Tcl mini-tutorial - mbl@lri.fr

Syntaxe - Un *script* est formé d'une suite de *commandes*. Chaque commande est formée de plusie *mots*. Le premier mot est le nom de la commande, les mots suivants les arguments. Les mots sont sépa par des espaces, les commandes sont séparées par des fins de ligne. Le backslash (\) et les groupa permettent d'insérer des caractères spéciaux dans les mots et d'écrire des commandes sur plusieurs lig (voir ci-dessous).

set a 12	affecte la valeur 12 à la variable a
expr 2 + 3	calcule la valeur de l'expression 2 + 3
puts Coucou	affiche Coucou sur la sortie standard

Variables - Il existe des variables simples et des variables tableaux. Le seul type de valeur manipulé Tcl est le type chaîne de caractères. L'indice d'un élément de tableau peut être n'importe quelle chaîne.

set a 12	affecte la valeur 12 à la variable simple a
set coul rouge	affecte la valeur rouge à la variable simple coul
set t(5) 12	affecte la valeur 12 à l'élément d'indice 5 du tableau t
set tab(bleu) 12	affecte la valeur 12 à l'élément d'indice blue du tableau tab
append coul vert	ajoute la chaîne vert à la fin de la valeur de la variable coul
unset a	rend la variable a indéfinie
unset t(5)	rend l'élément d'indice 5 du tableau t indéfini

Evaluation - L'évaluation d'une commande se fait en deux phases : substitution de chaînes de caractè dans la commande, puis exécution de la commande. Il y a deux formes de substitution : substitution variables et substitution de commandes. Les variables à substituer sont précédées d'un $ et les comman à substituer sont entre crochets [ ] :

$a	remplacé par la valeur de la variable a
$t(5)	remplacé par la valeur de l'élément d'indice 5 du tableau t
[commande argument ...]	remplacé par la valeur retournée par la commande

Exemples :

set b $a	$a est remplacé par la valeur de a, la commande devient set b 12, et la variable b prend la valeur 12.
set i [expr 2 + 3]	[expr 2 + 3] est remplacé par 5, la commande devient set i 5.
set x [expr $a + $i]	$a et $i sont remplacés par leurs valeurs (12 et 5), puis la commande expr est exécutée, puis la partie entre crochets est remplacée par le résultat (17), puis la commande set x 17 est exécutée.
set t($i) 12	$i est remplacé par 5, puis l'élément d'indice 5 du tableau t reçoit la valeur 12.
set c $t(5)	$t(5) est remplacé par 12, puis c reçoit 12.
set c $t($i)	$i est remplacé par 5, puis $t(5) est remplacé par 12, puis c reçoit 12.
set a$b 5	$b est remplacé par 12, puis la variable de nom "a12" reçoit la valeur 5.
set coli $b	affecte la valeur de b à la variable dont le nom est la concaténation des valeurs des variables col et i (rouge5).

Groupage - Plusieurs mots peuvent être groupés en les mettant entre accolades {...} ou entre quo "...". Il n'y a pas de substitutions entre les accolades alors qu'il y a substitution entre les quotes. caractère spécial (espace, fin de ligne, $, [, etc.) peut être protégé par un backslash.

set msg "il fait beau"	affecte la valeur "il fait beau" à la variable msg
set msg il\ fait\ beau	affecte la valeur "il fait beau" à la variable msg
set script {set a $b}	affecte la valeur "set a $b" à la variable script
set script "set a $b"	affecta la valeur "set a 12" à la variable script
set msg plusieurs\ lignes	le backslash en fin de ligne permet d'écrire une commande sur plusieurs lignes
set msg {plusieurs lignes}	les accolades permettent d'écrire des commandes sur plusieurs lignes (très utilisé pour les structures de contrôle)

Listes - Un certain nombre de commandes de Tcl considèrent une valeur formée de plusieurs m comme une liste. Généralement, ces listes sont notées entre accolades : {bleu blanc rouge}. Elles peuv être imbriquées : {a {b c} d} est une liste de trois éléments dont le second est une liste de deux éléme La liste vide est notée {}.

list a {b c} d	construit la liste {a {b c} d}.
concat a {b c} d	construit la liste {a b c d} (met les arguments à plat).
lappend lst a b c	équivalent à : set lst [concat $lst a b c] (mais plus efficace).
llength $lst	retourne le nombre d'éléments de la liste lst.
lindex $lst $i	retourne le i-ème élément de lst. Le premier élément a l'indice 0. L'indice spécial end permet d'accéder au dernier élément de la liste (idem dans lrange, lreplace, linsert).
lrange $lst 1 3	retourne la liste constituée des éléments 1 à 3 de la liste lst.
lreplace $lst 1 3 a b	retourne la liste lst dans laquelle les éléments d'indice 1 à 3 sont remplacés par les éléments a b.
linsert $lst 2 a b	retourne la liste constituée des éléments de lst en insérant les éléments a b avant l'élément d'indice 2 dans lst.
lsort {bleu blanc rouge}	retourne la liste triée par ordre alphabétique.
lsearch $lst toto	recherche toto dans la liste lst et retourne son indice ou -1 s'il n'est pas trouvé.
split a/b/c/d /	retourne la liste {a b c d} : convertit une chaîne en liste en utilisant le second argument comme séparateur.
join $lst /	opération inverse de split : retourne la concatènation des mots de lst en utilisant / comme séparateur.

