
rebel-readline

clojars [com.bhauman/rebel-readline "0.1.4"]

clojars [com.bhauman/rebel-readline-cljs "0.1.4"]

A terminal readline library for Clojure Dialects

```
#_=> ;; - Syntax highlighting
#_=> ;; - Auto indenting on newline
#_=> ;; - Eldoc type arglist display
user=> (for [x (range 5)
#_=>      y (range 10)]
#_=>    [x y])
([0 0] [0 1] [0 2] [0 3] [0 4] [0 5] [0 6] [0 7] [0 8] [0 9] [1 0] [1 1] [1 2] [1 3] [1 4] [1 5]
 [1 6] [1 7] [1 8] [1 9] [2 0] [2 1] [2 2] [2 3] [2 4] [2 5] [2 6] [2 7] [2 8] [2 9] [3 0] [3
 1] [3 2] [3 3] [3 4] [3 5] [3 6] [3 7] [3 8] [3 9] [4 0] [4 1] [4 2] [4 3] [4 4] [4 5] [4 6] [4
 7] [4 8] [4 9])
user=> ;; Ctrl-X Ctrl-D will give you the documentation for the current var
user=> (for [x (range 5)
#_=>      y (range 10)]
#_=>    [x y])
([0 0] [0 1] [0 2] [0 3] [0 4] [0 5] [0 6] [0 7] [0 8] [0 9] [1 0] [1 1] [1 2] [1 3] [1 4] [1 5]
 [1 6] [1 7] [1 8] [1 9] [2 0] [2 1] [2 2] [2 3] [2 4] [2 5] [2 6] [2 7] [2 8] [2 9] [3 0] [3
 1] [3 2] [3 3] [3 4] [3 5] [3 6] [3 7] [3 8] [3 9] [4 0] [4 1] [4 2] [4 3] [4 4] [4 5] [4 6] [4
 7] [4 8] [4 9])
user=> ;; What if the documentation is too long, you ask?
#_=> ;; a interactive less view of the docs allows scrolling
user=> (de
```

Why create a terminal readline library?

<https://github.com/bhauman/rebel-readline/blob/master/rebel-readline/doc/intro.md>

Important note!!!

The line reader will attempt to manipulate the terminal that initiates the JVM process. For this reason it is important to start your JVM in a terminal.

That means you should launch your Java process using the

- the java command
- the Clojure `clojure` tool (without readline support)
- lein trampoline
- boot - would need to run in boot's worker pod

Launching the terminal readline process from another Java process will not work.

It's best to not launch this readline behind other readline tools like `rlwrap`.

Quick try

Clojure tools If you want to try this really quickly install the Clojure CLI tools and then invoke this:

```
1 clojure -Sdeps "{:deps {com.bhauman/rebel-readline {:mvn/version\n\"0.1.4\"}}}" -m rebel-readline.main
```

That should start a Clojure REPL that takes its input from the Rebel readline editor.

Note that I am using the `clojure` command and not the `clj` command because the latter wraps the process with another readline program (`rlwrap`).

Alternatively you can specify an alias in your `$HOME/.clojure/deps.edn`

```
1 {
2   ...
3   :aliases {:rebel {:extra-deps {com.bhauman/rebel-readline {:mvn/
4     version \"0.1.4\"}}
5     :main-opts  [\"-m\" \"rebel-readline.main\"]}}
```

And then run with a simpler:

```
1 clojure -M:rebel
```

Leiningen Add `[com.bhauman/rebel-readline \"0.1.4\"]` to the dependencies in your `project.clj` then start a REPL like this:

```
1 lein trampoline run -m rebel-readline.main
```

Alternatively, you can add `rebel-readline` globally to `$HOME/.lein/profiles.clj`

```
1 {
2   ...
3   :user {:dependencies [[com.bhauman/rebel-readline \"0.1.4\"]]}
4 }
```

Then you can call

```
1 lein trampoline run -m rebel-readline.main
```

To make this less verbose you can use an alias in your `project.clj`:

```
1 {
2   ...
3   :aliases {"rebl" ["trampoline" "run" "-m" "rebel-readline.main"]}
4 }
```

Alternatively, you can do this globally in `$HOME/.lein/profiles.clj`:

```
1 {
2   ...
3   :user {:aliases {"rebl" ["trampoline" "run" "-m" "rebel-readline.main"
4     ]}}
4 }
```

Now you can start a rebel-readline REPL with `lein rebl`.

Boot

```
1 boot -d com.bhauman/rebel-readline call -f rebel-readline.main/-main
```

Clone repo Clone this repo and then from the `rebel-readline` sub-directory typing `lein trampoline run -m rebel-readline.main` will get you into a Clojure REPL with the read-line editor working.

Note that `lein run -m rebel-readline.main` will not work! See above.

How do I default to vi bindings?

In `~/.clojure/rebel_readline.edn` put

```
1 {:key-map :viins}
```

Config

In `~/.clojure/rebel_readline.edn` you can provide a map with the following options:

```
1 :key-map          - either :viins or :emacs. Defaults to :emacs
2
3 :color-theme       - either :light-screen-theme or :dark-screen-theme
4
5 :highlight         - boolean, whether to syntax highlight or not.
6                     Defaults to true
7
8 :completion        - boolean, whether to complete on tab. Defaults to
9                     true
```

```
9  :eldoc          - boolean, whether to display function docs as you
    type.
10                      Defaults to true
11
12  :indent         - boolean, whether to auto indent code on newline.
    Defaults to true
13
14  :redirect-output - boolean, rebinds root *out* during read to protect
    linereader
15                      Defaults to true
16
17  :key-bindings   - map of key-bindings that get applied after all other
    key                bindings have been applied
18
```

Key binding config You can configure key bindings in the config file, but your mileage may vary.

