How to Install and Use SBCL (Lisp) with Emacs under Linux

Norman Carver

January 30, 2010

1 Installing SBCL and SLIME

One of the best free versions of Common Lisp (for Linux systems) is *Steel Bank Common Lisp* (SBCL). While SBCL can be run from the command line (see Section 4), development is typically done using Emacs as the basis of an IDE. SLIME (*The Superior Lisp Interaction Mode for Emacs*) provides the glue that integrates Emacs and SBCL. So most people will want to install SBCL and SLIME. (We assume that a recent Emacs is already installed on your system....)

1.1 Downloading SBCL and SLIME

The first thing that has to be done is to download SBCL Lisp and the SLIME package that is the interface between Emacs and Lisp:

1. Get SBCL binary installation tar file from:
   `http://www.sbcl.org/platform-table.html`
   The tar file will be something like: sbcl-1.0.34-x86-linux-binary.tar.bz2
2. Get SLIME tar file from:
   `http://common-lisp.net/project/slime/snapshots/slime-current.tgz`

1.2 Installing SBCL

Once the tar files have been downloaded, you need to install SBCL.

Untar the downloaded SBCL file, e.g, like this:
```
  tar xjf sbcl-1.0.34-x86-linux-binary.tar.bz2
```
This will produce a subdirectory like:
```
  sbcl-1.0.34-x86-linux
```

This directory contains an installation script named "`install.sh`".

Installation now depends on whether one is root vs. not user:

- If this is a machine you can be root on:
  - The default installation goes under `/usr/local`:
    as root, cd into the extracted sbcl directory, and run: "`sh install.sh`"
  - The executable will be `/usr/local/bin/sbcl` so `/usr/local/bin` must be in users search PATHs in order for them to be able to invoke the command `sbcl`.

  - You can install to another directory by running:
    `INSTALL_ROOT=/other/dir sh install.sh`

- If this is a machine you cannot be root on:

  You will have to install to your home directory:
  cd into the extracted sbcl directory, and run:
  `INSTALL_ROOT=/home/username/sbcl sh install.sh`
  (Replacing username with your username on the system, or choose another subdirectory under your home directory.)

You can now delete the extracted sbcl subdirectory and tar file.

1.3 Installing SLIME

Extract the downloaded tar file to the desired directory:

- Either move the tar file to the desired directory and extract like:
  `tar xzf slime-current.tgz`
- Or, leave it where is is and use the "`-C /target_dir`" tar option to extract to a desired directory:
  `tar xzf slime-current.tgz -C /desired/dir`

The exacted slime directory can be put anywhere that it is readable by users, but its path must be known to all users for them to be able to use it, so it should be put someplace standard:

If this is a machine you can be root on, the standard parent directory is:
`/usr/share/emacs/site-lisp`
If this is a machine you cannot be root on, it can be placed in the home directory.

Since users must know the path of the installed SLIME directory, it is important to rename it to "slime" (exactly what directory name the tar file extracts to can vary, for example recently recent-slime.tgz files produce a directory like "slime-2010-01-26", which is obviously not appropriate for users).

You can now delete the downloaded SLIME tar file.

1.4 Setting up a User's .emacs File

In order to invoke SLIME/SBCL inside of Emacs, each user needs to have a `~/.emacs file`, and that file must contain:

```
(setq inferior-lisp-program "/usr/local/bin/sbcl")
(add-to-list 'load-path "/usr/share/emacs/site-lisp/slime")
(require 'slime)
(slime-setup '(slime-repl))
```

(Be sure to modify the paths to the sbcl binary and the slime directory depending on where they are installed!)

There are various optional packages for SLIME, which are loaded via the `slime-setup` form. Several popular ones can be loaded by specifying "`slime-fancy`" instead of "`slime-repl`" which is probably desirable.

2 Using SBCL/SLIME in Emacs

2.1 Starting SBCL/SLIME in Emacs

Once you have Emacs running, if you have installed and setup SBCL and SLIME as described in the previous section, you can start SLIME (which runs SBCL as a subprocess) by doing:
`M-x slime`
(`M-x` means Meta-x: hold down Meta/Alt key and type the x key)
Note that `M-x` allows you to enter Emacs commands: when you type `M-x`, it appears in the mini-buffer window at the bottom of the Emacs window, and waits for you to enter the commane to run (e.g., "slime").

