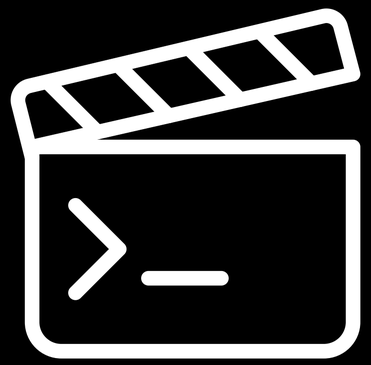


CLAP

Pharo

```
exampleWithNumber: x

<syntaxOn: #postcard>
"A ""complete"" Pharo syntax"
| y |
true & false not & (nil isNil)
 ifFalse: [ self perform: #add: with: x ].
y := thisContext stack size + super size.
byteArray := #[2 2r100 8r20 16rFF].
{ -42 . #($a #a #'I''m' 'a' 1.0 1.23e2 3.14s2 1) }
 do: [ :each |
    | var |
    var := Transcript
     show: each class name;
     show: each printString ].

^ x < y
```

method name · parameter · pragma · comment · local variable · binary message · unary message · boolean literals · nil literal · block · keyword message · pseudo variables · assignment · instance variable · integer literals · byte array · array generated at runtime · literal array · symbols · character · floating point · string · scaled decimal · local block variable · block parameter · global variable · cascade · keyword message · return instruction

other method definition examples:
unary
+ binaryMessageArgument
keyword: arg
keyword: arg1 withTwo: arg2

https://www.pharo.org

.....

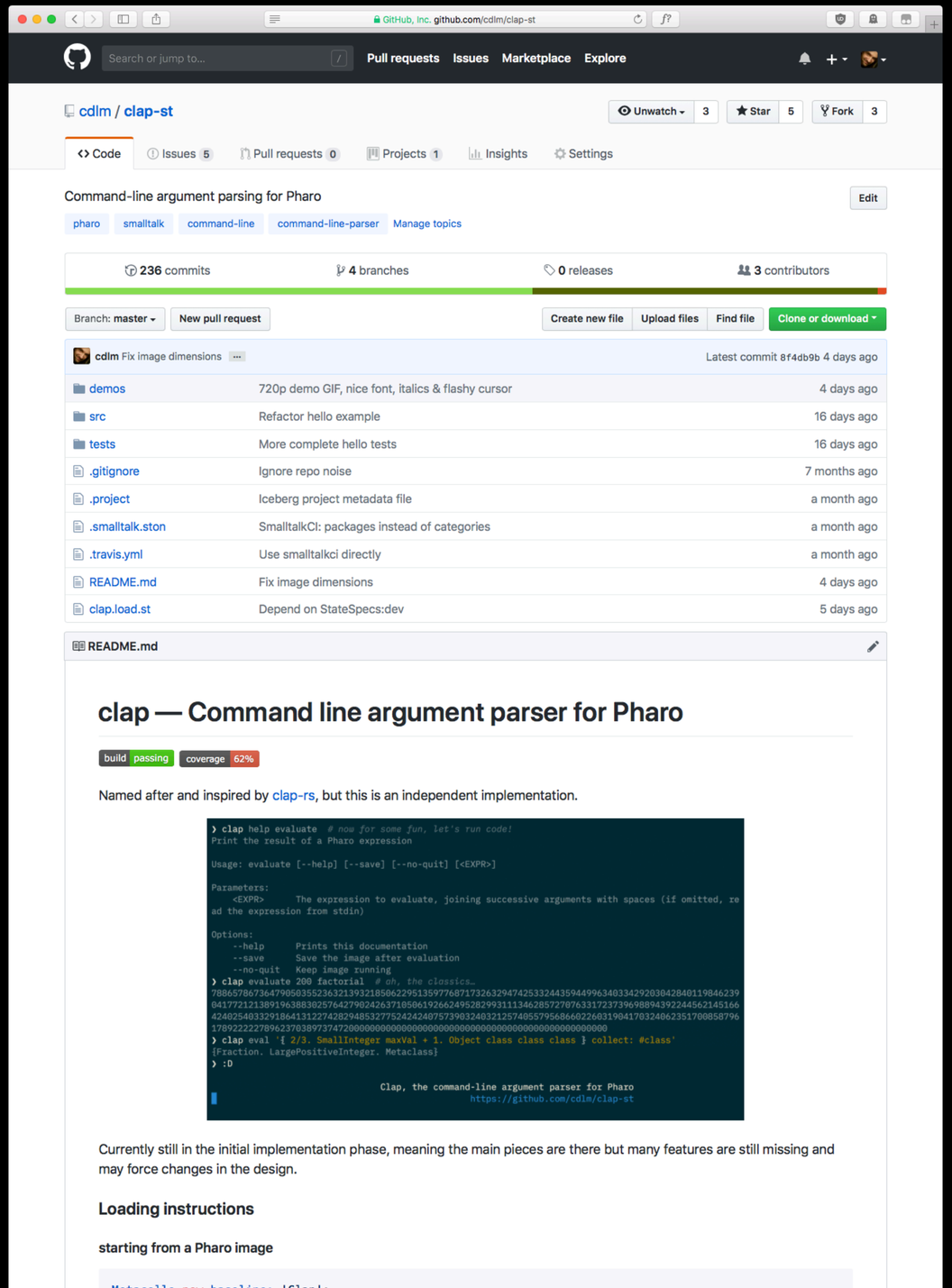
.....

