



UNIX

Cours 9 : Programmation SHELL (bash) (Partie 3)

Filipe Vasconcelos¹

¹ESME, Lille, filipe.vasconcelo@esme.fr

1 Tableaux

2 Fonctions



AAV4

Produire, tester et vérifier le résultat des scripts SHELL (en bash) pour réaliser des tâches algorithmiques simples à complexes



1 Tableaux

2 Fonctions



bash deux types de tableaux

- ❑ les tableaux « classiques » : les éléments sont indicés par des entiers
- ❑ les tableaux « associatifs » : les éléments sont indicés par des chaînes de caractères



Les tableaux « classiques »

Déclaration :

```
tableau=("valeur0" "valeur1" "valeur2")
```

Pour accéder à une case du tableau

```
 ${tableau[2]} #Remarque: le 1er indice du tableau est 0
```

Pour définir ou modifier l'élément d'un tableau :

```
tableau[3]="nouvelle valeur"
```

L'ensemble du contenu du tableau :

```
echo "${tableau[*]}" #Remarque: intégrité des chaînes non conservée
echo "${tableau[@]}" #Remarque: intégrité des chaînes sont conservée
```

Pour récupérer le nombre total d'élément du tableau

```
#!/bin/bash
tab=("lundi" "mardi" "mercredi")
echo "initialisation"
echo "première valeur: ${tab[0]}"
echo "deuxième valeur: ${tab[1]}"
echo "troisième valeur: ${tab[2]}"

echo
echo "affichage de tous les éléments"
echo "${tab[*]}"

echo
echo "affichage du nombre d'éléments"
echo "${#tab[@]}"

tab[0]="vendredi"
echo
echo "modification"
#exercice afficher les valeurs avec une boucle for
```

```
tab=("one two" three)

for element in "${tab[@]}"
do
    echo ${element}
done

for element in ${tab[@]}
do
    echo ${element}
done
```

one two
three

one
two
three

Attention certaines syntaxes de parcours de tableau peuvent amener à des confusions, notamment l'utilisation de \${tab[*]}. La meilleure parade est de tester son script.



Autres opérations sur des tableaux classiques

Liste de tous les indices définis

```
echo "${!tab[@]}"
```

Copie d'un tableau

```
tabcopy=( "${tab[@]}" )
```

Ajout d'éléments à un tableau

```
tab+=("four" "five")      # à la fin
tab=("zero" "${tab[@]}") # en tête
```

Suppression d'un tableau

```
unset tab
```



Tableau associatif

La déclaration se fait avec la commande interne declare

```
declare -A tabasso=([fr]=Paris [it]=Rome [pt]=Lisbonne)
```

Quelques opérations

Liste de toutes les clés d'un tableau associatif

```
echo "${!tabasso[@]}"
```

Pour parcourir un tableau associatif

```
for cle in "${!tabasso[@]}"
do
    echo ${tabasso[$cle]}
done
```

Ajout d'un élément

Exercice 1

Écrire un programme shell carte qui affiche le nom d'une carte tirée au hasard d'un jeu de 32 cartes. On utilisera deux tableaux : un tableau couleur et un tableau valeur. On utilisera la variable d'environnement RANDOM.

Exercice 2

Écrire un programme shell distrib qui crée un paquet de 5 cartes différentes tirées au hasard parmi un jeu de 32 cartes et affiche le contenu du paquet.



1 Tableaux

2 Fonctions



- ❑ Une fonction est un ensemble d'instructions permettant d'effectuer plusieurs tâches avec des paramètres d'entrée différents.
- ❑ La fonction permettra de rendre le programme bash plus lisible et structuré. Ainsi, il facilitera le développement de programme.
- ❑ Une fois que la fonction est définie, il est possible d'appeler cette fonction autant de fois que vous le souhaitez dans votre script.
- ❑ Une fonction doit être déclarée avant d'être utilisée.
- ❑ Les appels recursifs sont possibles.



Déclaration :

```
function foo{  
    # Exemple de fonction  
    echo "dans la fonction foo"  
}
```

ou

```
foo(){  
    echo "dans la fonction foo"  
}
```

Appel :

```
#simple appel de la fonction  
foo
```

- ❑ Les fonctions peuvent récupérer des arguments qui leur sont passés et renvoyer un code de sortie au script par l'instruction `return`.
- ❑ On accède à la valeur renournée par la variable prédéfinie `$?`
- ❑ La fonction se réfère aux arguments passés par leur position (comme s'ils étaient des paramètres positionnels), c'est-à-dire `$1`, `$2` et ainsi de suite.

Exemple :

```
max() {  
    if [ "$1" -ge "$2" ]  
    then  
        return $1  
    else  
        return $2  
    fi  
}  
read -p "Donnez deux nombres" n1 n2  
max ${n1} ${n2}  
echo "Le max de ${n1} et ${n2} est $?"
```



- La variable globale est définie à l'extérieur de la fonction. Elle est visible partout dans le script
- La variable locale est définie à l'intérieur de la fonction. Elle n'est visible que dans la fonction

Exemple :

```
c=0
function carre
{
    a=$1
    c=$((a*a))
    return
}
carre 9
echo "Le carré de 9 est ${c}"
```



Utilisation très courante

Affichage d'un help pour l'utilisateur

```
#!/bin/bash

usage() {
    echo "Usage : $0 [a] [n]"
    echo "a : une chaîne de caractères"
    echo "n : nombre entier"
    exit 1
}

[ $# -ne 2 ] && usage

...
<suite du script>
...
exit 0
```

Exercice 1

Écrire un programme shell fact qui fait appel soit à la version itérative ou recursive pour le calcul de la factorielle d'un nombre passé en paramètre du script. On écrira deux fonctions bash factiter et factrecur. Le script fact acceptera deux arguments le premier est une chaîne de caractère définissant la version à utiliser et le second le nombre dont on souhaite calculer la fonctionnel.

