Bash : les caractères de transformation de parametres

- Objet : suite de la série de wiki visant à maîtriser bash via les différents caractère spéciaux.
- 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 08/04/2014
  - Testé par Hypathie en Avril 2014
- Commentaires sur le forum : ici [1)]

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

Nota : Dans cette page l'expression “caractère générique” est employée dans le sens de métacaractère, glob ou pattern[2)].

Caractères et syntaxe de transformation de paramètres

On peut substituer des variables ou paramètres définis(es) précédemment par l'utilisateur dans un script, et cela permet d'en transformer le contenu. Cela permet aussi de donner une valeur à une variable déclarée mais nulle.

Syntaxe de substitution	**Significations**
`${!nom-paramètre}` (ne pas confondre avec l'appel de l'historique `!`, ou le `!=` des tests)	Remplace la valeur d'un paramètre
`${paramètre:-motif}`	Utilisation d'une valeur par défaut
`${paramètre:=motif}`	Attribution d'une valeur par défaut
`${paramètre:début:longueur}`	Extraction de sous-chaîne
`${paramètre:+motif}`	Utilisation d'une valeur différente
`${paramètre:?motif}`	Affichage d'erreur si paramètre inexistant
`${#paramètre}`	Remplace un paramètre précédent par sa longueur en caractères

Syntaxe de substitution	Significations
${paramètre##motif}	Supprime de $paramètre la plus longue partie de $motif qui correspond au début de $paramètre
${paramètre#motif}	Supprime à partir de $paramètre la plus courte partie de $motif qui correspond au début de $paramètre.
${paramètre%%motif}	Supprime de $paramètre la plus grande partie de $motif qui corresponde à la fin de $paramètre.
${paramètre%motif}	Supprime de $paramètre la plus petite partie de $motif correspondant à la fin de $paramètre.
${paramètre//motif/chaîne}	Remplace motif par chaîne pour transformer “paramètre”

Exemples sans caractères génériques (=pattern-less)

Certaines substitutions de paramètre recoupent les fonctionnalités de la commande expr.

script

```
#!/bin/bash
#!/bin/bash
chaine=abcABC123
echo ${#chaine}
echo `expr "$chaine" : '.*'`
```

```
9
9
```

l'opérateur : est équivalent à un match (voir expressions rationnelles) et commande “expr”

Substitution de paramètres par référence indirecte

script

```
#!/bin/bash
set -o posix
tante=tati
tati=tata
echo "tante = $tante" #référence directe (rappel)
```

```
echo "tante = ${!tante}" #référence indirecte
```

```
tante = tati
tante = tata
```

Substitution par modification

script

```
#!/bin/bash
set -o posix
var=ancienne-valeur
echo ${var:+nouvelle-valeur}
```

```
nouvelle-valeur
```

Substitution d'un paramètre non-déclaré ou de valeur nulle

script

```
#!/bin/bash
set -o posix
#vi=
echo ${vi-`whereis vi`}
vi=
echo ${vi:-`whereis vi`}
```

```
vi: /usr/bin/vi /usr/bin/X11/vi /usr/share/man/man1/vi.1posix.gz
/usr/share/man/man1/vi.1.gz
vi: /usr/bin/vi /usr/bin/X11/vi /usr/share/man/man1/vi.1posix.gz
/usr/share/man/man1/vi.1.gz
```

Attribution d'une valeur par défaut

script

```
#!/bin/bash
echo ${nom_utilasateur=`whoami`} # attribution d'une valeur par défaut,
rien n'a été déclaré
nom_utilisateur=moi  # déclaré et non-nul
echo ${nom_utilisateur:=`whoami`}
```

```
hypathie
moi
```

- Ou encore :

Attention : l'indice du premier caractère d'une chaîne de caractères est 0.

script

```
#!/bin/bash
echo "J'aime les chats"
surdite="j\' aime les Chats"   # Les caractères antislash, apostrophe et espace comptent pour un caractère aussi.
echo "T' aimes qui ? "
echo ${surdite:8} # le 8ième à partir de 0 est inclus
```

```
J'aime les chats
T' aimes qui ?
les Chats
```

