

Systemes d'exploitation

Chapitre 7-5

Neutralisation, redirection,
décodage des paramètres,
évaluation répétitive,
expr(1)

Neutralisation (2^e visite) (1)

➤ Rappel:

- La neutralisation est nécessaire parce que certains caractères sont interprétés différemment par le shell et par les commandes.
- Il n'y a pas assez de caractères dans notre alphabet !
- Donc, la neutralisation est nécessaire pour empêcher le shell de faire des substitutions inappropriées.

Neutralisation (2^e visite) (2)

- Les méta-caractères utilisés dans la neutralisation:

Caractère	Description
\	Neutralise la signification spéciale des méta-caractères " " \$ de l'interpréteur de commandes incluant le caractère spécial \ lui-même.
` `	Réalise la substitution des commandes. Les commandes placées entre " " sont remplacées par leur résultat lors du traitement par l'interpréteur de commandes.
' '	Neutralise la signification spéciale de tous les méta-caractères. Les apostrophes ne permettent pas la substitution des variables et des commandes.
" "	Neutralise la signification de tous les méta-caractères à l' exception de !, \ et \$. Les guillemets permettent la substitution des variables et des commandes.

Neutralisation (2^e visite) (3)

➤ Exemples d'utilisation des guillemets:

```
$ name=Fred
```

```
$ echo "Mon nom est $name"
```

```
Mon nom est Fred
```

```
$
```

```
$ echo "Mon nom est '$name'"
```

```
Mon nom est 'Fred'
```

```
$
```

```
$ echo "Mon nom est \$name"
```

```
Mon nom est $name
```

```
$
```

```
$ echo "Mon nom est \\$name"
```

```
Mon nom est \Fred
```

```
$
```

```
$ echo "La date d'aujourd'hui est : `date`"
```

```
La date d'aujourd'hui est : Fri Jun 4 11:34:28 EDT 1999
```

```
$
```

Neutralisation (2^e visite) (4)

➤ Exemple d'utilisation des apostrophes:

```
$ echo 'Mon nom est $name'  
Mon nom est $name  
$
```

```
$ echo 'La date d'aujourd'hui est `date`'  
La date d'aujourd'hui est `date`  
$
```

```
$ echo 'La date d'aujourd'hui est: `date`'  
La date d'aujourd'hui est: `date`  
$
```

```
$ echo 'La date d\'\'\'aujourd\'\'\'hui est: `date`'  
La date d'aujourd'hui est: `date`  
$
```

Neutralisation (2^e visite) (5)

➤ Exemple d'utilisation de l'anti-slash:

```
$ echo \ ` dat e \ `
` dat e `
$ echo \$name
$name
$ echo *
COURS4 Mai l TEST compt eusager gunzi p passwd. dat t est
$ echo \ *
*
$
```

```
$ echo ' La dat e d' \ ' ' auj our d' \ ' ' hui est : ' ` dat e `
La dat e d' auj our d' hui est : Mbn Jun 7 10: 36: 54 EDT 1999
$
```

```
$ echo ' La dat e d \ auj our d \ hui est : ' ` dat e `
La dat e d \ auj our d \ hui est : ` dat e `
$
```

Redirection (2^e visite) (1)

➤ Syntaxe spéciale:

Syntaxe	Signification
<<[-] mot f i n	Les données sont lues de l'entrée standard. La lecture des données se termine lorsque la chaîne motfin est rencontrée. Les caractères de la chaîne motfin peuvent être neutralisés. Si l'option — est utilisée, tous les caractères de tabulation dans la chaîne motfin et dans le document lus en entrée sont enlevés.
<&-	Fermeture de l'entrée standard.
>&-	Fermeture de la sortie standard.

Redirection (2^e visite) (2)

➤ Exemple d'utilisation:

```
#!/bin/sh
# Nom du programme: installe
#
echo "Répertoire d'installation: \c"
read REPERTOIRE
echo "Fichier log? (O/N): \c"
read LOG_OU_NON
echo "Message à l'écran? (O/N): \c"
read MSG_ECRAN
#
# Étapes d'installation
#
# : : : :
```

```
$ installe <<"FIN"
TOTO
O
N
FIN
$
```

Programme Bourne shell

Redirection de l'entrée standard avec
mot clé pour l'arrêt de lecture

"\$@" et "\$*" (1)

- Lorsque utilisé entre guillemet, la variable `$@` et la variable `$*` n'ont pas la même signification.
- Pour `"$@"` l'interpréteur de commandes substitue les paramètres de position en leur entourant par des guillemets.
- Ce n'est pas le cas pour `"$*"`.

