An Introduction to the Tcl Programming Language

Graham Phillips

Computer Science Department

University of Southern California

Los Angeles

Purpose

- Tcl is a scripting language for combining or “glueing” applications together.
- The Tcl interpreter may be embedded into applications.
- A toolkit for Tcl called Tk provides routines for developing GUI applications for the X Windows System.

Language overview

- Types: string only
- Control flow: assignment, if, while, for, foreach, switch, break, continue, procedures, exception handling.
- Built-in functions: string manipulation, file name handling, file access, math functions, environment variable manipulation, inter-application communication, process creation.

Authorities

The defining document for the language is “Tcl and the Tk Toolkit” by John K. Ousterhout, Addison-Wesley, 1994, ISBN 0-201-63337-X.

The WWW page called “The Tcl/Tk Project At Sun Microsystems Laboratories” may be found at

`http://www.smli.com/research/tcl/`

Syntax

Each *command* consists of one or more *words* where the words are separated by white-space:

<command> <arg1> <arg2> ... <argn>.

A Tcl *script* consists of one or more commands. Commands are separated by newlines or semi-colons.

```
% set a 2
```

```
% set a 24;  set b 15
```

Tcl provides three forms of substitution:

- variable substitution
- command substitution
- backslash substitution.

Variable substitution

A dollar sign $ causes variable substitution. (Similar to C-shell, except that Tcl does not use the = character).

% set kgrams 20

% expr $kgrams * 2.2046

44.092

Syntax cont.

Command substitution

Command substitution is performed using brackets []. The command is evaluated and everything, including the brackets, is replaced with the result of the command.

```
% set kilograms 20

% set pounds [expr $kilograms * 2.2046]

44.092
```

Backslash substitution

The purpose of backslash substitution is to insert special characters into the commands.

```
% set message Eggs:\ \$2.18 / dozen\nGasoline:\
\$1.49 / gallon

Eggs: $2.18 / dozen

Gasoline: $1.49 / gallon
```

"Tcl's variable substituting command handles only the most common substitutions. There exist scenarios where none of the preceding forms of substitution achieve the desired effect. More complicated situations can be handled with a sequence of commands." [Ousterhoud].

Syntax cont.

Quoting with double quotes

Double quotes disable word and command separators.

```
% set mwg "Eggs: \$2.18 / dozen\nGasoline: \$1.49
/ gallon
```

```
Eggs: $2.18 / dozen
```

```
Gasoline: $1.49 / gallon
```

Substitution occurs as usual inside double quotes (similar to C-shell).

```
% set a 2.1
```

```
% set msg "a is $a, the square of a is [expr
$a*$a]"
```

```
a is 2.1, the square of a is 4.41
```

Syntax cont.

Quoting with braces

Grouping words with braces { } prevents substitution.

```
% set msg {Eggs: $2.18 / dozen\nGasoline: $1.49 /
gallon}

Eggs: $2.18 / dozen\nGasoline: $1.49 / gallon
```

Comments

A comment line is started by the # character and extends to the end of the line. The # character must be in the position of the first word of a command word.

```
% set b 101   ;#      is legal but


% set b 100 #         is not legal.

wrong # args: should be "set varName ?newvalue?"
```

Variables

- Tcl maintains a set of variables which are (name, value) pairs.
- Variables are not declared.
- Variable names can be any length, are case sensitive and need not start with a letter.
- Variable values are strings or sets of strings separated by spaces. Variables are assigned using the `set` command and cleared using the `unset` command.
- Variables are accessed using the $ character.
- Variables may be local or global.

Associative arrays

Array elements are indexed by arbitrary strings.

```
% set earnings(January) 8799
```

Only one dimensional arrays are implemented. However more that two dimensional arrays may be simulated by concatenating indices

```
% set matrix(1,1)  140

% set matrix(1,2) 218
```

The following script however is unable to access matrix(1,2).

```
% expr $matrix(1, 2)

can't read "matrix(1, 2)": no such variable
```

Variables cont.

Predefined variables

The following variables *may* be predefined. Tcl does not require that all applications support the following variables, but most applications do.

`$argv0` is the name of the script file,

`$argv` is the list of command line arguments,

`$argc` is the number of command line arguments and

`$env` is an array variable whose elements are all of the processes's environment variables.

