Clojure

*Dev-Environments & Workflows - A deep dive investigation*

Brian Abbott, Loren Larsen, Late June, July 2019

# 

# 

[Overview 4](#_Toc31184776)

[Demo-Application as Common Analysis Target 4](#_Toc31184777)

[Clojure Tools and Developer Utilities 4](#_Toc31184778)

[CLJ-CLI 5](#_Toc31184779)

[Deps.EDN 6](#_Toc31184780)

[REPLs 6](#_Toc31184781)

[CLJ REPLs 7](#_Toc31184782)

[CLJS REPLs 7](#_Toc31184783)

[Figwheel 7](#_Toc31184784)

[Figwheel Main 7](#_Toc31184785)

[Leiningen 7](#_Toc31184786)

[Code Execution - REPL 8](#_Toc31184787)

[Code Execution - Uberjar 8](#_Toc31184788)

[Code Execution - lein run 8](#_Toc31184789)

[Code Execution - lein trampoline run 8](#_Toc31184790)

[Test Execution 8](#_Toc31184791)

[Test Selectors 8](#_Toc31184792)

[Profiles 8](#_Toc31184793)

[Plugins 9](#_Toc31184794)

[Ultra 9](#_Toc31184795)

[Lein-checkall 9](#_Toc31184796)

[Lein-check 9](#_Toc31184797)

[Lein-kibit 9](#_Toc31184798)

[Lein-eastwood 10](#_Toc31184799)

[Lein-bikeshed 10](#_Toc31184800)

[Lein-check-namespace-decls 10](#_Toc31184801)

[Lein-cljfmt 10](#_Toc31184802)

[Miscellaneous Developer Tools 11](#_Toc31184803)

[Boot 11](#_Toc31184804)

[Joker - a linter for ClojureScript 12](#_Toc31184805)

[REBL 12](#_Toc31184806)

[CLJS DevTools 12](#_Toc31184807)

[Multiple Dev-Tool Operation Strategies 12](#_Toc31184808)

[Custom (-main) 12](#_Toc31184809)

[REPL Invocation 12](#_Toc31184810)

[Specific to REBL (nREBL.Middleware) 12](#_Toc31184811)

[Base Editors and Associated Extensions 12](#_Toc31184812)

[VS Code 13](#_Toc31184813)

[Atom 13](#_Toc31184814)

[Lisp-Paredit 13](#_Toc31184815)

[Emacs/CIDER 13](#_Toc31184816)

[Light Table 13](#_Toc31184817)

[IntelliJ 13](#_Toc31184818)

[Sublime Text 13](#_Toc31184819)

[Brackets 14](#_Toc31184820)

[Dev Environment and Workflow Analysis 14](#_Toc31184821)

[Development Workflows Kinds 14](#_Toc31184822)

[Traditional Dev-Compile-Deploy-Test Workflow 14](#_Toc31184823)

[REPL driven Workflow 14](#_Toc31184824)

[“Literate Interactive Coding” Workflow 14](#_Toc31184825)

[Dev Environment Workflow Setups 14](#_Toc31184826)

[Atom and ProtoREPL 15](#_Toc31184827)

[Atom, Chlorine, and Cognitect's REBL 15](#_Toc31184828)

[Web Specific Stack Development 16](#_Toc31184829)

[ReFrame Tracing 16](#_Toc31184830)

[Re-Frame-10x - Tracing for ReFrame using X-Ray 16](#_Toc31184831)

[Advanced aspects of Clojure Development 16](#_Toc31184832)

[Testing 16](#_Toc31184833)

[Performance Analysis 16](#_Toc31184834)

[Environment related issues and resolutions 17](#_Toc31184835)

[Browser Caching 17](#_Toc31184836)

[Resources 18](#_Toc31184837)

[Written Web Content Resources 18](#_Toc31184838)

[ClojureScript - Atom 18](#_Toc31184839)

[A Clojure Style Guide 18](#_Toc31184840)

[Leiningen Tutorial 18](#_Toc31184841)

[YouTube Videos and Presentations 18](#_Toc31184842)

[Bruce Hauman, Figwheel, Literate Interactive Coding 19](#_Toc31184843)

[GitHub Repositories and Pages 19](#_Toc31184844)

[REBL with Leiningen 19](#_Toc31184845)

# 

# 

# Overview

The purpose of this project is to evaluate several Clojure Workflow Environments as well as to document their setup, requirements, and then any pros and cons as well as perhaps any interesting or unique features or tricks enabled by each environment combination.