Once SLIME starts, you end up in a window/pane called "`slime-repl`". REPL stands for *read-eval-print-loop*, also known as a "*Lisp Listener.*" This is an *interactive* Lisp environment, where you can type Lisp forms, and have them immediately evaluated/run. By default, the prompt is "`CL-USER> `". You can try out your Lisp by evaluating a simple form like (+ 1 2 3), producing:

```
CL-USER> (+ 1 2 3)
6
CL-USER>
```

Once SLIME/SBCL are running, if you open a new/existing file with a .lisp extension, Emacs will automatically use the slime/lisp *mode.*

2.2 Developing Using SLIME/SBCL

In the REPL (`slime-repl`) window, you will have both SLIME and REPL *menus*, while in a .lisp file window you will have only a SLIME menu. From these menus, you can take actions such as compiling an individual function definition in a .lisp file, compiling an entire file, checking parens, getting documentation strings, etc. There are also a set of *keybindings* (hot keys) that are defined for these windows (see below). Lisp functions will now also have font coloring, etc., applied.

The typical pattern of developing Lisp code is to divide the main Emacs window into two panes (e.g., using "C-x 3"), and then use one pane to display the Lisp Listener and the other to display the .lisp source file being edited. As each new function is defined in the source file, it can immediately be compiled to check for syntax errors, and once it compiles correctly, the REPL/Listener pane can be used to test it. Lisp's interactive environment and ability to compile code function by function supports a very *incremental and interactive style of coding*, which is simply not possible in most languages (where you must compile entire files, write driver programs to test each function, etc.).

Note that the syntax and arguments for a defined function will be given in the *mini-buffer line* at the bottom of the Emacs window as soon as you hit < *space* > afer entering a function name.

2.3 SLIME Keybindings/Shortcuts

SLIME loads a large number of Lisp-specific functions into Emacs, many of which can be invoked using predefined keybindings (shortcuts). In what follows, notation like "C-c" means control-c: hold down the Control key and type the c key. Notation like "M-e" means meta-e: hold down the Meta key (usually bound to the Alt key, though need not be in X11), and type the e key.

Selected Keybindings in the REPL Window:

< *return* >	Evaluate the current input if complete, else open a new line and indent.
C-< *return* >	Close off all open parens and evaluate.
C-c C-p	Goto previous prompt (then RETURN copies to current prompt).
C-c C-n	Goto next prompt (then RETURN copies to current prompt).
C-< *up* >	Bring up previous entry in history list.
C-< *down* >	Bring up next entry in history list.
C-a	Goto beginning of entered form (after prompt).
C-c C-u	Remove input from point back to prompt.
C-c M-o	Clear output from REPL buffer.
C-c C-c	Interrupt Lisp.

Selected Keybindings in Editor Windows:

< *tab* >	Indent the current Lisp code line.
C-c C-]	Close all parens here.
C-c C-c	Compile the top-level form at point.
C-c C-k	Compile and load the current buffer's source file.
C-c M-k	Compile (but don't load) the current buffer's source file.
C-c C-l	Load a Lisp file.
C-x C-e	Evaluate the expression before point.
C-M-x	Evaluate current top-level form.
C-c M-e	Evaluate last expression in output buffer.
M-n	Move to next compiler note (in source file).
M-p	Move to previous compiler note (in source file).
C-c M-c	Clear compiler notes (in source file).
< *space* >	Insert space, but lookup and display function argument list (in mini-buffer).
M-.	Go to the definition of the symbol at point.
M-,	Backtrack to previous M-. position.
M-*	same
C-c C-d d	Describe the symbol at point.
C-c C-f	Describe symbol (e.g., funtion).
C-c C-d a	Apropos search. Search Lisp symbol names for a substring match.
C-c < *tab* >	Complete the symbol at point.
M-< *tab* >	same
C-c C-s	Looks up and inserts the arguments for the function at point.
C-c C-m	Macroexpand the expression at point once.
C-c M-m	Fully macroexpand the expression at point.
C-c <	List the callers of function.
C-c >	List the callees of function.
C-c I	Invoke the Inspector.
C-c M-d	Disassemble function compilation (show assembly version).
C-c C-t	Toggle tracing of the function at point.
C-c C-b	Interrupt Lisp.

There are also a set of special REPL commands called *shortcuts* to do things like change the REPL's current directory. These commands are invoked by first typing a `,` (comma), which prompts for the command in the mini-buffer. Use the command "`help`" to get a list.

See the SLIME manual for complete information on SLIME functionality.

3 Configuring SBCL

3.1 SBCL Initialization File

By default, SBCL reads and evaluates the file `~/.sbclrc` when it starts. This file can be used to always load user-specific utilities and the like. Many Lisp users make use of personal sets of utility functions/macros so they have a customized version of Lisp. Such files should be loaded via `~/.sbclrc`.

