script bash : Tableaux

- Objet : script bash : Tableaux
- Niveau requis :
  débutant, avisé
- Commentaires : Fix Me!
- Débutant, à savoir : Utiliser GNU/Linux en ligne de commande, tout commence là !.
- Suivi :
  - Création par Hypathie le 18/03/2014
  - Testé par Hypathie Juin 2014
- Commentaires sur le forum : Lien vers le forum concernant ce tuto [1)]

Contributeurs, les Fix Me! sont là pour vous aider, supprimez-les une fois le problème corrigé ou le champ rempli !

Nota : Les autres wiki :

- debuter-avec-les-scripts-shell-bash
- script-bash-variables-arguments-parametres
- modification de variable et de paramètre
- script-bash-enchainement-de-commandes-et-etat-de-sortie
- script-bash-etat-de-sortie-et-les-tests
-
- script-bash-les-fonctions

Utiliser des tableaux

Les tableaux ne sont pas spécifiés par POSIX.
Les tableaux indexés unidimensionnels sont pris en charge en utilisant la syntaxe et la sémantique similaire à la plupart des shells comme Korn.
Pour plus détails voir : http://wiki.bash-hackers.org/syntax/arrays

Définitions

1. Un tableau est une variable contenant d'autres variables rangée en mémoire les unes à la suite des autres.
2. Un tableau est unidimensionnel.
3. Chaque variable contenue dans le tableau est nommée par le numéro de sa case et affectée d'une valeur (le contenu de sa case).
4. L'indice est le numéro de la case.
5. Appeler l'indice d'une case permet d'appeler la valeur correspondante à cette case : on dit “déréférencer l'élément du tableau”.

> Les termes “matrice”, “vecteur” ou “table” sont parfois synonymes de “tableau”;
> “l'indice” est parfois appelé “index”.

> Les cases d'un tableau peuvent ne pas être consécutives.

Quatre méthodes pour créer un tableau

- Avec la commande `declare` et l'option `-a` :

```
declare -a nom-tableau=(valeur0 valeur1 valeur2 ...)
```

- Directement :

```
nom-tableau(valeur0 valeur1 ...)
```

- Autre syntaxe :

```
nom-tableau=([indice0]=valeur0 [indice1]=valeur1 ...)
```

Si l'on souhaite créer un tableau avec des cases “manquantes”,
c'est-à-dire, donnée à la valeur de certaines cases, un indice qui n'est pas le numéro qui suit directement, il faut utiliser alors cette méthode.

- Avec la commande `read` :
  les valeurs sont données via le terminal par l'utilisateur.

```
read -a nom-tableau
```

Exemple

mon-script

```
#!/bin/bash
array=([2]=nom1 nom2 [6]=nom3)   # (1)

echo ${!array[@]}
tb=(nom1 nom2 nom3)
echo ${!tb[*]}
declare -a tab=(nom1 nom2 nom3)
echo ${!tab[@]}
read -a tableau
echo ${!tableau[*]}
```

> (1) On voit là que si l'on n'affecte pas un indice à une case,
> elle prend le numéro suivant de l'indice précédemment déclaré.

```
./mon-script
```

retour de la commande

```
2 3 6
0 1 2
0 1 2
nom nom nom nom nom
0 1 2 3 4
```

> `2 3 6` : la première valeur a l'indice 2 ; la deuxième a l'indice 3 ; la troisième a l'indice 6.
>
> `0 1 2` : Il y a une valeur dans les cases d'indice 0, d'indice 1, d'indice 2.
>
> `0 1 2` : idem
>
> `0 1 2 3 4` : Sans indice donné manuellement, les indices attribués aux valeurs suivent l'ordre des entiers naturels en commençant à zéro.

Récupération de valeur(s)

Valeur d'une case

```
${nom-tableau[indice]}
```

Valeurs de toutes les cases

Tous les éléments d'un tableau sont accessibles avec chacune de ces deux syntaxes :

```
${nom-tableau[*]}
```

```
${nom-tableau[@]}
```

Nombre d'éléments d'un tableau

Le nombre d'éléments d'un tableau est accessible par chacune de ces deux syntaxes :

```
${#nomtableau[*]}
```

```
${#nomtableau[@]}
```

n° des cases utilisées

Il est possible de récupérer le numéro des cases qui ont été utilisées dans un tableau,
Pour ce faire, la syntaxe est :

```
${!nom-tableau[*]}
```

Concaténer deux tableaux en un seul

Concaténer dans un seul tableau nouvellement initialisé les cases de tableaux existants :

```
nom-tableau3=("${nom-d-un-tableau1[@]}" "${nom-d-un-tableau2}" ...)
```

