IntroducciÃ³n a la biblioteca TclMySQL

by Diego Alberto Arias Prado
<dariapra(at)yahoo.es>

About the author:

Soy un ingeniero de TelecomunicaciÃ³n de Lugo, EspaÃ±a. No recuerdo exactamente cuÃ¡ndo empecÃ© a utilizar Linux, fue en 1995 Ã³ 1996. Antes era un usuario de Microsoft Windows y ni siquiera sabÃ-a de la existencia de Linux. La primera vez que vi un ordenador con sistema operativo Linux instalado fue en la universidad. Me pareciÃ³ muy interesante y no tardÃ© en instalarlo en mi ordenador en casa; recuerdo que mi primera distribuciÃ³n Linux fue Slackware.

Durante todos estos aÃ±os me he divertido mucho utilizando otras distribuciones Linux y algunas versiones de BSD, programando en lenguajes como Java o Tcl, utilizando servidores de web, de bases de datos, de aplicaciones... Linux no ha sido sÃ³lo para mÃ- una forma de divertirme ya que he tenido la

Abstract:

En este artÃ-culo se muestra la instalaciÃ³n y uso de MySQLTcl, una biblioteca del lenguaje de programaciÃ³n Tcl que permite hacer consultas SQL – en inglÃ©s, *queries* – (*select*, *insert*, *delete*...) desde *scripts* Tcl a servidores de bases de datos MySQL.

Tcl son las siglas de *Tool Command Language*, un lenguaje de programaciÃ³n de tipo *scripting* creado por John Ousterhout [1]. En realidad, Tcl consiste en dos cosas: ademÃ¡s de ser un lenguaje de programaciÃ³n de tipo *scripting*, es tambiÃ©n un intÃ©rprete de *scripts* Tcl. Tcl es un lenguaje de programaciÃ³n estructurado que utiliza tres estructuras con tres estructuras de datos bÃ¡sicas: cadenas (*strings*), listas y arrays. Otras caracterÃ-sticas de Tcl son el uso de expresiones regulares [2], la posibilidad de poder utilizar bibliotecas Tcl escritas por terceras personas y Tk, un *toolkit* que permite la creaciÃ³n de aplicaciones Tcl grÃ¡ficas.

MySQL es un servidor de bases de datos muy popular en el mundo del software libre que, en mi opiniÃ³n, no necesita presentaciÃ³n.

MySQLTcl es una biblioteca Tcl que permite hacer consultas a un servidor de bases de datos MySQL desde *scripts* Tcl. Actualmente, Paolo Brutti, Tobias Ritzau and Artur Trzewick son los autores y quienes mantienen esta biblioteca Tcl.

opotunidad de utilizar Linux
cuando trabajÃ© en
TelefÃ³nica I+D.

InstalaciÃ³n de la biblioteca MySQLTcl

Si su distribuciÃ³n Linux o sistema operativo BSD utiliza un sistema de paquetes (como por ejemplo RPM o DEB) o un sistema de *ports* (como por ejemplo Crux Linux o FreeBSD), puede usar el sistema de paquetes o *ports* para instalar la biblioteca MySQLTcl y pasar asÃ- a la siguiente secciÃ³n.

Si no es asÃ- o, simplemente, prefiere hacer la instalaciÃ³n de forma manual, en las siguientes lÃ-neas se muestran los pasos que hemos seguido. En vez de un pequeÃ±o manual que muestra paso a paso cÃ³mo hacer la instalaciÃ³n, estas lÃ-neas deben leerse como una especie de pequeÃ±a guÃ-a. En una distribuciÃ³n Linux Mandrake (version 9.2), desde bash:

```
$ ./configure --with-mysql-lib=/usr/lib
$ make
$ make install
```