.....

.....

Command Line Argument Parser

demo



github.com/cdlm/clap-st

hello! I'm Damien Pollet

RMOD team @ Inria Lille, France

besides Clap:

- Coral (a looong time ago)
- LaTeX typography for the Pharo books
- fari.sh (image builder/runner)

…and I teach Java… 😱

Use the command line?
please talk to me!

what

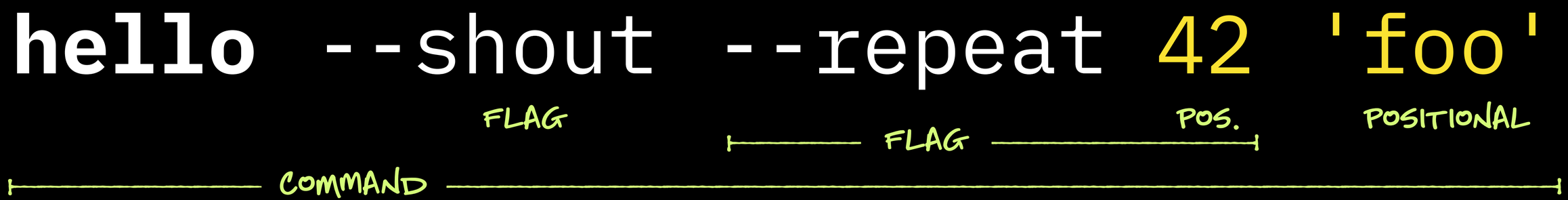
