Manipuler les chaînes de caractères

Table of Contents

Bash supporte un nombre surprenant d'opérations de manipulation de chaînes de caractères. Malheureusement, ces outils manquent d'unité. Certains sont un sous-ensemble de la substitution de paramètre et les autres font partie des fonctionnalités de la commande UNIX expr. Ceci produit une syntaxe de commande non unifiée et des fonctionnalités qui se recoupent, sans parler de la confusion engendrée.

Longueur de chaînes de caractères

```
${#chaine}
expr length $chaine
expr “$chaine” : '.*'
```

```
chaineZ=abcABC123ABCabc

echo ${#chaineZ}                 # 15
echo `expr length $chaineZ`      # 15
echo `expr "$chaineZ" : '.*'`    # 15
```

Longueur de sous-chaînes correspondant à un motif au début d'une chaîne

```
expr match “$chaine” '$souschaine'
expr “$chaine” : '$souschaine'
```

```
    chaineZ=abcABC123ABCabc
    #       |------|

    echo `expr match "$chaineZ" 'abc[A-Z]*.2'`   # 8
    echo `expr "$chaineZ" : 'abc[A-Z]*.2'`       # 8
```

Index

expr index $chaine $souschaine

Position numérique dans $chaine du premier caractère dans $souschaine qui correspond.

```
    chaineZ=abcABC123ABCabc
    echo `expr index "$chaineZ" C12`            # 6
                                                # C position.

    echo `expr index "$chaineZ" 1c`             # 3
    # 'c' (à la position #3) correspond avant '1'.
```

Ceci est l'équivalent le plus proche de strchr() en C.

Extraction d'une sous-chaîne

${chaine:position}

Extrait une sous-chaîne de $chaine à partir de la position $position.

Si le paramètre **$chaine** est «*» ou «@», alors cela extrait les paramètres de position, [30] commençant à **$position**.

${chaine:position:longueur}

Extrait **$longueur** caractères d'une sous-chaîne de **$chaine** à la position $position.

```
    chaineZ=abcABC123ABCabc
    #       0123456789.....
    #       indexage base 0.
```

```
    echo ${chaineZ:0}                                # abcABC123ABCabc
    echo ${chaineZ:1}                                # bcABC123ABCabc
    echo ${chaineZ:7}                                # 23ABCabc

    echo ${chaineZ:7:3}                              # 23A
                                                     # Trois caractères de la
sous-chaîne.
```

Est-il possible d'indexer à partir de la fin de la chaîne ?

```
    echo ${chaineZ:-4}                               # abcABC123ABCabc
    # Par défaut la chaîne complète, comme dans ${parametre:-default}.
    # Néanmoins...

    echo ${chaineZ:(-4)}                             # Cabc
    echo ${chaineZ: -4}                              # Cabc
    # Maintenant, cela fonctionne.
    #  Des parenthèses ou des espaces ajoutés permettent un échappement du
paramètre
    #+ de position.
```

Les arguments position et longueur peuvent devenir des « paramètres », c'est-à-dire représentés par une variable, plutôt que par une constante numérique.

Si le paramètre **$chaine** est «*» ou «@», alors ceci extrait un maximum de **$longueur** du paramètre de position, en commençant à $position.

```
    echo ${*:2}          # Affiche le deuxième paramètre de position et les
suivants.
    echo ${@:2}          # Identique à ci-dessus.
    echo ${*:2:3}        # Affiche trois paramètres de position, en
commençant par le deuxième.
```

expr substr $chaine $position $longueur

Extrait **$longueur** caractères à partir de $chaine en commençant à **$position.**

```
chaineZ=abcABC123ABCabc
#       123456789......
#       indexage base 1.

echo `expr substr $chaineZ 1 2`              # ab
echo `expr substr $chaineZ 4 3`              # ABC
```

```
expr match “$chaine” '($souschaine)'
```

