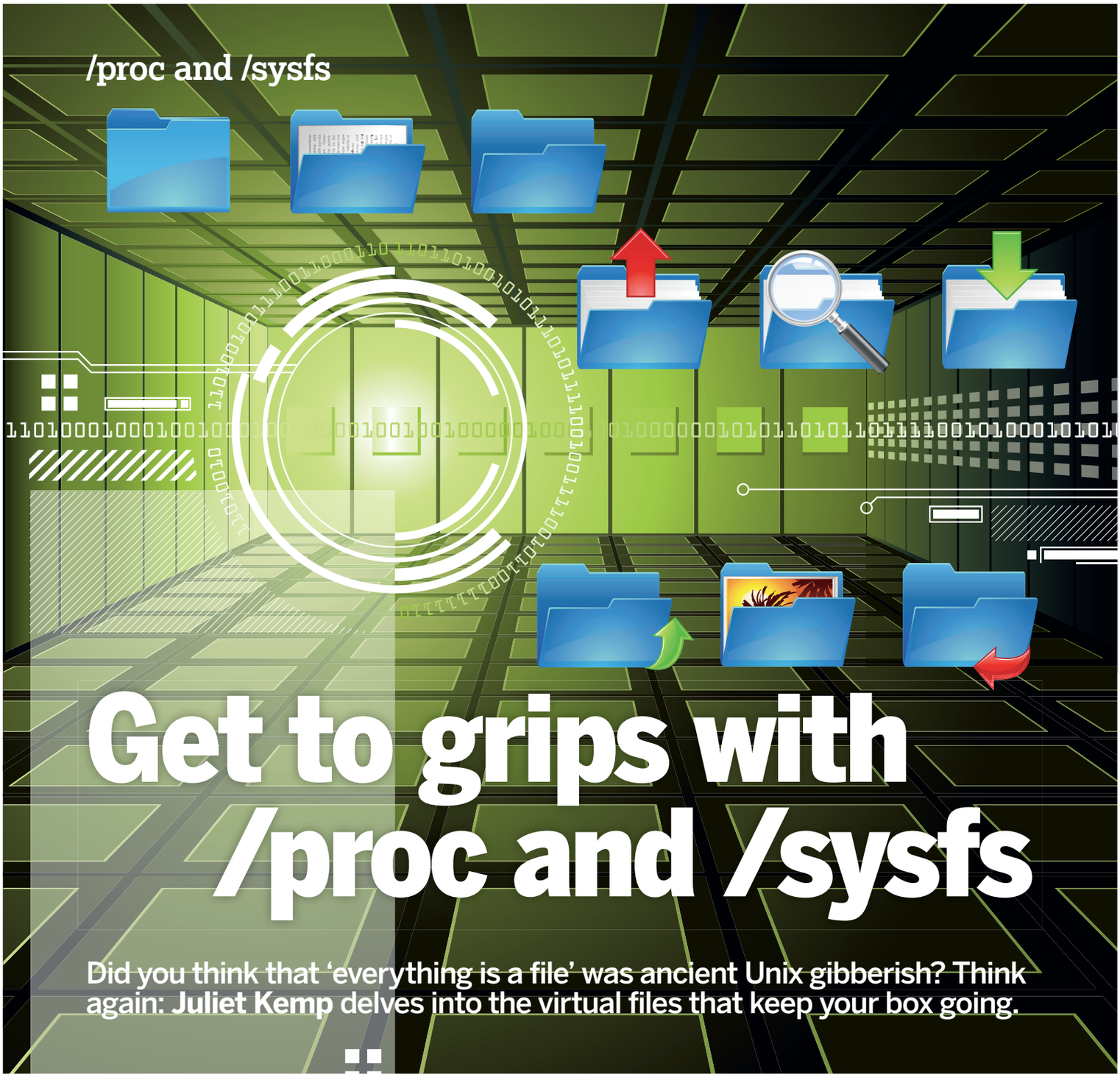


Get to grips with /proc and /sysfs

Did you think that 'everything is a file' was ancient Unix gibberish? Think again: **Juliet Kemp** delves into the virtual files that keep your box going.

The /proc directory (short for process filesystem), is fundamentally a myth: it doesn't really exist anywhere at all. Instead, it's a virtual filesystem, generated at boot and updated thereafter through interacting with the kernel. It doesn't use any disk space (because it doesn't really exist!), and it uses only a small amount of memory.

When you ask to read a file, information is retrieved by **/proc** talking to the kernel, and then the information is handed back to you as though it were a file. This makes it great both for communication between different parts of the system (it's used by various utilities, including the GNU version of *ps*, with the major security advantage that such utilities can operate entirely in userspace, just interacting with **/proc** rather than working in kernel space at all), and for pootling around with to satisfy your own interest in what's going on under the hood of your system.

