Permissions des systèmes de fichiers

Aperçu des permissions

Comme nous l'avons vu, Slackware Linux est un système d'exploitation multi-utilisateurs. De ce fait, les systèmes de fichiers sont également multi-utilisateurs. Cela signifie que chaque fichier ou répertoire dispose de ses propres permissions qui permettent d'attribuer ou refuser des privilèges aux différents utilisateurs. Il y a trois permissions basiques et trois jeux de permissions pour chaque fichier. Jetons un coup d'œil sur un fichier d'exemple.

```
darkstar:~$ ls -l /bin/ls
-rwxr-xr-x 1 root root 81820 2007-06-08 21:12 /bin/ls
```

Souvenez-vous, nous avons vu au chapitre 4 que ***ls*** *-l* liste les permissions pour un fichier ou un répertoire ainsi que l'utilisateur et le groupe qui *"possède"* le fichier. Dans notre cas, les permissions sont rwxr-xr-x, l'utilisateur est root et le groupe est également root. La section pour les permissions, qui est regroupée, est en fait constituée de trois ensembles. Le premier ensemble de trois lettres sont les permissions attribuées pour l'utilisateur qui possède le fichier. Le deuxième ensemble de trois lettres sont celles pour le groupe propriétaire et les trois dernières forment l'ensemble pour les permissions pour tous les autres.

Table 10.1. Permissions de /bin/ls

Ensemble	Liste	Signification
Propriétaire	rwx	Le propriétaire "root" peut lire, écrire et exécuter
Groupe	r-x	Le groupe "root" peut lire et exécuter
Autres	r-x	Tous les autres peuvent lire et exécuter

Les permissions s'expliquent d'elles-même, au moins pour les fichiers. Lire, écrire et exécuter permettent de lire le fichier, y écrire ou de l'exécuter. Mais que représentent ces permissions pour les répertoires ? En simplifiant, le droit en lecture permet de lister le contenu du répertoire (avec ***ls*** par exemple). Le droit en écriture donne la possibilité d'y créer de nouveaux fichiers tout comme de supprimer tout le contenu même si vous ne seriez pas en mesure de supprimer des fichiers à l'intérieur. Le droit en exécution permet de rentrer dans le répertoire proprement dit (avec la commande cd intégrée à ***bash*** par exemple).

Regardons les permissions pour un répertoire maintenant.

```
darkstar:~$ ls -ld /home/alan
drwxr-x--- 60 alan users 3040 2008-06-06 17:14 /home/alan/
```

Nous voyons ici les permissions pour mon répertoire personnel et ses propriétaires. Le répertoire est détenu par l'utilisateur alan et le groupe users. L'utilisateur dispose de tous les droits (rwx), le groupe ne dispose que des permissions pour lire et exécuter (r-x) et aucune autre personne n'est autorisée à faire quoi que ce soit.

chmod, chown, et chgrp

Maintenant que nous savons à quoi correspondent les permissions, comment les modifier ? Et dans le même registre, comment assigne t-on un utilisateur et un groupe propriétaire ? Les réponses sont dans cette section.

Le premier utilitaire que nous présenterons est la commande ***chown***(1). En utilisant ***chown***, nous pouvons (vous l'aurez deviné), changer le propriétaire d'un fichier ou d'un répertoire. ***chown*** est historiquement utilisé uniquement pour le changement de propriétaire, mais peut servir également pour le groupe propriétaire.

```
darkstar:~# ls -l /tmp/foo
total 0
-rw-r--r-- 1 alan users 0 2008-06-06 22:29 a
-rw-r--r-- 1 alan users 0 2008-06-06 22:29 b
darkstar:~# chown root /tmp/foo/a
darkstar:~# ls -l /tmp/foo
total 0
-rw-r--r-- 1 root users 0 2008-06-06 22:29 a
-rw-r--r-- 1 alan users 0 2008-06-06 22:29 b
```

En utilisant le caractère deux points après le compte utilisateur, vous pouvez indiquer un nouveau nom de groupe.

```
darkstar:~# chown root:root /tmp/foo/b
darkstar:~#  ls -l /tmp/foo
total 0
-rw-r--r-- 1 root users 0 2008-06-06 22:29 a
-rw-r--r-- 1 root root  0 2008-06-06 22:29 b
```

chown peut aussi être utilisé récursivement pour changer les attributions de tous les fichiers et répertoires sous le répertoire cible. La commande suivante changera tous les fichiers dans le répertoire `/tmp/foo` pour définir les attributions à root:root.

```
darkstar:~# chown -R root:root /tmp/foo/b
```

