CLASH: CLisp As SHell

Summary

Thus document describes how you can make CLISP your login shell on GNU/Linux.

You can probably use these instructions on other UNIX systems too.

DISCLAIMER

- Use at your own risk.
- Exercise care.
- Backup everything you change on your system.
- If anything goes wrong, do NOT blame us!

Author

Peter Wood

last modified: 2001-06-01

Notes marked NB from SDS were added by Sam Steingold.

Step 1: Compile CLISP

On Linux, you need to have built your CLISP with the `--with-module=bindings/glibc` option, and you should make sure that you have it compiled with readline. See the CLISP build instructions and Makefile.

Step 2: Make CLISP a valid shell

Put `/usr/bin/clisp` (or `/usr/local/bin/clisp`) in `/etc/shells` so it looks (something) like this:

```
# /etc/shells: valid login shells
#/bin/ash
/bin/bash
/bin/sh
/usr/bin/clisp
```

Step 3: Modify your PATH to be able to run X

You can set up your `$PATH` in `/etc/login.defs` by modifying the entry for `ENV_PATH` to look (something) like this:

```
ENV_PATH        PATH=/bin:/usr/bin:/usr/local/bin:/usr/X11R6/bin
```

Once you have your CLISP shell set up, you can control the environment variables (on Linux) via the glibc bindings (see `${clisp-src}/modules/bindings/glibc/linux.lisp`): setenv, putenv, getenv, etc.

NB from SDS

CLISP has a setfable built-in ext:getenv, e.g., `(setf (ext:getenv "foo") "bar")`.

Step 4: start X

The `startx` command is a shell script, so we cannot use it. As a temporary fix, I have defunned thus:

```
(defun startx ()
  (execute "/usr/X11R6/bin/xinit"))
```

Which works for me. Although I haven't investigated all the consequences of not going via shell's startx. Also I do not use GNOME or KDE which may complicate things for you if you do. I use fvwm2 and have this in my `.xinitrc`:

```
exec fvwm2
```

Once CLISP is set up, you will also be able to type:

```
> #[xinit]
```

from Lisp to start X.

NB from SDS

You should be able to run a shell script using the CLISP built-in function ext:shell.

Step 5: set up the reader macro

For conveniently running external programs (i.e., I do not want to have to type `(run-program "ls" :arguments '("-lh"))` every time) I have set up a read macro. Put the following in `somefile.lisp`:

```
(set-macro-character #\] (get-macro-character #\)))
(set-dispatch-macro-character #\# #\[
  (lambda (stream char1 char2)
    (declare (ignore char1 char2))
    (setf (readtable-case *readtable*) :preserve)
    (unwind-protect
         (let ((command-line (read-delimited-list #\] stream t)))
           (list 'ext:run-program (princ-to-string (car command-line))
                 :arguments `',(mapcar #'princ-to-string (rest command-line))))
      (setf (readtable-case *readtable*) :upcase))))
```

NB from SDS

I heavily modified this macro.

Note that ext:run-program will not execute shell scripts, while ext:shell will invoke a shell for each command - these are the trade-offs.

Step 6: set up the readline shortcut

It's also a PITA to have to type "#[" and "]" everytime you want to run a command, so add the following in your $HOME/.inputrc:

```
#for clash: prints the square brackets to run an external command
"\ec": "#[]\C-b"
```

When you type ESC-c (or META-c) readline will print "#[]" to the console and put the cursor inside the brackets. If you are running CLISP as your shell, and do this, you will then be able to run programs (more or less) normally. You will need to escape the dot in any filenames that start with a dot, e.g.,

```
> #[cat \.xinitrc]
```

and sometimes (eek) a colon.

Step 7: try it out

Do NOT modify your /etc/passwd yet!

Start up a clisp with

```
$ clisp -K full -i somefile.fas
```

Hit ESC c, type ls -l, hit Enter and see if it works.

You do not have command line completion for external programs inside the read-macro. So hitting tab will just give you a list of all CLISP's possibilties, which is not much use here. However, ext:run-program (which is part of what's hiding behind the read macro) does search your path for executables so you don't have to do #[/bin/ls -l]

NB from SDS

There are CLISP built-ins like ext:dir and ext:cd, and most other shell built-ins can be implemented in CLISP pretty easily.

Step 8: Dump a memory image

If you are happy, and think you can live with this setup then you must do the following:

1. Dump a memory image from a CLISP with the read macro and any convenience stuff loaded, as described in impnotes.
2. Make a backup of /usr/[local]/lib/clisp. (wherever yours is installed)
3. Rename the "base" directory in /usr/lib/clisp to "orig-base".
4. Now rename the "full" directory to "base".
5. Replace the memory image in your new "base" directory with the one you dumped.

Step 9: Dive in!

Change your /etc/passwd to reflect your new shell.

Light 13 candles in a circle round your Linux box, Sacrifice a white rooster, invoke Saint IGNUcious and ...

Login again.

If you start pining for Bash, you can run:

```
> #[bash]
```

You will have to explicitly source all your configuration files (.profile etc)

TODO

1. Get indentation on the command line.
2. Do command redirection.
3. Get name completion for arguments which are filenames.

Have fun,

Peter