Boot Can Build It

Alan Dipert
@alandipert

Adzerk

Micha Niskin
@michaniskin

Why a new build tool?

- Builds are processes, not specifications
- Most tools oriented around configuration instead of programming
- **We're programmers, we need to be able to program builds!**

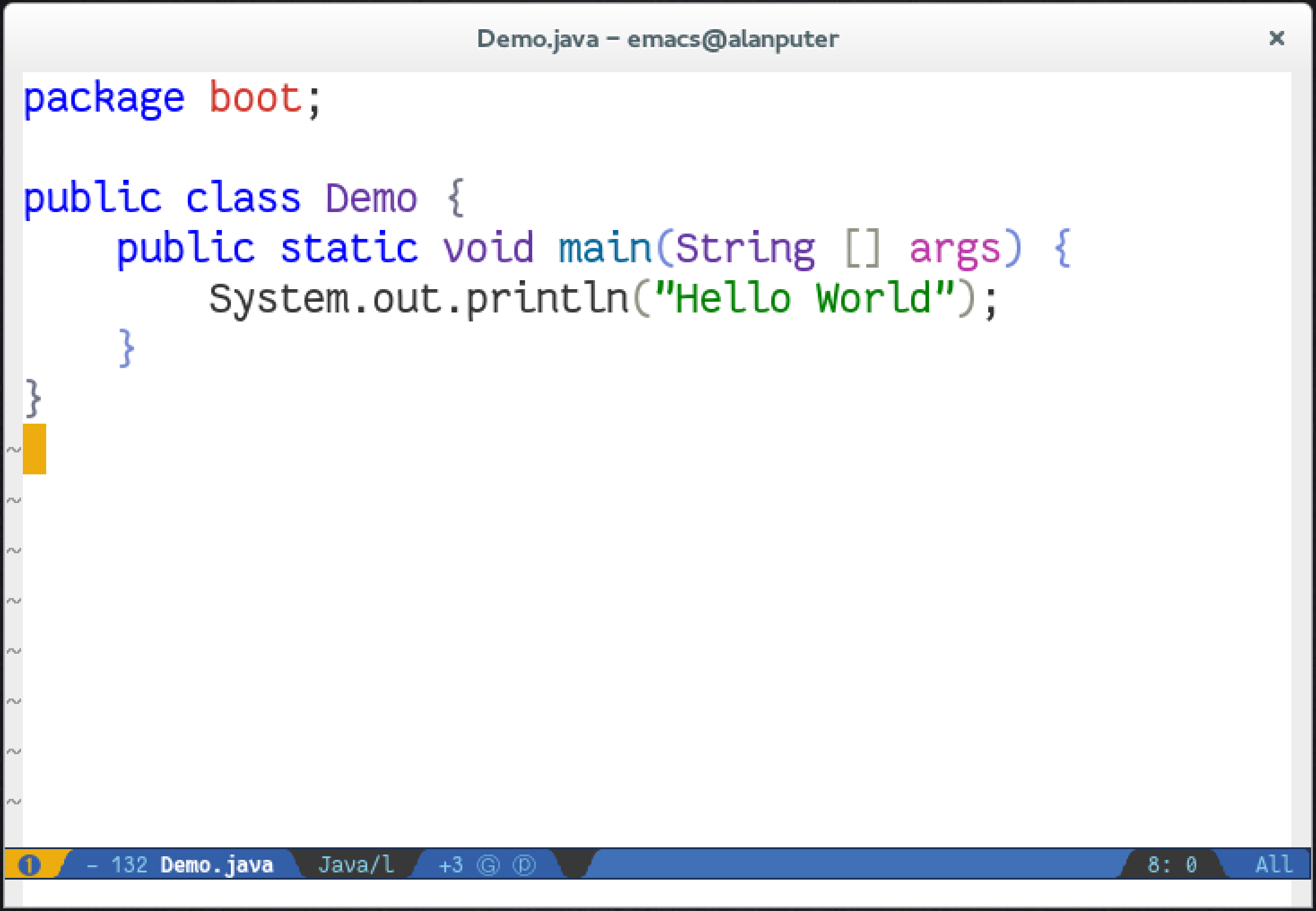
Our Dream Build Tool

- Made of many independent parts that each do one thing well
- Much better than one monolithic program
- Small things are only useful if composition left to user
- Our dream build empowers users to create and compose small parts

What is Boot?

