13 Gestion des fichiers

Ce chapitre décrit comment interagir avec des fichiers. Il couvre les thèmes suivants :

- fichiers, répertoires et liens ;
- copie, déplacement et suppression des fichiers et des répertoires ;
- compression et archive des fichiers ;
- recherche de fichiers ;
- gravure des CD et des DVD ;
- droits d'accès aux fichiers ;
- structure des répertoires Linux ;
- fichiers de périphériques.

13.1 Interaction avec des fichiers et des répertoires

Présentons rapidement les faits importants à propos des noms de fichiers :

- Sous Linux, les noms de fichiers sont limités à 255 caractères.
- La casse est importante : Linux distingue les minuscules des majuscules.
- Les caractères internationaux (comme les lettres accentuées) sont autorisés, mais peuvent poser problème lorsqu'un jeu de caractères différent est utilisé (par exemple sur un réseau). Depuis quelque temps, presque toutes les distributions utilisent l'UTF-8 comme jeu de caractères par défaut.
- Les noms de fichiers peuvent contenir plusieurs points. Par exemple, `README.bootutils.gz` est un nom tout à fait commun. Il s'agit d'un fichier README compressé sur le thème des utilitaires d'amorçage.
- Les fichiers qui commencent par un point se comportent comme des fichiers cachés. Ces derniers ne s'affichent pas par défaut avec `ls` ou les divers gestionnaires de fichiers.
- Les noms de fichiers qui ne sont pas interprétables tels quels par les commandes (comme ceux avec des espaces) doivent être encadrés par des guillemets (par exemple, `"a b"`).

La taille des fichiers est virtuellement illimitée. Les anciennes distributions posaient une limite à 2 Go. La limite se calcule actuellement plutôt en téraoctets. Selon la manière dont le noyau Linux est compilé et le système de fichiers utilisé, on peut avoir des fichiers encore plus volumineux.

Répertoires

Arborescence de répertoires

Sous Linux, l'arborescence commence par le caractère `/`. Les lettres de lecteur comme `C:` n'ont plus de sens et ne sont d'ailleurs plus possibles. Dans cet ouvrage, nous considérons que tous les répertoires suivants se situent *en-dessous* dans l'arborescence ; le répertoire racine se trouve donc tout en haut. Certains ouvrages adoptent la convention inverse, ce qui

correspond mieux à la structure d'arbre (la racine en bas, les ramifications vers le haut), mais c'est une terminologie peu courante.

Répertoire personnel

Comme nous l'avons vu au Chapitre 3, une fois connecté, vous accédez directement à votre répertoire personnel ou répertoire `home`. Les fichiers et répertoires qui s'y trouvent vous appartiennent. Les autres utilisateurs (à l'exception de `root`) ne peuvent ni les modifier, ni les supprimer.

Le répertoire personnel se trouve généralement dans l'arborescence à l'emplacement `/home/identifiant_utilisateur` (à l'exception de `root`, pour lequel le répertoire personnel est `/root`). Il serait certes pénible d'avoir à saisir ce chemin systématiquement ; c'est pourquoi le répertoire personnel est accessible grâce au raccourci ~ (caractère tilde). Vous pouvez également utiliser le raccourci `~identifiant` pour accéder aux comptes des autres utilisateurs.

Répertoires . et ..

Dans chaque répertoire, il existe deux sous-répertoires particuliers qui servent à gérer la hiérarchie des répertoires. Le répertoire nommé `.` est un lien vers le répertoire courant et le répertoire `..` est un lien vers le répertoire parent.

Les deux commandes suivantes montrent comment utiliser ces répertoires. La première copie le fichier `/etc/fstab` dans le répertoire courant. Si celui-ci est `/home/nom`, le chemin du nouveau fichier est `/home/nom/fstab`.

```
utilisateur$ cp /etc/fstab .
```

Dans le second exemple, nous commençons par nous déplacer dans le répertoire `~/linux` avec la commande `cd`. La commande `cp` crée une copie de sauvegarde du fichier `partie3.odt` dans le répertoire parent sous le nom `partie3.odt.bak`.

```
utilisateur$ cd ~/linux
utilisateur$ cp partie3.odt ../partie3.odt.bak
```

Si le répertoire personnel s'appelle `/home/nom`, nous avons copié le fichier `/home/nom/linux/partie3.odt` dans le fichier `/home/nom/partie3.odt.bak`.

Caractères spéciaux pour les répertoires

`~`	répertoire personnel
`.`	répertoire courant
`..`	répertoire parent du répertoire courant

Commandes élémentaires de gestion des fichiers et des répertoires

Même si KDE et Gnome fournissent des gestionnaires de fichiers, les utilisateurs avancés de Linux ont tendance à utiliser des commandes en mode texte. Le tableau suivant résume les commandes les plus importantes.

Gestion des fichiers et des répertoires

`cd`	modifie le répertoire courant
`cp`	copie des fichiers
`less`	affiche un fichier texte page par page

`ls`	affiche tous les fichiers d'un répertoire
`mkdir`	crée un nouveau répertoire
`mv`	déplace des fichiers ou modifie leur nom
`rm`	supprime des fichiers
`rmdir`	supprime des répertoires

Liste des fichiers (*ls*)

`ls` affiche une liste de tous les fichiers du répertoire courant. Si vous souhaitez aussi afficher les fichiers cachés, utilisez l'option `-a`. Si vous désirez afficher d'autres informations, telles que la taille du fichier ou le propriétaire, utilisez l'option `-l`. Par défaut, le résultat de `ls` est trié alphabétiquement. Pour trier la liste par date de dernière modification, par taille ou par extension, utilisez respectivement les options `-t`, `-S` et `-X`. La commande suivante affiche tous les fichiers `*.png` dans le répertoire `figures`, triés par taille (dans l'ordre décroissant).

```
utilisateur$ $ ls -l -S figures/*.png
-rw-r--r-- 1 michael users 58865 2008-03-08 12:53 ./these-gantt.png
-rw-r--r-- 1 michael users 53417 2008-07-19 17:08 ./schedule.png
-rw-r--r-- 1 michael users 42516 2008-06-27 12:44 ./dct-vect.png
-rw-r--r-- 1 michael users 30792 2008-03-28 14:09 ./interpol-luma.png
-rw-r--r-- 1 michael users 18984 2008-04-04 16:39 ./filter-samples.png
-rw-r--r-- 1 michael users 16834 2008-03-29 11:15 ./transform.png
-rw-r--r-- 1 michael users 15341 2008-03-24 11:25 ./4ss.png
-rw-r--r-- 1 michael users 14177 2008-04-04 16:33 ./filter-block.png
...
```

Une ligne typique du résultat de `ls -l` ressemble à ceci :

```
-rw-r--r-- 1 michael users 15341 2008-03-24 11:25 ./4ss.png
```

Il s'agit du fichier `4ss.png`, modifié la dernière fois le 24 mars 2008 à 11h25. Il a une taille de 15 341 octets et appartient à un utilisateur dont l'identifiant est `michael` dans le groupe `users`.

Les dix caractères en début de ligne indiquent le type de fichier et les droits d'accès. Le fichier peut être de cinq types différents : `-` indique un fichier normal, `d` un répertoire (*directory*), `b` ou `c` un fichier de périphérique (de type bloc ou caractère), `l` un lien symbolique. Les trois caractères suivants (`rwx`) précisent si le propriétaire peut lire, modifier et exécuter le fichier. Les caractères suivants fournissent des informations similaires pour les membres du groupe et tous les autres utilisateurs du système. Le chiffre situé après ce groupe de dix caractères indique le nombre de liens durs vers le fichier. Nous parlerons plus tard des liens durs et des droits d'accès.

La plupart des distributions configurent `ls` pour qu'il affiche les fichiers et répertoires dans différentes couleurs selon leur type. Si ce n'est pas votre cas, vous pouvez activer ce comportement avec l'option `--color`.