Extraction de sous-chaîne contenu dans la valeur de la variable ${paramètre:début:longueur}

script

```
#!/bin/bash
set -o posix
echo "abcdefghijklmnopqrstu : extraire le plus long mot possible : "
devinette=abcdefghijklmnopqrstu
echo -n ${devinette:0:1} ; echo -n ${devinette:13:1} ; echo -n ${devinette:19:1} ;
echo -n ${devinette:8:1} ; echo -n ${devinette:2:1} ; echo -n ${devinette:14:1} ;
echo -n ${devinette:13:1} ; echo -n ${devinette:18:1} ; echo -n ${devinette:19:1}
echo -n ${devinette:8:1} ; echo -n ${devinette:19:1} ; echo -n ${devinette:20:1}
echo -n ${devinette:19:1} ; echo -n ${devinette:8:1} ; echo -n ${devinette:14:1}
echo -n ${devinette:13:1} ; echo -n ${devinette:13:1} ; echo -n ${devinette:4:1}
echo -n ${devinette:11:1} ; echo -n ${devinette:11:1} ; echo -n ${devinette:4:1}
echo -n ${devinette:12:1} ; echo -n ${devinette:4:1} ; echo -n ${devinette:13:1}
echo ${devinette:19:1}
```

```
echo "extraire trois lettres à partir de la lettre f : "
echo ${devinette:5:3}  # f est la sixième mais 5 car la première est pointée zéro, on en prends 3.
echo "extraire les trois premières lettres"
set abcdefgh
echo ${1:0:3}   # 1 car avec set il s'agit de valeur de paramètre donc pas nom,
                # 0 pointe la première lettre, et on en prend 3.
```

```
abcdefghijklmnopqrstu : extraire le plus long mot possible :
anticonstitutionnellement
extraire trois lettres à partir de la lettre f :
fgh
extraire les trois premières lettres
abc
```

Substitution de la valeur d'une variable (ou d'un paramètre) par la longueur de sa chaîne ${#paramètre}

script

```
#!/bin/bash
set -o posix
mot="anticonstitutionnellement"
echo ${#mot}
set anticonstitutionnellement
echo ${#1}              #rappelons-nous du 1 dans ${1:0:3}
```

```
25
25
```

Suppression de la plus courte chaîne ${paramètre#modèle}

script

```
#!/bin/bash
set -o posix
mot=pingpong
echo ${mot#ping}
```

```
pong
```

Suppression de la plus longue chaîne $paramètre##modèle}

script

```
#!/bin/bash
set -o posix
mot=fichier.txt
echo ${mot##fichier}
```

```
.txt
```

Modifier un paramètre en supprimant ses derniers caractères

script

```
#!/bin/bash
set -o posix
var=fichier.txt
echo ${var%%.txt}
```

```
fichier
```

- Ou encore :

script

```
#!/bin/bash
set -o posix
var1=abc.fichier
echo ${var1%.fichier}
```

```
abc
```

Remplacer une partie d'une chaîne par un motif littéral

Toutes les occurrences du motif sont remplacées par la chaîne.

script

```
#!/bin/bash
set -o posix
var1=chienchien
```

```
echo ${var1//ien/at}
var2=viaouviaou
echo ${var2//v/m}
```

```
chatchat
miaoumiaou
```

- Remarque : ne changer que la première occurrence.

```
echo ${var/a/x}
var1=fichier.txt
echo ${var1/txt/sh}
```

```
xbcabc
fichier.sh
```

- Quand il s'agit de ne changer que la première occurrence, on peut choisir de changer la première ou la dernière :

script

```
#!/bin/bash
set -o posix
var=blablabla
echo ${var/#bla/bli}
var=blablabla
echo ${var/%bla/bli}
```

```
bliblabla
blablabli
```

- Pour supprimer un groupe de caractère :

```
${paramètre//motif/}
```

> permet de remplacer par rien toutes les occurrences du motif

```
${paramètre/motif/}
```

> permet de remplacer par rien la première occurrence du motif à l'intérieur de paramètre.

