Eine Einführung in die TclMySQL−Bibliothek

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

About the author:

Ich bin Ingenieur für Telekommunikation und wohne in Lugo, Spanien. Ich weiss nicht mehr genau, wann ich mit Linux begann, es war 1995 oder 1996. Vorher benutzte ich Microsoft Windows und ich wusste sogar gar nicht, dass es Linux gab. Einen Computer unter Linux sah ich erstmals an der Universität. Es sah für mich sehr interessant aus und kurz danach installierte ich es auf meinem Computer zu Hause. Meine erste Linux−Distribution war Slackware.

In all diesen Jahren hatte ich viel Spaß mit anderen Linux−Distributionen und einigen BSD−"Geschmacksrichtungen", Programmiersprachen wie Java oder Tcl, und der Benutzung von Datenbank−, Web− und Anwendungsservern. Linux war nicht nur Spaß für mich, ich hatte auch die Möglichkeit, Linux in meiner Arbeitsstätte Telefónica I+D zu nutzen.

Abstract:

Dieser Artikel beschäftigt sich mit der Installation und Benutzung von MySQLTcl, einer Tcl−Bibliothek, die die Anwendung von SQL−Abfragen (select, insert, delete...) auf einen MySQL−Datenbank−Server aus Tcl−Skripten heraus ermöglicht. Die benutzten Versionen von Tcl, MySQL−Server und der MySQLTcl−Bibliothek sind entsprechend 8.4.2, 4.0.15 and 2.40.

Tcl steht für Tool Command Language und wurde von John Ousterhout [1] erfunden. Tcl besteht eigentlich aus zwei Teilen: einer Skriptsprache und einem Interpreter. Tcl ist eine strukturierte Programmiersprache, die drei grundlegende Datenstrukturen nutzt: Zeichenketten, Listen und Felder. Die Eigenschaften von Tcl umfassen reguläre Ausdrücke [2], Tcl−Erweiterungsbibliotheken von Drittanbietern und Tk, eine Werkzeugsammlung zum Schreiben graphischer Anwendungen in Tcl.

MySQL ist ein in der "Open Source"−Gemeinschaft sehr populärer Datenbankserver, der keiner besonderen Hervorhebung bedarf.

MySQLTcl ist eine Tcl−Bibliothek, die die Abfrage eines MySQL−Datenbankservers aus einem Tcl−Skript heraus erlaubt. Die derzeitigen Autoren und Verwalter dieser Tcl−Bibliothek sind Paolo Brutti (Paolo.Bruti at tlsoft.it), Tobias Ritzau (tobri at ida.liu.se) und Artur Trzewick (mail at xdobry.de).

Installation der MySQLTcl−Bibliothek

Falls Ihre Linux−Distribution oder Ihr *BSD−Betriebssystem über Unterstützung für Pakete (wie z. B. RPM oder DEB) oder Ports (z. B.Crux Linux oder FreeBSD), verfügt, können Sie das Paket− oder Ports−System zum Installieren der MySQLTCL−Bibliothek verwenden und diesen Abschnitt überspringen.

Falls dies nicht der Fall ist oder Sie die manuelle Installation bevorzugen, zeige ich Ihnen in den nächsten Zeilen, welche Schritte ich durchgeführt habe. Diese Zeilen sollten als Richtschnur angesehen werden und nicht als Schritt−für−Schritt−Installation. Zum Beispiel in einer Linux Mandrake *Distribution* (Version 9.2) in einer bash−Shell:

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

Wenn während des "configure"−Schritts etwas falsch läuft, gibt Ihnen die Fehlerinformation Hinweise zur Bereinigung des Problems. Normalerweise liegt das Problem darin, dass das *configure*−Skript bestimmte Verzeichnisse oder Dateien nicht finden kann. In so einem Fall können Sie mit dem Skript "spielen", indem Sie ihm über Parameter mitteilen, wo die fehlenden Dateien oder Verzeichnisse zu finden sind. Ein anderes Beispiel: Unter FreeBSD 5.0 installiere ich MySQLTcl mit folgenden Optionen:

```
$ 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
```

Wie Ihnen sicher aufgefallen ist, war die Tcl−Version in diesem zweiten Beispiel 8.3; außerdem hatte die MySQLTcl−Bibliothek die Version 2.15 und die Version von MySQL war 3.23.54.

