damien@c196044: ~/Documents/Coral

```
Documents/Coral ./slides.coral
```

./slides.coral

http://rmod.lille.inria.fr/coral/

integrating Pharo smalltalk
with the shell

Contributors:

Jean-Baptiste Arnaud Olivier Auverlot Adrien Barreau
Camillo Bruni Marcus Denker Stéphane Ducasse Igor Stasenko
Damien Pollet

./slides.coral

image = prison

...particularly true in Squeak / Pharo

No easy way out:
* how do you chmod +x a file?
* ...run external commands?
* ...read environment variables
* ...access username, privileges?

No easy way in either:
* fileIn syntax impractical! !magic comments!.
* if an exception is raised in a headless image, does it really make a sound?
* there's RFB, sure, but how to evaluate code in an already running image?

./slides.coral

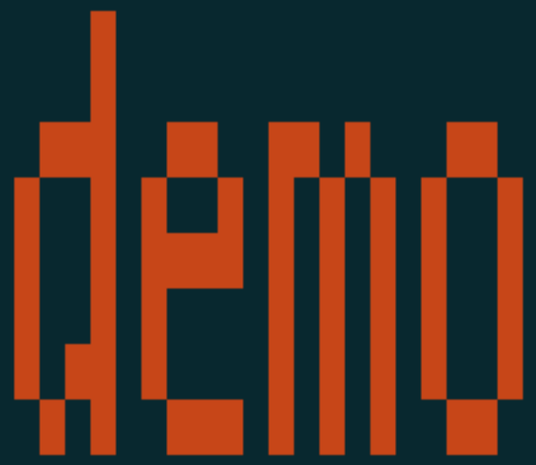
what's there

1. wrapper shell script
    * knows VM + image
    * options to run headless or not
    * passes arguments + misc. info to the image

2. syntax extension for writing code in a file
    * Smalltalk 80 has no syntax in between methods
    * script = shebang + evaluation chunks + method chunks
    * flat structure, no nesting

3. prepared image
    * Coral machinery at image startup
    * utility packages (OSProcess, easy stdin/out/err, Ansi escapes...)

./slides.coral



REPL slides.coral growing.coral

coral

```
Documents/Coral coral
                                                \   /
----------------------------------------          *
Pharo                                            /|=|\
open-source Smalltalk-inspired environment        |=|
----------------------------------------          |===|
> 42 + 51
93
> 'hello, Coral!'
'hello, Coral!'
>
```