`ls` n'affiche normalement que les fichiers présents dans le répertoire courant. Si vous désirez afficher ceux des sous-répertoires, utilisez l'option `-R`. Cette option, signifiant "récursif", s'applique à de nombreuses commandes sous Linux. La commande suivante fait la liste de tous les fichiers de tous les sous-répertoires (y compris les répertoires et fichiers cachés).

Cette liste est généralement très longue. C'est pour cette raison que l'on transfère le résultat de la commande vers `less` grâce à `| less` :

```
utilisateur$ ls -alR | less
```

Copier des fichiers

La commande `cp nom1 nom2` copie le fichier `nom1` sous le nom `nom2`. Pour copier plusieurs fichiers, appelez la commande `cp nom1 nom2 ... répertoire_cible`. Les commandes suivantes passent dans le répertoire `linux`, créent un répertoire `bak` et y copient tous les fichiers `*.odt`.

```
utilisateur$ cd linux
utilisateur$ mkdir bak
utilisateur$ cp *.odt bak/
```

Pour copier tout un répertoire, ainsi que son contenu, utilisez `cp -a`. Dans l'exemple suivant, `cd` sans argument permet d'aller dans le répertoire personnel. La deuxième commande crée une copie complète du répertoire `linux` sous le nom `linux-bak`.

```
utilisateur$ cd
utilisateur$ cp -a linux linux-bak
```

Supprimer des fichiers et des dossiers

`rm fichier` supprime le fichier indiqué irrévocablement. Par défaut, `rm` ne peut être utilisé que pour les fichiers, et non pour les répertoires. Pour ces derniers, la commande `rmdir répertoire` est prévue, mais elle ne fonctionne que lorsque le répertoire est vide. En pratique, on utilise donc le plus souvent les options `-rf` de `rm`. Cette commande supprime tous les sous-répertoires et fichiers de manière récursive et sans demander de confirmation. Par conséquent, `rm -rf` est une commande très dangereuse ! La commande suivante supprime la copie de sauvegarde créée au paragraphe précédent :

```
utilisateur$ rm -rf linux-bak/
```

Déterminer l'espace nécessaire aux fichiers et répertoires

`ls -l` indique la taille d'un fichier. Il est néanmoins souvent utile de connaître la place prise par tous les fichiers d'un répertoire, l'espace restant sur le disque dur, etc. Les commandes `df` et `du` ont été prévues à cet effet.

Déterminer la capacité d'un disque dur

`df` affiche pour toute partition et tout support de données monté dans le système de fichiers, sa taille totale et l'espace libre restant.

Dans l'exemple suivant, `df` affiche des résultats pour quatre partitions et supports de données. Dans le répertoire système (première ligne), il reste 620 Mo. La deuxième ligne indique la place qu'il reste dans le système de fichiers temporaire. Ce dernier n'a pas de signification dans un usage courant ; il permet des échanges de données rapides entre deux programmes. La troisième ligne est une partition Windows et la quatrième, un répertoire réseau.

```
utilisateur$ df
Sys. de fich.   1K-blocs       Occupé     Disponible    Capacité    Monté sur
/dev/hdc2        6940516      5962076         620192         91%    /
tmpfs             777764            0         777764          0%    /dev/shm
/dev/sda2        4715772      2040692        2675080         44%    /vfat
mars:/data      97881408     77136160       15773120         84%    /data
```

`df` permet également de savoir sur quelle partition se trouve physiquement un répertoire. Dans l'exemple suivant, le répertoire `/home/michael` se trouve sur la partition `/dev/sda6`, montée à l'emplacement `/home` dans l'arborescence :

```
utilisateur$ df /home/michael
Sys. de fich.   1K-blocs   Occupé Disponible Capacité Monté sur
/dev/sda6        9621848  4429352    4703720      49% /home
```

Déterminer la taille d'un répertoire

`du` permet de connaître la taille du répertoire courant et de tous les sous-répertoires. Il renvoie le résultat en kilo-octets. La commande suivante montre que le répertoire `Photos/mariage` occupe 3,2 Go et affiche la répartition de cet espace :

```
utilisateur$ cd Photos/mariage
utilisateur$ du
68704   ./Pierre
210864  ./Janine
51120   ./Julien
25056   ./MarieAure
1168700 ./Alice
287372  ./Loic
289168  ./Didier
44556   ./Bruno
47640   ./Lysiane
110928  ./Michael
140896  ./Florent
15688   ./David
109392  ./Annette
207208  ./Philippe
575192  ./Nicolas
3354012 .
```

Le programme `baobab`, fourni dans les versions récentes de Gnome, est un outil confortable pour afficher la taille des fichiers et des répertoires de manière graphique. Dans KDE, cette vue est intégrée à Konqueror.

Caractères joker

On souhaite souvent exécuter des commandes sur un ensemble de fichiers – par exemple, tous les fichiers se terminant par `.odt`. Pour cela, on utilise des caractères joker dans les paramètres des commandes Linux.

Caractères joker pour les noms de fichiers

`?`	exactement un caractère arbitraire
`*`	un nombre arbitraire (y compris 0) de caractères arbitraires
`[abc]`	exactement un des caractères entre crochets (a, b ou c)
`[a-f]`	exactement un caractère dans l'intervalle indiqué
`[!abc]`	un caractère différent des caractères entre crochets
`[^abc]`	idem

*** et ?**

`?` représente *un* caractère arbitraire, et `*` un nombre arbitraire (y compris 0) de caractères arbitraires. `*` représente n'importe quel caractère, y compris les points, à condition que le fichier ne commence pas par un point. Si vous désirez traiter tous les fichiers, il faut utiliser le raccourci `*` et non pas, comme sous DOS, `*.*`.

Vous pouvez sans problème utiliser plusieurs caractères joker. Ainsi, `*graph*` renverra tous les fichiers dont le nom contient `graph` : par exemple, `graphique.png`, `interfacegraphique` et `LISEZMOI.graph`.

[] et [!]

Lorsque les caractères `?` et `*` sont trop généraux, vous pouvez appliquer des restrictions supplémentaires grâce aux crochets. `[abc]` est joker pour l'une des trois lettres `a`, `b` ou `c`. Lorsque deux éléments sont séparés par un tiret, il s'agit de l'intervalle entre ces deux éléments. Par exemple, `[a-f]*` renverra tous les fichiers dont le nom commence par une lettre entre `a` et `f`. `*[_.-]*` représente un fichier dont le nom contient un tiret de soulignement, un point ou un tiret. On peut aussi exclure des caractères : `[!a-z]*` désigne tous les fichiers dont le nom ne commence pas par une minuscule, et `*.[hc]` tous ceux qui se terminent par `.c` ou `.h`.

Les caractères joker peuvent également être utilisés pour les répertoires. `*/*.odt` représente tous les fichiers `*.odt` dans un sous-répertoire du répertoire courant (à un seul niveau ; cela ne renvoie pas les fichiers dans un sous-sous-répertoire). `/usr/*bin/*` représente tous les fichiers qui se trouvent dans les répertoires `/usr/bin` et `/usr/sbin`.

L'extension des caractères joker n'est pas gérée par les commandes appelées, mais par l'interpréteur qui appelle la commande. `bash` connaît un certain nombre de caractères joker, en plus de ceux que nous venons de décrire.

Problèmes liés à l'utilisation des caractères joker

L'utilisation des caractères joker semble à première vue plus simple que ce qu'elle est réellement. Si vous rencontrez des problèmes avec ces caractères, vous pouvez mener quelques expériences grâce à la commande `echo chaîne`. Elle affiche tous les fichiers concernés par une combinaison de caractères joker sans les modifier.

Un problème classique est que `*` ne représente pas seulement les fichiers, mais aussi les répertoires. Par conséquent, `ls *` affiche tous les fichiers du répertoire courant, mais aussi le contenu de tous les sous-répertoires. Pour éviter cela, on utilise ici l'option `-d` ; ce n'est cependant pas le cas des autres commandes.

