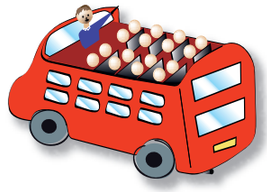
Illustration by: Seth Singh

The great Linux filesystem tour



Have you got lost in the Linux filesystem? Given up trying to figure out why things are where they are? Take the filesystem tour, as transcribed by **Dr Chris Brown**.

"Well, hello! Welcome to the Linux Filesystem Tour. My name is Manuel Page, and I will be your guide today. I and my bus driver, Hal D., are very pleased to have you on board. Just a couple of safety announcements before we start off – please keep your hands inside the bus at all times, and don't delete anything you might see along the way, unless you're sure you know what you're doing. OK, off we go.

We're starting our tour in the root directory, and I suppose you might say this is the, er, high point of the tour, because it's right at the top level of the directory hierarchy. (I don't write the jokes, folks, I just read this script.) Interestingly, the root directory is the only directory that doesn't have a name. Most people will tell you it's called **/**, but it isn't really. It's just that when we write down an absolute path name, we start with **/**, and in the case of the root directory, there's nothing else to write – we're done.

As we set off, on our left you'll notice a directory called **/root**. This is a little confusing: it isn't the root directory, it's just a directory called **root**. It's actually the private property of the system administrator. In fact, it's the super-user's home directory. I have an uncle who's the system administrator of a Solaris system, and he's always complaining that he doesn't have a private home directory. His home directory is **/**, and he hates it. How would you like to have to hang your washing out to dry in the town square? Can we take a look in **/root**, Hal? Oh, apparently not – Colonel Linux says no. Well, there's no harm in asking, but **/root**

is one of the few directories for which ordinary users don't have read permission. Generally speaking, Linux permissions implement a 'look but don't touch' policy. The main exception is in a user's home directory, where they can do whatever they want.

Speaking of Colonel Linux, our next port of call is **/boot**, which is where the colonel lives, so get your cameras or Print Screen buttons ready. He has a file there called something like **vmlinuz-2.6.19**, which is the (compressed) image of the Linux kernel. When Linux is booted, the boot loader (usually *Grub*) brings this file into memory and starts it running. You'll also notice a file there called something like **initrd-2.6.19.img**, which is an initial RAM disk image. It contains the modules the kernel needs when it's booting, before it can access the filesystem by itself. Don't delete these files, folks, or we won't be able to reboot. Do you want to put me out of a job?

Ah, the guys on the back row have noticed a directory called **lost+found**. Well done, guys! You'll see a directory of this name at the top level of any partition that contains an ext2 or ext3 filesystem. What's it for? Well, you might think it's a meeting point for stray BSD visitors, ha ha, but it's there for a program called *fsck*, which checks the consistency of the filesystem. If *fsck* finds a file that appears to be intact but doesn't actually have a name, it will create an entry for it in **lost+found**. To be honest, this hardly ever happens nowadays, so **lost+found** is probably just an empty directory. Just leave it alone, and stop worrying about it.

Configuration city

All right, coming up on your right you'll see a directory called **etc**. Historically, we think the name just stood for 'et cetera' (literally 'and other things'). This was the place to put all the stuff that didn't seem to fit anywhere else. We used to be able to stop off here for hot dogs, but nowadays it has become the home for a large collection of system configuration files and scripts. Some of these files are critically important; for example, **/etc/passwd** contains the account information for all the locally-defined logins (including root's). You'll also notice **/etc/inittab** there, which tells *init* what to do (*init* is a really special program because when Linux boots, it's the only program that gets started automatically by the kernel. It's responsible for starting all the other services, including the ones that let you log in.)

Also important is **/etc/fstab**, which tells us which other filesystems should be mounted. You probably don't mess with any of these unless you know what you're doing; errors in these files might prevent the system from booting or stop you from logging in. While in **/etc** you'll find the configuration files for various network services – there's **/etc/xinetd.conf**, which configures *xinetd*, and **/etc/syslog.conf**, which configures *syslog*. All of these files are plain text files, by the way, so the only tool you really need to configure Linux is a text editor. My preferred editor is *Vi*, but then Hal says I'm weird in other ways, too.

As **/etc** passes out of view on our right, you'll see **/home** coming up on the left. This is our residential district. Under here, you'll find the home directories of individual users. For example, Hal's home directory is **/home/hal** (isn't that right, Hal?). This is generally a pretty smart neighbourhood, but it's up to individual users what they do under here. Some folks scatter everything around in the one directory, others are really organised with things kept in multiple levels of carefully-named folders. By default, file permissions under **/home** allow you to list and examine other users' files, but you can't change them. Of course, individual users can tighten the permissions if they wish. Young Tom has a directory called **/home/tom/photos** on which only he has read permission. Tom, we'd all love to know what's in there...

