

Aide-mémoire de la programmation shell

Ce document d'accompagnement du stage «*Ecriture de scripts Shell*» rappelle les points principaux à retenir concernant la programmation pour shells Bourne et Korn.

Logilin Formations – <http://www.logilin.fr/>

Évaluation des expressions

| `variable=valeur`

affectation de *variable* avec la *valeur*.

Pas d'espace autour du signe égal !

| `tableau[rang]=valeur`

affectation d'un *rang* du *tableau* avec la *valeur*.

| `${variable}`

remplacé par le contenu de la *variable*,

| `${tableau[rang]}`

remplacé par le contenu du *rang* du *tableau*,

| `${variable-valeur}`

remplacé par la *valeur* si la *variable* n'est pas définie,

| `${variable=valeur}`

affectation de la *variable* si elle n'est pas définie,

| `${variable?valeur}`

afficher le *message* et fin du shell si *variable* indéfinie.

| `${#variable}`

est remplacé par la longueur du contenu de la *variable*,

| `${variable#motif}`

est remplacé par le contenu de la *variable* privé du plus

court préfixe correspondant au *motif*,

| `${variable%motif}`

est remplacé par le contenu de la *variable* privé du plus

court suffixe correspondant au *motif*,

| `${variable##motif} ${variable%%motif}`

suppression du préfixe ou suffixe le plus long possible.

| `~utilisateur/`

remplacé par le répertoire personnel de l'*utilisateur*,

| `ab{c,d,e}fg`

est développé en `abcfg abdfg abefg`

| `$(commande)`

remplacé par la sortie standard de la *commande*,

| `$((expression))`

remplacé par le résultat de l'évaluation arithmétique entière de l'*expression*.

Protection des caractères spéciaux

| `"$var1 $var2"`

garde la chaîne en remplaçant les variables par leurs valeurs,

| `'$var1 $var2'`

garde la chaîne inchangée (pas de remplacement),

| `\$var`

le *backslash* protège le `$`. Il n'est plus considéré comme caractère spécial (pas de remplacement).

Structures de contrôle

Boucles

| `while cmd_1 ; do`
 commandes

done
Répète les *commandes* tant que *cmd_1* renvoie vrai (0).

| `until cmd_1 ; do`
 commandes

done
Répète les *commandes* tant que *cmd_1* renvoie faux.

| `for variable in liste ; do`
 commandes

done
Répète les *commandes* en remplissant la *variable* avec les éléments successifs de la *liste*.

| `break`
sort directement d'une boucle `for`, `while` ou `until`.

| `continue`
passe à l'itération suivante de la boucle.

Tests

| `if cmd_1 ; then`
 cmd_2

elif *cmd_3* ; then
 cmd_4

else
 cmd_5

fi

Si *cmd_1* renvoie vrai exécute *cmd_2*. Sinon si *cmd_3* renvoie vrai, exécute *cmd_4*, sinon exécute *cmd_5*.

| `case expression in`
 motif_1) *cmd_1* ;;
 motif_2 | *motif_3*) *cmd_2* ;;
 *) *cmd_3* ;;

esac

Si l'*expression* peut correspondre au *motif_1*, exécute *cmd_1*, sinon si elle correspond au *motif_2* ou *motif_3*, exécute *cmd_2*, sinon exécute *cmd_3*.

Fonctions

| `fonction_1 ()`

 {
 commandes...
 }

définit la *fonction_1*.

| `fonction_1 valeur_1 valeur_2...`

invocation de *fonction_1*; dans la fonction les arguments sont dans `$1`, `$2...` et leur nombre dans `$#`.

| `local variable`

déclare une variable locale à la fonction

| `return valeur`

termine la fonction en renvoyant la valeur en retour.

Motifs du shell

| `*`

n'importe quelle chaîne de caractères (même vide),

| `?`

n'importe quel caractère,

| `* \? \\\`

Caractères `*`, `?`, `\`,

| `[liste]`

Caractères `l`, `i`, `s`, `t`, `e`

| `[b-e]`

Caractères `b`, `c`, `d`, `e`

| `[^liste]`

N'importe quel caractère hors de la liste

Redirections

| `commande < fichier`

entrée standard depuis *fichier*,

| `commande > fichier`

sortie standard vers *fichier*,

| `commande >> fichier`

sortie standard ajoutée en fin de *fichier*,

| `commande 2> fichier`

sortie d'erreur vers *fichier*,

| `commande 2>&1`

sortie d'erreur identique à sortie standard,

| `commande <<- ETIQUETTE`

lignes à envoyer

 vers l'entrée standard

 de la *commande*

ETIQUETTE

document en ligne envoyé vers l'entrée standard.

