Ecrire des règles UDEV

- Objet : Ecrire des règles UDEV
- Niveau requis :
  débutant, avisé
- Commentaires : Fix Me!
- Débutant, à savoir : Utiliser GNU/Linux en ligne de commande, tout commence là !
- Suivi :
  à-tester
  - Création par bendia le 27/03/2013
  - Testé par .... le ....
- Commentaires sur le forum : C'est ici[1)]

Nota : Contributeurs, les Fix Me! sont là pour vous aider, supprimez-les une fois le problème corrigé ou le champ rempli !

Article original dans le wiki du forum slackware-fr Traduit de l'anglais par Mushroom.

Nota

La version originale la plus récente de ce document peut être trouvée ici :

- http://www.reactivated.net/writing_udev_rules.html

Le numéro de version de cette traduction indique, pour ce qui est à gauche du tiret, la dernière version originale avec laquelle celle-ci a été synchronisée et, à droite du tiret, le nombre de révisions qu'elle a subies depuis cette dernière synchronisation.

Ce tutoriel étant loin de lui être d'un usage quotidien (même s'il est très intéressant), il se peut que cette feignasse de traducteur laisse l'original prendre quelques longueurs d'avance sur la traduction. Si vous constatez qu'une resynchronisation des versions serait souhaitable (ou pour toute autre remarque sur cette traduction), n'hésitez pas à me contacter. – Seb.

Introduction

Sur les noyaux Linux 2.6 et supérieurs, udev est conçu pour fournir une solution en espace utilisateur afin d'obtenir un répertoire /dev dynamique, grâce à une désignation constante des périphériques. La précédente implémentation de /dev, devfs, est à présent obsolète, et udev fait figure de sucesseur. « udev contre devfs » est un sujet de discussion sensible – vous devriez lire ce document (en) avant de faire les comparaisons.

Avec les années les choses pour lesquelles vous pouviez utiliser des règles udev ont changé, de

même que la flexibilité des règles elles-mêmes. Sur un système moderne, udev fournit une désignation constante toute prête pour certains types de périphériques, éliminant le besoin de règles personnalisées pour ces périphériques. Quoi qu'il en soit, certains utilisateurs réclameront toujours un niveau de personnalisation supplémentaire.

Ce document part du principe que vous avez udev installé et fonctionnel avec la configuration par défaut. Cela est généralement pris en charge par votre distribution Linux.

Ce document ne couvre pas chaque détail singulier de l'écriture de règles, mais vise à introduire tous les concepts principaux. Les détails plus fins peuvent être trouvés dans la page de manuel d'udev.

Ce document utilise des exemples divers (beaucoup d'entre eux sont entièrement fictifs) pour illustrer les idées et les concepts. Toute la syntaxe n'est pas explicitement décrite dans le texte les accompagnant, veillez-donc à regarder les exemples afin d'acquérir une complète compréhension.

Les concepts

Terminologie : devfs, sysfs, noeuds, etc.

Une introduction de base uniquement, qui peut ne pas être totalement exacte.

Sur les systèmes typiques basés sur Linux, le répertoire /dev est utilisé pour stocker des noeuds de périphériques, qui ressemblent à des fichiers et référencent certains périphériques au sein du système. Chaque noeud pointe sur une partie du système (un périphérique), qui peut exister ou non. Les applications de l'espace utilisateur peuvent utiliser ces noeuds de périphérique pour s'interfacer avec le matériel des systèmes. Par exemple le serveur X va « écouter » /dev/input/mice, de sorte qu'il pourra se baser sur les mouvements de la souris de l'utilisateur pour déplacer le curseur virtuel de la souris.

À l'origine, les répertoires /dev était juste peuplés avec tous les périphériques qui auraient pu potentiellement apparaître dans le système. Les répertoires /dev étaient ainsi traditionnellement très gros à cause de cela. devfs vint alors avec une approche plus gérable (notamment, il peuplait /dev uniquement avec le matériel connecté au système), ainsi que quelques autres fonctionnalités. Mais ce système souffrait à l'évidence de problèmes qui ne pouvaient être facilement résolus.

udev est la « nouvelle » manière de gérer les répertoires /dev, conçue pour se débarrasser de certains problèmes posés par les précédentes implémentations de /dev/, et pour fournir une voie plus sûre pour la suite. Afin de créer et de nommer les noeuds de périphériques de /dev qui correspondent à des périphériques présents dans le système, udev s'appuie sur la mise en adéquation des informations fournies par sysfs avec les règles fournies par l'utilisateur. Ce document vise à détailler le processus d'écriture des règles, une des seules tâches relatives à udev qui doivent être (optionnellement) effectuées par l'utilisateur.

sysfs est un nouveau système de fichiers des noyaux 2.6. Il est géré par le noyau, et exporte des informations basiques sur les périphériques actuellement branchés à votre système. udev peut ainsi utiliser ces informations pour créer les noeuds de périphériques correspondant à votre matériel. sysfs est monté sur /sys et il est possible d'en explorer le contenu. Vous pouvez si vous le voulez inspecter quelques un des fichiers stockés ici avant de vous attaquer à udev. Tout au long de ce document, j'utiliserai les termes « /sys » et « sysfs » de manière interchangeable.

Pourquoi ?

Les règles udev sont flexibles est très puissantes. Voici quelques unes des choses pour la réalisation desquelles vous pouvez utiliser des règles :

- rebaptiser un noeud de périphérique ;
- fournir un nom alternatif/constant à un noeud de périphérique en créant un lien symbolique vers le noeud de périphérique par défaut ;
- baptiser un noeud de périphérique en se basant sur la sortie d'un programme ;
- changer les permissions et les propriétés d'un noeud de périphérique ;
- lancer un script quand un noeud de périphérique est créé ou détruit (généralement quand un périphérique est connecté ou débranché) ;
- renommer des interfaces réseau.