> **"The /proc directory is fundamentally a myth: it doesn't really exist."**

In older Unices (such as BSD and Solaris), **/proc** is strictly process-related, but in Linux it's extended to non-process data as well, making it even more useful. This also means that in some cases you can use it to make changes as well as to read data. This is discussed later on in this article: you can make changes in **/proc** files that change system settings on the fly, but you can't change process data by editing files or directories in **/proc**. Which is probably for the best, as it could have interesting results! Do be aware when messing around with the bits of **/proc** that are editable that you're making changes to your system on the fly and that it's possible to screw things up...

/proc and processes

There are two main sections to **/proc**: process data and system data. To take a look at how **/proc** handles processes, start off by typing **ls /proc**. You'll see a long list of numbered directories: these are your processes, one directory per process ID. Have a look at your process list (using **ps -A**), and pick a process. (Note that if you're not logged in as root or using *sudo*, you'll still be able to see all the process directories and their directory listings, but you won't be able to look at the contents of all the files; in this case, you're better off using **ps -u** to list your own processes and picking one of them to look at. Even then you may not have access to all files – try a few different processes if you run into permissions errors.)

The process we'll use as a sample is

```
juliet  25175  0.0  0.0  18044  1552 pts/31   Ss   Jul14   0:00 /bin/bash
```

This corresponds to the directory **/proc/25175/**.

Use **ls -l** to look at the contents of that directory (the **-l** option will give you the target for any links, which is useful). The file **cmdline** tells you what command was used to launch this process: you may be warned that this might be binary, but in fact it's mostly text and easy to read.

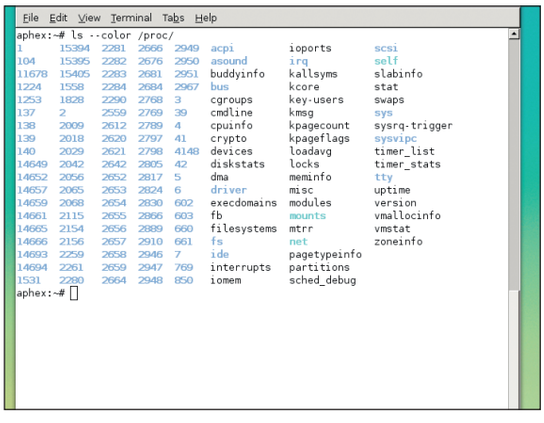
This information can be useful if you're trying to debug something that has a launch-related problem: you can check whether the path to the command was specified, or whether there's anything else making its way on to the command line, as may be the case with processes started from a script in **/etc/init.d/**. For example, on one of our machines running *Nagios 3*, the **/proc/2607/cmdline** file reads like this (the peculiar characters are correct: that's the text rendition of the binary parts):

```
/usr/sbin/nagios3^@-d^@/etc/nagios3/nagios.cfg^@
```

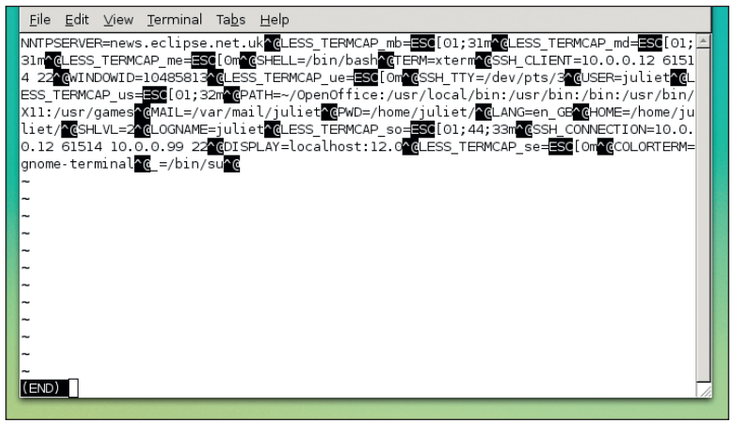
So it's loading the config file **/etc/nagios3/nagios.cfg**, which may be useful information if we're having config issues.

The **environ** file, similarly, gives you the command environment (again this is a text/binary file, but parts of it at least are parseable by eye), which is also useful for troubleshooting. Try starting the process on another machine and comparing the two files with **vimdiff** or **gvimdiff** (both easier on the eye than **diff** itself) to see if you have any differences that might explain the problems. This is especially useful if you can run something as user1 but not user2, or on one machine but not another.