Tcl−Grundlagen

In diesem Abschnitt erläutere ich einige Tcl−Grundlagen für interessierte Leser, die nicht in Tcl programmieren können. Wenn Sie bereits ein Tcl−Programmierer sind, können Sie diesen Abschnitt überspringen.

Sie können die Beispiele aus diesem (und auch den folgenden) Abschnitten in der Tcl−Shell (tclsh) nachvollziehen.

Variablen, Befehls− und Variablen−Ersetzung

Tcl−Variablen werden mit dem Befehl *set* erzeugt. Einige Beispiele:

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

In diesen beiden Beispielen haben wir zwei Variablen namens address und zip_code erzeugt. Die in diesen Variablen gespeicherten Werte sind *Edison Avenue*, 83 und *48361*; beide Werte sind Zeichenketten. Beachten Sie, dass wir zum Erzeugen der Variablen address geschweifte Klammern benutzt haben, weil die Zeichenkette Leerzeichen enthält. Variablen−Werte können über den *set*−Befehl abgerufen werden:

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

Nehmen wir an, wir möchten den Wert der Variablen address auf dem Bildschirm ausgeben. Dies kann über den Befehl *puts* erfolgen:

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

Der Parameter *stdout* wird an den Befehl *puts* übergeben. Dieser Parameter weist den Befehl *puts* an, auf die Standardausgabe, in unserem Fall den Bildschirm, auszugeben. Der zweite an den *puts*−Befehl übergebene Parameter ist *[set address]*. Die eckigen Klammern im zweiten Parameter informieren den Tcl−Interpreter darüber, dass der Wert innerhalb der Klammern ein weiterer Tcl−Befehl ist, der vor dem *puts*−Befehl ausgeführt werden muss; dies wird als Befehlsersetzung bezeichnet. Das gleiche kann auch auf eine andere Art errreicht werden:

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

In diesem Beispiel haben wir eine Variablen−Ersetzung vorgenommen: Das Zeichen *$* vor einer Variablen veranlaßt diese Variablenersetzung.

In einem vorhergehenden Beispiel haben wir gesehen, dass durch Worttrenner getrennte Worte mittels geschweifter Klammern in einer Zeichenkette zusammengefasst werden können. Eine andere Form der Gruppierung ist durch die Benutzung des Zeichens "" möglich. Diese zwei Formen wirken jedoch etwas unterschiedlich. Ein Beispiel:

```
% 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
```

In diesem Beispiel können Sie sehen, das bei der Verwendung geschweifter Klammern keine Befehls− und Variablen−Ersetzung stattfindet; dies geschieht aber, wenn doppelte Anführungszeichen für die Gruppierung benutzt werden.

Variablen werden mit dem *unset*−Befehl gelöscht:

```
% 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
```

Tcl–Zeichenketten

Die Zeichenkette ist eine der drei grundlegenden Tcl–Datenstrukturen. Eine Zeichenkette ist eine Menge von Zeichen. Eine Zeichenkette kann mit dem *set*–Befehl erstellt werden.

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

Beide Variablen, *surname* und *number*, sind Zeichenketten. Zeichenketten können mittels des Befehls *string* manipuliert werden. Die allgemeine Syntax für den *string*–Befehl ist *string operation stringvalue otherargs*. Hier folgen einige Beispiele zur Benutzung dieses Befehls:

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

Im Gegensatz zu Java oder Pascal ist Tcl keine stark typisierte Programmiersprache. Die folgenden Beispiele zeigen dies:

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

Mit dem Befehl *expr* wurde der Wert der Variablen *number* um 24.5 erhöht. Danach wurde die Variable *number* mit dem *string*–Befehl als Zeichenkette behandelt und davon die letzten vier Zeichen angezeigt.

Es sind noch weitere Zeichenketten–Operationen möglich als im vorherigen Beispiel gezeigt.

Tcl–Listen

Tcl–Listen sind ein Spezialfall von Zeichenketten, bei dem Listenelemente durch Worttrenner getrennt sind und eine spezielle Bedeutung haben.

```
% 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
```

Es gibt einige Tcl–Befehle, die der Listenmanipulation dienen. Wieder einige Beispiele:

```
% 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
```

Das letzte Beispiel zeigt, dass Zeichenketten und Listen tatsächlich die gleiche Datenstruktur darstellen.