Un caractère deux points suivi d'un nom de groupe, sans nom d'utilisateur, changera simplement le groupe pour le fichier et laissera les droits de l'utilisateur intacts.

```
darkstar:~# chown :wheel /tmp/foo/a
darkstar:~# ls -l /tmp/foo
ls -l /tmp/foo
total 0
-rw-r--r-- 1 root wheel 0 2008-06-06 22:29 a
-rw-r--r-- 1 root root  0 2008-06-06 22:29 b
```

La petite sœur de la commande ***chown*** est la moins utile ***chgrp***(1). Cette commande se comporte exactement comme ***chown***, excepté qu'elle ne peut modifier que les droits pour un groupe sur un fichier. Dès lors que ***chown*** peut déjà faire cela pourquoi s'embêter avec ***chgrp*** ? La réponse est

simple : beaucoup d'autres systèmes d'exploitation utilisent une version différente de ***chown*** qui ne peut pas changer les droits pour les groupes. Donc si d'aventure vous en rencontrez, vous saurez quoi faire.

Il y a une raison pour laquelle nous avons parlé des changements de propriétaire avant les changements de permissions. Ces dernières sont moins facile à appréhender. L'utilitaire pour changer les permissions d'un fichier ou d'un répertoire est ***chmod***(1). Sa syntaxe est très proche de celle de ***chown***, mais plutôt que d'indiquer un utilisateur ou un groupe, l'administrateur doit indiquer soit un ensemble de permissions octales ou un ensemble de permissions alphabétiques. Ni l'une, ni l'autre n'est facile à comprendre du premier coup. Nous commencerons par les permissions octales qui sont les moins compliquées.

Le nom permissions octales provient du fait qu'elles utilisent huit chiffres, de 0 à 7. Pour chaque permission est défini un chiffre qui est une puissance de deux, et ces chiffres sont additionnés pour obtenir la valeur finale des permissions pour un ensemble. Cela peut paraître confus, peut-être que le tableau ci-dessous rend les choses plus claires.

Table 10.2. Permissions octales

Permission	Signification
Lecture	4
Écriture	2
Exécution	1

En additionnant ces valeurs ensembles, vous pouvons obtenir n'importe quelle valeur entre 0 et 7 et définir toutes les combinaisons possibles de permissions. Par exemple, pour donner à la fois les privilèges en lecture et en écriture et interdire celui en exécution nous devons utiliser la valeur 6. Le nombre 3 donnerait les droits en lecture et en exécution mais interdirait de lire le fichier. Nous devons indiquer un nombre pour chacun des trois ensembles lorsque l'on utilise les permissions octales. Il n'est pas possible de ne donner des permissions que pour un utilisateur ou un groupe par exemple.

```
darkstar:~# ls -l /tmp/foo/a
-rw-r--r-- 1 root root  0 2008-06-06 22:29 a
darkstar:~# chmod 750 /tmp/foo/a
darkstar:~# ls -l /tmp/foo/a
-rwxr-x--- 1 root root  0 2008-06-06 22:29 a
```

chmod peut aussi utiliser des valeurs alphabétiques en combinaison avec + ou − pour définir ou révoquer des permissions. Bien que cela soit facile à mémoriser, il est souvent plus facile d'utiliser les permissions octales.

Table 10.3. Codes alphabétiques des permissions

Permission	Lettre
Lecture	r
Écriture	w
Exécution	x

Table 10.4. Codes alphabétiques des utilisateurs et groupes

Comptes concernés	Lettre
Utilisateur / propriétaire	u
Groupe	g
Reste du monde	o

Pour utiliser ces lettres avec ***chmod***, vous devez préciser quel ensemble doit être modifié. Vous devez aussi indiquer si vous ajoutez ou supprimez des permissions avec les symboles “+” et “-”. Plusieurs ensembles peuvent être changés à la fois en le séparant par des virgules.

```
darkstar:/tmp/foo# ls -l
total 0
-rw-r--r-- 1 alan users 0 2008-06-06 23:37 a
-rw-r--r-- 1 alan users 0 2008-06-06 23:37 b
-rw-r--r-- 1 alan users 0 2008-06-06 23:37 c
-rw-r--r-- 1 alan users 0 2008-06-06 23:37 d
darkstar:/tmp/foo# chmod u+x a
darkstar:/tmp/foo# chmod g+w b
darkstar:/tmp/foo# chmod u+x,g+x,o-r c
darkstar:/tmp/foo# chmod u+rx-w,g+r,o-r d
darkstar:/tmp/foo# ls -l
-rwxr--r-- 1 alan users 0 2008-06-06 23:37 a*
-rw-rw-r-- 1 alan users 0 2008-06-06 23:37 b
-rwxr-x--- 1 alan users 0 2008-06-06 23:37 c*
-r-xr----- 1 alan users 0 2008-06-06 23:37 d*
```

La méthode que vous devez utiliser est à votre convenance. Dans certains cas l'une est meilleure que l'autre, et un vrai Slacker saura s'adapter.

