Nico

Clojure for pleasure

Cool Clojure and how to get started quickly

Content

Chapter 1

Why Clojure the lucky stiff

Dell's presentation on Infrastructure monitoring

Storm (from Twitter) with a Clojure DSL

The UK Government should take a look at Clojure

Rich Hickey Interview

Chapter 2

Welcome to Clojure, through Leiningen

This book is about sharing my love for Clojure, why I use it, and why I will keep using it for a fair bit of time. You don't need to be a great IT geek, you just have to start using Clojure. Here we go.

Clojure

http://learn-clojure.com/clojure_tutorials.html

I will not go too much into the first steps details, since this is not the aim of this book. There is a completely new online book written by John that should get all the attention it deserves.

Do not forget to do a few clojure koans to make sure the basic and the roots are ok.

I have added a few trips and ticks in the 00_zero.clj file (in the samples), so feel free to go along and try it by yourself, once you have read the next section about how to get started with clojure.

Leiningen

https://github.com/technomancy/leiningen

To start using clojure, you actually do not install it. You start by installing Leiningen. Once you have it installed, you should be able to see the following:

```
[Niko@Modrzyks-MacBook-Pro][13:44][~/projects/] % lein
Leiningen is a tool for working with Clojure projects.

Several tasks are available:
classpath   Print the classpath of the current project.
clean       Remove compiled class files and jars from project.
compile     Compile Clojure source into .class files.
deploy      Build jar and deploy to remote repository.
deps        Download :dependencies and put them in :library-path.
help        Display a list of tasks or help for a given task.
install     Install current project or download specified project.
interactive Enter an interactive task shell. Aliased to "int".
jar         Package up all the project's files into a jar file.
javac       Compile Java source files.
new         Create a new project skeleton.
plugin      Manage user-level plugins.
pom         Write a pom.xml file to disk for Maven interop.
repl        Start a repl session either with the current project or standalone.
retest      Run only the test namespaces which failed last time around.
run         Run the project's -main function.
search      Search remote maven repositories for matching jars.
test        Run the project's tests.
test!       Run a project's tests after cleaning and fetching dependencies.
trampoline  Run a task without nesting the project's JVM inside Leiningen's.
uberjar     Package up the project files and all dependencies into a jar file.
upgrade
version     Print version for Leiningen and the current JVM.
```

```
Run lein help $TASK for details.
See also: readme, tutorial, copying, sample, deploying and news.
```

Now if you just want to start using some live clojure, you just go with:

```
[Niko@Modrzyks-MacBook-Pro][13:50][~/] % lein repl
REPL started; server listening on localhost port 27991
user=>
```

And there you are. Was simple no ?

And here is your mother of them all hello world

```
; yes indeed
(println "hello clojure!")

; include a new namespace in the current namespace
; and use a custom name
(require '[clojure.string :as string])

; use the newly imported namespace
(def new_string (string/replace "Hello World" "World" "Nico"))

; print
(println new_string)
```

Now you have to realize that all the libs available to the Java World through Maven repositories, and all the libs hosted on Clojars can be integrated in the project in no time by using the command:

lein deps

The libraries will come to your local project folder. Let's hack in no time.

Search clojars

To search a library online, use the following syntax:

```
lein search <libraryname>
```

For example

```
lein search jsoup
```

will return

```
== Results from clojars - Showing page 1 / 1 total [org.clojars.clizzin/jsoup "1.5.1"] jsoup HTML parser
```

Upload to clojars

There is a plugin called lein-clojars

You install it somehow like this:

```
lein plugin install lein-clojars 0.9.1
```

Then in a clojure project, just go and upload it with

```
lein push
```

Nothing else.

Then a lein search will (should) show it.

Examples

All the examples in this book can be run with: `(use 'clojure.string) (load-file "code/18_clojure_soup.clj")`

If you can not run them, let me know ;)

Jark

http://icylisper.in/jark/started.html

Next you probably going to get tired of the Java VM so very slow startup time. Here comes nrepl and Jark

Create a new leiningen project, and follow the jark install steps:

```
; project.clj
(comment
    ; include this in project.clj
(defproject book01 "1.0.0-SNAPSHOT"
  :dependencies [
    [org.clojure/clojure "1.3.0"]
    [jark "0.4.2" :exclusions [org.clojure/clojure]]
  ]))

; now let's start the embedded repl
(require '[clojure.tools.nrepl :as nrepl])
(nrepl/start-server 9000)
```

and then start the repl with the new code from core.clj

```
[Niko@Modrzyks-MacBook-Pro][14:11][~/projects/book01/] % lein repl
REPL started; server listening on localhost port 42326
user=> (load-file "src/book01/core.clj")
```

And now straight from the shell you can use and run remote clojure code

```
[Niko@Modrzyks-MacBook-Pro][14:16][~/Downloads/jark-0.4.2-Darwin-x86_64/] % jark -e "(+ 2 2)"
=> 4
```

Now that is some sweet fast starting time, so stop buzzing around.

