Lisp in the middle: using Lisp to manage a Linux system

Michael Raskin

TU Munich

2021-05-04

Overview

- What: random GNU/Linux system management
  ... beyond normal management: isolate some things
- With what: Common Lisp at core, mix CL/Bash around
- How: Lazily (lazy programmer, not lazy evaluation)
- Why:
  - Too many small changes wanted
  - Too many upstream changes *not* wanted
  - Common Lisp code keeps working the same
  - Hards part of upstream code not useful to me

Overview

- What: random GNU/Linux system management
  ... beyond normal management: isolate some things
- With what: Common Lisp at core, mix CL/Bash around
- How: Lazily (lazy programmer, not lazy evaluation)
- Why:
  - Too many small changes wanted
  - Too many upstream changes *not* wanted
  - Common Lisp code keeps working the same
  - Hards part of upstream code not useful to me

Overview

- What: random GNU/Linux system management
  … beyond normal management: isolate some things
- With what: Common Lisp at core, mix CL/Bash around
- How: Lazily (lazy programmer, not lazy evaluation)
- Why:
  - Too many small changes wanted
  - Too many upstream changes *not* wanted
  - Common Lisp code keeps working the same
  - Hards part of upstream code not useful to me

Overview

- What: random GNU/Linux system management
  ... beyond normal management: isolate some things
- With what: Common Lisp at core, mix CL/Bash around
- How: Lazily (lazy programmer, not lazy evaluation)
- Why:
  - Too many small changes wanted
  - Too many upstream changes *not* wanted
  - Common Lisp code keeps working the same
  - Hards part of upstream code not useful to me

Overview

- What: random GNU/Linux system management
  ... beyond normal management: isolate some things
- With what: Common Lisp at core, mix CL/Bash around
- How: Lazily (lazy programmer, not lazy evaluation)
- Why:
  - Too many small changes wanted
  - Too many upstream changes *not* wanted
  - Common Lisp code keeps working the same
  - Hards part of upstream code not useful to me

Overview

- What: random GNU/Linux system management
  ... beyond normal management: isolate some things
- With what: Common Lisp at core, mix CL/Bash around
- How: Lazily (lazy programmer, not lazy evaluation)
- Why:
  - Too many small changes wanted
  - Too many upstream changes *not* wanted
  - Common Lisp code keeps working the same
  - Hards part of upstream code not useful to me

Overview

- What: random GNU/Linux system management
  ... beyond normal management: isolate some things
- With what: Common Lisp at core, mix CL/Bash around
- How: Lazily (lazy programmer, not lazy evaluation)
- Why:
  - Too many small changes wanted
  - Too many upstream changes *not* wanted
  - Common Lisp code keeps working the same
  - Hards part of upstream code not useful to me

Talk overview

- How it looks like
  (not much to have a look!)
- What happens behind the curtain
- What else is there / should be there but isn't
- Lessons learned

Talk overview

- How it looks like
  (not much to have a look!)
- What happens behind the curtain
- What else is there / should be there but isn't
- Lessons learned

Talk overview

- How it looks like
  (not much to have a look!)
- What happens behind the curtain
- What else is there / should be there but isn't
- Lessons learned

Talk overview

- How it looks like

  (not much to have a look!)
- What happens behind the curtain
- What else is there / should be there but isn't
- Lessons learned

Demo

Brightness/CPU frequency change

Demo

Isolated Firefox

How it works? Invocation

User invokes in unprivileged Lisp image:

```
(ask-with-auth (:presence t)
  `(set-cpu-frequency "min")
  `(set-brightness 1))
```

How it works? Translation

Request translated:

```
("LIST"
   ("SET-CPU-FREQUENCY" "min")
   ("SET-BRIGHTNESS" 1))
```

- No symbols (except `NIL`), only strings
- Looks the same in all Lisps

How it works? Authentication

Client process sends

```
("REQUEST-UID-AUTH" "raskin")
```

Server sends the path to an access controlled file
Client process reads the file
Client process sends:

```
("WITH-UID-AUTH" "JDPBTP56Y3LALB6FMA54"
  ("WITH-PRESENCE-AUTH" "T"
    ("LIST" ("SET-CPU-FREQUENCY" "min")
             ("SET-BRIGHTNESS" 1))
    15))
```

How it works? Authentication

Client process sends

```
("REQUEST-UID-AUTH" "raskin")
```

Server sends the path to an access controlled file
Client process reads the file
Client process sends:

```
("WITH-UID-AUTH" "JDPBTP56Y3LALB6FMA54"
  ("WITH-PRESENCE-AUTH" "T"
    ("LIST" ("SET-CPU-FREQUENCY" "min")
             ("SET-BRIGHTNESS" 1))
    15))
```

How it works? Authentication

Client process sends

```
("REQUEST-UID-AUTH" "raskin")
```

Server sends the path to an access controlled file
Client process reads the file
Client process sends:

```
("WITH-UID-AUTH" "JDPBTP56Y3LALB6FMA54"
  ("WITH-PRESENCE-AUTH" "T"
    ("LIST" ("SET-CPU-FREQUENCY" "min")
             ("SET-BRIGHTNESS" 1))
    15))
```

How it works? Execution

Server interns names into special package
Physically present user asked,
then the following runs:

```
(socket-command-server-commands::set-cpu-frequency
  context "min")
```

(`context`: information about user confirmation etc.)

- Manual handling of keyword arguments — harder to forget filtering...

How it works? Execution

Server interns names into special package
Physically present user asked,
then the following runs:

```
(socket-command-server-commands::set-cpu-frequency
  context "min")
```

(`context`: information about user confirmation etc.)

