udev

From **Wikipedia article**:

> *udev* is a device manager for the Linux kernel. As the successor of *devfsd* and *hotplug*, *udev* primarily manages device nodes in the `/dev` directory. At the same time, *udev* also handles all user space events raised while hardware devices are added into the system or removed from it, including firmware loading as required by certain devices.

Related articles

udisks

udev replaces the functionality of both *hotplug* and *hwdetect*.

udev loads kernel modules by utilizing coding parallelism to provide a potential performance advantage versus loading these modules serially. The modules are therefore loaded asynchronously. The inherent disadvantage of this method is that *udev* does not always load modules in the same order on each boot. If the machine has multiple block devices, this may manifest itself in the form of device nodes changing designations randomly. For example, if the machine has two hard drives, `/dev/sda` may randomly become `/dev/sdb`. See below for more info on this.

Contents

Installation

udev is now part of **systemd** and is installed by default. See `systemd-udevd.service(8) (https://jlk.fjfi.cvut.cz/arch/manpages/man/systemd-udevd.service.8)` for information.

A standalone fork is available as `eudev (https://aur.archlinux.org/packages/eudev/)`[AUR] and `eudev-git (https://aur.archlinux.org/packages/eudev-git/)`[AUR].

About udev rules

udev rules written by the administrator go in `/etc/udev/rules.d/`, their file name has to end with *.rules*. The *udev* rules shipped with various packages are found in `/usr/lib/udev/rules.d/`. If there are two files by the same name under `/usr/lib` and `/etc`, the ones in `/etc` take precedence.

Writing udev rules

Warning: To mount removable drives, do not call `mount` from *udev* rules. In case of FUSE filesystems, you will get `Transport endpoint not connected` errors. Instead, you could use **udisks** that handles automount correctly or to make mount work inside *udev* rules, copy `/usr/lib/systemd/system/systemd-udevd.service` to `/etc/systemd/system/systemd-udevd.service` and replace `MountFlags=slave` to `MountFlags=shared`.**[3] (http://unix.stackexchange.com/a/154318)** Keep in mind though that *udev* is not intended to invoke long-running processes.

- To learn how to write *udev* rules, see **Writing udev rules (http://www.reactivated.net/writing_udev_rules.html)**.
- To see an example *udev* rule, follow the **Examples (http://www.reactivated.net/writing_udev_rules.html#example-printer)** section of the above guide.

This is an example of a rule that places a symlink `/dev/video-cam1` when a webcamera is connected. First, we have found out that this camera is connected and has loaded with the device `/dev/video2`. The reason for writing this rule is that at the next boot the device might just as well show up under a different name like `/dev/video0`.

```
# udevadm info -a -p $(udevadm info -q path -n /dev/video2)
```

```
Udevadm info starts with the device specified by the devpath and then
walks up the chain of parent devices. It prints for every device
found, all possible attributes in the udev rules key format.
A rule to match, can be composed by the attributes of the device
and the attributes from one single parent device.

  looking at device '/devices/pci0000:00/0000:00:04.1/usb3/3-2/3-2:1.0/video4linux/video2':
    KERNEL=="video2"
    SUBSYSTEM=="video4linux"
    ...
  looking at parent device '/devices/pci0000:00/0000:00:04.1/usb3/3-2/3-2:1.0':
    KERNELS=="3-2:1.0"
    SUBSYSTEMS=="usb"
    ...
  looking at parent device '/devices/pci0000:00/0000:00:04.1/usb3/3-2':
    KERNELS=="3-2"
    SUBSYSTEMS=="usb"
    ...
    ATTRS{idVendor}=="05a9"
    ...
    ATTRS{manufacturer}=="OmniVision Technologies, Inc."
    ATTRS{removable}=="unknown"
    ATTRS{idProduct}=="4519"
    ATTRS{bDeviceClass}=="00"
    ATTRS{product}=="USB Camera"
    ...
```