Pomegrenate

https://github.com/cemerick/pomegranate

This is the best way to add libraries at runtime. You still need the library itself to be available to the repl, by adding this to your project.clj file:

```
[com.cemerick/pomegranate "0.0.13"]
```

Then, here is how to import incanter in a running REPL session:

```
[missing 'code/11_pomegranata.clj' file]
```

Scala Integration

Since every source file in a lein project can be integrated, you can use the scalac-plugin

Here is the HelloWorld.scala file:

```
// HelloWorld.scala

class HelloWorld {
  def sayHelloToClojure(msg: String) =
    "Here's the second echo message from Scala: " concat msg
}
```

and here is how you call it from Clojure

```
(import HelloWorld)
(.sayHelloToClojure (HelloWorld.) "Hi there")
```

Now you can tell your friend you know more than one language on the JVM. :)

Java Integration

Note that the scala integration right above is also supported by the original java. This is actually built in leiningen. Here is how you put it together just in case:

```
(import HelloWorldJava)
(.sayHello (HelloWorldJava.) "World")
```

This is actually very good to keep old sources ready to be migrated, or actually the whole power of leiningen is that it is going to install everything you want from the command line to get you started with a regular java project as well. Yurk.

Hooke

https://github.com/technomancy/robert-hooke/

While searching for an example on the javac plugin above, I stubmled on a super example on how to add hooks on methods. Awesome.

```
[robert/hooke "1.1.2"]
```

```
(use 'robert.hooke)

(defn examine [x]
  (println x))

(defn microscope
  "The keen powers of observation enabled by Robert Hooke allow
  for a closer look at any object!"
  [f x]
  (f (.toUpperCase x)))

(defn doubler [f & args]
  (apply f args)
  (apply f args))

(defn telescope [f x]
  (f (apply str (interpose " " x))))
```

```
(add-hook #'examine #'microscope)
(add-hook #'examine #'doubler)
(add-hook #'examine #'telescope)

(examine "something")

; now we also have two append/prepend awesome macros
; to speed up adding hooooooks.
; see how it goes
(prepend print-name
  (print "The following person is awesome:"))

(print-name "Gilbert K. Chesterton")
```

This is like adding aspects to your clojure functions ! With no pointcuts ?? :)

Existing Leiningen Plugins

This is not the end, Leiningen provides plugins for all your needs. Groovy, Hadoop, ... you name it. It should be in the Leiningen plugins list

Writing your own plugin for Leiningen

Now if the above was not enough for your needs, here is how you create a plugin for your repetitive tasks.

Create a file under src/leiningen and add some clojure code:

```
(ns leiningen.todo)

(defn todo [project & args] (println "Hello TODO!!"))
```

Then a new task will be available to your current project.

To share the task with other, please have a look at writing leiningen plugins 101/

Clojure Contrib

http://dev.clojure.org/display/doc/Clojure+Contrib

Clojure contrib used to be a set of namespaces that were adding some great syntaxic sugar to a bunch of every day tasks. Now the jar file ended being over 2G :) big, and the core Clojure programmers decided to start making smaller projects.

Here is an example for parsing xml that takes an old example of parsing an XML RSS Feed, and make it work with today's Clojure 1.4 packaging.

Add this to your project.clj: `[org.clojure/data.zip "0.1.1"] [org.clojure/data.xml "0.0.6"]`

```
(require '[clojure.xml :as xml])
(require '[clojure.zip :as zip])
(use 'clojure.pprint)
(use 'clojure.data.zip.xml)

(defn parse-str [s]
    (zip/xml-zip (xml/parse (new org.xml.sax.InputSource
                                 (new java.io.StringReader s)))))
```

```
(defn parse-file [f]
    (zip/xml-zip (xml/parse (new org.xml.sax.InputSource
                                 (new java.io.FileReader f)))))

(def atom1 (parse-str "<?xml version='1.0' encoding='UTF-8'?>
                <feed xmlns='http://www.w3.org/2005/Atom'>
                  <id>tag:blogger.com,1999:blog-28403206</id>
                  <updated>2008-02-14T08:00:58.567-08:00</updated>
                  <title type='text'>n01senet</title>
                  <link rel='alternate' type='text/html' href='http://n01senet.blogspot.com/'/>
                  <entry>
                    <id>1</id>
                    <published>2008-02-13</published>
                    <title type='text'>clojure is the best lisp yet</title>
                    <author><name>Chouser</name></author>
                  </entry>
                  <entry>
                    <id>2</id>
                    <published>2008-02-07</published>
                    <title type='text'>experimenting with vnc</title>
                    <author><name>agriffis</name></author>
                  </entry>
                </feed>
                "))

;(pprint atom1)
(xml-> atom1 :entry text)
```

