Variables et environnement

Un shell script s'exécute comme un processus

Un script qui s'exécute, interprété par un shell, est un processus à part entière. Il bénéficie donc d'un espace de variables pour mémoriser des informations. Un processus, comme tous les autres processus s'exécute dans le contexte du système d'exploitation. Ce dernier maintient des données qui sont accessibles à ce processus, mais également à tous les autres processus qui s'exécutent dans le même contexte. On parle d'**environnement**.

L'environnement permet de définir des paramètres d'exécution qui sont propres à la situation locale d'exécution du programme.

Les variables d'environnement vu du shell

Dans bash (et les autres programmes), on peut définir des variables d'environnement. Ces variables sont stockées comme des chaînes ASCII (même s'il s'agit de nombres). La définition de ces variables est standard dans les systèmes d'exploitation basés sur POSIX et est propre au modèle général des processus. Elles peuvent donc être définies par des programmes écrits en C ou dans n'importe quel langage.

Les variables mises en place dans l'environnement sont maintenues par le système d'exploitation. Elles peuvent être visibles par les processus suivants qui sont exécutés dans le même contexte. Il faut pour cela les "exporter".

Spécificité du shell

Par rapport aux autres programmes (C ou autres langages), le shell confond ses propres variables avec l'environnement. Ainsi, écrire :

```
$>maVariable="une variable d'environnement"
```

définit une variable du script courant qui est une variable d'environnement de ce script.

Définition des variables en shell

Une variable est donc définie par une affectation d'une valeur à un symbole par une syntaxe comme ci-dessus. On doit noter :

- Que la syntaxe est valable comme une ligne d'un source de script shell, mais aussi, directement écrite dans la console. Un script est donc une séquence de syntaxe qui **correspond exactement** à ce qu'un opérateur pourrait taper sur la console système.
- Qu'il ne FAUT PAS mettre d'espaces autour du = (IMPERATIF)
- Que le nom de variable (symbole) doit être un "token" légitime, donc constitué (à peu près) avec les mêmes règles qu'une variable C.
- Que la valeur est une valeur littérale ou une expression assimilable à une chaîne de caractères.

Affichage des variables

Pour afficher une variable existante, on utilise la commande "echo". Cette commande demande l'affichage du contenu du symbole que représente le nom de la variable. Mais ce contenu doit être "totalement interprété". On indique ceci par le marquage **$** avant le nom du symbole.

Note : on retrouve cette commande en Php.

```
$>echo $maVariable
une variable d'environnement
```

Autrement dit la commande précédent doit se lire :

"affiche l'expression texte résolue du contenu de maVariable".

Ecrire :

```
$>echo maVariable
maVariable
```

ne fait qu'afficher l'argument de ligne de commande de la commande "echo", c'est-à-dire, dans ce cas, le mot "maVariable".

Note : on note que si cet argument est une phrase avec des espaces, il faut "quoter" l'argument :

```
$>echo "ma variable"
ma variable
```

Substitutions dans une expression variable

Le principal mode d'interprétation d'un symbole se fait par des opérations de "substitution". Les variables shell étant des chaînes de caractère, il est possible de faire figurer dans le flux de caractère des appels à des variables existantes.

Le processus d'interprétation substitue les symboles identifiés par leur valeur. Ce procédé est FONDAMENTAL, car il apparaît dans LA MAJORITE des technologies actuelles.

Exemple :

```
$>echo "ouah ! $maVariable"
ouah ! une variable d'environnement
```

de même, plus radicalement, que :

```
$>suffixe='nement'
$>echo environ$suffixe
environnement
```

En essayant avec des simples quotes, la réponse change complètement :

```
$>echo 'ouah ! $maVariable'
ouah ! $maVariable
```

C'est donc qu'il existe plusieurs contexte d'interprétation d'une expression chaîne de caractères. Il en existe à vrai dire trois :

- la chaîne littérale : marquée par des ' (simple quote)

  Ne substitue aucune variable ni séquence d'échappement (ajouter l'option -e pour ces dernières).

  Exemple :

  ```
  $>echo 'sans $interpretation aucune \n'
  sans $interpretation aucune \n
  ```

- la chaîne substituée : marquée par " (double quote)

  Substitue les noms de variables, mais pas les séquences d'échappement (ajouter l'option -e)

  Exemple :

  ```
  $>interpretation='une interprétation'
  $>echo "avec $interpretation aucune \n"
  avec une interprétation aucune \n
  ```

- la chaîne exécutée : marquée par ` (antiquote)

  Est intégralement substituée par le résultat de la commande qu'elle décrit. Substitue les variables.

  Exemple :

  ```
  $>cmd=ls
  $>echo `$cmd`
  C essais exercices docs
  ```

Substitutions ambigues

Examinons le code suivant :

```
$>suffixe='nement'
$>echo environ$suffixeal

?? environnemental

?? environal

?? environ
```

Malheureusement c'est le troisième résultat qui est produit. La fonction de substitution du shell ne sait pas "arrêter" le nom d'une variable à une variable

existante. Potentiellement, toutes existent. Il y a donc pour lui une ambiguité car la variable qu'il va chercher à substituer est une hypothétique variable $suffixeal qui n'existe pas (une variable qui n'existe pas vaut la chapine vide).

Alors comment on s'en sort ?

Réponse : Par le marquage explicite des limites de variables :

```
$>suffixe='nement'
$>echo environ${suffixe}al
environnemental
```

Il est **très conseillé** de généraliser cette pratique.