TCL Scripting for Cisco IOS

Petr Grygárek
Automation using TCL Scripting

- Network monitoring
- Troubleshooting tools
- Added intelligence
Tool Command Language (TCL)

- Invented by John K. Ousterhout, Berkeley, 1980s
- Interpreted language
  - support for compilation into bytecode
- Runtime available for many platforms
- http://www.tcl.tk/
Tool Command Language (TCL)


- Interpreted language
  - support for compilation into bytecode

- Intended for scripting, rapid prototyping, embedding into applications, creation of GUIs (TCL/Tk toolkit)

- Runtime engine available for many platforms
TCL Basic Features
(taken from http://en.wikipedia.org/wiki/Tcl)

• Prefix command notation
  • variable number of arguments

• No data types
  • all values treated as strings

• Everything can be dynamically redefined and overriden

• Object-oriented extensions are available

• Many extension libraries were developed
IOS Policies

• **Applets**
  - sequences of IOS commands
  - Stored in device's running config

• **TCL Scripts**
  - Programs in TCL
  - Stored on FLASH or external storage

• **Policies are subscribed with Embedded Event Manager (EEM) to be activated when specific event(s) occur(s)**
  - They also can be activated manually
Embedded Event Manager

- Detects interesting events
  - using Event Detectors
- Triggers specific policy based when a specific event (or combination of events) occurs
Event Detectors

• Monitor SW and HW components for specific events

• Examples of event detectors:
  • CLI,
  • Timer
  • Syslog
  • Object Tracking
  • interface state change detector
  • insertion/removal of module detector
  • ...

© 2009 Petr Grygarek, Advanced Computer Networks Technologies
How to Execute TCL Script from Cisco IOS

- `tclsh flash0:myScript.tcl`
- TCL interactive shell mode: `tclsh`
  - Unrecognized (i.e. non-TCL) commands are passed to IOS CLI
Basic TCL Commands and Structures
Getting Help

• info commands
• info exists <varName>
• info args <procName>
• info body <procName>
• info globals
• info vars

Command typed with wrong argument(s)
make tclsh to display usage help

# this is a comment
Assignments, Expressions, Displaying Outputs

set x 1
puts $x
set x [expr $x+1]
puts $x
incr x -10

set p1 kocour
set p2 mour
set p3 "$p1 $p2" -> kocour mour
“printf-like” Output
Formatting

set a 1
set s kocour
set f [format "int: %d, string: %s" $a $s]
f now contains “int: 1, string: kocour”

Text in [] is replaced with result of executed TCL code contained in block
Expr command

- Examples:
  - `set r [expr {rand()}]` -> float (0,1)
Arrays

- Array is treated as set of associated pairs
  - no space pre-allocation
  - keys of any type

```ruby
set a(1) 10
set a(dog) Zeryk
puts $a(1) -> 10
puts $a(dog) -> Zeryk
puts $a(2) -> can't read "a(2)": no such element in array
array set a "kocour mour number 2"
puts $a(kocour) -> mour
puts $a(number) -> 2
```
Array Functions

- unset a(1)
  - deletes one association from a

- unset a
  - destroys the whole array

set a(1) 10
set a(2) 20
array get a -> 1 10 2 20
array get a 1  -> 1 10
array size a  -> 2
array names a -> 1 2
Strings

• string <operation> <argument(s)?
  • e.g. string first “needle” $hay

hay {aa bb cc bb dd}
string first bb $hay -> 3
Lists

- List is a string consisting of values separated by whitespaces.
- List manipulation functions:
  - Llength,
  - lappend, linsert, lreplace, lrange
  - lindex, lsearch,
  - lsorth
Loops and Iterators

for {set i 0} {$i<10} {incr i} { puts $i }

set i 0
while {$i < 10} { puts $i; incr i }

set lst {1 2 3 4 5 6 7 8 9}
foreach {a1 a2 a3} $lst
    { puts "a1=$a1, a2=$a2, a3=$a3" }

a1=1,a2=2,a3=3
a1=4,a2=5,a3=6
a1=7,a2=8,a3=9
Conditional Execution

set x 1
if {$x < 10} { puts LESS } else { puts GREATER }
-> LESS
Procedures

proc myproc {p1 p2} {
    set res [expr $p1+$p2]
    return $res
}

set sum [myproc 10 20]
Files

set fd [open flash:f.txt w]
puts $fd kocour
puts $fd mour
close $fd

Router# more flash:f.txt

set fd [open flash:f.txt r]
while { [gets $fd line] > 0 } { puts $line }
close $fd