Chapter 3

Libraries

Now that we are ready for some hacking, here are sites, that refer interesting Clojure libraries:

- clojure-libraries on appspot.com/
- clojurewerkz
- Clojure on github
- http://programmers.stackexchange.com/questions/125107/what-are-the-essential-clojure-libraries-to-learn-beyond-the-basics-of-core
- http://www.clojure-toolbox.com/
- http://twitch.nervestaple.com/2012/01/12/clojure-hbase/
- http://clojure.jr0cket.co.uk/

Noir

https://github.com/ibdknox/noir http://www.webnoir.org/

It's dark in here ! Noir is probably the simplest way to write a functional web application in clojure.

This is how it looks like in Clojure code

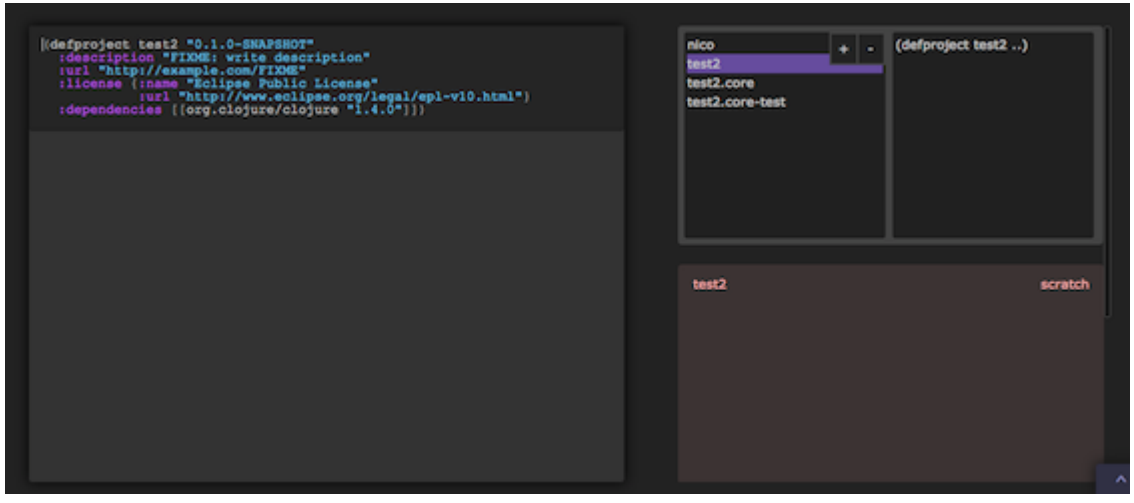
```
(ns my-app
  (:use noir.core)
```

```
  (:require [noir.server :as server]))

(defpage "/welcome" []
    "Welcome to Noir!")

(server/start 8080)
```

and/or this is how you would have it started in a few seconds with the leiningen command we installed earlier on

```
lein plugin install lein-noir 1.2.1
lein noir new my-website
cd my-website
lein run
```

Now the guy from Noir has also started a very cute project named Playground where you can do live execution of your code with a nice UI.



http-kit

https://github.com/shenfeng/http-kit

In the same time, you can turn yourself asynchrnous with http-kit.

```
[me.shenfeng/http-kit "1.2.0"]
```

```
(use 'me.shenfeng.http.server)

(defasync async [req]
  (.start (Thread. (fn []
                     (Thread/sleep 1000)
                     ;; return a ring spec response
                     ;; call (:cb req) when response ready
                     ((:cb req) {:status 200 :body "hello async"})))))

(run-server async {:port 8080})
```

The server will be started, and will delay answer to the client. No dependency so kind of very lightweight and wicked. It also incorporate a kinda cool http client, again with no dependencies. Give a try.

