

Gestion des paramètres en bash

- Objet : Gérer des paramètres utilisateur
- Niveau requis :
[avisé](#)
- Commentaires : Créer un script acceptant des paramètres de position, nommés et booléens.
- Débutant, à savoir : [Utiliser GNU/Linux en ligne de commande, tout commence là !](#) 😊

Introduction

Voici quelques exemples permettant de créer des scripts bash possédants une gestion plus fine des paramètres d'entrées plutôt que la simple utilisation de leur position (\$1, \$2, ...)

Avec ces scripts, vous pourrez supporter :

1. Des paramètres de position

```
./script.sh arg_1
```

1. Des paramètres appelant une valeur

```
./script.sh -a valeur_a --long_parametre_b valeur_b
```

1. Des paramètres booléens

```
./script -o
```

Le détail de fonctionnement des scripts est inclut en commentaire.

bonus : pour dé-commenter les scripts :

```
cat script.sh | sed '/^\s*#/d' | sed 's/#.*$//' > script_sans  
commentaires.sh
```

Paramètre booléen

Utilisation

```
./parametre_booleen.sh -o
```

```
Vrai
```

```
./parametre_booleen.sh
```

```
Faux
```

Script

[parametre_booleen.sh](#)

```
#!/bin/bash

# Paramètre booléen forme courte et longue -o/--long_option

# On recherche une correspondance soit avec la forme courte, soit avec la longue.
# On ajoute des espaces autour des variables pour une comparaison stricte et ne pas identifier de sous chaîne.

if [[ " $@" =~ " -o " ]] || [[ " $@" =~ " --long_option " ]] ; then
    PARAM=true # On peut affecter une valeur booléenne
fi

if [ -n "$PARAM" ]; then # Si la chaîne est non-nulle (~ que la variable a été créée (~ remplie))
    echo "Vrai"
else
    echo "Faux"
fi
```

Paramètre & valeur

Utilisation

```
./parametre_nomme.sh -a val_a
```

```
param_a = val_a
```

```
./parametre_nomme.sh
```

```
param_a =
```

Script

[parametre_nomme.sh](#)

```
#!/bin/bash
# Paramètre attendant une valeur (named parameter)
```

```
params=( "$@" ) # copie de du tableau des Paramètres

for i in $(seq $#); do # On itère un à un sur tout les paramètres
    j=$((i-1)) # seq commence à compter à 1, le premier paramètre est
à 0.
    if [[ "${params[$j]}" =~ "-a" ]] || [[ "${params[$j]}" =~ "-
-long_a" ]]; then
        PARAM_A=${params[$i]} # On récupère la paramètre succédant la
clé "-a", soit "value_a"
    fi
done

echo "param_a = $PARAM_A"
```

Paramètres nommé et court (-p) avec getopt

getopts fait parti des buildins de bash, ce qui permet d'être sûr son implémentation (dès lors que l'on utilise bash)

getopts ne prend que des arguments courts (-p), qui peuvent accepter un paramètre

Utilisation

```
./getopts_simple.sh -v -a val_a -b val_b -o
```

```
a=val_a
b=val_b
option_o=true
```

Script

[getopts_simple.sh](#)

```
#!/bin/bash

# Paramètres "short" (-p) avec la build-in getopt

help_msg () {
    echo "Usage: $(basename $0) [-h] [-v] [-a PARAM_A] [-b PARAM_B] [-
o]"
    echo 'Description'
    echo '  -h          display this help'
    echo '  -v          verbose'
    echo '  -a PARAM_A  Le paramètre A'
    echo '  -b PARAM_B  Le paramètre B'
```

```
    echo '    -o          option OPTION_0'
}

verbose_print () {                                # remplace un printf par un
printf + condition
    local MESSAGE="${@}"
    if [[ "${VERBOSE}" == true ]];then # à condition que la variable
globale $VERBOSE soit à true
        printf "${MESSAGE}\n"
    fi
}

# Si aucun argument donné, on affiche l'aide
if [[ ${#} -eq 0 ]]; then
    help_msg
    exit 1
fi
# Définition des paramètres
optstring=":hva:b:o" # getopt cherchera les paramètres listés ('-h',
'v', 'a',...)
# et attendra une valeur pour ceux suivis d'un
':' (-a value_a)

# Voici la liste des variables utilisées par getopt :
# OPTIND => index de l'argument
# OPTARG => valeur de l'option si définie ( -a option =>
OPTARG="option" )
# OPTERR => 0 ou 1, si égal à 1, bash affiche les messages d'erreur (à
1 par défaut en début de boucle)

OPTION_0=false

# itère sur les paramètres type
"-p",
while getopt ${optstring} arg ; do # ne prend pas en compte les
positionnel (arg)
# ! le type long (--long) n'est
pas accepté => cas '?'
    case ${arg} in
        h) # Si $arg contient la valeur "h"
            help_msg
            exit 0 # Quitte sans erreur
            ;; # fin du test, ne teste pas les arguments
suivant => retour en début de boucle
        v)
            VERBOSE='true' # variable globale à true
(lue dans verbose_print)
            verbose_print "Mode prolix activé."
    esac
done
```

```
;;
a)                                # Si $arg contient la valeur "a"
    PARAM_A="${OPTARG}"          # getopt a défini la variable $OPTARG
    # qui contient le paramètre succédant à "-a"
;;
b)                                # même chose
    PARAM_B="${OPTARG}"
;;
o)                                # On ne demande pas de valeur, on
    OPTION_0='true'              # passe simplement la variable à true si l'option est présente
;;
?)                                # l'argument n'a pas été
défini
    echo "Option invalide: -${OPTARG}."
    echo
    help_msg
    echo
    ;;
esac
done

printf "a=$PARAM_A \nb=$PARAM_B \noption_o=$OPTION_0\n"
verbose_print "\nMessage si et seulement si drapeau '-v'"
```