Now, Hal likes to speed through **/mnt**. It's not very exciting, folks, I'm afraid – just an empty directory to temporarily mount other filesystems on to. Right next to **/mnt** we come to **/media**. This directory contains sub-directories that are used as mount points for removable media such as floppy discs or CD-ROMs. Probably the younger ones among you won't remember floppy discs? Anyway, the idea is that the hotplug system mounts media on to here automatically when they're inserted.

We've stopped going in there on the tour, ever since an alarming incident last month when Hal drove into **/media/cdrom**, just to prove it was empty. Then someone shoved a CD in, the hotplug daemon woke up, and wallop – suddenly this entire hierarchy of holiday snaps opened up in front of us. Gave us the willies, I can tell you.

The virtual part of the tour

Ah, now, the directory we're coming up to, **/proc**, is really interesting because it doesn't actually exist. All the files in here are just a figment of the kernel's imagination – they don't correspond to any information that's actually spinning round on the disc. Colonel Linux was explaining to me the other week that he had all these internal data structures to keep track of things like memory usage and lots of per-process information like their environment »

About FHS

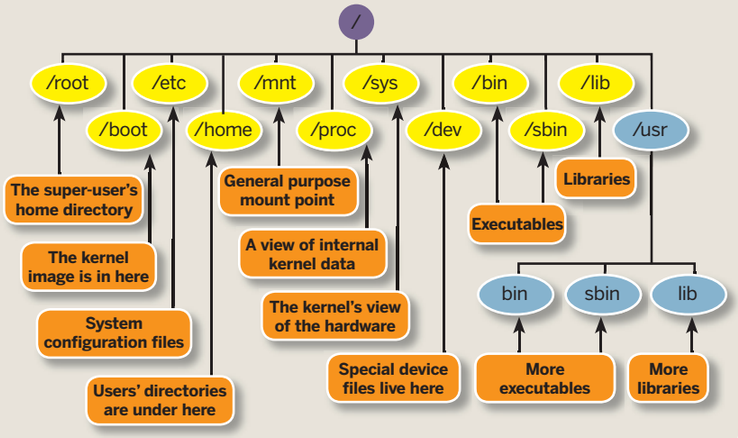
There is a document called the Filesystem Hierarchy Standard (FHS) available at **www.path name.com/fhs**. It is not a formal standards document (it is too short, too informal, and much too readable for ANSI, the IEC or ISO to put their names to it!), but it is by far the most thorough description of what should go where in a Unix/Linux filesystem, and why.

Filesystem flora and fauna

There are seven kinds of 'creature' living in the filesystem. When you do a long directory listing (**ls -l**), the very first character of each line tells you the type of creature you're looking at. These are shown in the first column of the table. We also did a population count of each type on an Ubuntu system; these figures are shown in the second column. Of course, as they used to say in the car adverts, "your mileage may vary".

Type	Population	Description
-	102,314	An ordinary file. This is by far the commonest type.
d	14,701	Directory. A directory is a container for other entries. Some people call them folders.
l	15,258	A symbolic link. These are tiny files that contain the name of some other file, similar to a shortcut in Windows. So if I have, for example, a symbolic link from **/etc/motd** to **/var/run/motd**, and a program opens **/etc/motd**, the kernel says, "Aha, that's a symbolic link. He doesn't really mean **/etc/motd**, he means **/var/run/motd**", and it opens that instead. Symbolic links are sometimes called symlinks or soft links.
c	785	A so-called character device (also sometimes called a raw device or a character special file). These entries serve to give names to devices. Some, like **/dev/console**, correspond to actual physical devices. Others correspond to pseudo-devices; for example, **/dev/random** provides access to the kernel's random number generator.
b	65	A block device. Block devices are most commonly disks – **/dev/hda2**, say, is partition 2 on hard drive a. Generally speaking, a character device supports reading and writing of a sequential byte stream, and a block device supports random access. The real distinction, however, is that the kernel provides a layer of buffering for block devices so that they are read and written a complete block at a time. It does not do this for character devices. Practically all device files live in **/dev**.
s	34	Unix-domain sockets. These are named 'communication endpoints' in the filesystem. They are used in a slightly similar way to TCP and UDP sockets, except that they only support inter-process communication between processes running on the same machine.
p	7	Named pipes. These are so rare they should be on the endangered species list! Like Unix-domain sockets, they are named endpoints used for inter-process communication.

