Tcl
Tool Command Language

Michael Grünewald

Interpreter Source Code Analysis 2010

syntax

- basic syntax

  command $argument_1$ $argument_2$ $argument_3$ …

- statements delimited by
  - line breaks
  - semicolons

- comments with #
  - must be at beginning of statement

grouping

- commands and arguments separated by spaces
- can be grouped:
  - using double quotes

    ```
    puts "hello world"
    ```

  - using braces

    ```
    puts {hello world}
    ```

- not variable substitution when using braces

variables

- can have any name
  - including spaces
- command: `set`
  - (optionally) sets a variable to a value
  - returns the value of a variable
- command: `incr`
  - increases the value of a variable
  - can be used with not existing variables (Tcl 8.5)

substitution

- variable substitution
  - using $
  - but not when grouped using braces
  - replaces *$name* with variable contents
- command substitution
  - using […]
  - replaces brackets with command result

lists

- commands for working with lists
- some change lists, some return new lists

- `list`
- `lappend`
- `index`
- `linsert`
- `llength`
- `lrange`
- `lrepeat`
- `lreverse`
- `lsearch`
- `lset`
- `lsort`
- …

arrays

- called *hashed arrays* or *dictionaries* in other languages

- standard `set` and `get` commands for entries
- `array` command:
  - `array names`
  - `array size`
  - `array get`
  - `array set`

math

- everything is a command: no math without commands
- command: `expr`
  - calculates expressions written in common infix notation
  - can use functions (like `sin`, `cos`, `log`, …)
  - functions are commands in the `::tcl::mathfunc` namespace

control structures

- `if` c_1 g_1 `elseif` c_2 g_2 … `else` g_e
- `switch` v `{`c_1 g_1 c_2 g_2 … `}`
- `while` c g
- `foreach` v l g
  - multiple loop variables possible
- `for` g_i c g_f g_b
  - `break`
  - `continue`
- `catch` g v
  - `error` m
- `return`

procedures

- `proc` *name args body*

- arguments list with names
- optional arguments with default values
- last argument may be called *args*

scopes and namespaces

Hasso Plattner Institut

- default:
  - variables outside procedures: global scope
  - variables in procedures: local scope
- `global` command
  - use global scope for variable name in procedure
- `upvar` command
  - for using variables by reference
  - `upvar #0 foo foo` equals `global foo`
  - related `uplevel` command
- qualified names
  - prefix with `::` for global scope

usage: Expect

- a Tcl library
- developed by NIST to automate and test applications

- `spawn`
  - executes a subprocess

- `expect`
  - waits for an output of the subprocess

- `send`
  - sends input to the subprocess

usage: Expect example

HPI Hasso Plattner Institut

```
#!/usr/local/bin/expect
spawn rlogin [lindex $argv 0]
expect -re "($|#|%) "
send "cd [pwd]\r"
expect -re "($|#|%) "
send "setenv DISPLAY $env(DISPLAY)\r"
interact
```

usage: Tk

- user interface toolkit for Tcl
- often used in other programming environments

- new ttk extension for native platform look

usage: Tk (2)

```
package require Tk

wm title . "Feet to Meters"
grid [ttk::frame .c -padding "3 3 12 12"] -column 0 -row 0 -sticky nwes
grid columnconfigure . 0 -weight 1; grid rowconfigure . 0 -weight 1

grid [ttk::entry .c.feet -width 7 -textvariable feet] -column 2 -row 1 -sticky we
grid [ttk::label .c.meters -textvariable meters] -column 2 -row 2 -sticky we
grid [ttk::button .c.calc -text "Calc" -command calculate] -column 3 -row 3 -sticky w

grid [ttk::label .c.flbl -text "feet"] -column 3 -row 1 -sticky w
grid [ttk::label .c.islbl -text "is equivalent to"] -column 1 -row 2 -sticky e
grid [ttk::label .c.mlbl -text "meters"] -column 3 -row 2 -sticky w

foreach w [winfo children .c] {grid configure $w -padx 5 -pady 5}
focus .c.feet
bind . <Return> {calculate}

proc calculate {} {
    if {[catch {
        set ::meters [expr {round($::feet*0.3048*10000.0)/10000.0}]
    }]!=0} {
        set ::meters ""
    }
}
```