Harnessing the Power of Open Data on the Web (& mocking/testing!!)

@sckottie

Scott Chamberlain
UC Berkeley / rOpenSci

scotttalks.info/jsm2018

THE LEONA M. AND HARRY B.
HELMSLEY
CHARITABLE TRUST

ROpenSci

ropensci.org

rOpenSci Does

Community **Software** **Peer Review**

ROpenSci

ORIGINS

$$\begin{tikzcd}
\textbf{Data acquisition} \arrow[d] \\
\textbf{Data manipulation/analysis/viz.} \arrow[d] \\
\textbf{writing (manuscript/report/etc.)} \arrow[d] \\
\textbf{publish}
\end{tikzcd}$$

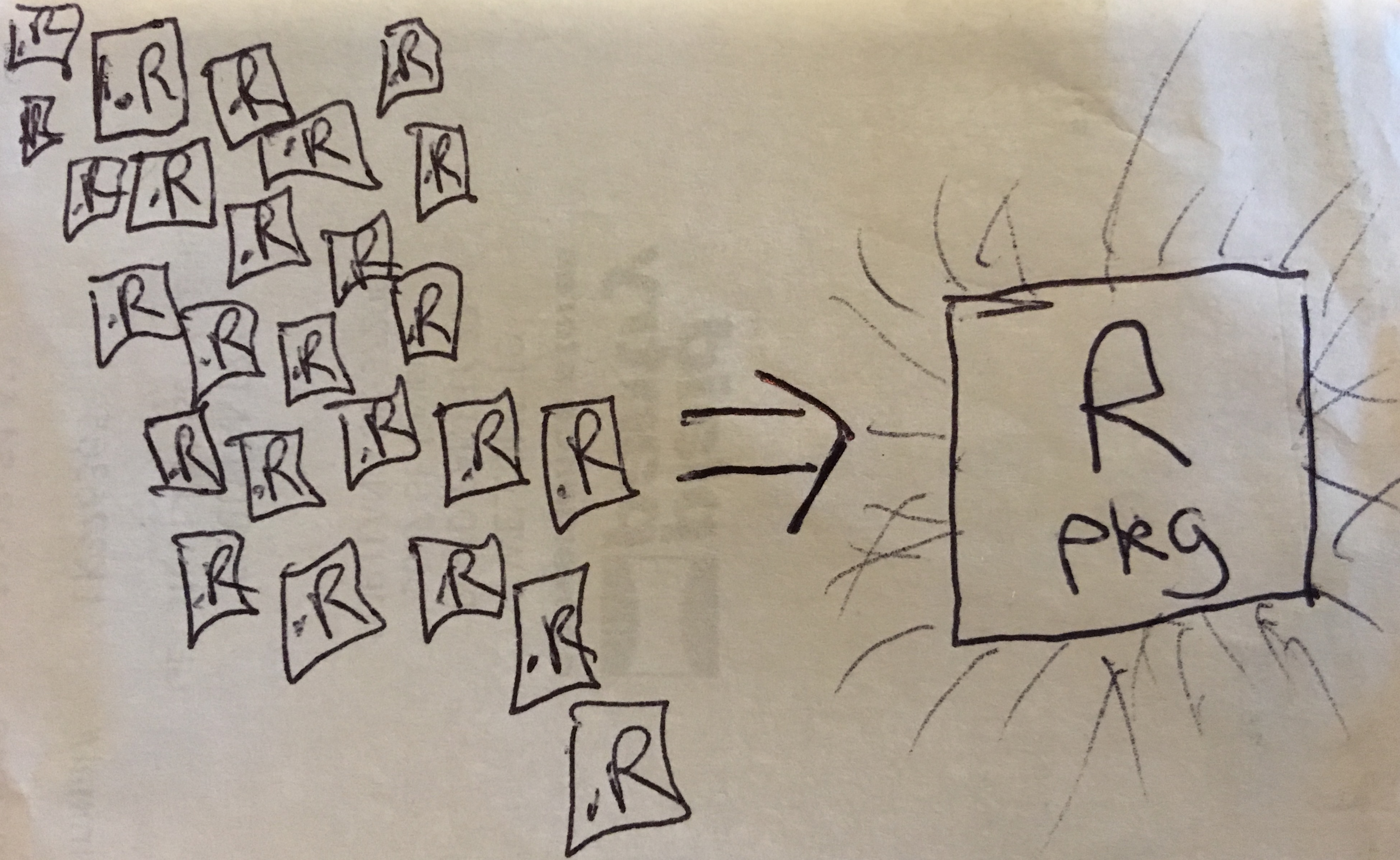
an unsolved problem in 2011:

