cli4clj
Easing the Implementation of Interactive Command Line Interfaces in Clojure for "Everyone"

Ruediger Gad

Terma GmbH, Space, Darmstadt, Germany

:clojureD
2019-02-23

What? & Why?

- Interactive Command Line Interfaces (CLIs)
- Clojure REPL
  - Powerful +
  - Requires Clojure Knowledge —?
  - Typical Users: Developers
- cli4clj
  - “CLIs for Everyone”
  - Ease Use & Implementation

Developer Perspective: cli4clj Configuration/Implementation Example

```
(ns cli4clj.minimal-example (:gen-class)
  (:require (cli4clj [cli :as cli])))
(defn divide [x y] (/ x y)) ;;; Used for example below.

(defn -main [& args]
  (cli/start-cli
    {:cmds
      {:test-cmd {:fn #(println "This is a test.")
                  :short-info "Test Command"
                  :long-info "Prints a test message."}
       :add {:fn (fn [summand1 summand2] (+ summand1 summand2))
             :completion-hint "Enter two values to add."}
       :divide {:fn divide}}
:allow-eval true, :alternate-scrolling (some #(= % "alt") args)}))
```

Developer Perspective: cli4clj Configuration/Implementation Example

```
(ns cli4clj.minimal-example (:gen-class)
  (:require (cli4clj [cli :as cli])))
(defn divide [x y] (/ x y)) ;;; Used for example below.

(defn -main [& args]
  (cli/start-cli
    {:cmds
      {:test-cmd {:fn #(println "This is a test.")
                  :short-info "Test Command"
                  :long-info "Prints a test message."}
       :add {:fn (fn [summand1 summand2] (+ summand1 summand2))
             :completion-hint "Enter two values to add."}
       :divide {:fn divide}}
:allow-eval true, :alternate-scrolling (some #(= % "alt") args)}))
```

Developer Perspective: cli4clj Configuration/Implementation Example

```
(ns cli4clj.minimal-example (:gen-class)
  (:require (cli4clj [cli :as cli])))
(defn divide [x y] (/ x y)) ;;; Used for example below.

(defn -main [& args]
  (cli/start-cli
    {:cmds
      {:test-cmd {:fn #(println "This is a test.")
                  :short-info "Test Command"
                  :long-info "Prints a test message."}
       :add {:fn (fn [summand1 summand2] (+ summand1 summand2))
             :completion-hint "Enter two values to add."}
       :divide {:fn divide}}
:allow-eval true, :alternate-scrolling (some #(= % "alt") args)}))
```

Developer Perspective: cli4clj Configuration/Implementation Example

```
(ns cli4clj.minimal-example (:gen-class)
  (:require (cli4clj [cli :as cli])))
(defn divide [x y] (/ x y)) ;;; Used for example below.

(defn -main [& args]
  (cli/start-cli
    {:cmds
      {:test-cmd {:fn #(println "This is a test.")
                  :short-info "Test Command"
                  :long-info "Prints a test message."}
       :add {:fn (fn [summand1 summand2] (+ summand1 summand2))
             :completion-hint "Enter two values to add."}
       :divide {:fn divide}}
:allow-eval true, :alternate-scrolling (some #(= % "alt") args)}))
```

Developer Perspective: cli4clj Configuration/Implementation Example

```
(ns cli4clj.minimal-example (:gen-class)
  (:require (cli4clj [cli :as cli])))
(defn divide [x y] (/ x y)) ;;; Used for example below.

(defn -main [& args]
  (cli/start-cli
    {:cmds
      {:test-cmd {:fn #(println "This is a test.")
                  :short-info "Test Command"
                  :long-info "Prints a test message."}
       :add {:fn (fn [summand1 summand2] (+ summand1 summand2))
             :completion-hint "Enter two values to add."}
       :divide {:fn divide}}
:allow-eval true, :alternate-scrolling (some #(= % "alt") args)}))
```

Developer Perspective: cli4clj Configuration/Implementation Example

```
(ns cli4clj.minimal-example (:gen-class)
  (:require (cli4clj [cli :as cli])))
(defn divide [x y] (/ x y)) ;;; Used for example below.

(defn -main [& args]
  (cli/start-cli
    {:cmds
      {:test-cmd {:fn #(println "This is a test.")
                  :short-info "Test Command"
                  :long-info "Prints a test message."}
       :add {:fn (fn [summand1 summand2] (+ summand1 summand2))
             :completion-hint "Enter two values to add."}
       :divide {:fn divide}}
:allow-eval true, :alternate-scrolling (some #(= % "alt") args)}))
```

Developer Perspective: cli4clj Configuration/Implementation Example

```
(ns cli4clj.minimal-example (:gen-class)
  (:require (cli4clj [cli :as cli])))
(defn divide [x y] (/ x y)) ;;; Used for example below.

(defn -main [& args]
  (cli/start-cli
    {:cmds
      {:test-cmd {:fn #(println "This is a test.")
                  :short-info "Test Command"
                  :long-info "Prints a test message."}
       :add {:fn (fn [summand1 summand2] (+ summand1 summand2))
             :completion-hint "Enter two values to add."}
       :divide {:fn divide}}
:allow-eval true, :alternate-scrolling (some #(= % "alt") args)}))
```