hello

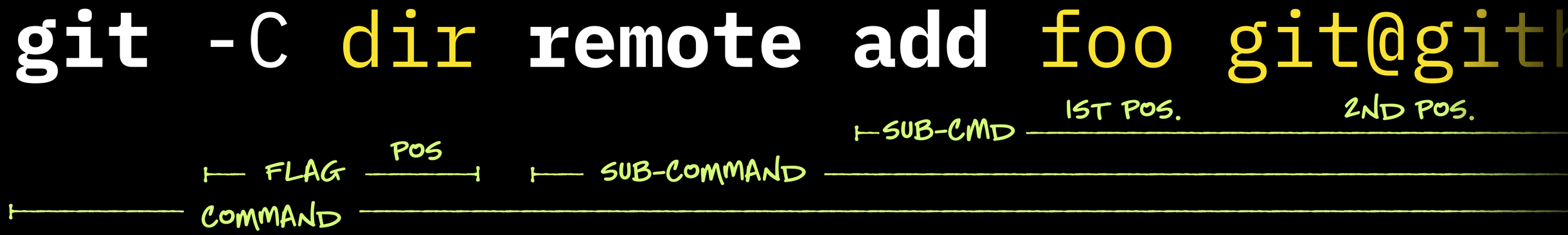
COMMAND

```
hello 'foo'
```

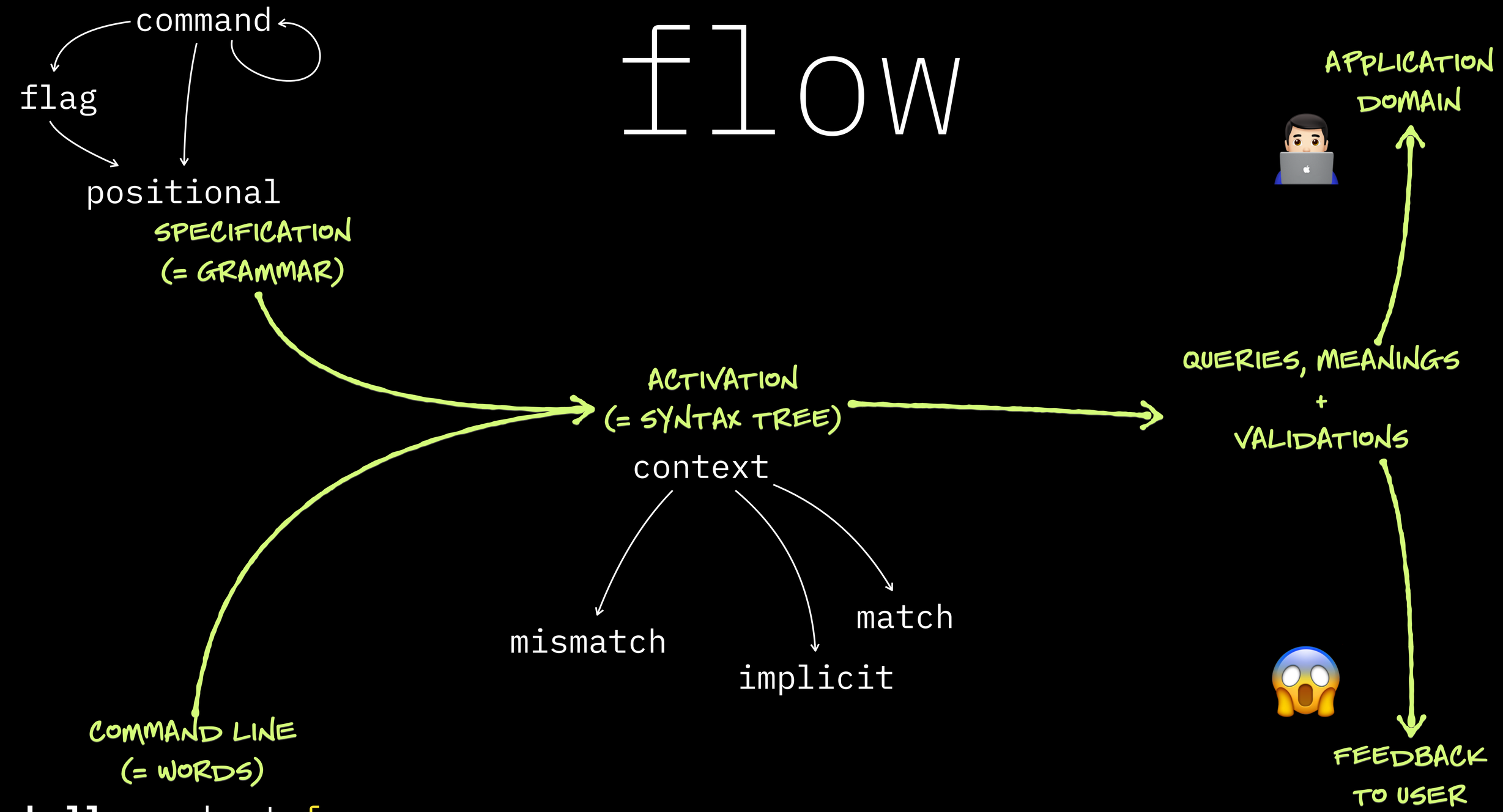
POSITIONAL


hello --shout 'foo'
COMMAND
FLAG
POSITIONAL


hello --shout --repeat 42 'foo'
FLAG
FLAG
POS.
POSITIONAL
COMMAND


git -C dir remote add foo git@gith
1ST POS.
2ND POS.
SUB-CMD
POS
FLAG
SUB-COMMAND
COMMAND