Getting data from the web

make R pkgs for specific data sources

At first embracing a niche seems like smaller impact

But …


.R
R
pkg

make R pkgs for specific data sources

Reduces redundant code

Fewer chances for mistakes

Central community where these pkgs exist (i.e., ropensci)

many niche packages together do great good

& very general purpose pkgs

magick
pdftools
spelling
tesseract
writexl
jqr

drake
eml
assertr
skimr
Rselenium
crul

geojsonio
plotly
charlatan
tic
git2r
visdat

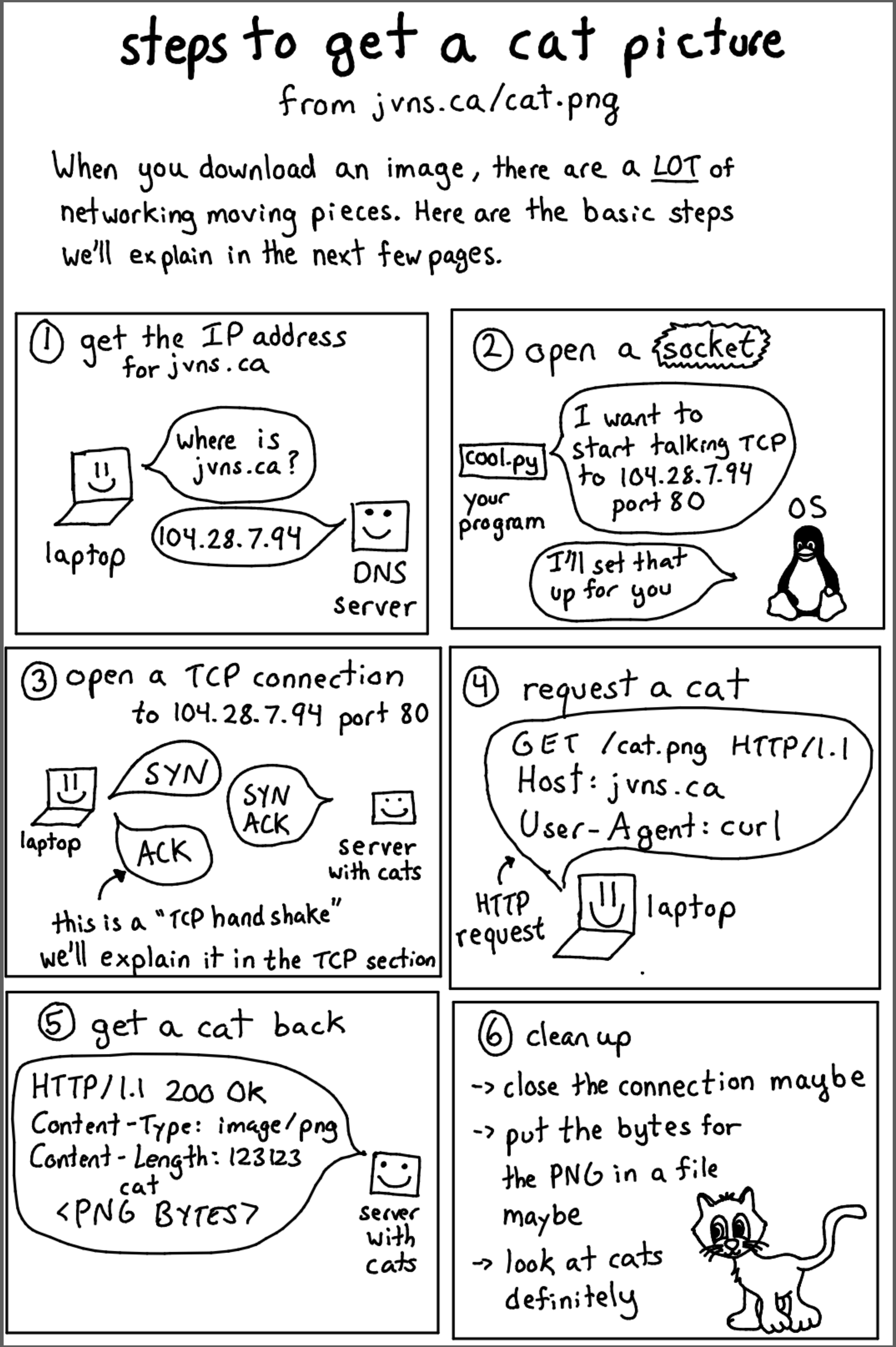
rOpenSci in numbers

6 staff
2 Bioconductor pkgs
192 CRAN pkgs
287 total pkgs
~70K commits
~500 code contributors
LOTS! of community members
> 500 citations in peer-reviewed lit
1 awesome peer review software system

HTTP Requests

Julia Evans

jvns.ca/networking-zine.pdf



steps to get a cat picture
from jvns.ca/cat.png

When you download an image, there are a LOT of networking moving pieces. Here are the basic steps we'll explain in the next few pages.

① get the IP address for jvns.ca

laptop: where is jvns.ca?

DNS server: 104.28.7.94

② open a socket

cool.py (your program): I want to start talking TCP to 104.28.7.94 port 80

OS: I'll set that up for you

③ open a TCP connection to 104.28.7.94 port 80

laptop: SYN

server with cats: SYN ACK

laptop: ACK

this is a "TCP handshake" we'll explain it in the TCP section

④ request a cat

```
GET /cat.png HTTP/1.1
Host: jvns.ca
User-Agent: curl
```

HTTP request

laptop

⑤ get a cat back

```
HTTP/1.1 200 OK
Content-Type: image/png
Content-Length: 123123
 <PNG BYTES>
```

cat

server with cats

⑥ clean up

-> close the connection maybe

-> put the bytes for the PNG in a file maybe

-> look at cats definitely

HTTP Requests: Problems

- Internet access required
- Internet speeds vary, **A LOT**
- Firewalls
- Proxies
- Power outages
- on and on

HTTP Requests: Development

When developing software, you:

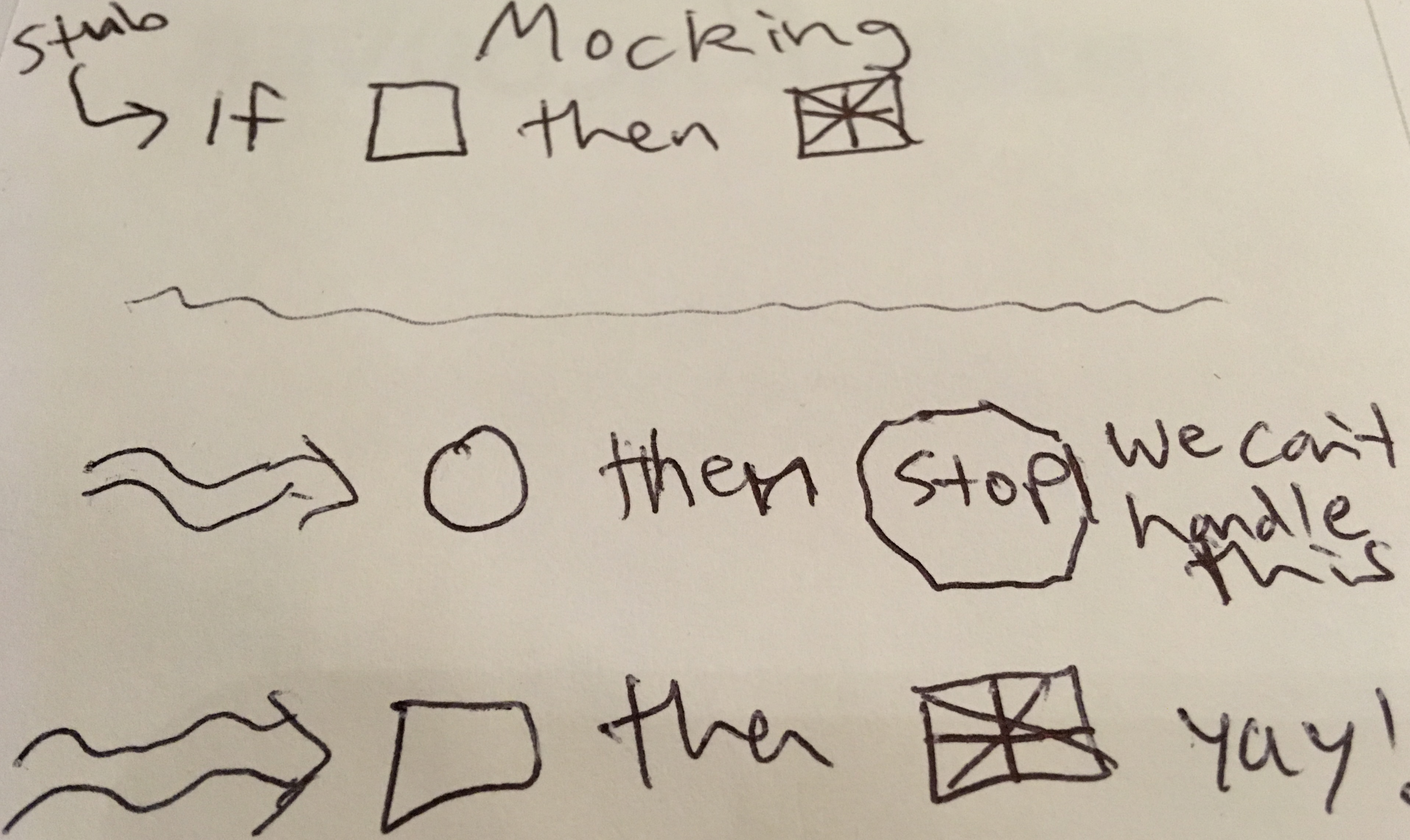
- Want to test functionality of the code
- Not whether:
  - the internet is up or down
  - fast or slow
  - remote server is up or down, misbehaving, etc.

