Chapitre 4
Les bases de la programmation shell

1. Présentation

Ce chapitre présente les fonctionnalités qui composent les bases de la programmation shell. Lorsque les scripts donnés en exemple ne sont pas compatibles avec l'interpréteur utilisé par le lecteur, nous invitons ce dernier à récupérer les exemples à partir de l'espace de téléchargement, qui sont fournis pour différents shells présentés dans ce livre.

2. Les variables utilisateur

Le shell permet de définir ou redéfinir des variables qui conditionnent l'environnement de travail de l'utilisateur. Il est également possible de définir d'autres variables, dites **variables utilisateur**, qui vont permettre de stocker des informations qui seront nécessaires à l'exécution d'un script.

2.1 Nommer une variable

Voici les règles à utiliser pour attribuer un nom à une variable :

– Le premier caractère fait partie de l'ensemble [a-zA-Z_].
– Les caractères suivants sont pris dans l'ensemble [a-zA-Z0-9_].

2.2 Définir une variable

Une variable est définie dès qu'elle est initialisée. Le contenu d'une variable est considéré par le shell comme une suite de caractères.

2.2.1 Affecter une valeur à une variable

Exemples

```
$ var1=mot1
$ echo $var1
mot1
$
```

Il ne faut pas mettre d'espace autour du symbole d'affectation : dans l'exemple suivant, le shell interprète **var1** comme la commande à lancer, **=** et **mot1** comme les deux arguments de la commande **var1**. Autrement dit, il n'interprète pas le signe **=** comme symbole d'affectation :

```
$ var1 = mot1
ksh: var1: not found
$
```

2.2.2 Affecter une valeur contenant au moins un espace

L'espace doit être protégé car c'est un caractère spécial du shell (séparateur de mots sur la ligne de commande).

Exemples

```
$ var2='mot1 mot2 mot3' #CORRECT
$ echo $var2
mot1 mot2 mot3
$ var2=mot1 mot2 mot3 #INCORRECT
ksh: mot2: not found
$
```

2.2.3 Variable indéfinie

Une variable qui n'a jamais été initialisée est vide.

Exemple

```
$ echo $vide

$
```

2.2.4 Retirer la définition d'une variable

La commande interne **unset** permet de rendre une variable indéfinie. L'option **-v** (option POSIX : **-v** comme variable) n'est pas documentée dans le manuel du ksh88, mais elle fonctionne.

posix	ksh93	bash

```
$ unset -v variable
```

ou

bourne	posix	ksh(88 et 93)	bash

```
$ unset variable
```

Exemple

Définition d'une variable **var** :

```
$ var=12
$ echo $var
12
```

Elle apparaît dans la liste des variables définies au niveau du shell :

```
$ set | grep var
var=12
```

La définition de la variable est retirée :

```
$ unset var
```

La variable est indéfinie :

```
$ echo $var

$ set | grep var
$
```

2.2.5 Isoler le nom d'une variable

Il faut faire attention en concaténant le contenu d'une variable et d'une chaîne de caractères à ce que le shell interprète correctement le nom de la variable.

Exemple

Pour le shell, le caractère _ fait partie du nom de la première variable :

```
$ fic=resu
$ datejour=20220306
$ newfic=$fic_$datejour
$ echo $newfic
20220306
```

Pour le shell, la première variable se nomme **fic_** (puisque le caractère souligné est autorisé à l'intérieur d'un nom de variable). Celle-ci est donc substituée par sa valeur (donc vide), puis concaténée avec le contenu de la variable **datejour**.

Pour faire comprendre au shell quels sont les caractères qui font partie du nom de la variable, il faut entourer le nom de cette dernière avec des **{}**.

Exemple

```
$ fic=resu
$ datejour=20220306
$ newfic=${fic}_$datejour
$ echo $newfic
resu_20220306
```

2.2.6 Variables numériques

Déclaration explicite d'un nombre entier

ksh	bash

La commande interne **typeset** permet de déclarer explicitement une variable comme étant un entier. Cette déclaration est facultative mais elle permet d'avoir un contrôle sur la valeur stockée et rend les calculs plus rapides.

Exemple

Déclaration, initialisation et affichage :

```
$ typeset -i nb=1
$ echo $nb
1
```

Contrôle sur la valeur affectée en ksh :

```
$ nb=a
ksh: a: bad number
```

Remarque

*En bash, la commande **typeset** est un synonyme de la commande **declare**.*

Déclaration explicite d'un nombre flottant

ksh93

Seul le ksh93 implémente l'arithmétique des nombres flottants.

Déclaration d'un nombre flottant avec une précision de 3 :

```
$ typeset -F3 tva=0.196
```

Cette fonctionnalité est détaillée à la section Arithmétique sur les flottants de ce chapitre.

2.2.7 Variables complexes

ksh93

Le ksh93 offre la possibilité de créer des variables complexes.

Exemple

La variable **user** contient une propriété **nom** qui contient la valeur **christie** et une propriété **uid** qui contient la valeur **203**.

```
$ user=( nom=christie uid=203 )
```

Afficher la variable **user** :

```
$ echo  $user
Nom : (
        nom=christie
        uid=203
)
```

Afficher une propriété particulière :

```
$ echo ${user.nom}
christie
$ echo ${user.uid}
203
```

Ce genre de variable est similaire au principe de la structure du langage C (ou d'une classe ne contenant que des propriétés).

Pour stocker des informations complexes, il est également possible d'utiliser des tableaux associatifs ksh/bash (cf. Aspects avancés de la programmation shell - Tableaux associatifs).

2.2.8 Variables accessibles en lecture seule

Définir une variable non modifiable :

posix	ksh	bash

```
$  readonly i=1
$  i=2
-bash: i : variable en lecture seule
$
```

Les commandes **`typeset -r`** (ksh, bash) et **`declare -r`** (bash) ont la même fonctionnalité que **`readonly`** mais ne sont pas normalisées **`POSIX`**.

bourne

```
$  i=1
$  readonly i
$
```

Liste des variables non modifiables :

posix	ksh	bash

```
$ readonly -p
```

bourne	ksh	bash

```
$ readonly
```

2.3 Substitution de variables

Le shell offre la possibilité d'attribuer une valeur par défaut aux variables non initialisées, ou au contraire, initialisées.

Expression ${variable:-valeur}

– Si la variable n'est pas vide, l'expression est substituée par **`$variable`**.

– Si la variable est vide, l'expression est substituée par **`valeur`**.

Exemple

```
$ fic=/tmp/christie.log
$ echo "Le fichier traité sera: ${fic:-/tmp/default.log}"
Le fichier traité sera: /tmp/christie.log
$ unset fic
$ echo "Le fichier traité sera: ${fic:-/tmp/default.log}"
Le fichier traité sera: /tmp/default.log
$ echo $fic

$
```