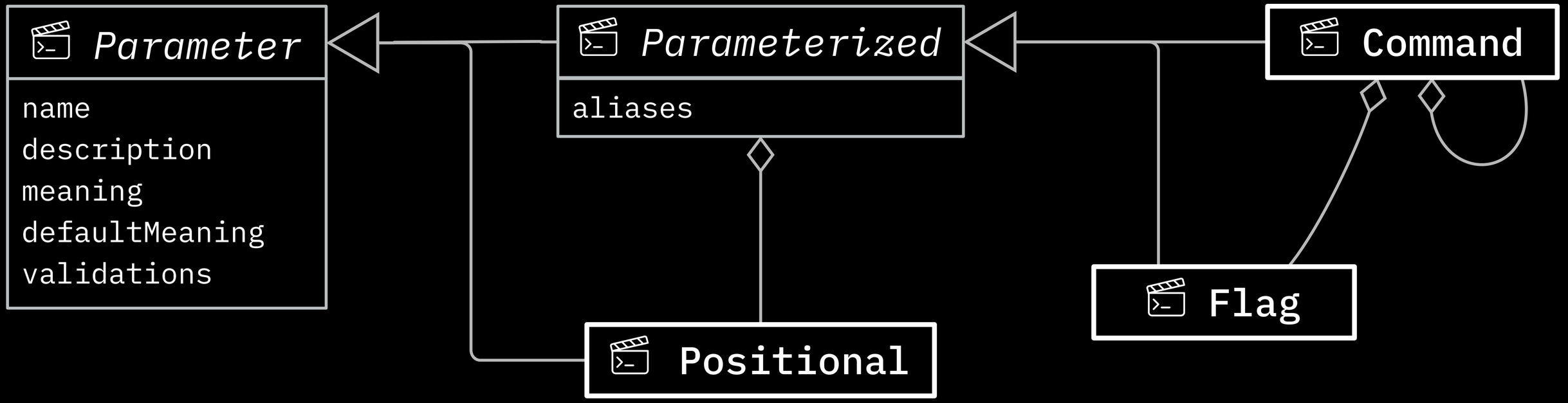
flow



hello --shout foo

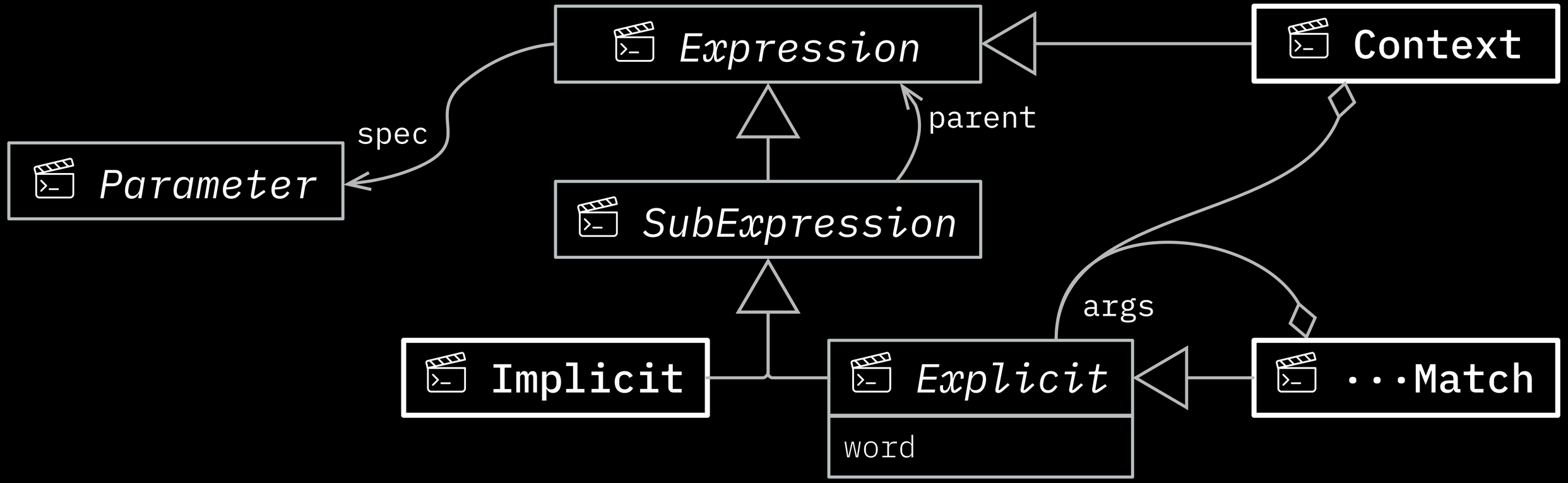
how

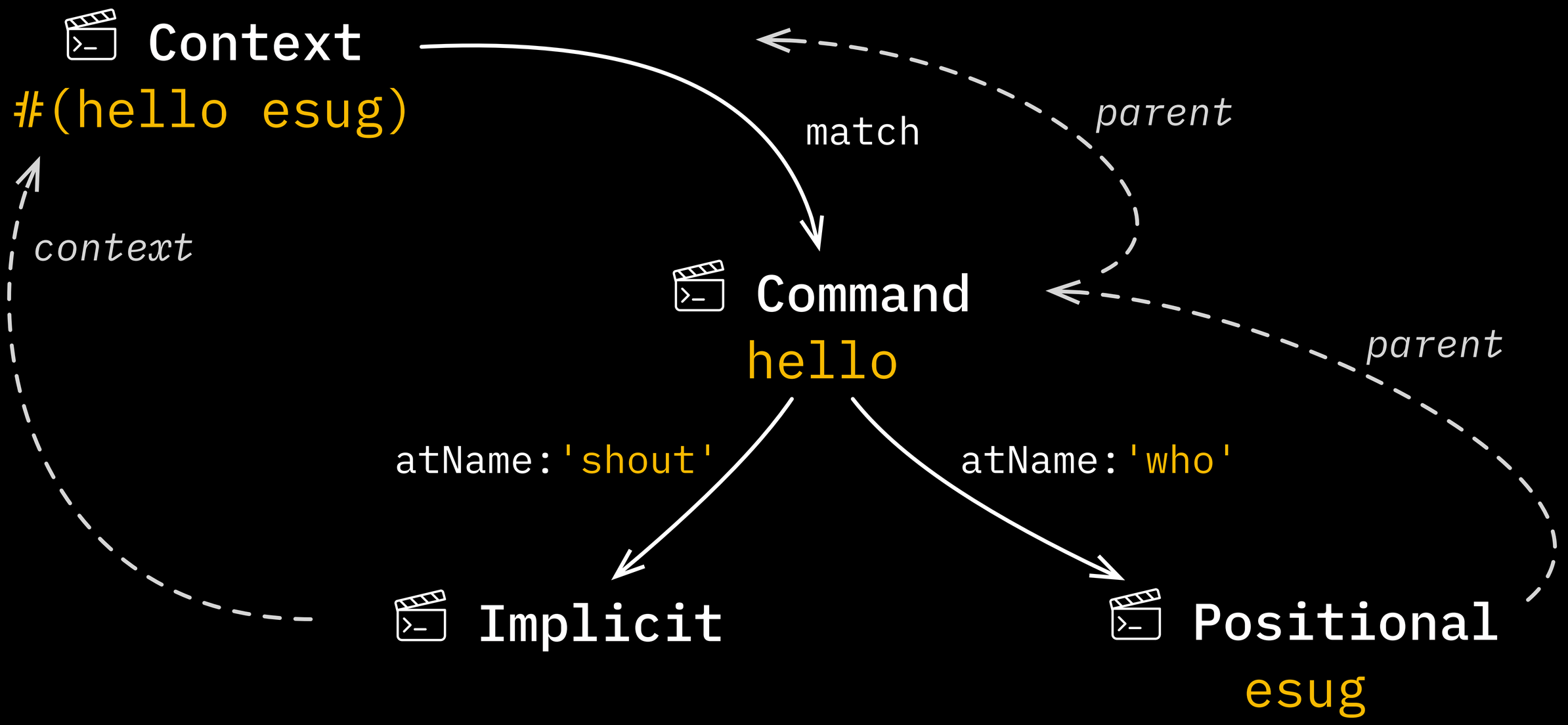
specify


Parameter
name
description
meaning
defaultMeaning
validations
Parameterized
aliases
Command
Flag
Positional

```
param := (ClapPositional withName: 'who')
    description: 'Recipient of greeting';
    defaultMeaning: [ 'world' ].

cmd := (ClapCommand withName: 'hello')
    description: 'Says hello';

    add: param;
    add: (ClapFlag withName: 'shout');

    meaning: [ :args |  👨‍💻💭  ]
```

activate


Expression
Context
spec
parent
Parameter
SubExpression
args
Implicit
Explicit
···Match
word


Context
#(hello esug)
match
parent
context
Command
hello
parent
atName:'shout'
atName:'who'
Implicit
Positional
esug

```
context := (ClapContext specification: cmd)