Expressions and Operators

Precedence of Operators

- The operators are similar to the ANSI C operators and have the same precedence.
- The % (mod) operator has a more consistent behavior in Tcl than C. In Tcl the remainder is always greater than or equal to zero.
- Short-circuit evaluation is used.
- The operators <,>, <=, >=. ==, != may be used with strings as well as real numbers and integers.

Literals

Numbers beginning with 0x are treated as hexadecimal numbers, otherwise those beginning with 0 are treated as octals. All other numbers are treated as decimal numbers.

355 decimal

0517 octal

0x14f hex

092 illegal

Expressions and operators cont.

Expression evaluation

A Tcl command returns a string by default. If evaluation is required then it must be explicitly requested.

```
% set x 4

4

% set y 4+10

4+10

% set y x+10

x+10

% set y [expr $x + 10]

14
```

Order of evaluation

Tcl commands are evaluated in two steps.

- interpreter parses all commands into words while performing substitutions
- executes the command returning a string

Expressions cont.

Conversions

When two operands are combined using a binary operator, Tcl uses arithmetic evaluation unless one or both of the operands cannot be parsed as a number. The result is converted back to a string.

```
% set x 0

% set x 00

% if {$x == $y} {set z 1}

1
```

String comparison can be forced.

```
% string compare $x $y

-1
```

The global variable `tcl_precision` controls the precision of the conversion. If `tcl_precision = 17` then no information is lost during the conversion.

Built-in math functions

Tcl supports a number of mathematical functions such as sin and exp. In most cases the functions have the same behavior as the ANSI C library procedures.

Lists

Creating lists

```
% concat {a b c} {d e} f {g h i}
a b c d e f g h i

% set z "$x $y"
a b c d e

% list {a b c} {d e} f {g h i}
{a b c} {d e} f {g h i}
```

Lists may have any depth.

```
% lindex {a b {c d e} f} 2
c d e
```

Lists cont.

Indexing lists: lindex

```
% lindex {John Anne Mary Jim} 1
Anne
```

Searching lists: lsearch

```
% set x {John Anne Mary Jim}
% lsearch $x Mary
2
% lsearch -glob $x A*
1
```

Sorting lists: lsort

The sorting command uses control switches such as `-decreasing` or `-integer`. User-defined sorting functions may be used as well.

```
% lsort -decreasing {John Anne Mary Jim}
Mary John Jim Anne
```

Lists cont.

Splitting and joining lists

```
% set y /usr/include/sys/types.h

% split $y /

{} usr include sys types.h

% join { {} usr include sys types.h} /

/usr/include/sys/types.h
```

File I/O

Dealing with filenames

Filename expansion is performed using the `glob` command which returns a list of the names of all files that match any of the pattern arguments. The special characters are ? * [] {} and \ (compare with C-shell's special characters ? * [] {} ~).

```
% glob *.c *.h

main.c hash.c hash.h
```

The `file` command is a general purpose command with many options for manipulating file names.

```
% file dirname /a/b/c

/a/b

% file extension src/main.c

.c

% file rootname src/main.c

src/main

% file tail /a/b/c

c
```

Statements and flow control

Tcl's control flow commands are similar to the control flow statements in C and C-shell.

IF statement

The `if` command evaluates an expression, tests its result and conditionally executes a script based on the result. It receives two arguments, the first is an expression while the second is a script. The expression is typically placed in braces to *delay substitution* until the command is executed. The following example sets a variable `x` to zero, if it is negative.

```
% if { $x < 0} {
   set x 0
}
```

An alternative form of the `if` statement contains `elseif` clauses.

```
% if {$x < 0} {
  ...
} elseif {$x == 0}  {
} elseif {$x == 1} {
   ...
}
```

The while statement

The `while` loop takes two arguments: an expression and a script. The `while` statement returns an empty string as the result. Note that the brace { must be on the same line as the preceding word or else the newline is treated as the command separator.

```
% set i 1

% while {$i <= 4} {

puts “$i squared is [expr $i*$i]” }

incr i 1

}

1 squared is 1

2 squared is 4

3 squared is 9

4 squared is 16
```

The for statement

The first argument of the `for` loop is an initialization script, the second is a termination script, the third is a condition and the fourth is the body of the loop. The first expression is executed as a Tcl script. If the condition evaluates to non-zero, `for` executes the body followed by the reinitialization script.

```
% for {set i 1} {$i <= 4} {incr i 1} {
puts "$i squared is [expr $i*$i]" }
1 squared is 1
2 squared is 4
3 squared is 9
4 squared is 16
```

