Introduction aux systèmes d'exploitation (IS1)

TP 7 – Variables du shell

1 Variables

En plus des usages que vous avez vus jusqu'ici, le shell propose d'autres fonctionnalités (c'est en fait un véritable langage de programmation, comme nous le verrons par la suite). En particulier, comme en Java, il est possible d'utiliser 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].

On peut assigner une valeur à la variable `var` avec l'instruction `var=`*valeur* (sans espace avant ni après le signe '='). Cette valeur peut être une chaîne de caractères ou un entier par exemple.

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 1 – *Variables du shell*

1. Vérifiez que la variable `SHELL` a déjà une valeur lorsque vous lancez votre terminal.
   Quelle est cette valeur ?
2. Déclarez une variable `NOM` contenant votre prénom **et** votre nom. Comment résoudre le problème de l'espace entre le prénom et le nom ? Affichez la valeur de `NOM` pour vérifier que l'affectation est correcte.
3. La commande `set` utilisée sans argument affiche la liste de toutes les variables dont la valeur est instanciée dans le shell courant. Utilisez la commande `less` pour faire défiler le résultat de la commande `set` page par page, à l'aide d'un tube. Vérifiez que la variable précédemment définie apparaît bien dans cette liste.
4. Écrivez une ligne de commande qui affiche "`Bonjour, Machin Chose !`" (remplacez Machin par votre prénom et Chose par votre nom) en utilisant la variable `NOM`.
5. Repérez avec `set` les variables donnant le chemin absolu de votre répertoire privé, de votre shell, ainsi que votre nom d'utilisateur. Écrivez une ligne de commande qui affiche "`Bonjour tout le monde, mon nom d'utilisateur est` *nom d'utilisateur*`, mon répertoire privé se trouve en` *réference absolu du répertoire privé*`, et mon shell est` *nom du shell* "

2 Scripts

- Un script contient des séquences de commandes telles que l'on pourrait les taper dans un terminal. Les commandes successives peuvent être séparées par des retours à la ligne (qui sont interprétés comme des points virgules).
- Tout fichier de script commence par une ligne permettant d'identifier le programme qui doit être utilisé pour l'exécuter. Dans le cas d'un script `bash`, la première ligne du fichier doit contenir : *`#!/bin/bash`* (vérifiez le chemin de votre `bash` avec la commande `which bash`)

[1] à l'exception de certaines variables spéciales du shell.

– Une ligne commençant par le caractère `#` est considérée comme un commentaire et n'est pas exécutée par le shell. Comme toujours, il est très important de bien commenter son script pour qu'il soit compréhensible pour le reste du monde.
– Par convention, l'extension d'un script est : `.sh`.
– Pour qu'un utilisateur puisse exécuter un script, il doit posséder les droits en **exécution**, mais aussi en **lecture** sur ce script (c'est un cas particulier où la lecture est nécessaire à l'exécution).
– Pour exécuter un script, on peut taper directement dans le terminal le chemin absolu du fichier, ou taper directement le chemin relatif du fichier en le faisant commencer par `./` ou taper la commande `bash` suivie d'un chemin du fichier.

Exercice 2 – *Premier script*

Écrivez un script `carapace.sh` qui affiche :

`Bonjour, mon nom est` *nom d'utilisateur*
`Attention, je vais créer un répertoire Carapace infranchissable`

puis qui crée un répertoire Carapace, et enfin qui retire les droits d'accès à ce répertoire à tout le monde. Exécutez ce script et vérifiez que tout s'est bien passé.

Paramètres. Comme vous l'avez remarqué, la plupart des commandes Unix peuvent être suivies d'un ou plusieurs paramètres, qui peuvent être des options, des noms de fichiers ou de répertoires, etc.

Il est aussi possible de passer des paramètres à vos scripts. Pour les manipuler, il existe plusieurs variables spéciales. En particulier :
– la variable `$#` contient le nombre de paramètres passés au script
– pour chaque entier `i` entre 1 et 9, la variable `$i` contient le `i`-ème paramètre
– la variable `$@` contient la liste de tous les paramètres séparés par des espaces
– la variable `$0` contient le nom du programme en cours d'exécution

Par exemple, si l'on tape la commande `./monscript.sh a "b c" d` dans un shell, alors `$#` vaudra 3, `$1` aura pour valeur `a`, `$2` aura pour valeur `b c`, et `$3` aura pour valeur `d`.

Exercice 3 – *Paramètres d'un script*

Écrivez un script `analyse.sh` qui affiche :
`Bonjour, vous avez rentré` *nombre de paramètres* `paramètres.`
`Le nom du script est` *nom du script*
`Le 3ème paramètre est` *3ème paramètre*
`Voici la liste des paramètres :` *liste des paramètres*
`Au revoir !`

3 Portée des variables

Exercice 4 – *Portée par défaut*

1. Vérifiez que la variable `NOM` contient toujours vos nom et prénom, sinon recréez-la.
2. 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 `Ctrl+D` ou `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 également aux processus fils du shell courant. De telles variables sont appelées variables *d'environnement*.