Écrire des règles udev n'est cependant pas une solution aux problèmes où aucun noeud de périphérique pour votre périphérique particulier n'existe. Même si aucune règle ne correspond, udev créera le noeud de périphérique avec le nom par défaut fourni par le noyau.

Avoir des noeuds de périphériques nommés de manière constante a plusieurs avantages. Admettons que vous ayez deux périphériques de stockage USB : une caméra numérique et une clé USB. Ces périphériques se voient généralement attribuer les noeuds de périphérique /dev/sda et /dev/sdb, mais l'attribution exacte dépend de l'ordre dans lequel ils ont été connectés à l'origine. Cela peut poser problème à certains utilisateurs, qui pourrait en tirer un grand profit si chaque périphérique pouvaient être nommé de manière constante à chaque fois, par exemple /dev/camera et /dev/flashdisk.

Les schèmes de dénominations constantes intégrés

udev fournit une dénomination constante prête à l'emploi pour certains périphériques. C'est une fonctionnalité très utile, et dans beaucoup de cas cela signifie que votre voyage se termine ici : vous n'avez à écrire aucune règle.

udev fournit une dénomination constante prête à l'emploi pour les périphériques de stockage du répertoire /dev/disk. Pour voir les noms constants qui ont été créés pour votre matériel de stockage, vous pouvez utiliser la commande suivante :

```
ls -lR /dev/disk
```

Cela fonctionne pour tous les types de stockage. Par exemple, udev a créé /dev/disk/by-id/scsi-SATA_ST3120827AS_4MS1NDXZ-part3, qui est un lien symbolique nommé de manière constante pour ma racine. udev crée également /dev/disk/by-id/usb-Prolific_Technology_Inc._USB_Mass_Storage_Device-part1 quand je branche ma clé USB, ce qui est aussi un nom constant.

Écrire des règles

Les fichiers de règles et leur sémantique

Lorsqu'il décide comment nommer un périphérique et quelles actions additionnelles doivent être effectuées, udev lit une série de fichiers de règles. Ces fichiers sont conservés dans le répertoire /etc/udev/rules.d, et doivent tous porter le suffixe .rules.

Les règles par défaut d'udev sont stockées dans /etc/udev/rules.d/50-udev.rules. Vous pouvez trouver intéressant de parcourir ce fichier – il contient quelques exemples, ainsi que quelques règles par défaut testant une trame /dev dans le style de devfs. Quoi qu'il en soit, vous ne devriez pas écrire de règles directement dans ce fichier.

Les fichiers dans /etc/udev/rules.d/ sont évalués dans l'ordre lexical, et dans certaines circonstances, l'ordre dans lequel les règles sont évaluées est important. En général, vous voulez que vos propres règles soient évaluées avant celles par défaut, je vous suggère donc de créer un fichier /etc/udev/rules.d/10-local.rules et d'y écrire toutes vos règles.

Dans les fichiers de règles, les lignes commençant par « # » sont traitées comme des commentaires. Toutes les autres lignes non-blanches sont des règles. Les règles ne peuvent s'étendre sur plusieurs lignes.

Un périphérique peut correspondre à plus d'une règle. Cela a des avantages pratiques, par exemple nous pouvont écrire deux règles qui correspondent au même périphérique, où chacune fournira son propre nom alternatif pour le périphérique. Les deux noms alternatifs seront créés, même si les règles sont dans des fichiers séparés. Il est important de comprendre qu'udev ne s'arrêtera pas d'analyser lorsqu'il trouvera une règle correspondant, il continuera à chercher et essaiera d'appliquer toutes les règles qu'il rencontrera au sujet de ce périphérique.

La syntaxe des règles

Chaque règle est construite à partir de séries de paires clé-valeur, qui sont séparées par des virgules. Les clés de correspondance sont les conditions utilisées pour identifier le périphérique sur lequel la règle agit. Quand toutes les clés de correspondance d'une règle correspondent au périphérique en train d'être traité, la règle est appliquée et les actions des clés d'assignation sont invoquées. Chaque règle doit être constituée au moins d'une clé de correspondance et d'une clé d'assignation.

Voici un exemple de règle illustrant ce qui précède :

```
  KERNEL=="hdb", NAME="my_spare_disk"
```

La règle ci-dessus contient une clé de correspondance (KERNEL) et une clé d'assignation (NAME). La sémantique de ces clés et leurs propriétés seront détaillées plus tard. Il est important de noter que la clé de correspondance est reliée à sa valeur par l'opérateur d'égalité (==), alors que la clé d'assignation est reliée à sa valeur par l'opérateur d'assignation (=).

Observez bien qu'udev ne supporte aucune forme de continuation de ligne. N'insérez aucune rupture de ligne (« \ ») dans vos règles, car cela amènera udev à considérer votre règle unique comme plusieurs règles et ne donnera donc pas le résultat escompté.

Les règles de base

udev fournit plusieurs clés de correspondance qui peuvent être utilisées pour écrire des règles

correspondant très précisément aux périphériques. Certaines des plus communes sont introduites ci-dessous, les autres seront introduites plus loin dans le document. Pour une liste complète, référez-vous à la page de manuel d'udev.

- KERNEL - recherche des correspondances avec le nom donné par le noyau pour le périphérique ;
- SUBSYSTEM - recherche des correspondances avec le sous-système du périphérique ;
- DRIVER - recherche des correspondances avec le nom du pilote prenant en charge le périphérique.

Une fois que vous avez utilisé une série de clés de correspondance pour correspondre à un périphérique précis, udev vous donne un contrôle fin sur ce qui doit ce passer ensuite, au travers d'un jeu complet de clés d'assignation. Pour une liste complète des clés d'assignation disponibles, voyez la page de manuel d'udev. Les clés d'assignation les plus basiques sont introduites ci-dessous. Les autres seront introduite plus loin dans ce document.

- NAME - le nom qui doit être utilisé pour le noeud de périphérique ;
- SYMLINK - une liste des liens symboliques qui font office de noms alternatifs pour le noeud de périphérique.