## Demo-Application as Common Analysis Target

In order to understand the features and functionality offered by each development environment, it is useful to have a common application that has a uniform set of features and artifacts within its codebase. Differences in a codebase are therefore nonexistent and prevented from influencing the behavior of our Dev-Env/Workflows. The application should be both small so that it can be developed, edited, bugged and debugged quickly. In addition, it should mirror some artifacts of the Videra/Nomi codebase. It should be a web-app using ClojureScript with Re-Agent, Re-Frame, etc. Additionally, it should use Datomic - perhaps, especially if we are to evaluate database development tools, it should be able to easily switch, and be configured for all product offerings of Datomic (on-prem, cloud, ions). For this purpose, we will create a fast, small ToDo-App Manager that is database backed. The repository for this codebase is here:

<https://github.com/clojure-land/CLJ-Dev-Environments-and-Workflows-AppAna>

Alternatively to building a new app -- although I feel that this is ideal as it would produce a simplified prototypical instance of our platform stack -- as an alternative to doing that, this repository is available and was in-fact created for testing out and comparing various Clojure Development environments. <https://github.com/dking1286/joyful_clojure>

## Clojure Tools and Developer Utilities

We will cover Tools and Utilities first as they will be incorporated into various workflows presented in the Dev Environments section. The toolset’s that we will capture are:

* CLJ-CLI and Deps.EDN
* REPLs
* Figwheel
* Figwheel Main
* Leiningen

Other/Misc:

* Boot
* Joker - a linter for ClojureScript
* Google Closure Tools

In addition, certain development tools become incompatible with each other by the way that they are launched. We will introduce a strategy at the end of this section allowing for multiple tools that require a main-hook to be started simultaneously.

### CLJ-CLI

Comparative to Leiningen, the CLJ set of CLI tools is virtually useless. However, Sean Corfield a Clojure architect and well respected clojurist in the community recommend it in the sense that, it is what his organization has and currently uses without any plans to migrate to Leiningen or Boot or any of the other recent dev-workflow innovations. An additional motivation for understanding this tool would be that, I believe Leiningen as well as Boot both use the CLJ-CLI utilities and setup under the hood to generate a managed set of commands and configuration where they then pass it off to the clojure command line tool. In addition, its inherently the ‘starting-point’ being the first and reference-implementation CLI from Clojure.org.

Within the CLJ-CLI installation set, there are two Clojure binaries, clojure and clj. They both ultimately run and point to clojure but, CLJ uses rlwrap to provide a better REPL experience (however both binaries provide a REPL). The binary artifacts installed by the setup process are as follows:

* bin/clojure - main tool
* bin/clj - wrapper for interactive REPL use (uses rlwrap)
* deps.edn - install level deps.edn file, with some default deps (Clojure, etc) and provider config
* example-deps.edn - commented example that gets copied to <config\_dir>/deps.edn
* libexec/clojure-tools-X.Y.Z.jar - uberjar invoked by clojure to construct classpaths

For the configuration files, generally the setup process copies the one from the install location to the user's home directory and into a sub-directory called .clojure. The following environment variables setup which config file will be read from and used during program execution:

* If $CLJ\_CONFIG is set, then use $CLJ\_CONFIG (explicit override)
* If $XDG\_CONFIG\_HOME is set, then use $XDG\_CONFIG\_HOME/clojure (Freedesktop conventions)
* Else use $HOME/.clojure (most common)

If Clojure is invoked without a deps.edn file, it will automatically create a cache directory and download all necessary dependencies. Which can be set using the following environment variables:

* If $CLJ\_CACHE is set, then use $CLJ\_CACHE (explicit override)
* If $XDG\_CACHE\_HOME is set, then use $XDG\_CACHE\_HOME/clojure (Freedesktop conventions)
* Else use config\_dir/.cpcache (most common)

So, then in the usual case, the cache directory would be in ~/.clojure/.cpcache.

Normally the CLJ tool is primarily a clojure REPL. However, it can perform an application style launch when it is passed the -m parameter and the namespace that the (-main) function is residing in.