Exécution des commandes

Ligne shebang

| `#! /bin/sh`
en tout début de script.

Pipeline

| `commande | commande | commande`
sortie standard injectée dans l'entrée de la suivante

Liste de pipelines

| `pipeline ; pipeline`
(exécution séquentielle)

| `pipeline & pipeline`
(exécution parallèle)

| `pipeline && pipeline`
(exécution dépendante)

| `pipeline || pipeline`
(exécution alternative)

Commandes composées

| `{ liste de pipelines }`
(regroupement de commandes)
| `(liste de pipelines)`
(sous-shell)

Commandes internes essentielles

echo

| `echo arguments`
affiche les arguments séparés par des espaces.
-n supprime le saut de ligne final
-e interprète les séquences spéciales.

read

| `read variables...`
remplit les *variables* avec les mots successifs de la ligne lue (séparateur : contenu de la variable IFS).
Dernière variable reçoit tout ce qui reste. Par défaut, utilise variable `REPLY`. Renvoie faux en fin de fichier.

exec

| `exec commande`
remplace le (script) shell en cours par la *commande*.
| `exec redirections`
applique les *redirections* indiquées au shell courant.

source

| `source script`
| `. script`
interprète le *script* dans le shell en cours.

exit

| `exit valeur`
termine le (script) shell courant en renvoyant la *valeur*.

test

| `test condition`
| `[condition]`
Laisser des espaces autour des crochets ! Renvoie une valeur vraie ou fausse suivant la condition.
Comparaisons de valeurs numériques :
-eq ... égale à ...
-ne ... différente de ...
-lt (**-le**) ... inférieure (ou égale) à ...
-gt (**-ge**) ... supérieure (ou égale) à ...
Test sur les chaînes :
-n longueur non nulle
-z longueur nulle.
Comparaisons de chaînes : `=`, `!=`, `<`, `>`
Tests sur les fichiers :
-a existence du fichier,
-b périphérique mode bloc,
-c périphérique caractère,
-d répertoire,
-f fichier normal,
-g bit Set-GID validé,
-G appartenant au groupe de l'utilisateur,
-h lien symbolique,
-k bit Sticky validé,
-N modifié depuis la dernière lecture,
-O appartient à l'utilisateur,
-p tube nommé (fifo),
-r peut être lu,
-s taille non-nulle,
-S socket,
-u bit Set-UID validé,
-w peut être écrit,
-x peut être exécuté.
Comparaisons de fichiers :
-ef ... même fichier physique que ...,
-nt ... modifié plus récemment que ...,
-ot ... modifié plus anciennement que ...
Test sur les descripteurs :
-t est un terminal

cd

| `cd repertoire`
change de repertoire de travail,
cd - revient au repertoire précédent,
cd revient au repertoire de connexion.

pwd

affiche le repertoire de travail en cours.

export

| `export variable`
Transfère la *variable* du shell dans l'environnement qui sera transmis aux processus fils ultérieurs.

env

affiche le contenu de l'environnement

set

| `set`
affiche les variables du shell et l'environnement,
| `set options`
configure des paramètres du shell :
-a exporter toutes les variables
-u refuser les variables indéfinies
-v afficher les lignes de commandes avant exécution
-x afficher les développements avant exécution

unset

| `unset variable`
efface la *variable*.

getopts

```
while getopts "ab:c" variable ; do
    case $variable in
        a) echo "opt° -a";;
        b) echo "opt° -b, arg. $OPTARG";;
        c) echo "opt° -c";;
        *) echo "opt° invalide"; exit 1;;
    esac
done

shift $((OPTIND - 1))
echo "arguments restants : "
echo "$@"
exit 0
```

Analyse la ligne de commande en fonction de la liste d'options. Si une option prend un argument (':' après sa lettre), il est dans `OPTARG`. Une fois les options lues, le rang du premier argument restant est dans `OPTIND`.

shift

| `shift n`
décale les arguments en ligne de commande de *n* rangs : `$0` reste inchangé, `$n+1` passe dans `$1`, `$n+2` dans `$2`, etc.