Comme sous-entendu ci-dessus, udev ne crée qu'un seul vrai noeud de périphérique par périphérique. Si vous souhaitez fournir des noms alternatifs pour ce noeud de périphérique, vous utilisez la fonctionnalité des liens symboliques. Avec l'assignation SIMLINK, vous maintenez en fait une liste de liens symboliques, qui pointeront chacun sur le vrai noeud de périphérique. Pour manipuler ces liens, nous introduisons un nouvel opérateur qui ajoute aux listes : +=. Vous pouvez ajouter plusieurs liens symboliques à la liste à partir d'une règle quelconque en séparant chacun d'eux avec un espace.

```
KERNEL=="hdb", NAME="my_spare_disk"
```

La règle ci-dessus dit : « recherche des correspondances avec un périphérique qui a été nommé hdb par le noyau, et au lieu de l'appeler hdb, appelle le noeud de périphérique mon_espace_disque ». Le noeud de périphérique apparaîtra alors en tant que /dev/mon_espace_disque.

```
KERNEL=="hdb", DRIVER=="ide-disk", SYMLINK+="sparedisk"
```

La règle ci-dessus dit : « recherche des correspondances avec un périphérique qui a été nommé hdb par le noyau ET dont le pilote est ide-disk, nomme le noeud de périphérique avec le nom par défaut et crée un lien symbolique vers celui-ci nommé sparedisk ». Notez que nous ne spécifions aucun nom pour le noeud de périphérique, udev utilise donc celui par défaut. Afin de préserver la trame standard de /dev, vos propres règles laisseront généralement le NOM de côté mais créeront des LIENS SYMBOLIQUES et/ou feront d'autres types d'assignation.

```
KERNEL=="hdc", SYMLINK+="cdrom cdrom0"
```

La règle ci-dessus est probablement plus commune aux types de règles que vous pourriez écrire. Elle crée deux liens symboliques en tant que /dev/cdrom et /dev/cdrom0, chacun pointant sur /dev/hdc. Une fois encore, pas d'assignation de NOM, le nom par défaut du noyau (hdc) est donc utilisé.

Rechercher des correspondances dans les attributs de sysfs

Les clés de correspondance introduites jusqu'ici ne fournissent que des capacités d'évaluation limitées. Dans la pratique, nous réclamons un contrôle plus fin : nous voulons identifier des périphériques en nous basant sur des propriétés avancées, comme les codes du vendeur, les numéros de produit exacts, les numéros de série, la capacité de stockage, le nombre de partitions, etc.

Beaucoup de pilotes exportent de telles informations dans sysfs, et udev nous permet d'incorporer des recherches de correspondances avec sysfs dans nos règles, ce en utilisant la clé ATTR avec une syntaxe un peu différente.

Voici un exemple de règle qui cherche des correspondances avec un un seul attribut de sysfs. Plus de détails, qui vous aideront à écrire des règles basées sur les attributs de sysfs, seront fournis plus loin dans ce document.

```
  SUBSYSTEM=="block", ATTR{size}=="234441648", SYMLINK+="my_disk"
```

La hiérarchie des périphériques

Dans les faits, le noyau Linux représente les périphériques comme une arborescence. Cette information est ensuite exposée au travers de sysfs et est utile lorsqu'on écrit des règles. Par exemple, la représentation du périphérique de mon périphérique disque dur est un enfant du périphérique disque SCSI, qui est à son tour un enfant du périphérique contrôleur Serial ATA, qui est à son tour un enfant du périphérique bus PCI. Il est probable que vous vous trouverez vous-même dans la nécessité de vous référer aux informations d'un parent des périphériques qui vous intéresse, par exemple le numéro de série de mon disque dur n'est pas exposé au niveau du périphérique, il est exposé par son parent direct au niveau du disque SCSI.

Les quatre principales clés de correspondance introduite jusqu'ici (KERNEL/SUBSYSTEM/DRIVER/ATTR) ne cherchent des correspondances que sur des valeurs compatibles avec le périphérique en question, et ne les cherchent donc pas sur les périphériques parents. udev fournit des variantes de ces clés de correspondances qui vont rechercher en remontant à travers l'arbre :

- KERNELS - recherche des correspondances avec le nom attribué par le noyau au périphérique ou à l'un de ses périphériques parents ;
- SUBSYSTEMS - recherche des correspondances avec le sous-système du périphérique ou de ses périphériques parents ;
- DRIVERS - recherche des correspondances avec le nom pilote prenant en charge le périphérique ou l'un de ses parents ;
- ATTRS - recherche des correspondances avec un des attributs sysfs du périphérique ou de l'un de ses parents.

Avec les considérations de hiérarchie à l'esprit, vous pouvez avoir l'impression que l'écriture de règles devient un peu compliquée. Rassurez-vous, il y a des outils qui aident à s'en tirer et qui seront introduits plus tard.

Les substitutions de chaînes

Lorsqu'on écrit des règles qui prendront potentiellement en charge plusieurs périphériques similaires, les opérateurs de substitution de chaînes façon printf d'udev sont très utiles. Vous pouvez simplement inclure ces opérateurs dans n'importe laquelle des assignations faites par vos règles, et udev les évaluera au moment de leur exécution.

Les opérateurs les plus communs sont %k et %n. %k s'évalue à partir du nom donné par le noyau au périphérique, par exemple « sda3 » pour un périphérique qui apparaîtrait (par défaut) en tant que /dev/sda3. %n s'évalue sur le numéro donné par le noyau au périphérique (le numéro de partition pour les périphériques de stockage), par exemple « 3 » pour /dev/sda3.

udev fournit aussi plusieurs autres opérateurs de substitution pour des usages plus avancés. Consultez la page de manuel dudev après avoir lu le reste de ce document. Il y a également une syntaxe alternative pour ces opérateurs – $kernel et $number pour les exemples ci-dessus. C'est pourquoi si vous désirez rechercher des correspondances avec un « % » littéral dans une règle, vous devez écrire « %% », de même que si vous désirez rechercher des correspondances avec un « $ » littéral, vous devez écrire « $$ ».

