Cours : fonctions et passage d'arguments

Passage d'arguments à un script shell

Passage sur la ligne de commande

Le mécanisme de passage d'arguments à un script shell n'est pas une définition du shell. Il répond à un standard plus général des systèmes d'exploitation (POSIX). Il est donc commun à tous les langages de programmation mais sous des mises en oeuvre légèrement différentes.

Pour passer des arguments à un script shell, on les ajoute sur la ligne de commande. Si `monscript` est un script exécutable, alors :

```
$>./monscript arg1 arg2 "arg3 est une chaîne" arg4
```

passe quatre arguments "chaîne" au script.

Récupération de la ligne de commande

A l'intérieur du script, les arguments passés sont rendus disponibles dans des variables prédéfinies $1, $2 .., $9.

Le premier argument $0 représente le nom du script lui-même.

ATTENTION : On ne peut accéder simultanément qu'à 10 arguments. Pour plu:s d'arguments, il faut effectuer un "décalage".

```
---- monscript----
#!/bin/bash

echo $0
echo $1
echo $2
echo $3
shift
echo $3
```

Le script précédent, alimenté par l'appel ci-avant, affiche le résultat suivant :

```
$>./monscript arg1 arg2 "arg3 est une chaîne" arg4
./monscript
arg1
arg2
arg3 est une chaîne
arg4
```

Explication :

La première ligne correspond à l'appel de la variable $0 qui donne le nom du script lancé (premier "mot" de la ligne de commande).

Les trois lignes qui suivent affichent respectivement les arguments 1, 2, 3, dans l'ordre de la ligne de commande.

Un décalage est opéré, qui décale vers la droite la correspondance des variables standard avec les "tokens" de la liogne de commande. L'argument $3 devient le quatrième valeur sur la ligne, ce qui se vérifie à l'affichage.

Mécanisme du shift

Définition d'une fonction

Une fonction est une sous-séquences d'instructions shell qui peut être appelée plusieurs fois par son nom. Comme le principe de fonctions le peremt dans les langages habituels, une fonction shell peut recevoir des paramètres.

L'idée de bash est de reproduire ici le même mécanisme que celui qui est utilisé pour le passage des paramètres en ligne de commande. On unifie ainsi le processus de passage de paramètres, quelque soit le niveau d'observation du système.

Déclaration de la fonction

```
<nomFonction>(){

... script de fonction

}
```

Voici un exemple :

```
listertar(){
   echo -n "Contenu de l'archive $1"
   if [ ${1##*.} = "tar" ]
   then
      echo "(non compressé)"
      tar tvf $1
   elif [ ${1##*.} = "gz" ]
   then
      echo "(gzip compressé)"
     tar xzvf $1
   elif [ ${1##*.} = "bz2" ]
   then
      echo "(bzip-2 compressé)"
     tar xzvf $1
   fi
}
```

Récupération des paramètres envoyés à la fonction

Comme vous pouvez le constater ci-dessus, la fonction utilise un argument $1. Cet argument n'est plus un argument de la ligne de commande, mais le premier argument de l'appel de la fonction. Le mécanisme de réception des paramètres de fonction fonctionne donc exactement, à l'intérieur de la fonction, comme pour l'appel d'un shell, avec les 10 variables prédéfinies $0 à $9 et la commande shift pour décaler.

Appel d'une fonction à partir d'un script

Puisque le passage des paramètres reprend l'intégralité du mécanisme de la ligne de commandes, pourquoi pas aller jusqu'au bout ?

L'utilisation d'une fonction revient alors à utiliser une nouvelle commande du shell :

```
#!/bin/bash

listertar(){
   echo -n "Contenu de l'archive $1"
   if [ ${1##*.} = "tar" ]
   then
      echo "(non compressé)"
      tar tvf $1
   elif [ ${1##*.} = "gz" ]
   then
      echo "(gzip compressé)"
     tar xzvf $1
   elif [ ${1##*.} = "bz2" ]
   then
      echo "(bzip-2 compressé)"
     tar xzvf $1
   fi
}

for f in /home/root/tarballs/*
do
    listertar f
done
```