#### Deps.EDN

Deps.edn is the traditional clojure way for handling, specifying, and managing dependencies. Here is an example of Deps.edn for the [clj-time](https://github.com/clj-time/clj-time) library. To work with this library, we need to declare it as a dependency so the tool can ensure it has been downloaded and add it to the classpath. The readme in most projects shows the name and version, such as [clj-time "0.14.2"]. Create a deps.edn file to declare the dependency:

|  |
| --- |
| {:deps  {clj-time {:mvn/version "0.14.2"}}} |

Once the REPL is restarted, the CLJ tool will download all necessary packages and their related dependencies and copy them to the local .m2 directory.

|  |
| --- |
| $ clj Downloading: clj-time/clj-time/0.14.2/clj-time-0.14.2.pom from https://clojars.org/repo/ Downloading: clj-time/clj-time/0.14.2/clj-time-0.14.2.jar |

More information on the CLJ/Deps toolset pair can be located here: <https://clojure.org/reference/deps_and_cli>

### REPLs

TDL: Incorporate info from: <https://lambdaisland.com/guides/clojure-repls/clojurescript-repls>

#### CLJ REPLs

#### 

#### CLJS REPLs

### 

### Figwheel

### Figwheel Main

During the recent Clojure meetup, it was suggested to me that we switch over from Figwheel to Figwheel Main as the active development on the Figwheel trunk has stopped and has moved over to Figwheel Main. Those were the words of the SLC Clojurists. I confirmed that that is the message from the Figwheel site with this message on their main page.

|  |
| --- |
| There is a new Figwheel in town!  [Figwheel Main](https://figwheel.org/) is a complete re-write of Figwheel and represents the latest and greatest version of Figwheel. It works great with Leiningen or the new Clojure CLI Tools.  So head over to [Figwheel Main](https://github.com/bhauman/figwheel-main) to give it a try |

However, in reviewing the commit history for both projects, Figwheel appears to be much more active then Figwheel Main - or at least somewhat more active. So perhaps Im am incorrectly understanding the relationship between the two projects but, either way, I will capture some usage scenarios for Figwheel Main at the suggestion of the figwheel authors.

### Leiningen

Leiningen is a Clojure CLI tool that is quite useful and well adopted within the Clojure community. Some of its major features are:

* Code Execution
  + REPL
  + Uberjar
  + Lein run
  + Lein trampoline run
* Test Execution
  + Test Selectors
* Profiles
* Plugins
* Templates
* Misc
  + Project level configuration
  + Repository search
  + Publish to clojars

#### Code Execution - REPL

#### Code Execution - Uberjar

#### Code Execution - lein run

#### Code Execution - lein trampoline run

#### Test Execution

Note that the command, lein test is said by the leiningen development team to be extremely slow to startup and, to only use it for regression testing as a final action before a commit or as part of a CI/CD setup but, not to leverage it as part of the development cycle.

##### Test Selectors

#### Profiles

Profiles are used to add various things into your project map in different contexts. For instance, during lein test runs, the contents of the :test profile, if present, will be merged into your project map.

#### Plugins

Some interesting and cool Leiningen Plugins - <https://github.com/technomancy/leiningen/wiki/plugins>

Below is a set of Leiningen plugins worth looking into.

##### Ultra

<https://github.com/venantius/ultra>. This looks like a cool and useful plugin. For any operations executing from the command line (such as lein run) it adds enhanced output. Here are the features it lists:

* A colorized REPL
* Syntax-highlighted source (from the REPL)
* Clearer test output
* Better stacktraces

<https://blog.venanti.us/ultra/> - blog describing functionality

##### Lein-checkall

Executes four other lein plugins: lein check, lein kibit, lein eastwood, lein bikeshed

* lein check
* lein kibit
* lein eastwood
* lein bikeshed

##### Lein-check

Lein-check is a built-in utility that comes with Leiningen - the documentation from the CLI help says the following: check - Check syntax and warn on reflection. From actually running this, it seems to do what articles on the web suggest - to traverse your local code base and ensure that all referenced code is available on the classpath or other dependency statements and, that the method signatures from the loaded code match the usage from the call sites in the source.

##### Lein-kibit

Lein-kibit is described as follows from the developers site: kibit is a static code analyzer for Clojure that uses [core.logic](https://github.com/clojure/core.logic) to search for patterns of code that could be rewritten in a way more idiomatic to clojure, using a function or macro. For example:

|  |
| --- |
| (if (some test)  (some action)  nil) |

Kibit will suggest using when instead:

|  |
| --- |
| (when (some test)  (some action)) |

This seems useful especially for developers who might not be intimately familiar with the Clojure way of doing things. The above example is illustrative in that, it is the clojure variant of the traditional imperative conditional. Most imperative programmers will think to write it that way, using the if. However, seeing the modification by kibit becomes a kind of learning tool.