Si algo ha ido mal durante el paso "configure", los mensajes de error le darÃ¡n informaciÃ³n que le ayudarÃ¡ a resolver el problema. Es frecuente que Ã©ste consista en que el *script* no consigue encontrar algunos ficheros o directorios. En casos como Ã©ste, la soluciÃ³n consiste en "jugar" con este *script* pasÃ¡ndole parÃ¡metros que le indiquen dÃ³nde puede encontrar esos ficheros y directorios "perdidos". Veamos otro ejemplo de instalaciÃ³n. En FreeBSD 5.0 la biblioteca MySQLTcl se puede instalar con las siguientes opciones:

```
$ export CPP=/usr/bin/cpp
$ ./configure --with-tcl=/usr/lib/local/tcl8.3
 --with-tclinclude=/usr/local/include/tcl8.3
 --with-mysql-include=/usr/local/include/mysql
 --with-mysql-lib=/usr/local/lib/mysql
$ make
$ make install
```

Como habrÃ¡ notado, en este segundo ejemplo la versiÃ³n de Tcl era 8.3; ademÃ¡s, la versiÃ³n de la biblioteca MySQLTcl era 2.15 y la versiÃ³n del servidor de bases de datos MySQL era 3.23.54.

Rudimentos de Tcl

En esta secciÃ³n se muestran de forma somera los rudimentos de Tcl a lectores que **no** sepan utilizar este lenguaje de programaciÃ³n. Si usted ya sabe programar en Tcl, puede pasar a la siguiente secciÃ³n.

Los ejemplos mostrados en esta secciÃ³n (y tambiÃ©n en las demÃ¡s) pueden ser reproducidos desde una consola Tcl (tclsh).

Variables. SustituciÃ³n de variables y comandos.

En Tcl las variables se crean con el comando *set*. Veamos algunos ejemplos:

```
% set address {Edison Avenue, 83}
```

```
Edison Avenue, 83
% set zip_code 48631
48631
```

En estos dos ejemplos hemos creado dos variables, llamadas *address* y *zip_code*. Los valores que estas variables contienen son, respectivamente, *Edison Avenue* y *48361*; ambos valores son cadenas (*strings*). NÃ³tese que para la creaciÃ³n de la variable *address* se han utilizado llaves (caracteres *{* y *}*); esto es asÃ- porque dicha cadena contiene espacios. Los valores de las variables se pueden obtener utilizando tambiÃ©n el comando *set*:

```
% set address
Edison Avenue, 83
% set zip_code
48631
```

Supongamos que queremos imprimir en pantalla el valor de la variable *address*. Esto se puede hacer utilizando el comando *puts*:

```
% puts stdout [set address]
Edison Avenue, 83
```

El comando *puts* recibe el parÃ¡metro *stdout*. Este parÃ¡metro indica al comando *puts* que debe imprimir en la salida standard (en inglÃ©s, *standard output*), que en nuestro caso es el monitor. El segundo parÃ¡metro que recibe el comando *puts* es *[set address]*. Los corchetes (caracteres *[* y *]*) de este parÃ¡metro indican al intÃ©rprete Tcl que lo que hay dentro es otro comando Tcl que debe ser ejecutado por el intÃ©rprete Tcl antes de que lo sea el comando *puts*; esto recibe el nombre de substituciÃ³n de comandos. Lo hecho en el anterior ejemplo se puede hacer tambiÃ©n de otra forma:

```
% puts stdout $address
Edison Avenue, 83
```

Lo que se ha hecho en este ejemplo se llama substituciÃ³n de variable: el caracter *$* que precede a un nombre de variable hace que dicha substituciÃ³n ocurra.

