Lecture 8: Links, Processes, Aliases

Inodes

If you run a touch command, such as:

```
touch test1 && ls -l test1
```

You get this:

```
-rw-r--r-- 1 eric eric 0 Oct 27 16:06 test1
```

Now we see that this file called `test1` has a reported file size of 0, which makes sense. But let's think about it for one second: if file size is zero, where does all this other information come from? For example: the permission level `rw-r--r--`, the user and group of `eric eric` and the date of the last modification.

It's not a lie that the size of the file is zero, but it's also true that there's another place where we keep information related to the file. This is the concept of the *inode*. Inodes and the contents of a file are stored in separate places. Think of an MP3 file, for example. Most of the file content is compressed audio information. But the file also includes *metadata*: the name of the artist, track name, year it was published, etc. Now imagine if that metadata is stored separately from the audio content, but with an address to link the two together.

We can see this address using our 'ls' command.

```
ls -li cars
```

Result:

```
 total 8
25826992 -rw-r--r-- 1 eric eric 447 Jul  6 16:22 cars
```

The `-i` option shows up the inode number for our files. When we partition a file system, we divide that file system into a certain number of blocks with an equal size. We also generate a fixed number of inodes to correspond to those data blocks. Once we are finished, we can't change the number of inodes. Just like how you can run out of disk space, you can run out of available inodes and you won't be able anymore new files.

There are several other resources on the Internet which are useful for trying to understand Inodes.

Inodes - an Introduction

theurbanpenguin - Understanding Inodes

Hard Links

We've actually been using hard links with every file and directory we've used so far. An inode will point to a location on the filesystem, and will contain important *metadata* about that file. But we humans prefer to give things names that make some sense to us. So we create *links* to each of these inodes, and these links have a human-friendly name that's easier to use.

Hard Link –> Inode –> Data on Filesystem

Every file on your system has a hard link already. When all links to an inode are removed, the kernel will consider that part of the hard drive to be available for storage.

Let's create a second hard link to cars:

```
ln cars link-to-cars
```

check again. notice link ## and same inode.

```
total 32K
29491882 drwxrwxr-x 2 eric eric 4.0K Jul  8 23:48 ./
29491880 drwxrwxr-x 4 eric eric 4.0K Jul  6 19:31 ../
25826992 -rw-rw-r-- 2 eric eric  447 Jul  6 16:22 cars
25826992 -rw-rw-r-- 2 eric eric  447 Jul  6 16:22 link-to-cars
```

Two things to notice: '1' has interated up to '2'. This is the number of links to a file. Second, we can see that the inode '25826992' is the same for both ~~files~~ **links**.

(Notice how easy it is to fall into our human pattern of thinking about things, we see a list of things on the screen, and we are very used to thinking of these things as 'files.' *But they are links!* There is still only one file here!)

```
ln link-to-cars another-link
```

```
total 40K
29491882 drwxrwxr-x 2 eric eric 4.0K Jul  8 23:51 ./
29491880 drwxrwxr-x 4 eric eric 4.0K Jul  6 19:31 ../
25826992 -rw-rw-r-- 3 eric eric  447 Jul  6 16:22 another-link
25826992 -rw-rw-r-- 3 eric eric  447 Jul  6 16:22 cars
25826992 -rw-rw-r-- 3 eric eric  447 Jul  6 16:22 link-to-cars
```

Now the number of links is *3*. All three of these ~~files~~ **links** have the same inode. And another thing: they have the same timestamp, same permission, same *metadata*. And again, remember that the number of *files* in this location hasn't changed!

How is it different from making a copy of a file?

```
cp cars copy-of-cars
```

```
total 48K
29491882 drwxrwxr-x 2 eric eric 4.0K Jul  8 23:56 ./
29491880 drwxrwxr-x 4 eric eric 4.0K Jul  6 19:31 ../
25826992 -rw-rw-r-- 3 eric eric  447 Jul  6 16:22 another-link
25826992 -rw-rw-r-- 3 eric eric  447 Jul  6 16:22 cars
29492005 -rw-r--r-- 1 eric eric  447 Jul  8 23:56 copy-of-cars
25826992 -rw-rw-r-- 3 eric eric  447 Jul  6 16:22 link-to-cars
```

So, the copy has a different inode and a different timestamp, and we know from experience that if we make edits to 'copy of cars', we *will not* see those changes in 'cars' or any of those other ~~files~~ **links**.

