Extraction de sous-chaîne et recherche de motifs

Cadre : traitement des variables du shell

Le shell permet aussi :

- la suppression automatique ou le remplacement de la plus petite ou la plus longue chaîne de caractères choisie ;
- de calculer la longueur d'une chaîne de caractères ;
- de réaliser des opérations arithmétiques sur des variables et des chaînes de caractère ;
- D'utiliser des fonctions.

Attention : ce qui suit est apparu avec la version 2 de Bash

1) suppression d'une portion initiale

Syntaxe :

${variable#*motif} : plus petite chaîne + motif supprimé de variable ${variable##*motif} : plus longue chaîne + motif supprimé de variable

Exemples

1.1)

```
eric@tomate:~$ ecole=utbmutbm
eric@tomate:~$ echo ${ecole#*t}
bmutbm
eric@tomate:~$
```

Commentaire : la plus petite chaîne de caractère contenant
(n'importe quelle chaîne, y compris la chaîne vide) + t
a été supprimée. **Donc "ut" a été supprimé**.

1.2)

```
eric@tomate:~$ ecole=utbmutbm
eric@tomate:~$ echo ${ecole##*t}
bm
eric@tomate:~$
```

Commentaire : ici, c'est la **plus longue** qui est **supprimée**.
Il ne reste donc que "bm"

2) suppression d'une portion finale

Syntaxe :

```
${variable%motif} : variable est raccourcie de
la plus petite chaîne contenant motif

${variable%%motif} : variable est raccourcie de
la plus longue chaîne contenant motif
```

Exemples

2.1)

```
eric@tomate:~$ adresse=root@sa_machine@sa_machine
eric@tomate:~$ echo ${adresse%@sa*}
root@sa_machine
```

Commentaire : @sa_machine a été supprimé, il en reste 1 : la plus courte chaîne contenant @sa* a bien été supprimée.

2.2)

```
eric@tomate:~$ echo ${adresse%%@sa*}
root
eric@tomate:~$
```

La plus longue chaîne contenant @sa*, ici @sa_machine@sa_machine a été supprimée.

3) Extraction de n caractères, à partir du p-ième

Syntaxe :

${variable:n:p}

ATTENTION : le premier caractère de la chaîne est numéroté 0

Exemple :

```
eric@tomate:~$ mon_nom=eric\ bachard
eric@tomate:~$ echo ${mon_nom:5:10}
bachard
eric@tomate:~$
```

Remarque : Il n'y a pas 10 lettres dans "bachard",
donc les premières trouvées, parmi 10, ont été affichées.

Exemple 1) Extraction d'un nom de machine :

```
eric@tomate:~$ adresse=tomate@beorg.fr
eric@tomate:~$ echo ${adresse%%@*}
tomate
eric@tomate:~$
```

Exemple 2) Extraction d'un nom de domaine :

```
eric@tomate:~$ echo ${adresse##*@}
beorg.fr
eric@tomate:~$
```

Exemple 3) Eliminer l'extension éventuelle d'une extension de fichier :

```
eric@tomate:~$ fichier=module1.c
eric@tomate:~$ echo ${fichier%%.*}
module1
eric@tomate:~$
```

Conseil : %% plutôt que % , car il peut exister des extensions avec plusieurs "**.**"

Exemple 4) Renommage de tous les fichiers dont l'extension est .TGZ en .tar.gz

```
eric@tomate:~/Linux/cours$ for i in 1 2 ; do touch fich$i"."TGZ ; done
eric@tomate:~/Linux/cours$ ls -l ./*.TGZ
-rw-r--r--    1 eric     eric            0 jun 20 21:49 ./fich1.TGZ
-rw-r--r--    1 eric     eric            0 jun 20 21:49 ./fich2.TGZ
```

Création des deux fichiers .TGZ

```
eric@tomate:~/Linux/cours$ for i in *.TGZ ; do mv $i ${i%TGZ}tar.gz
> done
eric@tomate:~/Linux/cours$ ls -l ./fich*.*
-rw-r--r--    1 eric     eric            0 jun 20 21:49 ./fich1.tar.gz
-rw-r--r--    1 eric     eric            0 jun 20 21:49 ./fich2.tar.gz
eric@tomate:~/Linux/cours$
```

Il ne reste plus de *.TGZ : ils ont été renommés en .tar.gz !

