Droits sur les fichiers, gestion des processus

1 Gestion des utilisateurs et droits

Dans les salles de TP réseau, les machines ne possèdent qu'un seul utilisateur "normal", l'utilisateur `moi`. Pour expérimenter les droits, nous allons d'abord devoir créer de nouveaux utilisateurs.

Exercice 1, Création d'utilisateurs :

1) Devenir super utilisateur (avec la commande `su`, mot de passe `moi`)

2) Créer deux groupes, `bleu`, `rouge` avec la commande `addgroup`.

3) Créer quatre utilisateurs `ciel`, `cyan`, `carmin`, `magenta`, qui appartiennent à leur groupe correspondant (`bleu` pour les deux premiers, `rouge` pour les deux suivants). Vous utiliserez `adduser` et `addgroup`.

4) Vérifiez, en consultant `/etc/passwd` et `/etc/group` que les utilisateurs et groupes que vous avez tenté de créer l'ont bien été !

5) Quels droits ont été attribués par défaut au répertoires personnels de chaque utilisateurs ? Est-ce que `carmin` peut afficher le contenu du répertoire personnel de `magenta` ? De `cyan` ?

6) Ouvrir quatre terminaux. Changer d'utilisateur avec `su <login>` dans chaque fenêtre devenir un utilisateur différent. Faire ensuite l'exercice suivant.

Exercice 2, Gestion des droits :

Pour cet exercice, on part du principe que vous êtes l'utilisateur `cyan`, mais vous pouvez être amené à changer d'identité (de fenêtre !) pour effectuer des tests.

1) Affichez les droits de votre racine (personnelle).

2) Repérez le nom de votre groupe (commande `id`).

3) Créez un répertoire `public` dans votre racine et faites en sorte que les membres de votre groupe puissent lire les fichiers qu'il contient. Votre répertoire racine doit rester illisible pour les membres de votre groupe et les autres utilisateurs : vérifier avec `magenta` !

4) Créez un répertoire `scripts` dans le répertoire `public`, puis créez dans le répertoire `script` un script Shell effectuant un simple `ls`. Faites en sorte que le script soit exécutable par les membres de votre groupe. Testez avec l'utilisateur `ciel` : cet utilisateur peut-il utiliser ce script pour afficher le contenu du répertoire personnel de `cyan` ?

5) L'exécution du script shell modifie-t-elle les droits du fichier (son propriétaire) ?

Exercice 3, Suppression d'utilisateurs :

Dans la suite, vous serez de nouveau l'utilisateur `moi`, vous devez faire un peu de ménage en tant que super utilisateur.

1) Supprimer d'abord le répertoire personnel de l'utilisateur `cyan` (avec un `rm`) : l'utilisateur peut-il encore se connecter ? Si oui, quel est son répertoire de connexion ?

2) Supprimer proprement vos utilisateurs jouets (`cyan`, `ciel`, `magenta`, `carmin`) avec `deluser`.

3) Supprimer les répertoires personnels restants de ces utilisateurs.

4) Supprimer proprement les groupes jouets (`bleu`, `rouge`) avec `delgroup`.

2 gestion des processus

Exercice 4, Découverte des processus :

Un processus est un programme en cours d'exécution sur le système. L'un des rôle du système d'exploitation est de permettre à un ensemble de processus de progresser dans leur excécution en même temps sur votre système : il doit partager les ressources disponibles (processeurs, mémoire, périphériques) entres les différents processus. Ainsi, à chaque instant de nombreux processus sont soit en cours d'exécution, soit bloqués en attente d'exécution, mais vous avez toujours l'impression qu'ils s'éxécutent en « même temps ».

1) Par défaut, la commande `ps` liste les processus rattachés au terminal (`TTY`) dans lequel elle est lancée. Ouvrez un terminal, et entrez la commande `ps` : parmi les informations listées, interprétez les colonnes `CMD`, `PID` et `TTY`.

2) Lancez `gedit &` (on reviendra après sur le « `&` »), puis entrez à nouveau `ps` : qu'observez-vous ?

3) Lancez deux fois la commande `xclock -update 1 &`, puis entrez à nouveau `ps` : qu'observez-vous ?

4) Maintenant, entrez la commande `bash`, puis lancez à nouveau une horloge à l'arrière plan avec `xclock -update 1 &`. Utilisez maintenant la commande `ps -l` pour afficher plus d'informations. Comment interprétez-vous la colonne `PPID` ?

5) Dans le résultat de la commande `ps -l` précédente, comment interprétez-vous la colonne `UID` ? Pour vous aider, vous pouvez aussi vous intéresser au résultat de la commande `id`.

6) Vous avez dû comprendre que les processus sont organisés d'une façon arborescente. Pour la visualiser, vous pouvez utiliser la commande `pstree` : à l'aide de `ps`, repérer le `PID` du premier processus `bash` lancé dans votre terminal ; on note n ce `PID` ; entrez la commande `pstree -p` n. Que constatez vous ?

7) Maintenant, ouvrez un *nouveau* terminal, puis entrez la commande `ps` : vous ne retrouvez pas dans la liste les processus que vous aviez lancés dans le premier terminal... Comment faire pour afficher la liste de tous les processus que vous avez lancés sur le système ?

8) On reprend la question précédente, mais depuis le terminal `tty1` : pour cela, tapez `Ctrl+Alt+F1` (les trois touches en même temps ; `Ctrl+Alt+F7` vous ramène à l'interface graphique) ; loguez-vous, puis afficher la liste de vos processus. Au passage : combien en avez-vous lancés ?