    arguments: #('hello' 'esug').


"same + immediate evaluation:"
cmd activateWith: #('hello' 'esug').
```

```
context match.      "a ClapCommandMatch(hello esug)"

context match at: param.
context match atName: 'who'.
                        "a ClapPositionalMatch(esug)"

context match atName: 'shout'.  "a ClapImplicit"
```

give meaning

```
cmd meaning: [ :args | 🤦‍♂️💭42 ].

context match value. "42"

(context match atName: 'who') value.   "esug"
(context match atName: 'shout') value. "false"
```

```
cmd meaning: [ :args |

    | recipient |
    recipient := args atName: 'who'.

    args context stdout
        << 'hello ' , recipient value;
        lf
]
```

SPEC

MATCH OF CMD

MATCH OF POSITIONAL

MEANING OF POSITIONAL (IMPLICIT MATCHES GIVE DEFAULT MEANING)

install

```
ClapCommandLineExamples class >> hello
    <commandline>
    ^ (ClapCommand withName: 'hello')
        description: 'Says hello';
        add: (ClapPositional withName: 'who')
            description: 'Recipient of greeting'
            defaultMeaning: [ ...
```

```
› ./pharo Pharo.image clap  hello esug
```

more

exit

```
context exitSuccess.
Exit success signal.
```

```
context exitFailure.
context exit: 1.
Exit failure signal.
```

```
"OK for interactive use (catches Exit)"
 cmd activateWith: #('hello' 'esug').
```

help

ClapCommand forHelp. **git help** commit

DOCUMENTS SPECIFIED SIBLING, OR PARENT
RUNS AUTOMATICALLY

ClapFlag forHelp. **git commit** --help

DOCUMENTS PARENT
REQUIRES EXPLICIT CHECK

```
meaning: [ :args |
    args atName: 'help' ifFound: [ :help |
        help value; exitSuccess ].
```