Note that in the case of some processes **environ** is one of the files that only root can access. If you're not root and this is true of your process, try looking at another one.



Directory listing of my desktop's /proc directory



As you can see here, environ contains some binary junk but is still readable.

exe has a link to the original executable file itself, if it still exists: a program can continue running after the executable has been deleted.

The **fd** directory allows you to check which files the process is accessing (information that you can also get via **strace**, but this can be slower, and **strace** output can be very verbose). **task** contains hard links to any tasks (ie other processes) started by this process: so, any child processes, but also to the process itself. Somewhat disappointingly, the directory **/proc/298/task/289/task** doesn't exist, so you can't keep looping round forever…

Where processes work

Find out what directories the process is accessing (or running with respect to) by looking at the links **cwd** and **root**. **cwd** is a link to the working directory of the process. For system processes this is likely to be **/**; for something like the *Bash* session you kicked off by logging on, it'll be your home directory; for other user processes it will of course reflect which directory the user was in when they started the process. **root** is a symlink to the root path, and is almost always **/**, unless the process is being run in a *chroot* jail. Again, this can be useful information when bugfixing: if a process is using a different root path and you weren't aware of it, unexpected things might happen.

maps shows the memory maps to executables and library files. For each executable or library, the memory address is given, together with the permissions (r/w/x, s(hared), and p(rivate – copy on write)), the offset into the file, and the device and device inode. For our *Bash:*

```
00400000-004ba000 r-xp 00000000 fe:00 4086
/bin/bash 006b9000-006c3000 rw-p 000b9000 fe:00 4086
/bin/bash 006c3000-00799000 rw-p 006c3000 00:00 0
[heap] 2adc4d2f3000-2adc4d30f000 r-xp 00000000 fe:00
105924   /lib/ld-2.7.so
```

Finally, **stat** and **status** both give status information, although the one that you'll want to look at is **status**, as this is nicely formatted, labelled and otherwise readily human-readable. This is the full list that **ps** pulls selected information from, and unlike **ps -l** or any other long-information **ps** option, **status** is arranged vertically rather than horizontally so it won't run sideways off your screen.

We haven't covered all of the files and links in the process directories, just the main ones: for further information, the *proc* man page is incredibly comprehensive and well worth a look. Or you can just poke around a bit and see what you find!

Quick tip

/proc/self will always point to the current process, ie the process that's currently accessing it.

/proc and information

So **/proc** deals with process information, and we've taken a look at what you can get out of that. It also deals with system information: this is what's hapening in all those other files and directories at the first level of the **/proc** directory listing that have real names rather than numbers.

You may already be familiar with *CPUinfo* and *Meminfo*, which tell you about the machine's CPU and memory respectively. *CPUinfo* includes information about processor CPU and cache, processor speed, and power management; *Meminfo* gives the total available memory and then a big stack of facts about cache, vmalloc, free memory, and so on. All frequently useful information and very easy to access and read from here.

You can also get uptime and version information from **uptime** and **version** (this is where the *uname* command gets its information from). And **/proc/cmdline** tells you what options were passed to the kernel at boot time, for example:

```
auto BOOT_IMAGE=Linux ro root=302 hdc=ide-scsi
```

This machine is using the **Linux** boot image (in this case you can check **/etc/lilo/lilo.conf** to find out which file this corresponds to – *Grub* uses **/boot/menu.lst** as a rule), with the root partition initially mounted read-only (this is normal). **root=302** indicates that the root partition is the '3 major, 2 minor' device. To find out which this is, check the partitions file, which lists devices by major and minor number. In our case, it looks like this:

major	minor	#blocks	name
3	0	39082680	hda
3	1	9767488	hda1
3	2	14651280	hda2
3	3	14161297	hda3
3	4	498015	hda4

so the root partition is **hda2** (as **df** will also tell us!). The final option in **/proc/cmdline** tells the kernel to treat the DVD drive (at **hdc**) as IDE-SCSI.

The **acpi/** directory contains ACPI (Advanced Configuration and Power Interface) information – what exactly is in here will depend on what reporting your hardware supports. **acpi/thermal_zone/** may contain information from the internal temperature sensors if this reporting is supported (unfortunately it isn't on our desktop!). There's also an **apm** file for looking at Advanced Power Management information if your system has that enabled (usually the case for laptops). This is the file that **apm -v** gets its information from, and it's useful for checking your battery status on a laptop if you're using a console and/or don't have a graphical widget running.