Pour illustrer le concept de substitution de chaînes, quelques exemples de règles sont présentés ci-dessous.

```
KERNEL=="mice", NAME="input/%k"
KERNEL=="loop0", NAME="loop/%n", SYMLINK+="%k"
```

La première règle s'assure que le noeud de périphérique de la souris apparaît exclusivement dans le répertoire /dev/input (par défaut, ce serait dans /dev/mice). La seconde règle s'assure que le noeud de périphérique nommé « loop0 » est créé en tant que /dev/loop/0 mais produit également un lien symbolique en tant que /dev/loop0, comme habituellement.

L'utilité de ces règles est discutable, dans la mesure où tout pourrait être réécrit sans aucun opérateur de substitution. Le vrai potentiel de ces substitutions deviendra manifeste dans la prochaine section.

Rechercher des correspondances sur les chaînes

Tout comme il peut rechercher des correspondances avec des chaînes exactes, udev permet d'utiliser des recherches de correspondances sur motifs, comme on en fait en shell. Il y a trois motifs supportés :

- * - correspond avec n'importe quel caractère, aucune fois ou plus ;
- ? - correspond avec n'importe quel caractère, exactement une fois ;
- [] - correspond avec n'importe lequel des caractères spécifiés entre les crochets, les plages sont également autorisées.

Voici quelques exemples qui incorporent les motifs ci-dessus. Notez l'utilisation des opérateurs de substitution de chaînes.

```
KERNEL=="fd[0-9]*", NAME="floppy/%n", SYMLINK+="%k"
```

```
  KERNEL=="hiddev*", NAME="usb/%k"
```

La première règle correspond avec tous les volumes de lecteurs de disquettes, et s'assure que tous les noeuds de périphériques sont placés dans le répertoire /dev/floppy, tout en créant des liens symboliques vers les noms par défaut. La seconde règle s'assure que les périphériques « hiddev » sont présents uniquement dans le répertoire /dev/usb.

Extraire les informations utiles de sysfs

L'arborescence de sysfs

L'idée d'utiliser les informations intéressantes issues de sysfs a été évoquée plus haut. Afin d'écrire des règles basées sur ces informations, vous devez toutefois au préalable connaître les noms des attributs et leurs valeurs actuelles.

sysfs a vraiment une structure très simple. Il est logiquement divisé en catégories, chacune contenant des fichiers (des attributs) qui habituellement ne contiennent eux-mêmes qu'une valeur. Quelques liens symboliques sont également présents et relient les périphériques à leurs parents. La question de la hiérarchisation de cette architecture a été traitée plus haut.

Certain répertoires sont référencés en tant que chemins de périphérique de plus haut niveau. Ces répertoires représentent des périphériques réels qui ont des noeuds de périphérique correspondant. Les chemins de périphérique de plus haut niveau ont comme caractéristique commune d'être des répertoires de sysfs contenant un fichier dev. La commande suivante les listera pour vous :

```
find /sys -name dev
```

Par exemple, sur mon système, le répertoire /sys/block/sda est le chemin de périphérique de mon disque dur. Il est relié à son parent, le périphérique du disque SCSI, par le lien symbolique /sys/block/sda/device.

Lorsque vous écrivez des règles basées sur les informations de sysfs, vous vous contentez de chercher des correspondances avec les attributs contenus dans certains fichiers au sein d'un maillon de la chaîne. Par exemple, je peux lire la taille de mon disque comme suit :

```
cat /sys/block/sda/size
```

```
  234441648
```

Dans une règle udev, je pourrais ensuite utiliser ATTR{size}==“234441648 pour identifier ce disque. Ou bien, puisqu'udev regarde dans chaque maillon de la chaîne du périphérique, je pourrais aussi rechercher des correspondances dans un autre maillon de celle-ci (par exemple dans les attributs de /sys/class/block/sda/device/), toujours en utilisant ATTRS. Il existe toutefois quelques points à prendre en compte lorsqu'on s'appuie sur plusieurs maillons de la chaîne à la fois, ceux-ci seront décrits plus tard.

Bien que cela constitue une introduction utile à la structure de sysfs et à la manière exacte dont udev recherche les valeurs, récolter manuellement les informations à travers sysfs est une large perte de temps.

```
udevinfo
```

C'est ici qu'entre en scène udevinfo, qui est probablement l'outil le plus direct que vous puissiez utiliser pour construire des règles. La seule chose que vous avez besoin de connaître est le chemin de périphérique au sein de sysfs du périphérique qui vous intéresse. Un exemple arrangé pour les besoins de la démonstration est présenté ci-dessous :

```
udevinfo -a -p /sys/block/sda
```

retour de la commande

```
       looking at device '/block/sda':
         KERNEL=="sda"
         SUBSYSTEM=="block"
         ATTR{stat}==" 128535     2246  2788977   766188    73998
317300  3132216  5735004        0   516516  6503316"
         ATTR{size}=="234441648"
         ATTR{removable}=="0"
         ATTR{range}=="16"
         ATTR{dev}=="8:0"

       looking at parent device
'/devices/pci0000:00/0000:00:07.0/host0/target0:0:0/0:0:0:0':
         KERNELS=="0:0:0:0"
         SUBSYSTEMS=="scsi"
         DRIVERS=="sd"
         ATTRS{ioerr_cnt}=="0x0"
         ATTRS{iodone_cnt}=="0x31737"
         ATTRS{iorequest_cnt}=="0x31737"
         ATTRS{iocounterbits}=="32"
         ATTRS{timeout}=="30"
         ATTRS{state}=="running"
         ATTRS{rev}=="3.42"
         ATTRS{model}=="ST3120827AS     "
         ATTRS{vendor}=="ATA     "
         ATTRS{scsi_level}=="6"
         ATTRS{type}=="0"
         ATTRS{queue_type}=="none"
         ATTRS{queue_depth}=="1"
         ATTRS{device_blocked}=="0"

       looking at parent device '/devices/pci0000:00/0000:00:07.0':
         KERNELS=="0000:00:07.0"
         SUBSYSTEMS=="pci"
         DRIVERS=="sata_nv"
         ATTRS{vendor}=="0x10de"
         ATTRS{device}=="0x037f"
```

