

Introduction aux systèmes d'exploitation (IS1)

TP 5 corrigé – Bilan et approfondissements

Semaine du 16 octobre 2006

Modalités

Ce sujet comporte 4 pages. Vous disposerez d'une semaine pour établir un compte-rendu répondant aux différentes questions qui sont posées dans ce TP. Les questions notées en italique sont plus générales : elles peuvent être ignorées si vous manquez de temps ; qui plus est, vous pourrez y réfléchir chez vous. Le compte-rendu pourra prendre la forme que vous préférez : courrier électronique, rapport manuscrit, fichier imprimé...

Enregistrer sa session (*facultatif*)

La commande `script` peut vous aider à établir votre compte-rendu. En effet, `script CR1` lance un nouveau shell et crée un fichier `CR1` qui contiendra l'ensemble des commandes que vous taperez dans le shell ainsi que les réponses des programmes. Ce fichier, ou les différents fichiers `CR1`, `CR2`, ... si vous utilisez plusieurs shells, pourra servir de réponse à la majorité des questions.

On vous demandera cependant de découper ces réponses par exercice et de n'en conserver que la partie intéressante (si vous avez fait des fautes de frappe, par exemple). L'exercice qui suit vous permettra de vous familiariser avec cet utilitaire.

Exercice 1 – *Enregistrer sa session.*

1. Dans un terminal, créez un sous-répertoire `sessions` depuis votre répertoire personnel (ou répertoire maison).
2. Depuis le même terminal, lancez une nouvelle session enregistrée à l'aide de la commande `script` de telle façon que la session soit enregistrée dans le fichier `CR1` situé dans le répertoire `sessions`.
3. Lancez une commande qui écrit sur le terminal la ligne de texte : *Souriez, cette session est enregistrée.*
4. Faites la liste des fichiers de votre répertoire personnel.
5. Faites la liste des fichiers du répertoire `~/sessions/`.
6. Quittez la session enregistrée en tapant `exit` depuis la ligne de commande.
7. Avec un éditeur de texte, ouvrez le fichier `~/sessions/CR1` et supprimez toutes les lignes qui ont à voir avec la question 4.

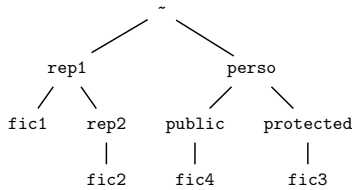
ATTENTION : si vous exécutez deux commandes `script` avec le même fichier en argument, votre première session enregistrée sera écrasée, et donc **perdue** ! Soyez prudents !

N'oubliez pas de lancer `script CRi` à chaque fois que vous lancez un nouveau shell pour conserver une trace de votre travail.

Arborescence et droits.

Exercice 2 – Échauffement

1. Reproduisez l'arborescence ci-dessous. Vous laisserez les droits par défaut sur les fichiers et les répertoires.



2. Je veux que le répertoire perso contienne mes données personnelles. Je souhaite :

- être le seul à pouvoir voir ce que contient perso.
- que seuls les utilisateurs de mon groupe puissent lire les fichiers contenus dans le répertoire protected.
- que tout les utilisateurs puissent lire les fichiers contenus dans le répertoire public.
- que personne ne puisse renommer mes fichiers ou mes répertoires, sauf moi.

3. Modifiez les droits du répertoire rep1 pour que personne ne puisse le lire. Affichez les droits du répertoire rep2.
4. Modifiez les droits du répertoire rep1 pour être le seul à pouvoir le lire et y écrire. Faites ce qui est nécessaire pour que vous et les utilisateurs de votre groupe puissiez lire le fichier fic1. Est-ce nécessaire de modifier les droits pour les autres utilisateurs afin qu'il ne puissent pas lire ce fichier ? Justifiez.

Variables

En plus des usages que vous avez vus jusqu'ici, le shell est un véritable langage de programmation, dont nous découvrirons plusieurs facettes dans la suite de ce module. En particulier, comme en Java, il est possible de déclarer et d'affecter des *variables*.

Une variable est repérée par un nom appelé *identificateur*, qui peut être n'importe quelle suite de caractères commençant par une lettre ou le caractère « souligné » (`'_'`) et ne contenant que des lettres, des chiffres ou le caractère souligné¹.

Chaque variable possède une *valeur*, qui peut être n'importe quelle chaîne de caractères. Par exemple, la valeur de la variable SHELL est `/usr/local/bin/bash`.

On peut déclarer et affecter simultanément une variable nommée `var` avec l'instruction `var="valeur"` (sans espace avant ni après le signe `'='`). On accède à la valeur associée à une variable en faisant précéder son identificateur du symbole `$`, comme par exemple dans la commande `echo $var` qui affiche la valeur de la variable `var`.

Exercice 3 – Familiarisation avec les variables.

1. Vérifiez que la variable SHELL a bien pour valeur `/usr/local/bin/bash`.
2. Déclarez une variable NOM contenant votre nom complet, et affichez-la pour vérifier que l'affectation est correcte.

¹À l'exception de certaines variables spéciales du shell.

3. La commande `set` utilisée sans argument affiche la liste de toutes les variables connues par le shell. Testez cette commande et repérez votre variable dans la liste.
4. Écrivez une ligne de commande qui affiche « Bonjour , Machin Chose ! » (si vous vous appelez Machin Chose, bien sûr) en utilisant la variable `NOM`.
5. Depuis le terminal courant, ouvrez un nouveau shell en tapant la commande `bash`, puis affichez la valeur de la variable `NOM` (quand vous avez terminé, tapez `exit` pour revenir au shell précédent). Faites de même dans un nouveau terminal lancé depuis la barre d'icônes du bureau.