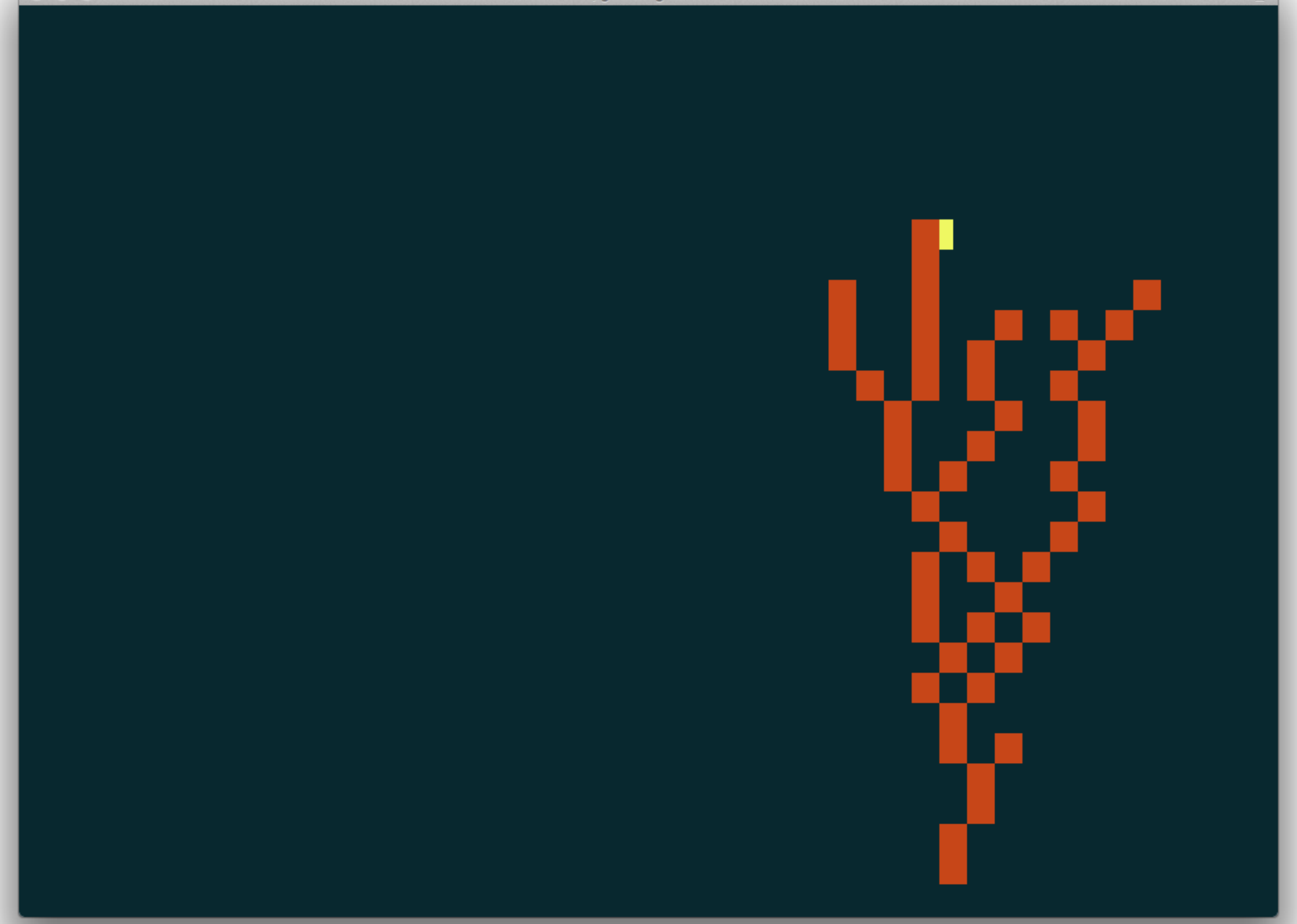
vim ./slides.coral

```
#! /usr/bin/env coral -i

[ | esug2011 |
    esug2011 := SlideDeck named: 'Coral @ esug 2011'.

    esug2011
        blankSlide: 'Coral' with: [ :s | s logo: 'Coral' ];
        previousWith: [ :s |
            s blue: [ s centered: 'http://rmod.lille.inria.fr/coral/'; margin: 1 ].
            s yellow: [ s centered: 'integrating Pharo smalltalk
with the shell' ] ];
        previousWith: [ :s |
            s margin: 2.
            s centered: 'Contributors:'.
            s margin: 1.
            s white: [
                #(('Jean-Baptiste Arnaud' 'Olivier Auverlot' 'Adrien Barreau')
                  ('Camillo Bruni' 'Marcus Denker' 'Stéphane Ducasse' 'Igor Stasenko'))
                        do: [:each | s centered: ('    ' join: each)] ].
            s yellow: [ s centered: 'Damien Pollet' ] ].

    esug2011
        slide: 'image  =  prison';
        previousWith: [ :s |
            s subtitle: '...particularly true in Squeak / Pharo' ];
        previousWith: [ :s |
            s margin: 2; text: 'No easy way out:';
                item: 'how do you chmod +x a file?';
 ./slides.coral [st]                                                L1 C1
"./slides.coral" 136L, 6333C
```

vim growing.coral

```
#! /usr/bin/env coral

CoralGrowing class >> width: width height: height
[
    ^ self basicNew initializeWidth: width height: height
]

CoralGrowing class >> new
[
    ^ self
        width: (OSProcess thisOSProcess environment at: #COLUMNS ifAbsent: [0]) asNumber
        height: (OSProcess thisOSProcess environment at: #LINES ifAbsent: [0]) asNumber
]

CoralGrowing >> initializeWidth: w height: h
[
    width := w // 2.
    height := h.
    heights := Array new: width withAll: height.
    finished := false
]

CoralGrowing >> frameDelay: aDelay
[
    delay := aDelay
]

CoralGrowing >> animate
 growing.coral [st]                                                    L1 C1
"growing.coral" 76L, 1703C
```