Tcl−Felder

Ein Feld kann als eine Liste mit einem Index angesehen werden, der auf Zeichenketten−Werten basiert. Ein Feld kann wie im folgenden Beispiel angelegt werden:

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

Feld−Werte können mittels des *set*−Befehls und Variablenersetzung abgerufen werden; wie im vorhergehenden Beispiel gezeigt wurde, wird der zeichenketten−basierte Index von Klammern begrenzt.

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

Der Befehl *array* liefert Informationen über eine Feldvariable zurück. Die nächsten Beispiele zeigen, was dieser Befehl leisten kann:

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

Datenbankverbindungen

Bevor aus einem Tcl−Skript Abfragen an eine Datenbank gerichtet werden können, muss eine Verbindung zum Datenbankserver aufgebaut werden. In diesem Abschnitt werden wir sehen, wie dies gemacht wird und wie auf Fehler reagiert wird, die beim Aufbau der Datenbankverbindung auftreten können.

Einrichten einer Datenbankverbindung

Nun folgt ein Beispiel eines Tcl−Skriptes, das eine Verbindung zu einem MySQL−Datenbankserver aufbaut:

```
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
```

Beachten Sie, dass die Zahlen in der linken Spalte und der Doppelpunkt nicht Teil des Tcl−Skriptes sind; sie dienen nur zur Markierung der Skriptzeilen. Beachten Sie auch, dass Sie in Abhängigkeit von der von Ihnen benutzten Linux−Distribution eventuell Zeile #0 ändern müssen, um den korrekten Pfad zur Tcl−Shell zu setzen.

Zeile #0 teilt der Shell mit, dass es sich bei dieser Datei um ein Tcl−Skript handelt und wo der Tcl−Interpreter zu finden ist.

Zeile #2 weist den Tcl−Interpreter an, bei der Ausführung der Befehle des Skriptes in der Bibliothek MySQLTcl zu suchen. Z. B. können wir in Zeile #10 den Befehl *mysqlconnect* sehen; wäre Zeile #2 nicht im Skript enthalten, würde der Tcl−Interpreter in Zeile #10 sich beim Befehl *mysqlconnect* mit einem "Befehl nicht gefunden"−Fehler beschweren.

In den Zeilen #5 und #6 werden Port−Nummer und Host gesetzt, zu denen das Tcl−Skript Verbindung aufnehmen will. In diesem Skript ist die Port−Nummer 3306 (der Standardport eines MySQL−Servers) und als Host wird die gleiche Maschine angesprochen, auf der das Tcl−Skript ausgeführt wird.

Die Zeilen #7 und #8 setzen den Namen des Datenbank−Benutzers und sein Passwort.

Der konkrete Verbindungsaufbau erfolgt in Zeile #10. Die Ausgabe des Befehls *mysqlconnect* wird in der Variablen gespeichert, die wir mit *mysql_handler* benannt haben. Diese Variable wird als Handle für die Datenbankverbindung benutzt. Sie wird benutzt für Abfragen an die Datenbank und auch für das Schliessen der Datenbankverbindung, wie in Zeile #12 gezeigt.

Fehlerbehandlung

Im vorhergehenden Abschnitt wurde Zeile #3 des Skriptes nicht erläutert. Das werden wir nun nachholen.

MySQLTcl−Bibliotheksbefehle können mit Fehlern enden. Wenn ein Fehler nicht abgefangen wird, bricht das Skript ab und dies sollte uns interessieren. Wir ändern das Skript aus dem vorigen Abschnitt wie folgt ab:

```
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: }
```

Wenn der Befehl *set mysql_handler [mysqlconnect −host $host...* auf einen Fehler trifft, wird dieser vom Befehl *catch* abgefangen. Der Befehl *catch* gibt 1 zurück, wenn der Befehl innerhalb der geschweiften Klammern bei der Ausführung einen Fehler verursacht, und 0, wenn kein Fehler auftritt. Die Variable *mysql_handler* speichert die Ausgabe des in geschweiften Klammern stehenden Befehls.

Wenn ein Fehler auftritt, wird eine Nachricht auf der Standardfehlerausgabe (stderr) ausgegeben, in unserem Fall auf dem Bildschirm. Es gibt eine Menge von Fehlerquellen beim Versuch, eine Datenbankverbindung aufzubauen: falsches Passwort, ungültiger Host oder Portnummer ... In diesem Fall ist es nützlich, mehr

Informationen als ein einfaches "es trat ein Fehler auf" zu erhalten.