"\$@" et "\$*" (2)

➤ Exemple d'utilisation:

```
1  #! /bin/sh
2  # Nom du fichier : etoile_commercial
3  # etoile_commercial : sert à montrer la distinction entre $* et $@
4
5  # boucler sur les paramètres de la ligne de commande en utilisant
6  # $*
7  echo "Utilisation de \"$*"
8  for OPTION in "$*"
9  do
10     echo "Itération: $OPTION"
11 done
12
13 # boucler sur les paramètres de la ligne de commande en utilisant
14 # $@
15 echo "Utilisation de \"$@"
16 for OPTION in "$@"
17 do
18     echo "Itération: $OPTION"
19 done
```

"\$@" et "\$*" (3)

➤ Exécution du programme précédent:

```
$ et oi l e _ a c o m m e r c i a l  A B C D
U t i l i s a t i o n  d e  $*
I t é r a t i o n :  A B C D
U t i l i s a t i o n  d e  $@
I t é r a t i o n :  A
I t é r a t i o n :  B
I t é r a t i o n :  C
I t é r a t i o n :  D
$
```

Le résultat n'est pas le même !

Truc: utilisez "\$@" dans vos programmes Bourne shell.

Déverminage (1)

- Pour simplifier la recherche des erreurs dans un programme Bourne shell:
 - Utiliser la commande `set(1)` pour activer les modes de déverminage.
 - Les options disponibles sont:

Option	Signification
- n	Lire les commandes mais ne pas les exécuter. Cette option est ignorée pour les programmes interactifs.
- V	Affiche les lignes lues du programme lors de son exécution.
- X	Afficher les commandes et les substitutions lors de leur exécution.

Déverminage (2)

➤ Voici un exemple:

- Ce programme utilise l'option de déverminage **-x**.

```
1  #!/bin/sh
2  # Nom du programme : settest3
3  # settest3 : montrer l'utilisation des options de set(1)
4  # pour le déverminage
5  #
6
7  # Utiliser l'option -v (affiche les commandes et leur argument)
8  set -x
9
10 cd ..
11 pwd
12 ls
13 echo `who`
```

Déverminage (3)

➤ Le résultat après l'exécution du programme précédent:

```
cent i 45> set t est 3
+ cd ..
+ pwd
/ export / home/ exa/ per sonnel / wong
+ ls
DeadLet t er s          r egex. t xt
DEMO                   REVI SI ON1

Doct or at             r l ogi n. t xt
DTPAD                  sort i e. t xt
expr. t xt             t el net . bmp
f t p. bmp             t el net _( Accès_à_di st ance)

f t p. t xt            TEST
HCLNFSD               t est . t xt
i nst al l e. l og     TOTO
LAB2                  user. t xt
Mail                  user t r i e. t xt
r anc at al og. t xt

+ who
+ echo wong pt s/ 6 m ar s 5 15: 20 ( ppp16. et snt l . ca)
wong pt s/ 6 m ar s 5 15: 20 ( ppp16. et snt l . ca)
cent i 46>
```

Décodage des paramètres (1)

- Il existe une commande simple pour le décodage systématique des paramètres de position.
- Il s'agit de la commande **getopts(1)**.

- La syntaxe de cette commande:

`getopts chaîne_options NOMVARIABLE [paramètres]`

- `chaîne_options` représente les options à reconnaître par `getopts(1)`
- `NOMVARIABLE` les options reconnues par `getopts(1)` sont placées dans cette variable
- `parametres` s'il existe, `getopts(1)` va tenter d'extraire les options à partir de cet argument

Décodage des paramètres (2)