- Par exemple :

script

```
#!/bin/bash
```

```
set -o posix
var=blablabla
echo ${var//bl/}
var=blablabla
echo ${var/bl/}
```

```
aaa
ablabla
```

Substitution de chaînes en utilisant les globs basiques (patterns ou caractères génériques)

Tout ce qui a été vu dans les exemples du paragraphe 1 (ci-dessus) peut porter à confusion avec les expressions rationnelles (correspondance, substitution) :

```
m/motif/          #avec le m facultatif
s/motif/chaîne
```

Cette confusion vient souvent de la ressemblance de la syntaxe `var/.../...` ou `var//.../...` que l'on utilise pour la substitution de paramètres, avec celle des expressions rationnelles (voir la note ci-dessous).

Elle est renforcée aussi par la ressemblance des globs `*` , `?` , `[ - ]` ainsi que par certains caractères utilisés avec les expressions rationnelles .

Attention donc, s'il est possible d'utiliser `*` , `?` et `[ - ]` dans les modifications de chaînes d'un paramètre que nous venons de voir, il n'est pas possible d'utiliser les expressions rationnelles directement.

Avec le shell Bash on utilise les *expressions rationnelles* au moyen de commandes externes qui elles les utilisent, comme grep, sed, awk, vi (par exemple : set -o vi).
Voir : bash-vii-globs-etendus-regex.

On n'utilise pas de *ER* pour ce qui concerne les substitutions de paramètres abordés ici, mais il est possible d'utiliser les caractères génériques basiques (pattern ou glob), ainsi que “les génériques de class” [[:class:]] au moyen de sa commande interne “shopt”, munie de l'option extglob.

Utilisation du caractère génériques " ? ".

script

```
#!/bin/bash
var=debian123456facile
echo ${var//2?4/.}
var1=debian123456facile
echo ${var1//3?5/.}
var2=debian123456facile
echo ${var2//4?6/.}
var3=debian123456facile
echo ${var3//?/.}
```

```
debian1.56facile
debian12.6facile
debian123.facile
..................
```

Utilisation du caractère générique "*".

script

```
#!/bin/bash
set -o posix
var=debian123456facile
echo ${var//1*6/.}
```

```
debian.facile
```

Suppression à gauche

```
echo $PWD
echo ${PWD#*/}
echo ${PWD##*/}
```

```
/home/hypathie
home/hypathie
hypathie
```

Suppression à droite

```
fichier="ABCD::12:03:2014"
echo ${fichier%:*}
echo ${fichier%%:*}
```

```
ABCD::12:03
```

```
ABCD
```

Utilisation des brackets [-]

script

```
#!/bin/bash
v=debian123456facile
echo ${v//[1-9]/.}
v1=debian123456facile
echo ${v1//[1-9]*/.}
v2=debian123456facile
echo ${v2//[1-9]*/}
v3=debian123456facile
echo ${v3//[1-9]*/.}
v4=debian123456facile
echo ${v4//*[1-9]/.}
v5=debian123456facile
echo ${v5//[a-z]/x}
v6=debian123456facile
echo ${v6//[a-z 0-9]/x}
echo ${v6//[0-9 a-z]/x}
```

```
debian......facile
debian.
debian
debian.
.facile
xxxxxx123456xxxxxx
xxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxx
```

