# Lmod Testing System

Robert McLay

March. 1, 2022

# Lmod Testing System

- Testing philosophy in Lmod
- Goals of testing Lmod
- Hermes/tm basic operations
- Details of how an Lmod test works
- Future Topics

# Alternative story

- How I used a testing tool I already had
- How to shoehorn the Lmod testing to use `tm`
- Why I'm using a system testing method, not unit tests.
- System tests came first for me, unit testing later.
- Lmod uses some unit tests as well.

# Testing philosophy in Lmod

- Lmod's success relies heavily on the testing system.
- Passing all the tests usually means a new version can be released.
- I don't think that anyone is using it beside Lmod (But it is very useful)
- My philosophy is to test features in general
- Not to setup a torture test
- No way I can test every possible scenario.
- My imagination is not that good.

# Goals of testing Lmod

- Test various features of Lmod.
- New feature won't break old features.
- Test Lmod on Linux/MacOS, Lua 5.1 to 5.4
- Make development of Lmod easier.
- Add tests of new bugs ⇒ Don't repeat them!

# It is hard to test everything

- Testing Old data with new versions(Collections, spiderT.lua)
- One test (end2end) builds Lmod and tests the built version
- All other tests use the source code directly
- Special hacks to use configuration options.
- Environment variable are checked NOT configuration options

# Hermes/tm Testing system

- Hermes is a group of tools to help with testing
- `tm` is the testing manager program.
- The main function of `tm` is to select tests and run them.
- Each test is independent!
- `tm` knows *nothing* about what is being tested.
- Must tell if test passed via special file (Lua file named t1.results)
- Three kinds of results
  1. Passed: All steps passed
  2. Failed: Did not produce a t1.results file
  3. Diffed: Produced diffs between gold files and test result files.

# `tm` flow

- `tm` searches for tests from the current directory down
- It is looking for files with the *.tdesc extension (testDir)
- Once all tests have been selected, it runs them all
- For each test directory a sub-dir tree is created.
- Typically: t1/<$TARG>-<date_time>-<uname -s>-<arch>-<test_name>
- The above dir is the outputDir
- The test is run in $outputDir
- The generated bash test script is named t1.script
- The log of the run is t1.log
- The results file are t1.result and t1.runtime

# Every project using **tm** must have an *acceptance* tool

- There must be an automatic way to decide a test passed.
- A numerical code can use an $L^2$ norm.
- The new answer can be different but close w/ numerical codes.
- Lmod use diff on stdout and stderr between gold and test results
- Filtering is required to deal with OS and file location differences
- To pass the filtered result *must* be the same.
- This is a major pain but it has been worth the effort.

# Test files (*.tdesc)

- The testDescript is a table describing the the test
- Some special parameters are:
  1. $(testDir): where the *.tdesc is located
  2. $(projectDir): where Hermes.db is located (top of the project)
  3. $(outputDir): where the test is run
  4. $(resultFn): The name of the results lua file.

# Lmod tests

- Uptil now this talk has been about `tm`
- Now lets talk about Lmod tests:
  - Each test contains multiple steps
  - Each step generates _stderr.### and _stdout.### files
  - These are combined and filtered into err.txt and out.txt
  - These file are compared with the gold files in $testDir
  - Result file is generated.
  - To pass all steps must be the same!

# extension.tdesc

```
local testName = "extensions"
testdescript = {
   keywords = {testName },
   active   = true,
   testName = testName,

   runScript = [[
     . $(projectDir)/rt/common_funcs.sh
     unsetMT;  initStdEnvVars
     export MODULEPATH_ROOT=$(testDir)/mf
     export MODULEPATH=$MODULEPATH_ROOT/Core
     rm -rf _stderr.* _stdout.* err.* out.* .lmod.d

     runLmod --version                         # 1
     runLmod avail                             # 2

     # combine _stdout.[0-9][0-9][0-9] -> _stdout.orig
     # cleanup _stdout.orig -> out.txt

     # combine _stderr.[0-9][0-9][0-9] -> _stderr.orig
     # cleanup _stderr.orig -> out.txt

     wrapperDiff --csv results.csv $(testDir)/out.txt out.txt
     wrapperDiff --csv results.csv $(testDir)/err.txt err.txt
     testFinish -r $(resultFn) -t $(runtimeFn) results.csv
   ]],
   tests = {
      { id='t1'},
   },
}
```