- Uses Maven for dependency resolution
- We use it to build Clojure, ClojureScript
- http://boot-clj.com

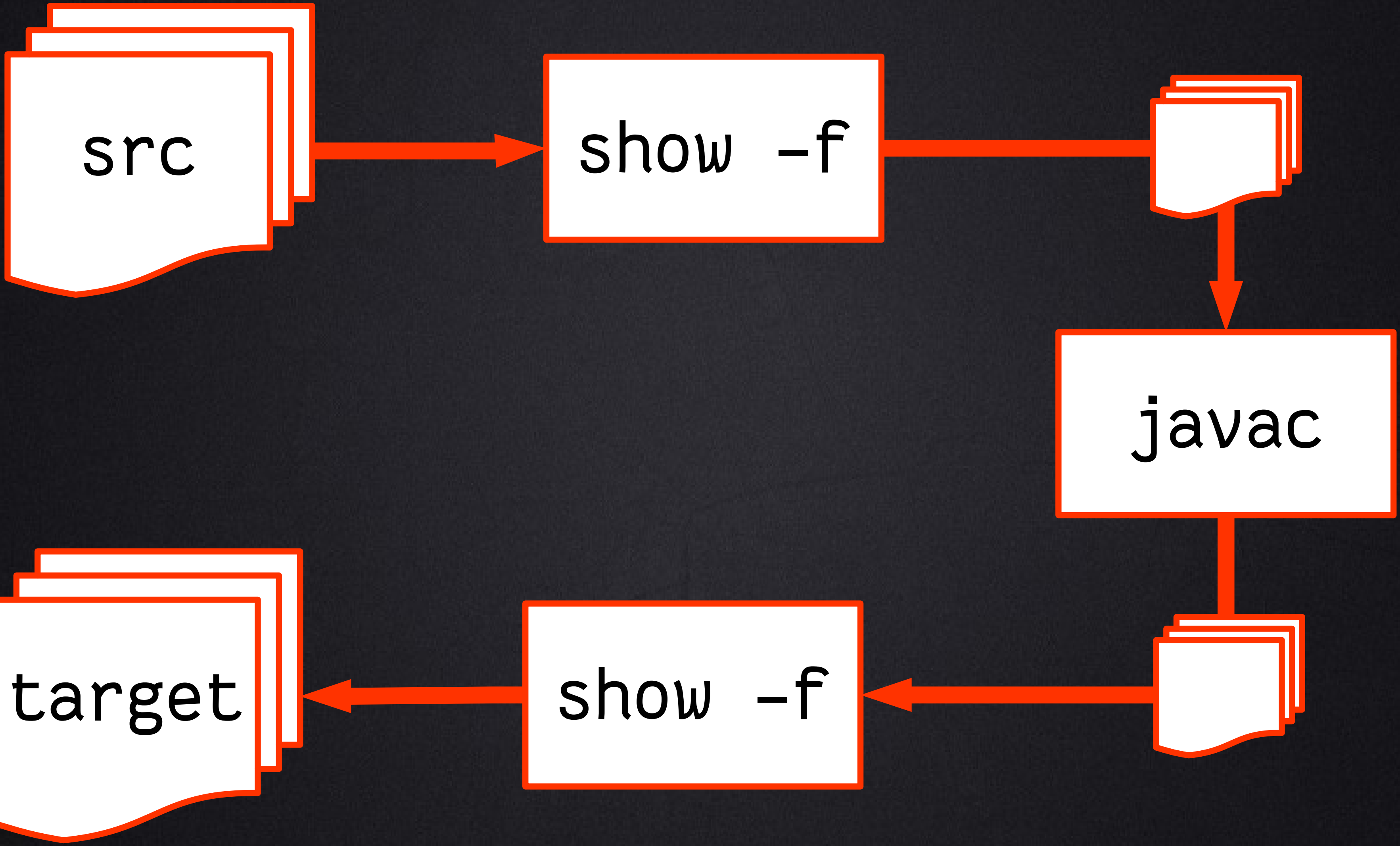
A Common Build Workflow



```
package boot;

public class Demo {
    public static void main(String [] args) {
        System.out.println("Hello World");
    }
}
```

