An Introduction To Tcl Scripting

John Ousterhout
Sun Microsystems Laboratories
`john.ousterhout@eng.sun.com`

Tcl/Tk Tutorial, Part II

Language Overview

Two parts to learning Tcl:

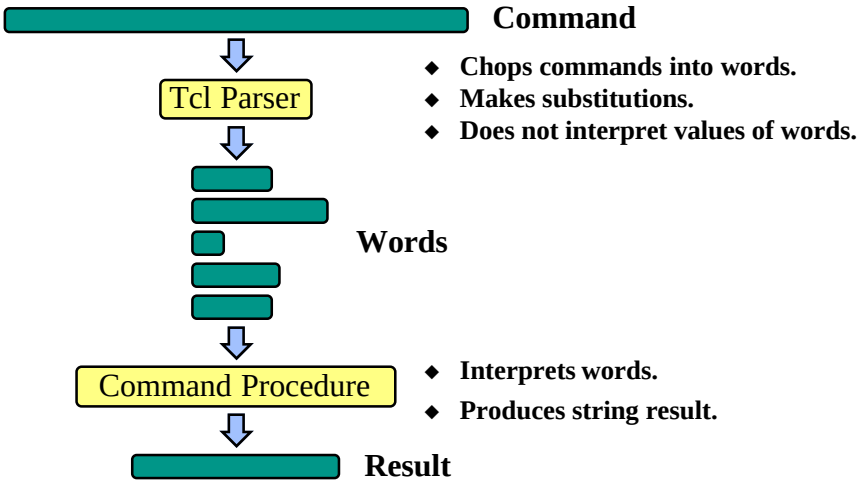
1. Syntax and substitution rules:
- Substitutions simple, but may be confusing at first.

2. Built-in commands:
- Can learn individually as needed.
- Control structures are commands, not language syntax.

Basics

- **Tcl script =**
  - Sequence of commands.
  - Commands separated by newlines, semi-colons.
- **Tcl command =**
  - One or more words separated by white space.
  - First word is command name, others are arguments.
  - Returns string result.
- **Examples:**
  - `set a 22; set b 33`
  - `set a 22`
    `set b 33`

Division Of Responsibility



- **Chops commands into words.**
- **Makes substitutions.**
- **Does not interpret values of words.**

- **Interprets words.**
- **Produces string result.**

Arguments

- **Parser assigns no meaning to arguments (quoting by default, evaluation is special):**

  C: `x = 4; y = x+10`
  y is 14

  Tcl: `set x 4; set y x+10`
  y is "x+10"

- **Different commands assign different meanings to their arguments:**

```
set a 122
expr 24/3.2
eval "set a 122"
button .b -text Hello -fg red
string length Abracadabra
```


Variable Substitution

- **Syntax: $***varName*
- **Variable name is letters, digits, underscores.**
- **May occur anywhere in a word.**

Sample command	Result
`set b 66`	`66`
`set a b`	`b`
`set a $b`	`66`
`set a $b+$b+$b`	`66+66+66`
`set a $b.3`	`66.3`
`set a $b4`	*no such variable*

Command Substitution

- **Syntax: [*script*]**
- **Evaluate script, substitute result.**
- **May occur anywhere within a word.**

Sample command	Result
`set b 8`	`8`
`set a [expr $b+2]`	`10`
`set a "b-3 is [expr $b-3]"`	`b-3 is 5`

Controlling Word Structure

- **Words break at white space and semi-colons, except:**
  - Double-quotes prevent breaks:
    `set a "x is $x; y is $y"`
  - Curly braces prevent breaks and substitutions:
    `set a {[expr $b*$c]}`
  - Backslashes quote special characters:
    `set a word\ with\ \$\ and\ space`
- **Substitutions don't change word structure:**

```
set a "two words"
set b $a
```

Expressions

- C-like (int and double), extra support for string operations.
- Command, variable substitution occurs *within* expressions.
- Used in `expr`, other commands.

