BSDLUA
(in three parts)

Evolved Unix Scripting

Ivan Voras <ivoras@freebsd.org>

What is BSDLUA?

- An experimental idea

- Use Lua for projects, tools, etc. which do not require C and would be more easily implemented in a scripting language

- An "in between" language – low-level features of C with integration capabilities of shell scripts

Why???

- The $1M question: what in the world would make someone program in something which is not /bin/sh ???

- /bin/sh is the best thing since the invention of the bicycle... from the time when Unix programmers had real beards...
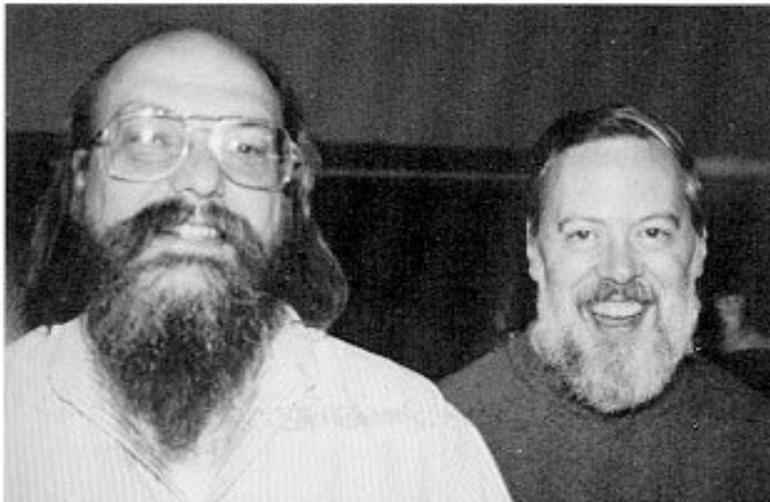
(More specifically)

- I personally miss a "higher level" scripting language in the base system

- In the beginning there was TCL (or so I heard... it was before my time)

- The there was Perl...

- Both were thrown out
  - For good reasons

What is wrong with shell scripts?

- Nothing … and everything
- Good sides: integration with system tools via executing programs, sourcing other scripts
- Bad sides: … somewhat depend on personal tastes … for me it's the syntax and lack of modern features
- Talking about the /bin/sh POSIX shell – more modern shells have nicer languages

Why not use /bin/sh?

- (for complex programs)

- Syntax from the 1970-ies

- No local variables

- No "proper" functions (with declared arguments)

- Need to escape strings more often than what would sanely be expected

- Relies on external tools for common operations (tr, grep, join, jot, awk…)

- Too much "magic" in operation

Why use Lua? (1)

- As a language:
- Nicer, modern language with lexical scoping
- Namespaces
- Dynamically and weakly typed
    - NULL, boolean, number, string, "table"
- "Table" data type for many uses
    - lists, records, OOP, metaprogramming
- First-class functions, closures, coroutines, tail calls
- Garbage collection, exceptions (sort of)

Why Lua might be better than Perl and TCL

- <u>For scripting, when compared to /bin/sh:</u>
- It is not a shell but it has an interactive interpreter
- /bin/sh is: ~~ 15,000 LOC, 47 files
- Lua (with libs) is: ~~ 17,000 LOC, 58 files
  - *It is small!*
- Its development is stable
  - It can be imported so not to conflict with ports
- MIT Licensed (BSD-friendly)
- Easy to embed to and from C!

About Lua

- Means "The Moon" in Portuguese (Brasilian) (and other Romance languages, cf. Latin "Luna", English "Lunatic")

- Originally an academic project

- Hugely successful as a general-purpose scripting language

  - Popular in gaming industry for story and game logic scripting!

