Rash - *adj* - Hurrying into action or assertion without due caution and regardless of prudence, hasty, reckless, precipitate.
I ♥ Unix Shells
Many issues

variable=foo
# silently reference undefined!
echo $variable
if [ $foo -eq $bar ]; then
cowsay "$foo equals $bar"
fi
empty string!

if [ $foo -eq $bar ]; then
cowsay "$foo equals $bar"
fi
if [ $foo -eq $bar ]; then
cowsay "$foo equals $bar"
fi

Error!
ls | grep rkt | xargs wc | cowsay
Racket and Shell

so

synergy

such together
ls /dev | grep tty | wc -l
Overview

**shell/pipeline**
- Byte streams

**shell/mixed-pipeline**
- Objects (and byte streams mixed)

**shell/pipeline-macro**
- Macro DSL wrapper for pipelines

**rash line syntax**
- Reader and macro support for line-based syntax
scsh comparison

shell/pipeline
• Byte streams

shell/mixed-pipeline
• Objects (and byte streams mixed)

shell/pipeline-macro
• Macro DSL wrapper for pipelines

rash line syntax
• Reader and macro support for line-based syntax
scsh comparison

shell/pipeline
• Byte streams

shell/mixed-pipeline
• Objects (and byte streams mixed)

shell/pipeline-macro
• Macro DSL wrapper for pipelines

rash line syntax
• Reader and macro support for line-based syntax
**scsh comparison**

**shell/pipeline**
- Byte streams

**shell/mixed-pipeline**
- Objects (and byte streams mixed)

**shell/pipeline-macro**
- Macro DSL wrapper for pipelines

**rash line syntax**
- Reader and macro support for line-based syntax
Overview

**shell/pipeline**
- Byte streams

**shell/mixed-pipeline**
- Objects (and byte streams mixed)

**shell/pipeline-macro**
- Macro DSL wrapper for pipelines

**rash line syntax**
- Reader and macro support for line-based syntax
(require shell/pipeline)

(run-pipeline ' (ls /dev)
              ' (grep tty)
              ' (wc -l))
(require shell/pipeline)

(run-pipeline 'ls /dev)
  '(grep tty)
  '(wc -l)
#:out "myfile.txt")
(require shell/pipeline)

(define (my-grep regex) ...

(run-pipeline '(ls /dev)
  `'((my-grep "tty")
    '(wc -l)
    #:out "myfile.txt")
Overview

shell/pipeline
• Byte streams

shell/mixed-pipeline
• Objects (and byte streams mixed)

shell/pipeline-macro
• Macro DSL wrapper for pipelines

rash line syntax
• Reader and macro support for line-based syntax
Overview

shell/pipeline
- Byte streams

shell/mixed-pipeline
- Objects (and byte streams mixed)

shell/pipeline-macro
- Macro DSL wrapper for pipelines

rash line syntax
- Reader and macro support for line-based syntax
(require shell/pipeline-macro)

(run-pipeline
 =unix-pipe= ls /dev
 =object-pipe= string-split _ "\n"
 =filter= regexp-match "tty"
 =object-pipe= length)
(require shell/pipeline-macro)

;; The | symbol is special to the default reader, so let's use % instead for now.

(run-pipeline
  % ls /dev
  %> string-split _ "\n"
  =filter= regexp-match "tty"
  %> length)
(require shell/pipeline-macro)

;; The | symbol is special to the default reader, so let's use % instead for now.

(run-pipeline
  ls /dev
  %> string-split _ "\n"
  =filter= regexp-match "tty"
  %> length)
;; I want this:
(run-pipeline
 =non-auto-globbing-unix-pipe= rm (glob "*.pdf"))

;; You maybe want this:
(run-pipeline
 =auto-globbing-unix-pipe= rm *.pdf)
=filter=
=monad-bind=
=for/stream=
=place=
=object-or-unix-by-binding=
=xargs=
=object-send=
=infix-math=

... use your imagination for more!
Overview

shell/pipeline
• Byte streams

shell/mixed-pipeline
• Objects (and byte streams mixed)

shell/pipeline-macro
• Macro DSL wrapper for pipelines

rash line syntax
• Reader and macro support for line-based syntax
#lang rash/demo/rc17

app + 1 2 3

;; returns 6
#lang rash/demo/rc17

`;; The current default line-macro is inserted.
+ 1 2 3
`;; returns 6
#lang rash/demo/rc17

```
cd ../projects/racket-pkgs
```
#lang rash/demo/rc17

run-pipeline ls | grep foobar
#lang rash/demo/rc17

;; pipeline is a good default line-macro
ls | grep foobar
#lang rash/demo/rc17

ls (list
    widget-dir old-widget-dir etc) | grep widget
#lang rash/demo/rc17

(require racket/string)
(define main-dir "~/home/userguy/project")

ls (string-append
    main-dir "/widget-dir") | grep widget
(define a (list 1 2 3))
(define b (map square a))

;; Look at the value of b
b
#lang rash/demo/rc17

(define a (list 1 2 3))
(define b (map square a))

;; Error!! This will now have a line-macro inserted!
b
#lang rash/demo/rc17

(define a (list 1 2 3))
(define b (map square a))

;; Wrap it in parens instead
(values b)
(define a (list 1 2 3))
(define b (map square a))

;;; or use an identity line macro
id b
Control Flow
(for ([str cow-phrases])
  (rash «|> display str | cowsay»))
(for ([str cow-phrases])
  @rash|{|> display str | cowsay|})
;; ▼foo bar▼ reads as (##upper-triangles ▼foo bar▼)
;; ##upper-triangles can be bound to the rash macro

(for ([script (list "delete-temp-files.sh"
               "clean-up.zsh"
               "rmstuff.rkt")])
  »grep "rm -rf" »which (id script)▼▼)
;;; {foo bar} reads as (%braces «foo bar»)
;;; %braces can be bound to the rash macro

(for ([script (list "delete-temp-files.sh"
                  "clean-up.zsh"
                  "rmstuff.rkt")])
  (grep "rm -rf" (which (id script))))
Disclaimer:

The library is not stable yet.
Interactive REPL demo
Conclusion

Shell
+
Racket
+
Easy mixing of both
=
Awesome

Question?

william@hatch.uno