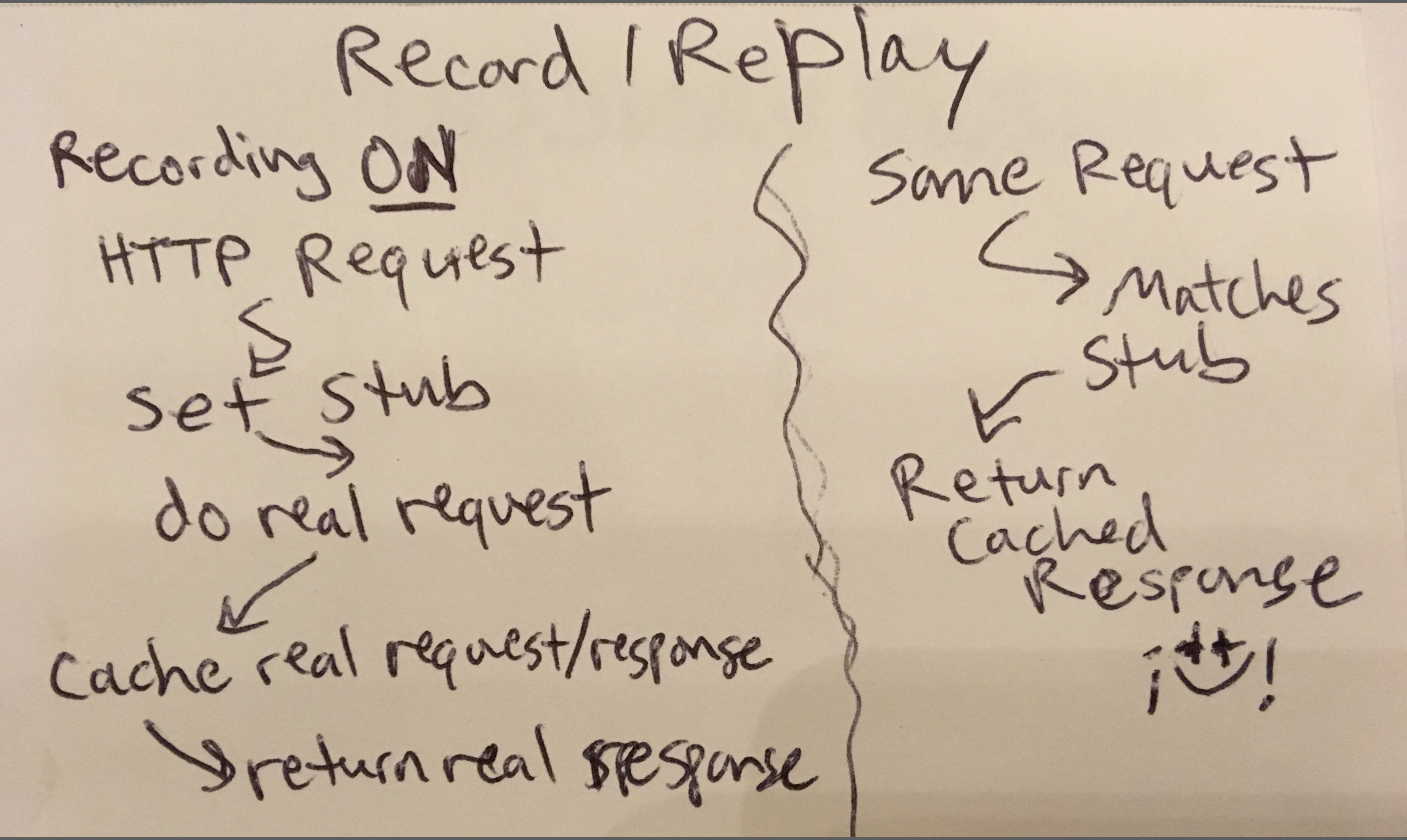
HTTP Requests: Solutions

Mocking

Record/replay

Mocking





Record / Replay

Recording **ON**

HTTP Request

↓

Set Stub

→ do real request

↓

Cache real request/response

→ return real response

Same Request

→ Matches Stub

↓

Return Cached Response

¡ !

Why?

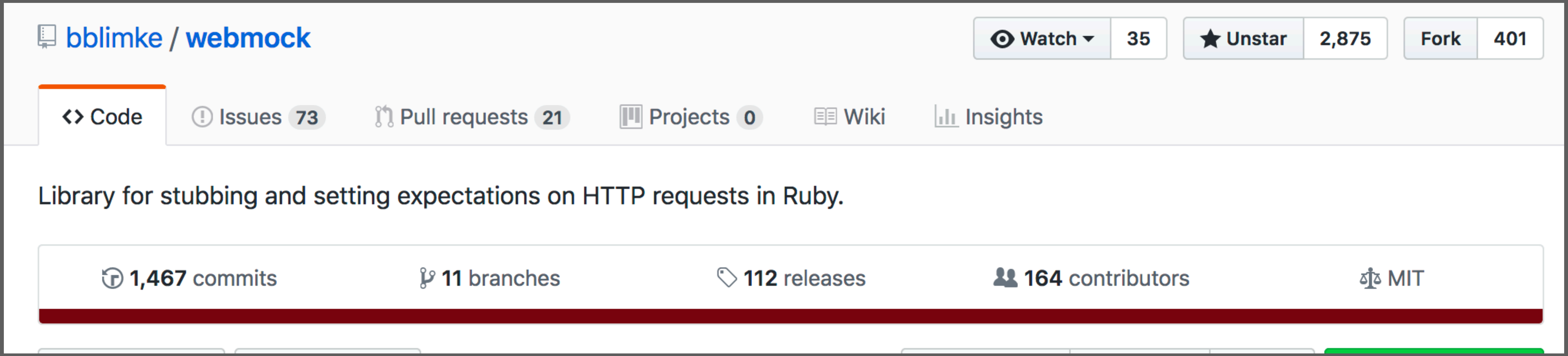
- Mocking:
  - No or slow internet access
  - Simulate situations that are hard to create/re-create (e.g. errors)
  - Return unusual responses
- Record/replay (e.g. for unit tests)
  - Package development cycle increases!
  - Less usage of data providers servers
  - Actually test your software, not internet connections/etc.