Traiter les répertoires avec */.

Lorsque vous souhaitez traiter tous les répertoires (sans toucher aux fichiers), vous pouvez utiliser la combinaison de caractères joker `*/.`. Elle représente tous les "fichiers" qui ont un lien vers eux-mêmes dans leur sous-répertoire, ce qui est uniquement le cas des répertoires.

```
utilisateur$ echo */.
```

Extension gérée par l'interpréteur de commandes

Le fait que l'extension des caractères joker soit gérée par l'interpréteur de commandes, et non directement par la commande, ne présente pas que des avantages. Par exemple, `ls -R *.odt` ne renvoie pas tous les fichiers `*.odt` des sous-répertoires du répertoire courant.

La raison est simple : l'interpréteur de commandes étend le motif `*.odt` dans le répertoire courant et passe à `ls` la liste des fichiers trouvés. Par conséquent, si vous n'avez pas de répertoire se terminant par `.odt`, `ls` a terminé et l'option `-R` ne peut rien y faire.

Pour ce type de recherche, il faut utiliser la commande `find`, dont nous parlerons plus tard :

```
utilisateur$ find . -name '*.odt'
```

Renommer des fichiers

Il n'est pas possible de renommer tous les fichiers `*.x` en `*.y` avec la commande `mv *.x *.y`. La raison est la même que dans le paragraphe précédent. Le motif `*.x` est remplacé par la liste de tous les fichiers correspondants et il n'existe pas de fichier correspondant à `*.y`. La commande `mv` a donc pour paramètres une liste de fichiers se terminant par `*.x` et l'expression `*.y` – et elle ne sait pas comment traiter ces arguments. `mv` ne sait de toute façon traiter plusieurs arguments à la fois que quand le dernier argument est un répertoire.

Renommer des fichiers avec *sed*

Les utilisateurs avancés d'Unix ont évidemment trouvé une parade à ces problèmes : ils utilisent l'éditeur de flux `sed`. Comme il est complexe à utiliser, les exemples ci-dessous s'apparentent plutôt à de la programmation de l'interpréteur de commandes.

Dans ce premier exemple, `ls` fournit la liste des fichiers qui doivent être renommés et la passe à `sed`. Celui-ci construit une liste de commandes `cp` grâce à la commande `s` (recherche d'expressions régulières et remplacement), puis transmet ces commandes à un nouvel interpréteur de commandes qui les lance. Tous les fichiers `*.xxx` sont alors copiés dans des fichiers `*.yyy`.

```
utilisateur$ ls *.xxx | sed 's/\(.*\)\.xxx$/cp & \1.yyy/' | sh
```

Une autre possibilité est de formuler cela sous la forme d'une boucle – nous supposons que l'interpréteur de commandes utilisé est Bash. Cette commande copie tous les fichiers `*.txt` dans des fichiers `*.txt~` (le caractère ~ en fin de fichier est souvent utilisé pour les fichiers de sauvegarde).

```
utilisateur$ for i in *.txt; do cp $i $i~; done
```

Fichiers cachés

Sous Linux, les fichiers commençant par un point sont considérés comme des fichiers cachés. `*` ne renvoie donc pas vraiment tous les fichiers d'un répertoire : ceux qui débutent par un point sont ignorés.

`.*` permet certes de récupérer les fichiers cachés, mais aussi les répertoires `.` et `..` (répertoire courant et répertoire parent). Selon la commande utilisée, cette syntaxe peut être fatale.

On peut utiliser le motif `.[!.]*`. Celui-ci représente tous les noms de fichiers dont le premier caractère est un point, dont le deuxième caractère existe et n'est pas un point, et qui ont un nombre arbitraire de caractères arbitraires.

```
utilisateur$ echo .[!.]*
```

Mais la solution la plus universelle est d'utiliser `find` :

```
utilisateur$ find . -name '.*'
```

13.2 Liens

Les liens permettent d'accéder à un même fichier depuis différents endroits du système de fichiers, sans que ce fichier ne soit physiquement enregistré plusieurs fois. Ils permettent d'éviter la redondance. Ils sont particulièrement courants dans les répertoires `/bin` et `/lib` – vous pouvez par exemple le voir en explorant `/usr/bin` ou `/usr/lib` avec `ls -l`.

Pour comprendre le concept de lien, le plus simple est de partir d'un exemple. Supposons qu'un répertoire `test` contienne le fichier `abc`. La commande `ln abc xyz` semble créer un nouveau fichier `xyz`. En réalité, `abc` et `xyz` sont deux liens vers le même fichier. Pour le vérifier, utilisez la commande `ls -l`. La deuxième colonne indique le nombre de liens associés à un fichier donné (dans notre cas, deux). Si vous utilisez également l'option `-i`, vous pouvez voir que les deux fichiers correspondent au même inode (ce terme sera expliqué plus tard), ce qui prouve qu'il s'agit bien du même fichier sur le système de fichiers.

```
utilisateur$ ls -li
13713417 -rw-r--r-- 1 michael users 205 2008-06-26 15:34 abc
utilisateur$ ln abc xyz
utilisateur$ ls -li
13713417 -rw-r--r-- 2 michael users 205 2008-06-26 15:34 abc
13713417 -rw-r--r-- 2 michael users 205 2008-06-26 15:34 xyz
```

Lorsque vous modifiez l'un des deux fichiers, l'autre est automatiquement modifié (puisqu'il ne s'agit en fait que d'un seul et même fichier). Si vous supprimez l'un des deux, vous ne faites que réduire le nombre de liens.

Des résultats étranges peuvent se produire lorsque vous modifiez avec un éditeur de texte un fichier avec un lien dur : le lien pointe, après le premier enregistrement, sur le fichier de sauvegarde et, après le second enregistrement, dans le vide.

La raison est la suivante : certains éditeurs créent un fichier de sauvegarde lors de l'enregistrement. Ce dernier est un renommage du fichier original, par exemple `abc` en `abc~`. Le fichier modifié est alors un fichier nouvellement créé, avec un nouvel inode et sans lien. Pour éviter cela, utilisez des liens symboliques.

Liens symboliques

Il existe deux types de liens. L'exemple ci-dessus a mis en place des liens durs, créés par défaut par la commande `ln`. Si vous utilisez l'option `-s`, vous créez un lien symbolique. Ces liens ont l'avantage de pouvoir être utilisés sur un disque physique différent de celui du fichier et de pouvoir également être appliqués à des répertoires (ces deux possibilités ne sont pas offertes par des liens durs).

`ls` affiche le fichier d'origine des liens symboliques. En revanche, il n'existe pas de compteur pour le nombre de liens associés au fichier d'origine.

La différence interne entre un lien symbolique et un lien dur est que le premier enregistre le chemin du fichier lié, tandis que le second enregistre l'inode du fichier.

```
utilisateur$ ln -s abc efg
utilisateur$ ls -li
13713417 -rw-r--r-- 2 michael users 205 2008-06-26 15:34 abc
13713415 lrwxrwxrwx 1 michael users   3 2008-06-26 16:01 efg -> abc
13713417 -rw-r--r-- 2 michael users 205 2008-06-26 15:34 xyz
```

Pour créer un lien symbolique, vous devez vous trouver dans le répertoire qui contiendra le lien. Sinon, il arrive que le lien ne pointe pas vers l'endroit souhaité.

Les liens symboliques se comportent également différemment des liens durs. Supprimer le fichier d'origine (en l'occurrence `abc`) ne modifie pas le lien vers ce fichier (`efg`), mais celui-ci pointe alors vers un emplacement vide. Inversement, si le lien symbolique est supprimé, cela n'a aucune influence sur le fichier d'origine.

On peut aussi créer un lien symbolique vers un répertoire. Cela peut cependant causer quelques confusions, car toute l'arborescence pointée par le lien symbolique semble recopiée. En fait, le lien vers le répertoire n'est qu'un chemin d'accès supplémentaire aux mêmes fichiers et sous-répertoires.