Exercice 5 – *Variables d'environnement*

1. Affichez la liste des variables d'environnement grâce à la commande `env`. Que remarquez-vous par rapport à la sortie de la commande `set` ?
2. On transforme une variable `var` en variable d'environnement grâce à la commande `export var`. Transformez `NOM` en variable d'environnement et regardez si elle est visible par un shell enfant. Inversement, exportez une variable dans un shell enfant et regardez si elle est visible par le shell parent. Qu'en déduisez-vous sur la portée des variables d'environnement ?
3. La commande `cd`, que vous connaissez bien, permet de changer de répertoire courant. Utilisée sans argument, elle 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.

4 Trouver le bon chemin

Exercice 6 – *La variable d'environnement* `PATH`

1. Quelle est la valeur de la variable d'environnement `PATH` ?
   Le caractère ' : ' sert de séparateur entre les différents chemins contenus dans la valeur de `PATH`.
2. Créez dans votre répertoire maison un répertoire que vous nommerez `bin_perso`, et créez dans ce répertoire un fichier, nommé `fic`, contenant la ligne suivante :
   ```
   echo "ceci est le résultat de l'exécution du fichier fic."
   ```
3. Vérifiez que vous avez les droits d'exécution sur le fichier `fic`. Modifiez-les si nécessaire.
   Exécutez-le depuis le répertoire `bin_perso` en tapant `~/bin_perso/fic` ou de manière écquivalente `./fic`, et décrivez le résultat.
   Essayez maintenant de taper `fic` . Cela vous permet-il d'exécuter le fichier `fic` ? Essayez maintenant de taper `fic` depuis un autre répertoire. Cela vous permet-il d'exécuter le fichier `fic` ?
4. Ajoutez maintenant à votre variable `PATH` le chemin `~/bin_perso` . Vous devrez pour celà affecter à la variable `PATH` son ancienne valeur à laquelle vous ajouterez la chaîne de caractères
   ```
   :~/bin_perso
   ```
5. Placez-vous maintenant dans votre répertoire maison, et tapez `fic`. Que se passe-t-il ? Qu'en déduisez-vous sur l'utilité de la variable `PATH` ?
6. Comme vous avez dû le remarquer dans l'exercice précédent, les valeurs des variables d'environnement sont accessibles à tous les processus fils du shell courant. Comment faire pour que le chemin `~/bin_perso/` soit inclus dans la valeur de la variable `PATH`, dès qu'on lance un nouveau terminal ?

5 Valeurs de retour

On revient sur la notion de valeur de retour vue au TP6. On rappelle qu'une commande renvoie une valeur dans la variable `$?` pour indiquer si l'exécution s'est déroulée normalement (valeur 0) ou pas (autre valeur). Certaines commandes utilisent cette variable pour renvoyer carrément leur résultat. C'est le cas de la commande `test` (parcourez rapidement sa page de manuel, vous en aurez besoin).

On peut forcer un script à se terminer et renvoyer une valeur donnée (par exemple ici 18) par la commande `exit 18`

Vous pourrez utiliser les connecteurs `&&`, `||` et `;` vus au TP6. Le parenthésage se fait par des accolades entourées d'espaces. Attention, les espaces sont importants, tout comme l'utilisation des accolades et non des parenthèses.

Exercice 7 – *Concaténateur*

Écrivez un script `concat.sh` qui prend en paramètre 2 mots et fait ce qui suit.

- si l'utilisateur rentre autre chose que 2 paramètres, indique à l'utilisateur qu'il doit rentrer exactement 2 paramètres, et quitte en renvoyant une erreur de code 18
- sinon le script calcule dans une variable `CONCAT` la concaténation des 2 mots rentrés puis affiche le résultat

Testez votre script pour vérifier son comportement sur chacun des cas ci-dessus.

Exercice 8 – *Test de fichiers 1*

Écrivez un script `modif-droits1.sh` qui prend en paramètre un chemin et :

- si l'utilisateur rentre 0 ou plusieurs paramètres, indique à l'utilisateur qu'il doit rentrer exactement un paramètre, et quitte en renvoyant une erreur de code 18
- si le chemin ne désigne aucun fichier dans le SGF, affiche "`Attention: le fichier` *nom du fichier* `n'existe pas`", et quitte avec le code 19
- si le chemin désigne un fichier régulier, ajoute les droits en lecture pour tous les utilisateurs sur ce fichier et affiche "`Droit en lecture ajouté sur le fichier` *nom du fichier*"
- si le chemin désigne un répertoire, retire les droits en écriture pour tous sur toute la sous-arborescence et affiche "`Droit en écriture retiré sur la sous-arborescence de` *nom du répertoire*"
- si le chemin désigne un autre type de fichier, ne fait rien.

Testez votre script sur une arborescence qui ne craint rien.

Exercice 9 – *Test de fichiers 2*

Écrivez un script `modif-droits2.sh` qui prend en paramètre un chemin et :

- si l'utilisateur rentre 0 ou plusieurs paramètres, indique à l'utilisateur qu'il doit rentrer exactement un paramètre, et quitte en renvoyant une erreur de code 18
- si le chemin ne désigne aucun fichier dans le SGF, affiche "`Attention: le fichier`*nom du fichier* `n'existe pas`", et quitte avec le code 19
- si le chemin désigne un fichier régulier avec droit en lecture, affiche le contenu du fichier,
- si le chemin désigne un fichier avec droit en exécution, retire ce droit et affiche "`Droit en exécution retiré sur le fichier` *nom du fichier*"

Testez votre script sur des fichiers qui ne craignent rien.