Extrait **$souschaine** à partir du début de **$chaine**.

```
expr “$chaine” : '($souschaine)'
```

Extrait **$souschaine** à partir du début de **$chaine**.

```
    chaineZ=abcABC123ABCabc
    #       =======

    echo `expr match "$chaineZ" '\(.[b-c]*[A-Z]..[0-9]\)'`   # abcABC1
    echo `expr "$chaineZ" : '\(.[b-c]*[A-Z]..[0-9]\)'`       # abcABC1
    echo `expr "$chaineZ" : '\(.......\)'`                   # abcABC1
    # Toutes les formes ci-dessus donnent un résultat identique.
```

```
expr match “$chaine” '.*($souschaine)'
```

Extrait **$souschaine** à la fin de **$chaine**.

```
expr “$chaine” : '.*($souschaine)'
```

```
    Extrait $souschaine à la fin de $chaine, et où $souschaine est une
expression rationnelle.

    chaineZ=abcABC123ABCabc
    #               ======

    echo `expr match "$chaineZ" '.*\([A-C][A-C][A-C][a-c]*\)'`    # ABCabc
    echo `expr "$chaineZ" : '.*\(......\)'`                       # ABCabc
```

Suppression de sous-chaînes

```
${chaine#souschaine}
```

Supprime la correspondance la plus petite de **$souschaine** à partir du début de **$chaine**.

```
${chaine##souschaine}
```

Supprime la correspondance la plus grande de **$souschaine** à partir du début de **$chaine**.

```
chaineZ=abcABC123ABCabc
#       |----|
#       |----------|

echo ${chaineZ#a*C}      # 123ABCabc
# Supprime la plus petite correspondance entre 'a' et 'C'.

echo ${chaineZ##a*C}     # abc
# Supprime la plus grande correspondance entre 'a' et 'C'.
```

```
${chaine%souschaine}
```

Supprime la plus petite correspondance de **$souschaine** à partir de la fin de **$chaine**.

```
Par exemple :

# Renomme tous les fichiers de $PWD
#+ en remplaçant le suffixe "TXT" par "txt".
# Par exemple, "fichier1.TXT" devient "fichier1.txt" . . .

SUFF=TXT
suff=txt

for i in $(ls *.$SUFF)
do
  mv -f $i ${i%.$SUFF}.$suff
  #  Ne modifie rien *en dehors* de la correspondance la plus courte
  #+ commençant du côté droit de $i . . .
done ### Ceci pourrait être condenser en une ligne si nécessaire.
```

```
${chaine\%\%souschaine}
```

Supprime la plus grande correspondance de **$souschaine** à partir de la fin de **$chaine**.

```
    chaineZ=abcABC123ABCabc
    #                    ||
    #        |------------|

    echo ${chaineZ%b*c}      # abcABC123ABCa
    #  Supprime la plus petite correspondance entre 'b' et 'c', à partir de
la fin
    #+ de $chaineZ.

    echo ${chaineZ%%b*c}     # a
    #  Supprime la plus petite correspondance entre 'b' et 'c', à partir de
la fin
    #+ de $chaineZ.
```

Cet opérateur est utilisé pour générer des noms de fichier.

Remplacement de sous-chaîne

```
${chaine/souschaine/remplacement}
```

Remplace la première correspondance de **$souschaine** par **$remplacement**.

```
${chaine\/\/souschaine/remplacement}
```

Remplace toutes les correspondances de **$souschaine** avec **$remplacement**.

```
    chaineZ=abcABC123ABCabc

    echo ${chaineZ/abc/xyz}           # xyzABC123ABCabc
                                      #  Remplace la première correspondance
de
                                      #+ 'abc' avec 'xyz'.

    echo ${chaineZ//abc/xyz}          # xyzABC123ABCxyz
                                      #  Remplace toutes les correspondances
de
                                      #+ 'abc' avec 'xyz'.
```

```
${chaine/#souschaine/remplacement}
```