Command Line: Compiling Java

The Boot Pipeline



Command Line: Installing a Jar

```
boot javac -- pom -- jar -- install
```

```
$ javac | pom | jar | install
```

```
(boot (javac) (pom) (jar) (install))
```

Unix Shell vs. Boot

	Process	*Connective*
Unix Shell	program	text
Boot	task	FileSet

Anatomy of a Task

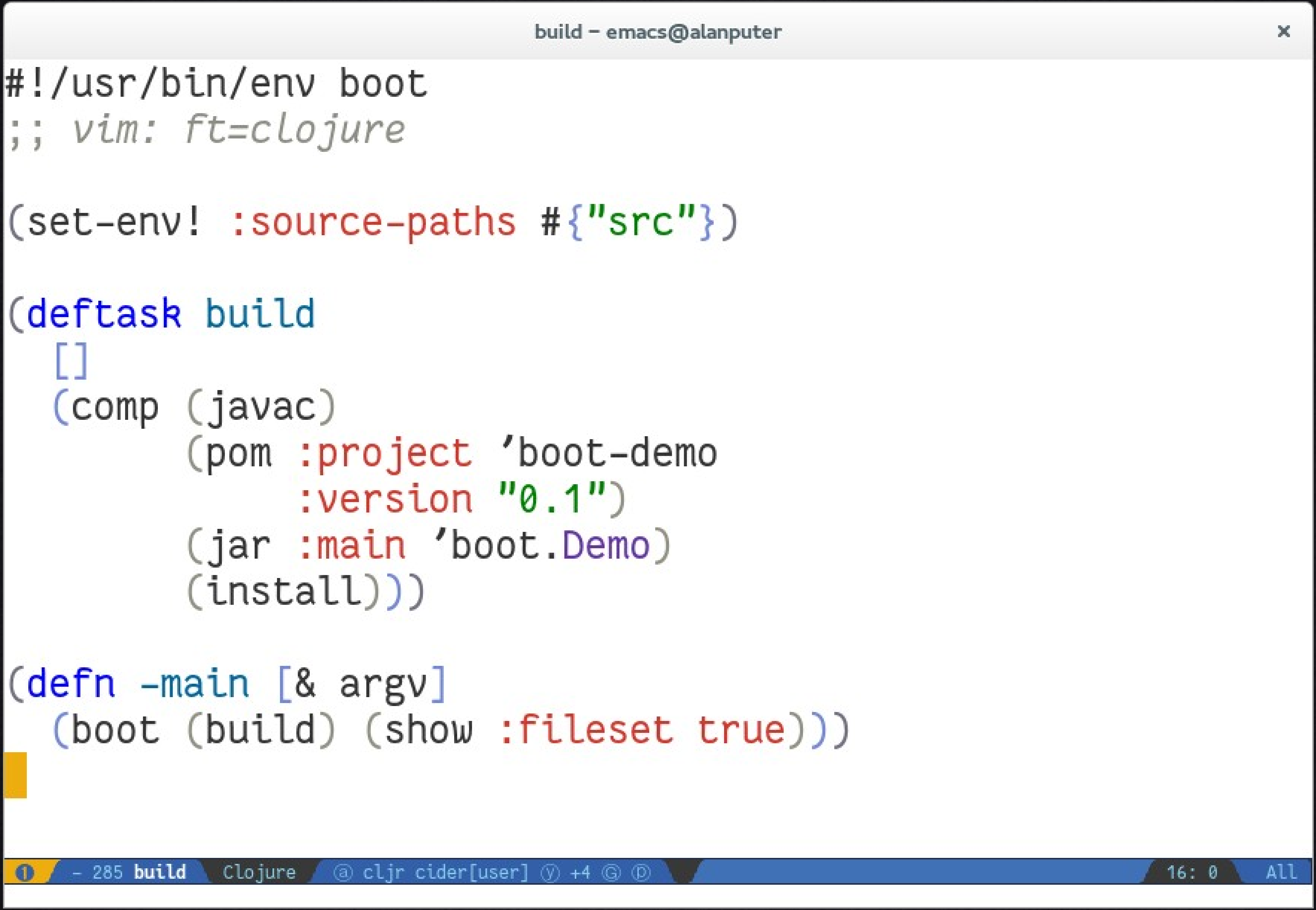
```
(deftask my-task []
  (let [history (atom [])]
    (fn [next-handler]
      (fn [fileset]
        (swap! history conj fileset)
        (next-handler fileset)))))
```