3.2 SBCL Compiler Settings

One issue with SBCL is that the compiler is *extremely verbose* by default. It gives lots of notes about making detailed type declarations to be able to achieve optimum speed. This can be changed, so that *only error messages* are emitted, by evaluating the following form once SBCL has started:

```
  (declaim (sb-ext:muffle-conditions sb-ext:compiler-note))
```

To return to producing detailed compiler notes, use:

```
  (declaim (sb-ext:unmuffle-conditions sb-ext:compiler-note))
```

4 Command Line SBCL

SBCL does not have to be run from inside of Emacs via SLIME. If one simply wants to run some function/file, SBCL can be invoked from the *command line* with the command "`sbcl`". This results in a Lisp Listener (REPL) process, whose prompt is "`* `". This approach does not allow for command line editing or command history, so is very limited. However, if one just wants to load a Lisp file and run some function, one could do:

```
> sbcl
* (load "desired-file")
* (function-defined-in-file arg1...)
```

You can quit the Listener using C-d (control-d) or the function call "`(quit)`".

5 SBCL and Scripting

SBCL can be used with Linux scripting in several ways:

1. One can write a standard shell script that starts SBCL and has it load a particular file that runs functions; the filename can even be entered from the command line.

2. One can write a standard shell script that starts SBCL and has it run (evaluate) Lisp code, including code entered from the command line.
3. One can write a script to start SBCL directly, and evaluate Lisp forms (this requires some special code in the `~/.sbclrc` file—see the SBCL manual).

An important point to note about scripting and .lisp files is that unlike most other languages, .lisp files can contain Lisp *expressions* as well as function definitions. So a file can call builtin Lisp functions, print out information, or even define some new functions and then run those. This means that the ability to *load* a .lisp file actually gives you the ability to *run* arbitrary Lisp code!

One thing to note from many of the script examples below is that one must often explicitly call the "`quit`" function in order to terminate the SBCL Listener.

Example .lisp file `program.lisp` for use in script examples below, prints out two lines and quits:

```
(write-line "Hello, World!")
(format t "The sum of 1+2+3 = ~D~%" (+ 1 2 3))
(quit)
```

Example Bash script that loads/runs the above Lisp file:

```
#!/bin/bash
/usr/local/bin/sbcl --noinform --load "/home/user/program"
```

Example Bash script that loads a Lisp filename entered from the command line:

```
#!/bin/bash
/usr/local/bin/sbcl --noinform --load "$1"
```

Example Bash script that evaluates/runs the same code as in `program.lisp` above:

```
#!/bin/bash
/usr/local/bin/sbcl --noinform\
  --eval '(write-line "Hello, World!")'\
  --eval '(format t "The sum of 1+2+3 = ~D~%" (+ 1 2 3))'\
  --eval '(quit)'
```

Example Bash script that evaluates/runs code entered from the command line:

```
#!/bin/bash
/usr/local/bin/sbcl --noinform\
  --eval "$1"\
  --eval '(quit)'
```

If the above script were called `evallisp`, you could run above code with a call like:
```
./evallisp '(progn (write-line "Hello, World!") (format t "The sum of 1+2+3 = ~D~%" (+ 1 2 3)))'
```

Example script to directly start SBCL and evaluate Lisp code as in `program.lisp` above:

```
#!/usr/local/bin/sbcl --noinform
(write-line "Hello, World!")
(format t "The sum of 1+2+3 = ~D~%" (+ 1 2 3))
```

If we have put the above script into an executable file named "`hello`" we would invoke/run that file from the Linux command line in the normal way:

```
./hello
```

6 The SBCL/SLIME Debugger

When a Lisp *runtime errors occurs*, the *debugger* is invoked. SLIME has a special debugger interface/window named, SLDB. The debugger produces output like:

```
debugger invoked on a TYPE-ERROR in thread 11184:
     The value 3 is not of type LIST.

   You can type HELP for debugger help, or (SB-EXT:QUIT) to exit from SBCL.

   restarts (invokable by number or by possibly-abbreviated name):
     0: [ABORT   ] Reduce debugger level (leaving debugger, returning to toplevel).
     1: [TOPLEVEL] Restart at toplevel READ/EVAL/PRINT loop.
   (CAR 1 3)
   0]
```

Information about the error is given in several places:

- The class of error is: `TYPE-ERROR`
- The specific error is: `The value 3 is not of type LIST.`
- The form that produced the error is: `(CAR 1 3)`

This last line is a prompt ("`0]`"), as the debugger is basically a special REPL. You can enter any of the listed integers for particular restarts. In the command line SBCL, there are also commands such as "`toplevel`" which throw you out of the debugger loop, and C-d (control-d) will throw you up one level in the debugger (and possibly out). SLIME opens a special debugger window/pane, and allows a number of single letter commands such as "`a`" for abort, etc.

You can use the SBCL debugger to examine stack frames, restart from various frames, examine and even change variable/argument values before restarting, etc. SLDB provides a set of keybindings to facilitate use of the debugger, including:

a	Abort restart.
q	Quit restart.
c	Continue restart.
n	Next stack frame.
p	Previous stack frame.
t	Toggle frame variable display.
v	View frame source code.
e	Evaluate expression in context of frame.
D	Disassemble frame function.