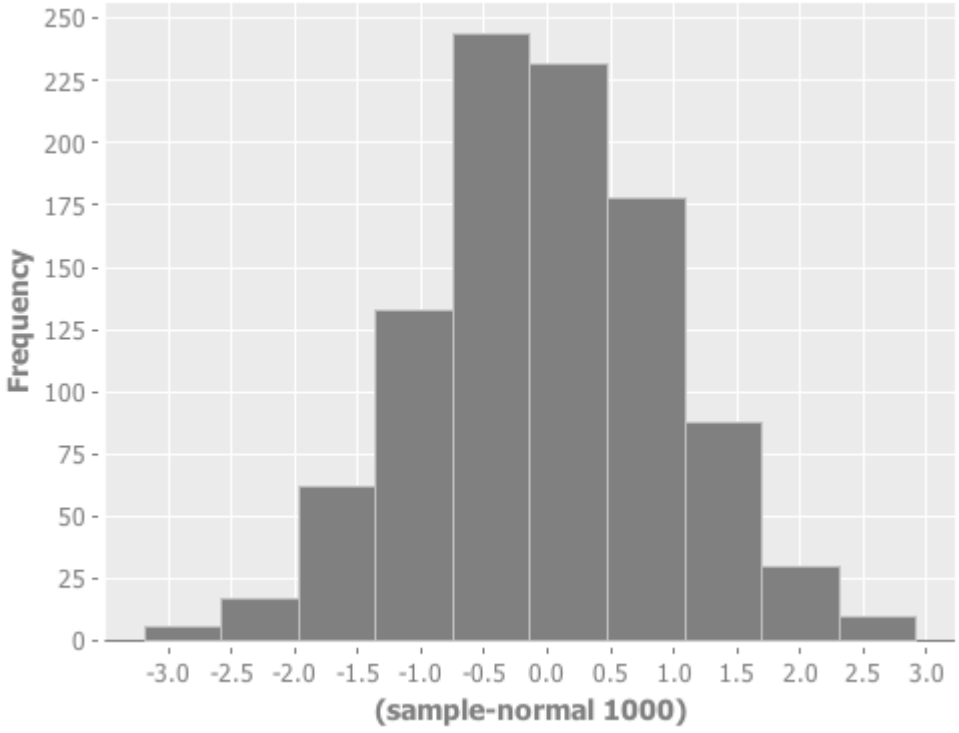
```
hello async
```

Incanter

Download and Get started

```
(use '(incanter core stats charts))

; Try an example: sample 1,000 values from a standard-normal distribution and view a histogram:
(view (histogram (sample-normal 1000)))

; this is saving the file as png
; wow. so fast.
(save (histogram (sample-normal 1000)) "test.png")

; Below only works if you have latex installed
; (def eq (str "f(x)=\\frac{1}{\\sqrt{2\\pi \\sigma^2}}" "e^{\\frac{-(x - \\mu)^2}{2 \\sigma^2}}"))
; (view (latex eq))
; (save (latex eq) filename)
```

The image below has been generated from the script above !



Whats your next diagram ?

Marginalia

https://github.com/fogus/marginalia

Marginalia is your best literate programming tool for clojure.

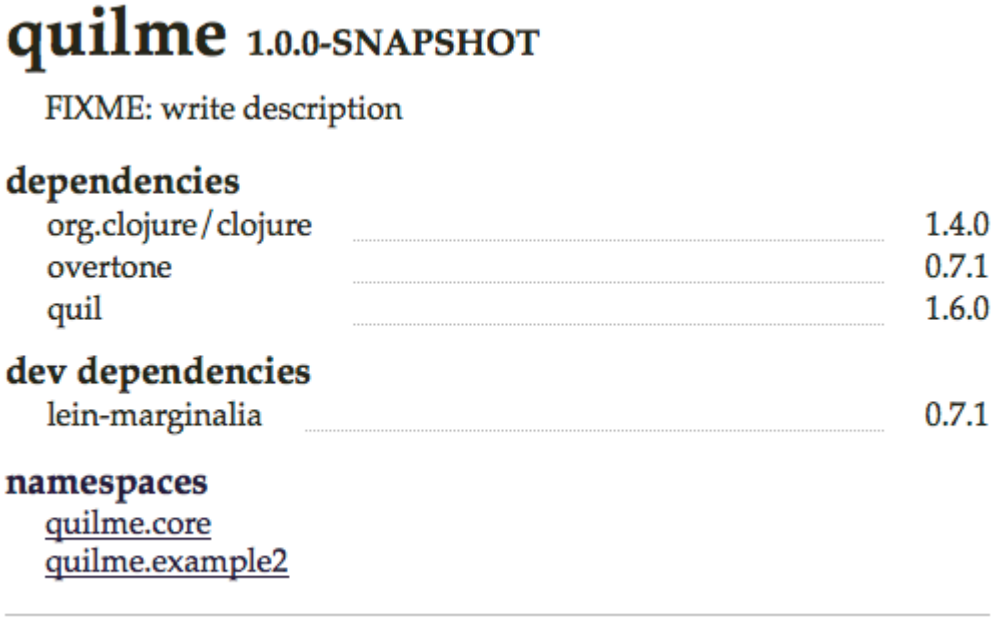
Install it as a dependency in your project.clj file:

```
:dev-dependencies [lein-marginalia "0.7.1"]
```