4. Handler
3. Middleware
2. Accumulated state
1. Task constructor

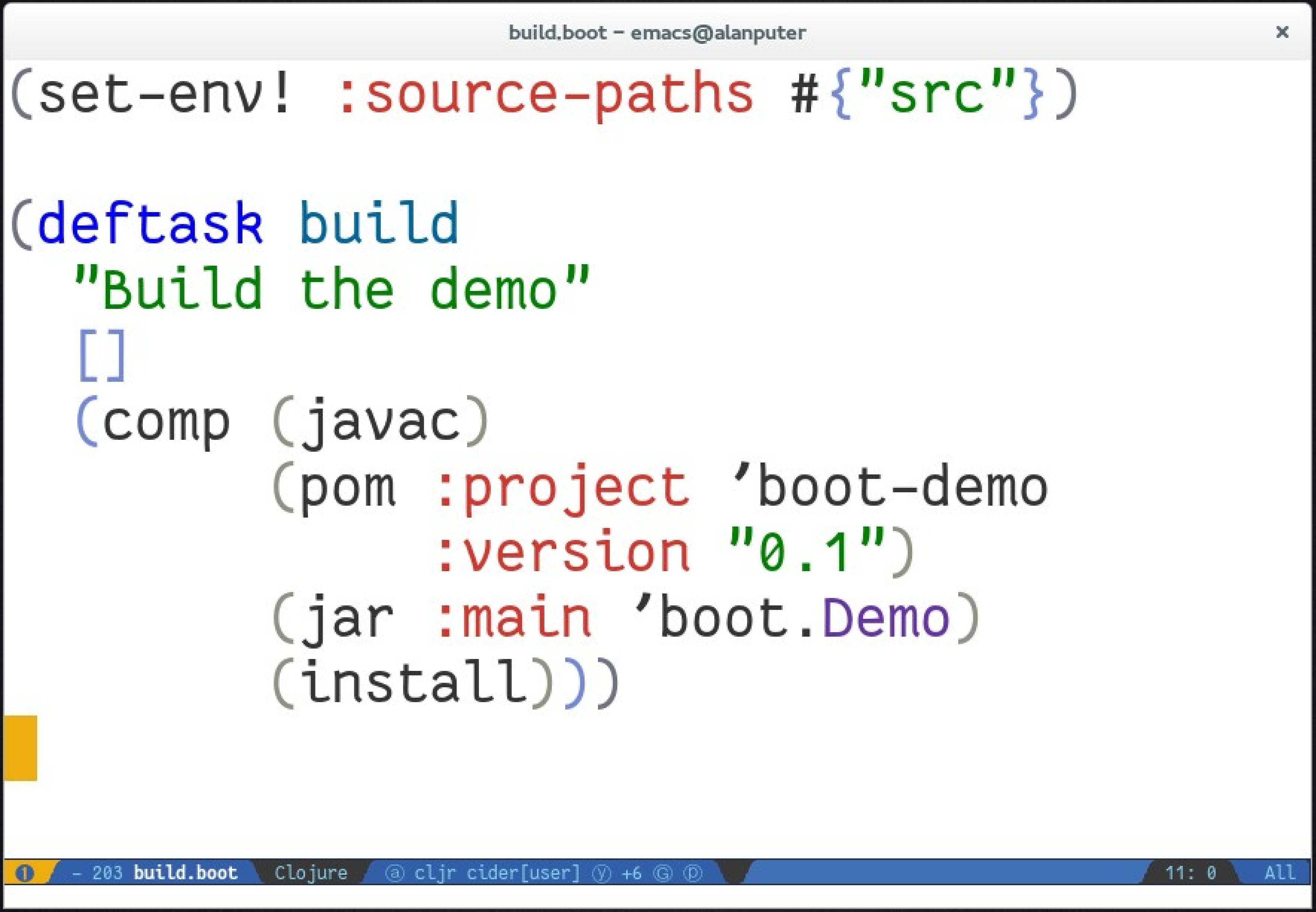
Anatomy of a Task

```
(deftask my-task []
  (let [history (atom [])]
    (with-pre-wrap [fileset]
      (swap! history conj fileset)
      (next-handler fileset))))
```