En un ejemplo anterior hemos podido ver como utilizando llaves, distintas palabras separadas por espacios podÃ-an ser agrupadas para formar una cadena. Existe otra forma de agrupaciÃ³n, que consiste en utilizar comillas (el caracter *"*) en vez de llaves. Sin embargo, estas dos formas de agrupaciÃ³n no funcionan igual. VeÃ¡moslo con un ejemplo:

```
% puts stdout "the zip code is [set address]"
the zip code is 48631
% puts stdout "the zip code is $address"
the zip code is 48631
% puts stdout {the zip code is [set address]}
the zip code is [set address]
% puts stdout {the zip code is $address}
the zip code is $address
```

En este ejemplo se puede ver como al utilizar llaves para formar una agrupaciÃ³n **no** se hace substituciÃ³n de comandos; sin embargo, esto **sÃ-** ocurre si se utilizan comillas para formar dicha agrupaciÃ³n.

Para borrar variables se utiliza el comando *unset*:

```
% unset address
% set address
can't read "address": no such variable
% unset zip_code
```

```
% set zip_code
can't read "zip_code": no such variable
```

Cadenas en Tcl

La cadena (*string*) es una de las tres estructuras de datos bÃ¡sicas en Tcl. Una cadena es un conjunto de caracteres. Para crear una cadena se puede utilizar el comando *set*.

```
% set surname Westmoreland
Westmoreland
% set number 46.625
46.625
```

Tanto la variable *surname* como la variable *number* son cadenas. Para manipular cadenas se utiliza el comando *string*. Veamos algunos ejemplos que muestran cÃ³mo se puede utilizar este comando:

```
% string length $surname
12
% set surname [string range $surname 0 3]
West
% puts stdout $surname
West
% string tolower $surname
west
```

A diferencia de Pascal o Java, Tcl no es lenguaje de programaciÃ³n de "tipos fuertes". El siguiente ejemplo es una muestra de ello:

```
% set number [expr $number + 24.5]
70.125
% string range $number 2 5
.125
```

Con el comando *expr*, el valor de la variable *number* fue incrementado en 24,5. A continuaciÃ³n, con el comando *string*, la variable *number* fue tratadada como si fuese una cadena, siendo mostrados sus cuatro Ãºltimos caracteres.

El comando *string* permite hacer mÃ¡s manipulaciones con cadenas que las mostradas en los anteriores ejemplos.

Listas en Tcl

En Tcl, las listas (*lists*) son un caso especial de cadenas en las que los elementos de la lista estÃ¡n separados por espacios en blanco y que, ademÃ¡s, tienen una interpretaciÃ³n distinta.

```
% set friends_list {Fany John Lisa Jack Michael}
Fany John Lisa Jack Michael
% set friends_list [string tolower $friends_list]
fany john lisa jack michael
% set friends_list
fany john lisa jack michael
```

En Tcl hay varios comandos que permiten manipular las listas. Veamos algunos ejemplos:

```
% lindex 2 $friends_list
```

```
lisa
% lrange $friends_list 2 4
lisa jack michael
% set friends_list [lsort -ascii $friends_list]
fany jack john lisa michael
% set friends_list
fany jack john lisa michael
% set friends_list [linsert $friends_list 2 Peter]
fany jack Peter john lisa michael
% string toupper $friends_list
FANY JACK PETER JOHN LISA MICHAEL
```

El Ãºltimo ejemplo muestra que cadenas (*strings*) y listas (*lists*) en el fondo son la misma estructura de datos.

Arrays en Tcl

Un *array* Tcl se puede ver como una lista cuyos Ã-ndices son, a su vez, cadenas. Un *array* se puede crear como se muestra en el siguiente ejemplo:

```
% set phone_numbers(fany) 629
629
% set phone_numbers(michael) 513
513
% set phone_numbers(john) 286
286
```

Los valores del *array* se pueden mostrar utilizando conjuntamente el comando *set* y substituciÃ³n de variables; como se mostrÃ³ en el anterior ejemplo, el Ã-ndice del *array* – que es una cadena – estÃ¡ entre parÃ©ntesis:

```
% set $phone_numers(michael)
513
```

El comando *array* muestra informaciÃ³n sobre una variable de tipo *array*. Veamos algunos ejemplos que muestran quÃ© se puede hacer con este comando:

```
% array size phone_numbers
3
% array names phone_numbers
fany john michael
```

Conexiones a bases de datos

Antes de hacer cualquier consulta a una base de datos desde un *script* Tcl, es necesario establecer una conexiÃ³n a un servidor de bases de datos. En esta secciÃ³n veremos cÃ³mo se hace y quÃ© hacer con los errores que se pueden dar al intentar establecer dichas conexiones.

