Lecture #20 – TCL

- Background

  TCL (tickle) [Tool Command Language] is a scripting lang (similar to C-shell or Perl)
  TK (tee-kay) is a user interface toolkit (i.e. scripting language for GUI interfaces)
  Expect is a scripting language extension to TCL / Tk for automating interactive apps

  Publicly, and freely available but not standard part of UNIX
  Written by John Ousterhout.

  Official source code is at ftp://ftp.scriptics.com/pub/tcl

  TCL is really two things – a language and a library
  As a language, TCL is primarily intended to help issue commands to interactive prog
  As a library, TCL is intended to be embedded in other applications

  TCL is very extensible (i.e. you can write new commands for TCL)

- General

  High-level script language like many of the others we have seen this term
  Interpreted (although it's possible to compile)
  Extensible (through C or TCL)
  FREE

  Each line has a "command" followed by "arguments" for all of its syntax
  This leads to some interesting syntax…

  Span lines with backslash (\) or {}

  The main TCL interpreter is "**tclsh**" which can be used in batch mode or interactive mode
  To use in a script, change your "#!" line to point to the location of "**tclsh**"
  To use interactive mode, type "**tclsh**" at the prompt

  The main Tk interpreter is called "**wish**" (i.e. wish hello.tcl)
  Tk is a superset of TCL (i.e. all of TCL works here + Tk)

  The main Expect interpreters are "**expect**" and "**expecttk**"
  Expect is a superset of TCL while expecttk is a superset of both TCL and Tk

- Variables

  Assignment is done with “set” and they are used with “$”

  Example:

  ```
  set foo “Rich”
  puts “Hello my name is $foo”
  ```

  Example:

  ```
  set month 3
  set day 25
  set year 2001
  set date “$month:$day:$year”
  puts $date
  ```

  Output: 3:25:2001

  Example:

  ```
  set foo “puts hi”
  eval $foo
  ```

- Expr

  Most expressions in TCL are evaluated with expr

  Example:

  ```
  expr 0 == 1     (result:  0 or false)
  expr 1 == 1     (result:  1 or true)
  expr 4 + 5      (result:  9)
  ```

- Command substitution

  Cmd substitution is done with square brackets (i.e. [])

  Example:

  ```
  set height 6.0
  puts “If I was an inch taller, I would be [expr $height + 1.0 / 12.0] feet tall”
  ```

- Control flow

If statement example:

```
set my_planet "earth"
if {$my_planet == "earth"} {
        puts "I feel right at home."
} elseif {$my_planet == "venus"} {
        puts "This is not my home."
} else {
        puts "I am neither from Earth, nor from Venus."
}

set temp 95
if {$temp < 80} {
        puts "It's a little chilly."
} else {
        puts "Warm enough for me."
}
```

Output:

```
I feel right at home.
Warm enough for me.
```

Switch statement example:

```
set num_legs 4
switch $num_legs {
        2 {puts "It could be a human."}
        4 {puts "It could be a cow."}
        6 {puts "It could be an ant."}
        8 {puts "It could be a spider."}
        default {puts "It could be anything."}
}

switch -regexp aaab {
        ^a.*b$ -
        b {format 1}
        a* {format 2}
        default {format 3}
}
```

For loop example:

```
for {set i 0} {$i < 10} {incr i 1} {
   puts "In the for loop, and i == $i"
}
```

While loop example:

```
set i 0
while {$i < 10} {
   puts "In the while loop, and i == $i"
   incr i 1
}
```

Foreach loop example:

```
foreach vowel {a e i o u} {
   puts "$vowel is a vowel"
}
```

- Procedures

Syntax: proc name arglist body

Example:

```
proc sum_proc {a b} {
   return [expr $a + $b]
}

proc magnitude {num} {
   if {$num > 0} {
      return $num
   }

   set num [expr $num * (-1)]
   return $num
}

set num1 12
set num2 14
set sum [sum_proc $num1 $num2]
puts "The sum is $sum"
puts "The magnitude of 3 is [magnitude 3] and of -2 is [magnitude -2]”
```

Output:

```
The sum is 26
The magnitude of 3 is 3 and of -2 is 2
```

Example (global and local variables):

```
proc dumb_proc {} {
   set myvar 4
   puts "The value of the local variable is $myvar"

   global myglobalvar
   puts "The value of the global variable is $myglobalvar"
}

set myglobalvar 79
dumb_proc
```

Output:

```
The value of the local variable is 4
The value of the global variable is 79
```

- Lists

Simple means to group items into a single entity, but use them in string context

Example:

```
set simple_list "John Joe Mary Susan"
puts [lindex $simple_list 0]
puts [lindex $simple_list 2]
```

Output:

```
John
Mary
```

Example (compond lists, and llength):

```
set simple_list2 "Mike Sam Heather Jennifer"
set compound_list [list $simple_list $simple_list2]
puts $compound_list
puts [llength $compound_list]
```

Output:

```
{John Joe Mary Susan} {Mike Sam Heather Jennifer}
2
```

Example: (linsert and lappend)

```
set mylist "Mercury Venus Mars"
puts $mylist
set mylist [linsert $mylist 2 Earth]
puts $mylist
lappend mylist Jupiter
puts $mylist
```

Output:

```
Mercury Venus Mars
Mercury Venus Earth Mars
Mercury Venus Earth Mars Jupiter
```

- Arrays

Example:

```
set myarray(0) "Zero"
set myarray(1) "One"
set myarray(2) "Two"

for {set i 0} {$i < 3} {incr i 1} {
   puts $myarray($i)
}
```

Output:

```
Zero
One
Two
```

Example (associative arrays):

```
set person_info(name) "Fred Smith"
set person_info(age) "25"
set person_info(occupation) "Plumber"

foreach thing {name age occupation} {
   puts "$thing == $person_info($thing)"
}
```

Output:

```
name == Fred Smith
age == 25
occupation == Plumber
```

Example (associative arrays with unknown indicies):

```
foreach thing [array names person_info] {
   puts "$thing == $person_info($thing)"
}
```

Output:

```
occupation == Plumber
age == 25
name == Fred Smith
```

- Strings

Example:

```
set str "This is a string"
puts "The string is: $str"
puts "The length of the string is: [string length $str]"
puts "The character at index 3 is: [string index $str 3]"
puts "The characters from index 4 through 8 are: [string range $str 4 8]"
puts "The index of the first occurrence of letter \"i\" is: [string first i $str]"
```

Output:

```
The string is: This is a string
The length of the string is: 16
The character at index 3 is: s
The characters from index 4 through 8 are: is a
The index of the first occurrence of letter "i" is: 2
```

- Input and Output

Example:

```
puts -nonewline "Enter your name: "
set bytesread [gets stdin name]

puts "Your name is $name, and it is $bytesread bytes long"
```

Output: (note that user input is shown in italics)

```
Enter your name: Rich

Your name is Rich, and it is 4 bytes long
```

Example:

```
set f [open "/tmp/myfile" "w"]

puts $f "We live in Texas. It's already 110 degrees out here."
puts $f "456"

close $f
```