Si **$souschaine** correspond au début de **$chaine**, substitue **$remplacement** à **$souschaine**.

```
${chaine/%souchaine/remplacement}
```

Si **$souschaine** correspond à la fin de **$chaine**, substitue **$remplacement** à **$souschaine**.

```
    chaineZ=abcABC123ABCabc

    echo ${chaineZ/#abc/XYZ}          # XYZABC123ABCabc
                                      #  Remplace la correspondance de fin
de
                                      #+ 'abc' avec 'XYZ'.

    echo ${chaineZ/%abc/XYZ}          # abcABC123ABCXYZ
                                      #  Remplace la correspondance de fin
de
                                      #+ 'abc' avec 'XYZ'.
```

Exemples

Insérer une ligne blanche entre les paragraphes d'un fichier texte

```
#!/bin/bash
# paragraph-space.sh

# Insère une ligne blanche entre les paragraphes d'un fichier texte.
# Usage: $0 <NOMFICHIER

LONGUEUR_MINI=45        # Il peut être nécessaire de changer cette valeur.
#  Suppose que les lignes plus petites que $LONGUEUR_MINI caractères
#+ terminent un paragraphe.

while read ligne  # Pour toutes les lignes du fichier...
do
  echo "$ligne"   # Afficher la ligne.

  longueur=${#ligne}
  if [ "$longueur" -lt "$LONGUEUR_MINI" ]
    then echo    # Ajoute une ligne blanche après chaque petite ligne.
  fi
done

exit 0
```

Générer « aléatoirement » une chaîne de huit caractères

```
    #!/bin/bash
    # rand-string.sh
    # Générer aléatoirement une chaîne de huit caractères.

    if [ "-n $1" ]  #  Si présence d'un argument en ligne de commande,
    then            #+ alors l'utiliser comme chaîne de départ.
      chaine0="$1"
    else            #  Sinon, utiliser le PID du script.
      chaine0="$$"
    fi

    POS=2  # On commence en position 2.
    LONG=8  # Extraction de huit caractères.

    chaine1=$( echo "$chaine0" | md5sum | md5sum )
    # Double mixage :            ^^^^^^   ^^^^^^

    chainealeatoire="${chaine1:$POS:$LONG}"
    # Peut se paramétrer       ^^^^ ^^^^^

    echo "$chainealeatoire"

    exit $?

    # bozo$ ./rand-string.sh mon-motdepasse
    # 1bdd88c4

    #  Non, ceci n'est pas recommandé
    #+ comme méthode sûre de génération de mots de passe.
```

Convertir des formats de fichiers graphiques avec une modification du nom du fichier

```
    #!/bin/bash
    #  cvt.sh:
    #  Convertit les fichiers image MacPaint contenus dans un répertoire
dans le
    #+ format "pbm".

    #  Utilise le binaire "macptopbm" provenant du paquetage "netpbm",
    #+ qui est maintenu par Brian Henderson (bryanh@giraffe-data.com).
    #  Netpbm est un standard sur la plupart des distributions Linux.

    OPERATION=macptopbm
    SUFFIXE=pbm         # Suffixe pour les nouveaux noms de fichiers.

    if [ -n "$1" ]
    then
      repertoire=$1      # Si le nom du répertoire donné en argument au
```

```
script...
    else
      repertoire=$PWD    # Sinon, utilise le répertoire courant.
    fi
    #  Suppose que tous les fichiers du répertoire cible sont des fichiers
image
    # + MacPaint avec un suffixe de nom de fichier ".mac".

    for fichier in $repertoire/*  # Filename globbing.
    do
      nomfichier=${fichier%.*c} #  Supprime le suffixe ".mac" du nom du
fichier
                                #+ ('.*c' correspond à tout ce qui se trouve
                                #+ entre '.' et 'c', inclus).
      $OPERATION $fichier > $nomfichier.$SUFFIXE
        # Redirige la conversion vers le nouveau nom du fichier.
        rm -f $fichier          # Supprime le fichier original après sa
convertion.
      echo "$nomfichier.$SUFFIXE"  # Trace ce qui se passe sur stdout.
    done

    exit 0
```