Essayez, autant que possible, de ne pas utiliser de chemins absolus, mais relatifs, lorsque vous créez un lien symbolique. Ainsi, vous éviterez des problèmes si le répertoire est monté par NFS ou si vous déplacez des répertoires.

Les liens symboliques et les liens durs ont chacun leurs avantages. Les premiers sont plus simples à gérer, tandis que les seconds demandent moins de place disque et sont plus rapides.

13.3 Chercher des fichiers

Il existe plusieurs possibilités pour chercher des fichiers sous Linux. Les plus importantes sont résumées dans le tableau ci-après. La commande à utiliser dépend du type de fichier recherché et des informations connues (nom du fichier, contenu, etc.).

Recherche de fichiers

`beagle-query`	cherche des fichiers selon leur contenu
`grep`	cherche du texte dans un fichier texte
`find`	cherche des fichiers selon leur nom, leur date, leur taille, etc.
`locate`	cherche des fichiers selon leur nom
`whereis`	cherche des fichiers dans des répertoires prédéfinis
`which`	cherche des programmes dans les répertoires de la variable d'environnement `PATH`

which et *whereis*

`which` cherche la commande passée en argument. Il renvoie le nom complet de la commande lorsque son nom est indiqué sans information de chemin.

Il parcourt les répertoires de la variable d'environnement `PATH` et est donc extrêmement rapide. Cette variable contient une liste de répertoires dans lesquels on peut trouver des

programmes. Notez qu'elle contient généralement plus de répertoires pour `root` que pour un utilisateur normal.

```
utilisateur$ which emacs
/usr/bin/emacs
```

`whereis` parcourt tous les répertoires usuels pour les fichiers exécutables, les fichiers de configuration, les pages de manuel et le code source, afin de trouver le nom de fichier indiqué. Il couvre plus de répertoires que `which` et ne se limite pas aux programmes. Cependant, il n'indique que les fichiers qui se trouvent dans les répertoires prédéfinis pour `whereis` (voir à ce sujet `man whereis`).

```
utilisateur$ whereis fstab
fstab: fstab: /etc/fstab /etc/fstab.pre-uuid /usr/include/fstab.h /usr/share/
➥man/man5/fstab.5.gz
```

locate

`locate motif` affiche les fichiers dont le nom complet (chemin + nom de fichier) correspond au motif passé en argument. La recherche est très rapide car `locate`, plutôt que de parcourir le système de fichiers, accède à une base de données qui contient tous les noms de fichiers du système. Certaines distributions n'affichent que les fichiers auxquels l'utilisateur a effectivement accès. Lancez `locate` en tant que `root` lorsque vous cherchez des fichiers système.

Exemple La commande suivante cherche le fichier de configuration de X, `xorg.conf` :

```
utilisateur$ locate xorg.conf
/etc/X11/xorg.conf
/etc/X11/xorg.conf.backup
/etc/X11/xorg.conf~
/usr/share/man/man5/xorg.conf.5.gz
```

On peut utiliser des caractères joker. Ainsi, `locate '*dvips'` ne renverra que les fichiers dont le nom se termine par `dvips`.

updatedb La qualité des résultats de la recherche dépend largement de la qualité de la base de données de `locate`. La plupart des distributions lancent la commande `updatedb` pour mettre à jour cette base au moins une fois par jour. Il est aussi possible de la lancer manuellement de temps à autre. Cela nécessite cependant des droits `root`.

find et *grep*

`find` est une commande à la fois extrêmement puissante et complexe pour chercher des fichiers. Elle prend en argument différents critères de recherche (motif pour le nom du fichier, date de création ou de dernier accès, etc.). `man find` fournit une référence complète de toutes les options de `find`. Les exemples suivants vous permettront de faire vos premiers pas avec `find`. Notez qu'il est plutôt long à s'exécuter, car il parcourt des pans entiers du système de fichiers, répertoire après répertoire.

find La commande `find`, sans paramètre supplémentaire, affiche une liste de tous les fichiers dans le répertoire courant et ses sous-répertoires.

La commande suivante cherche tous les fichiers qui commencent par `.e` dans le répertoire courant et ses sous-répertoires :

```
utilisateur$ find -name '.e*'
./.emacs.d
./.emacs~
...
./linux-2.6.19.2/linux/scripts/mod/.elfconfig.h.cmd
...
./.emacs
./.eclipse
```

Cet exemple cherche dans le répertoire `/usr/share/texmf` tous les fichiers `*.tex` présents dans un répertoire qui se termine par `latex` :

```
utilisateur$ find /usr/share/texmf -path '*latex/*.tex'
/usr/share/texmf/ptex/platex/base/plnews03.tex
/usr/share/texmf/ptex/platex/base/kinsoku.tex
/usr/share/texmf/ptex/platex/base/plnews08.tex
/usr/share/texmf/ptex/platex/base/pldoc.tex
...
```

L'exemple suivant affiche tous les répertoires contenus dans `/etc/`. Les fichiers réguliers ne sont pas montrés. La liste est triée alphabétiquement avec `sort`.

```
root# find /etc -type d | sort
/etc
/etc/acpi
/etc/acpi/events
/etc/alternatives
/etc/apache2
/etc/apache2/conf.d
...
```

La commande suivante affiche tous les fichiers des sous-répertoires de `/home` qui appartiennent aux utilisateurs du groupe `users` et qui ont été modifiés au cours des cinq derniers jours. `-ctime +5` affiche les fichiers modifiés il y a plus de cinq jours, et `-ctime 5` ceux modifiés il y a exactement cinq jours.

```
root# find /home -group users -ctime -5
...
```

Cette commande supprime tous les fichiers de sauvegarde dans le répertoire courant et ses sous-répertoires. Une liste est construite avec les résultats de `find`, puis elle est passée à `rm` :

```
utilisateur$ rm $(find . -name '*~')
```

Si `find` renvoie beaucoup de fichiers, il se peut que la commande ci-dessus affiche une erreur : la ligne de commande étendue est trop longue par rapport à la taille maximale autorisée. Dans ce cas, vous devez utiliser l'option `-exec` de `find`, ou la commande `xargs`. Reportez-vous aux pages de manuel `man find` et `man xargs` pour plus d'informations à ce sujet.

grep

La commande `grep` parcourt un fichier texte à la recherche d'un motif de recherche. Selon les options, elle affiche les lignes du fichier correspondant ou le nombre de lignes concernées. Le motif de recherche est une expression rationnelle.

La commande suivante cherche la chaîne de caractères "emacs" dans tous les fichiers `*.tex` du répertoire. Une liste de toutes les lignes correspondantes (préfixées par le nom du fichier) s'affiche à l'écran.

```
utilisateur$ grep emacs *.tex
```

Cet exemple compte le nombre de fois que la fonction `arctan` est utilisée dans les fichiers `*.c` du répertoire :

```
utilisateur$ grep -c arctan\(.*\) *.c
```

`grep -v` renvoie comme résultat toutes les lignes qui ne contiennent pas le motif de recherche. Dans l'exemple suivant, `grep` supprime toutes les lignes qui commencent par le caractère `#` dans le fichier `configfile`. La commande `cat` supprime toutes les lignes vides. Le résultat est enregistré dans le fichier `nocomments`.

```
utilisateur$ grep -v '^#' configfile | cat -s > nocomments
```

Combiner *find* et *grep*

Combiner `find` et `grep` permet de mener des recherches particulièrement efficaces. Cet exemple affiche tous les fichiers `*.tex` qui contiennent la chaîne de caractères "emacs" :

```
utilisateur$ find -name '*.tex' -type f -exec grep -q emacs {} \; -print
```