Then, just use it:

```
lein marg
```

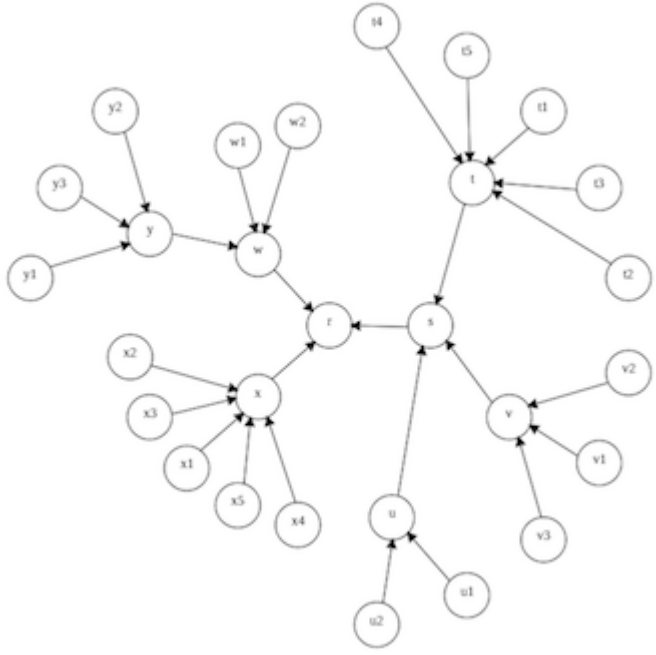
And dependending on the amount of comments you wrote in the code, you will get something similar to this:



Lacij

https://github.com/pallix/lacij

A library that can quickly create SVG diagram with automatic layout, like this one:



Then, a graph can be drawn like this:

```
(use 'lacij.graph.svg.graph)

(-> (create-graph :width 800 :height 400)
```

```
(add-default-node-style :fill "lightgreen")
(add-default-edge-style :stroke "royalblue")
(add-node :hermes "Hermes" :x 10 :y 30 :style {:fill "lightblue"})
(add-node :zeus "Zeus" :x 300 :y 150 :rx 15 :ry 15)
(add-node :ares "Ares" :x 300 :y 250 :style {:fill "lavender" :stroke "red"})
(add-edge :father1 :hermes :zeus "son of"
          :style {:stroke "darkcyan" :stroke-dasharray "9, 5"})
(add-edge :father2 :ares :zeus)
(add-default-node-attrs :rx 5 :ry 5)
(add-node :epaphus "Epaphus" :x 450 :y 250)
(add-edge :epaphus-zeus :epaphus :zeus)
(add-node :perseus "Perseus" :x 600 :y 150)
(add-edge :perseus-zeus :perseus :zeus)
(add-label :father2 "son of" :style {:stroke "crimson"
                                     :font-size "20px"
                                     :font-style "italic"})
(build)
(export "styles.svg"))
```

The advantage is that you can add and remove nodes dynamically, thus giving a dynamic view about live typologies.

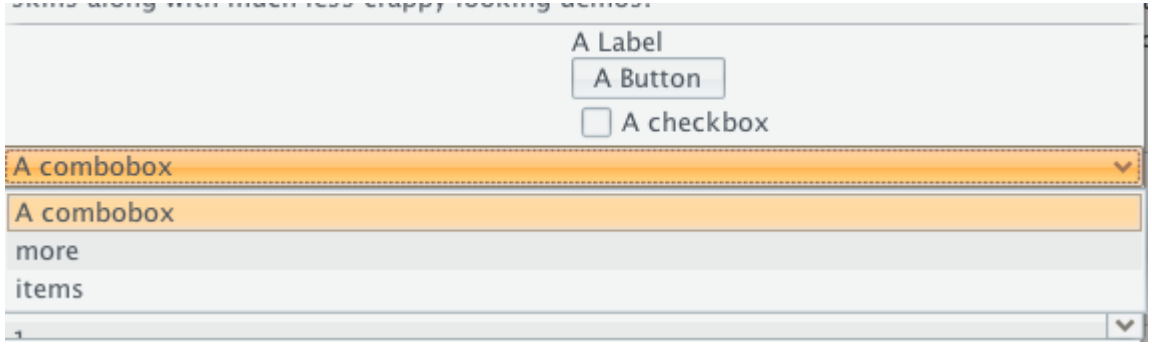
Seesaw

https://github.com/daveray/seesaw

Make cool UIs in no time.

