

1 6 - Lua Stand-alone

Although Lua has been designed as an extension language, to be embedded in a host C program, it is also frequently used as a stand-alone language. An interpreter for Lua as a stand-alone language, called simply `lua`, is provided with the standard distribution. The stand-alone interpreter includes all standard libraries, including the debug library. Its usage is:

```
lua [options] [script [args]]
```

The options are:

- `-e stat`: executes string *stat*;
- `-l mod`: "requires" *mod*;
- `-i`: enters interactive mode after running *script*;
- `-v`: prints version information;
- `--`: stops handling options;
- `-:` executes `stdin` as a file and stops handling options.

After handling its options, `lua` runs the given *script*, passing to it the given *args* as string arguments. When called without arguments, `lua` behaves as `lua -v -i` when the standard input (`stdin`) is a terminal, and as `lua -` otherwise.

Before running any argument, the interpreter checks for an environment variable `LUA_INIT`. If its format is `@filename`, then `lua` executes the file. Otherwise, `lua` executes the string itself.

All options are handled in order, except `-i`. For instance, an invocation like

```
$ lua -e'a=1' -e 'print(a)' script.lua
```

will first set `a` to 1, then print the value of `a` (which is '1'), and finally run the file `script.lua` with no arguments. (Here `$` is the shell prompt. Your prompt may be different.)

Before starting to run the script, `lua` collects all arguments in the command line in a global table called `arg`. The script name is stored at index 0, the first argument after the script name goes to index 1, and so on. Any arguments before the script name (that is, the interpreter name plus the options) go to negative indices. For instance, in the call

```
$ lua -la b.lua t1 t2
```

the interpreter first runs the file `a.lua`, then creates a table

```
arg = { [-2] = "lua", [-1] = "-la",
        [0] = "b.lua",
        [1] = "t1", [2] = "t2" }
```

and finally runs the file `b.lua`. The script is called with `arg[1]`, `arg[2]`, as arguments; it can also access these arguments with the `vararg` expression `'...'`.

In interactive mode, if you write an incomplete statement, the interpreter waits for its completion by issuing a different prompt.

If the global variable `_PROMPT` contains a string, then its value is used as the prompt. Similarly, if the global variable `_PROMPT2` contains a string, its value is used as the secondary prompt (issued during incomplete statements). Therefore, both prompts can be changed directly on the command line or in any Lua programs by assigning to `_PROMPT`. See the next example:

```
$ lua -e"_PROMPT='myprompt> '" -i
```

(The outer pair of quotes is for the shell, the inner pair is for Lua.) Note the use of `-i` to enter interactive mode; otherwise, the program would just end silently right after the assignment to `_PROMPT`.

To allow the use of Lua as a script interpreter in Unix systems, the stand-alone interpreter skips the first line of a chunk if it starts with `#`. Therefore, Lua scripts can be made into executable programs by using `chmod +x` and the `#!` form, as in

```
#!/usr/local/bin/lua
```

(Of course, the location of the Lua interpreter may be different in your machine. If `lua` is in your `PATH`, then

```
#!/usr/bin/env lua
```

is a more portable solution.)