From the video4linux device we use `KERNEL=="video2"` and `SUBSYSTEM=="video4linux"`, then we match the webcam using vendor and product ID's from the usb parent `SUBSYSTEMS=="usb"`, `ATTRS{idVendor}=="05a9"` and `ATTRS{idProduct}=="4519"`.

```
/etc/udev/rules.d/83-webcam.rules
```

```
KERNEL=="video[0-9]*", SUBSYSTEM=="video4linux", SUBSYSTEMS=="usb", ATTRS{idVendor}=="05a9", ATTRS{idProduct}=="4519", SYMLINK+="video-cam1"
```

In the example above we create a symlink using `SYMLINK+="video-cam1"` but we could easily set user `OWNER="john"` or group using `GROUP="video"` or set the permissions using `MODE="0660"`. However, if you intend to write a rule to do something when a device is being removed, be aware that device attributes may not be accessible. In this case, you will have to work with preset device **environment variables**. To monitor those environment variables, execute the following command while unplugging your device:

```
# udevadm monitor --environment --udev
```

In this command's output, you will see value pairs such as `ID_VENDOR_ID` and `ID_MODEL_ID`, which match your previously used attributes "idVendor" and "idProduct". A rule that uses device environment variables may look like this:

/etc/udev/rules.d/83-webcam-removed.rules

```
ACTION=="remove", SUBSYSTEM=="usb", ENV{ID_VENDOR_ID}=="05a9", ENV{ID_MODEL_ID}=="4519", RUN+="/path/to/your/script"
```

List attributes of a device

To get a list of all of the attributes of a device you can use to write rules, run this command:

```
# udevadm info -a -n [device name]
```

Replace `[device name]` with the device present in the system, such as `/dev/sda` or `/dev/ttyUSB0`.

If you do not know the device name you can also list all attributes of a specific system path:

```
# udevadm info -a -p /sys/class/backlight/acpi_video0
```

Testing rules before loading

```
# udevadm test $(udevadm info -q path -n [device name]) 2>&1
```

This will not perform all actions in your new rules but it will however process symlink rules on existing devices which might come in handy if you are unable to load them otherwise. You can also directly provide the path to the device you want to test the *udev* rule for:

```
# udevadm test /sys/class/backlight/acpi_video0/
```

Loading new rules

udev automatically detects changes to rules files, so changes take effect immediately without requiring *udev* to be restarted. However, the rules are not re-triggered automatically on already existing devices. Hot-pluggable devices, such as USB devices, will probably have to

be reconnected for the new rules to take effect, or at least unloading and reloading the ohci-hcd and ehci-hcd kernel modules and thereby reloading all USB drivers.

If rules fail to reload automatically:

```
# udevadm control --reload
```

To manually force *udev* to trigger your rules:

```
# udevadm trigger
```

Udisks

See **Udisks**.

Tips and tricks

Accessing firmware programmers and USB virtual comm devices

The following rule will allow users in the `users` group to access the **USBtinyISP (http://www.ladyada.net/make/usbtinyisp/)** USB programmer for AVR microcontrollers.

```
/etc/udev/rules.d/50-usbtinyisp.rules
```

```
SUBSYSTEMS=="usb", ATTRS{idVendor}=="1781", ATTRS{idProduct}=="0c9f", GROUP="users", MODE="0660"
SUBSYSTEMS=="usb", ATTRS{idVendor}=="16c0", ATTRS{idProduct}=="0479", GROUP="users", MODE="0660"
```

Use *lsusb* to get the vendor and product IDs for other devices.