➤ Un exemple utilisant getopt(1):

```
1  #!/bin/sh
2  # Nom du fichier : install_lq
3  # install_lq : sert à montrer l'utilisation de getopt(1)
4
5  # Boucler et décoder les options
6  while getopts lq OPT
7  do
8      case "$OPT" in
9          l) OPTON="$OPT"
10             echo "OPTION $OPT reçue" ;;
11          q) OPTON="$OPT"
12             echo "OPTION $OPT reçue" ;;
13          ?) echo "Option invalide détectée"
14             echo "Usage: install [-lq]"
15             exit 1 ;;
16      esac
17      echo "Valeur de \${OPTARG} : ${OPTARG}"
18  done
```

On indique à getopt(1) de reconnaître les options -l, -q, -lq

Décodage des paramètres (3)

➤ Les résultats de ce programme:

```
$ install_lq -l
OPTION l reçue
Valeur de $OPTIND: 2
$ install_lq -l -q
OPTION l reçue
Valeur de $OPTIND: 2
OPTION q reçue
Valeur de $OPTIND: 3
$ install_lq -l q
OPTION l reçue
Valeur de $OPTIND: 1
OPTION q reçue
Valeur de $OPTIND: 2
$ install_lq -x
./install_lq : option incorrecte -- x
Option invalide détectée
Usage: install [-l q]
$ install_lq -l x
OPTION l reçue
Valeur de $OPTIND: 1
./install_lq : option incorrecte -- x
Option invalide détectée
Usage: install [-l q]
$
```

Décodage des paramètres (4)

➤ Un autre exemple: `install_lq2 [-l logfile -q] [nom_rep]`

```

1  #!/bin/sh
2  # Nom du fichier : install_lq2
3  # install_lq2 : sert à montrer l'utilisation de getopts(1)
4
5  # Boucler et décoder les options
6  while getopts l:q OPT
7  do
8      case "$OPT" in
9          l) OPTON="$OPTARG"
10             LOGARG="$OPTARG"
11             echo "OPTION $OPTARG reçue et son argument est $LOGARG" ;;
12          q) OPTON="$OPTARG"
13             echo "OPTION $OPTARG reçue" ;;
14          ?) echo "Option invalide détectée"
15             echo "Usage: install [-l logfile -q] [nom_repertoire]"
16             exit 1 ;;
17      esac
18      echo "Valeur de \$OPTARG: $OPTARG"
19  done
20  # Chercher le paramètre nom_repertoire
21  shift `expr $OPTIND - 1`
22  if [ "$1" ]
23  then
24      REPERTOIRE="$1"
25      echo "Répertoire d'installation: $REPERTOIRE"
26  fi

```

Décodage des paramètres (5)

➤ Résultat de l'exécution du programme précédent:

```
$ install_lq2 -l
./install_lq2 : l'option nécessite un argument -- l
Option invalide détectée
Usage: install [-l logfile -q] [nom_repertoire]
Vous avez reçu du courrier
$ install_lq2 -l msg.log -q
OPTION l reçue et son argument est msg.log
Valeur de $OPTIND: 3
OPTION q reçue
Valeur de $OPTIND: 4

$ install_lq2 -l msg.log -q TOTO
OPTION l reçue et son argument est msg.log
Valeur de $OPTIND: 3
OPTION q reçue
Valeur de $OPTIND: 4
Répertoire d'installation: TOTO
$ install_lq2 TOTO
Répertoire d'installation: TOTO
$
```

Évaluation répétitive (1)

- La commande `eval(1)` permet la substitution des variables et la substitution des commandes AVANT l'exécution d'une commande.
- Exemple:

`CMD=pwd`

`eval "$CMD" &`

Contenu de la variable : —→ `COMMANDE=pwd`

Ligne de commande : —→ `eval $COMMANDE &`

Évaluée par l'interpréteur

Exécution : —→ `pwd &`

Évaluation répétitive (2)

```

1  #!/bin/sh
2  # Nom du programme : eval test 2
3  # eval test 2 : montrer comment créer un tableau par eval (1)
4  #
5  # Fonction créer_tableau : Boucler sur les paramètres de position et
6  # les placer dans un tableau nommé VAL
7  créer_tableau () {
8
9  for VAR in "$@"
10 do
11     eval VAL$#=\ "$VAR" \
12     shift
13 done
14 }
15
16 # Programme principal
17 créer_tableau "$@" # placer les paramètres de position dans VAL
18
19 while [ $# -ge 1 ]
20 do
21     echo "\$VAL$# = \c"
22     eval echo "\$VAL$#" # afficher le contenu du tableau VAL
23     shift
24 done

```

On peut aussi créer des noms de variables pendant l'exécution du programme !