In Zeile #3 wurde die Variable *mysqlconnect* global deklariert. Eine globale Variable kann aus jedem Teil eines Tcl−Skriptes angesprochen werden; dies steht in Beziehung zum Geltungsbereich von Variablen in Tcl; ein Thema, das nicht in diesem Artikel behandelt wird. Die Bibliothek MySQLTcl erstellt und verwaltet ein globales Feld namens *mysqlstatus*. Dieses Feld besteht aus folgenden Elementen:

Element	Bedeutung
code	Wenn kein Fehler auftritt, entspricht *mysqlstatus(code)* Null; ansonsten wird *mysqlstatus(code)* auf den Fehlercode des MySQL−Servers gesetzt. Wenn der Fehler nicht vom MySQL−Server erzeugt wurde, wird *mysqlstatus(code)* auf −1 gesetzt. Der Wert von *mysqlstatus(code)* wird nach jeder Ausführung eines MySQLTcl−Bibliotheksbefehls aktualisiert.
command	Der zuletzt ausgeführte MySQLTcl−Bibliotheksbefehl, bei dem ein Fehler auftrat, wird in *mysqlstatus(command)* gespeichert. Der Wert von *mysqlstatus(command)* wird nach jeder nicht erfolgreichen Ausführung eines MySQLTcl−Befehls aktualisiert; daher wird *mysqlstatus(command)* nach einer erfolgreichen Ausführung eines MySQLTcl−Befehles **nicht** aktualisiert.
message	Der Wert von *mysqltcl(message)* wird nach jeder fehlerhaften Ausführung eines MySQLTcl−Befehles mit einer Zeichenkette aktualisiert, die eine Fehlernachricht enthält. Wie *mysqlstatus(command)* wird mysqlstatus(message) **nicht** aktualisiert, wenn ein MySQLTcl−Befehl erfolgreich ausgeführt wurde.

Es gibt ein weiteres Element im globalen Feld *mysqlstatus*, welches keinen Bezug zur Fehlerbehandlung hat:

Element	Bedeutung
nullvalue	Zeichenkette, die benutzt wird, um bei der Anzeige von SQL−Abfrageergebnissen den Nullwert darzustellen. Als Fehlwert wird eine leere Zeichenkette genutzt; *musqlstatus(nullvalue)* kann auf eine beliebige Zeichenkette gesetzt werden.

Daher kann das vorherige Stück Code unter Benutzung des globalen Feldes *mysqlstatus* wie folgt neu geschrieben werden, damit mehr Informationen verfügbar sind, wenn ein Fehler beim Aufbau einer Datenbankverbindung auftritt:

```
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: }
```

Offensichtlich kann das globale Feld *mysqlstatus* für die Behandlung von weit mehr Fehlern genutzt werden als nur für diejenigen, welche beim Aufbau einer Datenbankverbindung auftreten können.

Grundlegende Befehle der MySQLTcl−Bibliothek

In diesem Abschnitt werden die einfachsten MySQLTcl−Bibliotheksbefehle dargestellt und die Benutzung mit einigen Beispielen gezeigt. Für eine vollständige Referenz verweise ich auf die Handbuchseite zur

MySQLTcl−Bibliothek.
Die in diesem Abschnitt behandelten Befehle sind in der folgenden Tabelle dargestellt. Parameter sind unterstrichen. Ein Parameter zwischen zwei Fragezeichen ist optional; das Zeichen | bedeutet "oder".

Befehl	kurze Beschreibung
mysqlconnect ?option value ...?	verbindet zu einer Datenbank; ein Verbindungs−Handle wird zurückgegeben, welches von anderen MySQLTcl−Befehlen benutzt werden muss
mysqluse handle dbname	verbindet ein MySQL−Handle mit der angegebenen Datenbank
mysqsel handle sql_statement ?−list | −flatlist?	schickt einen SQL−Select−Befehl an die Datenbank
mysqlexec handle sql_statement	schickt einen SQL−Befehl an die Datenbank
mysqlclose handle	schliesst eine Datenbankverbindung

mysqlconnect

Dieser Befehl wurde bereits im Abschnitt "Datenbankverbindung" diskutiert. Er aktzeptiert einen zusätzlichen Parameter −db, der noch nicht gezeigt wurde. Mit dem Parameter −db wird die Datenbank festgelegt, die bei weiteren SQL−Befehlen benutzt wird. Ein Beispiel:

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

Die "Ziel"−Datenbank für weitere MySQLTcl−Befehle, die das *mysql_handler*−Datenbank−Handle nutzen, ist diejenige mit dem Namen *jsmith_database*.