The foreach statement

The `foreach` statement executes a Tcl script for each element in a list.

```
% foreach i {1 2 3 4} {
puts "$i squared is [expr $i*$i]" }
```

The switch statement

The `switch` statement provides a concise notation for multiple `if` statements. When a match is found, it executes the corresponding script and returns the value of that script as its value. Patterns need not be exhaustive.

```
% switch $x {a {incr t1} b {incr t2} c {incr t3}}

% switch $x a {incr t1} b {incr t2} c {incr t3}
```

The first scripts may be written across several lines as follows:

```
% switch $x {
      a {incr t1}
      b {incr t2}
      c {incr t3}}
```

The behavior of the matching can be modified with `-exact, -glob` and `-regexp`.

```
% foreach i $x {
   switch -regexp $i {
     a           { ... }
     ^[0-9]+$    {incr num_digits }
     default     { ... }
}
```

Statements and flow control cont.

The eval command

The `eval` command accepts any number of arguments. The `eval` command concatenates the arguments with separator spaces and then *executes* the result as a script. The eval command has two primary purposes.

- To *save commands* as variables that may be executed at a later stage

```
% set cmd "set a 0"

...

% eval $cmd
```

- To *force another level of parsing*. Compare the following two code fragments

```
% set vars {a b c d}

    foreach i $vars {

        unset $i

}

% set vars {a b c d}

    eval unset $vars
```

Procedures

Tcl procedures are implemented using the `proc` keyword. The first argument to `proc` is the procedure name, the second is a list of arguments and the third is the procedure body.

- Procedures are not declared. Once the `proc` command completes, a new command exists which may be invoked like any other Tcl command.
- Within the procedure, the procedure arguments are local variables while all other variables are global variables.
- Global variables must be referenced using the `global` command which sets variable bindings (e.g. `global x y`). Once invoked, the bindings remain effective until the procedure ends.
- Procedure arguments may be passed by value or by reference.
- Arguments may have default values.
- A variable number of arguments is permitted.
- Tcl does not provide a form of variable equivalent to *static* variable in C and therefore global variables must be used for purposes like this.

An implementation of the factorial function.

```
% proc fac x {
  if $x <= 1 {return 1}
  expr $x * [fac [expr $x -1 ]]
  }
```

Procedures cont.

Call by reference is implemented using the `upvar` command. The following example prints the contents of an array.

```
% proc parray name {
    upvar $name a
    foreach el [lsort [array names a]] {
        puts "$el = $a($el)"
    }
}
% set info(age) 37
% set info(position) "Vice President"
% parray info
age = 37
position = Vice President
```

Tcl procedures may be implemented in C for higher performance and these C routines may be invoked in the same way as the original Tcl scripts.

Features not covered

Exception handling

Exceptions may be trapped using the Tcl `catch` command, allowing unforeseen asynchronous events to be processed.

Concurrency

Tcl does not support multi-threading.

String manipulation

Tcl supports two mechanisms for matching strings. The first is called "globbing" that is based on the UNIX mechanism for file name expansion. The second uses regular expressions.

Processes

Tcl provides several commands to manipulate processes. New processes can be created, environment variables can be read and modified, and processes can be terminated.

Comments

A WWW page entitled “Comparisons of Tcl with other systems may be found at `http://icemcfd.com/tcl/comparison.html`

Below are some extracts from this page.

“Tcl was not designed to be a serious programming language. It was designed to be a scripting language. It lacks arrays, it lacks structure from which you can make linked lists” [Richard Stallman - GNU Project]

“After a few thousand lines of code, I spent most of my time getting my program to hold together and trying to map and unmap my data structures to strings. In the end I switched to STk.” [Robert DeLine - CMU]

“I didn’t design Tcl for building huge programs with 10’s or 100’s of thousands of lines of Tcl ... I don’t claim that Tcl is without flaws. Some of the flaws, like the lack of a compiler and the lack of module support, will get fixed over time” [John Ousterhoud - Sun Microsystems]

“Tcl/Tk is lexically too complicated. The reason is, the application into which the scripting language will be embedded doubtless have its own notational conventions. To have “all the good characters already taken” by the scripting language means that lots of quoting needs to go on.”[Wayne Throop]

“When it comes to writing serious applications requiring complex typing (eg. a project management system for concurrent engineering, or a product documentation and information system for mechanical engineering), Tcl/Tk is not well suited primarily because it is lacking a proper type system.” [Juergen Wagner].