



## DEVICE-NAMING SYSTEM

# Connect your devices with udev

Neil Bothwick explains how to code a name for your hardware when 'Bob' just won't do.



'Everything is a file' is one of the Unix creeds. It sounds strange at first, but in a way it's true. Of course, we're not suggesting that your hard disk is a file – we all know it's a precision-engineered piece of electromechanical hardware designed to store as much of your valuable data as possible before crashing the heads into the disk and destroying the lot.

However, your hard disk *is* represented as a file in the Linux filesystem, usually as `/dev/hda/`. You probably already know this, but any piece of hardware you connect to your computer is represented by a device file in `/dev`, be it your MP3 player or your webcam.

The `/dev` directory was originally a standard directory containing device files for every piece of hardware likely to be connected. This usually meant that whenever a driver was installed, the relevant files were created in `/dev`. This had two really important disadvantages. The first was that as more devices were supported, the number of files in the directory was becoming unmanageable.

It also meant that if you tried to connect a piece of hardware for which there was no device file, you had to create it yourself, first scouring Google for the correct major and minor device number to pass to the `mknod` command.

As the number of devices supported by Linux increased, especially the huge number of removable devices that could be connected to USB or IEEE1394 (aka Firewire) ports, this became unacceptable. Not only was `/dev` becoming totally unwieldy, but we were in danger of running out of major and minor device numbers to cover every possible device that could be connected, even though any one computer would only ever see a tiny fraction of them.

The solution was `devfs`, a system in the Linux kernel that would react to devices being connected or discovered and

create their `/dev` entries automatically. While this improved the situation, there are some issues with `devfs` that mean its use is now deprecated. The main problem with `devfs` is that it has a number of bugs, ranging from annoying to serious, some of which cannot be fixed.

## Knowing u – a-ha!

Udev is a new alternative developed by Greg Kroah-Hartman that can do all that `devfs` needs to do but in user space, avoiding the need to keep any code for it inside the kernel. Using the new `/sys` filesystem from kernel 2.6 and the hotplug system for connecting peripherals, all the device node creation is handled by user space programs. As `devfs` is not being actively maintained now, udev has become the default choice. If you have installed a recent distribution, you probably already have udev without realising it.

At this point, you may be wondering why all this matters to you. After all, the main differences between `devfs` and udev seem to lie in the implementation, and how they affect the system from a development point of view. So how does it affect the end user? Well, we've saved one of udev's best advantages until last, and it's a feature that will make a real difference to you.

The feature is called persistent device naming, and it works like this. Devices are normally named in the order in which they are connected. That's fine if you only have one of each type of device, but this is becoming less common. For example, many devices use the USB storage module to appear as disk drives. These include digital cameras, MP3 players, USB key disks and memory card readers, as well as external disk drives.

If you connect your camera, say, it will often be seen as `/dev/sda/`. If you then hook up your USB keyring it will appear



as `/dev/sdb/`. But if you connect the keyring first, that will appear as `/bedev/sda/`.

This makes dealing with these devices through `fstab` entries or automounters more complex than it needs to be. The situation is potentially worse with printers. I have two USB printers: a laser for text documents and an inkjet for printing photographs. One is `/dev/lp0` and the other `/dev/lp1`, but which gets which depends on which is detected first. If one of the printers is turned off when I boot, the devices can be reversed.

Udev fixes this nonsense by enabling you to specify your own device names for each product. Using a simple set of rules, udev will set the device name according to the identification data available from each device. It will also create symlinks, so a device can have more than one name. For example, a DVD-ROM drive could be accessed as any one of `/dev/hdc`, `/dev/cdrom` or `/dev/dvd`. So, how do we write our own udev rules?

## Making up the rules

The rules are contained in files in `/etc/udev/rules.d`. The default file is usually called `50-udev.rules`. Don't change this file as it could be overwritten when you upgrade udev. Instead, write your rules in a file called `10-udev.rules`. The low number ensures it will take priority over any definitions in the default file.

Each time a device is detected by the hotplug system, the files are read in order, line by line, until a match is found. This may be useful in more complex systems as you can set up specific rules followed by more general ones – but we're getting ahead of ourselves here.