Execute on VGA cable plug in

Create the rule `/etc/udev/rules.d/95-monitor-hotplug.rules` with the following content to launch **arandr (https://www.archlinux.org/packages/?name=arandr)** on plug in of a VGA monitor cable:

```
KERNEL=="card0", SUBSYSTEM=="drm", ENV{DISPLAY}=":0", ENV{XAUTHORITY}="/home/username/.Xauthority", RUN+="/usr/bin/arandr"
```

Some display managers store the .Xauthority outside the user home directory. You will need to update the ENV{XAUTHORITY} accordingly. As an example **GNOME Display Manager** looks as follows:

```
$ printenv XAUTHORITY
```

```
/run/user/1000/gdm/Xauthority
```

Detect new eSATA drives

If your eSATA drive is not detected when you plug it in, there are a few things you can try. You can reboot with the eSATA plugged in. Or you could try:

```
# echo 0 0 0 | tee /sys/class/scsi_host/host*/scan
```

Or you could install **scsiadd (https://aur.archlinux.org/packages/scsiadd/)**[AUR] (from the AUR) and try:

```
# scsiadd -s
```

Hopefully, your drive is now in `/dev`. If it is not, you could try the above commands while running:

```
# udevadm monitor
```

to see if anything is actually happening.

Mark internal SATA ports as eSATA

If you connected a eSATA bay or an other eSATA adapter the system will still recognize this disk as an internal SATA drive. **GNOME** and **KDE** will ask you for your root password all the time. The following rule will mark the specified SATA-Port as an external eSATA-Port.

With that, a normal GNOME user can connect their eSATA drives to that port like a USB drive, without any root password and so on.

/etc/udev/rules.d/10-esata.rules

```
DEVPATH=="/devices/pci0000:00/0000:00:1f.2/host4/*", ENV{UDISKS_SYSTEM}="0"
```

Note: The `DEVPATH` can be found after connection the eSATA drive with the following commands (replace `sdb` accordingly):

```
# udevadm info -q path -n /dev/sdb
/devices/pci0000:00/0000:00:1f.2/host4/target4:0:0/4:0:0:0/block/sdb
```

```
# find /sys/devices/ -name sdb
/sys/devices/pci0000:00/0000:00:1f.2/host4/target4:0:0/4:0:0:0/block/sdb
```

Setting static device names

Because *udev* loads all modules asynchronously, they are initialized in a different order. This can result in devices randomly switching names. A *udev* rule can be added to use static device names.

See also **Persistent block device naming** for block devices and **Network configuration#Change device name** for network devices.

Video devices

For setting up the webcam in the first place, refer to **Webcam configuration**.

Using multiple webcams, useful for example with **motion (https://aur.archlinux.org/packages/motion/)**[AUR] (software motion detector which grabs images from video4linux devices and/or from webcams), will assign video devices as /dev/video0..n randomly on boot. The recommended solution is to create symlinks using an *udev* rule (as in the example in **#Writing udev rules**):

/etc/udev/rules.d/83-webcam.rules

```
KERNEL=="video[0-9]*", SUBSYSTEM=="video4linux", SUBSYSTEMS=="usb", ATTRS{idVendor}=="05a9", ATTRS{idProduct}=="4519", SYMLINK+="video-cam1"
KERNEL=="video[0-9]*", SUBSYSTEM=="video4linux", SUBSYSTEMS=="usb", ATTRS{idVendor}=="046d", ATTRS{idProduct}=="08f6", SYMLINK+="video-cam2"
KERNEL=="video[0-9]*", SUBSYSTEM=="video4linux", SUBSYSTEMS=="usb", ATTRS{idVendor}=="046d", ATTRS{idProduct}=="0840", SYMLINK+="video-cam3"
```

Note: Using names other than `/dev/video*` will break preloading of `v4l1compat.so` and perhaps `v4l2convert.so`

Printers

If you use multiple printers, `/dev/lp[0-9]` devices will be assigned randomly on boot, which will break e.g. **CUPS** configuration.