The best tutorial I have found so far is here

And by trying the example, here is a style you can get after a few lines:



clj-xpath

https://github.com/kyleburton/clj-xpath

Now you been looking on how to process those xml files as fast as possible, here is a super way to do it, clj-xpath.

```
; import clj
(use 'clj-xpath.core)

; slurp the remote document
(def xdoc (slurp "http://google-web-toolkit.googlecode.com/svn/trunk/samples/expenses/pom.xml"))

; show the top element
($x:tag "/*" xdoc)

; show the developer name
```

```
($x:text* "/project/developers/developer/name" xdoc)

; describe all the dependencies
(doseq [node ($x "/project/dependencies/dependency" xdoc)]
  (prn (format "%s %s %s"
               ($x:text "./groupId"    node)
               ($x:text "./artifactId" node)
               ($x:text "./version"    node))))
```

clojure-soup

https://github.com/mfornos/clojure-soup

This is a wrapper to provide awesome parsing of html files, whether local or remote. Once you have added the library, check out the following example:

```
; start jsoup
(use 'jsoup.soup)

; Get all Emoji names concatenated by single bars from 'emoji-cheat-sheet.com':

($ (get! "http://www.emoji-cheat-sheet.com/")
   "li div:has(span.emoji)" (text)
   (map #(clojure.string/replace % ":" ""))
   (clojure.string/join "|"))
```

Enlive

https://github.com/cgrand/enlive

HTML parser, and Templating framework at the same time, Enlive does a super job of making HTML fun. (yes you read me.)

This is an example taken from a slick tutorial by David Nolen.

```
(html/deftemplate index "tutorial/template1.html"
  [ctxt]
  [:p#message] (maybe-content (:message ctxt) "Nothing to see here!"))
```

You declare templates in a regular html files, thus your designer can do his Dreamweaver work the way he/she usually does it. Then as a developper you just come and stick content at the location that has been decided. Slick uh ?

Docjure

https://github.com/ative/docjure

The best way to have fun with spreadsheet in clojure. Relies on the Apache POI library but with some clojure sauce so it can actually be eaten.

This is how you write a simple spreadsheet.

```
(use 'dk.ative.docjure.spreadsheet)

;; Create a spreadsheet and save it
```

```
(let [wb (create-workbook "Price List"
                          [["Name" "Price"]
                           ["Foo Widget" 100]
                           ["Bar Widget" 200]])
      sheet (select-sheet "Price List" wb)
      header-row (first (row-seq sheet))]
  (do
    (set-row-style! header-row (create-cell-style! wb {:background :yellow,
                                                       :font {:bold true}}))
    (save-workbook! "spreadsheet.xlsx" wb)))
```

For more examples, I suggest you look at the "horrible" documentation :)

Postal

https://github.com/drewr/postal

I am pretty sure you knew how to send a mail before, but look at the awesome way of doing this in Clojure.

```
(use 'postal.core)

(send-message ^{ :host "smtp.gmail.com"
                 :user "<username>"
                 :pass "<password>"
                 :ssl :yes!!!11
               }
               {  :from "hellonico@gmail.com"
                  :to "hellonico@gmail.com"
                  :subject "Hi!"
```

```
:body [
    {:content "<html>新橋</html>" :type "text/html; charset=utf-8"}
    {:type :attachment :content (java.io.File. "code/mail.txt")}
    {    :type :inline
         :content (java.io.File. "code/mail.txt")
         :content-type "application/text"}
]})
```

This is including attachment and Japanese encoding. yey

Lobos

http://budu.github.com/lobos/

Here is comes Lobos, or how to manage your database directly from a Clojure REPL.

```
; create SQL tables with lobos
(use 'lobos.connectivity
     'lobos.core
      lobos.schema)

(def h2
  {:classname   "org.h2.Driver"
   :subprotocol "h2"
   :subname      "./korma"})

 ; seems lobos is not working anymore without clojure-contrib
(create h2
  (table :users
```

```
    (varchar :first 100)
    (varchar :last 100)))
```

You define your database connection, and just go and create, drop delete, what every you need. This is also very useful for testing.

Korma

http://sqlkorma.com/docs

Korma makes actually enjoyable to write SQL queries.

clj-dns

https://github.com/brweber2/clj-dns

DNS Querying in Clojure.

```
[com.brweber2/clj-dns "0.0.2"]
```

How easy it is to do lookup and reverse lookup. Need anything else ?

```
(use 'clj-dns.core)
(import 'org.xbill.DNS.Type)

; regular lookup
(pprint (:answers (dns-lookup "www.google.com" Type/A)))
```

```
; reverse lookup
(reverse-dns-lookup "173.194.38.113")
```

Clostache

https://github.com/fhd/clostache

```
[de.ubercode.clostache/clostache "1.3.0"]
```

Some very simple templating magic.

```
(use 'clostache.parser)

(render "Hello, {{name}}!" {:name "Felix"})

(render "<ul>{{#names}}<li>{{.}}</li>{{/names}}</ul>" {:names ["Felix" "Jenny"]})

(render-resource "code/sample.mustache" {:name "Michael"})
```

clojure-csv

https://github.com/davidsantiago/clojure-csv

```
[clojure-csv/clojure-csv "2.0.0-alpha1"]
```

Now that we have mustache, I actually have added a longer example showing how to convert a csv files containing people addresses, to a vcard vcf file format. Took a few minutes to write the code, not a single exception, and all using multiple cores to convert the map.