What do you think will happen if we make a change to 'link-to-cars?' `vi link-to-cars`

```
 1 plym    fury    77      73      2500
2 chevy   nova    79      60      3000
3 ford    mustang 65      45      17000
4 volvo   gl      78      102     9850
5 ford    ltd     83      15      10500
6 Chevy   nova    80      50      3500
7 fiat    600     65      115     450
8 honda   accord  81      30      6000
9 ford    thundbd 84      10      17000
10 toyota  tercel  82      180     750
11 chevy   impala  65      85      1550
12 ferd    bronco  83      25      9525
```

So, I changed the last line (12) of the file to say 'ferd' instead of 'ford.'

```
cat another-link
```

```
 plym    fury    77      73      2500
chevy   nova    79      60      3000
ford    mustang 65      45      17000
volvo   gl      78      102     9850
ford    ltd     83      15      10500
```

```
Chevy   nova    80      50      3500
fiat    600     65      115     450
honda   accord  81      30      6000
ford    thundbd 84      10      17000
toyota  tercel  82      180     750
chevy   impala  65      85      1550
ferd    bronco  83      25      9525
```

Remember: We are looking at the same data, which has the same inode pointing to it. We just used a different link to find that inode.

What happens when we remove the original file?

```
rm cars
```

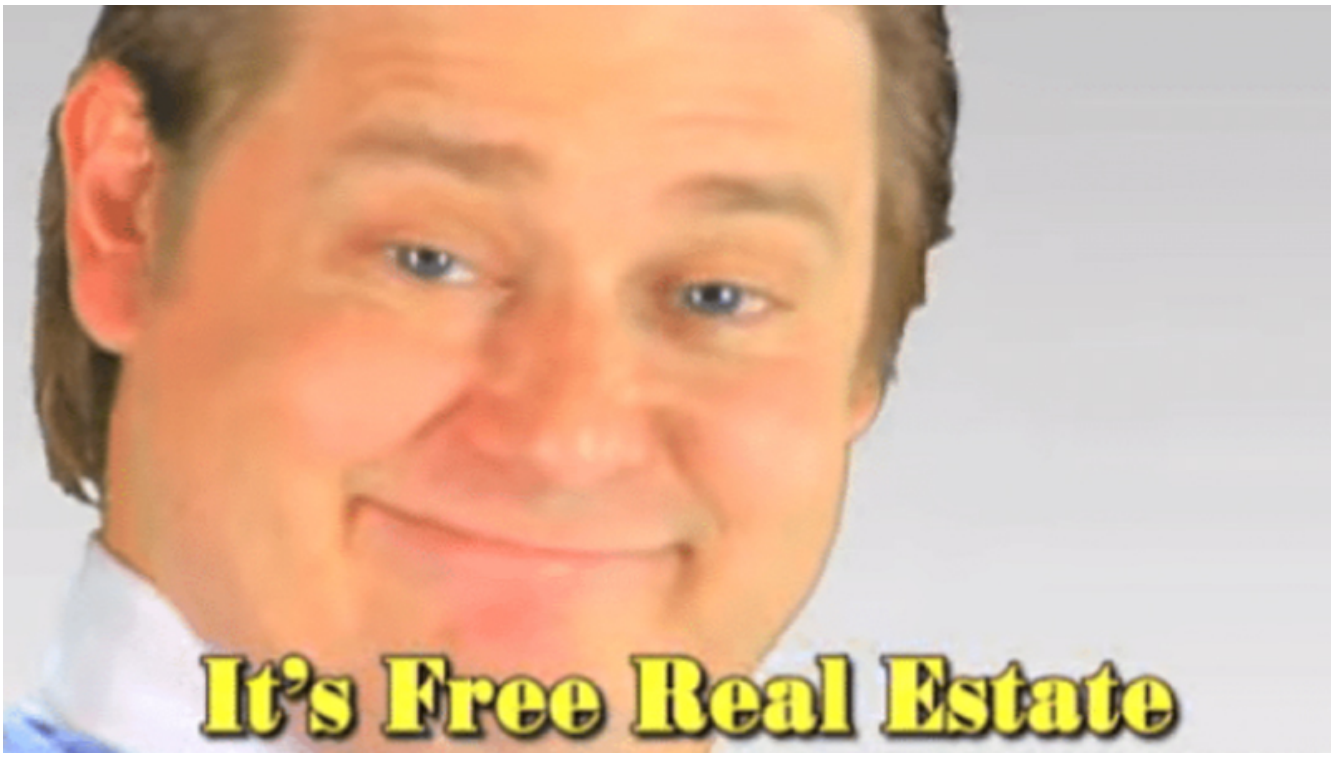
```
total 40K
29491882 drwxrwxr-x 2 eric eric 4.0K Jul  9 00:12 ./
29491880 drwxrwxr-x 4 eric eric 4.0K Jul  6 19:31 ../
25826992 -rw-rw-r-- 2 eric eric  447 Jul  6 16:22 another-link
29492005 -rw-r--r-- 1 eric eric  447 Jul  8 23:56 copy-of-cars
25826992 -rw-rw-r-- 2 eric eric  447 Jul  6 16:22 link-to-cars
```

