An Introduction
To Writing Tcl Scripts

John Ousterhout

Computer Science Division
Department of EECS

University of California at Berkeley

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 syntax.

Basics

Tcl script =

- Sequence of commands.
- Commands separated by newlines, semi-colons.

Tcl command =

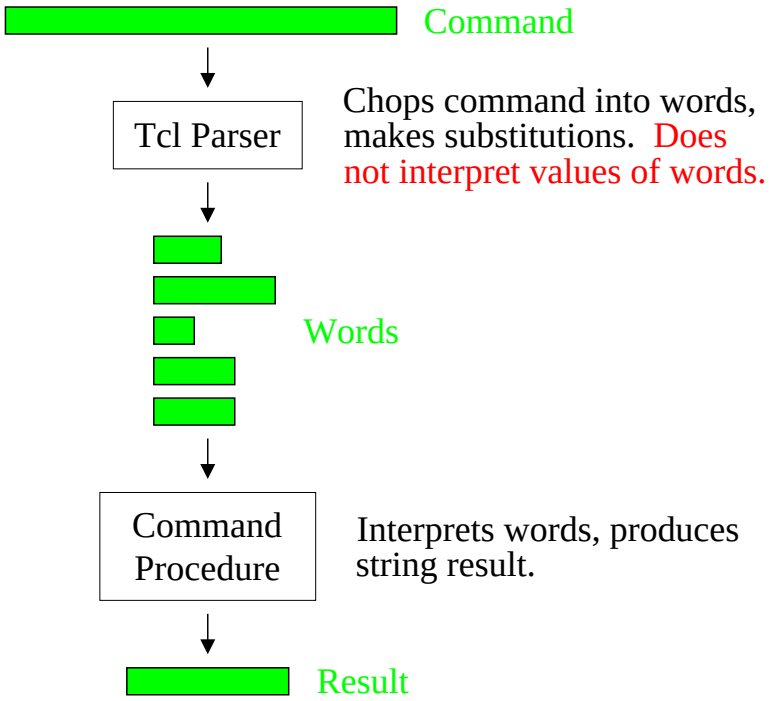
- One or more words separated by spaces.
- First word is command name, others are arguments.

Examples:

```
set a 22; set b 33

set a 22
set b 33
```


Division of Responsibility



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 within 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***]**
- Execute 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 "Funny word; has spaces"
```

- Curly braces prevent breaks and substitutions:

```
set a {nested {} braces}
```

- 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.
- Support for command and variable substitution within expressions.
- Used in **`expr`**, other commands.

Sample command	Result
`set b 5`	`5`
`expr ($b*4)-3`	`17`
`expr $b<=2`	`0`
`expr {$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    linsert   lreplace
foreach   list      lsearch
lappend   llength   lsort
lindex    lrange
```

- Example:

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

Control Structures

- C-like appearance.
- No new syntax: just commands that take Tcl scripts as arguments.
- Example:

```
if {$x < 3} {
   puts stdout "x is too small!"
   set x 3
}
```

- Commands:

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


Procedures

- **proc** command defines procedure:

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

name — list of argument names — body

- Procedures behave just like built-in commands:

```
sub1 3          returns 2
```

- Arguments can have defaults:

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

- Can have variable number of arguments:

```
proc foo {a b args} { ... }
```

gets list of extra args

- Scoping: local and global variables.

Errors

- Errors normally abort commands in progress, application displays 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 {$n + i*i}]..."
   ("foreach" body line 2)
   invoked from within
"foreach i {1 2 3 4 5} {
   set n [expr {$n + i*i}]
}"
```


Advanced Error Handling

- Can catch 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

1. **String manipulation commands:**

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

2. **File I/O commands:**

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

3. **Subprocesses with `exec` command:**

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

4. **History (`history` command).**

Additional Tcl Features, cont'd

5. **Associative arrays:**

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

6. **Variable scoping:**

```
global    uplevel    upvar
```

7. **Autoloading:**

- Tcl procedures loaded on demand.
- Search path of directories.

8. **Access to Tcl internals:**

```
info      rename     trace
```

Tcl Syntax Summary

1. Script = commands separated by newlines, semi-colons.
2. Command = words separated by white space.
3. **$** causes variable substitution.
4. **[]** causes command substitution.
5. **""** quotes white space and semi-colons.
6. **{}** quotes all special characters.
7. **** quotes next character, provides C-like substitutions.
8. **#** for comments (must be at beginning of command).