Sweeet.

```
; https://github.com/davidsantiago/clojure-csv/blob/master/test/clojure_csv/test/core.clj

(import  '[java.io StringReader])
(use 'clojure.java.io)
(use  'clojure-csv.core)

; dumb example
(parse-csv "a,b,c")


;
; Longer example on how to convert a CSV list of people
; to vcard format
;
(defn l[o]
(let [
   tel_or_email (get o 3)
   is_email (re-find #"@" tel_or_email)
   ]
   {
      :id (first o)
      :first (get o 1)
      :last (get o 2)
      :email (if is_email tel_or_email)
      :tel (if (not is_email) tel_or_email)
   }
   ))

(use 'clostache.parser)
```

```
; not all field are there, this depends on your own implementation
(defn vcard
  "Converts the intermal map representation into a format meeting the vCard specification"
  [record]
  (render "BEGIN:VCARD
VERSION:3.0
N:{{first}};{{last}};
{{#tel}}
TEL;TYPE=CELL;TYPE=PREF:{{tel}}
{{/tel}}
EMAIL;TYPE=PREF:{{email}}
END:VCARD
" record))

; now the core of the work
; 1 line to parse the csv file
(def ppl (slurp "/Users/Niko/Dropbox/perso/contacts.csv"))
; 1 line to convert each line to a vcard
(def vcards (map #(vcard (l %)) (parse-csv ppl)))

(spit "contacts.vcf" (apply str vcards))
```

This is greatly inspired by a web based address book application available on github.

clj-rss

https://github.com/yogthos/clj-rss

```
[clj-rss "0.1.2"]
```

```
[missing 'code/35_rs.clj' file]
```

This is taken from here.

clj-growlnotify

https://github.com/franks42/clj-growlnotify

```
[clj-growlnotify "0.1.1"]
```

```
(use 'clj-growlnotify)

(growl-notify "my title" "my message")
(growl-notify "my title" "my message" :name "myapp" :sticky true)
(growl-notify {:title "my title" :message "my message" :name "myapp" :sticky true})
```

When you run the script above, if you have the growl notification system (osx, windows) installed on your machine, you will get a nice message like the one below:

cheshire

https://github.com/dakrone/cheshire

Or how to do proper json fun in the Clojure style. Cheshire is there for good.

```
[cheshire "4.0.2"]
```

```
(use '[cheshire.core])

(generate-string {:foo "bar" :baz 5})

;; parse some json
(parse-string "{\"foo\":\"bar\"}")
;; => {"foo" "bar"}

;; parse some json and get keywords back
(parse-string "{\"foo\":\"bar\"}" true)
;; => {:foo "bar"}

;; parse some json and munge keywords with a custom function
(parse-string "{\"foo\":\"bar\"}" (fn [k] (keyword (.toUpperCase k))))
;;; => {:FOO "bar"}
```

Note how you can get keywords back, or do some extra mapping on dates. No more JSON problems.♫

Vaadin

http://dev.vaadin.com/wiki/Articles/ClojureScripting

This is not so much a library than a way to develop slick web application using Vaadin, but without the java boiler plate code.

clj-opennlp

https://github.com/dakrone/clojure-opennlp Natural Language Processing in Clojure.

Tokenizer for text analysis.

clj-http

https://github.com/dakrone/clj-http Http Slurping

Cascalog

https://github.com/nathanmarz/cascalog/wiki Hadoop Query from Clojure

Aleph

https://github.com/ztellman/aleph Websockets

Lamina

https://github.com/ztellman/lamina Event workflow for clojure

Conduit

http://www.intensivesystems.net/tutorials/stream_proc.html Stream processing in Clojure

Chapter 4

Processing, OpenCV and imaging

Quil

https://github.com/quil/quil

Quil is to Processing what Clojure is to Java, some fresh air.