Example:

```
1  {
2  ...
3  :key-bindings { :emacs [["^D" :clojure-doc-at-point]]
4                  :viins [["^J" :clojure-force-accept-line]] }
5  }
```

Serialized keybindings are tricky and the keybinding strings are translated with [org.jline.keymap.KeyMap/translate](#) which is a bit peculiar in how it translates things.

If you want literal characters you can use a list of chars or ints i.e (`\\ \\d`) instead of the serialized key names. So you can use (`4 4`) in place of `"^D^D"`.

The best way to look up the available widget names is to use the `:repl/key-bindings` command at the REPL prompt.

Note: I have found that JLine handles control characters and alphanumeric characters quite well but if you want to bind special characters you shouldn't be surprised if it doesn't work.

Quick Lay of the land

You should look at `rebel-readline.clojure.main` and `rebel-readline.core` to give you top level usage information.

The core of the functionality is in `rebel-readline.clojure.line-reader` everything else is just support.

Quick Usage

These are some quick examples demonstrating how to use the rebel-readline API.

The main way to utilize this readline editor is to replace the `clojure.main/repl-read` behavior in `clojure.main/repl`.

The advantage of doing this is that it won't interfere with the input stream if you are working on something that needs to read from `*in*`. This is because the line-reader will only be engaged when the REPL loop is reading.

Example:

```
1 (rebel-readline.core/with-line-reader
2   (rebel-readline.clojure.line-reader/create
3     (rebel-readline.clojure.service.local/create))
4   (clojure.main/repl
5     :prompt (fn [])) ;; prompt is handled by line-reader
6     :read (rebel-readline.clojure.main/create-repl-read)))
```

Another option is to just wrap a call you your REPL with `rebel-readline.core/with-readline-in` this will bind `*in*` to an input-stream that is supplied by the line reader.

```
1 (rebel-readline.core/with-readline-in
2   (rebel-readline.clojure.line-reader/create
3     (rebel-readline.clojure.service.local/create))
4   (clojure.main/repl :prompt (fn[])))
```

Or with a fallback:

```
1 (try
2   (rebel-readline.core/with-readline-in
3     (rebel-readline.clojure.line-reader/create
4       (rebel-readline.clojure.service.local/create))
5     (clojure.main/repl :prompt (fn[])))
6   (catch clojure.lang.ExceptionInfo e
7     (if (-> e ex-data :type (= :rebel-readline.jline-api/bad-terminal))
8       (do (println (.getMessage e))
9           (clojure.main/repl))
10      (throw e))))
```

Services

The line reader provides features like completion, documentation, source, apropos, eval and more. The line reader needs a Service to provide this functionality.

When you create a `rebel-readline.clojure.line-reader` you need to supply this service.

The more common service is the `rebel-readline.services.clojure.local` which uses the local clojure process to provide this functionality and its a good example of how a service works.

https://github.com/bhauman/rebel-readline/blob/master/rebel-readline/src/rebel_readline/clojure/service/local.clj

In general, it's much better if the service is querying the Clojure process where the eventual REPL eval takes place.

However, the service doesn't necessarily have to query the same environment that the REPL is using for evaluation. All the editing functionality that rebel readline provides works without an environment to query. And the apropos, doc and completion functionality is still sensible when you provide those abilities from the local clojure process.

This could be helpful when you have a Clojure REPL process and you don't have a Service for it. In this case you can just use a `clojure.service.local` or a `clojure.service.simple` service. If you do this you can expect less than optimal results but multi-line editing, syntax highlighting, auto indenting will all work just fine.

Key-bindings

Bindings of interest

- Ctrl-C => aborts editing the current line
- Ctrl-D at the start of a line => sends an end of stream message which in most cases should quit the REPL
- TAB => word completion or code indent if the cursor is in the whitespace at the start of a line
- Ctrl-X_Ctrl-D => Show documentation for word at point
- Ctrl-X_Ctrl-S => Show source for word at point
- Ctrl-X_Ctrl-A => Show apropos for word at point
- Ctrl-X_Ctrl-E => Inline eval for SEXP before the point

You can examine the key-bindings with the `:repl/key-bindings` command.

Commands

There is a command system. If the line starts with a "repl" namespaced keyword then the line-reader will attempt to interpret it as a command.

Type `:repl/help` or `:repl` TAB to see a list of available commands.

You can add new commands by adding methods to the `rebel-readline.commands/command` multimethod. You can add documentation for the command by adding a method to the `rebel-readline.commands/command-doc` multimethod.

CLJS

See <https://github.com/bhauman/rebel-readline/tree/master/rebel-readline-cljs>

nREPL, SocketREPL, pREPL?

Services have not been written for these REPLs yet!!

But you can use the `rebel-readline.clojure.service.simple` service in the meantime.

Contributing

Please contribute!

I'm trying to mark issues with `help wanted` for issues that I feel are good opportunities for folks to help out. If you want to work on one of these please mention it in the issue.

If you do contribute:

- if the change isn't small please file an issue before a PR.
- please put all PR changes into one commit
- make small grokable changes. Large changes are more likely to be ignored and or used as a starting issue for exploration.
- break larger solutions down into a logical series of small PRs
- mention it at the start, if you are filing a PR that is more of an exploration of an idea

I'm going to be more open to repairing current behavior than I will be to increasing the scope of rebel-readline.

I will have a preference for creating hooks so that additional functionality can be layered on with libraries.

If you are wanting to contribute but don't know what to work on reach out to me on the clojurians slack channel.

License

Copyright © 2018 Bruce Hauman

Distributed under the Eclipse Public License either version 1.0 or (at your option) any later version.