Introduction aux systèmes d'exploitation (IS1)

TP 6 – Variables et expansions du shell
Correction

Variables

Exercice 1 – *Variables du shell*

1. En utilisant la commande `echo $SHELL`, on affiche la valeur de la variable `SHELL` : `/usr/local/bin/bash`
2. Si on exécute la commande `NOM=prénom nom`, le shell renvoie un message d'erreur, expliquant qu'il ne trouve pas la commande `nom`. En effet, le caractère espace est interprété par le shell comme un séparateur de chaînes de caractères. Il existe deux manières d'inclure un espace dans la valeur d'une variable :
   – Echapper le caractère espace à l'aide du caractère `\`. Cela donne la commande `NOM=prénom\ nom`.
   – Utiliser des guillemets pour transformer deux chaînes de caractères en une seule. Cela donne la commande `NOM="prénom nom"`.
3. En redirigeant, à l'aide d'un tube, la sortie de la commande `set` vers l'entrée de la commande `less` (`set | less`), on vérifie aisément que la valeur de la variable `NOM` est bien listée.
4. Pour afficher `Bonjour, prénom nom !`,
   il suffit d'utiliser la commande `echo Bonjour, $NOM \!`.
5. On remarque que la valeur de la variable `NOM` n'est accessible ni dans un processus bash lancé depuis le shell courant, ni dans un terminal lancé depuis l'interface graphique. On en déduit que la valeur des variables que l'on définit dans un shell n'est accessible que depuis ce shell.

Exercice 2 – *Variables d'environnement*

1. En utilisant la commande `env | less`, on remarque que toutes les variables listées par `env` sont également listées par `set`. On en déduit donc que les variables d'environnement forment un sous-ensemble des variables du shell courant.
2. Après avoir déclaré notre variable `NOM` comme variable d'environnement grâce à le commande `export NOM`, on remarque que la valeur de la variable `NOM` est alors accessible aux shells lancés depuis le shell courant, mais toujours pas aux shells lancés depuis l'interface graphique. On en déduit que la valeur des variables d'environnement du shell courant, est accessible depuis les shells (et tous les processus) fils du shell courant.
3. En cherchant un peu parmi les variables d'environnement, on propose la variable `HOME`. En affectant à cette variable le chemin de la racine (`HOME=/`), on vérifie que cela modifie bien le comportement de la commande `cd`.

Exercice 3 – *La variable d'environnement* `PATH`

1. En utilisant la commande `echo $PATH` on obtient la valeur suivante :
   `/usr/bin`
   `:/bin`
   `:/usr/sbin`
   `:/sbin`
   `:/usr/local/bin`
   `:/usr/X11/bin`
   `:/Applications/MAMP/Library/bin/`
2. Nous utilisons la suite de commandes suivantes :

   ```
   mkdir ~/bin_perso
   echo echo \" Ceci est le résultat de l\'exécution du fichier fic. \" > ~/bin_perso/fic
   ```

3. On vérifie les droits du fichier `fic` grâce à la commande
   `ls -l ~/bin_perso/fic`
   puis, si nécessaire, on modifie les droits grâce à la commande
   `chmod u+x ~/bin_perso/fic`.
   Déplaçons-nous maintenant dans le dossier `bin_perso` (`cd ~/bin_perso`).
   Lorsqu'on exécute le fichier `fic` grâce à la commande `./fic` (ou `fic` tout court), s'affiche dans le terminal :
   `Ceci est le résultat de l'exécution du fichier fic.`
   Ce qui correspond bien au résultat de l'exécution de la commande `echo` qui s'y trouve. Cependant si l'on veut exécuter `fic` depuis un autre répertoire que `bin_perso`, il est nécessaire de donner son chemin complet.
4. Il suffit d'utiliser la commande `PATH=$PATH:/home/login/bin_perso/`. (sans espace entre `$PATH` et `:/home/...`). Une fois cette commande exécutée, la valeur de la variable `PATH` est égale à son ancienne valeur suivie de la chaînes de caractères `:/home/login/bin_perso/`
5. Dorénavant on peut exécuter `fic` en donnant seulement son nom sans le chemin complet depuis n'importe quel répertoire de l'arborescence. On en déduit que la variable `PATH` contient la liste des répertoires dans lesquels le shell va chercher l'exécutable correspondant à la commande que l'on tape.
6. Pour que la valeur de la variable `PATH` soit modifiée dans chaque nouveau processus bash, il suffit copier la ligne de commande la modifiant, dans notre fichier `.bashrc`. En effet, à chaque fois qu'un processus bash est lancé, il exécute les commandes contenues dans le fichier `.bashrc` avant de donner la main à l'utilisateur. Si l'on veut également que cette nouvelles valeur soit accessible aux proocessus fils des processus bash, il faut alors la déclarer comme variable d'environnement. Cela donne la commande :

