Welcome to Lua 5.2

About Lua

Lua is a powerful, fast, lightweight, embeddable scripting language developed by a team at PUC-Rio, the Pontifical Catholic University of Rio de Janeiro in Brazil. Lua is free software used in many products and projects around the world.

Lua's official web site provides complete information about Lua, including an executive summary and updated documentation, especially the reference manual, which may differ slightly from the local copy distributed in this package.

Installing Lua

Lua is distributed in source form. You need to build it before using it. Building Lua should be straightforward because Lua is implemented in pure ANSI C and compiles unmodified in all known platforms that have an ANSI C compiler. Lua also compiles unmodified as C++. The instructions given below for building Lua are for Unix-like platforms. See also instructions for other systems and customization options.

If you don't have the time or the inclination to compile Lua yourself, get a binary from LuaBinaries. Try also Lua for Windows, an easy-to-use distribution of Lua that includes many useful libraries.

Building Lua

In most Unix-like platforms, simply do "`make`" with a suitable target. Here are the details.

1. Open a terminal window and move to the top-level directory, which is named `lua-5.2.2`. The Makefile there controls both the build process and the installation process.

2. Do "`make`" and see if your platform is listed. The platforms currently supported are:

   ```
   aix  ansi  bsd  freebsd  generic  linux  macosx  mingw  posix  solaris
   ```

   If your platform is listed, just do "`make xxx`", where xxx is your platform name.

   If your platform is not listed, try the closest one or posix, generic, ansi, in this order.

3. The compilation takes only a few moments and produces three files in the `src` directory: lua (the interpreter), luac (the compiler), and liblua.a (the library).

4. To check that Lua has been built correctly, do "`make test`" after building Lua. This will run the interpreter and print its version string.

If you're running Linux and get compilation errors, make sure you have installed the `readline` development package. If you get link errors after that, then try "`make linux MYLIBS=-ltermcap`".

Installing Lua

Once you have built Lua, you may want to install it in an official place in your system. In this case, do "`make install`". The official place and the way to install files are defined in the Makefile. You'll probably need the right permissions to install files.

To build and install Lua in one step, do "`make xxx install`", where xxx is your platform name.

To install Lua locally, do "`make local`". This will create a directory `install` with subdirectories `bin`, `include`, `lib`, `man`, and install Lua as listed below. To install Lua locally, but in some other directory, do "`make install INSTALL_TOP=xxx`", where xxx is your chosen directory.

```
bin:
      lua  luac
include:
      lua.h  luaconf.h  lualib.h  lauxlib.h  lua.hpp
lib:
      liblua.a
man/man1:
      lua.1  luac.1
```

These are the only directories you need for development. If you only want to run Lua programs, you only need the files in bin and man. The files in include and lib are needed for embedding Lua in C or C++ programs.

Customization

Three kinds of things can be customized by editing a file:

- Where and how to install Lua — edit `Makefile`.
- How to build Lua — edit `src/Makefile`.
- Lua features — edit `src/luaconf.h`.

You don't actually need to edit the Makefiles because you may set the relevant variables in the command line when invoking make. Nevertheless, it's probably best to edit and save the Makefiles to record the changes you need.

On the other hand, if you need to customize some Lua features, you'll need to edit `src/luaconf.h` before building and installing Lua. The edited file will be the one installed, and it will be used by any Lua clients that you build, to ensure consistency. Further customization is available to experts by editing the Lua sources.

We strongly recommend that you enable dynamic loading in `src/luaconf.h`. This is done automatically for all platforms listed above that have this feature and also for Windows.

Building Lua on other systems

If you're not using the usual Unix tools, then the instructions for building Lua depend on the compiler you use. You'll need to create projects (or whatever your compiler uses) for building the library, the interpreter, and the compiler, as follows:

```
library:
     lapi.c  lcode.c  lctype.c  ldebug.c  ldo.c  ldump.c  lfunc.c  lgc.c  llex.c  lmem.c
     lobject.c  lopcodes.c  lparser.c  lstate.c  lstring.c  ltable.c  ltm.c  lundump.c  lvm.c
     lzio.c  lauxlib.c  lbaselib.c  lbitlib.c  lcorolib.c  ldblib.c  liolib.c  lmathlib.c  loslib.c
     lstrlib.c  ltablib.c  loadlib.c  linit.c
interpreter:
     library,  lua.c
compiler:
     library,  luac.c
```

To use Lua as a library in your own programs you'll need to know how to create and use libraries with your compiler. Moreover, to dynamically load C libraries for Lua you'll need to know how to create dynamic libraries and you'll need to make sure that the Lua API functions are accessible to those dynamic libraries — but *don't* link the Lua library into each dynamic library. For Unix, we recommend that the Lua library be linked statically into the host program and its symbols exported for dynamic linking; `src/Makefile` does this for the Lua interpreter. For Windows, we recommend that the Lua library be a DLL.

As mentioned above, you may edit `src/luaconf.h` to customize some features before building Lua.

Changes since Lua 5.1

Here are the main changes introduced in Lua 5.2. The reference manual lists the incompatibilities that had to be introduced.

Main changes

- yieldable pcall and metamethods
- new lexical scheme for globals
- ephemeron tables
- new library for bitwise operations
- light C functions
- emergency garbage collector
- `goto` statement
- finalizers for tables

Here are the other changes introduced in Lua 5.2:

Language

- no more fenv for threads or functions
- tables honor the `__len` metamethod
- hex and `\z` escapes in strings
- support for hexadecimal floats
- order metamethods work for different types

- no more verification of opcode consistency
- hook event "tail return" replaced by "tail call"
- empty statement
- `break` statement may appear in the middle of a block

Libraries

- arguments for function called through `xpcall`
- optional 'mode' argument to load and loadfile (to control binary x text)
- optional 'env' argument to load and loadfile (environment for loaded chunk)
- `loadlib` may load libraries with global names (RTLD_GLOBAL)
- new function `package.searchpath`
- modules receive their paths when loaded
- optional base in `math.log`
- optional separator in `string.rep`
- `file:write` returns `file`
- closing a pipe returns exit status
- `os.exit` may close state
- new metamethods `__pairs` and `__ipairs`
- new option 'isrunning' for `collectgarbage` and `lua_gc`
- frontier patterns
- `\0` in patterns
- new option `*L` for `io.read`
- options for `io.lines`
- `debug.getlocal` can access function varargs

C API

- main thread predefined in the registry
- new functions `lua_absindex`, `lua_arith`, `lua_compare`, `lua_copy`, `lua_len`, `lua_rawgetp`, `lua_rawsetp`, `lua_upvalueid`, `lua_upvaluejoin`, `lua_version`.
- new functions `luaL_checkversion`, `luaL_setmetatable`, `luaL_testudata`, `luaL_tolstring`.
- `lua_pushstring` and `pushlstring` return string
- `nparams` and `isvararg` available in debug API
- new `lua_Unsigned`

Implementation

- max constants per function raised to 2^{26}
- generational mode for garbage collection (experimental)
- NaN trick (experimental)
- internal (immutable) version of ctypes
- simpler implementation for string buffers
- parser uses much less C-stack space (no more auto arrays)

Lua standalone interpreter

- new `-E` option to avoid environment variables
- handling of non-string error messages

License

Lua is free software distributed under the terms of the MIT license reproduced below; it may be used for any purpose, including commercial purposes, at absolutely no cost without having to ask us. The only requirement is that if you do use Lua, then you should give us credit by including the appropriate copyright notice somewhere in your product or its documentation. For details, see this.

Last update: Fri Feb 22 09:24:20 BRT 2013