La commande suivante affiche tous les fichiers du répertoire courant, dont la taille est inférieure à 10 Ko et dont le contenu correspond à l'expression rationnelle `case.*in`. La liste des fichiers trouvés est enregistrée dans le fichier `resultat`.

```
utilisateur$ find -name '*' -maxdepth 1 -size -10k  \
        > -exec grep -q case.*in {} \; -print > resultat
```

~~Beagle~~

~~Beagle est un outil de recherche relativement récent, qui fournit certains avantages en plus de ceux des outils traités ci-dessus :~~

- ~~La recherche est rapide ; elle s'appuie comme `locate` sur un fichier d'index.~~
- ~~L'index de recherche est actualisé à chaque modification de fichiers ; il est donc toujours à jour.~~
- ~~La recherche inclut le contenu des fichiers. Alors que la combinaison `find`/`grep` se limite aux fichiers texte, Beagle permet de gérer davantage de formats : documents OpenOffice.org, pages Web (y compris le cache de Konqueror), courriers électroniques (KMail et Evolution), fichiers PDF, etc.~~

~~Beagle présente néanmoins certains inconvénients. Il est développé en C# et nécessite donc d'installer Mono. Son fichier d'index est très volumineux. Il existe des formats que Beagle ne prend pas encore en compte, comme les courriers électroniques de Thunderbird ou les fichiers LaTeX. Les systèmes de données NFS posent également problème : ils peuvent être parcourus, mais cela prend du temps et on ne peut pas actualiser automatiquement l'index~~.

```
READ DVD STRUCTURE[#0h]:
 Media Book Type:          02h, DVD+RW book [revision 2]
 Media ID:                 RICOHJPN/W01
 Legacy lead-out at:       221280*2KB=453181440
...
```

13.6 Droits d'accès, utilisateurs et groupes propriétaires

Linux est conçu comme un système multi-utilisateur : certains mécanismes définissent qui peut accéder à quels fichiers, qui peut les modifier, etc.

Depuis le noyau 2.6, il dispose également d'une gestion avancée des droits grâce aux ACL (*Access Control Lists*, listes de contrôle d'accès). Les ACL seront traités plus en détail à la section 13.8.

Données d'accès par fichier

Chaque fichier ou répertoire est enregistré avec les informations suivantes :

- le propriétaire du fichier ;
- le groupe associé au fichier ;
- les bits d'accès (`rwxrwxrwx` pour lecture/écriture/exécution par le propriétaire, les membres du groupe et le reste du monde) ;
- quelques bits supplémentaires pour certaines fonctions spéciales.

Le propriétaire du fichier est en général la personne qui a créé le groupe. Le groupe associé au fichier est alors normalement le groupe principal du propriétaire.

Les informations d'accès `r`, `w` et `x` définissent qui peut lire, modifier et exécuter le fichier. Elles sont séparées en trois groupes pour le propriétaire, les membres du groupe et les autres utilisateurs du système. Cela permet de donner plus de droits au propriétaire du fichier qu'à d'autres utilisateurs. Ces informations s'appellent bits d'accès, car elles sont enregistrées en interne grâce à un codage en bits. Ce n'est pas la notation `rwxrwxrwx` qui est utilisée en interne, mais 9 bits d'accès, ainsi que 3 bits spéciaux, souvent représentés également sous forme octale.

On peut connaître les bits d'accès, le propriétaire et le groupe d'un fichier grâce à la commande `ls -l`. Pour un fichier texte standard, `ls` renvoie le résultat suivant :

```
michael$ ls -l lisezmoi.txt
-rw-r--r-- 1 michael users 7484 2008-02-19 12:45 lisezmoi.txt
```

En voici une brève explication. Le premier caractère indique le type de fichier : `-` pour un fichier régulier, `d` pour un répertoire, `l` pour un lien symbolique, etc. Ce fichier peut être lu et modifié par son propriétaire, `michael`. Comme il s'agit d'un fichier texte, le premier bit `x` est désactivé : le fichier ne peut pas être exécuté. Tous les autres utilisateurs, qu'ils soient membres du groupe `users` ou non, peuvent le lire (mais pas le modifier).

Si Michael souhaite que seuls les membres du groupe `users` puissent lire le fichier, il doit désactiver le dernier bit `r` :

```
michael$ chmod o-r lisezmoi.txt
michael$ ls -l lisezmoi.txt
-rw-r----- 1 michael users 7484 2008-02-19 12:45 lisezmoi.txt
```

Supposons à présent que l'accès au fichier soit restreint à deux utilisateurs, `michael` et `kate`. Il faut pour cela créer un nouveau groupe auquel seuls Michael et Kate appartiennent, par exemple `doc`. L'appartenance du fichier au groupe doit ensuite être modifiée grâce à `chgrp` :

```
michael$ chgrp doc lisezmoi.txt
michael$ ls -l lisezmoi.txt
-rw-r----- 1 michael doc 7484 2008-02-19 12:45 lisezmoi.txt
```

Droits d'accès sur des répertoires

Les 9 bits d'accès sont également valides pour les répertoires, mais leur signification est différente. Le bit `r` permet aux utilisateurs de voir le contenu d'un répertoire avec `ls`. Le bit `x` leur permet d'aller dans ce répertoire. Lorsque `x` et `w` sont actifs, de nouveaux fichiers peuvent être créés dans le répertoire.

Droits d'accès sur des périphériques

Sous Linux, on accède à divers composants matériels (lecteur de disquette, imprimante, modem, interface série, lecteur de bandes, etc.) *via* les périphériques.

Pour définir, de manière ciblée, quel utilisateur a le droit d'accéder à quels périphériques, ceux-ci sont répartis en divers groupes d'utilisateurs. Par exemple, les périphériques `/dev/ttyS*` pour les ports série appartiennent souvent au groupe `uucp` ou `dialout` :

```
utilisateur$ ls -l /dev/sda
crw-rw---- 1 root dialout 4, 65 2008-06-17 22:14 /dev/ttyS1
```

Si l'administrateur désire que l'utilisateur `hubert` puisse accéder directement à un modem *via* l'interface série, il peut ajouter `hubert` au groupe `dialout` avec la commande `usermod -G`.

Dans certaines distributions, lorsqu'un utilisateur se connecte, les droits d'accès des périphériques les plus courants sont configurés de sorte que l'utilisateur actuel ait un accès illimité aux périphériques correspondants. Par conséquent, les droits d'accès sont souvent modifiés. Fedora utilise pour cela PAM ; SUSE met en œuvre le démon `resmgrd`.

Bits spéciaux

Le sens des trois fois 3 bits `rwxrwxrwx` est simple à comprendre. En plus de ces bits, certaines informations supplémentaires peuvent être enregistrées. La connaissance de ces bits spéciaux n'est généralement nécessaire qu'aux administrateurs système.

Bit setuid

Le bit setuid, ou bit suid, fait en sorte qu'un programme soit toujours lancé comme s'il était démarré par son propriétaire. Ce dernier est souvent `root` ; ainsi, le programme peut être lancé comme s'il l'était par `root`.

Ce bit est utilisé pour donner aux utilisateurs sans privilège des droits qui ne s'appliquent qu'à l'exécution d'un programme donné. Cela peut néanmoins facilement représenter un risque de sécurité, en particulier lorsque le programme exécuté démarre d'autres programmes.

C'est pourquoi il vaut mieux éviter autant que possible l'utilisation des bits setuid. La commande `mount` peut constituer l'une des rares exceptions à cette règle :

```
utilisateur$ ls -l /bin/mount
-rwsr-xr-x 1 root root 81368 2008-04-29 13:57 /bin/mount
```

`ls -l` affiche, à la place du bit `x`, la lettre `s`. La valeur octale du bit setuid est 4000.

Bit setgid

Le bit setgid a un comportement comparable au bit setuid, à ceci près qu'il s'applique aux groupes et non au propriétaire de l'exécutable. `ls -l` affiche pour le programme correspondant la lettre `s` à la place de la lettre `x` dans les bits d'accès du groupe. La valeur octale du bit est 2000.