De ahora en adelante se utilizarÃ¡n indistintmente los tÃ©rminos "conexiÃ³n a bases de datos" y "conexiÃ³n a servidores de bases de datos".

Estableciendo una conexiÃ³n a una base de datos

Veamos el ejemplo de un *script* Tcl que establece una conexiÃ³n con un servidor de bases de datos MySQL.

```
0:  #!/usr/bin/tclsh8.4
1:
2:  package require mysqltcl
3:  global mysqlstatus
4:
5:  set port {3306}
6:  set host {127.0.0.1}
7:  set user {john_smith}
8:  set password {jsmith_password}
9:
10: set mysql_handler [mysqlconnect -host $host
    -port $port -user $user -password $password]
11:
12: mysqlclose $mysql_handler
```

NÃ³tese que los nÃºmeros mostrados en la columna izquierda junto con los dos puntos a continuaciÃ³n **no** son parte del *script* Tcl; se muestran para numerar las lÃ-neas del *script*. NÃ³tese tambiÃ©n que dependiendo de la distribuciÃ³n Linux utilizada es posible que haya que modificar la lÃ-nea nÃºmero 0 cambiando el *path* del intÃ©rprete Tcl.

La lÃ-nea nÃºmero 0 le dice al *shell* que el fichero es un *script* Tcl y dÃ³nde encontrar el intÃ©rprete Tcl.

La lÃ-nea nÃºmero 2 le dice al intÃ©prete Tcl que busque en la biblioteca MySQLTcl cuando ejecute los restantes comandos del *script*. Por ejemplo, en la lÃ-nea nÃºmero 10 podemos ver el comando *mysqlconnect*; si la lÃ-nea nÃºmero 2 no existiese, al ejecutar el intÃ©rprete Tcl la lÃ-nea nÃºmero 10 el comando *mysqlconnect* no serÃ-a encontrado y habrÃ-a un error.

En las lÃ-neas nÃºmero 5 y 6 se fijan el puerto (*port*) y el *host* con el que el *script* Tcl intentarÃ¡ establecer la conexiÃ³n. En este *script* el nÃºmero de puerto es 3306 (el puerto por defecto en el que un servidor de bases de datos MySQL "escucha") y el *host* es la misma computadora en la que el *script* se estÃ¡ ejecutando.

En las lÃ-neas nÃºmero 7 y 8 se fijan el nombre del usuario de la base de datos y su *password*.

En la lÃ-nea nÃºmero 10 es donde realmente se establece la conexiÃ³n con el servidor de bases de datos. La salida del comando *mysqlconnect* se guarda en la variable de nombre *mysql_handler*. Esta variable – a la que llamaremos *handler* – se utiliza para hacer consultas SQL a las bases de datos y tambiÃ©n para cerrar la conexiÃ³n, tal y como se muestra en la lÃ-nea nÃºmero 12.

Manejo de los errores

En la anterior subsecciÃ³n no se explicÃ³ quÃ© hacÃ-a la lÃ-nea nÃºmero 3; lo haremos en Ã©sta.

