Breve manual de Tcl

Sergio Herrería Alonso

10 de diciembre de 2004

1. Introducción a Tcl

Tcl (*Tool Command Language*) es un lenguaje de programación de comandos muy popular para el desarrollo de pequeñas aplicaciones en entornos UNIX (aunque también existe una versión disponible para Windows). Permite programar de forma rápida y sencilla aplicaciones no demasiado complejas. Sin embargo, la velocidad de ejecución de éstas no será muy elevada ya que nos encontramos ante un lenguaje interpretado y no ante un lenguaje compilado.

Una característica muy importante de este lenguaje es la facilidad con la que se pueden añadir nuevos comandos a los ya existentes en el Tcl estándar. Estos nuevos comandos pueden implementarse utilizando el lenguaje de programación C e integrarse de manera sencilla en Tcl. Así, se han escrito bastantes extensiones para la realización de ciertas tareas comunes como, por ejemplo, OTcl (Tcl orientado a objetos) o Tk (que permite crear interfaces gráficas de usuario).

Se pueden ejecutar comandos Tcl de dos modos:

1. Modo interactivo: directamente, a través del intérprete de comandos *tclsh*, se pueden introducir y ejecutar comandos Tcl de forma interactiva.

2. Modo no interactivo: los comandos se guardan en un fichero. Se pueden ejecutar de dos maneras:

   - Llamando al intérprete de comandos de Tcl pasándole como parámetro el nombre del fichero.
   - Ejecutando directamente el fichero de comandos. Para que esto sea posible, la primera línea del fichero debe incluir el *path* hacia el intérprete de comandos (por ejemplo, `#!/usr/bin/tclsh`). Además, el fichero tiene que tener los permisos adecuados (permiso de ejecución).

Este documento no pretende ser una guia completa del lenguaje Tcl. Simplemente, se ofrece como manual de ayuda para la realización de las prácticas de la asignatura *Sistemas de Conmutación*. Para obtener más información sobre cómo utilizar un determinado comando Tcl, se recomienda consultar la página *man* correspondiente (`man n comando`) o cualquier libro que describa este lenguaje [1, 2].

2. Variables y valores

Tcl trata principalmente con cadenas de texto (*strings*). Cuando es necesario (por ejemplo, a la hora de realizar operaciones aritméticas), Tcl convierte automáticamente los *strings* en números.

Para crear una variable y asignarle un valor se utiliza el comando `set`:

```
% set a 100
```

Cuando el símbolo `$` precede al nombre de una variable, el intérprete sustituye dicha variable por su valor. Por ejemplo:

```
% puts $a
100
```

Para realizar operaciones aritméticas se utiliza el comando `expr`:

```
% expr 2*$a
200
```

Una cadena contenida entre corchetes ([cadena]) se considera que es un comando: la cadena se evalúa como si fuese un comando Tcl y se sustituye por el resultado obtenido. Esto no ocurre si la cadena está entre comillas ("cadena"). Para ejecutar realmente dicha cadena se requiere el comando `eval`:

```
% set b [expr 2*$a]
200
% set b "expr 2*$a"
expr 2*100
% eval $b
200
```

3. Estructuras de control

Los comandos de control de flujo son similares a sus equivalentes en el lenguaje C. Los principales operadores de comparación son: `<`, `>`, `<=`, `>=`, `==`, `!=`, `&&`, `||`, `!`. Los ejemplos siguientes pueden ser útiles para ilustrarnos en la utilización de estos comandos.

- **if**

```
% set i 1
% if {$i<0} {
puts "Negativo"
} elseif {$i==0} {
puts "Cero"
} else {
puts "Positivo"
}
Positivo
```

- **for**

```
% for {set i 1} {$i<=3} {incr i} {
puts $i
}
1
2
3
```

El comando `break` finaliza la ejecución del bucle inmediatamente. El comando `continue` fuerza a que se ejecute la iteración siguiente (evidentemente, siempre y cuando se cumpla la condición de continuación del bucle).

- **while**

```
% set i 1
% while {$i<=3} {
puts $i
incr i
}
1
2
3
```

4. Operaciones con cadenas de texto

- **string length**
  Devuelve el número de caracteres que forman una cadena.

```
% string length "ABCD"
4
```

- **string index**
  Devuelve el carácter situado en una determinada posición dentro de una cadena. Al primer carácter le corresponde el índice 0. Podemos referirnos al último carácter de una cadena mediante la palabra `end`.

```
% string index "ABCD" 1
B
% string index "ABCD" end
D
```

- **string range**
  Devuelve la subcadena formada por las caracteres que se encuentran entre dos posiciones de una cadena.

```
% string range "ABCDEF" 3 end
DEF
```

- **string first, last**
  Busca la primera (última) ocurrencia de una subcadena en una cadena y devuelve el índice a partir del cual se encuentra dicha subcadena (si no se encuentra, devuelve -1).

```
% string first "CD" "ABCDABCD"
2
% string last "CD" "ABCDABCD"
6
```

- **string map**
  Reemplaza en una cadena las subcadenas indicadas por otras nuevas.

```
% string map {AB 12 C 3 D 4} "ABCD"
1234
```

- **string match**
  Devuelve 1 si la cadena coincide con el patrón proporcionado (o 0 en caso contrario). Para obtener completa información sobre la especificación de patrones se puede consultar la ayuda de la herramienta `flex`.

```
% string match "*B*" "ABC"
1
```

- **regexp**
  Permite buscar patrones en una cadena y asignarlos a variables.

  ```
  % set cadena "El resultado es 100."
  % regexp {[0-9]+} $cadena resultado
  % puts $resultado
  100
  ```

5. Arrays

Los arrays en Tcl son asociativos: cualquier cadena puede ser índice de un array.

```
% set miarray(0) 0
% set miarray(elemento) 1
% puts $miarray(0)
0
% puts $miarray(elemento)
1
```

Los comandos más importantes para manejar arrays son: `array exists`, `array get`, `array names`, `array size`...