The number of links is now *2*. But I can still open 'copy-of-cars' or 'another-link' to see the same data. *What will take it lose this data?*

```
rm link-to-cars another-link
```

```
29491882 drwxrwxr-x 2 eric eric 4.0K Jul  9 00:12 ./
29491880 drwxrwxr-x 4 eric eric 4.0K Jul  6 19:31 ../
29492005 -rw-r--r-- 1 eric eric  447 Jul  8 23:56 copy-of-cars
```

At this point, all links to that original data have been removed. The kernel will note this, and will assume that...



free real estate

What does this mean? The data might still accessible, as long as it doesn't get overwritten by anything else. This is why data recovery is sometimes still possible for data that has been 'deleted.'

To clean up, let's restore cars from our copy: `mv copy-of-cars cars`

Symbolic Links

Hard links point to an inode, and will point you to the data even if the original link is removed. Symbolic act more like hyperlinks. Consider that image above... all I had to do to add it to the document was to type in a URL. > https://i.kym-cdn.com/entries/icons/original/000/021/311/free.jpg

That creates a link to the image which sits on a server somewhere, and it isn't mine. I'm not spending any money to store that file on a server, but I'm hoping that the file will exist there for as long as this document stays relevant. But if that image gets moved or deleted, my hyperlink will look like this:

A broken link

This should give you an idea about the advantages and drawbacks of symbolic links. They are cheap (they don't take up space) but fragile (they can be easily broken). They are a great way to create shortcuts on your filesystem.

```
 cd sample_dir1/sample_dir
$ tree -C ## keep the colour!

 .
└── sample_dir
    ├── admin
    ├── cambridge
    │   ├── cafeteria
    │   └── library
    │       └── dir_practice
    ├── faculty
    ├── history.exe
    ├── markham
    │   ├── annex
    │   ├── annex2
    │   ├── building1
    │   └── parking
    ├── outline.doc
    ├── oxford
    │   ├── outline.doc
    │   ├── programming
    │   │   └── report.pdf
    │   └── security
    └── stenton
        ├── gen_ed
        │   └── cars
        ├── lib_arts
        │   ├── english.txt
        │   └── match.doc
        └── parking2

13 directories, 13 files
```

in your assignment, you are asked to run many commands on 'cars' from 'sample_dir1.' It gets tiring to keep having to type stenton/gen_ed/cars..

```
 ln -s sample_dir/stenton/gen_ed/cars car-shortcut

 total 24K
drwxrwxr-x 3 eric eric 4.0K Jul  9 00:37 ./
drwxr-xr-x 3 eric eric 4.0K Jul  8 22:21 ../
lrwxrwxrwx 1 eric eric   30 Jul  9 00:37 car-shortcut -> sample_dir/stenton/gen_ed/cars
drwxrwxr-x 8 eric eric 4.0K Jul  6 19:31 sample_dir/
```