Pour les répertoires, le bit setgid fait en sorte que les fichiers créés dans le répertoire appartiennent au groupe du répertoire (et non, comme souvent, au groupe de l'utilisateur qui crée le fichier).

Sticky bit

Le sticky bit s'applique aux répertoires. Il fait en sorte que chaque utilisateur pouvant modifier les fichiers du répertoire ne puisse supprimer que ses propres fichiers (et non ceux des autres utilisateurs). Ce bit est par exemple activé sur le répertoire `/tmp`. Tous les utilisateurs peuvent créer des fichiers temporaires. Il faut, en revanche, éviter que les utilisateurs puissent renommer ou supprimer des fichiers qui ne leur appartiennent pas.

`ls -l` affiche pour le répertoire correspondant la lettre `t` à la place de la lettre `x` dans les bits d'accès de l'utilisateur. La valeur octale du bit est 1000. Attention : la signification du sticky bit est spécifique à Linux ; d'autres variantes d'Unix peuvent lui associer un sens différent.

```
utilisateur$ ls -ld /tmp/
drwxrwxrwt 20 root root 8192 2008-06-30 11:59 /tmp/
```

Bits spéciaux dans *ls*

Lorsque vous lancez la commande `ls -l`, il arrive que les bits spéciaux s'affichent sous la forme `S` ou `T`. Il ne s'agit pas de bits spéciaux supplémentaires, mais d'une indication : les bits setuid, setgid ou sticky ont été mal utilisés. `S` signifie que le bit setuid ou setgid est positionné, mais que le bit `x` ne l'est pas (setuid et setgid n'ont alors pas de sens). `T` signifie que le sticky bit est positionné, mais que le répertoire n'a pas le bit d'accès `x` pour le groupe "autres".

Changer les droits d'accès d'un fichier ou d'un répertoire

chmod

La commande `chmod` permet de changer les droits d'accès d'un fichier ou d'un répertoire. Sa syntaxe est de la forme `chmod ensemble+droits fichier` ou `chmod ensemble-droits fichier`. `ensemble` représente les utilisateurs concernés : `u` pour l'utilisateur propriétaire du fichier, `g` pour le groupe propriétaire du fichier, `o` pour les autres utilisateurs du système et `a` pour tous les utilisateurs sans distinction. `droits` représente les droits concernés : `r` pour la lecture (*read*), `w` pour l'écriture (*write*) et `x` pour l'exécution (*execute*).

Ainsi, pour permettre à tous les utilisateurs de lire un fichier, on utilisera :

```
utilisateur$ chmod a+r fichier
```

Pour restreindre la modification d'un fichier à son propriétaire, la syntaxe utilisée sera :

```
utilisateur$ chmod og-w fichier
```

`chmod` permet également d'activer les bits setuid (`u+s`), setgid (`g+s`) et sticky (`+t`).

L'option `-R` offre la possibilité d'effectuer un `chmod` récursif (en parcourant des répertoires).

Valeurs octales

À la place des lettres, on peut aussi indiquer trois ou quatre octets sous forme de chiffres. À chaque bit `u`, `g` et `o` est associé un chiffre. Chaque chiffre est l'addition de valeurs : 4, 2 et 1 pour `r`, `w` et `x`. Ainsi, `660` correspond à `rw-rw----` et `777` à `rwxrwxrwx`. Les bits setuid, setgid et sticky ont respectivement pour valeurs 4000, 2000 et 1000 – par exemple, `chmod 1777` correspond à tous les accès sur un répertoire pour tout le monde, mais avec le sticky bit.

13.7 Propriétaire, groupe et bits d'accès de nouveaux fichiers

Cette section concerne les facteurs qui définissent les informations d'accès des nouveaux fichiers. Pour tester cela facilement, utilisez la commande `touch`. Elle crée un nouveau fichier vide si le fichier en paramètre n'existe pas.

Exemple

L'utilisateur `michael` crée le nouveau fichier `monFichier1`. Il n'est pas surprenant que ce fichier appartienne à l'utilisateur `michael`. `users` est utilisé automatiquement pour le groupe (`users` est le groupe principal de `michael`).

```
michael$ touch monFichier1
michael$ ls -l monFichier1
-rw-r--r-- 1 michael michael 0 2008-06-30 14:47 monFichier1
```

`michael` appartient à de nombreux autres groupes (ce que l'on peut voir avec la commande `groups`). Pour créer un fichier sans qu'il appartienne au groupe primaire, il faut changer de groupe actif grâce à la commande `newgrp` :

```
michael$ groups
users adm dialout cdrom floppy audio dip video plugdev lpadmin scanner
michael$ newgroup audio
michael$ touch monFichier2
michael$ ls -l monFichier2
-rw-r--r-- 1 michael audio 0 2008-06-30 14:49 monFichier2
```

Bits d'accès

La gestion des bits d'accès est un peu plus compliquée. Par défaut, les nouveaux fichiers reçoivent les bits d'accès `rw-rw-rw` (soit, en octal, 666) et sont lisibles et modifiables par tout le monde. Les nouveaux fichiers exécutables (créés par un compilateur) reçoivent les bits `rwxrwxrwx` (soit, en octal, 777) et sont donc de plus exécutables par tout le monde.

Ces paramètres de base sont cependant généralement trop libéraux pour la plupart des utilisateurs. C'est pourquoi tous les interpréteurs de commandes proposent un `umask`. Il s'agit d'une valeur numérique qui indique les bits qui doivent être retirés des bits d'accès par défaut.

Le plus souvent, le masque 022 (`----w--w-`) est utilisé. Les nouveaux fichiers reçoivent donc les bits d'accès 666-022 = 644 (`rw-r--r--`), et les nouveaux programmes le masque 777-022 = 755 (`rwxr-wr-x`). Vous pouvez connaître la valeur courante du masque grâce à la commande `umask` (qui permet également de modifier le masque) :

```
michael$ umask
0022
```

Le paramétrage par défaut du masque se trouve dans le fichier de configuration des interpréteurs de commandes. Pour Bash, il s'agit généralement du fichier `/etc/profile` ou `/etc/bashrc`. Les valeurs qui y sont définies peuvent être redéfinies dans le fichier `~/.bashrc` de chaque utilisateur.

13.8 Listes de contrôle d'accès et attributs étendus

ACL (listes de contrôle d'accès)

Le concept d'utilisateurs et de groupes sous Unix est testé et approuvé depuis des dizaines d'années. Il est très simple, mais dans certains cas, ce système n'est pas suffisant.

C'est pourquoi un nouveau système de gestion de droits, plus fin, a été développé, basé sur les ACL (*Access Control Lists*, listes de contrôle d'accès). Les ACL permettent, pour tout fichier et répertoire, de définir qui peut y accéder en lecture et en écriture et qui ne le peut *pas*, ce que les droits d'accès Unix standard ne permettent pas. Elles étendent les droits d'accès et peuvent offrir de nouveaux droits ou défaire certains droits existants.

Les ACL sont fournies de manière standard par Linux depuis le noyau 2.6. Elles existaient auparavant sous la forme de correctifs. Pour certains systèmes de fichiers, comme jfs et xfs, les ACL sont toujours actives. Pour les systèmes de fichiers ext2, ext3 et reiserfs, il faut monter la partition avec l'option `acl` pour les activer. Ce n'est pas le cas pour la plupart des distributions actuelles.

Ce n'est pas parce que les ACL fournissent davantage de possibilités qu'il faut se débarrasser de la gestion de droits conventionnelle ! Pour les administrateurs avancés de réseaux étendus, elles améliorent la sécurité ou, au minimum, facilitent la gestion. Mais pour la plupart des utilisateurs de Linux, la gestion habituelle des droits suffit amplement. Une utilisation incorrecte des ACL peut entraîner des failles de sécurité. C'est pourquoi elles sont rarement proposées par défaut dans les distributions.