./growing.coral



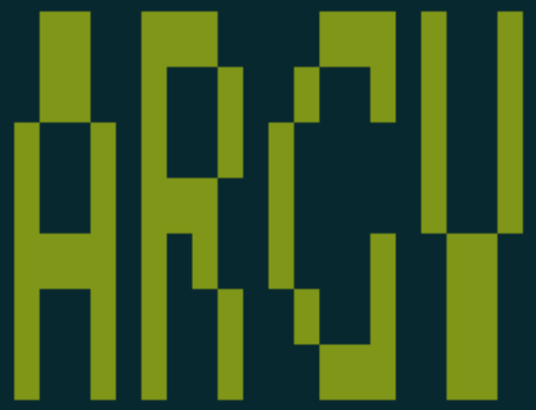
./slides.coral

NOT: coding in a bunch of text files
* I want my debugger, refactoring browser, etc
* version control? monticello + git = recipe for insanity

RATHER: easy management & invocation of images
* think RVM or cloud tools
* most scripts will only contain invocation-specific code
* all complex code written as usual in the browser

Coral syntax as a replacement for the chunk format
* no nesting, clear self contained chunks
* more kinds of chunks (packages, categories, class comments...)

./slides.coral



It's always a chore to parse command line arguments...

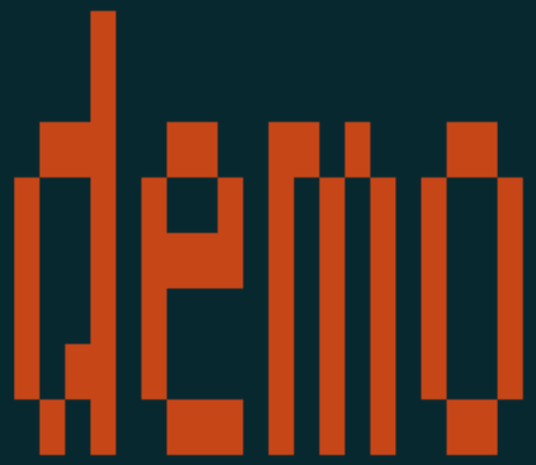
DSL to specify:
* commands, with options, positional parameters, and subcommands
* options (-z/--zorglub) with parameters
* positional parameters with a datatype parser (URLs, numbers...)
* description & documentation for all of those

Parses ARGV with PetitParser

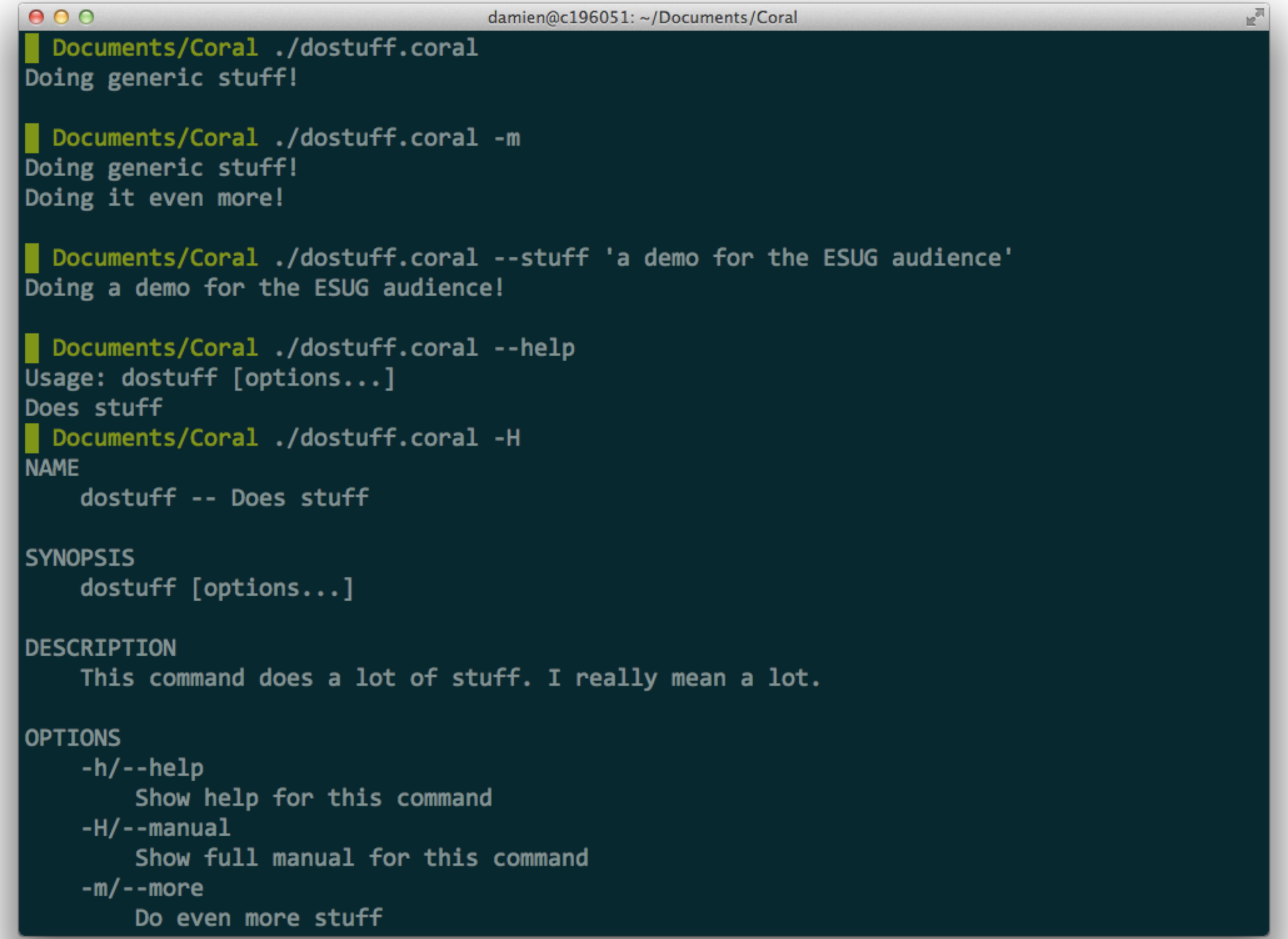
Generates help, man page, shell completion...