validate

git -C dir remote add foo git@gith

VALID REPO

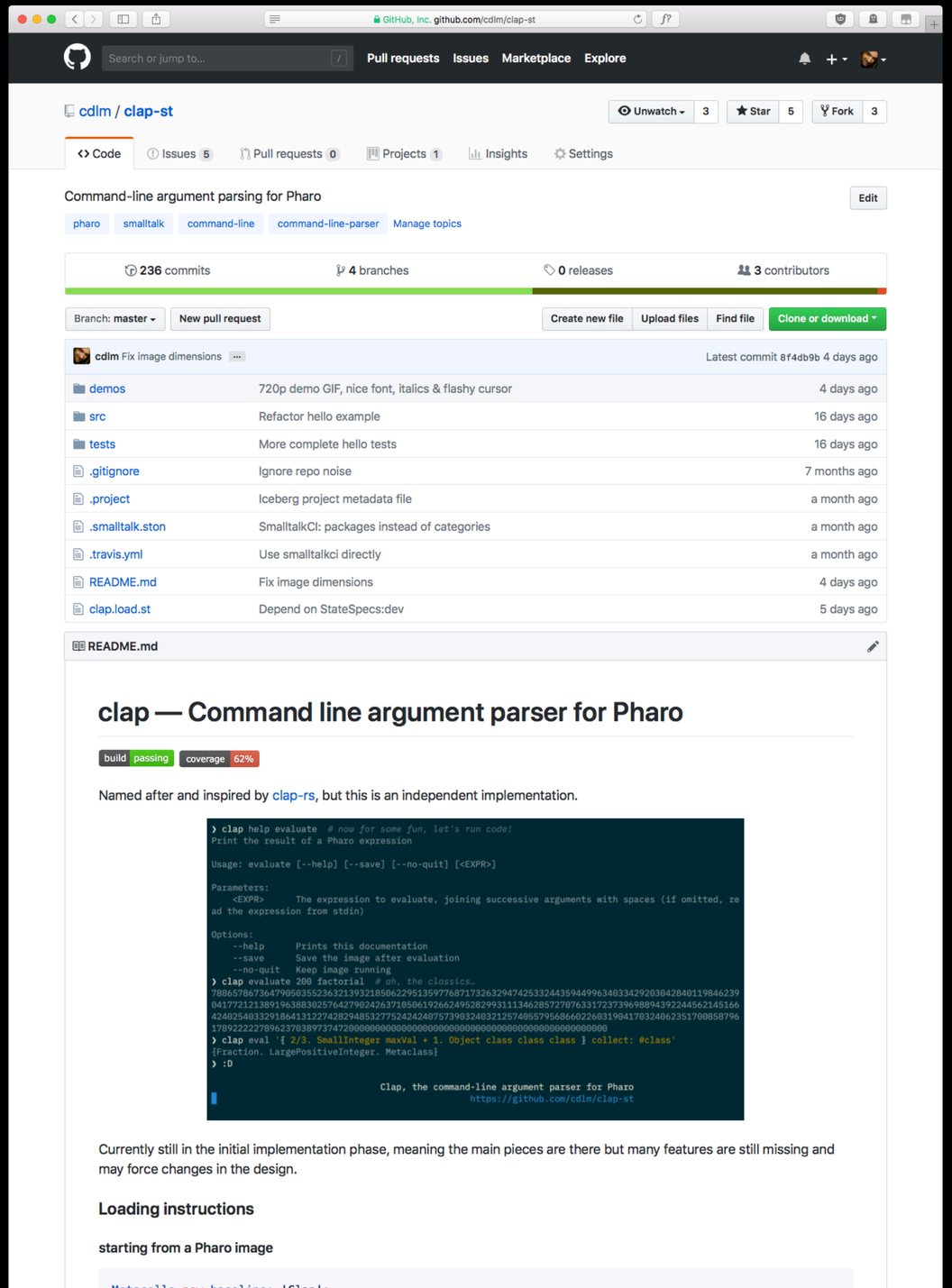
VALID NAME

VALID URL

⏱ work in progress…

```
error[E0382]: use of moved value: `s1`
 --> src/tmp.rs:4:20
  |
3 |     let s2 = s1;
  |         -- value moved here
4 |     println!("{}", s1);
  |                    ^^ value used here after move
  |
  = note: move occurs because `s1` has type `std::string::String`, which does no

error: aborting due to previous error
```

