down to the file at inode `#12` don't give the process *search* permission, the process won't be able to reach the file's data inode that way and won't be able to access the file's data using those directories; but, perhaps some other directories may lead the process to the same inode `#12`, if the file has another name.

To access and read the data in a file path such as:

```
/home/alex/literature/barfoo
```

you need appropriate *search* permissions on the ROOT directory inode, the `home` directory inode, the `alex` directory inode, the `literature` directory inode, and finally *read* permissions on the `barfoo` file data inode `#12`.

It is the file data inode `#12` permissions that determine whether or not you can read or change the *data* of the file. Reading or changing the data in the file requires permissions on the inode `#12` that contains the data blocks of the file itself.

It is the `literature` directory inode permissions (inode `#7`) that determine what you can do with the *name* `barfoo` of the file, because the `literature` directory (inode `#7`) is where the name `barfoo` is kept. Changing, linking to, or removing the name of a file operates on the inode of the *directory* in which the file name appears; altering the name has nothing to do with reading or changing the inode that contains the data blocks of the file itself.

You can have no permissions on the inode that contains the data blocks of the file itself (it may even be owned by some other user) and still you may be able to rename or remove one of the names of the file from a directory on whose inode you do have permissions. The name(s) of a file is(are) stored in separate inodes from the data blocks of the file.

Names are separate from the things that they name. The permissions of the names are also separate from the permissions of the data.

- Changing a *name* in a directory inode requires write and execute permissions on the *directory* inode containing the name. No permissions are needed on the inode containing the *data* of the thing being renamed. (Some recent Linux kernels have added security that changes this.)
- Changing the *content* of a file only requires write permissions on the data inode of the *file* itself, not on the inode of any parent directory that holds one of the names of the file.

Names are separate from the things they name, so two sets of permissions (two inodes) are always involved when a process tries to access a thing.

7 Exercise Questions on Hard Links and Directories

Index Top

- Normally when you do `ls -l dir` you see the permissions of the *contents* of the directory, not the directory itself. What command and options are needed to see the access permissions and link count of the directory inode itself, instead of the *contents* of a directory? (See the Worksheets and also

- When you are inside a directory, what is the name you use to refer to the directory itself? (This name works inside any directory.) What name always refers to the unique parent directory?
- How many links (names) does a brand new, empty directory have? Why isn't it just one link, as it is for a new file? (In other words, why does a new file have one link and a new directory have more than that?)
- Why does creating a sub-directory in a directory cause the directory's link (name) count to increase by one for every sub-directory created? (Recall that a link count is a count of names.)
- Why doesn't the link (name) count of the directory increase when you create files in the directory?
- Give the Unix command and its output that shows the inode number and owners of the following directories. Only show the given directory; do not show any other directories:

    a. your current directory
    b. your parent directory
    c. your HOME directory
    d. the directory named `/home`
    e. the ROOT directory
    f. the directory named `/root`

Note: Show only one line of output for each single directory; do not show the contents of the directory. Use a command (and options) that will show only the directory itself, not its contents. (RTFM)

Author:
| Ian! D. Allen, BA, MMath - idallen@idallen.ca - Ottawa, Ontario, Canada
| Home Page: http://idallen.com/ Contact Improv: http://contactimprov.ca/
| College professor (Free/Libre GNU+Linux) at: http://teaching.idallen.com/
| Defend digital freedom: http://eff.org/ and have fun: http://fools.ca/

Plain Text - plain text version of this page in Pandoc Markdown format

THIS TERM IS FINISHED - MATERIAL MAY NOT BE UPDATED - SEE CURRENT TERM FOR UPDATES
This Term Is Finished - Material may not be updated - See current term for updates