Inspired by Denis Defreyne's Cri (yup, Smalltalk has lots to learn from Ruby ;)

./slides.coral




```
dostuff.coral
```


damien@c196051: ~/Documents/Coral
Documents/Coral ./dostuff.coral
Doing generic stuff!
Documents/Coral ./dostuff.coral -m
Doing generic stuff!
Doing it even more!
Documents/Coral ./dostuff.coral --stuff 'a demo for the ESUG audience'
Doing a demo for the ESUG audience!
Documents/Coral ./dostuff.coral --help
Usage: dostuff [options...]
Does stuff
Documents/Coral ./dostuff.coral -H
NAME
dostuff -- Does stuff
SYNOPSIS
dostuff [options...]
DESCRIPTION
This command does a lot of stuff. I really mean a lot.
OPTIONS
-h/--help
Show help for this command
-H/--manual
Show full manual for this command
-m/--more
Do even more stuff

vim dostuff.coral

```
#! /usr/bin/env coral
[   "Let's define a command"
    | command |

    command := (CLICommandParser named: 'dostuff')
        aliases: #('dostuff.coral');
        summary: 'Does stuff';
        description: 'This command does a lot of stuff. I really mean a lot.'.

    "Declare each of its options"
    command option -'h' --'help';
        description: 'Show help for this command';
        do: [ :cmd |
            cmd err << cmd help withUnixLineEndings.
            cmd exitSuccess ].
    command option -'H' --'manual';
        description: 'Show full manual for this command';
        do: [ :cmd |
            cmd err << cmd manual withUnixLineEndings.
            cmd exitSuccess ].

    command option -'m' --'more';
        description: 'Do even more stuff'.

    command option -'s' --'stuff'; any;
         description: 'Specify stuff to do'.

    "Now give it some behavior"
 dostuff.coral [st]                                                L1 C1
"dostuff.coral" 41L, 1233C
```

vim dostuff.coral

```

    "Now give it some behavior"
    command do: [ :cmd |
        |stuff|
        stuff := cmd atOptionNamed: #stuff ifNone: [ 'generic stuff' ].

        cmd << 'Doing ' << stuff << '!'; newline.
        (cmd hasOptionNamed: #more)
            ifTrue: [ cmd << 'Doing it even more!'; newline ]
    ].

    "Now we can invoke our command"
    command run
]
" vim: set ft=st :"
~
~
~
~
~
~
~
~
~
~
~
~
~
dostuff.coral [st]                                                        L27 C1
```

