

KISSed by TCL

[Martin Scherbaum](#)

Kennen Sie das UNIX-Prinzip KISS? - Keep It Small and Simple (oder Keep It Small and Stupid ;-)). Jede Aufgabe wird von einem kleinen Programmchen übernommen, das nur diese Aufgabe, diese aber dafür schnell, gut und richtig kann. Dieses Prinzip macht einen großen Teil der Flexibilität von UNIX aus, wo die Applikationen "nur noch" die kleinen Programme über sog. Pipes oder Sockets verbinden und darauf große Systeme aufbauen. Linux ist eben aus diesen kleinen Bausteinen und ihren Verknüpfungen aufgebaut. Dies steht im Gegensatz zur klassischen "monolithischen" Applikationsentwicklung, die alle Aufgabe in ein einziges großes Programm packt, das dann schwerfällig und schwer zu warten wird.

Doch irgendwann geht dem KISS-Prinzip ein bißchen die Luft aus. Dann nämlich, wenn z.B. eine einheitliche Benutzeroberfläche gefordert ist. Oder wenn beim Zusammenschalten der kleinen Programmchen ein bißchen mehr gemacht werden soll, als nur Daten von einem ins andere zu schieben. Diese Lücke kann zu einen gewissen Teil die Shell-Programmierung schließen (z.B. mit bash, ksh, csh). Aber auch da ist - ob der beschränkten programmiertechnischen Möglichkeiten der Shell - ziemlich schnell das Ende der Möglichkeiten erreicht.

Um diese Beschränkungen zu umgehen, erfand Ende der achtziger Jahre des letzten Jahrhunderts der Amerikaner Dr. John Ousterhout im Rahmen seiner Universitätskurse die Skriptsprache TCL. TCL wird üblicherweise wie engl. "tickle" (kitzeln) ausgesprochen. Te Ce El ist aber auch erlaubt ;-)

TCL ist eine sogenannte Glueing Language, d.h. das ursprüngliche Ziel war es, mit TCL die o.g. UNIX-Programmchen besser miteinander zu verbinden und zusätzlich weitergehende Programmierung mit den Daten, die diese Programmchen erzeugen, realisieren zu können.

Relativ schnell genügte aber das pure TCL nicht mehr, da auch UNIX-Anwender gerne eine graphische Benutzeroberfläche für Programme nutzen. In der Konsequenz wurde das Kern-TCL erweitert um das graphische Toolkit für X11, genannt Tk. Deshalb liest man heute auch meist von TCL/Tk.

Im Laufe der Zeit wurde TCL offener und das Konzept der TCL-Extensions wurde ausgebaut. Ein wesentlicher Vorteil von TCL war von Anfang an die Existenz wohldefinierter Schnittstellen zum TCL-Interpreter (wir erinnern uns: TCL ist eine interpretierte Skript-Sprache), insbesondere eine einfach zu handhabende C-Schnittstelle, die eine einfache und schnelle Integration eigenen C-Codes in den TCL-Interpreter ermöglichte.

So wundert es nicht, daß mittlerweile TCL-Extensions für fast alle wesentlichen Anwendungsbereiche unter UNIX verfügbar sind. Das reicht von einfachen Befehlserweiterungen für den Standard-TCL-Interpreter bis hin zu Datenbankanbindungen (zB PostgreSQL, MySQL, ...), über graphische Erweiterungen (Tix, TkGraph, Gipsy) bis hin zu Erweiterungen, die TCL objektorientiert werden lassen, inclusive einer Klassenbibliothek für GUI-Elemente (iTCL, iTk, iWidgets). Auch lassen sich z.B. mit expect andere (UNIX-)Programme fernsteuern, die sonst nur interaktiv zu bedienen sind (zB passwd, login).

Der zentrale Bestandteil von TCL ist der Interpreter, der Eingaben in der TCL-Shell (tclsh) bzw. der Windowing Shell für Tk (wish) parst, in Bytecode übersetzt und dann ausführt. Dies kann interaktiv in der Shell bzw. automatisch über sog. Skripte geschehen. Skripte sind nichts anderes als TCL-Programme. Im Gegensatz zu z.B. C, als Compiler-Sprache, entfällt bei TCL der Schritt des Kompilierens. Das Programm wird quasi im Quelltext gestartet. Das macht TCL insbesondere fürs Rapid Prototyping interessant, wo es weniger auf vollständige Programme als auf schnelle betrachtbare

Prototypen ankommt. Dies gilt besonders dann, wenn man die Idee für eine neue Applikation als Benutzeroberfläche mal kurz zeigen will, anstatt sie nur auf dem Reißbrett zu planen. Hier ein kurzes Beispiel, wie man mit einfachen Mitteln eine kleine Applikation erzeugen kann, die auf Knopfdruck "Hello World" auf der Konsole ausgibt:

```
button .b -text "Drück mich" -command { puts stderr "Hello World" }
pack .b -expand yes -fill both
```

Um es einzugeben, startet man in aus der Kommandozeile die TCL-Windowing-Shell `wish`:

```
user@meinrechner:~> wish
```

und gibt den oben gezeigten Code interaktiv ein. Oder man schreibt den Code in eine Datei (hier: `mein_erstes_tcl_programm.tcl`) und ruft diese von der Kommandozeile aus auf:

```
user@meinrechner:~> wish mein_erstes_tcl_programm.tcl
```

Und wenn es noch professioneller sein soll, schreibt man in die ersten Zeilen der TCL-Datei folgenden Kopf (er muß **wirklich** mit dem 1. Zeichen der Datei anfangen!):

```
#!/bin/bash
# \
exec wish $0 ${1+"$@"}
```

und macht die Datei ausführbar. Dann kann man sie direkt von der Kommandozeile aus aufrufen:

```
user@meinrechner:~> ./mein_erstes_tcl_programm.tcl
```

Sieht doch schon richtig wie ein X11-Programm aus, wenn man das o.g. TCL-Programm laufen läßt, oder?