You can create following rule, which will create symlinks under `/dev/lp/by-id` and `/dev/lp/by-path`, similar to **Persistent block device naming** scheme:

/etc/udev/rules.d/60-persistent-printer.rules

```
ACTION=="remove", GOTO="persistent_printer_end"

# This should not be necessary
#KERNEL!="lp*", GOTO="persistent_printer_end"

SUBSYSTEMS=="usb", IMPORT{builtin}="usb_id"
ENV{ID_TYPE}!="printer", GOTO="persistent_printer_end"

ENV{ID_SERIAL}=="?*", SYMLINK+="lp/by-id/$env{ID_BUS}-$env{ID_SERIAL}"

IMPORT{builtin}="path_id"
ENV{ID_PATH}=="?*", SYMLINK+="lp/by-path/$env{ID_PATH}"

LABEL="persistent_printer_end"
```

Waking from suspend with USB device

First, find vendor and product ID of your device, for example

```
# lsusb | grep Logitech
```

```
Bus 007 Device 002: ID 046d:c52b Logitech, Inc. Unifying Receiver
```

Find where the device is connected to using:

```
# grep c52b /sys/bus/usb/devices/*/idProduct
```

```
/sys/bus/usb/devices/1-1.1.1.4/idProduct:c52b
```

Now change the `power/wakeup` attribute of the device and the USB controller it is connected to, using the following rule:

/etc/udev/rules.d/50-wake-on-device.rules

```
ACTION=="add", SUBSYSTEM=="usb", DRIVER=="usb", ATTRS{idVendor}=="046d", ATTRS{idProduct}=="c52b", ATTR{power/wakeup}="enabled", ATTR{driver/1-1.1.4/power/wakeup}="enabled"
```

Note: By default, the USB host controllers are all enabled by default for wakeup. The status can be checked using `# cat /proc/acpi/wakeup`.

Triggering events

It can be useful to trigger various *udev* events. For example, you might want to simulate a USB device disconnect on a remote machine. In such cases, use `udevadm trigger`:

```
# udevadm trigger -v -t subsystems -c remove -s usb -a "idVendor=abcd"
```

This command will trigger a USB remove event on all USB devices with vendor ID `abcd`.

Triggering desktop notifications from a udev rule

Invoking an external script containing calls to `notify-send` via *udev* **can sometimes be challenging (https://bbs.archlinux.org/viewtopic.php?id=212364)** since the notification(s) never display on the Desktop. Here is an example of what commands and environmental variables need to be included in which files for `notify-send` to successfully be executed

from a *udev* rule. NOTE: a number of variables are hardcoded in this example, thus consider making them portable (i.e., $USER rather than user's shortname) once you understand the example.

1) The following *udev* rule executes a script that plays a notification sound and sends a desktop notification when screen brightness is changed according to power state on a laptop. Create the file:

```
/etc/udev/rules.d/99-backlight_notification.rules
```

```
Play a notification sound and send a desktop notification when screen brightness is changed according to power state on a laptop (a second ''udev''
rule actually changes the screen brightness)
# Rule for when switching to battery
ACTION=="change", SUBSYSTEM=="power_supply", ATTR{type}=="Mains", ATTR{online}=="0", ENV{DISPLAY}=":0", ENV{XAUTHORITY}="/home/USERNAME/.Xauthority"
RUN+="/usr/bin/su USERNAME_TO_RUN_SCRIPT_AS -c /usr/local/bin/brightness_notification.sh"
# Rule for when switching to AC
ACTION=="change", SUBSYSTEM=="power_supply", ATTR{type}=="Mains", ATTR{online}=="1", ENV{DISPLAY}=":0", ENV{XAUTHORITY}="/home/USERNAME/.Xauthority"
RUN+="/usr/bin/su USERNAME_TO_RUN_SCRIPT_AS -c /usr/local/bin/brightness_notification.sh"
```

Note: 1) `USERNAME_TO_RUN_SCRIPT_AS` and `USERNAME` need to be changed to that of the shortname for the user of the graphical session where the notification will be displayed and 2) the script needs to be executed with `/usr/bin/su`, which will place its ownership under the user of the graphical session (rather than root/the system) where the notification will be displayed.