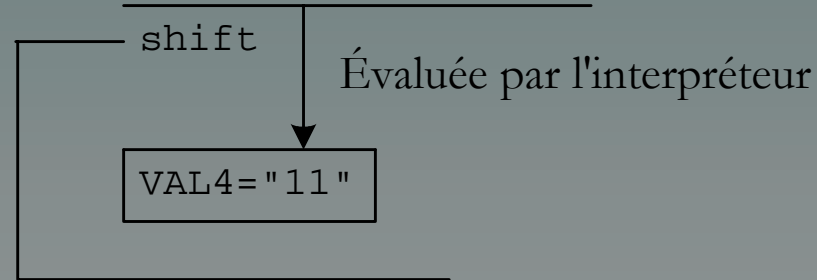
Évaluation répétitive (3)

Itération 1

Paramètres de position → 11 42 33 7

\$# → 4

Lignes de commande : → eval VAL\$#=\ "\$1"\"

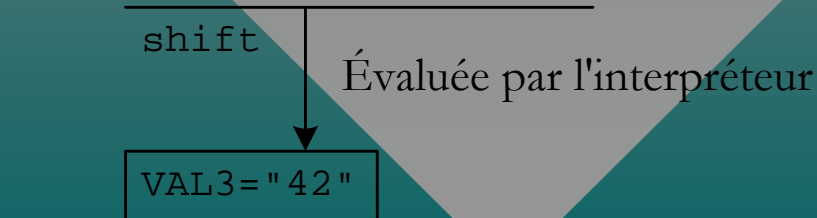


Itération 2

Paramètres de position → 42 33 7 ←

\$# → 3

Lignes de commande : → eval VAL\$#=\ "\$1"\"



Commande expr(1)

- Cette commande permet l'évaluation des expressions spécifiées par les arguments placés dans la ligne de commandes
- Evaluation logique et relationnelle

`expr "$e1 \ | $e2"` (OU logique)

`expr "$e1 \& $e2"` (ET logique)

`expr "$e1 \<= $e2"` (relationnelle)

etc.

Commande expr(1)

➤ Evaluation arithmétique

`expr "$e1 + $e2"` (addition)

`expr "$e1 * $e2"` (multiplication)

➤ Manipulation des chaînes

`expr "$e1 : $e2"` (appariement)



Trouver la chaîne `$e2` dans la chaîne `$e1`. `$e2` peut contenir des expressions régulières

Commande expr(1)

➤ Appariement des chaînes

`expr length $e1` (longueur)

`expr substr $e1 pos ncar` (sous-chaîne)

`expr index $e1 $e2` (position)



Retourner la position de `$e2` dans `$e1`

Il faut utiliser la version
`/usr/ucb/expr` !!

Commande expr(1)

➤ Exemple:

```
$ A="J'aime le système Unix"
```

```
$ /usr/ucb/expr length "$A"
```

```
22
```

```
$ expr "$A" : '.*le.*'
```

```
22
```

```
$ expr "$A" : '.*le\(.*\)
```

```
0
```

```
$ expr "$A" : '.*le\(.*)'
```

```
système Unix
```

```
$ echo '.*le\(.*)'
```

```
.*le\(.*)
```

Commande expr(1)

```
1  #!/bin/sh
2  # Nom du programme : exprtest2
3  # exprtest2 : montrer l'utilisation de expr(1) pour
4  # obtenir le dernier membre d'une chemin de répertoire
5  #
6
7  echo "Donner un chemin de répertoire: \c"
8  read CHEMIN
9  DERNIER_MEMBRE=`expr "$CHEMIN" : '.*\/\(.*\)' \| "$CHEMIN" `
10 echo "Le dernier membre de $CHEMIN est:\n$DERNIER_MEMBRE"
```

Commande expr(1)

DERNIER_MEMBRE=`expr "\$CHEMIN" : '.*\/\(.*\)' \| "\$CHEMIN" `



DERNIER_MEMBRE=`expr e1 \| e2`



e1 → "\$CHEMIN" : '.*\/\(.*\) '

e2 → "\$CHEMIN"