4) Calcul de la longueur d'une chaîne

Syntaxe :

```
${#variable}
```

Exemple :

```
eric@tomate:~$ nom=dictionnaire
eric@tomate:~$ echo ${#nom}
12
eric@tomate:~$
```

Remarque : **une variable non définie a une longueur nulle**

Exemple :

```
eric@tomate:~$ echo ${#existe_pas}
0
eric@tomate:~$
```

Attention :

```
eric@tomate:~$ echo $UID
1000
```

1000 = valeur de l'UID d'un utilisateur

```
eric@tomate:~$ echo ${#UID}
4
eric@tomate:~$
```

4 = nombre de chiffres contenus dans "1000"

5) Calcul arithmétique

Syntaxe :

$((opérations))

Attention : $[opérations] est obsolète...

- Mêmes régles de priorité qu'en C, mêmes opérateurs...
- On préfère toutefois abuser des parenthèses pour éviter les pièges de règles de priorité des opérations.
- Inconvénient : plus lourd
- Avantage : plus facile à débugguer

Exemples :

1) opération simple :

```
eric@tomate:~$ echo $(((3 * 2) - (6 + (4 * 2 ))))
-8
```

2) utilisation d'une variable x intermédiaire :

```
eric@tomate:~$ echo $((x = 5 - 10 ))
-5
eric@tomate:~$ echo $x
-5
```

3) opérations sur variables :

```
eric@tomate:~$ echo $((y = x * x * x + 1))
-124
eric@tomate:~$ echo $y
-124
eric@tomate:~$
```

6) Invocation de commande

- On place dans liste le résultat de ls -l :

```
eric@tomate:~/Linux$ liste=$(ls -l)
eric@tomate:~$
```

- Que l'on peut afficher (pas très bien présenté) :

```
eric@tomate:~/Linux$ echo $liste
total 8 drwxr-xr-x 3 eric eric 4096 jun 15 21:49 cours drwxr-xr-x 4 eric
eric 4096 jun 19 21:20 scripts
eric@tomate:~$
```

- Pour améliorer la présentation :

```
eric@tomate:~/Linux$ echo "$liste"
total 8
drwxr-xr-x  3 eric  eric      4096 jun 15 21:49 cours
drwxr-xr-x  4 eric  eric      4096 jun 19 21:20 scripts
eric@tomate:~/Linux$
```

7) Fonctions et portées des variables

- Définition d'une fonction : voir poly cours

Syntaxe :

```
function ma_fonction ( )
{
...instructions
}
```

- Une variables définie par variable=valeur (sans autres précisions) est accessible (définie) dans l'ensemble du processus en cours.

- Même chose pour les fonctions.

Ce qu'il faut retenir des fonctions :

- Une fonction, dès lors qu'elle a accès au contenu d'une variable, peut la lire (accès en lecture) et/ou la modifier (accès en écriture).

- On peut déclarer localement, par rapport à une fonction, une variable. Dans ce cas, la variable en question n'est accessible qu'aux sous-processus et/ou sous fonctions.

- Si cette variable existait déja : une variable de même nom est créée, masquant la précédente, jusqu'à ce que la fonction en question se termine.

Exemples :

1) accès en lecture :

```
eric@tomate:~$ var=123        #On déclare var, avec la valeur 123
eric@tomate:~$ function affiche_var () { echo $var; }
eric@tomate:~$ affiche_var
123
```

Remarque : la fonction affiche_var est accessible avec la complétion !!

```
eric@tomate:~$ var=123
eric@tomate:~$ function modifie_var () {  var=123; }
eric@tomate:~$ var=456
eric@tomate:~$ echo $var
456
eric@tomate:~$ modifie_var
eric@tomate:~$ echo $var
123
eric@tomate:~$
```

FIN du cours initiation à Unix...

Bibliographie :

1) **Langages de script sous Linux**
Christophe BLAESS
Editions Eyrolles 2002

2) **Le Système Linux**
Matt Welsh, Matthias Kalle Dalheimer & Lar Kaufman
Editions O'Reilly 1998 2ème édition 1998

3) **Linux in a Nutshell**
Jessica P. Hekman
Editions O'Reilly 1997

4) **Unix : le tout en poche**
Dave Taylor & James C. Armstrong J
r
Editions Simon and Shuster MacMillan 1998