Modifier la case : caractères {...^...}, {...^^...}, {...,...}, {...,,...}

script

```
#!/bin/bash
var="je vais apprendre à utiliser le shell."
echo ${var^}
echo ${var^}
echo ${var^^[ai]}
var1="JE VAIS APPRENDRE À UTILISER LE SHELL."
echo ${var1,}
echo ${var1,,}
echo ${var1,,e}
```

```
echo ${var1,,[AE]}
```

```
Je vais apprendre à utiliser le shell.
Je vais apprendre à utiliser le shell.
je vAIs Apprendre à utIlIser le shell.
jE VAIS APPRENDRE À UTILISER LE SHELL.
je vais apprendre À utiliser le shell.
JE VAIS APPRENDRE À UTILISER LE SHELL.
Je VaIS aPPReNDRe À UTILISeR Le SHeLL.
```

commande shopt -s extglob et braquet de classe <nowiki>[[:class:]]</nowiki>

```
([[:lower:]])
```

Elle désigne une suite de caractères minuscules : on peut retirer la première occurrence, ou toutes les occurrences à des caractères.

script

```
#!/bin/bash
shopt -s extglob
var=123abcd45ef
echo ${var/+([[:lower:]])/} #1ière occ. de munisc.

var1=123a45bcdef
echo ${var1/+([[:lower:]])/}
var2=ab123cdefgh45ij
echo ${var2//+([[:lower:]])/} #toutes occ. de minusc.
```

```
12345ef
12345bcdef
12345
```

rappel pour le + des lignes 4, 6, 8
${paramètre:+motif} ⇒ utilisation d'une valeur différente

et d'autres encore :

```
([[:digit:]])
```

Exemples

script

```
#!/bin/bash
shopt -s extglob
var1=123abcd456
echo ${var1//+([[:digit:]])/}
var2=ABC123EFG
echo ${var2//+([[:upper:]])/}
var3=abc?ABC\!123.de\;gh\:45,6
echo ${var3//+([[:punct:]])/}
var4="AB c d 1 2 3"   #Ne pas oublier les " "
echo ${var4//+([[:space:]])/}
```

```
abcd
123
abcABC123degh456
ABcd123
```

Négation de caractères [^...] et [!...] ; extraction de symboles [.SYMBOL.]

script

```
#!/bin/bash
shopt -s extglob
var1=123abcd456
echo ${var1//+([^abc])/}
var2=123abcd456
echo ${var2//+([^[:digit:]])/}
var3=12\/3abc@d456
echo ${var3//+([!abcd])/}
var4=123abcd456
echo "On ne remplace par rien donc c'est pareil :
    ${var4//+([.SYMBOL.])/}"
var5=123abcd456
echo "On remplace par l'inverse de ce qu'il n'y a pas :
    c'est-à-dire par : ${var5//+([^[.SYMBOL.]])/}:"
var6="ÇSYMBOLa alors !"
echo ${var6//+([.SYMBOL.])/}
```

```
abc
123456
abcd
On ne remplace par rien donc c'est pareil :
    123abcd456
On remplace par l'inverse de ce qu'il n'y a pas :
    c'est-à-dire par : :
```

```
Ça alors !
```

Donc :

- Remarque sur les syntaxes : `$var/.../...` et `$var//.../...` .
- Attention de ne pas confondre les globs étendus et les expressions régulières

Avec shopt -s extglob :

1. On peut utiliser les globs simples (* , ? , [ ]) et les globs étendus : [ ], |, ?, @, !, +, ^
1. Mais on ne peut pas utiliser les expressions régulières (où les globs étendus sont “ré-utilisés” avec une syntaxe différente, à côté d'autres caractères : la distinction entre globs étendus et *ER* est développée ici).

À voir aussi : http://wiki.bash-hackers.org/syntax/pattern

- Pour les classes de caractères, attention à la norme POSIX : http://hyperpolyglot.org/shell :

Non POXIS : `[[:nom-classe:]]`

POSIX : `[:upper:]` ; `[:alnum:]`, etc.

Une liste plus complète ici : les-ensembles-de-caracteres-possibles
Voir le wiki suivant : globs étendus et expressions régulières

- Pour les autres options de la commande interne shopt :

Voir man bash (tout à la fin, c'est long) au paragraphe “commandes internes”.

tuto précédent :

Bash : les tableaux

tuto suivant :

Bash : globs étendus et regex

1)

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

2)

definition-usuelle-de-metacaractere-et-detail