Convertir des fichiers audio en ogg

```
    #!/bin/bash
    # ra2ogg.sh : Convertit des fichiers audio de streaming (*.ra) en ogg.

    # Utilise le programme "mplayer" :
    #      http://www.mplayerhq.hu/homepage
    #      Vous aurez peut-être besoin d'installer les codecs appropriés
    #+     pour que ce script fonctionne.
    # Utilise la bibliothèque "ogg" et "oggenc" :
    #      http://www.xiph.org/


    PREFIXE_FICHIER_RESULTAT=${1%%ra}      # Supprime le suffixe "ra".
    SUFFIXE_FICHIER_RESULTAT=wav           # Suffixe pour le fichier wav.
    FICHIER_RESULTAT="$PREFIXE_FICHIER_RESULTAT""$SUFFIXE_FICHIER_RESULTAT"
    E_SANSARGS=65

    if [ -z "$1" ]          # Un nom de fichier à convertir doit être
spécifié.
    then
      echo "Usage: `basename $0` [nom_fichier]"
      exit $E_SANSARGS
    fi
```

```
#########################################################################
    mplayer "$1" -ao pcm:file=$FICHIER_RESULTAT
    oggenc "$FICHIER_RESULTAT"  # Corrige l'extension du fichier ajoutée
automatiquement pas oggenc.
#########################################################################

    rm "$FICHIER_RESULTAT"      # Supprime le fichier temporaire *.wav.
                        # Si vous voulez le conserver, commentez la ligne ci-
dessus.

    exit $?
```

Émuler getopt

Une simple émulation de **getopt**[1)].

```
    #!/bin/bash
    # getopt-simple.sh
    # Auteur : Chris Morgan
    # Utilisé dans le guide ABS avec sa permission.


    getopt_simple()
    {
        echo "getopt_simple()"
        echo "Les paramètres sont '$*'"
        until [ -z "$1" ]
        do
          echo "Traitement du paramètre : '$1'"
          if [ ${1:0:1} = '/' ]
          then
              tmp=${1:1}               # Supprime le '/' devant...
              parametre=${tmp%%=*}     # Extrait le nom.
              valeur=${tmp##*=}        # Extrait la valeur.
              echo "Paramètre : '$parametre', valeur: '$valeur'"
              eval $parametre=$valeur
          fi
          shift
        done
    }

    # Passe toutes les options à getopt_simple().
    getopt_simple $*

    echo "test vaut '$test'"
    echo "test2 vaut '$test2'"

    exit 0
```

Pour utiliser ce script il faut indiquer deux paramètres:

```
sh getopt_example.sh /test=valeur1 /test2=valeur2
```

Manipuler des chaînes de caractères avec awk

Un script Bash peut utiliser des fonctionnalités de manipulation de chaînes de caractères de **awk** comme alternative à ses propres fonctions intégrées.

```
#!/bin/bash
# substring-extraction.sh

Chaine=23skidoo1
#      012345678    Bash
#      123456789    awk
# Notez les différents systèmes d'indexation de chaînes :
# Bash compte le premier caractère d'une chaîne avec '0'.
# Awk  compte le premier caractère d'une chaîne avec '1'.

echo ${Chaine:2:4} # position 3 (0-1-2), longueur de quatre caractères
                                        # skid

# L'équivalent awk de ${string:position:longueur} est
substr(string,position,longueur).
echo | awk '
{ print substr("'"${Chaine}"'",3,4)      # skid
}
'
#  Envoyé un "echo" vide à awk donne une entrée inutile, et donc permet
d'éviter
#+ d'apporter un nom de fichier.

exit 0
```

[1)]

getopt analyse les arguments de la ligne de commande. en utilisant des constructions d'extraction de sous-chaînes