REPL: deftask



```
#!/usr/bin/env boot
;; vim: ft=clojure

(set-env! :source-paths #{"src"})

(deftask build
  []
  (comp (javac)
        (pom :project 'boot-demo
             :version "0.1")
        (jar :main 'boot.Demo)
        (install)))

(defn -main [& argv]
  (boot (build) (show :fileset true)))
```



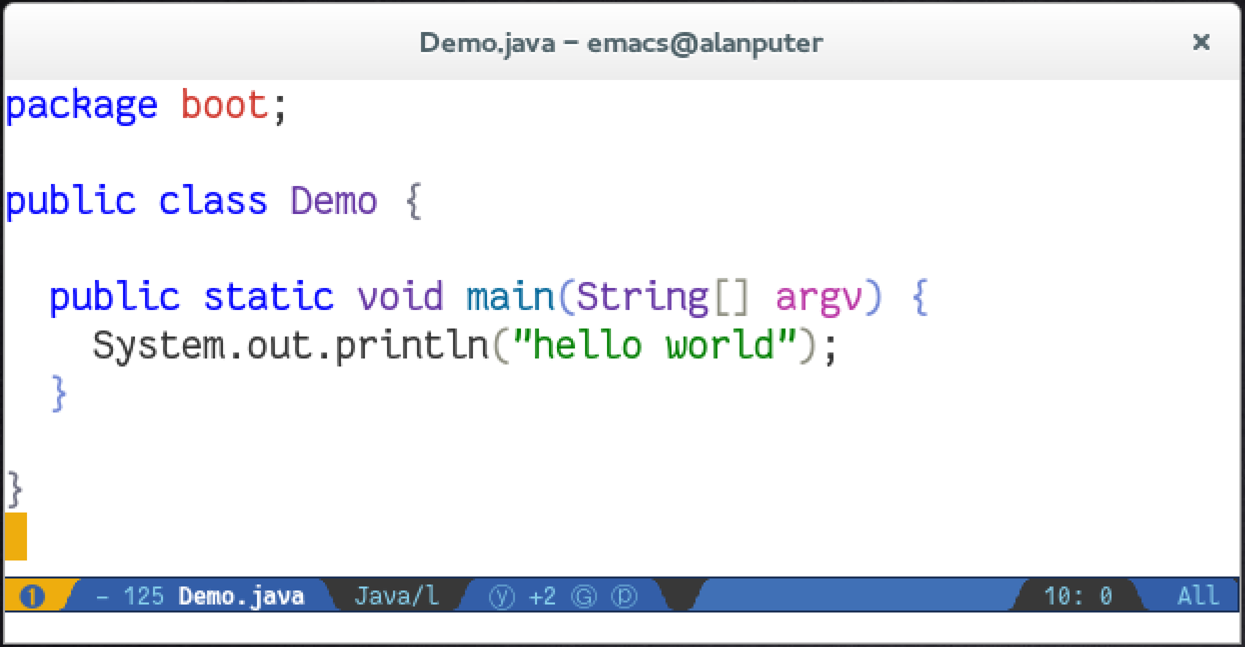
build.boot - emacs@alanputer

```clojure
(set-env! :source-paths #{"src"})

(deftask build
  "Build the demo"
  []
  (comp (javac)
        (pom :project 'boot-demo
             :version "0.1")
        (jar :main 'boot.Demo)
        (install)))
```

```
- 203 build.boot   Clojure   cljr cider[user] +6   11: 0   All
```

