lfe(1)

Robert Virding

2008-2020

NAME

lfe - Lisp Flavoured Erlang (LFE) shell

SYNOPSIS

`lfe` is a simple LFE repl (read-eval-print loop) in which you can enter sexprs which then are evaluated and the value printed. You can also define local functions and macros as well as set variables. It can read commands either from the standard input or from a file.

The LFE repl is implemented in the module `lfe_shell`.

BUILT-IN SHELL FUNCTIONS

These are defined as normal functions and macros and can be called from anywhere in the shell. They can even be redefined. They can also be explicitly called (: lfe_shell …).

(c File [Options])

Compile and load an LFE file. Assumes default extension `.lfe`.

(: c Command Arg …)

(c:Command Arg …)

All the commands in the standard Erlang shell can be reached in this way.

(cd Dir)

Change the working directory.

(clear)

Clear the REPL output.

(ec File [Options])

Compile and load an Erlang file.

(ep Expr [Depth])

(epp Expr [Depth])

Print/prettyprint a value in Erlang form to either the specified depth or if no value is given the full depth.

(flush)

Flush any messages sent to the shell.

(h)

(help)

Print usage info.

(h Mod)

(h Mod Mac)

(h Mod Fun Arity)

Print out help information of a module/macro/function.

(i [(list Pid …)])

Print information about a list of pids. If no list is given then print information about currently running processes in the system.

(i x y z)

Print information about the about #Pid<x.y.z>

(l Module …)

Load modules.

(ls)

(ls dir)

List files in a directory. If no directory is given then list files in the current directory.

(m [Module …])

Print out module information. If no modules are given then print information about all modules.

(memory)

**(memory type)

Returns the memory allocation information. If a type or a list of types is given then only for those types.

(p Expr [Depth])

(pp Expr [Depth])

Print/prettyprint a value to either the specified depth or if no value is given the full depth.

(pid x y z)

Create a pid from x, y, z.

(pwd)

Print the current working directory.

(q)

Quit - shorthand for `init:stop/0`.

(regs)

Print information about the registered processes in the system.

(nregs)

Print information about all the registered processes in the system.

(uptime)

Print the node uptime.

BUILT-IN SHELL COMMANDS

These are special forms which are only recognised at the top-level in shell input. The cannot be redefined.

(reset-environment)

Resets the environment to its initial state. This will clear all variables, functions and macros that have been set.

(run File)

Execute all the shell commands in File. All defined variables, functions and macros will be saved in the environment if there are no errors.

(set Pattern Expr)

(set Pattern (when Guard) Expr)

Evaluate Expr and match the result with Pattern binding variables in it. These variables can then be used in the shell and also rebound in another set.

(slurp File)

Slurp in a source LFE file and makes all functions and macros defined in the file available in the shell. Only one file can be slurped at a time and slurping a new file basically does an unslurp first.

(unslurp)

Revert back to the state before the last slurp removing all function and macro definitions both in the slurped file and defined in the shell since then.

SHELL FUNCTIONS AND MACROS

Functions and macros can be defined in the shell. These will only be local to the shell and cannot be called from modules. The forms are the standard forms for defining functions and macros.

(defun Fun …)

Define a function in the shell.

(defmacro Macro …)

Define a macro in the shell.

BUILT-IN SHELL VARIABLES

+, ++, +++

The three previous expressions input.

, **, **

The values of the previous three expressions.

-

The current expression input.

SHELL ENVIRONMENT

The shell maintains an environment of local function and macro definitions, and variable

bindings. The environment can be accessed using the built-in shell variable $ENV. This can be useful when calling functions like macroexpand and macro-function which unless an explicit environment is given will only search the default environment.

STARTING THE LFE SHELL

After installing the best way is probably to start Erlang directly running the LFE shell with:

```
lfe [flags]
```

From a normal Erlang shell the best way to start the shell is by calling:

```
17> lfe_shell:server().
```

Giving the user switch commands:

```
--> s lfe_shell
--> c
```

will create a job running the LFE shell and connect to it. This also works when starting a remote shell.

Flags that LFE recognizes include the following:

- `-nobanner` - starts LFE without showing the banner
- `-h` or `--help` - provides command line usage help
- `-e` or `-eval` - evaluates a given sexpr in a string
- `-prompt` - users may supply a value here to override the default `lfe>` prompt; note that `-prompt classic` will set the prompt to the original `>` and `-prompt` with no associated value will cause no prompt to be displayed at all. These also work when node names are provided (with either `-sname` or `-name`). Furthermore, users may override the default formatting of node names in prompts by providing a prompt value containing the string `~node` (which will be substituted with the actual name of the node).

There can be multiple string expressions to be evaluated; each one must be prefixed with an `-e` or `-eval`. String expressions are run in the LFE repl so shell commands and functions are allowed. They are all run in the same invocation of the repl so:

```
$ lfe -e "(set aaa 42)" -e "(set bbb 84)" -e "(pp (tuple aaa bbb))"
#(42 84)
```

If there are string expressions then the LFE repl will `not` be run.

RUNNING LFE SHELL SCRIPTS

The LFE shell can also be directly called to run LFE shell scripts with:

```
lfe [flags] file [args]
```

This will start the shell, run a script with LFE shell commands and then terminate the shell. The following built-in variables are also bound:

script-name

The name of the script file as a string.

script-args

A list of the arguments to the script as strings. If no arguments have been given then this will be an empty list.

Note that if there are any string expressions to be evaluated then these must come before the name of the script file and its arguments. These expressions will be evaluated before the script and the script will use the environment from the string expressions.

It is possible to run both string expressions and an LFE shell script and they are then run in the same LFE repl.

SEE ALSO

lfescript(1), **lfe_guide(7) lfe_doc(3)**