2) Contents of the executable script to be run on trigger of the *udev* rule:

```
/usr/local/bin/brightness_notification.sh
```

```
#!/usr/bin/env bash

export XAUTHORITY=/home/USERNAME_TO_RUN_SCRIPT_AS/.Xauthority
export DISPLAY=:0
export DBUS_SESSION_BUS_ADDRESS="unix:path=/run/user/UID_OF_USER_TO_RUN_SCRIPT_AS/bus"

/usr/bin/sudo -u USERNAME_TO_RUN_SCRIPT_AS /usr/bin/paplay --server /run/user/UID_OF_USER_TO_RUN_SCRIPT_AS/pulse/native /home/USERNAME/.i3/sounds/Cl
ick1.wav > /dev/null 2>&1

/usr/bin/notify-send -i /usr/share/icons/gnome/256x256/status/battery-full-charging.png 'Changing Power States' --expire-time=4000
```

Note: 1) `USERNAME_TO_RUN_SCRIPT_AS`, `UID_OF_USER_TO_RUN_SCRIPT_AS` and `USERNAME` needs to be changed to that of the shortname for the user and user's UID of the graphical session where the notification will be displayed; 2) `/usr/bin/sudo` is needed when playing audio via pulseaudio; and, 3) three environmental variables (i.e., `XAUTHORITY`, `DISPLAY` and `DBUS_SESSION_BUS_ADDRESS`) for the user of the graphical session where the notification will be displayed need to be defined and exported.

Warning: The `XAUTHORITY`, `DISPLAY` and `DBUS_SESSION_BUS_ADDRESS` environment variables must be defined correctly.

3) Load/reload the new *udev* rule (see above) and test it by unplugging the power supply to the laptop.

Tip: See also **xpub (https://github.com/Ventto/xpub)** as a method for getting the user's display environment variables and exporting the last into *udev* rules via `IMPORT` key.

Troubleshooting

Blacklisting modules

In rare cases, *udev* can make mistakes and load the wrong modules. To prevent it from doing this, you can blacklist modules. Once blacklisted, *udev* will never load that module. See **blacklisting**. Not at boot-time *or* later on when a hotplug event is received (eg, you plug in your USB flash drive).

Debug output

To get hardware debug info, use the **kernel parameter** `udev.log-priority=debug`. Alternatively you can set

```
/etc/udev/udev.conf
```

```
udev_log="debug"
```

This option can also be compiled into your initramfs by adding the config file to your `FILES` array

```
/etc/mkinitcpio.conf
```

```
FILES="... /etc/udev/udev.conf"
```

and then rebuilding the initramfs with

```
# mkinitcpio -p linux
```

udevd hangs at boot

After migrating to LDAP or updating an LDAP-backed system *udevd* can hang at boot at the message "Starting UDev Daemon". This is usually caused by *udevd* trying to look up a name from LDAP but failing, because the network is not up yet. The solution is to ensure that all system group names are present locally.

Extract the group names referenced in *udev* rules and the group names actually present on the system:

```
# fgrep -r GROUP /etc/udev/rules.d/ /usr/lib/udev/rules.d | perl -nle '/GROUP\s*=\s*"(.*?)"/ && print $1;' | sort | uniq > udev_groups
# cut -f1 -d: /etc/gshadow /etc/group | sort | uniq > present_groups
```

To see the differences, do a side-by-side diff:

```
# diff -y present_groups udev_groups
...
network                                                         <
nobody                                                          <
ntp                                                             <
optical                                                           optical
power                                                           | pcscd
rfkill                                                          <
root                                                              root
scanner                                                           scanner
smmsp                                                           <
storage                                                           storage
...
```

In this case, the `pcscd` group is for some reason not present in the system. **Add the missing groups**. Also, make sure that local resources are looked up before resorting to LDAP. `/etc/nsswitch.conf` should contain the following line:

```
group: files ldap
```

BusLogic devices can be broken and will cause a freeze during startup

This is a kernel bug and no fix has been provided yet.