Comme vous pouvez le voir, udevinfo se contente de produire une liste des attributs que vous pouvez

utiliser tels quels comme clés de correspondances dans vos règles udev. À partir de l'exemple ci-dessus, je pourrais produire (par exemple) l'une ou l'autre de ces deux règles :

```
  SUBSYSTEM=="block", ATTR{size}=="234441648", NAME="my_hard_disk"
  SUBSYSTEM=="block", SUBSYSTEMS=="scsi", ATTRS{model}=="ST3120827AS",
NAME="my_hard_disk"
```

Les deux exemples précédents sont colorisés dans le documents original afin d'en dégager la structure. Le wiki ne nous permettant pas de faire ça simplement, nous avons préféré réadapter le texte à l'absence de couleur (NdT).

Vous aurez peut-être remarqué la manière dont sont combinés dans les règles les attributs du périphérique qui nous intéresse (/block/sda) et ceux de ses parents (/devices/pci0000:00/0000:00:07.0/host0/target0:0:0/0:0:0:0 et /devices/pci0000:00/0000:00:07.0). Alors qu'il est permis de combiner les attributs du périphérique avec ceux d'un unique parent, on ne peut pas utiliser les attributs faisant référence à des périphériques parents différents, autrement votre règle ne fonctionnera pas. Par exemple, la règle suivante est invalide car elle cherche des correspondances sur des attributs (observez les valeurs de ATTRS{model} et de DRIVERS) de deux périphériques parents :

```
  SUBSYSTEM=="block", ATTRS{model}=="ST3120827AS", DRIVERS=="sata_nv",
NAME="my_hard_disk"
```

Avec udevinfo, vous êtes habituellement généreusement fourni en attributs, et vous devez en choisir un certain nombre pour construire vos règles. En général, vous préférerez opter pour des attributs qui identifient votre périphérique de manière à la fois constante et humainement reconnaissable. Dans l'exemple ci-dessus, j'ai choisi la taille de mon disque et le numéro correspondant à son modèle. Je n'ai pas utilisé des nombres insignifiants tels que ATTRS{iodone_cnt}==“0×31737”.

Vous pouvez également observer les effets de la hierarchie des périphériques dans la sortie d'udevinfo. Le premier bloc, qui correspond au périphérique qui nous intéresse, est accessible grâce aux clés de correspondance standard telles que KERNEL et ATTR. Les deux blocs suivants, qui correspondent aux périphériques parents, sont pour leurs parts accessibles grâce aux variantes de ces clés, à savoir SUBSYSTEMS et ATTRS, prévues pour parcourir ce type de périphériques. C'est pourquoi il est vraiment simple de jouer avec cette structure hierarchique, assurez-vous juste d'utiliser les valeurs exactes suggérées par udevinfo.

Un autre point à noter aussi est qu'il est fréquent que le texte des attributs apparaîsse entourré d'espaces (regardez par exemple ST3120827AS, ci-dessus). Dans vos règles vous pouvez soit spécifier ces espaces supplémentaires, soit les élaguer comme je l'ai fait.

La seule véritable difficulté dans l'utilisation dudevinfo est en fait que vous devez connaître le chemin de périphérique de plus haut niveau (/sys/block/sda dans l'exemple ci-dessus), ce qui n'est pas toujours évident. Toutefois, dans la mesure où vous écrivez généralement des règles pour des noeuds de périphériques existant déjà, vous pouvez utiliser udevinfo pour retrouver le chemin de périphérique à votre place :

```
udevinfo -a -p $(udevinfo -q path -n /dev/sda)
```

Les méthodes alternatives

Même si udevinfo est très certainement la manière la plus directe de lister les attributs exacts à partir desquels vous pouvez bâtir des règles, certains utilisateurs sont plus heureux avec d'autres outils. Des utilitaires comme usbview (en) affichent des jeux d'informations similaires, la plupart desquelles peuvent être utilisées dans vos règles.

Utilisations avancées

Contrôler les permissions et les propriétés

udev vous permet d'utiliser des assignations supplémentaires dans vous règles afin de contrôler les permissions et les propriétés attribuées à chaque périphérique.

L'assignation GROUP vous permet de définir quel groupe Unix devrait possèder le noeud de périphérique. Voici un exemple où le groupe « video » possèdera les périphériques framebuffer :

```
KERNEL=="fb[0-9]*", NAME="fb/%n", SYMLINK+="%k", GROUP="video"
```

La clé OWNER, peut-être moins utile, vous permet de définir quel utilisateur Unix devrait avoir la propriété sur le noeud de périphérique. En admettant, cas peu commun, que vous désireriez voir « john » possèder vos périphériques de disquettes, vous pourriez écrire :

```
KERNEL=="fd[0-9]*", OWNER="john"
```

Par défaut, udev crée les noeuds avec des permissions Unix établies à 0660 (droits en lecture/écriture pour le propriétaire et le groupe). Si vous en avez besoin, vous pouvez redéfinir ces permissions sur certains périphériques en employant des règles comportant l'assignation MODE. À titre d'exemple, la règle suivante dit que le noeud « inotify » devrait être accessible en lecture et en écriture pour tout le monde :

```
KERNEL=="inotify", NAME="misc/%k", SYMLINK+="%k", MODE="0666"
```

Utiliser des programmes externes pour nommer les périphériques

Dans certaines circonstances, vous pouvez avoir besoin de plus de flexibilité que ce que les règles standard d'udev peuvent fournir. Dans ces cas-là, vous pouvez demaner à udev de lancer un programme puis d'utiliser la sortie standard de ce programme pour nommer les périphériques.