Highlights of the filesystem tour



» variables. In the olden days, commands like *ps* (which displays information about running processes) used to fish around inside the memory image of the kernel to snag the information it needed. The colonel wasn't too wild about this – he said that it felt like having a postmortem performed on you while you were still alive. So he came up with the idea of making this information available as if it were a collection of files. That way, programs can find the information they need just by opening and reading these imaginary files, just like they would open and read any other file.

Hal's going to drive into **/proc** so we can look around. You can get a hint that there's something weird about **/proc** because if you do an **ls -l** in here most of the files have zero length, but if you examine their content with *cat* or *less*, they're not empty!

As I said, the so-called files in here show us the content of internal kernel data structures as plain text. For example, the file **cmdline** shows us the arguments that the kernel was booted with. The file **cpuinfo** shows us what the kernel knows about the CPU (or CPUs) it's executing on. The file **meminfo** tells us more about the virtual memory system that we probably wanted to know. And so on.

Sorry, madam, would you mind sitting down? Hal's just going to take this tight bend to show you a collection of directories with names like '3412'. These names are process IDs, and their directories contain yet more imaginary files that provide access to per-process information. There is actually some documentation on all of this (try **man 5 proc** for details), but much of the information is at too low a level to be intelligible to the average tourist. In most cases, it's better to use programs like *top* and *ps*, which will show you the per-process information in a more digestible form.

Mostly, we think of **/proc** as a read-only file system, but in fact there are some 'files' under **/proc/sys** that contain various kernel-tuning parameters that you can adjust by writing to them. For example, we can reduce a parameter called **TCP FIN TIMEOUT** from 60 to 50 like this:

```
# cd /proc/sys/inet/ipv4
# cat tcp_fin_timeout
60
# echo 50 > tcp_fin_timeout
# cat tcp_fin_timeout
50
```

OK, hands up those of you who don't have the faintest idea what a **TCP FIN TIMEOUT** is, or why you might want to adjust it? Yes, most of you. I thought as much. For 99% of us, the best thing is to leave this stuff alone.

Just down the street from **/proc** we come to **/sys**. This is another of those 'imaginary' filesystems. It was added to the 2.6 kernel to make it easier for kernel-level code, such as device drivers, to exchange data with programs running in user space. The hierarchy under **/sys** enables you to see the hardware environment (the busses, devices and so on) that the kernel has discovered, but unless you're rewriting, say, the Linux hotplug subsystem, you should probably ignore it entirely. There is a book */proc et /sys*, written by Olivier Daudel and published by O'Reilly, that documents all this... in French.

Toilet stop

We're going to visit **/dev** now. 'Dev' is short for 'devices', and there are some strange critters living in here. They don't really behave like ordinary files. If you do an **ls -l** in here and look carefully, you'll see a 'b' or a 'c' as the first character on the line. The b's are so-called block devices and represent devices that are block-structured and can be randomly accessed – usually this means disk partitions. For example, this little guy just here, **/dev/hda1**, is the first partition on the first hard drive. He's a block device. On this system, **hda1** is the root partition (it might be different on your machine depending on how you installed it); in fact, everything we've visited so far on our tour sits on the partition represented by this guy, so he's kept pretty busy. As soon as you click on Shutdown, he's looking forward to a bath, cocoa and bed.

On the left you'll see a large number of what are known as 'tty' devices (hi, guys!). They're character devices, representing character-based terminals. For example, Linux is typically configured to support six virtual terminals. From your graphical desktop you can reach them with the key combinations Ctrl+Alt+F1 through Ctrl+Alt+F6 (and you can get back to the desktop with Ctrl+Alt+F7). Anyway, these six virtual terminals are the devices **/dev/tty1** through **/dev/tty6**.

Moving on, straight ahead of the bus lives a very strange guy called **/dev/null**. He lives in that cave with the dark entrance. As we get closer you can see it has 'Abandon hope all ye who enter here' inscribed around the cave entrance. We've seen folk go in there, but no one ever comes out. Some people call it a black hole and use it to throw away unwanted output from programs. Er, we seem to be getting a little close to the entrance, Hal. Don't go in... you'll never... don't go in, Hal. Turn around...!

Phew, that was close.