De plus, de nombreuses commandes et programmes ne les gèrent pas correctement. Un fichier copié peut alors soudain perdre les informations ACL de l'original. Par ailleurs, la plupart des gestionnaires de fichiers ne savent ni lire, ni modifier les ACL (Konqueror est une exception).

Samba est de loin le programme le plus important profitant réellement des ACL. Grâce aux ACL, les droits d'accès Windows peuvent être reproduits sous Linux.

Attributs étendus

Les attributs étendus (EA, *Extended Attributes*) sont étroitement liés aux ACL. Ils permettent d'enregistrer, pour chaque fichier, des paires attribut/valeur. Vous pouvez, par exemple, associer à un fichier texte l'attribut `charset` et la valeur `utf8` pour enregistrer le jeu de caractères utilisé par ce fichier. Cela ne présente un avantage que lorsqu'il existe des programmes capables d'exploiter ces informations. Les attributs étendus doivent être activés lors du montage des systèmes de fichiers ext2/3 avec l'option `user_xattr`. Les attributs étendus sont, par exemple, utilisés par le programme de recherche Beagle.

Vous trouverez plus de détails sur les ACL et les attributs étendus dans les pages de manuel `acl`, `getfacl`, `setfacl`, `attr(5)`, `getfattr` et `setattr`. Les adresses suivantes peuvent également être des sources d'informations intéressantes :

http://www.unixgarden.com/index.php/administration-systeme/initiation-aux-listes-de-controle-d'acces-acl-et-aux-attributs-etendus-ea-sous-linux ; **http://www.suse.de/~agruen/** (en anglais) ; **http://acl.bestbits.at/** (en anglais).

Prérequis

Dans l'exemple suivant, on suppose que le paquetage `attr` et les commandes `attr`, `getfattr` et `setfattr` sont installés et que vous travaillez dans un système où les ACL et les

EA sont activés. Dans le cas d'un système de fichiers ext3, le résultat de `mount` doit ressembler à ceci :

```
utilisateur$ mount
...
/dev/hdc5 on /test type ext3 (rw,acl,user_xattr)
...
```

Si ce n'est pas le cas, les exemples suivants rencontreront une erreur du type "Opération non supportée". Il faut alors modifier les options de `mount` dans `/etc/fstab` et remonter le système de fichiers. Le Chapitre 23 sur l'administration des systèmes de fichiers traite de cette problématique.

Listes de contrôle d'accès

getfacl

Sur un système de fichiers avec des ACL, les droits d'accès standard sont également valides. Ils sont souvent décrits comme des ACL minimales. `getfacl` affiche ces droits sous la forme d'ACL :

```
michael$ touch fichier1
michael$ getfacl fichier1
# file: fichier1
# owner: michael
# group: users
user::rw-
group::r--
other::r--
michael$ ls -l fichier1
-rw-r--r-- 1 michael users 0 2008-06-30 17:03 fichier1
```

setfacl

`setfacl` permet uniquement de définir des droits d'accès supplémentaires. Les commandes suivantes donnent à l'utilisateur `isa` et au groupe `doc` des droits en lecture et en modification sur le fichier, et suppriment tout accès à l'utilisateur `pierre` :

```
michael$ setfacl -m isa:rw fichier1
michael$ setfacl -m g:doc:rw fichier1
michael$ setfacl -m pierre:- fichier1
```

La liste de droits affichée par `getfacl` est maintenant plus longue. `ls` affiche les droits d'accès du masque ACL dans la partie du groupe. Les lettres des bits d'accès sont suivies du caractère + qui indique qu'il existe des règles ACL.

```
michael$ getfacl fichier1
# file: fichier1
# owner: michael
# group: users
user::rw-
user:isa:rw-
```

```
user:pierre:---
group::r--
group:doc:rw-
mask::rw-
other::r--
michael$ ls -l fichier1
-rw-rw-r--+ 1 michael users 0 2008-06-30 17:03 fichier1
```

Les ACL sont typiquement utilisées lorsque vous désirez donner des droits à un utilisateur sur vos fichiers, sans pour autant les rendre accessibles à d'autres (par exemple un groupe). Normalement, vous devez pour cela demander à un administrateur de créer un nouveau groupe pour vous-même et l'autre utilisateur. Les ACL permettent de se passer de cette étape.

Masques ACL

Les masques ACL limitent les droits qui peuvent être donnés par des règles ACL. Par exemple, si vous définissez le masque ACL à `r`, aucune règle ACL ne peut définir de droits en écriture ou en exécution à un utilisateur. Le masque ACL a donc priorité sur les règles ACL. Il n'a cependant aucune incidence sur les droits d'accès traditionnels de l'utilisateur et du groupe sur le fichier.

Chaque modification d'une règle ACL entraîne la mise à jour du masque de manière que toutes les règles ACL existantes soient comprises dans le masque. Celui-ci s'affiche dans `getfacl` et `ls -l`.

Vous pouvez définir explicitement un masque grâce à la syntaxe `setfacl -m m:`*`rwx`*, afin de limiter les droits des ACL. Mais attention, ce masque n'est valable que jusqu'à la prochaine définition d'ACL, qui renouvelle le masque.

ACL standard

Pour les répertoires, vous pouvez définir un deuxième jeu de règles : l'ACL standard. Celle-ci ne s'applique pas aux accès du répertoire ; elle définit le motif d'ACL pour les nouveaux fichiers.

Attributs étendus

setfattr* et *getfattr

Les exemples suivants montrent comment enregistrer des attributs avec `setfattr` et les lire avec `getfattr`. Le nombre d'attributs par fichier dans un système de fichiers ext2 est limité.

```
michael$ touch fichier2
michael$ setfattr -n user.language -v fr fichier2
michael$ setfattr --name=user.charset --value=utf8 fichier2
michael$ getfattr -d fichier2
# file: fichier2
user.charset="utf8"
user.language="fr"
```

`getfattr` n'affiche normalement que les attributs dont le nom commence par `user`. Si vous désirez voir d'autres attributs, vous devez indiquer leur nom avec l'option `-n` ou un motif avec l'option `-m`.

```
utilisateur$ getfattr -n security.selinux -d fichier2
# file: fichier2
security.selinux="user_u:object_r:user_home_t:s0^000"
```

13.9 Structure des répertoires sous Linux

Un système Unix classique se compose de milliers de fichiers. Certaines règles concernant l'emplacement des fichiers se sont établies au fil du temps. Cette section donne un premier aperçu de la structure des répertoires sous Linux.

Le système de données commence par le répertoire racine. En général, il ne contient pas de fichiers, mais les répertoires suivants :

- `/bin` contient les commandes élémentaires de Linux que tous les utilisateurs peuvent lancer. Vous trouverez d'autres programmes dans `/usr/bin`.
- `/boot` contient les fichiers nécessaires à l'amorçage du système (généralement *via* GRUB ou LILO). La plupart des distributions y placent également le noyau.
- `/dev` contient tous les fichiers de périphériques. Ces derniers permettent d'accéder à presque tous les composants matériels (par exemple, l'interface série ou une partition du disque dur).
- `/etc` contient les fichiers de configuration du système. Ils gèrent le démarrage du système, la disposition du clavier, les configurations de base pour divers composants et programmes (par exemple la configuration réseau). `/etc` contient quelques sous-répertoires pour regrouper certains fichiers de configuration, par exemple `/etc/X11` pour les fichiers relatifs à X.
- `/home` contient le répertoire personnel de chaque utilisateur du système. Il s'agit d'un répertoire particulier dans lequel l'utilisateur se trouve automatiquement après s'être connecté. Ce dernier a un accès illimité aux fichiers qui s'y trouvent.
- `/lib[64]` contient les bibliothèques partagées ou des liens symboliques vers ces bibliothèques. Ces fichiers sont utilisés pour exécuter des programmes. `/lib/modules` contient les modules du noyau qui peuvent être activés ou désactivés pendant l'exécution du système. Vous trouverez des bibliothèques supplémentaires dans `/usr/lib[64]`. Le répertoire `/lib/firmware` contient le microcode (*firmware*) de certains composants matériels (par exemple, une carte Wi-Fi).
- `/lost+found` est normalement vide. S'il contient des fichiers, il s'agit de fragments de fichiers qui n'ont pas pu être remis en ordre lors de la tentative de réparation du système de fichiers (`fsck`).
- `/mnt` et `/media` contiennent des sous-répertoires, comme `cdrom` ou `floppy`, dans lesquels on peut monter des systèmes de fichiers externes.
- `/opt` est prévu pour les paquetages additionnels. Certaines distributions (par exemple SUSE) utilisent les répertoires `/opt/gnome` ou `/opt/kde` pour enregistrer les paquetages et bibliothèques spécifiques à Gnome ou KDE.
- `/proc` contient les sous-répertoires qui correspondent à tous les processus courants. Il ne s'agit pas de vrais fichiers.
- `/root` contient les données de l'utilisateur `root`. Elles ne se trouvent usuellement pas dans `/home`. `/home` est souvent une partition séparée et, en cas de problème sur cette partition, `root` peut ainsi travailler dans son répertoire, même lorsque `/home` n'est pas montée.