  - Also used in: Wireshark, Vim, Apache, lighttpd, VNC and others

How does Lua look like?

```lua
tbl = { x = "Free", y = "BSD" }
for k,v in pairs(tbl) do print(k,v) end
-- this is a comment
print(string.format("%s\t%s", tbl.x, tbl.y))
```

- Not very sigil-dependant (@#%${}!)
- Functions can return more than 1 value
- for … do … end
- if … then … else … elseif … end
- Built-in "foreach" statement
- "-- …" are single-line comments

Lua's standard library

- **math**: trig. functions, sqrt, pow, log, random...
- **table**: table functions: array slices, sorting...
- **string**: slicing, upper / lower, regexp...
- **io**: simple file IO: open, read, write...
- **os**: *very* basic OS functions: time, getenv...
- **debug**: introspection, profiling

Advanced example

```lua
function _range(n)
  local x
  for x = 1,n do coroutine.yield(x) end
end

-- Python-like range() function
function range(n)
  return coroutine.wrap(
                function () _range(n) end)
end

for x in range(10) do print(x) end
```

Extending Lua

```c
/**
 * Records a message in the system's syslog.
 */
static int
bsd_syslog(lua_State *L)
{
if (!lua_isnumber(L, 1) || !lua_isstring(L, 2))
luaL_error(L, "Argument error. Expecting integer priority and "
    "message string");
syslog(lua_tointeger(L, 1), lua_tostring(L, 2));
return (0);
}
```

Lua's stacks

- Interfaces to C use "virtual stacks"
- (more like a list with random access than a stack, negative indices possible: -1=top)
- First function argument is S(1), second S(2)...
- Returning values from C: lua_pushnumber(42)
- Creating tables:
    - lua_newtable()
    - lua_pushnumber(X)
    - lua_setfield(L, -2, "fieldname")

Extending Lua - an example

```c
/**
 * Records a message in the system's syslog.
 */
static int

bsd_syslog(lua_State *L)

{

if (!lua_isnumber(L, 1) || !lua_isstring(L, 2))

 luaL_error(L, "Argument error. Expecting integer priority and "

  "message string");

syslog(lua_tointeger(L, 1), lua_tostring(L, 2));

return (0);

}
```

Boilerplate C declaration

Validate arguments

Error / exception (doesn't return)

Fetch arguments and call syslog(3)

Number of returned values

Where to find Lua?

- On the Internet, here:

    - **www.lua.org**

- A large community with a good track record

- Lua's progress in time:

    - First version: 1993.

    - Lua 2.0 – 1994., Lua 3.0 1997., Lua 4.0 2000.

    - Lua 5.0 2003.

    - **Lua 5.1 2006.**   (most recent: 5.1.4, 2008.)

    - (Lua 5.2 in 2011?)

Lua and FreeBSD?

- Maybe, if enough people get interested

- Lua is smaller and easier to maintain than:

  - TCL

  - Perl

  - Python

  - Ruby

  - ...

- If it would stop people writing large projects in shell, that would be its biggest success :)

End of part one

- … which introduced Lua

  - Motivation

  - What is Lua?

  - How does it look like?

  - Why is Lua a good (or at least *good enough*) choice for scripting?

Questions?

BSDLUA ?

- My pet project
- "*Lua extended with common OS-level functions and constants usable for shell scripting*"
- 100% Lua-compatible, language is not modified
- Only addition: added a "stdlib.lua" script with some common code available to all scripts, etc.
- "Evolved" scripting: uses OS-level functions, not only generic OS-independed ones
- "BSD"LUA: a large part is cross-BSD

The big picture

- The goal is to make a scripting environment friendly for both C and shell programmers and enable them to be productive while also introducing a better language

- Is it a reasonable goal?

    - **Yes**: a relatively low learning curve, most power retained from shell scripts (+ ???)

    - **No**: we already have C and /bin/sh so is it worth the trouble? (+ ???)

The concept

- BSDLUA consists of roughly three parts:

  - <u>libc wrappers</u>
    Offer some of the most commonly used libc functions and syscalls

  - <u>utility / shell-like functions</u>
    Add some convenient shell-like functionalities (file testing, program execution and inspection)

  - <u>FreeBSD-specific libraries</u>
    Implement wrappers for some useful FreeBSD system libraries (libkvm, libgeom, ...?)

libc wrappers

- Same names and semantics as libc symbols, Lua-adapted types

- Divided into two namespaces: <u>posix</u> and <u>bsd</u>.