Expressions - La commande expr évalue une expression selon la syntaxe et les opérateurs du langage Il existe quelques fonctions prédéfinies (notamment abs(), sqrt() et les fonctions trigonométriques).

expr (-$b + sqrt($b*$b - 4*$a*$c)) / (2*$a)	calcule la racine d'un trinôme
expr $z != 0 \|\| ($x < 5 && $y < 5)	expression booléenne

Structures de contrôle - Les structures de contrôle sont des commandes Tcl "standard". L'utilisat judicieuse des accolades permet de les écrire sur plusieurs lignes. *Attention à toujours mettre l'accol ouvrante à la fin de la ligne et non pas en début de ligne suivante.*

```
if {$x < 0} {
        puts negatif
} else {
        set z 0
}
```

conditionnelle : le premier argument ($x < 0) est passé à expr. La partie else est optionnelle. On peut enchaîner plusieurs tests en utilisant elseif à la place de else.

```
while {$n > 0} {
        incr n -1
}
```

boucle : le test est évalué par expr.
incr incrémente la valeur de la variable n de -1
Le corps peut contenir des break et continue, comme en C.

```
foreach e $lst {
        puts $e
}
```

énumération d'une liste : e vaut les éléments successifs de la liste lst. On peut utiliser break et continue.

```
foreach {x y} $points {
        lineto $x $y
}
```

si on a plusieurs variables de boucle, on énumère plusieurs éléments de liste à chaque tour de boucle.

```
foreach x $lx y $ly  {
lineto $x $y
}
```

cette forme permet de parcourir plusieurs listes en parallèle. Si les listes ne sont pas de même longueur, elles sont complétées par des éléments vide.

```
switch -exact $car {
        a { puts a }
        b -
        c { puts "b ou c" }
        default { puts ?}
}
```

sélection par cas :
à chaque cas est associé un script.
si plusieurs cas partagent le même script, on utilise un "-"
cas par défaut (optionnel)

Définition de commandes - La commande proc permet de créer de nouvelles commandes qui s utilisées ensuite comme les commandes standard de Tcl. Toute variable utilisée dans une procédure locale à cette procédure. Il faut utiliser la commande global pour accéder aux variables globales.

```
proc fact {n} {         la liste de paramètres est entre accolades.
        if {$n <= 1} {  le corps de la commande est un script Tcl standard.
                return $n       la commande return permet de spécifier la valeur de retour.
        }
        return [expr $n * [fact [expr $n -1]]]
}
puts [fact 10]                  appel de la nouvelle commande.

set x 0                 initialisation de variable globale.
proc f {} {
        global x                déclaration de l'utilisation d'une variable globale.
        incr x                  sans la déclaration "global" c'est une variable locale de
        return $x               nom x qui serait utilisée...
}
```

Erreurs - Si une erreur est détectée à l'exécution, un message d'erreur est affiché et l'exécution s'arr On peut provoquer une erreur d'exécution par error, et on peut "attraper" une erreur d'exécution par cat

```
if {$x < 0} {
        error "x negatif"       provoque une erreur d'exécution avec pour message
}                               "x negatif".

catch {set a $b}                si une erreur a lieu (par exemple b n'est pas défini),
                                celle-ci est ignorée.
if [catch {set a $b}] { si une erreur a lieu, catch retourne une valeur non nulle.
        set b 0
}
if [catch {expr $a + $b} res] { si une erreur a lieu, res contient le message
        puts $res                       d'erreur, sinon il contient le résultat de
}                                       l'exécution de la commande.
```

Création de scripts - Pour pouvoir exécuter un script Tcl directement, il suffit de le rendre exécuta (chmod a+x fich.tcl) et de mettre au début du fichier la ligne suivante :

```
#! /usr/local/tcltk/bin/tclsh8.0
```

Si un script devient trop gros, il est préférable de le séparer en plusieurs fichiers. La commande sou permet de charger un script stocké dans un autre fichier :

```
source fich.tcl         exécute le script contenu dans le fichier fich.tcl
```

On peut se passer de la commande source en utilisant l'"auto-loading" : les fichiers seront chargés fonction des besoins. Pour cela :

1- dans le script principal, insérer la ligne suivante au début :

```
set auto_path [linsert $auto_path 0 tcl-lib]
```

2- mettre les fichiers à charger dans un directory, par exemple tcl-lib

3- aller dans ce directory (cd tcl-lib), lancer tclsh8.0 et exécuter la commande :

```
auto_mkindex . *.tcl
```

cette commande crée ou met à jour un fichier de nom tclIndex.

A chaque fois que l'on change les fichiers du directory tcl-lib, il faut refaire l'étape 3.

Autres commandes utiles - Consulter la documentation HTML et/ou le manuel pour avoir renseignements sur ces commandes.

```
string ...                      opérations sur les strings.
array ...                       opérations sur les tableaux.
info ...                opérations générales.
file ...                        opérations sur les fichiers (voir aussi open close puts gets)
eval command arg ...    évalue la commande (identique à [eval cmd arg ...]).
exit 0                          provoque la fin de l'exécution.
trace ...                       définit un script exécuté lorsqu'une variable est modifiée.
```