Qu'en déduisez-vous sur la portée des variables du shell ?

Dans certains cas, on souhaite que la valeur d'une variable soit accessible à d'autres processus du système. De telles variables sont appelées variables *d'environnement*.

6. Affichez la liste des variables d'environnement grâce à la commande `env`. Que remarquez vous par rapport à la sortie de la commande `set` ?
7. On transforme une variable `var` en variable d'environnement grâce à la commande `export var`. Transformez `NOM` en variable d'environnement et répétez la question 5. Qu'en déduisez-vous sur la portée des variables d'environnement ?
8. Utilisée sans argument, la commande `cd` fixe le répertoire courant au répertoire personnel de l'utilisateur. Ce comportement est en fait déterminé par la valeur d'une certaine variable d'environnement.

Repérez cette variable dans la liste des variables d'environnement, et faites en sorte que la commande `cd` renvoie par défaut vers le répertoire racine.

Liens

- 1n La commande `ln` sert à créer des *liens*, dits *physiques* (*hard links*), et des *liens symboliques* (*soft links*). Elle s'utilise de façon similaire à la commande `cp` :

`ln [-s] source destination`

où *source* est le nom du fichier dont on crée un lien, et *destination* est le nom du lien.

`-s` indique que le lien est symbolique ; par défaut (c'est à dire sans cette option) le lien est physique ;

Exercice 4 – Créons des liens

1. Placez-vous dans votre répertoire maison, `~`. Créez un répertoire `rep` et un fichier `fic` dans ce répertoire. Retournez dans `~`. Donnez trois façons de désigner le fichier `fic` depuis `~`.
2. Comme expliqué ci-dessus, la commande « `ln` » sert à créer des *liens*. Utilisez-la pour créer un lien **physique**² du fichier `fic` dans `~` sous le nom de votre choix. On désignera ce lien sous le nom `lfic` dans la suite.

²Autrement dit, pas un lien symbolique.

3. Modifiez avec un éditeur de texte le fichier `lfic`. Que constatez-vous pour le fichier `fic`? Réciproquement, modifiez `fic`, lisez `lfic` et concluez.
4. Modifiez les droits d'accès au fichier `fic` pour les membres du groupe. Que constatez-vous pour `lfic`? (Bonus) *Pouvez-vous avancer une explication?*
5. La commande « `ln` » peut aussi créer des liens *symboliques* avec l'option `-s`. Créez un lien symbolique du fichier `fic` dans `~`.
On désignera ce lien sous le nom `sfic` dans la suite. Regardez toutes les informations concernant les fichiers `lfic` et `sfic`. Quelles différences notez-vous?
6. Essayez de modifier les droits d'accès au fichier `sfic`. Que constatez-vous?
7. Modifiez les droits d'accès au répertoire `rep` pour ne plus y avoir accès. Essayez d'afficher le contenu de `lfic` et `sfic`. Que constatez-vous?
(Bonus) *Pouvez-vous avancer une explication?*

Processus et signaux

Exercice 5 – *Le compte est bon*

1. Copiez dans votre répertoire personnel le fichier `chiffres` situé dans le répertoire `/users/monitIS1/`.
2. Modifiez les droits de **vo**tre copie du fichier de manière à pouvoir l'exécuter.
3. Lancez le programme `chiffres` au premier plan. Quelle ligne de commande devez-vous taper?
4. Tuez le programme.

Pour pouvoir lancer directement ce programme depuis n'importe quel répertoire en tapant simplement `chiffres`, il faut indiquer au shell une liste de répertoires où rechercher les fichiers exécutables.

Cette liste est contenue dans une variable d'environnement (cf. exercice 3) appelée `PATH`. Par exemple, si `PATH` vaut `/bin:/usr/bin` (liste de deux noms absolus de répertoires séparés par `:`), la commande `chiffres` sera recherchée d'abord dans `/bin` puis dans `/usr/bin`, et si elle n'est trouvée ni dans l'un ni dans l'autre, une erreur est produite.

5. Quelle est la valeur actuelle de la variable `PATH`?
6. Modifiez la variable `PATH` afin que le shell recherche les exécutables en *dernier* dans votre répertoire personnel. Affichez la nouvelle valeur de `PATH` et vérifiez que votre modification fonctionne.

Le programme `chiffres` a la fâcheuse manie de s'amuser avec les signaux qu'il reçoit, comme vous le verrez dans les deux questions suivantes...

7. Lancez maintenant `chiffres` en avant-plan. Comment le faire passer en arrière plan?
8. Cette question est **facultative** et doit être traitée à la fin de la séance s'il vous reste du temps. Dialoguez avec le processus `chiffres` à l'aide des signaux `SIGCONT`, `SIGALRM`, `SIGPIPE`, `SIGHUP`, `SIGILL`, `SIGUSR1`, `SIGTRAP`, `SIGBUS`, `SIGUSR2` pour parvenir à ce que le programme `chiffres` termine de façon "propre", c'est à dire sans que vous ayez à lui envoyer un signal `SIGKILL` ou `SIGINT`. Vous pouvez vous servir de `xcalc`.