The basic format of a rule is:

```
key1="value", key2="value", ... keyN="value", name="value",
symlink="value"
```

You must provide at least one key and a name. Extra keys are optional, but all must match for the rule to be applied. Symlinks are optional too. Here is an example of a udev rule, used to detect and name an iRiver MP3/Ogg player.

```
BUS="usb", KERNEL="sd[a-z]1", SYSFS{product}="iRiver
H300 Series", NAME="%k", SYMLINK="usb/iriver"
```

The first three items are keys used to identify the device. The **NAME**, as you would expect, defines the name to be used. `%k` is the name that the kernel would have given it, such as `/dev/sda1`, so this rule leaves the name unchanged, but sets a symlink to `/dev/usb/iriver`. The `/dev/usb` directory does not need to exist, as udev will create it when needed and delete it when the last device in there is removed. There is no standard convention to use `/dev/usb`; I just find it convenient to have all hotplugged USB devices appear here.

There are other substitutions that can be used in **NAME** and **SYMLINK**. After `%k`, `%n` is probably the most useful (it contains the kernel number of the device). If `%k` contains `sda3`, `%n` contains `3`. See the udev man page for a full list of substitutions.

## Configuring udev

The real work is done by the keys, of course, so how do we know what to use here? There are several keys available but the three most useful ones are **BUS**, **KERNEL** and **SYSFS**.

- **BUS** covers how the device is connected.
- **KERNEL** refers to the standard kernel identification of the device (as used by `devfs` or a static `/dev`).
- **SYSFS** keys use the information on each device that appears in the `/sys` directory. This directory was added for kernel 2.6 and is a virtual filesystem, somewhat like `/proc`, containing information on various devices.

You can browse through this filesystem to find information on a device, but udev provides a tool to make this task easier. The `udevinfo` command is used to extract information from `/sys`. You will need to be logged in as root to do most of this, so open

a terminal window and type `su` to become root. Now plug in your USB device, wait a few seconds for it to be detected, type `dmesg` and look for information on the device at the end of the output. It will look something like this:

```
usb 1-1: new high speed USB device using ehci_hcd and
address 6
scsi8 : SCSI emulation for USB Mass Storage devices
usb-storage: device found at 6
usb-storage: waiting for device to settle before scanning
Vendor: TOSHIBA Model: MK2004GAL Rev: JC10
Type: Direct-Access ANSI SCSI revision: 00
SCSI device sdd: 39063024 512-byte hdwr sectors (20000
MB)
sdd: assuming drive cache: write through
SCSI device sdd: 39063024 512-byte hdwr sectors (20000 MB)
sdd: assuming drive cache: write through
sdd: sdd1
Attached scsi disk sdd at scsi8, channel 0, id 0, lun 0
Attached scsi generic sg5 at scsi8, channel 0, id 0,
type 0
usb-storage: device scan complete
```

This tells us that the device has been detected as

`/dev/sdd` with a single partition at `/dev/sdd1`. It is the partition we're interested in here, although some mass-storage devices have no partitions (much like a floppy disk). Note that the device is called `sdd` because there are other pseudo-SCSI devices on this computer – a few SATA hard drives. If you have standard IDE drives and no other USB storage devices connected, it is more likely to be `/dev/sda`.

Now that we know how the device is named, we can use `udevinfo` to find the key information. First we need to find out where in `/sys` the information is contained, which we do with:

```
udevinfo -q path -n /dev/sdd1
```

This tells us that it is at `/block/sdd/sdd1` (this is relative to `/sys` so if you want to look at the information directly, look in `/sys/block/sdd/sdd1`. Now give this information to `udevinfo` to see the device details. You will get a lot of output, so enlarge your terminal window to full screen and pipe it through a pager like `less` or, my favourite, `most`.

```
udevinfo -a -p /block/sdd/sdd1 | less
```

You can combine the two stages with:

```
udevinfo -a -p $(udevinfo -q path -n /dev/sdd1) | less
```