Making a new task

- We're done composing existing tasks
- It's time to make our own task



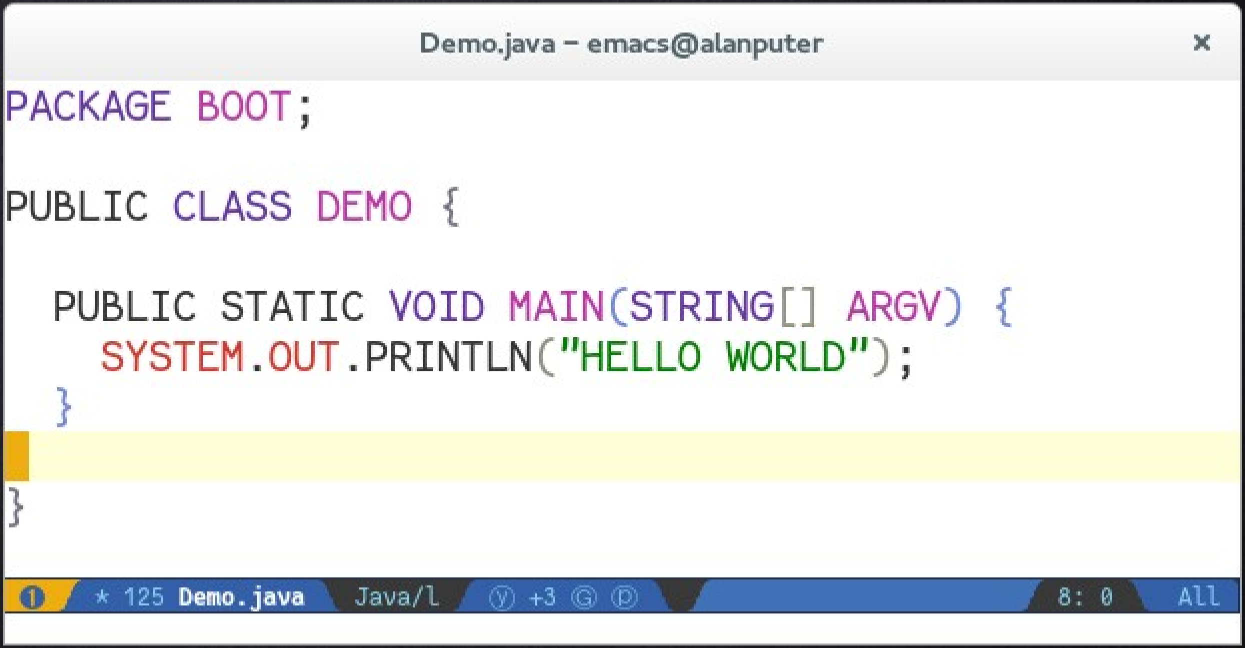
Demo.java – emacs@alanputer

```
package boot;

public class Demo {

  public static void main(String[] argv) {
    System.out.println("hello world");
  }

}
```

```