   ```
   echo export PATH=\$PATH:/home/login/bin_perso/ >> ~/.bashrc
   ```

   Remarque : nous avons bien fait attention à échapper le $ pour que la valeur de la variable `PATH` soit expansée à chaque fois qu'est exécuté le `.bashrc` et non au moment où l'on y écrit celle ligne de commandes.

Expansions

Exercice 4 – *Parlez-vous* `bash` *? Leçon 1*

1. `echo "A             B" > leçon1`
2. `echo L\'ensemble" \{1,2,3\} est inclus dans N >>  leçon1`
3. `echo Elle "s'écria" \"Ciel, mon mari \! \" >> leçon1`
4. `echo '$HOME' = "$HOME" >> leçon1`
5. `echo '19 * 216' = $((19*216)) >> leçon1`

Exercice 5 – *Parlez-vous* `bash` *? Leçon 2*

Dans cet exercice, nous essayons de donner les réponses les plus générales pour qu'elles fonctionnent encore dans des répertoires qui contiennent plus de fichiers que ceux précisés par l'exercice. Nous utilisons donc les expressions :

1. `[mM]arin` pour désigner les fichiers `marin` et `Marin`,
2. `marin*` pour désigner les fichiers `marin` et `marine` et `marino`,
3. `[mM]ar*in` pour désigner les fichiers `marin` et `Marin` et `Martin`,
4. `Martin?*` pour désigner les fichiers `Martine` et `Martinique`,
5. `Ma*r*ine` pour désigner les fichiers `Marine` et `Martine` et `Maurine`.

Exercice 6 – *Outils pour les chaînes de caractères*

1. `${#mot}` semble désigner le nombre de caractères de la valeur de la variable `mot`.
2. `${mot :4}` semble désigner la valeur de la variable `mot`, amputée des 4 premiers caractères.
3. `${mot :4 :3}` semble désigner les trois premiers caractères de la valeur de la variable mot, à partir du cinquième caractère.
4. `${mot#unmot}` semble désigner la valeur de la variable mot à laquelle on a retiré au début, la chaîne de caractères `unmot` (on retire le *préfixe* `unmot`).
5. `${mot%long}`semble désigner la valeur de la variable mot à laquelle on a retiré à la fin, la chaîne de caractères `long`(on retire le *suffixe* `long`).

Exercice 7 – *Mon premier script*

Voilà ce que contient le notre fichier `env_perso`, que l'on aura pris soin de rendre exécutable et de placer dans le répertoire `bin_perso` (qui est déjà mentionné dans la variable `PATH`).

```
echo Votre login est \: `whoami
echo La valeur de votre variable HOME est \:  "$HOME"
echo Nous sommes aujourd'hui le \: `date`
echo Votre exécutable xemacs est \: `which xemacs`
echo $((`date "+%Y+%m+%d"`))

XEMACS=`which xemacs`
echo "${XEMACS%xemacs}" > repx
BASH=`which bash`
echo "${SHELL%bash}" > reps
cmp repx reps && echo "xemacs et bash sont dans le même répertoire"  \\
|| echo "xemacs et bash ne sont pas dans le même répertoire"

echo Le type de shell que vous utilisez est \: `basename $SHELL`
```