SUID, SGID, et le "Sticky" Bit

Nous n'avons pas tout à fait terminé avec les permissions. Il reste trois permissions *“spéciales”* en plus de celles présentées précédemment. Ce sont les permissions SUID, SGID, et le sticky bit. Lorsqu'un fichier à une (ou plus) de ces permissions définies, il se comporte d'une manière particulière. Les permissions SUID et SGID modifient la façon dont une application est lancée, alors que le sticky bit limite les suppressions de fichiers. Ces permissions sont définies avec ***chmod*** comme pour lecture, écriture ou exécution, mais avec une astuce.

SUID et SGID signifient respectivement *“Set User ID”* et *“Set Group ID”*. Lorsqu'un programme s'exécute avec un de ces droits, il s'exécute avec les permissions de l'utilisateur ou du groupe défini sans tenir compte de l'utilisateur qui l'a lancé. Regardons un des programmes SUID courant, le petit ***passwd*** et les fichiers qu'il modifie.

```
darkstar:~# ls -l /usr/bin/passwd /etc/passwd /etc/shadow
-rw-r--r-- 1 root root    1106 2008-06-03 22:23 /etc/passwd
-rw-r----- 1 root shadow   627 2008-06-03 22:22 /etc/shadow
-rws--x--x 1 root root   34844 2008-03-24 16:11 /usr/bin/passwd*
```

Remarquez les permissions sur ***passwd***. Au lieu d'un `x` pour le compte utilisateur nous avons un `s`. Cela nous indique que ***passwd*** est un programme SUID, et que quand nous l'exécutons, le processus

sera lancé en tant qu'utilisateur *“root”* et non pas comme l'utilisateur qui aura effectivement lancé la commande. La raison pour cela apparaît de manière visible lorsque l'on regarde les deux fichiers que cette commande modifie. Ni le fichier `/etc/passwd` ou le fichier `/etc/shadow` ne sont modifiables que par par root. Du fait que les utilisateurs ont besoin de modifier leurs informations personnelles, ***passwd*** doit être exécuté en tant que root pour pouvoir modifier ces fichiers.

Et que fait dont le sticky bit ? Le sticky bit empêche la possibilité de déplacer ou supprimer un fichier et n'est défini que sur des répertoires. Les utilisateurs autres que root ne peuvent pas déplacer ou supprimer un fichier dans un répertoire avec le sticky bit actif à moins qu'il ne soit propriétaires de ce fichier. Normalement toute personne avec le droit en écriture peur faire cela, mais le sticky bit bloque cela pour tout le monde sauf le propriétaire (et root, bien sûr). Jetons un œil sur un répertoire *“sticky”* courant.

```
darkstar:~# ls -ld /tmp
drwxrwxrwt 1 root root   34844 2008-03-24 16:11 /tmp
```

Naturellement, étant un répertoire pour les fichiers temporaires du système `/tmp` requiert d'être lisible, modifiable et exécutable par n'importe qui. Et comme n'importe que utilisateur peut avoir un ou deux fichiers conservés ici à n'importe quel moment cela semble logique d'éviter que d'autres utilisateurs ne puissent les supprimer, ce qui explique pourquoi le sticky bit est défini. Vous pouvez le voir par la présence d'un `t` à la place d'un `x` dans les permissions pour le reste du monde.

Table 10.5. Permissions SUID, SGID et “Sticky”

Permission	Valeur octale	Valeur alphabétique
SUID	4	s
SGID	2	s
Sticky	1	t

Lorsque vous utiliser les permissions octales, vous devez préciser une valeur octale supplémentaire en premier. Par exemple, pour recréer les permissions de `/tmp`, nous utiliserons 1777. Pour recréer les permissions sur `/usr/bin/passwd`, nous devrions utiliser 4711. D'une manière générale, à chaque fois que le premier des quatre octets n'est pas indiqué ***chmod*** assume que sa valeur est de zéro.

```
darkstar:~# chmod 1777 /tmp
darkstar:~# chmod 4711 /usr/bin/passwd
```

Utiliser des valeur de permissions alphabétiques est légèrement différent. En partant du principe que les deux fichiers suivants auraient des permissions de 0000 (aucune permission définie), voici comment nous les définirions :

```
darkstar:~# chmod ug+rwx,o+rwt /tmp
darkstar:~# chmod u+rws,go+x /usr/bin/passwd
```

Navigation

Chapitre précédent : Utilisateurs et groupes

Chapitre suivant : Travailler avec les systèmes de fichiers

Sources

- Source originale : http://www.slackbook.org/beta

- Publication initiale d'Alan Hicks, Chris Lumens, David Cantrell, Logan Johnson
- Traduction initiale de escaflown
- Traduction de Ellendhel

slackbook, filesystem, permissions, suid, sgid, sticky bit, chmod, chown, chgrp