Zur Zeit gibt es die TCL-Shell und die meisten Erweiterungen für alle gängigen UNIX-Systeme, Windows und Mac OS. Die graphischen Applikationen nutzen jeweils den Native Look des Betriebssystems und unterstützen fast überall fast alle TCL-Befehle, dh. TCL-Code ist weitgehend Betriebssystem unabhängig. Nur wenn einem OS die entsprechenden Fähigkeiten fehlen, ist der TCL-Code nicht vollständig portabel.

Die Verwendung einer Programmiersprache oder die Zuneigung dazu ist eine sehr persönliche (wenn nicht sogar religiöse?) Entscheidung.

Nichtsdestoweniger gibt es eher objektive Gründe, warum z.B. TCL anderen Sprachen vorzuziehen ist, meist abhängig von der geplanten Anwendungssituation.

Was kann TCL nicht so gut?

- String-Umwandlungen: das beschäftigt den Interpreter, da in TCL erst einmal alles ein String ist und nur wenn nötig z.B. als Zahl interpretiert wird. High-Performance-Anwendungen sollte man also eher als TCL-C-Erweiterung schreiben, um alle übrigen TCL-Vorteile nutzen zu können, aber bei der Performance keine Kompromisse einzugehen.
- Größe des Programms: da TCL immer seinen Interpreter braucht, muß man ihn fairerweise zum eigentlich winzigen TCL-Skript dazurechnen. TCL-Applikationen sind also dann klein, wenn man viel TCL-Code hat. Dann fällt nämlich der Interpreter nicht mehr so ins Gewicht. Da auf den meisten UNIX- bzw. Linux-Systemen die TCL-Shell mittlerweile ins Grundsystem verankert ist, hat dieses Argument nur noch wenig Relevanz.

Diese "Nachteile" sind aber für sehr viele Programme, gerade für interaktive, so gut wie gar nicht relevant. Also: lieber die Vorteile von TCL:

Was kann man mit TCL gut?

- TCL ist leicht zu lernen. Zum einen gibts gute Literatur (s.u.), zum anderen ist die Anzahl der Befehle überschaubar und die Syntax einfach: `befehlsword argument1 argument2 argument3 argumentn` Alle TCL-Befehle und Sprachkonstruktionen folgen diesem Muster.
- mit TCL sieht man schnell Erfolge. Anwendungsgebiete: Rapid Prototyping, die kleine Applikation für Zwischendurch
- TCL-Code ist kompakt: man braucht sich als Programmierer nicht mehr mit terminierenden Null-Zeichen am Ende von Strings beschäftigen, TCL alloziert den Speicher von alleine; assoziative Arrays entsprechen mehr menschlichem Problemlösungsdenken als Arrays, über die ein Zahlenindex läuft; sehr flexibles Listenhandling; der Programmierer kann sich auf sein Problem konzentrieren
- TCL hat ein kraftvolles Interface zu anderen Programmen (da kommt es ja her ;-)) im Betriebssystem.
- mit den TCL-Extensions ist so gut wie jeder Anwendungsbereich zugänglich (u.a. Datenbankzugriff, TCL-CGI-Skripte, GUI-Programmierung, Netzwerkzugriff)
- TCL, seine Programmierschnittstelle und auch die meisten Extensions sind gut dokumentiert. Darauf hat Erfinder Dr. John Ousterhout viel Wert gelegt.
- TCL-Code ist gut strukturierbar und somit gut lesbar (im Gegensatz zu "write-only"-Code anderer Skriptsprachen ;-)))
- TCL ist in einigen gängigen Programmierumgebungen verwendbar bzw. integriert, zB GNU Emacs, ActiveState Komodo, Mid Innovator

Links zum Thema TCL

- Scriptics, früherer Arbeitgeber von Dr. Ousterhout, <http://www.scriptics.com>
- ActiveState, aktuelle Homepage der TCL-Community, <http://www.activestate.com/Solutions/Programmer/Tcl.plex>
- Offizielle (aber schwer erreichbare) TCL-Homepage, <http://www.tcltk.com>

Bücher über TCL/Tk und Extensions

- Wenn das die TCL-Bibel ist... ("das blau-gelbe Buch"), [Tcl und Tk, John Ousterhout](#)
- ... dann ist das das Neue TCL-Testament, [Practical Programming in Tcl and Tk, Brent B. Welch](#)
- Weiterführendes..., [Effektiv Tcl/Tk programmieren, Harrison/McLennan](#)
- Für Sysadmins und Nicht-Interaktive, [Exploring Expect, Don Libes](#)
- TCL-Extensions, [Tcl/Tk Tools, Mark Harrison](#)

Über den Autor:

[Martin Scherbaum](#) ist Computerlinguist und war bei SuSE schon sehr früh dabei (Personalnummer 14!), programmiertechnische Probleme löst er in TCL oder der bash... 