## Picking the right keys

The key information is divided into sections: you will generally be looking for matches in one of the first few sections that appear. You cannot mix information from different directories in `/sys` – all keys used in a single rule must come from the same section of `udevinfo`'s output. Here are the relevant sections from the output of the above command:

```
looking at the device chain at '/sys/devices/
pci0000:00/0000:00:10.4/usb1/1-1/1-1:2.0/host8/
target8:0:0/8:0:0:0':
BUS="scsi"
[snip]
looking at the device chain at '/sys/devices
```





```

<< pci0000:00/0000:00:10.4/usb1/1-1/1-1:2.0/host8/
target8:0:0':
  BUS=""
  [snip]
looking at the device chain at '/sys/devices/
pci0000:00/0000:00:10.4/usb1/1-1/1-1:2.0/host8':
  BUS=""
  ID="host8"
  SYSFS{detach_state}="0"
looking at the device chain at '/sys/devices/
pci0000:00/0000:00:10.4/usb1/1-1/1-1:2.0':
  BUS="usb"
  [snip]
looking at the device chain at '/sys/devices/
pci0000:00/0000:00:10.4/usb1/1-1':
  BUS="usb"
  ID="1-1"
  SYSFS{bConfigurationValue}="2"
  SYSFS{bDeviceClass}="00"
  SYSFS{bDeviceProtocol}="00"
  SYSFS{bDeviceSubClass}="00"
  SYSFS{bMaxPower}=" 98mA"
  SYSFS{bNumConfigurations}=" 1"
  SYSFS{bNumInterfaces}=" 1"
  SYSFS{bcdDevice}="0100"
  SYSFS{bmAttributes}="c0"
  SYSFS{detach_state}="0"
  SYSFS{devnum}="6"
  SYSFS{idProduct}="3003"
  SYSFS{idVendor}="1006"
  SYSFS{manufacturer}="iRiver"
  SYSFS{maxchild}="0"
  SYSFS{product}="iRiver H300 Series"
  SYSFS{serial}="0123456789AB"
  SYSFS{speed}="480"
  SYSFS{version}=" 2.00"
    
```

The last section shown (several have been omitted) has information specific to the piece of hardware to be named. You should be looking for information that will uniquely identify your device. It's usually enough to use the model name or manufacturer code. Where these use a generic term, I have used an Epson printer that had a model string of 'USB printer'. You may need to use something like a serial number, but it is not normally necessary to be this specific unless you have more than one device of the same model.

I only have one MP3 player, so the product key should be distinctive enough. However... it isn't. The reason for this is that the entry for **sdd** also contains this key, and possibly an **sg** entry too, so we need a way to differentiate between the partition and the disk containing it. This is why the rule above has a **KERNEL** key too. This key uses a pattern to match the first partition on any SCSI disk -- USB storage devices are identified as SCSI disks. So **/dev/sdd1** matches this, but **/dev/sdd** does not.

You can use some standard pattern-matching characters in the keys: **\*** matches zero or more characters, **?** matches one or more characters and **[]** matches any one of the characters within the brackets. The above **KERNEL** match could just as well have been written as **sd?1**. The **BUS="usb"** part of the rule is not really necessary, but it does make things a little clearer when you have a number of rules. So the final rule is:

```

BUS="usb", KERNEL="sd[a-z]1", SYSFS{product}="iRiver
H300 Series", NAME="%k", SYMLINK="usb/iriver"
    
```

The code means: find the device on the USB bus that the kernel identifies as the first partition of a disk and has the product ID of iRiver H300 Series, give it its original name and create a symlink to this name from **/dev/usb/iriver**. You can use **udevtest** to test your rule without disconnecting and reconnecting the device. Give it the path to your device in **/sys** and it will report which device nodes and symlinks it will create.

You could put **usb/iriver** in the **NAME** field, but using symlinks means that the old-style kernel name is still there should anything need it. Equally, you could put your name in the **NAME** field and **%k** in the symlink. With udev, you control exactly how your devices are named.

Whichever way you do it, pick one and stick to it to avoid confusion later. You can create multiple symlinks to the same device but list them in the **SYMLINK** section, separated by spaces. Here's an entry for a DVD-ROM drive that covers all the bases with the old-style name, a devfs style name and symlinks to **/dev/cdrom** and **/dev/dvd**. Wherever software may look for a CD or DVD, it will find it.

```

KERNEL="hdc", NAME="%k", SYMLINK="dvd cdrom cdroms/
cdrom0"
    
```

Note that all names, whether in **KERNEL** keys or the **NAME** and **SYMLINK** assignments, are relative to **/dev/b/**.

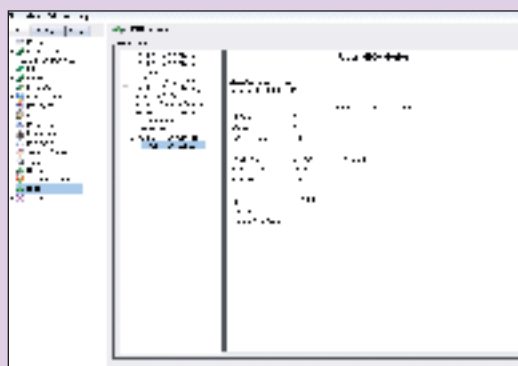
How about the situation with two printers? Once again, you can use **udevinfo** to find information unique to each. It is usually sufficient to use a model description, but if you have two devices of the same model, you can still distinguish between