##### Lein-eastwood

Eastwood is a Clojure lint tool that uses the [tools.analyzer](https://github.com/clojure/tools.analyzer) and [tools.analyzer.jvm](https://github.com/clojure/tools.analyzer.jvm) libraries to inspect namespaces and report possible problems. One downside to Eastwood is that it only supports Clojure on Java, not ClojureScript.

##### Lein-bikeshed

Bikeshed checks for common coding errors. In looking at the output from a Bikeshed run, it appears to check the following aspects of written code:

* Checking for lines longer than 80 characters
* Checking for lines with trailing whitespace
* Checking for files ending in blank lines
* Checking for redefined var roots in source directories
* Checking whether you keep up with your docstrings
* Checking for arguments colliding with clojure.core functions

##### Lein-check-namespace-decls

Clojure linter that checks that namespace declarations are cleaned the way `clj-refactor` would clean them. Lein-check-namespace-decls Checks the following:

* Namespaces and imports in the ns declaration are sorted
* No unused namespaces are present (configurable)
* :require forms are used instead of :use
* The top-level forms inside the ns form follow the [conventional order](https://github.com/bbatsov/clojure-style-guide#comprehensive-ns-declaration): :gen-class, :refer-clojure, :require, :import
* Namespaces in :require use shared prefixes (e.g. (:require [clojure [string :as str] [data :as data]]))

##### Lein-cljfmt

lein-cljfmt is a tool for formatting Clojure code [idiomatically](https://github.com/bbatsov/clojure-style-guide) (based on the Clojure Style Guide). It can turn something like this:

|  |
| --- |
| ( let [x 3  y 4]  (+ (\* x x  )(\* y y)  )) |

Into well formatted Clojure code such as this:

|  |
| --- |
| (let [x 3  y 4]  (+ (\* x x) (\* y y))) |

### Miscellaneous Developer Tools

There are a plethora of exceptional Development tools created by the Clojure community. In this section, we will capture some of the seemingly more useful ones.

#### Boot

|  |
| --- |
| brian@BRIANs-MacBook-Pro:~$ boot  Downloading https://github.com/boot-clj/boot/releases/download/2.7.2/boot.jar... Running for the first time, BOOT\_VERSION not set: updating to latest. WARNING: An illegal reflective access operation has occurred WARNING: Illegal reflective access by dynapath.defaults**$fn\_\_1516$fn\_\_1517** (file:/Users/brian/.boot/cache/lib/2.7.2/aether.uber.jar) to method java.net.URLClassLoader.addURL(java.net.URL) WARNING: Please consider reporting this to the maintainers of dynapath.defaults**$fn\_\_1516$fn\_\_1517** WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations WARNING: All illegal access operations will be denied in a future release Retrieving maven-metadata.xml from https://repo.clojars.org/ Retrieving boot-2.8.3.pom from https://repo.clojars.org/ (2k) Retrieving boot-2.8.3.jar from https://repo.clojars.org/ (2k) Retrieving clojure-1.8.0.jar from https://repo1.maven.org/maven2/ (3538k) #http://boot-clj.com #Fri Jun 28 06:37:38 MDT 2019 BOOT\_VERSION=2.8.3 BOOT\_CLOJURE\_VERSION=1.8.0 BOOT\_CLOJURE\_NAME=org.clojure/clojure |

#### Joker - a linter for ClojureScript

It is good to have found Joker - the majority of the Clojure Linters that I have looked at so far do not function in any way with ClojureScript.

#### REBL

<http://rebl.cognitect.com/>

#### CLJS DevTools

<https://github.com/binaryage/cljs-devtools>

### Multiple Dev-Tool Operation Strategies

#### Custom (-main)

One option suggested for getting around Development Tools requiring a CLI hook is to create a custom main function that then, in addition to operating the regular application main, calls out to an additional dev-main, perhaps on configuration setting, which then starts a list of configured dev-tools.

#### REPL Invocation

If the method in which the tool is supposed to be initialized is known, it can also be started from any running REPL.

#### Specific to REBL (nREBL.Middleware)

The desire to launch (and thereby use) REBL in ways other than what it was originally written for was the original motivation for this section and an overall investigation into a multi-toolset integration plan. However, while performing this, we have identified the following extension which specifically connects an application to REBL, through nREPL. The project is located at the following URL.