This is how your quil-ed processing sketch now looks like:

```
(ns quilme.core
  (:use quil.core))

(defn setup []
  ;(smooth)                    ;;Turn on anti-aliasing
  (frame-rate 5)               ;;Set framerate to 1 FPS
  (background 0))              ;;Set the background colour to
                               ;;  a nice shade of grey.
(defn draw []
  (stroke (random 255))        ;;Set the stroke colour to a random grey
  (stroke-weight (random 10))  ;;Set the stroke thickness randomly
  (fill (random 255))          ;;Set the fill colour to a random grey

  (let [diam (random 100)      ;;Set the diameter to a value between 0 and 100
        x    (random (width))  ;;Set the x coord randomly within the sketch
        y    (random (height))] ;;Set the y coord randomly within the sketch
```

```
    (ellipse x y diam diam)))        ;;Draw a circle at x y with the correct diameter

(defsketch example                   ;;Define a new sketch named example
  :title "Oh so many grey circles"   ;;Set the title of the sketch
  :setup setup                       ;;Specify the setup fn
  :draw draw                         ;;Specify the draw fn
  :render :opengl
  :decor false
  :size [800 600])                   ;;You struggle to beat the golden ratio
```

Note the decor set to false, that hides most of the uglyness of the Window borders.

And all the examples you have ever dreamed from the Generative Art book have been implemented in Clojure/Quil.

Chapter 5

OpenCV, Arduino

Physics with Clojure

- http://nakkaya.com/2010/07/21/physics-with-clojure/
- http://antoniogarrote.wordpress.com/2011/01/30/ocr-with-clojure-tesseract-and-opencv/

Cloduino

https://github.com/nakkaya/clodiuno

Once you have uploaded the Firmata protocol on your board `File -> Examples -> Firmata -> StandartFirmata`

You can send a Clojure SOS.

```
(ns sos
   (:use :reload-all clodiuno.core)
   (:use :reload-all clodiuno.firmata))

(def short-pulse 250)
(def long-pulse 500)
(def letter-delay 1000)

(def letter-s [0 0 0])
```

```
(def letter-o [1 1 1])

(defn blink [board time]
  (digital-write board 13 HIGH)
  (Thread/sleep time)
  (digital-write board 13 LOW)
  (Thread/sleep time))

(defn blink-letter [board letter]
  (doseq [i letter]
    (if (= i 0)
      (blink board short-pulse)
      (blink board long-pulse)))
  (Thread/sleep letter-delay))

(defn sos []
  (let [board (arduino :firmata "/dev/tty.usbserial-A900adPT")]
    ;;allow arduino to boot
    (Thread/sleep 5000)
    (pin-mode board 13 OUTPUT)

    (doseq [_ (range 3)]
      (blink-letter board letter-s)
      (blink-letter board letter-o)
      (blink-letter board letter-s))

    (close board)))
```

There are so many more great Samples that Nakkaya has been writing. Make sure you go through them.

Chapter 6

Live Music

Overtone

https://github.com/overtone/overtone/wiki/Getting-Started

Because you really need to live audio programming to be a real VJ these days. This is how you install it:

```
(defproject tutorial "1.0"
  :dependencies [ [org.clojure/clojure "1.3.0"]
                  [overtone "0.7.1"] ])
```

Once the library is in your project, type

```
(use 'overtone.live)
```

And now you can define an instrument

```
(definst foo [] (saw 220))
```

And make some sound !

(foo) ; Call the function returned by our synth 4 ; returns a synth ID number (kill 4) ; kill the synth with ID 4 (kill foo) ; or kill all instances of synth foo

Chapter 7

In the Clouds

- http://architects.dzone.com/articles/how-monitoring-ec2-clojure-and

clj.monitor

https://github.com/killme2008/clj.monitor

JRobin wrapper for Clojure?

http://www.jrobin.org/index.php/Main_Page

Parallel SSH

http://blog.rjmetrics.com/Parallel-SSH-and-system-monitoring-in-Clojure/

Pallet

http://palletops.com/

Pallet is the mother of them all of cloud infrastructure tool. See the list of providers it supports! They are actually doing this through jclouds

Heroku

Debugging clojure on Heroku

VMFest

https://github.com/tbatchelli/vmfest

Easy VirtualBox wrapper for easy cloud management.

```
; you can define your machine this way:
(def my-machine
     (instance my-server "my-vmfest-vm"
        {:uuid "/Users/tbatchelli/imgs/vmfest-Debian-6.0.2.1-64bit-v0.3.vdi"}
        {:memory-size 512
         :cpu-count 1
         :network [{:attachment-type :host-only
                    :host-only-interface "vboxnet0"}
                   {:attachment-type :nat}]
         :storage [{:name "IDE Controller"
                    :bus :ide
                    :devices [nil nil {:device-type :dvd} nil]}]
         :boot-mount-point ["IDE Controller" 0]}))
```

```
; then this is the way to start/stop.. any of the virtual machines you have
(start my-machine)
(pause my-machine)
(resume my-machine)
(stop my-machine)
(destroy my-machine)
```

Make sure you also look at the playground and have a look at the tutorial

- http://www.intensivesystems.net/tutorials/stream_proc.html
- https://github.com/sattvik/Clojure-Android-Examples
- Android?: http://kinjo.github.com/ojag-clojure/#slide61

Clojure for pleasure
Nico