The 'l' in permissions tells us this is a symbolic link. But also we can see where it's pointing directly from

```
 ls -l .

 cat car-shortcut

  plym    fury    77      73      2500
chevy   nova    79      60      3000
ford    mustang 65      45      17000
volvo   gl      78      102     9850
ford    ltd     83      15      10500
Chevy   nova    80      50      3500
fiat    600     65      115     450
```

```
honda   accord  81      30      6000
ford    thundbd 84      10      17000
toyota  tercel  82      180     750
chevy   impala  65      85      1550
ford    bronco  83      25      9525
```

Our cat command is redirected to the file in sample_dir/stenton/gen_ed.

What happens if that file gets moved or deleted?

```
mv sample_dir/stenton/gen_ed/cars sample_dir/stenton/gen_ed/cars.old
```

```
cat: cars: No such file or directory
```

We now have a broken link.

Directory Links

In Linux, directories are actually just a special kind of file. So links should work the same way, right?

```
ln sample_dir/stenton/gen_ed/ gen_ed-link
```

```
ln: sample_dir/stenton/gen_ed: hard link not allowed for directory
```

To create hard links for directories, you need root privileges, unfortunately. But symbolic links work fine:

```
ln -s sample_dir/stenton/gen_ed/ gen_ed-link
```

```
total 24K
drwxrwxr-x 3 eric eric 4.0K Jul  9 00:51 ./
drwxr-xr-x 3 eric eric 4.0K Jul  8 22:21 ../
lrwxrwxrwx 1 eric eric   30 Jul  9 00:37 cars -> sample_dir/stenton/gen_ed/cars
lrwxrwxrwx 1 eric eric   25 Jul  9 00:51 gen_ed-link -> sample_dir/stenton/gen_ed/
drwxrwxr-x 8 eric eric 4.0K Jul  6 19:31 sample_dir/
```

Symbolic links can be very useful for fixing broken dependencies. For example, an application might be expecting to find a certain configuration file or other software in the wrong location. But you're system might be maintaining that file elsewhere. By creating a symbolic link, you can satisfy the broken dependencies easily.

Alias

notice that I like to use `ll` instead of `ls` ... this is purely personal preference. I like being able to see timestamps and other information quickly. What is `ll` ?

`ll` does the same thing as `ls -l` . It lists files one per line with all details. `ll` is not a separate program. If you look in '/bin', you will find the programm `ls` but not `ll` . **ll is an alias of ls.** This means that ll='ls -l'. And that's exactly how we can type it in:

```
alias ll='ls -l'
```

Keep in mind that we haven't made this alias persistent yet. If we log out, the next time we log in to Matrix that alias will be lost. We'll get to making things persistent in a bit.

Processes

In Linux, everything gets a number. We have Inodes, User IDs, Group IDs, and so on. It only makes sense, then, that we'd also be numbering the *processes* that are running on our system.

A process, put simply, is the execution of a piece of code. It takes up space in memory, and is performing something useful (usually!). It's a little bit different from a program, though. For example, if you open up a

browser, you might see many processes in your list. For most browsers, every tab you have open will be running in a separate process.

When you start a computer, it will go through a POST stage where hardware is checked. From there, you get to a bootloader. If you have many Operating Systems installed, this is where you will be given a screen to choose which Operating System to use. From there, the Linux kernel is started. And the kernel will start a 'System and Service Manager.' This manager is responsibe for scheduling, basically allocating CPU time to many different processes. On most modern systems, this is *Systemd* and it will always have a process ID of 1. It is the parent of all other processes.