6. Listas

Son agrupaciones de elementos de Tcl. Cualquier elemento Tcl puede formar parte de una lista, incluido otras listas. La sintaxis para crear una lista es la siguiente:

```
% set meses {enero febrero marzo}
```

- **llength**
  Devuelve el número de elementos que forman la lista.

  ```
  % llength $meses
  3
  ```

- **lindex**
  Devuelve el elemento que ocupa una posición determinada en una lista. Al elemento inicial de la lista le corresponde el índice 0.

  ```
  % lindex $meses 1
  febrero
  ```

- **linsert**
  Permite añadir elementos a una lista a partir de una posición determinada.

  ```
  % linsert $meses end abril mayo
  enero febrero marzo abril mayo
  % puts $meses
  enero febrero marzo
  ```

Atención: este comando no modifica la lista `meses`, sino que devuelve una lista con los nuevos elementos añadidos.

- **lappend**
  Añade elementos a una lista a partir del final de la misma. Este comando sí modifica realmente la lista.

```
% lappend meses abril mayo
enero febrero marzo abril mayo
% puts $meses
enero febrero marzo abril mayo
```

- **lsearch**
  Busca elementos de la lista que verifiquen un determinado patrón. Devuelve el índice del primer elemento que cumple con el patrón o -1 si no se encuentra ningún elemento.

```
% lsearch $meses feb*
1
```

- **lreplace**
  Devuelve la lista resultado de reemplazar en una lista los elementos seleccionados por otros nuevos.

```
% lreplace $meses 0 1 ENERO FEBRERO
ENERO FEBRERO marzo abril mayo
```

  Si no se especifican los nuevos elementos, simplemente se eliminan los elementos seleccionados.

```
% lreplace $meses 0 1
marzo abril mayo
```

- **split**
  Crea una lista a partir de una cadena. El delimitador por defecto es el espacio en blanco.

```
% split "uno dos tres"
uno dos tres
% split "uno,dos,tres" {,}
uno dos tres
```

- **foreach**
  Este comando asigna a una variable un elemento de una lista en cada paso.

```
% foreach i {1 2 3} {
    puts $i
}
1
2
3
```

7. Procedimientos

Un procedimiento se define de la siguiente manera:

```
proc nombre_proc {arg1 arg2 ...} {
  ...
 return $var
}
```

Por defecto, todas las variables creadas dentro de un procedimiento son locales a dicho procedimiento. Para acceder a variables globales:

- `global variable`: permite acceder dentro de un procedimiento a una variable global.
- `upvar oldvar newvar`: hace que la variable global *oldvar* sea accesible en el procedimiento actual a través de una variable de nombre *newvar*.

8. Comandos de entrada/salida

- **open**
  Abre un fichero y devuelve un identificador de canal que se puede utilizar en futuras llamadas a otros comandos de E/S.

  ```
  % set fichero [open "fichero.txt" r]
  ```

  Se permiten los siguientes modos de acceso:

  - `r`: sólo lectura (el fichero debe existir).
  - `r+`: lectura y escritura (el fichero debe existir).
  - `w`: sólo escritura (si existe, lo sobreescribe).
  - `w+`: lectura y escritura (si existe, lo sobreescribe).
  - `a`: sólo escritura (el fichero debe existir, añade datos al final del fichero).
  - `a+`: lectura y escritura (el fichero debe existir, añade datos al final del fichero).

  Si el primer carácter del nombre del fichero es el símbolo `|`, el `open` funciona como una tubería: permite ejecutar programas externos y que su salida esté disponible a través del identificador devuelto:

  ```
  % set listado [open "| ls" r]
  ```

- **close**
  Este comando cierra el canal establecido para manejar un fichero.

  ```
  % close $fichero
  ```

- **gets**
  Permite leer líneas de un fichero. Devuelve el número de caracteres leidos o -1 si se produce un error.

  ```
  % gets $fichero linea
  ```

- **puts**
  Permite escribir líneas en un fichero. Con la opción `nonewline` se evita escribir el salto de línea.

```
% puts $fichero "1 2 3"
```

- **file**
  Permite comprobar el estado de un fichero. Se puede ejecutar con varias opciones: `exists`, `executable`, `extension`, `type`, `size`.

  ```
  % file exists "fichero.txt"
  1
  ```

9. Otros comandos

- **Comentarios**

  Toda línea que empieza con el carácter `#` se considerará un comentario ignorándose su contenido.

- **Parámetros pasados a la aplicación**

  Los parámetros que se le pasan a una aplicación Tcl a través de la línea de comandos se guardan en una lista denominada `argv`. Por tanto, para obtener el primer parámetro:

  ```
  % lindex $argv 0
  ```

- **Ejecución de programas externos**

  Se pueden ejecutar programas externos a la aplicación Tcl de dos maneras:

  - Mediante el comando `open`: ver sección 8.1.
  - Mediante el comando `exec`: `exec comando`.

- **Generación de números aleatorios**

  El comando `srand` permite inicializar la semilla del generador de números aleatorios. El comando `rand` genera un número aleatorio entre 0 y 1. Por ejemplo:

  ```
  % expr srand(1)
  7.82636925943e-06
  % expr rand()
  0.131537788143
  ```

Referencias

[1] Ousterhout, John K., *Tcl and the Tk toolkit*, Addison-Wesley Publishing Company, 1994.

[2] Harrison, Mark, *TCL/TK tools*, O'Reilly, 1997.