./slides.coral

status

Read-Eval-Print Loop
	* really simplistic, but works

Syntax
	* based on PetitParser by Lukas Renggli
	* not GNU Smalltalk compatible (could be done, but not a goal)

Terminal interaction
	* basic ANSI escape sequences
	* arguments parsing

Host system interaction
	* OSProcess
	* FileSystem

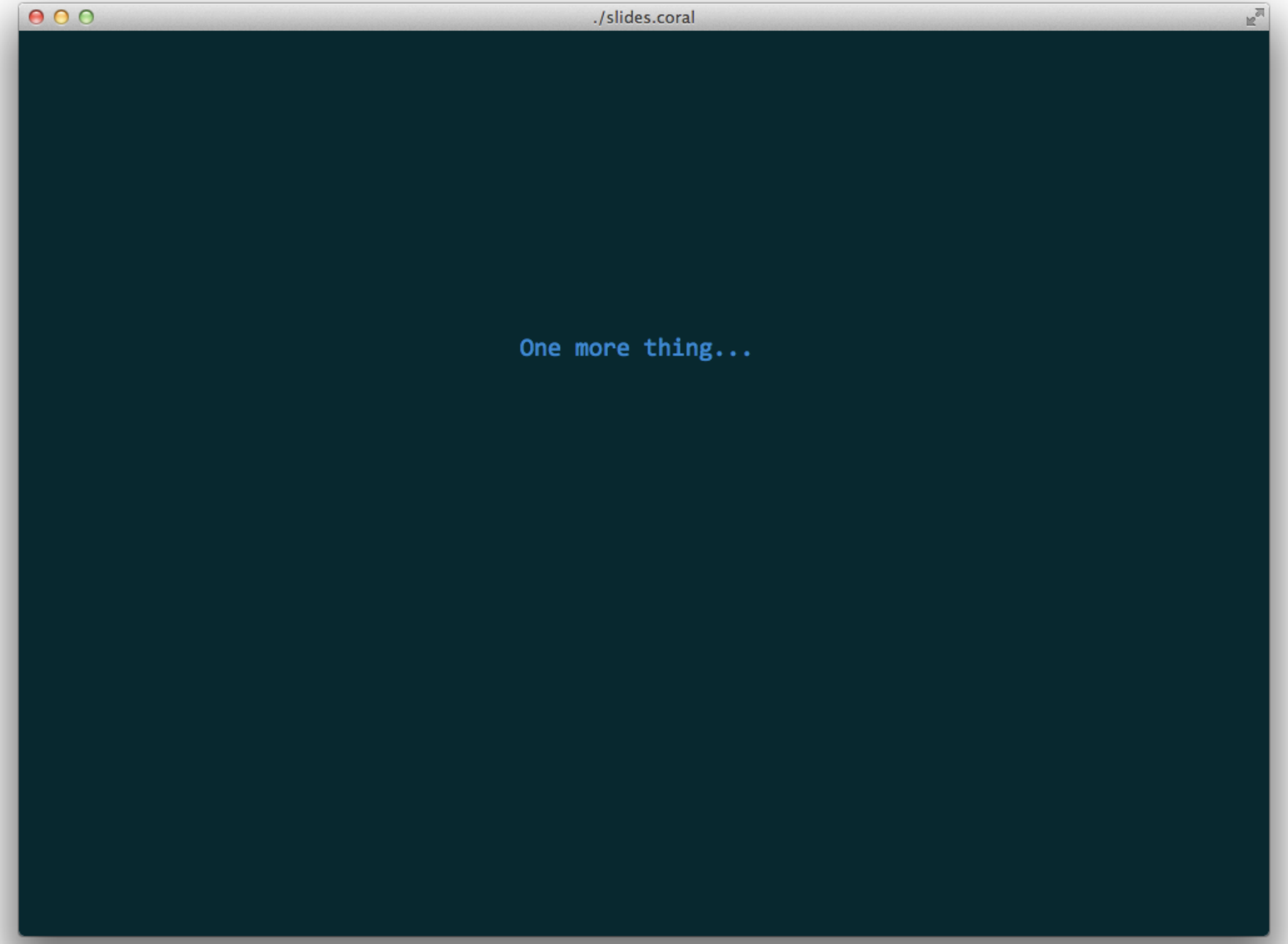
./slides.coral

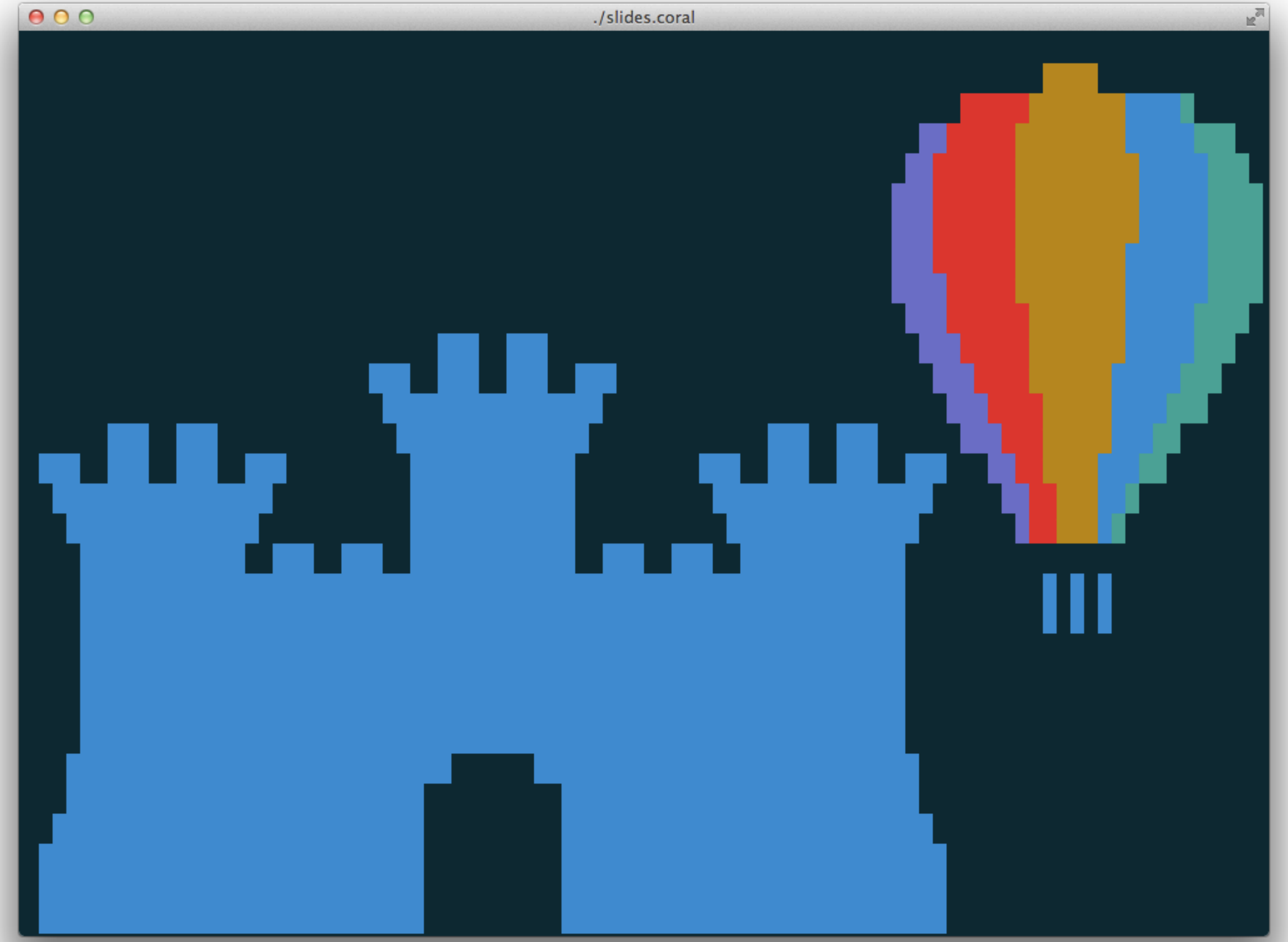
future

short term
* handle command-line subcommands
* prepare a proper beta release

medium term
* proper error reporting
* improved REPL, history, multi-line editor (Camillo Bruni)
* lots of fixes to streams, OSProcess, VMs...
* minimized image

sci-fi...
* full text-mode inspector/debugger like https://github.com/pry/pry


./slides.coral
One more thing...


./slides.coral

./slides.coral

integrating Pharo smalltalk
with the shell

http://rmod.lille.inria.fr/coral/

Questions?