Al ejecutarse los comandos de la biblioteca MySQLTcl pueden aparecer errores. Si dichos errores no son capturados (*caught*), la ejecuciÃ³n del *script* Tcl se detendrÃ¡ y es posible que eso no nos interese.

```
10: if [catch {mysqlconnect -host $host -port $port
    -user $user -password $password} mysql_handler] {
11:     puts stderr {error, the database connection could not be established}
12:     exit
13: } else {
14:     mysqlclose mysql_handler
15: }
```

Si durante la ejecuciÃ³n del comando *set mysql_handler [mysqlconnect –host $host...* se produce algÃºn error, dicho error serÃ¡ capturado por el comando *catch*. El comando *catch* devuelve 1 si al ejecutarse el comando que hay entre las llaves aparece un error o 0 si **no** hay ningÃºn error. La variable *mysql_handler* guarda la salida del comando ejecutado que estÃ¡ entre las llaves (caracteres *{* y *}*).

Si hay algÃºn error, se imprime en la salida standard de error (*stderr*) – que serÃ¡, en nuestro caso, el monitor – un mensaje de error. Las posibles causas por las que puede haber un error al intentar establecer una conexiÃ³n con una base de datos son varias: *password* incorrecto, *host* o nÃºmero de puerto incorrectos... En casos asÃ-, el disponer de mÃ¡s informaciÃ³n que la dada por un mensaje del tipo "se ha producido un error" puede ser Ãºtil.

En la lÃ-nea nÃºmero 3, la variable *mysqlstatus* se declarÃ³ global. Las variables globales son aquellas que son accesibles desde cualquier parte de un *script* Tcl; esto tiene que ver con el Ã¡mbito (*scope*) de las variables Tcl, tema que no trataremos en este artÃ-culo. La biblioteca MySQLTcl crea y mantiene un *array* global llamado *mysqlstatus* que tiene los siguientes elementos:

elemento	significado
code	Si no ha habido ningÃºn error, entonces *mysqlstatus(code)* contiene 0; en caso contrario, el contenido de *mysqlstatus(code)* serÃ¡ el cÃ³digo nÃºmerico MySQL del error. Si ha habido un error y no tiene su origen en el servidor de bases de datos MySQL, entonces el contenido de *mysqlstatus(code)* serÃ¡ –1. El contenido de *mysqlstatus(code)* se actualiza despÃºes de la ejecuciÃ³n de cualquier comando de la biblioteca MySQLTcl.
command	El Ãºltimo comando de la biblioteca MySQLTcl que produjo un error se guarda en *mysqlstatus(command)*. El contenido de *mysqlstatus(command)* se actualiza despuÃ©s de la ejecuciÃ³n sin Ã©xito de cualquier comando de la biblioteca MySQLTcl; por lo tanto, el contenido de *mysqlstatus(command)* no se actualiza despuÃ©s de la ejecuciÃ³n sin error – esto es, con Ã©xito – de cualquier comando de la biblioteca MySQLTcl.
message	El contenido de *mysqlstatus(message)* se actualiza despuÃ©s de la ejecuciÃ³n sin Ã©xito de un comando cualquiera de la biblioteca MySQLTcl con una cadena que contiene un mensaje de error. Al igual que *mysqlstatus(command)*, *mysqlstatus(message)* **no** se actualiza despuÃ©s de la ejecuciÃ³n sin error – esto es, con Ã©xito – de cualquier comando de la biblioteca MySQLTcl.

Existe otro elemento en el *array* global *mysqlstatus* que no tiene relaciÃ³n con el manejo de errores:

elemento	significado
nullvalue	Cadena utilizada para representar el valor *null* cuando se muestran resultados de una consulta SQL.. El valor por defecto es cadena vacÃ-a y puede ser cambiado a la cadena que queramos.

Utilizando el *array* global *mysqlstatus*, el anterior fragmento de cÃ³digo se podrÃ-a reescribir para que mostrase al usuario mÃ¡s informaciÃ³n cuando se produjese un error al intentar establecer la conexiÃ³n con el servidor de bases de datos:

```
10: catch {mysqlconnect -host $host -port $port
    -user $user -password $password} mysql_handler
11: if {$mysqltatus(code) != 0} {
12:     puts stderr $mysqlstatus(message)
13: } else {
```

```
14:     mysqlclose mysql_handler
15: }
```

Huelga decir que el *array* global *mysqlstatus* se puede utilizar para manejar mÃ¡s errores que los que se puedan producir al intentar establecer una conexiÃ³n con el servidor de bases de datos.

