script bash : Fonctions

- Objet : script bash : Fonctions
- 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-sorie-et-les-tests
- script-bash-les-tableaux
-

Définition d'une fonction

man bash

```
 Définitions des fonctions de l'interpréteur
       Une  fonction  de  l'interpréteur est un objet qui est appelé
comme une
       commande simple et qui exécute une commande composée  avec  un
nouveau
       jeu  de  paramètres  positionnels. Les fonctions de
l'interpréteur sont
       déclarées de la façon suivante :

       nom () commande-composée [redirection]
       function nom [()] commande-composée [redirection]
```

Différentes syntaxes

- Syntaxes avec la commande `function` :

```
function nom-de-la-fonction
{
suite-de-commandes
}
nom-de-la-fonction
```

ou

```
function nom-de-la-fonction {
suite-de-commandes
}
nom-de-la-fonction
```

- `function` : c'est la commande `function` ;
- entre les `{ }` c'est le corps de la fonction ;
- ne pas oublier l'espace avant `{` ;
- on y place la ou les commandes exécutées par l'appel de la fonction;
- l'appel de la fonction se fait après sa définition;
- c'est le fait de mentionner le nom de la fonction qui l'appelle ou permet d'exécuter les commandes du corps de la fonction.

- Exemple :

mon-script

```
#!/bin/bash
function f {

echo "Bonjour tout le monde"
}
f
```

```
./mon-script
```

retour de la commande

```
Bonjour tout le monde
```

Les mots réservés `function` et `}` doivent être les premiers d'une commande pour qu'ils soient reconnus.

Sinon il suffit de mettre `;}` (avec un espace devant `;`)

- Syntaxe avec `nom-de-la-fonction (){` :

mon-script

```
#!/bin/bash
mafonction (){
echo hello
}
mafonction
```

```
./mon-script
```

retour de la commande

```
coucou
```

- Syntaxe proche de C :

mon-script

```
#!/bin/bash
mafonction ()
{
echo hello
}
mafonction
```

Une fonction ne peut pas être vide !
Il faut mettre forcément des commandes dans le corps de la fonction.

Les arguments du script et les paramètres d'une fonction

- On voit souvent les termes arguments et paramètres comme des synonymes car les arguments passés au script peuvent être récupérés pour être utilisés comme paramètres passés à une/des fonction(s) déclarées et implémentées dans le script.
- Voyons d'abord comment passer des paramètres à une fonction du script.

- Les appels des arguments d'une fonction sont placés dans le corps de celle-ci.
- Et les arguments sont placés de gauche à droite et du premier au dernier, à côté du nom d'appel de la fonction

- la valeur du premier argument est référencée par $1
- la valeur du deuxième argument est référencée par $2, etc.
- les paramètres spéciaux $0 (Contient le nom du script tel qu'il a été invoqué), $# (Le nombre de paramètres passés au script), $@ (l'ensemble des arguments, un argument par paramètre) , $* (l'ensemble des paramètres sous la forme d'un seul argument), $? (Le code retour de la dernière commande), $$ (le PID su shell qui exécute le script), $! (le PID du dernier processus lancé en arrière-plan)

mon-script.sh

```
#!/bin/bash
function f
{
echo $0
echo $USER
echo $3 $1 $2
echo $#
echo $@
echo $*
}
f chez debian facile   #chez : premier argument
                       #debian : deuxième argument
                       #facile : troisième argument
```

```
./mon-script
```

retour de la commande

```
./mon-script
hypathie
facile chez debian
3
chez debian facile
chez debian facile
```

- Les paramètres de la fonction peuvent être les paramètres passés au script.

ParamToScript.sh

```
#!/bin/bash
function f
{
echo "SCRIPT_NAME : " $0
echo "USER : " $USER
echo "Script Argument 1 : "$1 "Script Argument 2 : "$2 "Script Argument
```

```
2 : "$3
}
f $1 $2 $3
```

```
./ParamToScript.sh arg1 arg2 arg3
```

retour de la commande

```
SCRIPT_NAME :  ./ParamToScript.sh
USER :  hypathie
Script Argument 1 :  \nScript Argument 2 :  \Script Argument 2 :
```

La commande interne shift

Cette commande permet de décaler la numérotation des paramètres de position de la fonction

script minipoesie

```
function minipoesie
{
echo "nom complet : $0"
echo " "
echo "$*" # avant 'shift 1'
shift 1
echo "$*" # après 'shift 1'
echo "$*" # avant 'shift 2'
shift 2
echo -e "\t$*" # après 'shift 2'
echo -e "\t  $*" # avant 'shift 3'
shift 3
echo -e "\t    $*" # après 'shift 3'
}
minipoesie hypathie chez debian facile
```

```
./minipoesie
```

retour de la commande

```
nom complet : ./minipoesie

hypathie chez debian facile
chez debian facile
chez debian facile
    facile
      facile
```

```
        facile
```

Et voilà, vous devriez maintenant être capable de tout comprendre de l'exemple de ce lien :

- Fonctionnalités avancées du Shell : Les fonctions

Et n'oubliez pas de retourner à la comparaison :

- se creuser un peu les méninges !