- 125 Demo.java  Java/l  ⓨ +2 Ⓖ ⓟ  10: 0  All
```

Java

Fortran



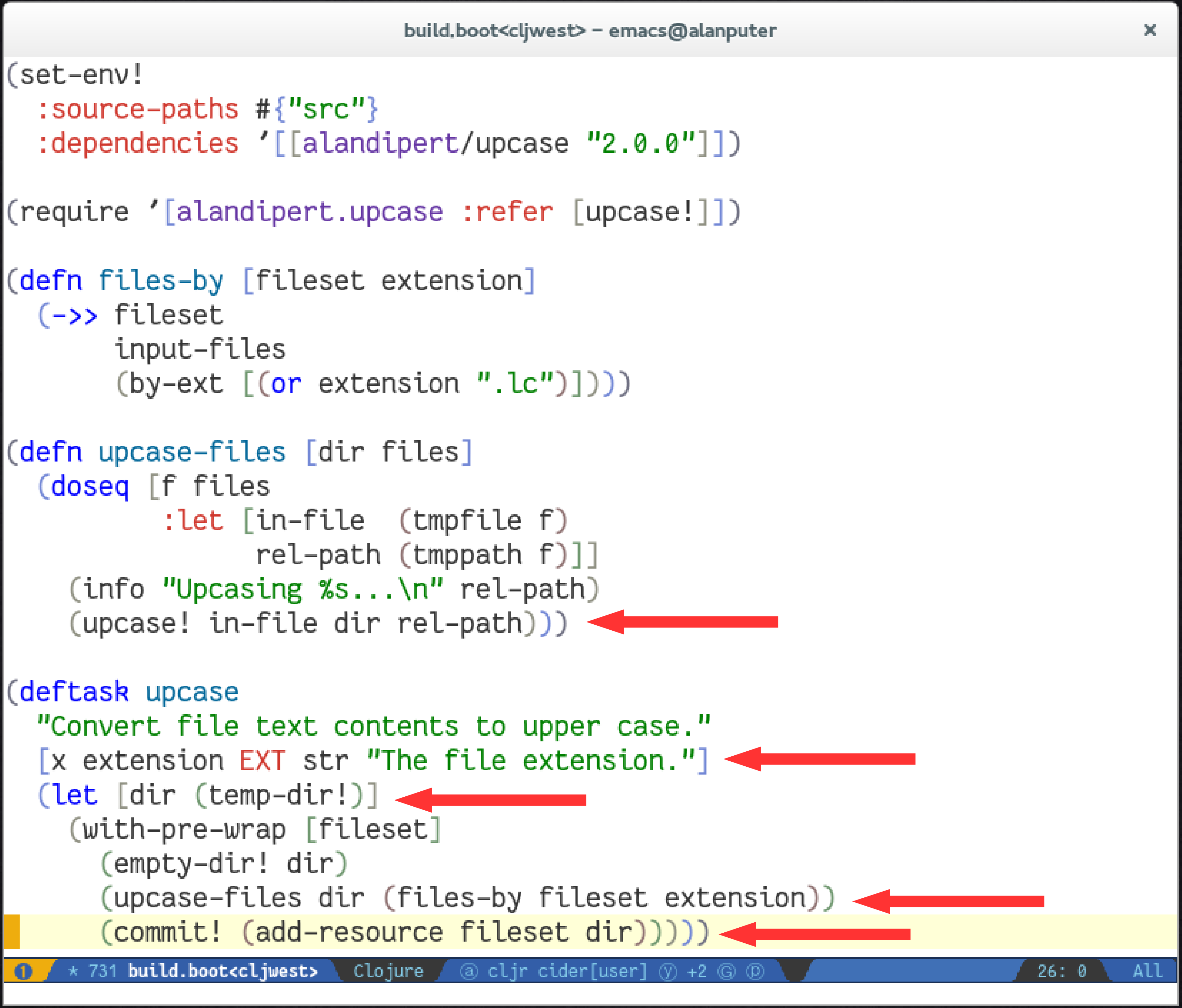
Demo.java – emacs@alanputer

```
PACKAGE BOOT;

PUBLIC CLASS DEMO {

  PUBLIC STATIC VOID MAIN(STRING[] ARGV) {
    SYSTEM.OUT.PRINTLN("HELLO WORLD");
  }

}
```

```
* 125 Demo.java  Java/l  ⓨ +3 Ⓖ ⓟ  8: 0  All
```