Comandos bÃ¡sicos de la biblioteca MySQLTcl

En esta secciÃ³n se muestra el uso de los comando mÃ¡s bÃ¡sicos de la biblioteca MySQLTcl con algunos ejemplos. Para mayor informaciÃ³n, consulte la pÃ¡gina *man* de la biblioteca MySQLTcl.
Los comandos cuyo uso se muestra en esta secciÃ³n son los que aparecen en la siguiente tabla. Los parÃ¡metros estÃ¡n subrayados. Si un parÃ¡metro se muestra entre dos sÃ-mbolos *?*, significa que es opcional; el caracter | significa o lÃ³gico (*or*).

comando	breve descripciÃ³n
mysqlconnect ?option value ...?	establece una conexiÃ³n con un servidor de bases de datos; devuelve un *handler* que debe ser utilizado por otros comandos de la biblioteca MySQLTcl
mysqluse handle dbname	asocia un *handler* MySQL con la base de datos especificada por *dbname*
mysqsel handle consulta_sql ?–list | –flatlist?	envÃ-a una consulta SQL a una base de datos
mysqlexec handle sql_statement	envÃ-a una consulta SQL distinta de *select* a una base de datos
mysqlclose handle	cierra una conexiÃ³n establecida a un servidor de bases de datos

mysqlconnect

El uso de este comando ya fue tratado en la secciÃ³n "Estableciendo una conexiÃ³n a una base de datos". Este comando acepta un parÃ¡metro extra no mostrado antes: *–db*. Con el parÃ¡metro *–db* se fija la base de datos que serÃ¡ objeto – o dicho de otra forma, base de datos asoaciada al *handler* – de futuras consultas SQL. Veamos un ejemplo:

```
% set mysql_handler [mysqlconnect -h 127.0.0.1 -p 3306 \\
  -user john_smith -password jsmith_password -db jsmith_database]
```

La base de datos objetivo de los comandos de la biblioteca MySQLTcl que utilicen el *handler mysql_handler* serÃ¡ la que tiene por nombre *jsmith_database*.

(Note que los caracteres \\ **no** son parte del comando; significan que que la siguiente lÃ-nea tambiÃ©n es parte del comando.).

mysqluse

Este comando permite cambiar la base de datos asociada del *handler* MySQL que este comando recibe como primer parÃ¡metro.

mysqlsel

Este comando envÃ-a una consulta SQL a la base de datos asociada al *handler* MySQL que este comando recibe como primer parÃ¡metro. Si el parÃ¡metro *sql_statement* no es una consulta SQL *select*, se producirÃ¡ un error.

Hay un tercer parÃ¡metro opcional, que puede ser *list* o *flat_list*. Veamos con un ejemplo cÃ³mo afecta este parÃ¡metro a la salida de este comando. Supongamos que la base de datos asociada con el *handler* MySQL contiene una tabla como la mostrada a continuaciÃ³n:

id	first_name	last_ name	phone
26	Karl	Bauer	8245
47	James	Brooks	1093
61	Roberto	Castro Portela	6507

Utilizamos el comando *mysqlsel* para enviar una consulta SQL a la base de datos:

```
% mysqlsel $mysql_handler {select first_name, last_name from people order by id asc} -list
{Karl Bauer} {James Brooks} {Roberto {Castro Portela}}
% mysqlsel $mysql_handler {select first_name, last_name from people order by id asc} -flatlist
Karl Bauer James Brooks Roberto {Castro Portela}
```

En el primer ejemplo (*–list parameter*), el comando devuelve una lista cuyos elementos son, a su vez, listas. En el segundo ejemplo (*–flatlist parameter*), el comando devuelve una Ãºnica lista cuyos elementos han sido concatenados en su totalidad.