> **"sys/ is the directory that corresponds to kernel variables."**

Get locked

locks displays the files that are currently locked by the system. Here's a few sample lines:

```
1: FLOCK  ADVISORY  WRITE 4056 03:02:588739 0 EOF
2: FLOCK  ADVISORY  WRITE 2747 03:02:596797 0 EOF
3: POSIX  ADVISORY  WRITE 2728 03:02:596796 0 EOF
4: POSIX  ADVISORY  WRITE 2705 03:02:642377 0 EOF
5: POSIX  ADVISORY  READ  2507 03:02:572375 4 4
```

FLOCK locks result from an **flock** system call; **POSIX** locks from a newer **lockf** system call. **ADVISORY** locks (unlike **MANDATORY** locks) don't prevent other processes from accessing the data, just from locking it, and you are also shown whether the lock is for read or write access. The fifth column is the process ID of the process holding the lock, and the next column is the file ID, in the form of **major-device:minor-device:inode**. The final columns show the start and end of the locked region: so in the first four lines that's the whole file (0 to end of file). This is probably more of academic than practical interest, but occasionally it can be useful, for example if you're trying to retrieve deleted files by inode number.

To find out what filesystems your system supports, check out **/proc/filesystems**, which lists the available filesystems and marks them with **nodev** if they're virtual or networked. This file is useful if you're trying to connect disks from other systems (either locally or remotely) and want to find out quickly if it's going to work without recompiling the kernel.

kcore is more memory information – unfortunately it's not human-parseable at all, though (but it is useful to *GDB* and other debuggers). **/proc/kmsg** deals with kernel messages. If you have a look at this file as root you should be able to see kernel messages, although if there's not much kernel activity this may look blank, and on at least one system we've tried it on it doesn't seem to do much at all.

Change network settings

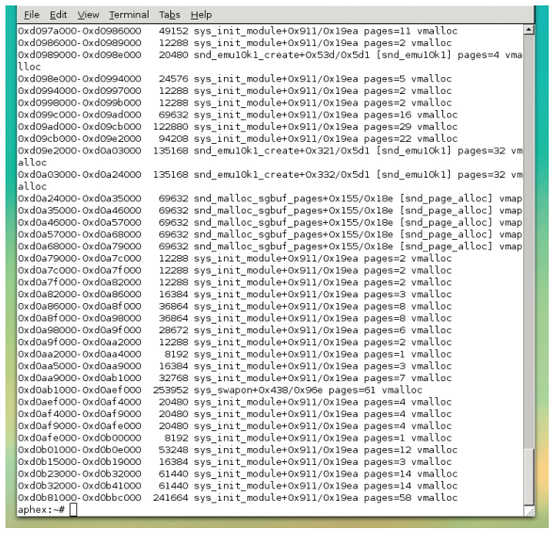
The **net/** directory provides the raw info for various networking information commands, such as *route*. In most cases it'll probably be easier to get the data from the relevant commands: technically these files are human-readable but there's largely just a lot of numbers in there! Column headings are provided if you want to take a look.

The **sys/** directory is genuinely useful for more than just interest's sake, as it is the directory that corresponds to kernel variables. As mentioned earlier, the **text/system** communication of **/proc** in some cases goes both ways: as well as accessing information, you can also change it by editing files if the permissions are correct. **sys/** is the main directory in which you might find yourself doing this.

For example, look at **/proc/sys/net/ipv4/** to see some networking values that you can change if you want (and if you

Quick tip

To find out which file is referred to by a particular inode number, you can use **find / -inum INODE_NUM**. This may take a long time on a large system!



› proc: full of fascinating yet sometimes baffling information (this screenshot shows /proc/vmallocinfo).

have root privileges). Have a look at **/proc/sys/net/ipv4/tcp_keepalive_time**. This sets how long (in seconds) it takes the TCP keepalive routines to send the first keepalive probe. The default is 7,200 seconds (2 hours); if you want to start keepalives sooner, you can edit this file accordingly. The rate at which they're sent after the first one is governed by **/proc/sys/net/ipv4/tcp_keepalive_intvl** (75-second gaps by default), and the number of dropped keepalives before the connection is marked inactive is set in **/proc/sys/net/ipv4/tcp_keepalive_probes** (this is nine by default). If you've got a flaky network, this last one may be worth increasing: to change it to 15, use:

```
echo 20 > /proc/sys/net/ipv4/tcp_keepalive_probes
```

(as root). Note that if you're changing settings here, you shouldn't use an editor – use **echo** instead, as shown.

This avoids the risk of the kernel changing the value under you while you're editing, which could happen if you use an editor to open the file – bear in mind that these files don't really exist, but are just a pretend file getting values into and out of the kernel. Similarly, you should use *cat* (piped through less if necessary) to look at files.