Try this: `ps` - This will show you a list of processes running *in this shell window* only.

```
  PID TTY          TIME CMD
7163 pts/12   00:00:00 ps
57666 pts/12   00:00:00 bash
```

Note: This will make sense later on, but you can also use `echo $$` to print your shell's process ID number.

This is fine, but what we really want is to look at *all* the running processes, not just local ones.

To get a different picture, try this:

`ps -ef` this will show you *all* processes. This is set with the `-a` flag. What is `-x` for? Don't worry about it, it's handling a legacy edge-case, basically.

`ps -ef` returns a huge list. How big?

```
ps -ef | wc -l
```

```
164
```

164 lines on Matrix. It's easier to grep for something: `ps -ef | grep sshd`

Try it out, it gives you a list of users logged in with SSH.

```
ps afux
```

This is going to give a list of *all* processes, and will show you parent/child relationships. This is usually easier to see from `top` though.

```
ps -U <username>
```

This is to see processes connected to a certain user.

There are two common ways to see processes in Linux. `ps` will spit out a list of running processes and then exit. `top` or `htop` will keep running, and you can watch your processes running in real-time `htop` is slightly easier to read than `top`, because it adds colour and some graphical elements, but it isn't installed on Matrix (and you don't permission to install it!) so it's usually safer to use `top`.

To learn more about Top, this video is a decent primer.

Killing Processes

Seeing a list of processes is educational, but really you'll usually be looking at processes if something has gone wrong. For example, to diagnose a memory leak, you might want to monitor `top` to see which process is using the most memory. Or you might be looking for *zombie* processes. (We're not going to cover zombies in this course).

If you think a process is misbehaving, you will have to *kill* it. There are two ways to do so: - Send a *TERMINATE* Signal (15). This is safe, but you will lose unsaved data. - Send a *KILL* Signal (9). This could cause system instability, and should only be a last resort.

By default, these commands use the TERMINATE signal. To kill from the command line, we can use `ps -ef | grep <application>` to find the PID. Then:

```
kill <pid>
```

Or:

```
pkill <application name>
```

This will kill *all* processes of that application, so use PIDs to be more specific.

There's one more command you might find useful. If you are using Linux with a GUI, you can often use `xkill` to click on a misbehaving window with the mouse.

Jobs

Let's say you are running a command that will take a long time to complete: `grep -r 'foo' / > foo.log 2> error.log`

This command is going to look for 'foo' in every single file in the filesystem. This could take minutes, and we don't want to stare at an empty terminal that we can't interact with.

This is where jobs come in handy. First we suspend the process: `ctrl-z`

```
^Z
[1]+  Stopped                 grep --color=auto -r 'foo' / > foo.log 2> error.log
```

Then type in `bg` to put it in the background.

```
[1]+ grep --color=auto -r 'foo' / > foo.log 2> error.log &
```

If you `ll foo` you will be able to see the file growing. The job is still running. You can also bype in `jobs`

```
[1]+  Running                 grep --color=auto -r 'foo' / > foo.log 2> error.log &
```

Let's do `top` again.

```
 top
<Ctrl+z
bg
jobs
```

what does that do?

```
 [1]-  Running                 grep --color=auto -r 'foo' / > foo.log 2> error.log &
[2]+  Stopped                 top
```

Use `fg` plus a job number to bring that job back up. `fg 2` You should see top come back up. *Remember*: processes and jobs are not the same thing. Jobs are a way that we can multitask from a command line. *One more thing*: If you enter a command and end it with &, that command will run in the background. It does the same thing as Ctrl+z and bg.

Summary

- `ln <target> <link-name>` : create a hard link of
- `ln -s <target> <link-name>` : create a symbolic link
- `alias <what you want to type>='<command to be run>'`
- `top` : system monitor, show all processes
- `ps -ef` : show all processes
- `fg, bg` : move commands to foreground or background respectively
- `jobs` : see list of jobs
- `kill` : terminate a process by PID
- `pkill` : terminate a process by program name
- `$$` : an environment variable which contains the *current processe's* ID number. Useful for scripting!