Introduction aux systèmes d'exploitation (IS1)

TP 5 corrigé – Bilan et approfondissements

Semaine du 16 octobre 2006

Modalités

Ce sujet comporte 7 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).

   Correction
   ```
   $ script sessions/CR1
   ```

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`.

   Correction
   ```
   $ cd ~ ou cd
   $ mkdir sessions
   ```

3. Lancez une commande qui écrit sur le terminal la ligne de texte : `Souriez, cette session est enregistrée.`

   Correction
   ```
   $ echo Souriez, cette session est enregistree.
   ```

4. Faites la liste des fichiers de votre répertoire personnel.

Correction
`$ ls`

5. Faites la liste des fichiers du répertoire `~/sessions/`.

Correction
`$ ls 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.

Correction Les sessions enregistrées contiennent de nombreuses lignes inutiles. Pensez pour votre compte-rendu à éditer vos fichiers pour ne conserver que les parties intéressantes !

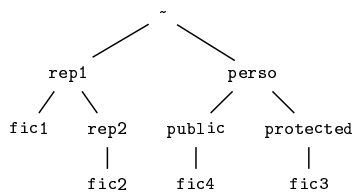
ATTENTION : si vous exécutez deux commandes `script` avec le même fichier en argument, votre première session enregistrée sera écrasée, et donc **perdue** ! Soyez prudents !

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.

Correction On commence par créer les répertoires et les fichiers :

```
$ cd
$ mkdir rep1 rep1/rep2 perso perso/public perso/protected
$ touch rep1/fic1 rep1/rep2/fic2 perso/public/fic3 perso/protected/fic3
perso/public/fic4
```

Puis on modifie leurs droits :

```
$ chmod u+r,go-r perso
$ chmod ug+r,o-r protected protected/fic3
$ chmod a+r perso/public perso/public/fic3
$ chmod -R go-w perso rep1 (-R permet de traiter récursivement les répertoires en argument)
```

Il faut aussi faire attention aux droits de traversée des répertoires :

```
$ chmod ugo+x . perso perso/public
$ chmod chmod ug+x,o-x perso/protected
```

3. Modifiez les droits du répertoire `rep1` pour que personne ne puisse le lire. Affichez les droits du répertoire `rep2`.

Correction
`$ chmod a-r rep1` (raccourci pour `ugo-r`)
`$ ls -ld rep1/rep2` (l'option -d indique qu'on s'intéresse au répertoire `rep2` lui-même et pas à son contenu)
Autre possibilité :
`$ ls -la rep1/rep2` (-a affiche les fichiers cachés. Le répertoire `rep2` sera donc ici symbolisé par le répertoire `.`)

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.

Correction
```
$ chmod u+rw,go-rw rep1
$ chmod ug+r,o-r rep1/fic1
```
Si l'on suppose que tout le monde possède les droits en traversée (`x`) sur `rep1`, alors il est nécessaire de retirer les droits en lecture sur `fic1` pour les autres utilisateurs.

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 souligné[1].

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

On peut déclarer et affecter simultanément une variable nommée `v r` avec l'instruction `v r="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 $v r` qui affiche la valeur de la variable `v r`.

Exercice 3 – Familiarisation avec les variables.

1. Vérifiez que la variable `SHELL` a bien pour valeur `/bin/b sh`.

Correction
`$ echo $SHELL`

2. Déclarez une variable `NOM` contenant votre nom complet, et affichez-la pour vérifier que l'affectation est correcte.

Correction
`$ NOM="Marilyn Monroe" ; echo $NOM`

[1] À 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.

Correction La liste des variables est assez longue, mais elle est triée par ordre alphabétique (majuscules d'abord), ce qui facilite le repérage. Deux possibilités pour se faciliter la vie :
`$ set | less` ou encore mieux : `$ set | grep NOM`.

4. Écrivez une ligne de commande qui affiche « `Bonjour, M chin Chose !` » (si vous vous appelez Machin Chose, bien sûr) en utilisant la variable `NOM`.

Correction
`$ echo Bonjour, $NOM !`

5. Depuis le terminal courant, ouvrez un nouveau shell en tapant la commande `b sh`, 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 ?

Correction La variable `NOM` n'est définie ni dans un sous-shell lancé en tapant `bash`, ni dans un nouveau terminal créé depuis la barre de lancement. Conclusion : les variables ne sont par défaut définies que dans le shell courant.

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` ?

Correction La liste des variables d'environnement est plus courte que la liste obtenue par la commande `set`. C'est en fait une sous-liste : tous ses éléments apparaissent **aussi** dans la liste des variables.

7. On transforme une variable `v r` en variable d'environnement grâce à la commande `export v r`. Transformez `NOM` en variable d'environnement et répétez la question 5.

Qu'en déduisez-vous sur la portée des variables d'environnement ?