hackR

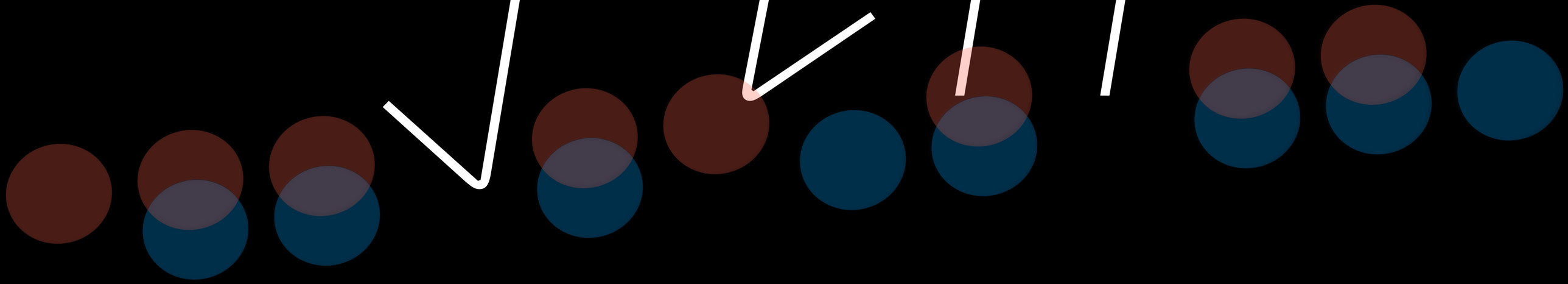


github.com/cdlm/clap-st

- ✓ *basic architecture*
- ✓ *workspace & commandline*
- ✓ *pharo commands*
  - *version & evaluate → more to port*
- ↺ *validations*
  - *error handling & feedback*

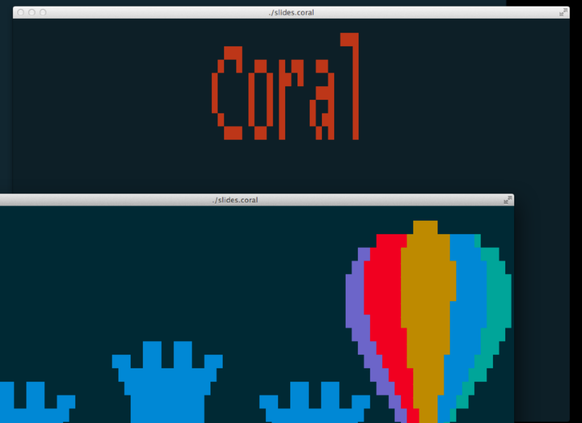
- → *common meanings*
- → *Coral-like DSL*
- → *port fari.sh*
- → *shell completions…*

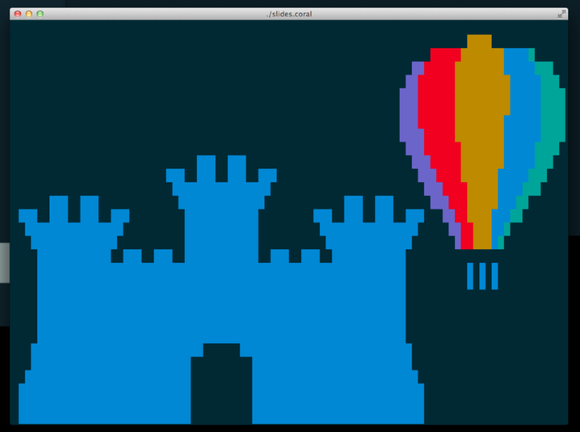
fin



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]
"dostuff.coral" 41L, 1233C
```





Clap demo (YouTube)

Coral slides

Pharo postcard

Clapping Music

IBM Plex fonts