<https://github.com/RickMoynihan/nrebl.middleware>

## Base Editors and Associated Extensions

Note: While this list aims to be comprehensive, we will not likely evaluate every plugin or editor. We have an idea of our ideal editors at this time and will maintain a target on those however, we will give a cursory investigation to everything in this list and anything that strikes us as unique or interesting will be considered for a complete investigation and workflow capture.

* VS Code
* Atom
* Emacs/CIDER
* Light Table
* IntelliJ
* Sublime Text
* Brackets

### VS Code

### Atom

#### Lisp-Paredit

<https://atom.io/packages/lisp-paredit>

### Emacs/CIDER

cider -

cider-decompile - decompilation extension for cider

cider-eval-sexp-fu - Briefly highlights an evaluated sexp

cider-hydra - Hydras for CIDER

### Light Table

### IntelliJ

* Cursive
* La Clojure

### Sublime Text

### Brackets

Selected Combinations for analysis

# Dev Environment and Workflow Analysis

In this section we will arrive at the purpose of this document - to capture and understand and then therefore compare, a variety of different development workflows and, the environments that support them. Once we’ve identified several workflows that stand out, we can perform exhaustive discovery whereby we begin to establish a total mastery of a select set of the environments that we believe will enable the greatest degree of individual programming productivity.

## Development Workflows Kinds

* Traditional Dev-Compile-Deploy-Test
* REPL driven Workflow
* “Literate Interactive Coding” Workflow (AKA: on-the-fly-programming, just in time programming, conversational programming)

### Traditional Dev-Compile-Deploy-Test Workflow

### REPL driven Workflow

### “Literate Interactive Coding” Workflow

## Dev Environment Workflow Setups

Atom

Atom Packages

* Proto REPL
* Sayid
* Proto REPL Sayid

### Atom and ProtoREPL

### 

### Atom, Chlorine, and Cognitect's REBL

# Web Specific Stack Development

## ReFrame Tracing

### Re-Frame-10x - Tracing for ReFrame using X-Ray

<https://github.com/Day8/re-frame-10x>

<https://github.com/Day8/re-frame-debux>

<https://github.com/flexsurfer/re-frisk>

# Advanced aspects of Clojure Development

## Testing

## Performance Analysis

# 

# Environment related issues and resolutions

## Browser Caching

Chrome Caching Disable -- insert screen cast link

# 

# 

# Resources

## Written Web Content Resources

This includes Guides, Blogs, Articles, important StackOverflow conversations

### ClojureScript - Atom

A resource from the Atom editor for installing and configuring various Clojure and CLJS plugins.

<https://clojurescript.org/tools/atom>

### A Clojure Style Guide

The formatting outlined in this guide is implemented by several of the Clojure Development tools.

<https://github.com/bbatsov/clojure-style-guide>

The distributed version is available at: <https://guide.clojure.style/>

### Leiningen Tutorial

<https://github.com/technomancy/leiningen/blob/stable/doc/TUTORIAL.md>

<https://medium.com/@daniel.oliver.king/a-clojure-development-environment-that-gets-out-of-your-way-c11e6711ead3>

The Ultimate Guide To Clojure REPLs  
<https://lambdaisland.com/guides/clojure-repls>

How to use Clojure 1.10 pREPL to connect to a remote server (WIP)  
<https://blog.jakubholy.net/how-to-use-clojure-1.10-prepl/>

## YouTube Videos and Presentations

Proto REPL, a New Clojure Development and Visualization Tool - Jason Gilman

<https://www.youtube.com/watch?v=buPPGxOnBnk>

### Bruce Hauman, Figwheel, Literate Interactive Coding

Interactive programming Flappy Bird in ClojureScript

<https://www.youtube.com/watch?v=KZjFVdU8VLI>

Bruce Hauman - Developing ClojureScript With Figwheel

<https://www.youtube.com/watch?v=j-kj2qwJa_E>

"Literate interactive coding: Devcards" by Bruce Hauman

<https://www.youtube.com/watch?v=G7Z_g2fnEDg>

Devcards Blog live coding example

<https://www.youtube.com/watch?v=DPHkBp9Mkzk>

## GitHub Repositories and Pages

### REBL with Leiningen

This repository demonstrates how to integrate Cognitect’s REBL with the Leiningen Build tool.

<https://github.com/eccentric-j/lein-rebl-example>