The gritty side of town

Well, after all that imaginary stuff I'm sure you'd like to see some real files again, and there are plenty of them over here in what I think of as our industrial estate, made up of the two directories **/bin** and **/sbin**. For bin, think "binary" – most of what you'll see in here are executable programs. Why are there two? Well, the idea is

History lesson

The name 'tty' originally stood for 'teletype'. A teletype was a mechanical typewriter-style printing device with a keyboard. One model in particular, the ASR33, was extremely popular on mini-computers in the 1970s, when even Colonel Unix was still in adolescence and Colonel Linux was just a twinkle in his father's eye. Teletypes are long gone, but the name has stuck. Nowadays, a tty is a character-based screen of some sort.

Understanding the root partition

One of the principles guiding the organisation of the filesystem is to allow it to be split across multiple disk partitions (or multiple disks) in a rational manner, and to allow appropriate pieces of it to be shared between machines.

Key to this is the notion of the root partition. When Linux boots, the kernel attaches a single filesystem partition all by itself. This is known as the root partition. Any other partitions that need to be attached are mounted by the *mount* command, usually under control of entries in the file **/etc/fstab**. Because in the early stages of startup, only the root filesystem is available, it must contain everything needed for the system to function and attach the other pieces of the filesystem. Tools on the root partition include the *init* program (which starts all the other processes), a shell, *mount* and the **/etc/fstab** file. The File System Hierarchy standard specifies a number of directories that must lie within the root partition.

that stuff that ordinary users might want to use, such as *Vi* and *tar* and *rm* and *date*, lives in **/bin**, whereas things that only the super-user is likely to want to use are in **/sbin**. You can think of the 's' in **/sbin** as standing for 'system' or perhaps 'super-user'. For example, *ifconfig* (which sets network card parameters) and *iptables* (which establishes firewall rules) are in **/sbin**, because only root is allowed to do those things. On most Linux distributions, **/bin** is included on the search path for a normal user account but **/sbin** is not. Of course, ordinary users can easily access these commands using full path names such as **/sbin/iptables**, but it won't do them a lot of good because most of these commands won't let you do anything unless you're root.

There's an important corner of our industrial estate called **/lib**. Actually it's rather a large corner. 'Lib' stands for 'library', and the files in here are shared libraries required by the system programs in **/bin** and **/sbin**. (If you're from a Windows world you will know them as DLLs.) One critically important library that's used by practically everything is the standard C library, **libc.so**. If you chose to delete this file, almost everything would instantly stop working.

> **"You may feel a bit of a bump as we cross the mount point into /usr."**

While we're still looking at our industrial estate, we're going to end our tour by taking a quick look inside a very important directory called **/usr.** I should warn you that **/usr** is usually on a different partition, so you may feel a bit of a bump as we cross the mount point. Go slow please, Hal!

As we look around in **/usr**, we see directories that seem to repeat some of those in the root directory. In particular we see **/usr/bin**, **/usr/sbin** and **/usr/lib**. Indeed, these do contain the same sort of stuff that we saw in **/bin**, **/sbin** and **/lib** earlier. That is, **/usr/bin** contains user-level commands, **/usr/sbin** contains system administration commands, and **/usr/lib** contains libraries. So why is this stuff spread across two separate sets of directories? The answer is to do with partitioning. The stuff in **/usr** is often on a separate partition, which doesn't get attached into the filesystem until a relatively late stage in the boot process. Those really critical components used in the early stages of booting must lie within the root partition, not in **/usr**.

Splitting stuff up in this way also keeps to a minimum those parts of the filesystem that need to be intact to do a single-user boot. The stuff in **/bin**, **/sbin**, and **/lib** is needed, but the stuff in **/usr** isn't. It's also possible to mount **/usr** into the filesystem read-only, for improved security, and on a network it may be possible to share **/usr** out from a single file server, at least among machines sharing a common hardware architecture.

Actually, it turns out that the great majority of executables and libraries live in **/usr**, and relatively few in the root partition. A check on the disk space usage of the system I'm currently running looks like this:

```
$ du -sh /bin /sbin /lib /usr/bin /usr/sbin /usr/lib
4.9M    /bin
6.3M    /sbin
109M    /lib
92M     /usr/bin
5.7M    /usr/sbin
625M    /usr/lib
```

The figures you'll see on your own system will be different of course, but the general message will be the same: most stuff is under **/usr**, and the root partition can be relatively small.

There are a few directories like **/var** and **/tmp** and **/opt** that we haven't visited, but I know I have to get you back in time for you to catch a few big downloads at the FTP mirrors, so we'll return to the root directory where we began and close our tour. Take care getting off the bus. We hope you'll spend a few minutes in our souvenir shop, where you can buy public key rings with a plastic Tux on the fob and postcards that say 'I did an ls -R of /proc and survived!'. So long!" LXF