Fonctions et redirections

man bash

```
   nom () commande-composée [redirection]
   function nom [()] commande-composée [redirection]
```

`[redirection]`

Utiliser le pipe

Voir le tuto : Le pipe ou tuyau

script getip.sh

```
#!/bin/bash
getip (){
/sbin/ifconfig ${1:-eth0} | egrep "([0-9]{1,3}\.){3}[0-9]{1,3}"
}
getip
```

ou

script getip.sh

```
#!/bin/bash
getip (){
/sbin/ifconfig ${1:-eth0}
}
getip | egrep "([0-9]{1,3}\.){3}[0-9]{1,3}"
```

```
./getip.sh
```

retour de la commande

```
inet adr:192.168.0.22  Bcast:192.168.0.255  Masque:255.255.255.0
```

Fonction, variables et paramètres passés au script

Créer une fonction qui permet d'afficher une variable et les paramètres passés au script.

script essai.sh

```
#!/bin/bash
var=coucou

fct()
{
    echo "$var"
    echo "$1 $2"
}

fct $1 $2
```

```
./essai.sh a b
```

retour de la commande

```
coucou
a b
```

Fonctions et autres commandes

À savoir : Comme la commande “exit”, la commande “return” permet de changer le code de retour, mais contrairement à “exit”, return fait sortir de la fonction (arrête l'exécution des commandes du corps de la fonction), sans arrêter l'ensemble du programme (script).

script

```
#!/bin/bash

# La fonction "fct1" a pour condition d'exécution de ses commandes
# un test qui vérifie qu'elle a un argument.
# Comme elle en a un, il s'exécute 'echo $1' et 'return 0' qui attribue
le code de retour 0.
# && : exécution de la seconde cmd, si le code de retour de la première
```

```
est 0 : c'est le cas
# donc exécution de 'echo $?
# || : exécution de la seconde cmd, si le code de retour de la première
est différente de zéro
# donc pas d'exécution de 'echo "coucou"'

function fct1
{
    if [ $1 ]; then
        echo "$1"
        return 0
    fi
}
fct1 yep && echo $? || echo "coucou"
echo " "                              # pour espacer le retour
# pas d'argument, donc c'est le 'else' qui s'exécute
# donc exécution de 'echo "pas d'argument...' et de 'return 1'
# || : exécution de la seconde cmd, si le code de retour de la première
est différente de zéro
# donc exécution de 'echo "coucou"'

fct2()
{
    if [ $1 ]; then
        echo "$1"
    else
        echo "pas d'argument à cette fonction"
        return 1
        echo "après return"   # On est sortie de la fonction : pas
d'exécution de 'echo'
    fi
}
fct2 || echo "coucou"
```

```
yep
0

pas d'argument à cette fonction
coucou
```

Commande source

Il est possible d'utiliser la fonction d'un script dans un autre script.

- Soit le script “echocolor.sh” ci-dessous :

script echocolor.sh

```
# voici des variables affectée des codes couleurs qu'on trouve sur le net
noir='\e[0;30m'
gris='\e[1;30m'
rougefonce='\e[0;31m'
rouge='\e[1;31m'
vertfonce='\e[0;32m'
vertclair='\e[1;32m'
orange='\e[0;33m'
jaune='\e[1;33m'
bleufonce='\e[0;34m'
mauve='\e[1;34m'
fuschia='\e[0;35m'
rose='\e[1;35m'
cyan='\e[0;36m'
bleuclair='\e[1;36m'
blanc='\e[1;37m'
normal='\e[0;m'

echocolor()
{
    echo -e "${@}${normal}"
}
```

Voir sur internet, par exemple : ici pour les codes couleur.

- Dans le script “source.sh” ci-dessous, on va se servir de la fonction “echocolor” du script “echocolor.sh” pour coloriser les sorties des commandes.

script source.sh

```
#!/bin/bash
source echocolor.sh  # ou .nom_du_script

minipoesie()
{
echo "$*"
shift 1
echo "$*"
echo "$*"
shift 2
echo -e "\t$*"
echo -e "\t  $*"
shift 3
echo -e "\t    $*"
}
minipoesie hypathie chez debian facile
echocolor $orange "c'est une mauvaise poésie"
```

L'exécution de ./source.sh permet de coloriser en orange la chaîne “c'est une mauvaise poésie”.

OU MIEUX :

script source.sh

```
#!/bin/bash
source echocolor.sh  # ou .nom_du_script

minipoesie()
{
echocolor $bleuclair  "$*"
shift 1
echocolor $rose  "$*"
echocolor $jaune  "$*"
shift 2
echocolor -e $cyan "\t$*"
echocolor -e $fuschia "\t  $*"
shift 3
echocolor -e $mauve "\t    $*"
}
minipoesie hypathie chez debian facile
```

Avec les couleurs, c'est un peu plus joli 😃

1)
N'hésitez pas à y faire part de vos remarques, succès, améliorations ou échecs !