

# Tcl/Tk GUIDE

by Jason Bechtel

---

## What is Tcl/Tk?

Tcl is a scripting language. That means that it is best suited to high-level orchestration of other programs and tasks. Tk is the graphical extension to Tcl that allows for the creation of graphical user interfaces.

## Where do I start?

Tcl is an interpreted language at heart. So, you need to install an interpreter in order to use it. The interpreter is called *tclsh* (or *wish* for Tk). It can be used interactively or run on a Tcl/Tk (Tcl from now on) script that is stored in an ASCII text file. You can obtain most information about Tcl from [scripts.com](http://scripts.com) including how to download and install the latest version of the interpreter. Oh, and Tcl interpreters exist for Windows, UNIX, and Macintosh.

## Can we just program already?

We'll start with commands for Tcl in general, which all apply in Tcl and Tk scripts.

## Variables:

```
set <variable> <value>
$<variable> gives the value in variable.
value can be an int, float, or string.
eg: set x $y
eg: puts $x
```

## Commands and Comments:

A semi-colon or a carriage return separates commands. Comments begin with a '#' which must be preceded by a ';' if it follows a command on the same line.  
eg: set x 0 #bad comment

```
puts $x ;#good comment
#good comment
```

## I/O:

Standard I/O is performed with *puts* and *gets*:

```
eg: puts { 'Type your name: ' }
```

```
gets name
```

```
puts $name
```

File I/O requires a pipe to be opened:

```
set fileid [open {filename} r]
```

```
gets $fileid line
```

```
puts $line
```

```
close $fileid
```

## Quoting hell:

Commands are interpreted in two passes.

The first substitutes for variables. The second evaluates the resulting command.

Quoting using '{', '[', and '"' control the evaluation of the command. Curly braces disable substitution. Square brackets initiate a function call that performs a substitution and then interprets the command. Quotes allow substitution to occur. Then during evaluation, quotes and braces also work to group words into a single argument.

```
set var {I'm a string }
```

```
puts {$var }
```

```
> $var
```

```
puts { '$var' }
```

```
> '$var'
```

```
puts "${var}"
```

```
> {I'm a string }
```

The backslash (\) is the escape character.

Some common escape sequence strings:

```
\a audible alert (bell)
\f formfeed (clear screen)
\n newline
\t tab
```

## Math:

The command is *expr*. Examples:

```
set x [expr -3*4+5]
```

```
set x [expr "$a + $b"]
```

```
set x [expr sqrt($y)]
```

## Procedures:

The *proc* command sets up a procedure:

```
proc myfunc { arg1 arg2 } {
}
```

Then call it in a script just by typing the name of the procedure:

```
myfunc $x $y
```

Tip: You can store common functions off in a file and use the *source* command to include them:

```
source myfile.tcl
```

## More Variables:

Sharing variables between procedures:

```
proc myfunc { y } {
    global outside_var
    upvar $y ref_var
}
```

*global* causes *outside\_var* to be evaluated in the global scope. *upvar* causes *y* to refer to *ref\_var* in the next scope level up.

## Structured programming:

```
if {$x < $limit} ?then? {
    script } elseif {$x == 10} {
    script } else {
    script }
```

```
while {[eof $fileid]} { }
```

```
for {set x 0} {$x < $lim} {incr x} { }
```

```
foreach varname <list> { }
```

## Lists:

LISP anyone? In simplest form:

```
set x { one two three }
```

```

set y {me you {buckle my shoe} }
index x 2
> three
index y 2
> buckle my shoe
set z [list a b 3 string]
linsert x 1 foo; puts $x
> one foo two three
lrange z 0 2
> a b 3
lsort x; concat x z
> one three two a b 3 string

```

### Strings:

Strings are a basic type, so it's easy

```

set x {I'm a string}; set y string
set x_len [string length $x]
string index $y I
> t
string range $x 2 5
> m a s
string compare "foo" "bar"
> 1
string compare "bar" "foo"
> -1
string compare "foo" "foo"
> 0
string toupper "foobar"
> FOOBAR
string trim $y tr
> sing
Finally, C-style string formation:
set x [format "%s%d%4.2f" str 2 pi]
> str23.14

```

### Between lists and strings:

Lists can be turned into strings with the *join* command and strings can be chopped into lists with the *split* command:

```

set strvar [join {a b 3 string} ]
> a b 3 string
set listvar [split "cs.ua.edu" .]
> cs ua edu

```

### Regular Expressions:

Regular expressions are supported by several functions. There are also the explicit functions *regexp* and *regsub*. The simplest general forms are

```

regexp exp string
regsub exp string subSpec varName

```

Expressions use the following notation:

```

^ beginning of a string
$ end of a string
. any single character
* any count 0-n of the previous pattern
+ any count 1-n of the previous pattern
[] any char of a set of chars in
[^] any char NOT in the set of chars
() subSpec

```

Examples:

```

set sample "Where there is a will "
set result [regexp {[a-z]+} $sample match]
puts "$result match: $match"
> 1 match: here
set result [regexp {[A-Za-z]+} +([a-z]+) ] \
$sample match sub1 sub2 ]
puts "Match: $match 1: $sub1 2: $sub2"
> Match: Where there 1: Where 2: there
regsub "will" $sample "lawsuit" sample2
puts "New: $sample2"
> Where there is a lawsuit
Note: exps are subject to interpreter
substitution. Escape characters accordingly.

```

### Associative Arrays:

Tcl supports a very flexible array structure. Indices are indicated with strings rather than numerals and the data type of the values in

the array can vary from index to index and through time for a single index.

Examples:

```

set myArray(first) 'NSF REU'
set myArray(last) 1620
set myArray(3) twelve
set myArray(first) 4.78

```

Some additional commands:

array exists arrayName  
array names arrayName ?pattern

Returns a list of indices *in no particular*

*order*. Pattern restricts the list to indices with values matching *pattern*.

array size arrayName

### Between Arrays and Lists:

array get arrayName

Returns a list in which each odd member of the list (1, 3, 5, etc) is an index into the associative array. The list element following a name is the value of that array member.  
array set arrayName dataList

Converts a list into an associative array.

*dataList* is a list in the format of that returned by *array get*.

---

### Where do I go from here?

There is a lot in Tcl without even getting into Tk. And there are a plethora of extensions that have been written for various specialized functions. I recommend going to [scriptics.com](http://scriptics.com) for book recommendations. Also, download TclTutor by Clif Flynt. It comes in handy.