Developer Perspective: cli4clj Configuration/Implementation Example

```
(ns cli4clj.minimal-example (:gen-class)
  (:require (cli4clj [cli :as cli])))
(defn divide [x y] (/ x y)) ;;; Used for example below.

(defn -main [& args]
  (cli/start-cli
    {:cmds
      {:test-cmd {:fn #(println "This is a test.")
                  :short-info "Test Command"
                  :long-info "Prints a test message."}
       :add {:fn (fn [summand1 summand2] (+ summand1 summand2))
             :completion-hint "Enter two values to add."}
       :divide {:fn divide}}
:allow-eval true, :alternate-scrolling (some #(= % "alt") args)}))
```

Application User Perspective: “Basic Commands”

```
cli#
cli# test-cmd
This is a test.
cli# add 1 2
3
cli# divide 2 3
2/3
```

Application User Perspective: “Basic Commands”

```
cli#
cli# test-cmd
This is a test.
cli# add 1 2
3
cli# divide 2 3
2/3
```

Application User Perspective: “Basic Commands”

```
cli#
cli# test-cmd
This is a test.
cli# add 1 2
3
cli# divide 2 3
2/3
```

Application User Perspective: Help

```
cli# help
add -- Arguments: [[summand1 summand2]]
...
divide -- Arguments: [[x y]]
...
test-cmd
        Test Command
        Prints a test message.
```

Application User Perspective: Help

```
cli# help
add -- Arguments: [[summand1 summand2]]
...
divide -- Arguments: [[x y]]
...
test-cmd
        Test Command
        Prints a test message.
```

Application User Perspective: Tab Completion

```
cli# <TAB>
...   add   ...  divide  ...  help  ...  test-cmd
cli# a<TAB>
cli# add
cli# add <TAB>
Arguments: [[summand1 summand2]]   Enter two values to add.
...
```

Application User Perspective: Tab Completion

```
cli# <TAB>
...   add   ...  divide  ...  help  ...  test-cmd
cli# a<TAB>
cli# add
cli# add <TAB>
Arguments: [[summand1 summand2]]   Enter two values to add.
...
```

Application User Perspective: Tab Completion

```
cli# <TAB>
...   add   ...  divide  ...  help  ...  test-cmd
cli# a<TAB>
cli# add
cli# add <TAB>
Arguments: [[summand1 summand2]]   Enter two values to add.
...
```

Application User Perspective: Clojure Interoperability

```
cli# (reduce add [1 7 0 1])
9
cli# (map divide [1 7 0 1] [1 8 6 4])
(1 7/8 0 1/4)
```

Application User Perspective: Alternate Scrolling Mode

```
> test-cmd
This is a test.
> add 1 2
3
> divide 2 3
2/3
```



```
cli#
```

Application User Perspective: Alternate Scrolling Mode

```
> test-cmd
This is a test.
> add 1 2
3
> divide 2 3
2/3
```

```
cli#
```

Application User Perspective: Alternate Scrolling Mode

```
> test-cmd
This is a test.
> add 1 2
3
> divide 2 3
2/3
```

```
cli#
```

Application User Perspective: Alternate Scrolling Mode

```
> test-cmd
This is a test.
> add 1 2
3
> divide 2 3
2/3
```

```
cli#
```

Developer Perspective: Automated Testing

```
(ns cli4clj.test.minimal-example
  (:require
    (clojure [test :as test])
    (cli4clj [cli-tests :as cli-tests]
             [minimal-example :as mini-example])))

(test/deftest example-test
  (let [test-cmd-input ["add 1 2"
                        "divide 3 2"]
        out-string (cli-tests/test-cli-stdout
                     #(mini-example/-main "") test-cmd-input)]
    (test/is (=
               (cli-tests/expected-string ["3" "3/2"])
               out-string))))
```

Developer Perspective: Automated Testing

```
(ns cli4clj.test.minimal-example
  (:require
    (clojure [test :as test])
    (cli4clj [cli-tests :as cli-tests]
             [minimal-example :as mini-example])))

(test/deftest example-test
  (let [test-cmd-input ["add 1 2"
                        "divide 3 2"]
        out-string (cli-tests/test-cli-stdout
                     #(mini-example/-main "") test-cmd-input)]
    (test/is (=
               (cli-tests/expected-string ["3" "3/2"])
               out-string))))
```

Developer Perspective: Automated Testing

```
(ns cli4clj.test.minimal-example
  (:require
    (clojure [test :as test])
    (cli4clj [cli-tests :as cli-tests]
             [minimal-example :as mini-example])))

(test/deftest example-test
  (let [test-cmd-input ["add 1 2"
                        "divide 3 2"]
        out-string (cli-tests/test-cli-stdout
                    #(mini-example/-main "") test-cmd-input)]
    (test/is (=
              (cli-tests/expected-string ["3" "3/2"])
              out-string))))
```

Developer Perspective: Automated Testing

```
(ns cli4clj.test.minimal-example
  (:require
    (clojure [test :as test])
    (cli4clj [cli-tests :as cli-tests]
             [minimal-example :as mini-example])))

(test/deftest example-test
  (let [test-cmd-input ["add 1 2"
                        "divide 3 2"]
        out-string (cli-tests/test-cli-stdout
                     #(mini-example/-main "") test-cmd-input)]
    (test/is (=
               (cli-tests/expected-string ["3" "3/2"])
               out-string))))
```

More

- Persistent History
- Aliases (Shortcuts)
- Customizable
- “Embedded CLIs”

End

https://github.com/ruedigergad/cli4clj

https://ruedigergad.com/category/libs/cli4clj

Thank you very much for your attention!

Questions?

Ruediger Gad
Terma GmbH, Space
Darmstadt, Germany

ruga@terma.com
r.c.g@gmx.de