Â¿QuÃ© ocurre si el comando *mysqlsel* no recibe un tercer parÃ¡metro? En este caso, la salida del comando *mysqlsel* es el nÃºmero de filas resultado de la consulta SQL:

```
% mysqlsel $mysql_handler {select first_name, last_name from people order by id asc}
3
```

mysqlexec

El comando *mysqlexec* envÃ-a una consulta SQL que **no** es *select* a la base de datos asociada con el *handler* que el comando recibe como primer parÃ¡metro. Si el parÃ¡metro *sql_statement* es una consulta SQL *select* no se producirÃ¡ ningÃºn error, pero tampoco el comando harÃ¡ nada.

Supongamos que tenemos el ejemplo mostrado en la anterior subsecciÃ³n:

```
% mysqlexec $mysql_handler {delete from people where id=26}
1
% mysqlsel $mysql_handler {select first_name, last_name, phone from people order by id asc} -list
{James Brooks 1093} {Roberto {Castro Portela} 6507}
% mysqlexec $mysql_handler \\
  {insert into people (id, first_name, last_name, phone) values (58, "Carla", "di Bella", 8925)}
1
% mysqlsel $mysql_handler {select first_name, last_name, phone from people order by id asc} -list
{James Brooks 1093} {Carla {di Bella} 8925} {Roberto {Castro Portela} 6507}
```

Obviamente, se pueden hacer otras consultas SQL que no sean ni *delete* ni *insert*. Por ejemplo, es posible actualizar (*update*) el valor de una fila ya existente:

```
% mysqlexec $mysql_handler {update people set phone=3749 where first_name="James"}
1
% mysqlsel $mysql_handler {select first_name, last_name, phone from people order by id asc} -list
{James Brooks 3749} {Carla {di Bella} 8925} {Roberto {Castro Portela} 6507}
```

En caso de que no haya ningÃºn error, el comando *mysqlexec* devuelve el nÃºmero de filas afectdas por la consulta SQL *non−select* enviada a la base de datos.

mysqlclose

Como se vio con anteriodad, el comando *mysqlclose* cierra una conexiÃ³n ya establecida a una base de datos. Este comando recibe un parÃ¡metro, el *handler* MySQL de la conexiÃ³n que se quiere cerrar.

Otros comandos de la biblioteca MySQLTcl

La biblioteca MySQLTcl tiene mÃ¡s comandos que los vistos en esta secciÃ³n. Estos comandos permiten obtener informaciÃ³n sobre las propias bases de datos, transformar (*escaping*) cadenas para hacerlas aptas ser parte de consultas SQL, hacer consultas SQL anidadas... Una buena referencia es la propia pÃ¡gina *man* de MySQLTcl, que es parte de la instalaciÃ³n de esta biblioteca.

Referencias

[1] una visiÃ³n ligeramente escÃ©ptica sobre John K. Ousterhout y Tcl (en inglÃ©s):
http://www.softpanorama.org/People/Ousterhout/index.shtml

[2] tutorial sobre las expresiones regulares en Tcl (en inglÃ©s):
http://www.mapfree.com/sbf/tcl/book/select/Html/7.html

TclTutor es una aplicaciÃ³n gratuÃ-ta e interactiva para aprender Tcl (en inglÃ©s):
http://www.msen.com/~clif/TclTutor.html

documentaciÃ³n de MySQL, en varios formatos (HTML, PDF...) e idiomas:
http://www.mysql.com/documentation/index.html

Webpages maintained by the LinuxFocus Editor team © Diego Alberto Arias Prado "some rights reserved" see linuxfocus.org/license/ http://www.LinuxFocus.org	Translation information: en −−> −− : Diego Alberto Arias Prado <dariapra(at)yahoo.es> en −−> es: Diego Alberto Arias Prado <dariapra(at)yahoo.es>

2005−01−10, generated by lfparser_pdf version 2.51