Sample command	Result
`set b 5`	`5`
`expr ($b*4) - 3`	`17`
`expr $b <= 2`	`0`
`expr $a * cos(2*$b)`	`-5.03443`
`expr {$b * [fac 4]}`	`120`
`set a Bill`	`Bill`
`expr {$a < "Anne"}`	`0`

Lists

- Zero or more elements separated by white space:
  `red green blue`
- Braces and backslashes for grouping:
  ```
  a b {c d e} f
  one\ word two three
  ```
- List-related commands:

`concat`	`lindex`	`llength`	`lsearch`
`foreach`	`linsert`	`lrange`	`lsort`
`lappend`	`list`	`lreplace`	

- Examples:
  `lindex {a b {c d e} f} 2` ➪ `c d e`
  `lsort {red green blue}` ➪ `blue green red`

Control Structures

- C-like appearance.
- Just commands that take Tcl scripts as arguments.
- Example: list reversal.

```
set b ""
set i [expr [llength $a] - 1]
while {$i >= 0} {
    lappend b [lindex $a $i]
    incr i -1
}
```

- Commands:

```
if        for     switch   break
foreach   while   eval     continue
```


Procedures

- `proc` command defines a procedure:

```
proc sub1 x {expr $x-1}
```

name — list of argument names — body

- Procedures behave just like built-in commands:

```
sub1 3    ⇨ 2
```

- Arguments can have defaults:

```
proc decr {x {y 1}} {
    expr $x-$y
}
```

- Scoping: local and global variables.

Procedures, cont'd

- **Variable-length argument lists:**

```
proc sum args {
    set s 0
    foreach i $args {
        incr s $i
    }
    return $s
}

sum 1 2 3 4 5
⇨ 15
sum
⇨ 0
```


Errors

- **Errors normally abort commands in progress, application displays error message:**

```
set n 0
foreach i {1 2 3 4 5} {
    set n [expr {$n + i*i}]
}
⇨ syntax error in expression "$n + i*i"
```

- **Global variable `errorInfo` provides stack trace:**

```
set errorInfo
⇨ syntax error in expression "$n + i*i"
    while executing
"expr {$n + i*i}"
    invoked from within
"set n [expr {$n + i*i}]..."
    ("foreach" body line 2)
    ...
```

Advanced Error Handling

- Can intercept errors:

```
catch {expr {2 +}} msg
⇨ 1
set msg
⇨ syntax error in expression "2 +"
```

- Can generate errors:

```
error "bad argument"
```

- Global variable `errorCode` holds machine-readable information about errors (e.g. UNIX `errno` value).

Additional Tcl Features:

- String manipulation commands:

```
regexp  format  split   string
regsub  scan    join
```

- File I/O commands:

```
open    gets    seek    flush     glob
close   read    tell    cd
puts    source  eof     pwd
```

- Subprocesses with `exec` command:

```
exec grep foo << $input | wc
```

Additional Tcl Features, cont'd

- **Associative arrays:**

```
set x(fred) 44
set x(2) [expr $x(fred) + 6]
array names x
⇨ fred 2
```

- **Variable scoping:**

```
global   uplevel    upvar
```

- **Access to Tcl internals:**

```
info     rename     trace
```


Additional Tcl Features, cont'd

- **Autoloading:**
  - `unknown` procedure invoked when command doesn't exist.
  - Loads procedures on demand from libraries.
  - Uses search path of directories.
- **Coming soon (Tcl 7.5):**
  - Dynamic loading of binaries: `load` command.
  - Security: Safe-Tcl.
  - Event-driven I/O.
  - Socket support.

More On Substitutions

- **Keep substitutions simple: use commands like `format` for complex arguments.**
- **Use `eval` for another level of expansion:**

```
exec rm *.o
⇨ *.o: No such file or directory
glob *.o
⇨ a.o b.o
exec rm [glob *.o]
⇨ a.o b.o: No such file or directory
eval exec rm [glob *.o]
```


Commands And Lists: Quoting Hell

- **Lists parse cleanly as commands: each element becomes one word.**
- **To create commands safely, use list commands:**

`button .b -text Reset -command {set x $initValue}`
(`initValue` read when button invoked)

`... -command "set x $initValue"`
(fails if `initValue` is "`New York`": command is "`set x New York`")

`... -command "set x {$initValue}"`
(fails if `initValue` is "`{`": command is "`set x {{}`")

`... -command [list set x $initValue]`
(always works: if `initValue` is "`{`" command is "`set x \{`")

Tcl Syntax Summary

- **Script = commands separated by newlines or semicolons.**
- **Command = words separated by white space.**
- **`$` causes variable substitution.**
- **`[]` causes command substitution.**
- **`""` quotes white space and semi-colons.**
- **`{}` quotes all special characters.**
- **`\` quotes next character, provides C-like substitutions.**
- **# for comments (must be at beginning of command).**