To enable IP forwarding, change the value in **/proc/sys/net/ipv4/ip_forward** to 1 (0 disables it).

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

There are plenty of other files you can play around with. Basically, if a file has the write attribute set, you can change it – but do be aware that you could screw up your system by doing this! The good news is that changes made in this way will last only until the next boot; to make changes permanent, use **sysctl** instead. Working directly with the **/proc/sys/** settings is a useful way to experiment before making permanent changes.

There's plenty of other variables in **sys/** that can be fiddled with. Changing **/proc/sys/fs/file-max** will change the number of filehandles that are available – this will get rid of error messages stating that no more files can be opened because the maximum number of open files has been reached. The default is 4,096, but you can use any number. (It's probably best to change this only if you actually start seeing these errors.) You can also do the same thing for inodes with **/proc/sys/fs/inode-max**, although the total number of inodes available overall on the system can't be changed in this way.

/proc/sys/kernel/ctrl-alt-del enables you to set the response to the Ctrl+Alt+Del key combination. 1 will set this to be a graceful shutdown (like typing **shutdown -h** now; 0 will be an immediate shutdown (like turning the power off). 1 is probably a safer value (after all, if you really have to do an immediate non-clean shutdown you can always turn the power off).

You can use **/proc/sys/kernel/hostname** to configure your network hostname – be careful doing this if you have DHCP, as you might create a conflict.

Quick tip

You can use **sysctl** to manipulate kernel parameters, as well – type **sysctl -a | less** to look at a list of parameters.

Recognise devices

/proc/sys/ isn't the only place where you can usefully edit values. Another useful trick is to use **/proc/scsi/scsi** to get your system to recognise a new hot-swap SCSI drive. You will need to know a few pieces of information:

- The host adapter ID (the first adapter is 0).
- The SCSI channel on the host adapter (the first channel is designated 0).
- The SCSI ID of the device.
- The LUN number (the first LUN is designated 0).

Then use

```
echo "scsi add-single-device a b c d" > /proc/scsi/scsi
```

to add your disk to the system. After that you can mount filesystems, format it, or whatever else you need to do – use **fdisk -l** if need be to check which device it is.

As with the process directories, there's a lot of information in **/proc/** and we haven't covered all of it here. Spend some time having a poke through **/proc** with the man page to find out what else is there and what you might be able to change for maximum tweak value.

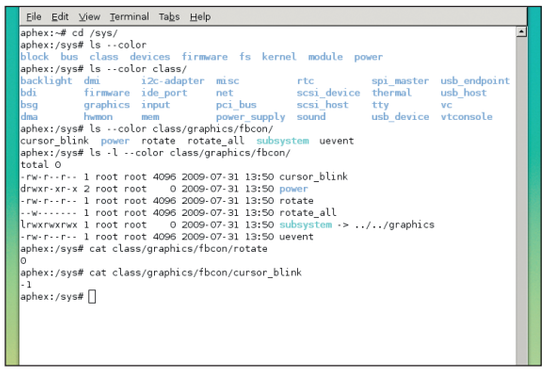
sysfs

Until the 2.6.x kernels, device information was also all kept in **procfs**. However, since the 2.5 kernel development cycle, this information has been moved to sysfs, which is another virtual filesystem, exporting device and driver information from the kernel to userspace.

Every time a new driver or device/class device is added, a new directory somewhere in **/sys** is created. **/sys/devices/** is organised to mirror the physical layout of the various devices, so you can see parent/child relationships. **/sys/bus/** has a similar structure, ending up with symbolic links (and thus showing you which bus a particular device belongs to).

Another way to look at devices according to which sort of device they use is via **/sys/class/** (this is likely to be the most useful/comprehensible approach). Some of the attributes here, once you've gone far enough down the directory tree, may be editable. For example, rotation of the framebuffer console is governed by **/sys/class/graphics/fbcon/rotate**: if you set this to 2 instead of 0, your console will be upside down! (This only works if it's supported in your distribution's kernel.)

Not all devices have attributes: the best bet is to look around the **/sys** directory, or search **www.kernel.org** for the relevant documentation. LXF



Here's the contents of sysfs on a typical desktop machine.

More information

Unfortunately the **proc** documentation can be a bit lacking in some cases. Start with the man page; you may also be able to find some information in **/usr/src/linux/Documentation/**.

Alternatively, take a look through **/proc** and then Google for whatever file or directory names you find. There's a really useful article at **www.ibm.com/developerworks/linux/library/l-adfly.html**, which focuses on things that you can change on the fly with **/proc**; then there's also **www.kernel.org** for all your kernel needs.