Pour utiliser cette fonctionnalité, vous avez juste à spécifier le chemin absolu (et les éventuels paramètres) du programme à lancer avec l'assignation PROGRAM. Vous pourrez ensuite utiliser certaines des variantes de l'opérateur de substitution %c dans les assignations NAME/SYMLINK.

Les exemples suivants font référence à un programme fictif dont le chemin est /bin/device_namer. device_namer prend un argument qui est le nom donné au périphérique par le noyau. Se basant sur ce nom, device_namer opère alors sa magie et imprime sur la sortie standard un certain résultat

découpé en plusieurs sections. Chacune de ces sections est un simple mot qui est séparé des autres par un espace unique.

Dans notre premier exemple, nous partons du principe que device_namer affiche un certain nombre de sections, chacune étant destinée à faire un lien symbolique (soit un nom alternatif) pour le périphérique qui nous intéresse.

```
  KERNEL=="hda", PROGRAM="/bin/device_namer %k", SYMLINK+="%c"
```

L'exemple suivant part quant à lui du principe que device_namer affiche deux sections, la première étant le nom du périphérique, la seconde le nom d'un lien symbolique supplémentaire. Nous introduisons ici l'opérateur de substitution %c{N}, qui se réfère à la section N de la sortie :

```
  KERNEL=="hda", PROGRAM="/bin/device_namer %k", NAME="%c{1}",
SYMLINK+="%c{2}"
```

L'exemple d'après considère que device_namer affiche une section pour le nom du périphérique, suivie par un nombre quelconque de sections qui serviront à former des liens symboliques supplémentaires. Nous introduisons à présent l'opérateur de substitution %c{N+}, qui évalue chacune des sections N, N+1, N+2, ... jusqu'à la fin de la sortie.

```
  KERNEL=="hda", PROGRAM="/bin/device_namer %k", NAME="%c{1}",
SYMLINK+="%c{2+}"
```

Les sections de la sortie peuvent par ailleurs être utilisées dans n'importe quelle clé d'assignation, pas seulement NAME et SYMLINK. L'exemple ci-dessous utilise un programme fictif pour déterminer quel groupe Unix devrait possèder le périphérique :

```
  KERNEL=="hda", PROGRAM="/bin/who_owns_device %k", GROUP="%c"
```

Lancer des programmes externes suite à certains événements

Une autre raison éventuelle pouvant amener à écrire des règles udev est de désirer lancer un programme lorsqu'un périphérique est connecté ou déconnecté. Par exemple vous pourriez vouloir exécuter un script pour automatiquement télécharger les photos de votre appareil lorsque celui-ci est connecté.

Veillez à ne pas confondre cela avec la fonctionnalité PROGRAM décrite plus haut. PROGRAM est utilisé pour lancer des programmes qui affichent des noms de périphériques (et ne devraient rien faire en dehors de cela). Lorque ces programmes sont lancés, le noeud de périphérique n'est d'ailleurs pas encore créé, donc agir en se la basant sur le périphérique est de toutes manières impossible.

La fonctionnalité introduite ici vous permet de lancer un programme après que le noeud de périphérique a été mis en place. Ce programme peut donc agir sur le périphérique, toutefois son exécution ne doit pas durer trop longtemps car l'exécution d'udev est vraiment suspendue pendant que les programmes ainsi lancés sont exécutés. Une manière de contourner cette limitation est de s'assurer que votre programme se change lui-même immédiatement en tâche de fond.

Voici un exemple de règle présentant l'utilisation de la liste d'assignation RUN :

```
  KERNEL=="sdb", RUN+="/usr/bin/my_program"
```

Lorsque /usr/bin/my_program est exécuté, diverses parties de l'environnement d'udev sont disponibles en tant que variables d'environnement, parmi lesquelles les valeurs des clés telles que SUBSYSTEM. Vous pouvez également utiliser la variable d'environnement ACTION pour détecter si le périphérique a été connecté ou déconnecté – ACTION prendra alors respectivement pour valeur « add » ou « remove ».

Notez qu'udev ne lance ces programmes sur aucun terminal actif et ne les exécute pas non plus dans le contexte du shell. Assurez-vous par ailleurs que votre programme est bien marqué exécutable et, si c'est un script shell, qu'il commence avec le shebang approprié (par exemple « #!/bin/sh »). Dans tous les cas n'espérez voir aucune sortie standard s'imprimer sur votre terminal.

Interagir avec l'environnement

udev fournit pour les variables d'environnement une clé ENV pouvant être employée à la fois comme une clé de correspondance et d'assignation.

Pour ce qui est de l'assignation, vous pouvez par ce biais définir des variables d'environnement avec lesquelles vous pourrez plus tard chercher des correspondances. Vous pouvez également définir des variables d'environnement utilisables par n'importe quel programme externe invoqué au moyen des techniques mentionnées plus haut. Un exemple de règle fictive définissant une variable d'environnement est présenté ci-dessous :

```
  KERNEL=="fd0", SYMLINK+="floppy", ENV{some_var}="value"
```

Pour ce qui est de la recherche de correspondance, vous pouvez ainsi vous assurer que les règles sont activées uniquement selon la valeur d'une variable d'environnement. Notez que l'evironnement perçu par udev ne sera cependant pas le même que celui de l'utilisateur tel qu'on l'obtient à partir de la console. Une règle impliquant une correspondance avec l'environnement est présentée ci-dessous :

```
  KERNEL=="fd0", ENV{an_env_var}=="yes", SYMLINK+="floppy"
```

La règle ci-dessus ne crée le lien /dev/floppy que si $an_env_var a la valeur « yes » dans l'environnement d'udev.