## ALL MAPPED OUT

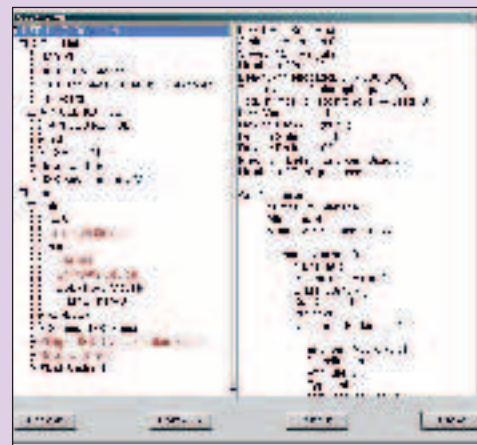
Forgotten the specs of a device? Here's how to run a query

You can use **udevinfo** to query devices in the udev database, but there are graphical alternatives. The information they provide isn't as graphical, nor is it in a format suitable for pasting directly into a rule. However, they are handy for an overview of what information is available on a device.

KDE users can use the KDE Info Centre to view information on various classes of devices. If you're working with USB devices, you can also get some of this information from **USBView**. This may not have been installed by default but should be on most distros' installation discs.



The GTK program **USBView** (above right) maps installed devices. You can also try KDE's Info Centre (far right).





them with the serial numbers. These are the rules I use for my laser and deskjet printers:

```
BUS="usb", SYSFS{product}="Samsung ML-1510_700",
NAME="%k", SYMLINK="printers/laser"
BUS="usb", SYSFS{product}="deskjet 5100", NAME="%k",
SYMLINK="printers/colour"
```

By using **NAME="%k"**, the printers still have their usual designation of `/dev/lp0` and `/dev/lp1`, but whichever way around these names are allocated, `/dev/printers/colour` and `/dev/printers/laser` point to the correct devices. Although these rules are for two USB printers, you could use similar rules if you have one parallel and one USB printer.

## Network name-calling

Udev is not limited to devices found in `/dev`. It also works with network devices, as they still appear in `/sys`. If you have two network cards in your computer, you need to know which is which. If they are different cards, you can get away with using the order in which you load the modules to determine which is **eth0** and which is **eth1**, but wouldn't it be easier if you could give them more meaningful names, and also work easily with more than one of the same type of card?

Information on your Ethernet device is contained in `/sys/class/net`.

```
# udevinfo -a -p /sys/class/net/eth0
looking at class device '/sys/class/net/eth0':
SYSFS{addr_len}="6"
SYSFS{address}="00:03:0d:06:52:b5"
SYSFS{broadcast}="ff:ff:ff:ff:ff:ff"
# udevinfo -a -p /sys/class/net/eth1
looking at class device '/sys/class/net/eth1':
SYSFS{addr_len}="6"
SYSFS{address}="00:09:5b:24:dc:fb"
SYSFS{broadcast}="ff:ff:ff:ff:ff:ff"
```

`SYSFS{address}` contains the MAC address of the network hardware. This is unique for every network card, so it provides a guaranteed way of distinguishing between them. To give these interfaces more meaningful and persistent names, use the following rules:

```
KERNEL="eth*", SYSFS{address}="00:03:0d:06:52:b5",
NAME="inet"
KERNEL="eth*", SYSFS{address}="00:09:5b:24:dc:fb",
NAME="lan"
```

You can't use symlinks here, because each interface can only have one name and they are not device files in `/dev`. These rules will only take effect when the interfaces are initialised. You can **rmmod** the modules and reload them with **modprobe** (provided they are modules and not built into the kernel), or reboot to reload them. Now the two interfaces are named **inet** and **lan**, far more useful for a box acting as a firewall or gateway, and it results in easier to read *iptables* rules too.

Memory card readers can cause difficulties if cards are inserted or removed while they are connected to the computer. This is particularly acute with multi-card readers.

Only the cards present when the device was connected will be registered. For empty slots the device for the disk will be created, say `/dev/sda`, but not for any partitions like `/dev/sda1`, so it's impossible to mount a card if you insert it after connecting the reader.

This is not too much of a problem with external readers, as you'd simply unplug it before inserting the card, but this is hardly practical if you have an internal card reader. Fortunately, udev provides a solution. Instead of **NAME=**, use **NAME{all\_partitions}=**. udev will now create 15 partition nodes as well as one for the disk. These rules work with an unbranded four slot multi-card reader:

```
BUS="scsi", KERNEL="sd?", SYSFS{model}="USB SD
Reader", NAME{all_partitions}="usb/sd"
BUS="scsi", KERNEL="sd?", SYSFS{model}="USB CF
Reader", NAME{all_partitions}="usb/cf"
BUS="scsi", KERNEL="sd?", SYSFS{model}="USB SM
Reader", NAME{all_partitions}="usb/sm"
BUS="scsi", KERNEL="sd?", SYSFS{model}="USB MS
Reader", NAME{all_partitions}="usb/ms"
```

Ensuring a removable device always has the same device node really comes into its own when combined with one of the systems of automatically mounting new devices, such as **supermount**.

