Cours : manipulation des chaînes et des variables

Troncature de valeurs chaînes

Il est très courant, en script shell, à avoir à composer et segmenter (*splitting* & *chopping*) des chaînes de caractères. Par exemple, une grande part des traitements a affaire à des noms de fichiers absolus qui sont des chaînes structurées. D'autres paradigmes informatiques s'expriment également sous forme de chaînes structurées. En voici quelques exemples :

Exemples de valeurs chaînes structurées

- /home/etudes1/sources/C/monTP.c
- http://moodle.vfedu.fr/chemin/ressource.html
- moodle.vfedu.fr
- pserver:cvs.services.eisti.fr;user=root;password=XXXXX:/repos
- {{1,2,3,4},{"rouge","bleu","vert","jaune"}}
- {'john' => 'Jean', 'pete' => 'Pierre', 'jack' => 'Jacques', 'will' => 'Guillaume'}
- etc.

Les outils de troncature et d'extraction de chaîne

Le shell fournit certains outils syntaxiques puissants pour effectuer un certain nombre d'opérations sur les chemins du système de fichier et les chaînes de façon plus générale, en vue d'en récupérer un morceau, un préfixe ou un suffixe.

Commandes spécifiques aux chemins de fichiers

Les chemins considérés ici le sont essentiellement pour leur nature chaîne de caractère et non pour le fichier ou le répertoire qu'ils désignent.

basename

La commande `basename` permet d'obtenir le nom "local", on dit aussi "canonique" de l'objet du système de fichier qu'il représente (fichier ou répertoire).

```
$>basename /etc/httpd/httpd.conf
httpd.conf
$>basename /home/users/vfremaux
vfremaux
```

dirname

La commande `dirname` obtient la portion de chaîne opposée à la commande `basename`. Elle récupère la partie chemin en éliminant le nom canonique. On note que le séparateur "/" qui sépare les deux parties n'est pas relevé par dirname.

```
$>dirname /etc/httpd/httpd.conf
/etc/httpd
$>dirname /home/users/vfremaux
/home/users
```

On peut facilement et avantageusement récupérer le résultat de cette commande dans une variable :

```
$>myDIR=`dirname /home/users/vfremaux/profile.conf`
$>echo $myDIR
/home/users/vfremaux

$>echo ${myDIR}/unautrefichier.txt
/home/users/vfremaux/unautrefichier.txt
```

Vous remarquez que nous avons utilisé la substitution dite "chaîne exécutée" à antiquotes. Nous pouvons obtenir le même résultat avec la syntaxe suivante :

```
$>myDIR=$(dirname /home/users/vfremaux/profile.conf)
$>echo $myDIR
/home/users/vfremaux
```

Voici un autre exemple qui récupére dans une variable les fichiers commençant par "pa" dans le répertoire standard de configuration :

```
$>myFILES=$(ls /etc | grep ^pa)
$>echo $myFILES
pam.conf pam.d pam_ldap.conf pam_ldap.conf.old pango papersize passwd passwd-
```

Troncatures de "pro"

La séparation des chemins ne suffit pas toujours à faire ce dont on a besoin. Il est des cas où des outils plus puissants de découpage et d'extraction sont nécessaires. Nous pouvons alors tirer avantage de la richesse de la syntaxe d'**expansion de variable** : ${...}.

Note : ne pas confondre expansion de variable ${...} et syntaxe de substitution $(...).

Voici quelques exemples qui introduisent les possibilités de ces syntaxes :

```
$>$myVAR="cincinnati.gif"
$>echo ${myVAR##*ci}
nnati.gif

$>echo ${myVAR#*ci}
ncinnati.gif
```

Au passage, voici une vue de Cincinnati, Ohio

Extraction de suffixe (par élimination de préfixe)

L'opérateur ## utilisé dans le premier exemple applique à une variable un filtre de préfixe. Un filtre de préfixe supprime un certain préfixe dans la valeur de la variable. Ce préfixe est déterminé par une expression variable, un motif utilisant des jokers. * représente ici n'importe quelle chaîne.

Le motif "`*ci`" est donc capable, en principe, de repérer deux positions dans la chaîne "`cincinnati.gif`" :

```
cincinnati.gif // * vaut pour "cin"
cincinnati.gif // * vaut pour ""
```

L'une correspond à un motif "court", l'autre à un motif plus long. Dans les deux cas la règle énoncée par le motif "`*ci`" est respectée.

L'opérateur # permet de réaliser ce filtrage au plus court, il produira le suffixe résultant "`ncinnati.gif`". ## le réalise au plus long (*greedyness* = gourmandise du motif), il produit donc le suffixe "`nnati.gif`".

Comme procédé mnémotechnique, on peut se rappeler que # est plus court que ##, donc # recherche une correspondance plus courte que ##. La justification de choix du caractère # pour cet opérateur n'est compréhensible que par les américains : l'organisation du clavier US place le caractère "#" juste à gauche du caractère "$" et "%" juste après le caractère "$" : on peut en déduire le sens.

Extraction de préfixe (par élimination de suffixe)

Dans la même logique, les opérateurs % et %% permet d'effectuer un filtrage de suffixe pour récupérer un certain préfixe. Cet opérateur est très utilisé pour traiter les extensions de fichier.

```
$>$myDIST="distribution.tar.gz"
$>echo ${myDIST%%.*}
distribution
```

```
$>echo ${myDIST%.*}
distribution.tar
```

Vous remarquerez que le motif a été construit dans le sens inverse (pensé à partir de la fin).

Note : si le motif est une chaîne constante à partir de la fin, il n'est pas nécessaire de mentionner une * :

```
$>$myDIST="distribution.tar.gz"
$>echo ${myDIST%%.gz}
distribution.tar
```

Extraction généralisées

Dans des cas encore plus complexes, il est possible de récupérer une partie centrale d'une chaîne. La syntaxe utilise une définition classique d'une sous-chaîne à partir de deux informations : la position du caractère de début et la longueur de la séquence qui la constitue.

```
$>$myVAR="projet pluridisciplinaire"
$>echo ${myVAR:7:5}
pluri
```

ATTENTION : Ces opérateurs ne sont pas combinables entre eux.

Un exemple simple d'application

Voici un petit script qui teste si un fichier est une archive "tar".

```
---- mytar.sh -----
#!bin/bash

if [ "${1##.}" = "tar" ]
   echo "c'est une archive"
else
   echo "ce n'est apparemment pas une archive"
fi
```

Note : on remarque, au début d'un fichier script, la mention de l'interpréteur qui doit être invoqué pour interpréter ce source. Cette technique est généralisée dans les systèmes POSIX. Cette convention est commune pour n'importe quel langage de script qui dispose d'un interpréteur de commandes adéquat (perl, php, etc.)

Une fois ce script écrit dans le fichier `mytar.sh`, il faut rendre ce fichier exécutable. Sinon, il s'agit d'un simple fichier texte qui ne peut être lancé comme commande.

```
$>chmod ug+x mytar.sh

(ajoute l'exécution à l'utilisateur courant et au groupe)
ou encore

$>chmod 755 mytar.sh
```

```
(fait la même chose mais en marquant directement le code de droit)
```

Essayez le script en donnant un nom de fichier :

```
$>./mytar.sh unearchive.tar
c'est une archive
```

Commentaires sur le script

Le script ci dessus utilise une notion que vous verrez plus tard : le passage de variables. Dites-vous ici que la variable $1 ou, sous sa forme desambigue ${1} est le premier paramètre qui suit le nom de la commande-script dans la ligne de commande.