U1YTUBH53 : any spectacular clojure alternative to <https://github.com/Raynes/conch>?
U1YTUBH53 : clojure's built-in `sh` is limited (no support for processing stdout/stderr as streams)
U1C72J3J4 : Hi clojurians! I'm trying to pattern-match a hashmap, but this is what I get:```(m/match [{:arst false}]
  [{:qwfp #"stuff.*"}] 1
  [{:arst false}] 2)
ClassCastException clojure.lang.Keyword cannot be cast to java.lang.CharSequence clojure.core/re-matcher
(core.clj:4775)```
What's going on here?

U07S8JGF7 : <@U1C72J3J4> I pasted exactly what you have into the repl and got `2`.
U1CD720KB : <@U1C72J3J4> Looks like that error message indicates a problem with the regex
U07S8JGF7 : Just to double check, what version of core.match are you using?
U1C72J3J4 : <@U07S8JGF7> `[org.clojure/core.match "0.3.0-alpha4"]`, with `(:require [clojure.core.match :as
m][clojure.core.match.regex])` in my ns
U1C72J3J4 : <@U1CD720KB> `(re-matches #"stuff.*" "stuff123")` works, so no regex issues as well
U1C72J3J4 : clojure version: `org.clojure/clojure "1.9.0-alpha15`
U07S8JGF7 : I'm not sure what the design of this is, but it looks like it might be a boog.
U07S8JGF7 : macroexpanding the above yields:```
```
(try
 (clojure.core/cond
  (clojure.core/instance? clojure.lang.ILookup x)
  (try
   (clojure.core/let
    [x_qwfp__6982
     (if
      (clojure.core/instance? clojure.lang.ILookup x)
      (clojure.core/get x :qwfp :clojure.core.match/not-found)
      (clojure.core.match/val-at* x :qwfp :clojure.core.match/not-found))]
    (clojure.core/cond
     (clojure.core/re-matches #"stuff.*" x_qwfp__6982)
     1
     :else
     (throw clojure.core.match/backtrack)))
   (catch
    Exception
    e__6002__auto__
    (if
     (clojure.core/identical? e__6002__auto__ clojure.core.match/backtrack)
     (do
      (try
       (clojure.core/let
        [x_arst__6983
         (if
          (clojure.core/instance? clojure.lang.ILookup x)
          (clojure.core/get x :arst :clojure.core.match/not-found)
          (clojure.core.match/val-at* x :arst :clojure.core.match/not-found))]
        (clojure.core/cond
         (clojure.core/= x_arst__6983 false)
         2
         :else
         (throw clojure.core.match/backtrack)))
       (catch
        Exception
        e__6002__auto__
        (if
         (clojure.core/identical? e__6002__auto__ clojure.core.match/backtrack)
         (do (throw clojure.core.match/backtrack))
         (throw e__6002__auto__)))))
     (throw e__6002__auto__))))
  :else
```

```
 (throw clojure.core.match/backtrack))
(catch
 Exception
 e__6002__auto__
 (if
  (clojure.core/identical? e__6002__auto__ clojure.core.match/backtrack)
  (do
   (throw
    (java.lang.IllegalArgumentException.
     (clojure.core/str "No matching clause: " x))))
  (throw e__6002__auto__))))
```

U07S8JGF7 : woof that was a lot, sorry ya'll
U07S8JGF7 : Anyways it looks like it's probably calling `re-matches` with `:clojure.core.match/not-found`.
U07S8JGF7 : There should probably be a guard there in `to-source ::m/regex`.
U07S8JGF7 : <https://dev.clojure.org/jira/browse/MATCH-123>
U1C72J3J4 : <@U07S8JGF7> thank you, following the report...
U07S8JGF7 : <@U1C72J3J4> Be sure to upvote it if you want attention paid.
U297WCSHK : is there a formatter for clojure code that I can adapt progressively without converting the whole codebase at one? I tried parinfer for Atom, but it is too intrusive for me
U5XMV6DQT : emacs
U051SS2EU : `cljfmt` is a good plugin and unlike editor based options every collaborator can easily have the same rules
U051SS2EU : <https://github.com/weavejester/cljfmt>
U0B4ZBBKM : Is there an obvious way to write the following without repeating `x`? `(if (some-pred? x) y x)`Meaning if `x` is ok, just use it, otherwise use the alternative `y`

U5XMV6DQT : it could be written like ```(or (somefn x) y)```, but inside somefn there definitely would be repetitions of x or % argument
U1CTH1TUY : you have three values you care about, `x,` `y`, and `(some-pred? x)`, so you will need those three elements. if x is truthy, you could write the pred to return x or false and use `(or (some-pred? x) y)`, though that means custom predicates.
U1CTH1TUY : Doesn't work if you need pred to succeed on x being false / nil though
U5XMV6DQT : ```complement```
U11SJ6Q0K : I think `(if (foo? x) y x)` is readable and repeating one name, once as a parameter to the predicate, isn't bad
U056QFNM5 : <@U0B4ZBBKM> You can do that with a macro if you so desire. Something like this:
```
(defmacro pass-or-alternative [pred-expr value-expr alternative-expr]
  `(let [value# ~value-expr]
     (if (~pred-expr value#)
       value#
       ~alternative-expr)))
```

U1CTH1TUY : you don't need a macro, just `(fn [pred x y] (if (pred x) x y)`, then you'd just use it like `(my-fn pred x y)`
U0DJC1V3R : you do need a macro, you don't always want to evaluate `y`
U051SS2EU : if y has side effects the macro version is better
U08TWB99B : although the original question did not mention any lazy evaluation requirements