- Manual handling of keyword arguments — harder to forget filtering…

Updates

- Persistent state should be predictable
  - Normal approaches: global mutable state, lots of it
  - I want something `cons`-like, not `(setf (elt x 7))`!
    - Nix
    - Guix
  - Guix is in Guile, but using Nix because of package coverage
- Updating the policy code: exit daemon and restart
  - Runtime state: in SQLite, on RAM FS
  - No expectation of perfection of policy code, crashes are fine
  - Can integrate multiple daemons if desired

Updates

- Persistent state should be predictable
    - Normal approaches: global mutable state, lots of it
    - I want something `cons`-like, not `(setf (elt x 7))`!
        - Nix
        - Guix
    - Guix is in Guile, but using Nix because of package coverage
- Updating the policy code: exit daemon and restart
    - Runtime state: in SQLite, on RAM FS
    - No expectation of perfection of policy code, crashes are fine
    - Can integrate multiple daemons if desired

Updates

- Persistent state should be predictable
  - Normal approaches: global mutable state, lots of it
  - I want something `cons`-like, not `(setf (elt x 7))`!
    - Nix
    - Guix
  - Guix is in Guile, but using Nix because of package coverage
- Updating the policy code: exit daemon and restart
  - Runtime state: in SQLite, on RAM FS
  - No expectation of perfection of policy code, crashes are fine
  - Can integrate multiple daemons if desired

Updates

- Persistent state should be predictable
  - Normal approaches: global mutable state, lots of it
  - I want something `cons`-like, not `(setf (elt x 7))`!
    - Nix
    - Guix
  - Guix is in Guile, but using Nix because of package coverage
- Updating the policy code: exit daemon and restart
  - Runtime state: in SQLite, on RAM FS
  - No expectation of perfection of policy code, crashes are fine
  - Can integrate multiple daemons if desired

Updates

- Persistent state should be predictable
  - Normal approaches: global mutable state, lots of it
  - I want something `cons`-like, not `(setf (elt x 7))`!
    - Nix
    - Guix
  - Guix is in Guile, but using Nix because of package coverage
- Updating the policy code: exit daemon and restart
  - Runtime state: in SQLite, on RAM FS
  - No expectation of perfection of policy code, crashes are fine
  - Can integrate multiple daemons if desired

What else is there

- Request authorisation by entering root password
- Generic password entry in dedicated VT
- Isolating interactive CLI programs with terminal forwarded
- `shell-with-mounted-devices`

What else is there

- Request authorisation by entering root password
- Generic password entry in dedicated VT
- Isolating interactive CLI programs with terminal forwarded
- `shell-with-mounted-devices`

What else is there

- Request authorisation by entering root password
- Generic password entry in dedicated VT
- Isolating interactive CLI programs with terminal forwarded
- `shell-with-mounted-devices`

What else is there

- Request authorisation by entering root password
- Generic password entry in dedicated VT
- Isolating interactive CLI programs with terminal forwarded
- `shell-with-mounted-devices`

What else should be there but isn't

- Prevent DoS via too many user confirmations
- More options to prove client process details
- Contribute other-side-of-the-socket system query support to IOLIB
- JSON/XML protocol for interaction with non-Lisps?
- Multi-device support
- Integration with service supervision (Shepherd?)

What else should be there but isn't

- Prevent DoS via too many user confirmations
- More options to prove client process details
- Contribute other-side-of-the-socket system query support to IOLIB
- JSON/XML protocol for interaction with non-Lisps?
- Multi-device support
- Integration with service supervision (Shepherd?)

What else should be there but isn't

- Prevent DoS via too many user confirmations
- More options to prove client process details
- Contribute other-side-of-the-socket system query support to IOLIB
- JSON/XML protocol for interaction with non-Lisps?
- Multi-device support
- Integration with service supervision (Shepherd?)

What else should be there but isn't

- Prevent DoS via too many user confirmations
- More options to prove client process details
- Contribute other-side-of-the-socket system query support to IOLIB
- JSON/XML protocol for interaction with non-Lisps?
- Multi-device support
- Integration with service supervision (Shepherd?)

What else should be there but isn't

- Prevent DoS via too many user confirmations
- More options to prove client process details
- Contribute other-side-of-the-socket system query support to IOLIB
- JSON/XML protocol for interaction with non-Lisps?
- Multi-device support
- Integration with service supervision (Shepherd?)

Lessons learned

- Not too hard to move system management to Lisp!
  And to grow scope as desired
- There are things I still prefer to do in POSIX Shell
- SBCL REPL doesn't promise to be shell to start interactive programs
- Trying to write idiomatic Lisp highlights missing idioms on UIs
- My comfort now relies on even more idiosyncratic tools...

Lessons learned

- Not too hard to move system management to Lisp!
  And to grow scope as desired
- There are things I still prefer to do in POSIX Shell
- SBCL REPL doesn't promise to be shell to start interactive programs
- Trying to write idiomatic Lisp highlights missing idioms on UIs
- My comfort now relies on even more idiosyncratic tools...

Lessons learned

- Not too hard to move system management to Lisp!
  And to grow scope as desired
- There are things I still prefer to do in POSIX Shell
- SBCL REPL doesn't promise to be shell to start interactive programs
- Trying to write idiomatic Lisp highlights missing idioms on UIs
- My comfort now relies on even more idiosyncratic tools...

Lessons learned

- Not too hard to move system management to Lisp!

  And to grow scope as desired
- There are things I still prefer to do in POSIX Shell
- SBCL REPL doesn't promise to be shell to start interactive programs
- Trying to write idiomatic Lisp highlights missing idioms on UIs
- My comfort now relies on even more idiosyncratic tools...

Thanks!

Thanks for your attention!

Questions?

https://github.com/7c6f434c/lang-os/