FileSet

- A little anonymous git repo
- Real files underneath but 100% managed
- Basis for Classpath
- Immutable
- Query API
- Add, remove
- Commit: mutates underlying files



build.boot<cljwest> – emacs@alanputer

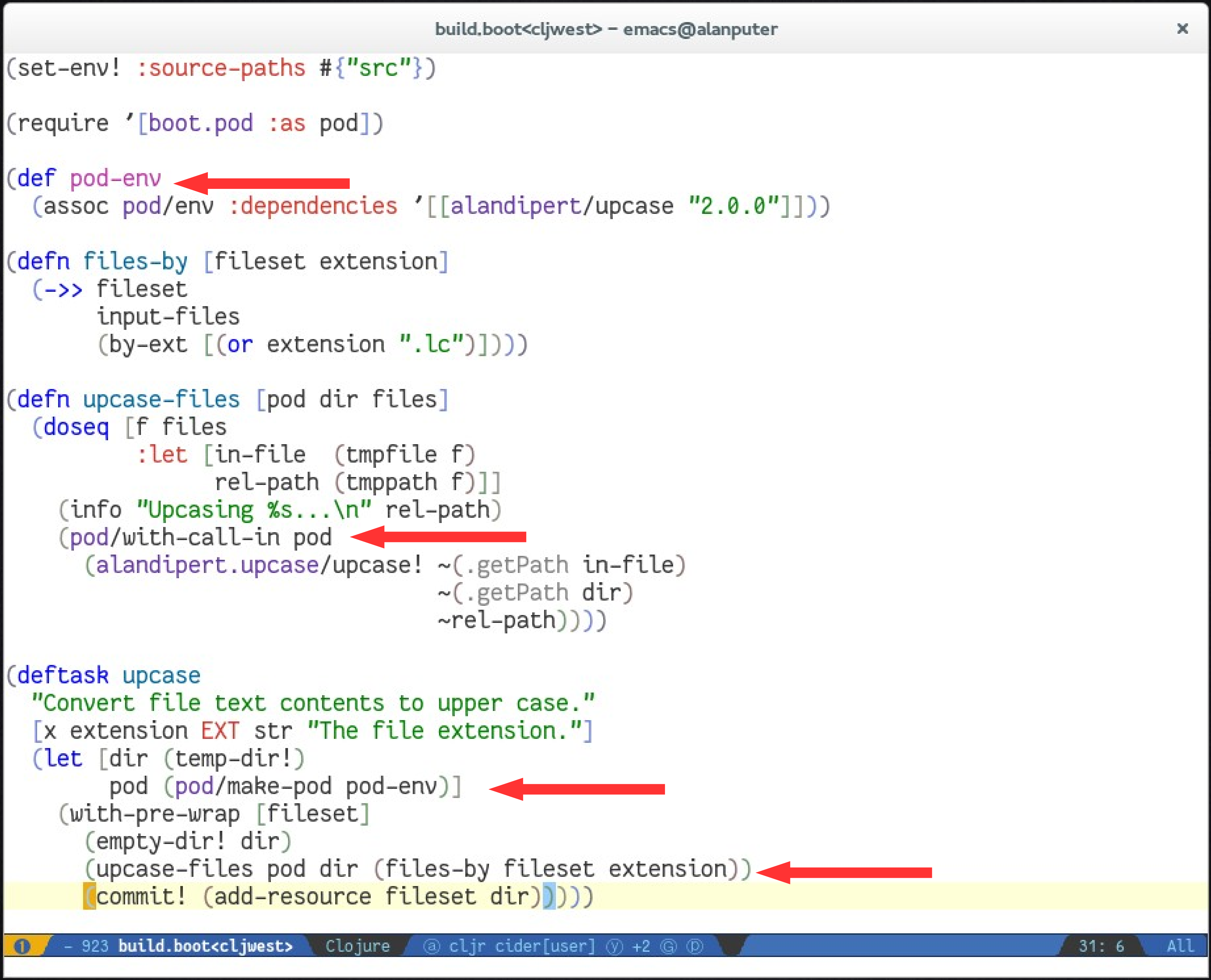
```clojure
(set-env!
  :source-paths #{"src"}
  :dependencies '[[alandipert/upcase "2.0.0"]])

(require '[alandipert.upcase :refer [upcase!]])

(defn files-by [fileset extension]
  (->> fileset
       input-files
       (by-ext [(or extension ".lc")])))

(defn upcase-files [dir files]
  (doseq [f files
          :let [in-file  (tmpfile f)
                rel-path (tmppath f)]]
    (info "Upcasing %s...\n" rel-path)
    (upcase! in-file dir rel-path)))

(deftask upcase
  "Convert file text contents to upper case."
  [x extension EXT str "The file extension."]
  (let [dir (temp-dir!)]
    (with-pre-wrap [fileset]
      (empty-dir! dir)
      (upcase-files dir (files-by fileset extension))
      (commit! (add-resource fileset dir)))))
```

```
* 731 build.boot<cljwest>   Clojure   cljr cider[user] +2   26: 0   All
```

Pods

- How we avoid “dependency hell”
- Isolated Clojure runtimes
- Each can have different dependencies
- Easy to create, run code inside of
- *Some things can't be passed between pods*

REPL: Pod



```
(set-env! :source-paths #{"src"})

(require '[boot.pod :as pod])

(def pod-env
  (assoc pod/env :dependencies '[[alandipert/upcase "2.0.0"]]))

(defn files-by [fileset extension]
  (->> fileset
       input-files
       (by-ext [(or extension ".lc")])))

(defn upcase-files [pod dir files]
  (doseq [f files
          :let [in-file  (tmpfile f)
                rel-path (tmppath f)]]
    (info "Upcasing %s...\n" rel-path)
    (pod/with-call-in pod
      (alandipert.upcase/upcase! ~(.getPath in-file)
                                 ~(.getPath dir)
                                 ~rel-path))))

(deftask upcase
  "Convert file text contents to upper case."
  [x extension EXT str "The file extension."]
  (let [dir (temp-dir!)
        pod (pod/make-pod pod-env)]
    (with-pre-wrap [fileset]
      (empty-dir! dir)
      (upcase-files pod dir (files-by fileset extension))
      (commit! (add-resource fileset dir)))))
```

- 923 build.boot<cljwest> Clojure cljr cider[user] +2 31: 6 All

http://boot-clj.com

Thank You!

Alan Dipert
@alandipert

Adzerk

Micha Niskin
@michaniskin