Mocking HTTP Requests

Stubbing and setting expectations on HTTP requests

Mocking HTTP Requests

Ported from Ruby's webmock

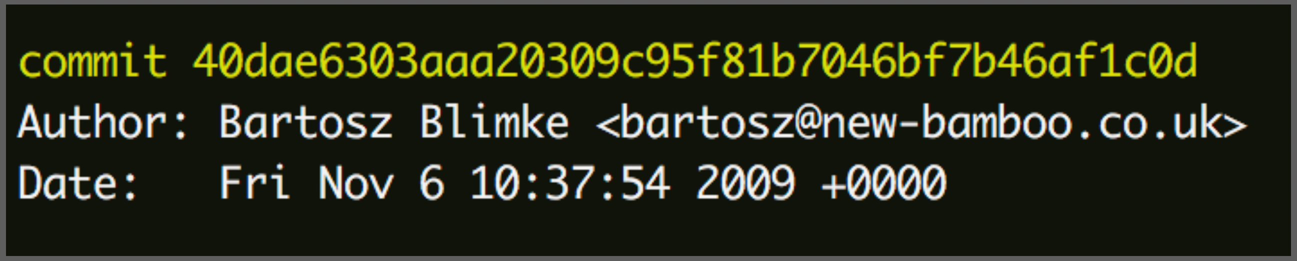

bblimke / webmock

Library for stubbing and setting expectations on HTTP requests in Ruby.

Mocking HTTP Requests

Ported from Ruby's webmock

Take advantage of prior art!
Build on hard-earned knowledge



```
commit 40dae6303aaa20309c95f81b7046bf7b46af1c0d
Author: Bartosz Blimke <bartosz@new-bamboo.co.uk>
Date:   Fri Nov 6 10:37:54 2009 +0000
```

webmockr R package

github.com/ropensci/webmockr

webmockr

- Hooks into an http package. When mocking is enabled, *webmockr* handles the request instead of the http package
- Set stubs - stubs are what's matched against: HTTP method, URL, query parameters, headers
- Optionally set what you'd like returned: status code, body, headers
- Optionally force a timeout or other error to occur

webmockr

- Adapters:
  - crul - github.com/ropensci/crul ✅ (on CRAN)

```
install.packages("webmockr")
```

  - httr - github.com/r-lib/httr ✅ (try it!)

```
remotes::install_github("ropensci/webmockr@adapter-httr")
```

  - curl - github.com/jeroen/curl (try it, but a ways to go)

```
remotes::install_github("ropensci/webmockr@adapter-curl")
```

webmockr

```
# before mocking is turned on
library(crul)
HttpClient$new(url = "https://httpbin.org")$get('get')
#> <crul response>
#>   url: https://httpbin.org/get
#>   request_headers:
#>     User-Agent: libcurl/7.54.0 r-curl/3.2 crul/0.6.0
#>     Accept-Encoding: gzip, deflate
#>     Accept: application/json, text/xml, application/xml, */*
#>   response_headers:
#>     status: HTTP/1.1 200 OK
#>     connection: keep-alive
#>     server: gunicorn/19.9.0
#>     date: Fri, 27 Jul 2018 18:59:09 GMT
#>     content-type: application/json
#>     content-length: 325
#>     access-control-allow-origin: *
#>     access-control-allow-credentials: true
#>     via: 1.1 vegur
#>   status: 200
```