Paramètres nommé et court (-p) et arguments positionnels avec getopt

getops ne gère pas les arguments positionnels, ce script permet leur implémentation

Utilisation

```
./get_opt_example.sh -v pos_1 -a val_a -b val_b -o pos_2
```

```
Mode prolix activé.
a=val_a
b=val_b
option_o=true
positionnal= pos_1 pos_2
```

Script

[nom.sh](#)

```
optstring=":hva:b:o"

OPTION_0=false
while [ $# -gt 0 ]; do          # Tant que $@ contient des éléments

    unset OPTIND                # on supprime les variables $OPTIND
    (position de l'argument suivant dans $@)
    unset OPTARG                # et $OPTARG définies dans le
    précédent tour de boucle

    while getopts ${optstring} arg; do  # Même chose que précédemment
        case ${arg} in
            h)
                help_msg
                exit 0
                ;;
            v)
                VERBOSE='true'
                verbose_print "Mode prolixé activé."
                ;;
            a)
                PARAM_A="${OPTARG}"
                ;;
            b)
                PARAM_B="${OPTARG}"
                ;;
            o)
                OPTION_0='true'
                ;;
            ?)
                echo "Option invalide: -${OPTARG}."
                echo
                help_msg
                echo
                ;;
        esac
    done

    # getopts itère jusqu'au premier argument positionnel
    # (c-à-d qui n'est pas sous forme '-p' ou ne correspond pas à une
    valeur (-p valeur))
    #
    # Si en début de tour,
    #
    #     $@ = -a value_a -b value_b arg_1 ...
    #
    # optargs s'arrête à arg_1,
    # on a traité 4 arguments : '-a', 'value_a', 'b' et "value_b", et
    getopts à défini
```

```

#
#     OPTIND = 5
#
# le prochain argument à traiter
#

    shift $((OPTIND-1)) # On consomme 5-1 arguments avec le "shift",
la liste des arguments $@ devient:
    #
    #     $@ = arg_1 ...

    ARGS="${ARGS} $1 " # L'argument positionnel se retrouve en $1
    shift              # Une fois récupéré, on le consomme et on
recommence la boucle jusqu'à ce qu'il n'y ai plus d'arguments

done

printf "a=$PARAM_A \nb=$PARAM_B \noption_o=$OPTION_0
\npositionnal=$ARGS\n"
verbose_print "Message si drapeau '-v'"

```

Tout type de paramètre

Pour permettre l'utilisation d'argument long (-long) ou créer une implémentation compatible POSIX, on peut se passer de getopt.

Utilisation

```
./all_parameters_type.sh -a val_a --long_b val_b -o pos_1 pos_2
```

```

positional : pos_1 pos_2
param_a : val_a
param_b : val_b
option_o : true

```

Script

[all_parameters_type.sh](#)

```

#!/bin/bash

help_msg () {
    printf ""
    usage: script [-h] [-o] [-a NAMED_A] [-b NAMED_B]

