TEXAS ADVANCED COMPUTING CENTER

WWW.TACC.UTEXAS.EDU

TEXAS
The University of Texas at Austin

The complicated story about TCL break

Robert McLay

May. 3, 2022

Outline

- Let's talk about TCL break (and LmodBreak)
- Lmod didn't really support TCL break at all until Lmod 8.6 (really Lmod 8.7)
- Years ago mailing list question: support for break
- Lmod 6 and below could not support break
- Why?

Reminder: How Lmod works

- In order to have a command effect the current shell
- A simple module command for bash is given below
- The $LMOD_CMD command generate shell commands as text
- The eval "..." evaluate the text to change the current shell
- For the rest of this talk: focus on what $LMOD_CMD produces

```
module () { eval "$($LMOD_CMD bash "$@")"; }
```

Reminder: How Lmod TCL processing works

- Internally Lmod knows when a file is a TCL modulefile
- No *.lua extension ⇒ TCL modulefile
- The program tcl2lua.tcl is called to process the tcl
- It converts TCL modulefile into Lua with Lmod module commands

```
setenv FOO bar ⇒ setenv("FOO","bar")
prepend-path PATH /prgm/bin ⇒ prepend_path("PATH","/prgm/bin")
break ⇒ LmodBreak() -- Only for bare breaks
```

TCL Break

```
for {set i 0} {$i < 5} {incr i} {
  puts stderr "$i"
  if { $i == 3 } {
      break  # This breaks out of the loop
  }
}
break # This causes the modulefile
      # to stop being processed.
```

Why was TCL break such a problem for Lmod?

- TCL break stops processing the current module
- It ignores any changes in a module that has a break
- But it keeps all other modules loaded.
- `module load A B C D`
- Where C has a break
- Then A B are loaded but C and D are not.

LmodError is different

- `module load A B C D`
- Where C has an LmodError()
- No modules are loaded.

Lmod waits to produce output

- When loading several modules, Lmod waits
- All module actions are completed internally
- Then Lmod generates shell command output.
- Lmod 6 and earlier wouldn't know what changes to ignore when processing a break.
- Lmod produces either an error or environment changes not both.

Lmod 7+ was a complete re-write of Lmod

- It was needed to support Name/Version/Version (N/V/V) modulefiles
- Before Lmod only supported N/V or C/N/V
- Lmod 7+ now has a FrameStk (AKA the stack-frame)
- The FrameStk contains a stack of the environment var table (varT) and the module table (mt)

FrameStk: varT and mt

- The table varT contains key-value pairs that represent the new env. var values
- The table mt is the module table containing the currently loaded modules among other things
- The Module Table is stored in the environment via $_ModuleTable001_ etc.

assignment versus deepcopy() in Lua

```
a = {}
a[1] = "foo"
b    = a
b[1] = "bar"
print(a[1])  -> ``bar'' not ``foo''
```

- Lua tries to be efficent
- It just copies reference
- As shown above.
- Lmod provides deepcopy() function.
- This creates a new table

FrameStk

- Before each module: Deep Copy copies the previous varT and mt to top of FrameStk.
- Each evaluation of modulefile is updated on the top of the FrameStk
- When the current modulefile evaluation is completed
- The FrameStk is pop'ed
- The previous stack values are replaced with current

FrameStk implications

- Cannot trust local values of mt
- Lmod constantly has to refresh mt:
- mt = frameStk:mt()
- Because a module load might have updated it.

LmodBreak or TCL break

- If LmodBreak() is called, the current module changes are ignored
- LmodBreak() causes the previous values to be current
- FrameStk:pop() pops the stack.
- The FrameStk code is shown below:

```
function M.LmodBreak(self)
   local stack           = self.__stack
   local count           = self.__count
   stack[count].mt       = deepcopy(stack[count-1].mt)
   stack[count].varT     = deepcopy(stack[count-1].varT)
end

function M.pop(self)
   local stack           = self.__stack
   local count           = self.__count
   stack[count-1].mt     = stack[count].mt
   stack[count-1].varT   = stack[count].varT
   stack[count]          = nil
   self.__count          = count - 1
end
```

Support for TCL break

- Lmod 8.6+ added support LmodBreak()
- Lmod 8.6+ added support a bare TCL break
- Lmod 8.7+ added support for regular break and bare break

TCL Break strangeness

```
for {set i 0} {$i < 5} {incr i} {
  puts stderr "$i"
  if { $i == 3 } {
      break  # This breaks out of the loop
  }
}
break # This causes the modulefile
      # to stop being processed.
```

- TCL treats a bare break as an error
- Tmod 3, 4 and 5 catch the error
- Lmod 8.7+ now catch the error too!

To support regular and bare break in TCL in tcl2lua.tcl

```
set sourceFailed [catch {source $ModulesCurrentModulefile } errorMsg] # (1)
set returnval 0
if { $g_help && [info procs "ModulesHelp"] == "ModulesHelp" } {
   # handle module help
   ...
}
if {$sourceFailed} {
    if { $sourceFailed == 3 || $errorMsg == {invoked "break" outside of a loop}} {
        set returnVal 1
        myBreak
        showResults
        return $returnVal
    }
    reportError $errorMsg
    set returnVal 1
}
showResults
return $returnVal
```

- line 1 evaluate the TCL modulefile
- $sourceFailed will be non-zero for TCL errors
- $sourceFailed == 3 means a bare break has been found.

What happens when?

```
% cat C.lua
load("X","Y")
LmodBreak()

% module load A B C D
% module list
Currently Loaded Modules:
  1) A    2) B
```

- Module A and B are loaded internally
- When loading C, modules X and Y are loaded internally
- When LmodBreak() is encounter, processing of C stops
- Also the effects of X and Y are ignored.

Lmod 8.6.15 could create an endless loop

```
% cat foo3/1.0
#%Module
catch {set foo $env(FOO)}
if { [info exists foo] } {
  puts stderr "already set"
  break
}
setenv FOO "just me"

% module load foo3/1.0; module load foo3/1.0
already set
already set
...continues until crtl+C ...
```

- Loading foo3/1.0 twice causes an endless loop
- Why?
- The second load forces foo3/1.0 to unload (which it can't)
- Lmod then tries to re-load foo3/1.0 which causes the unload etc.

LmodBreak() is a no-op on unload

- A bare TCL break becomes an LmodBreak() when translated.
- LmodBreak() does nothing during unload.
- This prevents the endless loop shown above. (Lmod 8.7+)

Conclusions

- Implementing break is trickier then you might think.
- Lmod now can support bare breaks finally in Lmod 8.7
- The FrameStk is the price to be paid to support break.

Future Topics

- Next Meeting: June 7th 9:30 US Central (14:30 UTC)
- Show how Lmod processes a module load command, stepping through the codebase.
- Suggestions?