- `/sbin` contient les commandes d'administration système. En général, les programmes ne pouvant être lancés que par `root` y sont enregistrés.
- `/share` contient parfois des fichiers indépendants de l'architecture. `/usr/share` est un endroit plus usuel pour cela.
- `/srv` contient, dans certaines distributions, les données pour les processus serveurs : par exemple `/srv/www` pour les documents servis par Apache, `/srv/ftp` pour les fichiers FTP, etc.
- `/sys` contient le système de fichiers `sysfs` du noyau 2.6. Tout comme `/proc`, il contient des données sur l'état de l'ordinateur.
- `/tmp` contient des fichiers temporaires. Ces derniers peuvent aussi se trouver dans le répertoire `/var/tmp`.
- `/usr` contient toutes les applications, le système X, le code source de Linux, etc. Idéalement, on ne devrait y trouver que des fichiers statiques (non modifiables). Ainsi, on pourrait monter le répertoire `/usr` dans sa propre partition en lecture seule. Cela aurait l'avantage d'une plus grande sécurité, mais l'inconvénient d'une maintenance très pénible ; c'est en pratique rarement le cas.
- `/var` contient les fichiers variables. Les sous-répertoires importants sont `adm` (fichiers d'administration dépendant de la distribution), `lock` (fichiers de verrous pour l'accès aux périphériques), `log` (fichiers de journalisation), `mail` (fichiers du courrier électronique, pouvant également se trouver dans `spool/mail`) et `spool` (fichiers tampon, comme les fichiers à imprimer, les messages, etc.).

Le principe de la structure des répertoires se situant directement sous la racine est relativement simple à comprendre. Les problèmes apparaissent avec la division de `/usr` et de `/var` en d'innombrables sous-répertoires. En principe, de nombreux répertoires sont nommés comme au niveau de la racine, par exemple `bin` pour les programmes exécutables.

Mais il existe plusieurs groupes de programmes exécutables : les commandes texte, les programmes X, etc. Par conséquent, il existe plusieurs possibilités pour ranger ces programmes. Pour des raisons historiques, plusieurs chemins sont généralement maintenus par le biais de liens. Ainsi, `/usr/bin/X11` correspond aux mêmes programmes que `/usr/X11R6/bin`.

Le répertoire *`/usr`*

`/usr/bin`	programmes exécutables
`/usr/games`	jeux et parfois un lien vers `/usr/share/games`
`/usr/include`	fichiers "include" de C
`/usr/lib[64]`	bibliothèques, parmi lesquelles de nombreux sous-répertoires pour le compilateur C, divers langages de programmation et de gros paquetages comme `emacs` ou LaTeX
`/usr/local`	applications et données qui ne font pas partie de la distribution
`/usr/sbin`	programmes exécutables par `root`
`/usr/share`	fichiers indépendants de l'architecture et documentation
`/usr/src`	code source de Linux, voire d'autres programmes
`/usr/tmp`	lien vers `/var/tmp` (fichiers temporaires)

13.10 Fichiers de périphériques

Le système de fichiers de Linux ne contient pas seulement des fichiers et des répertoires, mais aussi des périphériques. Il s'agit de fichiers spéciaux, dans lesquels on ne peut pas enregistrer de données mais qui, pour la plupart, fournissent un lien vers le noyau.

Numéros de périphériques majeur et mineur

Les périphériques permettent d'accéder à de nombreux composants matériels de l'ordinateur : disque dur, lecteur de disquette, ports série et parallèle, mémoire, etc. Ils sont définis par trois informations : les numéros de périphériques majeur et mineur (*Major Device Number* et *Minor Device Number*) et le type d'accès (par bloc ou par caractère).

Le numéro majeur indique le pilote du noyau Linux qui gère le périphérique. Il existe de nombreux pilotes, listés dans `/usr/src/linux/Documentation/devices.txt`. Le numéro mineur permet de distinguer plusieurs périphériques distincts qui partagent le même pilote, par exemple deux partitions d'un même disque. Le type d'accès indique si le périphérique a un tampon (ce qui est le cas pour tous les périphériques en mode bloc, tels que les disques durs), ou pas (par exemple, l'interface série ou parallèle).

Si vous explorez le contenu du répertoire `/dev` avec `ls -l`, vous verrez, à la place de la taille du fichier, les numéros de périphériques (majeur et mineur). Le premier caractère des droits d'accès est `b` ou `c`, pour les périphériques bloc ou caractère.

```
utilisateur$ ls -l /dev/sda?
brw-rw---- 1 root disk 8, 1 2008-06-17 22:15 /dev/sda1
brw-rw---- 1 root disk 8, 3 2008-06-17 22:14 /dev/sda3
...
```

Fonctionnement interne

En interne, le répertoire `/dev` ne contient que des inodes, qui sont les plus petites unités gérées par un système de fichiers, mais pas de réels fichiers. On peut créer de nouveaux fichiers de périphériques avec la commande `mknod`. Cela est cependant rarement nécessaire, car la gestion du matériel de Linux s'en occupe en théorie automatiquement.

Pour des raisons de sécurité, seul `root` et les membres d'un groupe donné peuvent accéder à certains périphériques. Pour que d'autres utilisateurs puissent y accéder, vous pouvez les ajouter au groupe auquel appartient le périphérique.

Certains fichiers de périphériques ont une fonction particulière. `/dev/null` est un "trou noir" : les données qui sont envoyées disparaissent pour toujours – il est, par exemple, utilisé pour les sorties de commandes qui ne doivent pas s'afficher. `/dev/zero` est une source sans fin d'octets 0, par exemple utilisé pour remplir un fichier d'une taille donnée de zéros.

Système *udev*

Autrefois, les distributions créaient un grand nombre de fichiers de périphériques – Red Hat 9 en créait, par exemple, presque 8000. Mais chaque ordinateur n'en utilisait au mieux que quelques centaines.

Pour corriger cela, le système de fichiers `devfs` du noyau 2.4 devait créer les fichiers de périphériques dynamiquement à la demande. Il n'a cependant jamais eu le succès escompté et a été depuis supprimé du noyau.

Le système `udev` a pris la suite de `devfs` dans le noyau 2.6. Les fichiers de périphériques sont également créés dynamiquement dès que le noyau reconnaît des composants matériels. On peut voir que le système `udev` est actif lorsque le processus `udevd` existe. Ce dernier est

démarré au début du processus de démarrage InitV. Sa configuration se trouve dans le répertoire `/etc/udev`.

Vous trouverez une description complète des périphériques définis sous Linux, ainsi que des numéros associés dans le fichier `/usr/src/linux/Documentation/devices.txt` (le code source du noyau doit être installé).