Options supplémentaires

Une autre assignation pouvant se révéler utile est la liste OPTIONS. Peu d'options sont disponibles :

```
  all_partitions - crée toutes les partitions possibles pour un périphérique
de type bloc, au lieu de seulement celles initialement détectées ;
  ignore_device - ignore complètement l'évènement ;
  last_rule - s'assure qu'aucune règle située après n'aura d'effet.
```

Par exemple, la règle ci-dessous définit le groupe propriétaire du noeud de mon disque dur, puis s'assure qu'aucune règle après elle ne peut avoir d'effet :

```
  KERNEL=="sda", GROUP="disk", OPTIONS+="last_rule"
```

Exemples

Imprimante USB

Je mets mon imprimante sous tension, et il lui est assigné le noeud de périphérique /dev/lp0. Peu satisfait d'un nom aussi insignifiant, je décide d'utiliser udevinfo pour m'aider à écrire une règle qui me fournira un nom alternatif :

```
udevinfo -a -p $(udevinfo -q path -n /dev/lp0)
```

retour de la commande

```
      looking at device '/class/usb/lp0':
        KERNEL=="lp0"
        SUBSYSTEM=="usb"
        DRIVER==""
        ATTR{dev}=="180:0"

      looking at parent device
'/devices/pci0000:00/0000:00:1d.0/usb1/1-1':
        SUBSYSTEMS=="usb"
        ATTRS{manufacturer}=="EPSON"
        ATTRS{product}=="USB Printer"
        ATTRS{serial}=="L72010011070626380"
```

Ma règle devient alors :

```
  SUBSYSTEM=="usb", ATTRS{serial}=="L72010011070626380",
SYMLINK+="epson_680"
```

Appareil photo USB

Comme beaucoup, mon appareil photo s'identifie comme un disque dur externe connecté sur via le port USB en utilisant le protocole SCSI. Pour accéder à mes photos, je monte le volume et je copie les images sur mon disque dur.

Les appareils photos ne fonctionnent cependant pas tous de cette manière : certains d'entre eux, à l'instar des appareils supportés par gphoto2 (en), n'utilisent pas de protocole de stockage. Dans le cas de gphoto, vous n'avez aucun intérêt à écrire des règles pour votre périphérique, dans la mesure où il est entièrement contrôlé en espace utilisateur (et non par un module du noyau prévu à cet effet).

Une des difficultés fréquentes avec les appareils photos USB est que généralement ils s'identifient eux-mêmes comme des disques avec une seule partition, dans le cas qui suit /dev/sdb avec la partition /dev/sdb1. Le noeud /dev/sdb m'est inutile, mais /dev/sdb1 est intéressant car c'est lui que je

veux passer à la commande mount. Il y a toutefois ici un problème parce que dans sysfs liées et que les attributs intéressants produits par udevinfo pour /dev/sdb1 sont identiques à ceux produits pour /dev/sdb. Il en résulte qu'avec votre règle vous pouvez obtenir indistinctement des correspondances avec le disque en tant que tel ou sa partition, ce qui n'est pas ce que vous attendez car votre règle devrait être spécifique.

Pour contourner ce problème, vous avez seulement besoin de réfléchir à ce qui diffère entre sdb et sdb1. Et c'est étonnamment simple : le nom lui-même diffère, de sorte que nous pouvons utiliser une simple recherche de correspondance avec le champ NAME :

```
udevinfo -a -p $(udevinfo -q path -n /dev/sdb1)
```

retour de la commande

```
      looking at device '/block/sdb/sdb1':
        KERNEL=="sdb1"
        SUBSYSTEM=="block"

      looking at parent device
'/devices/pci0000:00/0000:00:02.1/usb1/1-1/1-1:1.0/host6/target6:0:0/6:
0:0:0':
        KERNELS=="6:0:0:0"
        SUBSYSTEMS=="scsi"
        DRIVERS=="sd"
        ATTRS{rev}=="1.00"
        ATTRS{model}=="X250,D560Z,C350Z"
        ATTRS{vendor}=="OLYMPUS "
        ATTRS{scsi_level}=="3"
        ATTRS{type}=="0"
```

Voici-donc ma règle :

```
  KERNEL=="sd?1", SUBSYSTEMS=="scsi", ATTRS{model}=="X250,D560Z,C350Z",
SYMLINK+="camera"
```

Disque dur USB

Un disque dur USB se configure à peu près comme un appareil photo USB, de la manière décrite ci-dessus. Cependant leurs modes d'utilisation conventionnels différent. Dans l'exemple de l'appareil photo, j'ai expliqué que je n'étais pas intéressé par le noeud sdb, celui-ci n'étant en effet réellement utile que pour le partitionnement (par exemple avec fdisk). Or, pourquoi voudrais-je partitionner mon appareil photo !?

Bien sûr, quand vous avez 100Go de disque dur USB, il est à l'inverse parfaitement compréhensible de vouloir le partitionner, auquel cas nous pouvons tirer profit des substitutions de chaînes d'udev :

```
  KERNEL=="sd*", SUBSYSTEMS=="scsi", ATTRS{model}=="USB 2.0 Storage Device",
SYMLINK+="usbhd%n"
```

Cette règle crée des liens symboliques tels que :

```
  /dev/usbhd - le noeud à l'usage de fsdisk ;
  /dev/usbhd1 - la première partition (montable) ;
  /dev/usbhd2 - la seconde partition (montable).
```

Lecteur de cartes USB

Les lecteurs de cartes USB (CompactFlash, SmartMedia, etc.) sont encore une autre classe de périphériques de stockage USB, avec des exigences d'utilisation différentes.

Ces périphériques n'informe généralement pas l'ordinateur hôte du changement de médias. D'où, si vous branchez d'abord le périphérique sans média puis insérez ensuite une carte, l'ordinateur ne s'en rendra pas compte, si bien que vous ne pourrez pas obtenir votre noeud de partition montable sb1 pour le média.