Some devices, that should be treated as removable, are not

You need to create a custom *udev* rule for that particular device. To get definitive information of the device you can use either `ID_SERIAL` or `ID_SERIAL_SHORT` (remember to change `/dev/sdb` if needed):

```
$ udevadm info /dev/sdb | grep ID_SERIAL
```

Then we create a rule in `/etc/udev/rules.d/` and set variables for either udisks or udisks2.

For udisks, set `UDISKS_SYSTEM_INTERNAL="0"`, which will mark the device as "removable" and thus "eligible for automounting". See **udisks(7) (http://manpages.ubuntu.com/manpages/trusty/man7/udisks.7.html)** for details.

```
/etc/udev/rules.d/50-external-myhomedisk.rules
```

```
ENV{ID_SERIAL_SHORT}=="serial_number", ENV{UDISKS_SYSTEM_INTERNAL}="0"
```

For udisks2, set `UDISKS_AUTO="1"` to mark the device for automounting and `UDISKS_SYSTEM="0"` to mark the device as "removable". See **udisks(8) (https://jlk.fjfi.cvut.cz/arch/manpages/man/udisks.8)** for details.

```
/etc/udev/rules.d/99-removable.rules
```

```
ENV{ID_SERIAL_SHORT}=="serial_number", ENV{UDISKS_AUTO}="1", ENV{UDISKS_SYSTEM}="0"
```

Remember to reload *udev* rules with `udevadm control --reload`. Next time you plug your device in, it will be treated as an external drive.

Sound problems with some modules not loaded automatically

Some users have traced this problem to old entries in `/etc/modprobe.d/sound.conf`. Try cleaning that file out and trying again.

Note: Since `udev>=171`, the OSS emulation modules (`snd_seq_oss`, `snd_pcm_oss`, `snd_mixer_oss`) are not automatically loaded by default.

IDE CD/DVD-drive support

Starting with version 170, *udev* does not support CD-ROM/DVD-ROM drives that are loaded as traditional IDE drives with the `ide_cd_mod` module and show up as `/dev/hd*`. The drive remains usable for tools which access the hardware directly, like *cdparanoia*, but is invisible for higher userspace programs, like KDE.

A cause for the loading of the ide_cd_mod module prior to others, like sr_mod, could be e.g. that you have for some reason the module piix loaded with your **initramfs**. In that case you can just replace it with ata_piix in your `/etc/mkinitcpio.conf`.

Optical drives have group ID set to "disk"

If the group ID of your optical drive is set to `disk` and you want to have it set to `optical`, you have to create a custom *udev* rule:

```
/etc/udev/rules.d
```

```
# permissions for IDE CD devices
SUBSYSTEMS=="ide", KERNEL=="hd[a-z]", ATTR{removable}=="1", ATTRS{media}=="cdrom*", GROUP="optical"

# permissions for SCSI CD devices
SUBSYSTEMS=="scsi", KERNEL=="s[rg][0-9]*", ATTRS{type}=="5", GROUP="optical"
```

See also

- **udev home page (https://www.kernel.org/pub/linux/utils/kernel/hotplug/udev/udev.html)**

- **An Introduction to udev (https://www.linux.com/news/hardware/peripherals/180950-udev)**
- **udev mailing list information (http://vger.kernel.org/vger-lists.html#linux-hotplug)**
- **Scripting with udev (http://jasonwryan.com/blog/2014/01/20/udev/)**
- **Writing udev rules (http://www.reactivated.net/writing_udev_rules.html)**
- **Device and Module Handling on an LFS System (http://www.linuxfromscratch.org/lfs/view/6.1/chapter07/udev.html)**
- **Running GUI or accessing display variables from udev rules (https://github.com/Ventto/xpub)**

Retrieved from "https://wiki.archlinux.org/index.php?title=Udev&oldid=508680"