# $(projectDir)/rt/common_funcs.sh

- Common bash shell functions are in this file
- runLmod: runs the Lmod command
- runBase: base command (explained later)
- cleanup: Makes output generic (canonical?)
- initStdEnvVars: set standard env vars, cleans up my env
- unsetMT: remove moduletable from env

# runLmod

```
runLmod ()
{
   ############################################################
   # turn off file globbing if it is not already off
   ...

   runBase $LUA_EXEC $projectDir/src/lmod.in.lua bash --regression_testing "$@"
   eval `cat _stdout.$NUM`

   ############################################################
   # turn on file globbing for users who want it.
   ...
}
```

# runBase

```
runBase ()
{
   COUNT=$(($COUNT + 1))
   numStep=$(($numStep+1))
   NUM=`printf "%03d" $numStep`
   echo "===========================" >  _stderr.$NUM
   echo "step $COUNT"                 >> _stderr.$NUM
   echo "$@"                          >> _stderr.$NUM
   echo "===========================" >> _stderr.$NUM

   echo "===========================" >  _stdout.$NUM
   echo "step $COUNT"                 >> _stdout.$NUM
   echo "$@"                          >> _stdout.$NUM
   echo "===========================" >> _stdout.$NUM

   numStep=$(($numStep+1))
   NUM=`printf "%03d" $numStep`
   "$@" > _stdout.$NUM 2>> _stderr.$NUM
}
```

# Cleanup for stderr

```
cat _stderr.[0-9][0-9][0-9] > _stderr.orig
cleanUp _stderr.orig err.txt
```

- Combine all stderr files into _stderr.orig
- Use the cleanup shell function to canonicalize err.txt output

# Cleanup for stdout

```
cat _stdout.[0-9][0-9][0-9] > _stdout.orig
joinBase64Results  -bash  _stdout.orig _stdout.new
cleanUp _stdout.new out.txt
```

- Combine all stdout files into _stdout.orig
- Convert all base64 text into regular text
- Use the cleanup shell function to canonicalize out.txt output

# Cleanup script

- converts local path names into "ProjectDIR"
- converts path to lua or sha1 to generic names
- Cleans up error msgs
- And many other fixes.

# Cleanup script (II)

_stderr.orig:

```
============================
step 8
/opt/apps/lua/lua/bin/lua /Users/mclay/w/lmod/src/lmod.in.lua bash --rtesting -
t avail
============================
/Users/mclay/w/lmod/rt/avail/mf/Core:
PrgEnv
admin/
admin/admin-1.0
```

err.txt:

```
============================
step 8
lua ProjectDIR/src/lmod.in.lua bash --rtesting -t avail
============================
ProjectDIR/rt/avail/mf/Core:
PrgEnv
admin/
admin/admin-1.0
```

- Cleanup: _stderr.orig ⇒ err.txt

# Deciding if a test passes

```
rm -f results.csv
wrapperDiff --csv results.csv $(testDir)/out.txt out.txt
wrapperDiff --csv results.csv $(testDir)/err.txt err.txt
testFinish -r $(resultFn) -t $(runtimeFn) results.csv
```

- `wrapperDiff` is a hermes tool that runs diff and generates a csv file (results.csv)
- It also removes the Lmod version info from err.txt
- `testFinish` is another hermes tool that converts results.csv into $resultFn
- Then `tm` reads $resultFn to decide if this test passes

# Future Topics

- Write one new test.
- Explain how Mname object converts names into a filename.
- More internals of Lmod?