```

Description

Paramètres optionnels

```
-h, --help          show this help message and exit
-a NAMED_A, --long_a NAMED_A
                    Le paramètre PARAM_A
-b NAMED_B, --long_b NAMED_B
                    Le paramètre PARAM_B
-o, --option        L'option OPTION_0
"""
}
```

```
# get named parameters
```

```
# On utilisera par la suite la fonction shift, qui "consomme" les arguments du tableau $@
```

```
# si $@ = "-a value_a --long_b value_b"
```

```
# Executer :
```

```
# shift
```

```
# Consomme le premier argument ("-a"):
```

```
# $@ = "value_a --long_b value_b"
```

```
#
```

```
# $$ compte le nombre d'argument restants, et par conséquent est réduit de 1 à chaque "shift"
```

```
# Prenons un exemple simple, "dans l'ordre d'écriture de la fonction"
```

```
# Soit la commande:
```

```
#
```

```
# ./script.sh -a value_a --long_b value_b -o positional
```

```
#
```

```
# La variable $@ contient alors "-a value_a --long_ value_b -o positional"
```

```
# On initialise la liste des paramètres de position
```

```
POSITIONAL=""
```

```
# On boucle tant qu'il y a des arguments
```

```
while [ $# -gt 0 ] ; do      # $$ Nombre d'arguments passés en paramètre du script
```

```
    key="$1"                # => key = "-a", le premier argument
```

```
    case $key in            # Teste $key pour les cas suivant:
```

```
        -h|--help)
```

```
            help_msg        # exécute la fonction help ()
```

```
            exit 0          # et quitte le script
```

```
            ;;
```

```
        -a|--long_a)        # la clé "-a" est reconnue
```

```
            PARAM_A="$2"    # la valeur du paramètre PARAM_A vient
```



```

juste après, soit $2
    shift                # consomme un argument, $@ ne contient plus
que "value_a -b value_b"
    shift                # consomme un deuxième argument, $@ = "-b
value_b"
    ;;                  # On peut passer au tour suivant, avec $key
prenant la valeur $1, soit "-b"
    -b|--long_b)        # (2e tour) : $key à la valeur "-b" et est
reconnue
    PARAM_B="$2"        # Comme précédemment, on prend l'argument
suivant $key/$1 soit $2
    shift                # Et on consomme la clé ("-b")
    shift                # Puis la valeur du paramètres ("value_b")
    ;;
    -o|--option)        # (3e tour) : la clé "-o" est reconnue
    OPTION_0=true       # On affecte une valeur booléenne (on
pourra la tester avec [ -z OPTION_0 ] par exemple )
    shift                # Cette fois ci on ne passe qu'un
paramètres, puisqu'il n'y a pas de valeur donnée
    ;;
*)                      # La clé ne correspond à
aucun autre cas
    if ! [[ "$key" =~ "^-" ]]; then # Si elle ne commence pas
par un "-"
        POSITIONAL="$POSITIONAL $key" # C'est un paramètre
positionnel, on l'ajoute à la liste
        shift                # On passe le paramètres (1
seul shift)
    else                # Si la clé commence
par "-"
        echo "Argument non défini : '$key'" # C'est quelle ne
fait pas partie de la liste définie plus haut
        exit 1                # On arrête le
programme
    fi
esac
done # FIN! Bien-sûr l'exemple est arrangé dans le "bon ordre", mais
la position des arguments n'a pas d'importance.

# Paramètres parsés
printf "positional : $POSITIONAL \n" # Contient : "positional"
printf "param_a : $PARAM_A \n"      # Contient : "value_a"
printf "param_b : $PARAM_B \n"      # Contient : "value_b"
printf "option_o : $OPTION_0 \n"    # Contient : "true"

```

Séparer les paramètres

Dans le cas de paramètres inconnus à l'avance, ou pour un script plus léger, cette fonction permet de

trier les différents arguments en fonction de leur type supposé (positional(*arg*), named (*-n arg*), option (*-o*)) et les stocke dans 3 variables différentes.

A default, un groupe ``-n arg`` est considéré comme **named** et non comme un **option** + **positional**

Par exemple:

1. ``-named value_1 value_2`` \Rightarrow ``-long value_1`` est supposé comme groupe **named**, ``value_2`` est supposé **positional**
2. ``-option -long value`` \Rightarrow ``-option`` est considéré comme **option**, ``-long value``, comme un groupe **named**

tri sommaire des arguments (!robustesse!)

Utilisation

```
split_args -a value_a pos_1 -o --long_b value_b pos_2
```

```
POSITIONAL : pos_1 pos_2  
NAMED : -a value_a --long_b value_b  
OPTION : -o
```

Script

fonction_split_args

```
split_args () {  
    POSITIONAL_ARGS=""  
    NAMED_ARGS=""  
    OPTION_ARGS=""  
    while [ $# -gt 0 ] ; do  
  
        if [[ "$1" =~ ^- ]]; then  
            if ! [[ "$2" =~ ^- ]] && [ -n "$2" ] ; then  
                NAMED_ARGS="$NAMED_ARGS $1 $2"  
                shift  
            else  
                OPTION_ARGS="$OPTION_ARGS $1"  
            fi  
        else  
            POSITIONAL_ARGS="$POSITIONAL_ARGS $1"  
        fi  
        shift  
    done  
}
```

From:

<http://debian-facile.org/> - **Documentation - Wiki**

Permanent link:

<http://debian-facile.org/utilisateurs:david5647:tutos:bash-gerer-les-parametres>



Last update: **24/03/2021 17:20**