(Beachten Sie, dass die Zeichen \\ nicht Bestandteil der Befehle sind; sie bedeuten nur, dass der Befehl auf der folgenden Zeile fortgesetzt wird.)

mysqluse

Dieser Befehl erlaubt es, die mit dem MySQL−Handle assoziierte Datenbank mit derjenigen zu tauschen, die als erster Parameter an diesen Befehl übergeben wird.

mysqlsel

Dieser Befehl sendet eine SQL−select−Anweisung an die mit dem MySQL−Handle assoziierte Datenbank. Wenn der Parameter sql_statement keine SQL−select−Anweisung enthält, gibt es einen Fehler.

Es gibt einen dritten optionalen Parameter, der −list oder −flat_list sein kann. Wir zeigen an einem Beispiel, wie dieser Parameter die Ausgabe des Befehls beeinflusst. Nehmen wir an, dass es in der mit dem MySQL−Handle assoziierten Datenbank eine Tabelle namens *people* mit folgendem Inhalt gibt:

id	first_name	last_ name	phone

26	Karl	Bauer	8245
47	James	Brooks	1093
61	Roberto	Castro Portela	6507

Wir benutzen den Befehl *mysqlsel*, um eine SQL−select−Anweisung an die Datenbank zu schicken:

```
% 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}
```

Im ersten Beispiel (*Parameter −list*) gibt der Befehl eine Liste zurück, deren Elemente wiederum Listen sind. Im zweiten Beispiel (*Parameter −flatlist*) gibt die Anweisung eine einzelne Liste zurück, in der alle Elemente miteinander verkettet wurden.

Was geschieht, wenn der *mysqlsel*−Befehl keinen dritten Parameter erhält? In diesem Fall ist die Ausgabe des *mysqlsel*−Befehls die Anzahl der von der Abfrage betroffenen Zeilen:

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

mysqlexec

Die *mysqlexec*−Anweisung sendet eine SQL−Anweisung an die mit dem MySQL−Handle assoziierte Datenbank. Wenn der Parameter *sql_statement* eine SQL−select−Anweisung enthält, gibt es keinen Fehler, es wird aber auch nichts ausgeführt.

Nehmen wir das Beispiel aus dem vorherigen Unterabschnitt:

```
% 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}
```

Natürlich können andere SQL−Anweisungen als delete oder insert mit diesem Befehl an die Datenbank geschickt werden. Z. B. kann eine Zeile aktualisiert werden:

```
% 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}
```

Die Anweisung *mysqlexec* gibt bei erfolgreicher Ausführung die Anzahl der von dem SQL−Befehl betroffenen Zeilen zurück.

mysqlclose

Wie bereits gesehen, schliesst der Befehl *mysqlclose* eine Datenbankverbindung. Der Parameter für diese Anweisung ist das MySQL−Handle der Datenbankverbindung, die wir schliessen möchten.

Weitere Befehle der MySQLTcl−Bibliothek

Die MySQLTcl−Bibliothek verfügt über mehr als die in diesem Abschnitt gezeigten fünf Befehle. Diese Anweisungen erlauben den Abruf von Informationen über die Datenbank, die Maskierung von Zeichenketten, um sie für Abfragen aufzubereiten, verschachtelte Abfragen ... Eine gute Referenz ist die MySQLTcl−eigene Handbuchseite, die Teil der Installation der MySQLTcl−Bibliothek ist.

Referenzen

[1] eine etwas skeptische Betrachtung über John K. Ousterhout und Tcl:
http://www.softpanorama.org/People/Ousterhout/index.shtml

[2] ein Tutorial zu regulären Ausdrücken in Tcl:
http://www.mapfree.com/sbf/tcl/book/select/Html/7.html

TclTutor ist eine freie und interaktive Anwendung zum Erlernen von Tcl:
http://www.msen.com/~clif/TclTutor.html

MySQL−Dokumentation in verschiedenen Formaten (HTML, PDF...):
http://www.mysql.com/documentation/index.html

Webpages maintained by the LinuxFocus Editor team

http://www.LinuxFocus.org

Translation information:
en −−> −− : Diego Alberto Arias Prad <dariapra(at)yahoo.es>
en −−> de: Hermann−Josef Beckers <:beckerst(at)lst−oneline.de>

2005−01−11, generated by lfparser_pdf version 2.51