    - "posix" contains standard POSIX calls

    - "bsd" contains "everything else"

    - Not necessarily a good idea, I'm considering folding them all into "unix" namespace

- What are the most common functions? Some analysis is probably in order…

    - stat(), getenv(), open(), read(), write(), chdir()

stat(2) example

- Example: stat() call in Lua:

```
st = posix.stat("file.lua")
print(table.tostring(st))
print(st.st_size)

print(posix.stat("file.lua").st_size)
```

stat(2) example

- Example: stat() call in Lua:

```
st =  posix.stat("file.lua")
print(table.tostring(st))
print(st.st_size)

print(posix.stat("file.lua").st_size)
```

BSDLUA call

Returned value is a table

Unix / shell-like functions

- Namespace "shell"

- The idea is to make the "usual suspects" in programming easier … like "normal" scripting languages

- "`if [-r $FILE ]`" → "`shell.r_ok(file)`"

  – By analogy with access(2)

- Backticks → function `shell.ss()`

  – By analogy with system(3), with a shorter name

- Ideas welcome...

Shell-to-BSDLUA example

```
# If CONFFILE was specified at the command-line, make
# sure that it exists and is readable.


sanity_conffile() {
        if [ ! -z "${CONFFILE}" ] && [ ! -r "${CONFFILE}" ]; then
                echo -n "File does not exist "
                echo -n "or is not readable: "
                echo ${CONFFILE}
                exit 1
        fi
}
```

(from portsnap)

Shell-to-BSDLUA example

```lua
-- If CONFFILE was specified at the command-line, make
-- sure that it exists and is readable.

function sanity_conffile(conffile)
    if string.len(conffile) ~= 0 and not shell.r_ok(conffile) then
        print("File does not exist or is not readable: " .. conffile)
        posix.exit(1)
    end
end
```

Caveats

- Lua strings are 1-indexed (not 0 as in C)

- Lua "not equal" is "~=" not "!="

- String concatenation is "..", addition is "+"

    - 1 + "1" == 2
    - 1 .. "1" == "11"

- 1-line comments begin with "--"

    - Multi-line comments are strange "--[[…\n…]]--"

- It is its own language…

FreeBSD-specific libraries

- libcurses – terminal drawing

- Planned for "really soon now":

    - libgeom – FreeBSD's GEOM information

    - libkvm – kernel introspection

    - The plan: make BSDLUA good enough so that system utilities can be created in it

    - And then: create some demo utilities like a curses-based fdisk

Future plans

- Finish libc wrappers, shell utility functions

- Implement more FreeBSD-specific libraries

- Make proof-of-concept reimplementation of some FreeBSD utilities

- Start advertising it more...

End of part two

- ...which introduced BSDLUA

- Input required from the audience!

  – Overall – is it a good idea?

  – What calls / functions to implement next?

  – Style – what about posix/bsd namespaces?

  – Of course, interested developers can join!

Questions?

Why Lua could succeed where Perl failed?

- Small
- Can be maintained separately from the ports version, no conflicts with the ports version
- Much slower development of core language features (a couple of versions in a decade)
- Very easy to extend with wrappers of C functions
- Very easy to extend C programs with Lua scripting

BSDLUA for developers

- "Do not ask what you can do for Lua, ask what Lua can do for you"

- What problems or smaller issues can it solve for the developers?

- What could it offer to users?

For userland developers

- An "in-between" language
  - Access to more low-level features than shell
  - More enjoyable to program in than shell code
- Embedding C in Lua
  - To extend with functionalities, syscalls, etc.
  - To make it a *glue language* more powerful than shell code
- Embedding Lua in C
  - Add scripting functionality to C programs
- (both directions are easy)

(For kernel developers)

- ???

- Make your syscalls easily available to shell-like environment

- Solaris and Python...? ZFS utilities...?

For users and administrators

- If it's in the base system, you can depend on it being there

- Much less steep learning curve than shell code

- Convenient

- Peace on Earth, world domination of BSD and all other good things…

Possible (current) use cases for Lua in base

- Making some larger shell scripts less complex
    - Including maybe some in /etc/rc.d
- Scripting the package management system
- Scripting the installer
- Scripting other programs with complex user interactions (hastd? (u)carp? devd?)

Effectively embedding Lua support in a C program

- Create a Lua state structure

- Create a Lua environment, populate it with:

  - Constants and variables

  - Functions important to your program which you want to be used from Lua

  - Objects, tables, etc.

- Execute a Lua snippet (a file, a string) in this Lua context

- Profit !!!

The End

BSDLUA

ivoras@freebsd.org

Looking for opinions, ideas, co-developers...