Clojure Cheat Sheet

Functions and Macros Overview

Documentation

`doc find-doc source` (clojure.contrib.repl-utils)

Data Structures

Numbers

Computation:	`+ - * / inc dec quot rem min max rationalize`
Comparison:	`== < > <= >= zero? pos? neg?`
Bitwise ops:	`bit-{and, or, xor, not, flip, set, shift-right, shift-left, and-not, clear, test}`
Integer ops:	`odd even`
Random num:	`rand rand-int`
BigInt ops:	`with-precision`
Unchecked:	`unchecked-{add, dec, divide, inc, multiply, negate, remainder,subtract}`

Strings

`str string? pr-str prn-str print-str println-str with-out-str`

Characters

`char char-name-string char-escape-string`

Lists

Create a list:	`'() list list*`
List as stack:	`peek pop`
Examine a list:	`list?`

Vectors

Create a vector:	`[] vector vec`
Examine a vector:	`get nth peek rseq vector?`
'Change' a vector:	`assoc pop subvec replace`

Maps

Create:	`{} hash-map sorted-map sorted-map-by`
'Change':	`assoc dissoc select-keys merge merge-with zipmap`
Examine:	`get contains? find keys vals map?`
Entry:	`key val`

StructMaps

Setup:	`create-struct defstruct accessor`
Individual:	`struct-map struct`

ArrayMaps

Create:	`array-map`

Sets

Create a set:	`#{} hash-set sorted-set set get conj disj contains? count seq`
Operations:	`union difference intersection`
Rel. algebra:	`select index rename join project map-invert rename-keys`

Misc.

Collections:	`count conj seq`
Keywords:	`keyword keyword?`
Symbols:	`symbol symbol? gensym`

Sequences

Seq in, Seq out

Get shorter seq:	`distinct filter remove for`
Get longer seq:	`cons concat lazy-cat mapcat cycle interleave interpose`
Head-items missing:	`rest frest rrest drop drop-while nthrest for`
Tail-items missing:	`take take-nth take-while butlast drop-last for`
Rearrangment:	`reverse sort sort-by`
Nested seqs:	`split-at split-with partition`
Process each item:	`map pmap mapcat for replace seque`

Using a Seq

Extract special item:	`first ffirst rfirst second nth when-first last`
Construct a coll:	`zipmap into reduce set vec into-array to-array-2d`
Pass items to fn:	`apply`
Get a boolean:	`empty? not-empty some not-any? every? not-every? reduce seq? counted? sorted? contains? reversible? sequential? associative?`
Search a seq:	`some filter`
Force evaluation:	`doseq dorun doall`

Creating a Lazy Seq

From collection:	`seq vals keys rseq subseq rsubseq`
From producer fn:	`lazy-seq repeatedly iterate`
From constant:	`repeat replicate range`
From other objects:	`line-seq resultset-seq re-seq tree-seq file-seq xml-seq iterator-seq enumeration-seq`

Reader Macros

`'`	Quote 'form → (quote form)
`\`	Character literal
`;`	Single line comment
`^`	Meta ^form → (meta form)
`@`	Deref @form → (deref form)
`` ` ``	Syntax-quote
`~`	Unquote
`~@`	Unquote-splicing
`#"`p`"`	Regex Pattern p
`#^`	meta data
`#'`	Var quote #'x → (var x)
`#()`	#(...) → (fn [args] (...))

Special Forms

`def if do let quote var fn loop recur throw try monitor-enter monitor-exit`

Macros

Creation:	`defmacro definline macroexpand-1 macroexpand`
Branching:	`and or when when-not when-let when-first if-not if-let cond condp`
Looping:	`for doseq dotimes while`
Arranging code:	`.. doto ->`
Dynamic scopes:	`binding locking time with-in-str with-local-vars with-open with-out-str with-precision`
Lazy things:	`lazy-cat lazy-cons delay`
Documenting:	`assert comment doc`

Multimethods

Creation:	`defmulti defmethod`
Remove:	`remove-method`
Prefer:	`prefer-method`
Relationship:	`derive isa? parents ancestors descendants make-hierarchy`

Vars and global environment

Def variants:	`defn defn- definline defmacro defmethod defmulti defonce defstruct`
Interned vars:	`declare intern binding find-var var`
Var objects:	`with-local-vars var-get var-set alter-var-root var?`
Var validators:	`set-validator get-validator`
Var metadata:	`doc find-doc test`

Refs and Transactions

Create a ref:	`ref`
Examine a ref:	`deref`
Transaction macros:	`dosync io!`
In transaction only:	`ensure ref-set alter commute`
Ref validators:	`set-validator get-validator`

Agents and Asynchronous Actions

Creation:	`agent`
Examine:	`agent-errors`
Change state:	`send send-off clear-agent-errors`
Block waiting:	`await await-for`
Ref validators:	`set-validator get-validator`
Watchers:	`add-watch remove-watch`
Thread handling:	`shutdown-agents`

Loading

Loading libs:	`require use`
Listing loaded libs:	`loaded-libs`
Loading misc:	`load load-file load-reader load-string`

Printing

Print to *out*:	`pr prn print println newline`
Print to string:	`pr-str prn-str print-str println-str with-out-str`

Namespace

Current:	`*ns*`
Creating:	`in-ns ns create-ns`
Switching:	`in-ns ns create-ns`
Adding:	`alias def import intern refer`
Finding:	`all-ns find-ns`
Examining:	`ns-name ns-aliases ns-map ns-interns ns-publics ns-refers ns-imports`
From symbol:	`resolve ns-resolve namespace`
Removing:	`ns-unalias ns-unmap remove-ns`

Java Interoperation

Misc:	`. .. Classname/ Classname. new bean comparator enumeration-seq import iterator-seq memfn add-classpath set!`
Proxys:	`construct-proxy get-proxy-class proxy proxy-mappings proxy-super update-proxy`
Arrays:	`aclone alength aget aset aset-<type> amap <type>-array areduce make-array to-array into-array to-array-2d`
Primitives:	`int long float double char num boolean short byte bigdec bigint`
Exceptions:	`catch finally throw throw-if try`

Other

Boolean and comparison:	`= == identical? not= not true? false? nil?`
Creating fns:	`fn #() partial comp complement constantly`
Regex:	`#"pattern" re-matcher re-find re-matches re-groups re-seq`
XML:	`parse xml-seq`
Inspector:	`inspect inspect-table inspect-tree`
Misc.:	`identity assert with-open eval compile force hash name`

Zippers

Create:	`zipper`
Get zipper:	`seq-zip vector-zip xml-zip`
Get location:	`up down left right leftmost rightmost`
Get seq:	`lefts rights path children`
'Change':	`make-node replace edit insert-child insert-left insert-right append-child remove`
Move:	`next, prev`
Misc:	`root node branch? end?`

Parallel

Aggregate:	`pany pmax pmin psummary preduce`
Get collection:	`psort pvec pdistinct pfilter-dupes pfilter-nils`
Array	`par`

$Revision: 1.02, $Date: July 10, 2009
Steve Tayon (steve.tayon googlemail com)