By adding suitable lines to `/etc/fstab`, you can have a device mount when you connect it and unmount when you remove it, and the user doesn't have to do anything. If your kernel has **supermount** enabled you can have a device automount with a line like this in `/etc/fstab`.

```
none /mnt/camera supermount fs=auto,dev=/dev/usb/
camera,--,users,sync,noatime 0 0
```

This assumes that you have set up a udev rule to create `/dev/usb/camera` when you connect your digital camera and that the directory `/mnt/camera` exists. Note that not all digital cameras work as USB storage devices, so make sure yours does before trying to get this to work. You must use the **sync** option when creating *fstab* entries for this. The option ensures that data is written to the device immediately. Without it, you could copy files to the device, wait for the copy to finish, unplug the device and find that when you reconnect it the files are nowhere to be seen.

Some people don't like **supermount**. An alternative is **autofs**. If your distro uses this you need to add a line like:

```
/media /etc/autofs/auto.media
```

to `/etc/autofs/auto.master`. Then create the file `/etc/autofs/auto.media` and add a line like

```
camera -fstype=auto,users,sync,noatime,umask=0
:/dev/usb/camera
```

Unlike with **supermount**, you do not need to create the `/media/camera` or `/media` directories.


## Access privileges

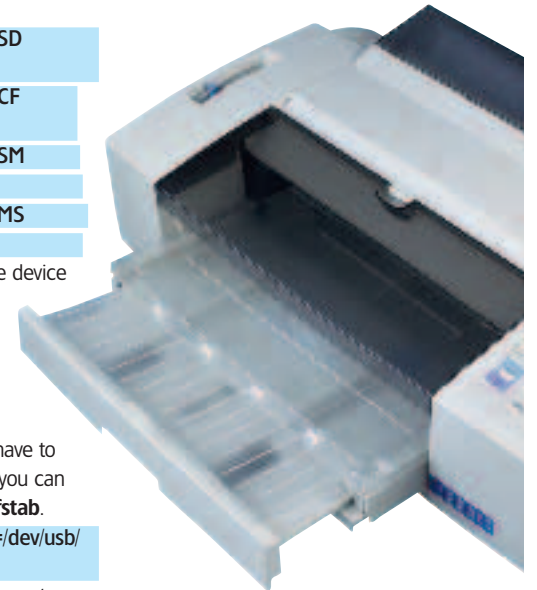
So far, we have only looked at udev's naming rules. But you can also control the permissions of each device node created by udev. The default file for this is `/etc/udev/permissions.d/50-udev.permissions` so put your own settings in `/etc/udev/permissions.d/10-udev.permissions`. You normally will not need to do anything here, but you can easily change the permissions or ownership of any device. The format is one device per line, giving owner, group and permissions, separated by colons. The device name can include pattern-matching characters. The following

```
printers*:root:print:0660
```

would restrict access to printers to only those users who are members of the **print** group.

There is a significant difference between **devfs** and **udev** in terms of module handling. The former will load kernel modules for new devices. Udev is purely about creating device nodes – module loading needs to be taken care of by hotplug scripts or by adding them to `/etc/modules` or `/etc/modprobe.conf`, depending on your distro.

We doubt you'll miss any **devfs** features – the fact that udev operates outside the kernel makes it much more user-friendly, and it supports symlinks and network devices too. Now that **devfs** is obsolete, udev is being shipped with almost every distro, and it makes sense for you to master writing udev rules. If you want to go beyond this tutorial and learn more, check out our More Information box, right. 



## MORE INFORMATION

The man pages for **udev** and **udevinfo** provide useful information and there are some useful websites, notably [www.reactivated.net/udevrules.php](http://www.reactivated.net/udevrules.php). Gentoo is one of the few distributions that does not include **udev** by default in its latest release (the 2005 release will default to **udev**, though). If you're a Gentoo user, you'll appreciate the excellent tutorial on switching over to **udev**, which is easy when explained properly, at [http://webpages.charter.net/decibelshelp/LinuxHelp\\_UDEVPrimer.html](http://webpages.charter.net/decibelshelp/LinuxHelp_UDEVPrimer.html).