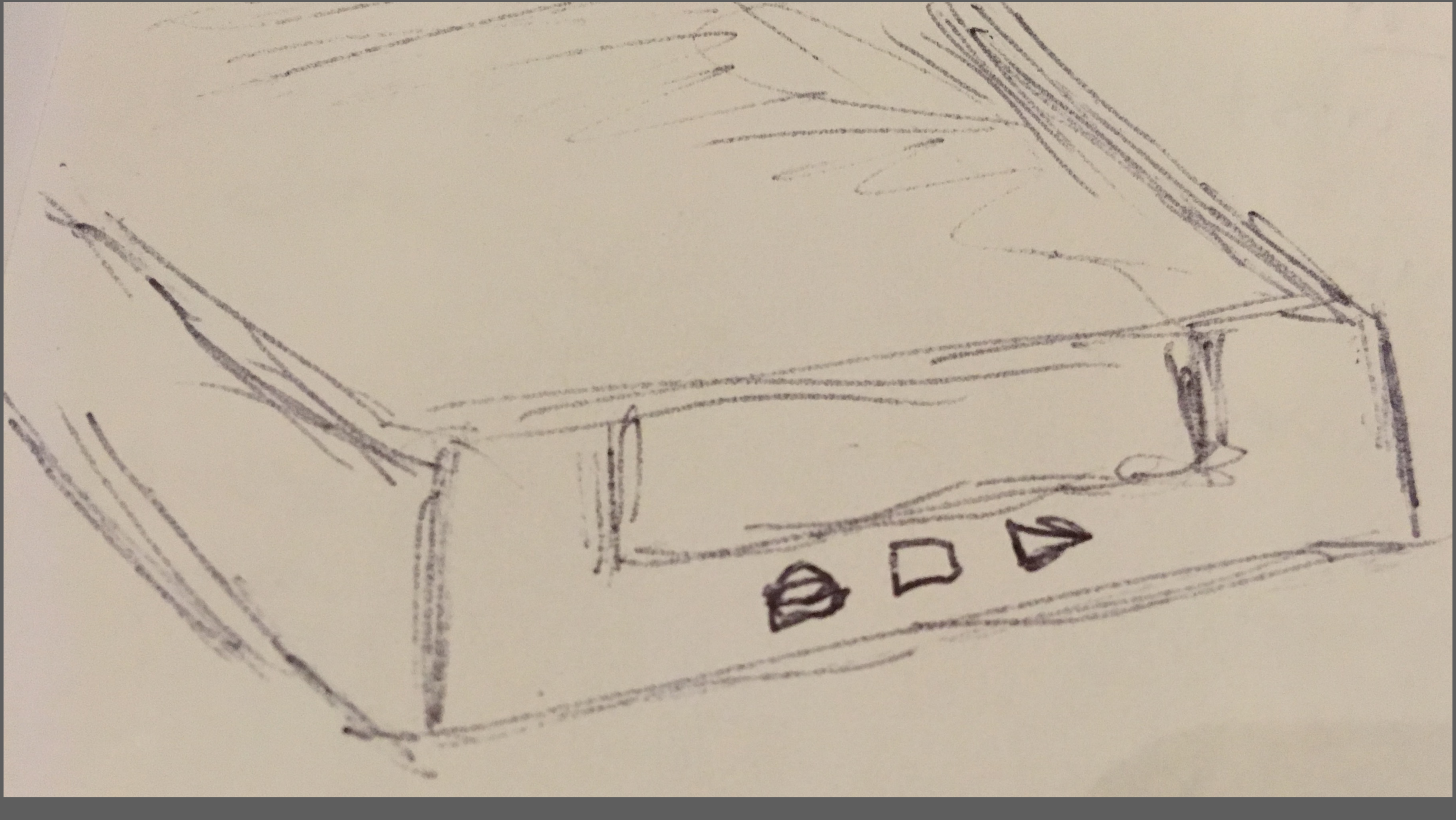
webmockr

```r
# load webmockr, enable
library(webmockr)
webmockr::enable()

# errors now
HttpClient$new(url = "https://httpbin.org")$get('get')
#> Error: Real HTTP connections are disabled.
#> Unregistered request:
#>   GET https://httpbin.org/get   with headers {Accept-Encoding: gzip, deflate,
#>
#> You can stub this request with the following snippet:
#>
#>    stub_request('get', uri = 'https://httpbin.org/get') %>%
#>      wi_th(
#>        headers = list('Accept-Encoding' = 'gzip, deflate', 'Accept' = 'application/json')
#>      )
#> ============================================================
```