Correction Après avoir tapé la commande `export NOM` et lancé un nouveau shell avec la commande `bash`, on remarque que contrairement à la question 5 `NOM` est bien définie dans ce nouveau shell. Par contre, elle n'est toujours pas définie dans un terminal lancé depuis la barre de lancement.
Conclusion : une variable d'environnement est accessible à tous les processus **descendants** du processus où elle a été définie, et uniquement à ces processus.

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.

Correction Il s'agit de la variable d'environnement `HOME`. On peut la repérer dans la liste grâce à sa valeur (ou en concsultant `man bash`...).
Pour faire en sorte que `cd` renvoie par défaut vers le répertoire racine, il suffit de taper
`$ HOME=/`
Il est amusant de remarquer que le caractère spécial `~` utilisé seul est aussi remplacé par la valeur de cette variable.

Liens

`ln` 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 dir 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 `~`.

Correction
`$ cd ; mkdir rep ; touch rep/fic`
`$ ls rep/fic ; ls /home/mmonroe/rep/fic ; ls /rep/fic ; ls perso/../rep/fic ; ls ../mmonroe/rep/fic` (et bien d'autres...)

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.

Correction
`$ ln rep/fic lfic`

3. Modifiez avec un éditeur de texte le fichier `lfic`. Que constatez-vous pour le fichier `fic` ? Réciproquement, modifiez `fic`, lisez `lfic` et concluez.

Correction Toute modification du fichier `lfic` modifie `fic` et vice-versa.

4. Modifiez les droits d'accès au fichier `fic` pour les membres du groupe. Que constatez-vous pour `lfic` ?

Correction La modification des droits de `lfic` entraîne la même modification sur `fic` et vice-versa.

(Bonus) Pouvez-vous avancer une explication ?

Correction `lfic` et `fic` représentent en fait le même fichier sur le disque dur. Ils sont totalement indistinguables (comme des clones dans les films de clones). Cela explique qu'ils contiennent toujours les mêmes données et possèdent les mêmes droits.

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 ?

[2] Autrement dit, pas un lien symbolique.

Correction
`$ ln -s rep/fic sfic ; ls -l lfic sfic`
On remarque que contrairement à `lfic`, `sfic` possède tous les droits d'accès, et qu'il est listé sous la forme `sfic -> rep/fic`. On voit aussi que le premier caractère de la chaine de droits de `lfic` est un `l` (lettre l minusule).

6. Essayez de modifier les droits d'accès au fichier `sfic`. Que constatez-vous ?

Correction Les modifications que l'on essaie de faire sur `sfic` sont directement répercutées sur `fic`. Par contre, les droits du lien `sfic` lui-même ne changent pas.

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 ?

Correction
`$ chmod u-x rep`
Il est toujours possible de lire `lfic` (la commande `cat lfic` fonctionne), mais pas `sfic` (on a un message d'erreur `Permission denied`).

(Bonus) Pouvez-vous avancer une explication ?

Correction Comme dit précédemment, un lien physique est simplement un nouveau nom pour un fichier existant. Comme on possède toujours les droits en lecture sur le répertoire maison, on peut toujours lire `lfic`.
Par contre, `sfic` est simplement un « raccourci » vers le fichier `rep/fic`, donc y accéder nécessite de posséder les droits en traversée sur `rep`.
De façon similaire, si l'on efface le fichier `rep/fic`, on pourra toujours utiliser le lien physique `lfic`, mais plus le lien symbolique `sfic` qui pointera alors dans le vide (on parle de lien brisé, ou pendant).

Processus et signaux

Exercice 5 – Le compte est bon

1. Copiez dans votre répertoire personnel le fichier `chiffres` situé dans le répertoire `/users/monitIS1/`.

Correction
`$ cp /users/monitIS1/chiffres ~`

2. Modifiez les droits de **votre copie** du fichier de manière à pouvoir l'exécuter.

Correction
`$ chmod u+x ~/chiffres`

3. Lancez le programme `chiffres` au premier plan. Quelle ligne de commande devez vous taper ?

Correction On peut taper
`$ ~/chiffres`
ou, depuis le répertoire maison :
`$ ./chiffres`
mais la commande
`$ chiffres`
toute seule ne fonctionne pas

4. Tuez le programme.

Correction Depuis un autre terminal, on recherche le numéro de processus de `chiffres` avec la commande `ps`, puis on tue le programme avec
`$ kill -9` n
où n est le numéro de processus trouvé.

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` ?

Correction
`$ echo $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.

Correction
`$ PATH=$PATH :`
`$ cd ; chiffres` (si cette ligne fonctionne, alors la modification est bien prise en compte)

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 le programme `chiffres` en avant-plan. Comment pouvez-vous le faire passer en arrière plan ?

Correction Visiblement, le programme `chiffres` se moque de la combinaison `Ctrl-Z`. Pour le faire passer à l'arrière-plan, il faut lui envoyer depuis un autre terminal le signal `SIGSTOP` (qu'il ne peut pas ignorer) à l'aide de la commande `kill`, puis le faire reprendre en arrière plan avec la commande `bg`.

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`, `SIGBUS`, `SIGILL`, `SIGUSR1`, `SIGTRAP`, `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`.