tell $fd, seek $fd <pos>
file <operation> <argument(s)>
e.g. file delete flash:f.txt
Handling Script Arguments

sample.tcl:
puts "\n"
puts "Argument count: $argc"
puts "Argv0: $argv0"
puts "Argv: $argv"
puts "Individual arguments:"
foreach  {iterVar} $argv { puts $iterVar }

router #tclsh http://10.0.0.2/sample.tcl  aaa bbb ccc
-> Argument count: 3
Argv0: http://10.0.0.2/kocour.tcl
Argv: aaa bbb ccc
Individual arguments:
 aaa
 bbb
 ccc
Interactions between TCL Policies and IOS CLI
Running TCL Scripts

- **From TCL shell**
  - "source" TCL command
    
    ```
    router(tcl)#source flash:mysrc.tcl
    router(tcl)#source http://10.0.0.2/kocour.tcl
    ```

- **From IOS exec mode**
  - "tclsh" command followed by script name
    
    ```
    router#tclsh http://10.0.0.2/kocour.tcl args
    ```

- Arguments cannot be passed to script using source command

- Multiple scripts may run in parallel.
Exec Mode Commands

- `log_user 0/1` - disables/enables displaying of CLI commands outputs
- `set cliOutput [exec "sh ip interface brief"]` - works both in interactive TCL shell and TCL scripts
Config Mode Commands (TCL Shell)

```tcl
ios_config "hostname MYNAME"
ios_config "router rip" "network 10.0.0.0" "end"
```

- It is recommended to exit from TCL shell for the configuration changes to take effect.
- Always end the configuration commands with end to avoid locking.
Config Mode Commands (EEM TCL Policies)

- FD-style functions
- `cli_open`, `cli_write`, `cli_read`, `cli_exec`
  - `cli_exec = cli_write + cli_read`
- Work in TCL scripts, NOT in interactive TCL shell.
- On the other hand, `ios_config` does NOT work in TCL scripts (?)
Dealing with Interactive Commands

Router(tcl)#typeahead

Router(tcl)#exec "copy run flash:x.x"

• Does not work in TCL Shell interactive mode

• Alternative: file prompt quiet IOS command
Policy Registration with EEM

• Either applet or TCL script may be registered to be activated when an event is detected
  
  event manager directory user policy flash:/scripts
  event manager policy myScript.tcl

• Specification of the event to trigger the policy is defined at the beginning of policy's TCL script:
  
  ::cisco::eem::event_register_timer cron name myCron1 cron_entry "0-59 0-23 * * 0-7"
Checking Registered Policies

- `show event manager policy available [user | system]`
- `show event manager policy registered`
- `sh event manager history event`

EEM policies have to be stored on some local filesystem to ensure their availability regardless of the current state of the connectivity to any external storage server.
Manual Policy Launching

• event manager run myScript.tcl
  • only applicable for policies registered with none event
Specification of Policy's Environment

- Router(config)#event manager environment myVariable myValue
- Router(config)#event manager session cli username kocour
- sh event manager session cli username
Example Policy
(launched manually)

::cisco::eem::event_register_none
namespace import ::cisco::eem::*
namespace import ::cisco::lib::*
array set cliconn [ cli_open ]
puts $cliconn
cli_exec $cliconn(fd) "hostname CHANGED-NAME"
cli_close $cliconn(fd)
EEM Applets

- Definition consists of
  - Events to be detected to trigger the applet
    - available events may vary with different IOS/EEM versions
  - Sequence of IOS commands to be executed
    - Sorted lexicographically according to line tags
Most Interesting Supported Features (1)

- Reaction to composite events
- Reacting to interface status change
- Processing of RIB change events
- Reacting to IOS object status change
  - Enhanced Object Tracking
- Reacting to Syslog messages
- Reacting to increased resource utilization (CPU, memory, ...)
- Integration with SLA monitoring
- Timers & Counters events
Most Interesting Supported Features (2)

- Sockets Library
- SNMP Library (outgoing/incoming messages)
- SMTP Library
- Integration with Netflow
- CLI library & events
  - issuing IOS exec and config commands
  - interception of command handling process
  - creation of user commands and/or extending command parameters
Most Interesting Supported Features (3)

- Messaging between policies running in parallel, policy synchronization
- Policy priorization
  - multiple scheduling queues, nice, ...
- Persistent storage to keep script's internal state between runs
- Remote Procedure Call (RPC)
- XML-PI
- TCL scripts debugging support
References