Un solution possible est alors de tirer parti de l'option all_partitions, qui créera seize noeuds de partition pour chacun des périphériques de type bloc qui correspondent à la règle :

```
  KERNEL="sd*", SUBSYSTEMS=="scsi", ATTRS{model}=="USB 2.0 CompactFlash
Reader", SYMLINK+="cfrdr%n", OPTIONS+="all_partitions"
```

Vous aurez ainsi désormais des noeuds appelés cfrdr, cfrdr1, cfrdr2, cfrdr3, ..., cfrdr15.

Palm Pilot USB

Ces périphériques fonctionnent comme des périphériques USB-série, de sorte que par défaut vous n'obtiendrez que le noeud de périphérique ttyUSB1. Les utilitaires de palm se basent sur /dev/pilot, beaucoup d'utilisateurs voudront donc utiliser une règle pour le générer.

Ce billet du blog de Carsten Clasohm (en) paraît être la référence absolue pour cela. La règle de Carsten est exposée ci-dessous :

```
  SUBSYSTEMS=="usb", ATTRS{product}=="Palm Handheld", KERNEL=="ttyUSB*",
SYMLINK+="pilot"
```

Notez cependant que la chaîne qui qualifie le produit (« product ») varie d'un produit à l'autre, assurez-vous donc (grâce à udevinfo) de celle qui convient à votre cas.

Lecteurs CD/DVD

J'ai deux lecteurs optiques sur cet ordinateur : un lecteur de DVD (hdc), et un lecteur/graveur (hdd). Je ne m'attends pas à ce que ces noeuds de périphériques changent, à moins que je ne refasse les branchements physiques de mon système. Toutefois, beaucoup d'utilisateurs aiment avoir pour des raisons pratiques des noeuds de périphérique tels que /dev/dvd.

Comme nous connaissons le nom donné par le noyau pour ces périphériques (clé KERNEL), l'écriture

de la règle est simple. Voici quelques exemples pour pour mon système :

```
SUBSYSTEM=="block", KERNEL=="hdd", SYMLINK+="dvdrw", GROUP="cdrom"
```

Interfaces réseau

Même si on peut s'y référer grâce à leurs noms, les interfaces réseau n'ont cependant habituellement aucun noeud de périphérique leur étant associé. Malgré cela, le processus d'écriture des règles reste le même.

Dans la mesure où elle est unique, le plus rationnel est de simplement chercher dans la règle des correspondances avec l'adresse MAC de votre interface. Assurez-vous toutefois que vous utilisez l'adresse MAC exacte, telle que retournée par udevinfo, car votre règle ne fonctionnera pas si le critère de recherche est inexact.

```
udevinfo -a -p /sys/class/net/eth0
```

retour de la commande

```
looking at class device '/sys/class/net/eth0':
  KERNEL=="eth0"
  ATTR{address}=="00:52:8b:d5:04:48"
```

Voici ma règle :

```
KERNEL=="eth*", ATTR{address}=="00:52:8b:d5:04:48", NAME="lan"
```

Pour que cette règle prenne effet, vous devrez recharger le pilote réseau. Vous pouvez soit décharger puis recharger le module, soit redémarrer le système. Vous aurez également besoin de reconfigurer votre système de façon à utiliser lan à la place de eth0. J'ai eu des problèmes pour que cela fonctionne (l'interface n'avait pas été renommée) jusqu'à ce que je me sois complètement débarrassé des références à eth0. Après cela, vous devriez pouvoir utiliser lan à la place d'eth0 dans tous vos recours à ifconfig ou aux utilitaires similaires.

Tester et déboguer

Mettre vos règles en action

Du moment que vous avez un noyau récent avec le support pour inotify, udev surveillera de lui-même votre répertoire de règles et relèvera automatiquement les modifications que vous ferez sur les fichiers de règles.

Malgré cela, udev ne cherchera pas à reconfigurer automatiquement les périphériques et n'essaiera pas non plus d'appliquer les nouvelle(s) règle(s). Par exemple, si vous écrivez une règle qui ajoute un lien symbolique supplémentaire pour votre appareil photo alors que celui-ci est connecté, ne vous attendez pas à ce que le lien symbolique apparaisse dans la foulée.

Pour faire appparaître le lien symbolique, vous pouvez soit déconnecter puis reconnecter l'appareil, soit, en alternative pour les périphériques inamovibles, exécuter udevtrigger.

Dans le cas où votre noyau ne supporte pas inotify, les nouvelles règles ne seront pas détectées. Dans cette configuration, vous devez lancer udevcontrol reload_rules après toute modification des fichiers de règles pour que celles-ci prennent effet.

udevtest

Si vous connaissez les chemins de périphérique de plus haut niveau dans sysfs, vous pouvez utiliser udevtest pour qu'il montre les actions qu'udev ferait. Cela peut vous aider à déboguer vos règles. Par exemple, en admettant que vous vouliez déboguer une règle qui agit sur /sys/class/sound/dsp :

```
udevtest /class/sound/dsp
```

retour de la commande

```
    main: looking at device '/class/sound/dsp' from subsystem 'sound'
    udev_rules_get_name: add symlink 'dsp'
    udev_rules_get_name: rule applied, 'dsp' becomes 'sound/dsp'
    udev_device_event: device '/class/sound/dsp' already known, remove
possible symlinks
    udev_node_add: creating device node '/dev/sound/dsp', major = '14',
minor = '3', mode = '0660', uid = '0', gid = '18'
    udev_node_add: creating symlink '/dev/dsp' to 'sound/dsp'
```

Notez que le préfixe /sys a été retiré de l'argument fourni sur la ligne de commande d'udevtest. C'est parce qu'udevtest opère sur les chemins de périphérique. Notez également qu'udevtest est un pur outil de test/débogage, il ne crée aucun noeud de périphérique, malgré ce que sa sortie suggère !

1)
N'hésitez pas à y faire part de vos remarques, succès, améliorations ou échecs !