A la fin de cet exercice, fermez proprement tous les terminaux que vous avez ouverts en cours de route !

Exercice 5, Découverte des signaux :

Les signaux sont une manière simple de communiquer avec les processus, sans passer par les entrées et sorties standards. On expérimente ici simplement pour essayer de comprendre les bases. Un signal est un code entier `sig`, que l'on peut envoyer à un processus identifié par son `pid` : la commande `kill -sig pid` envoie le signal `sig` au processus `pid`. En pratique, on n'utilise pas les code entiers, mais des noms plus facile à retenir :

— `TERM` pour demander aimablement au processus de se terminer : certains processus peuvent l'ignorer.
— `KILL` pour demander énergiquement au processus de se terminer : aucun processus ne peut ignorer.
— `STOP` pour demander au processus de s'arrêter.
— `CONT` pour demander au processus de reprendre son exécution.

Par exemple, pour envoyer le signal `CONT` au processus 69007, la commande à utiliser est `kill -CONT 69007`. On ne va travailler qu'avec ces quatres signaux. En fait, il en existe une (très) longue liste : tapez `kill -l` pour afficher toute la liste.

Ouvrez deux terminaux. Dans l'un, lancer `xclock -update 1`, et notez le `pid` du processus correspondant.

1) Envoyez le signal `STOP` à `xclock` : qu'observer vous ?

2) Envoyez le signal `CONT` à `xclock` : qu'observer vous ?

3) Envoyez le signal `TERM` à `xclock` : qu'observer vous ?

4) Faites les mêmes manipulations avec `gedit` : lorsque vous envoyez `STOP` à `gedit`, que constatez-vous ?

5) A votre avis, quand dans un terminal vous tapez `Ctrl+C` pour fermer un processus, que se passe-t-il ?

Exercice 6, Gestion basique des jobs :

En utilisant lui même des signaux, Bash fournit des moyens de gérer l'interaction avec les processus : Notamment, dans un unique bash, vous pouvez travailler avec plusieurs processus : vous le savez déjà, mais le but de l'exercice est d'aller un peu plus loin.

Par défaut, lorsque vous lancez un processus dans un terminal avec Bash, ce nouveau processus prend le contrôle du terminal : c'est ce qui se passe par exemple quand vous lancez `gedit` ; même si `gedit` ouvre sa propre fenêtre graphique, le Bash ne reprend pas son exécution normale, et attend que `gedit` se termine pour ré-afficher le prompt. On dit que `gedit` est lancé à l'*avant-plan*.

Par contre, quand vous lancez `gedit &`, une fenêtre `gedit` s'ouvre, et vous conserver la possibilité d'entrer des commandes à Bash. On dit que `gedit` a été lancé à l'*arrière-plan*.

1) Depuis un terminal, lancez `gedit` à l'avant-plan, puis, dans le terminal, tapez `Ctrl+Z` : qu'observez-vous ?

2) Dans le terminal, entrez `fg` (pour *foreground*) : que se passe-t-il ?

3) Dans le terminal, tapez à nouveau `Ctrl+Z`, puis entrez `bg` (pour *background*) : qu'observez-vous ?

Dans les questions précédentes, vous avez-fait passer un processus de l'avant à l'arrière plan dans le terminal. Pour jongler avec plus de processus à l'aide de bash, on peut utiliser le systèmes de jobs fournis par Bash. Commencez par créer deux fichiers textes bidons, disons `titi` et `tutu` dans votre répertoire courant.

1) Ouvrez `titi` avec `nano titi` (oui, utilisez `nano`, pas `gedit` !). Faites passer `nano` à l'arrière-plan : vous devez revenir au Bash.

2) Ouvrez `tutu` avec `nano tutu`. A nouveau, faites passer `nano` à l'arrière-plan pour revenir au Bash. Ensuite, entrez la commande `jobs` : comment interpétez vous le résultat de cette commande ? En utilisant la commande `fg` (`man bash`), faites passer `nano titi` puis `nano tutu` à l'avant-plan.

3) A quoi tout cela peut bien servir ?

3 Système de fichiers, manipulations en mode utilisateur

Les manipulations de cette section se font comme d'habitude, directement au terminal.

Exercice 7, Liens physiques et symboliques : Expérimentez les liens en Linux/UNIX :

1) Créez un fichier `foo` contenant la ligne `foo is foo`, puis affichez son numéro d'inode (voir le man de `ls`) : le numéro d'inode identifie de façon unique l'emplacement du fichier dans le système de fichiers.

2) Créez une copie de ce fichier avec `cp`.

3) Créez un lien physique puis un lien symbolique vers le fichier `foo` (commande `ln`).

4) Affichez le contenu de ces quatre fichiers.

5) Comparez les inodes de ces quatre fichiers et expliquez.

6) Supprimez le fichier `foo` et regardez le contenu des trois fichiers restant.

7) Créez un nouveau fichier `foo` contenant la ligne `foo is not so foo`.

8) Consultez à nouveau les inodes des quatre fichiers et leur contenu. Expliquez.

Exercice 8, Commande `ssh` : À l'aide de la commande `ssh` ou de son copain `scp` :

1) Se connecter sur la machine du voisin avec le compte moi/moi (en récupérant l'adresse IP de ladite machine, en demandant à son voisin d'utiliser `ifconfig`). Vérifier que le contenu de `tmp` est différent que celui de votre machine locale.

2) Copier un fichier de votre compte de votre machine locale vers le `/tmp` de la machine distante.

3) Effectuer un `ls` sur une machine distante (sans s'y loguer auparavant).

4) Tenter d'ouvrir le lecteur de CD de la machine distante !