Exemple

mon-script

```
#!/bin/bash
# méthodes 1 :
tab=(salade_verte, savoyarde, niçoise,)
echo ${tab[0]}
echo ${tab[1]}
echo ${tab[2]}
echo "Nous avons dit : ${tab[*]} c-à-d ${#tab[@]} choix."
echo " "
#méthode 2:
declare -a tableau=(entrecôte, cabillaud, fruits_de_mer, frites,
gratin_dauphinois)
echo ${tableau[0]}
echo ${tableau[1]}
echo ${tableau[2]}
echo "Nous avons dit : ${tableau[*]} c-à-d ${#tableau[@]} choix de
plus."
echo " "
#méthode 3 :
t=( [0]=glace, [1]=tarte-maison, [2]=yahourt, [3]=pudding, [4]=fruits,)
echo ${t[0]}
echo ${t[1]}
echo ${t[2]}
echo ${t[3]}
echo ${t[4]}
echo "Nous avons dit : ${t[*]} c-à-d ${#t[@]} choix de plus."
#méthode 4 par la lecture :
echo "Écrivez ce qu'il manque à cette liste, svp ? : "
read -a tb
echo "vous auriez souhaité : ${tb[*]} "
echo "la liste est maintenant : ${tab[*]} ${tableau[*]} ${t[*]}
${tb[*]}"
declare -a arrayconcat=("${tab[@]}" "${tableau[@]}" "${t[@]}"
"${tb[@]}")
echo "Cela nous fait une liste de ${#arrayconcat[@]} choix !"
```

Nombres entiers

Si l'on souhaite créer un tableau d'entiers, on utilise la commande `declare` avec l'option `-ai`. (l'ordre des options n'a pas d'importance.) Les syntaxes précédentes peuvent aussi être utilisées :

mon-script

```
#!/bin/bash
declare -ai tab=(18 2 -2)
echo ${tab[@]}
tableau=([0]=2 [2]=4 [4]=6)
echo ${tableau[@]}
tb=(1 2 3 4)
echo ${tb[@]}
read -a t
echo ${t[@]}
```

```
./mon-script
```

retour de la commande

```
18 2 -2
2 4 6
1 2 3 4
5 6 7 8 9
5 6 7 8 9
```

Ajout suppression

- Pour ajouter un élément à un tableau : `tableau[${#tableau[*]}]=element`

mon-script

```
#!/bin/bash
tableau=( bleu jaune rouge )
echo ${#tableau[*]}
tableau[${#tableau[*]}]=couleurs
echo ${tableau[*]}
```

```
3
bleu jaune rouge couleurs
```

- Pour ajouter un élément au début d'un tableau : `tableau=( element ${tableau[*]} )`

mon-script

```
#!/bin/bash
tab=( noir blanc )
echo ${tab[*]}
tab=( colorless ${tab[*]} )
echo ${tab[*]}
```

```
noir blanc
colorless noir blanc
```

- pour supprimer un tableau : `unset nom-tableau`
- pour supprimer la case d'un tableau : `unset nom-tableau[indice] …`

Tableau associatif

Un tableau associatif est un tableau uni-dimentionnel à ceci près que les index sont des chaînes de caractères.
Il en va de même que précédemment :

- l'indice se récupère ainsi : `${!tableau[@]}`;
- la valeur se récupère ainsi : `${tableau[@]}`;

mon-script

```
#!/bin/bash
declare -A eleves=([secondeA]=30 [secondeB]=29)
for i in "${!eleves[@]}"
   do
   echo "l'indice ici nom de la classe est : $i"
   echo "le nombre d'élèves est ici la valeur : ${eleves[$i]}"
   done
```

```
./mon-script
```

retour de la commande

```
l'indice ici nom de la classe est : secondeB
le nombre d'élèves est ici la valeur : 29
l'indice ici nom de la classe est : secondeA
le nombre d'élèves est ici la valeur : 30
```

- Il faut utiliser @ (et non *) pour que la boucle permette d'afficher **chacune des valeurs**.

Conclusion

Récapitulatif

mon-script

```
#!/bin/bash
tab=( rouge vert bleu)
echo ${tab[1]}
echo ${tab[@]}

tab[1]="orange"
echo ${tab[1]}
echo ${tab[@]}

echo $tab # $tab est un équivalment de $tab0 : il affichera le première élément

echo ${tab[@]} # affiche tous les éléments de même ${tab[*]}
echo ${#tab[@]} # pour avoir la longueur (nombres d'éléments dans le tableau):
                # ${tab[@]}-1 (moins un car on commence à zéro!
tab[8]="vert" # nlle affectation sur tab[8]
echo ${!tab[@]} # ! avoir la liste des indices du tableau
echo ${tab[@]}

for i in ${!tab[*]} ; do
    echo $i : ${tab[$i]}
done
```

Se creuser un peu les méninges

Comparez ses méthodes pour associer à l'affichage, un à un , deux éléments du retours :

- Avec un tableau :

  mon-script

  ```
  #!/bin/bash
  tableau=([1]=a [2]=b [3]=c)
  for var in "${!tableau[@]}" ; do
      echo "$var : ${tableau[$var]}"
  done
  ```

- Avec les paramètres de position et une boucle :

mon-scipt

```
set a b c
n=1
for v in $@
do
    echo "$n: $v"
    ((n++))
done
```

Merci à enikar pour ce script et cette réflexion sur IRC 😎

Mais aussi, après avoir vu script-bash-les-fonctions avec le script de captnfab : Fonctionnalités avancées du Shell: Les fonctions

Une astuce

Lister avec numérotation tous les fichiers d'un répertoire

mon-script

```
#!/bin/bash
directory=(/*) # (1)

for i in ${!directory[@]} ; do
    echo "$i : ${directory[$i]}" # (2)
done
```

> (1) : Tableau composé par les chemins de dossiers et fichiers depuis la racine.
> (2) : Pour afficher tous les éléments du tableau.

La suite c'est ici

- script-bash-les-fonctions

1)

N'hésitez pas à y faire part de vos remarques, succès, améliorations ou échecs !