webmockr

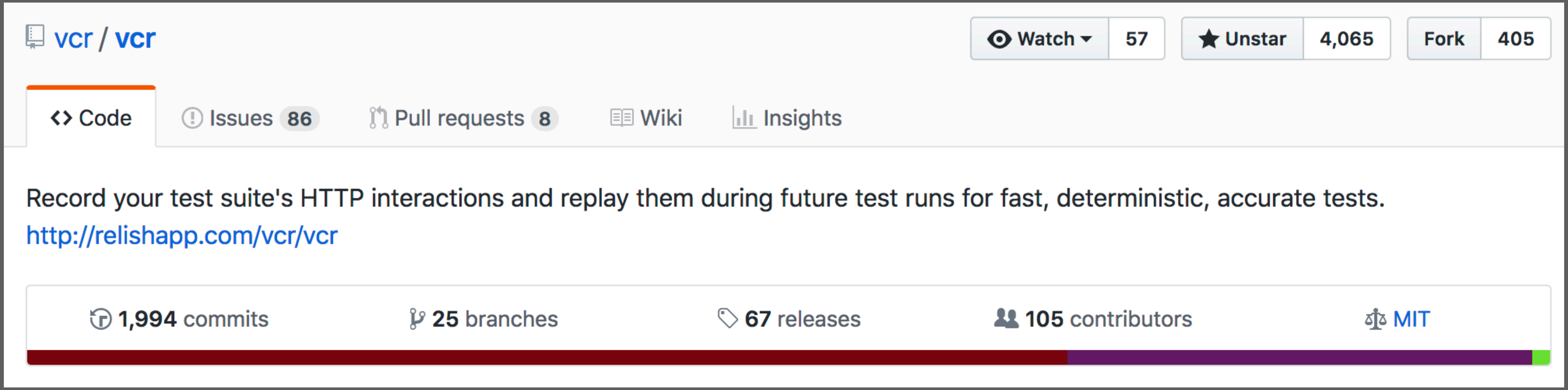
```
# stub registry is empty
stub_registry()
#> <webmockr stub registry>
#>  Registered Stubs

# set stub
stub_request("get", "https://httpbin.org/get") %>%
  wi_th(
    query = list(hello = "world")) %>%
    to_return(status = 418)
# stub registry now not empty
stub_registry()
#> <webmockr stub registry>
#>  Registered Stubs
#>    get: https://httpbin.org/get?hello=world   | to_return:   with status 418

# get mocked request
con <- HttpClient$new(url = "https://httpbin.org")
con$get('get', query = list(hello = "world"))
#> <crul response>
#>   url: https://httpbin.org/get?hello=world
#>   request_headers:
#>     User-Agent: libcurl/7.54.0 r-curl/3.2 crul/0.6.0
#>     Accept-Encoding: gzip, deflate
#>     Accept: application/json, text/xml, application/xml, */*
#>   response_headers:
#>   params:
#>     hello: world
#>   status: 418
```

let’s talk about vcr’s!



Record/Replay HTTP Requests

Ported from Ruby’s vcr


vcr / vcr

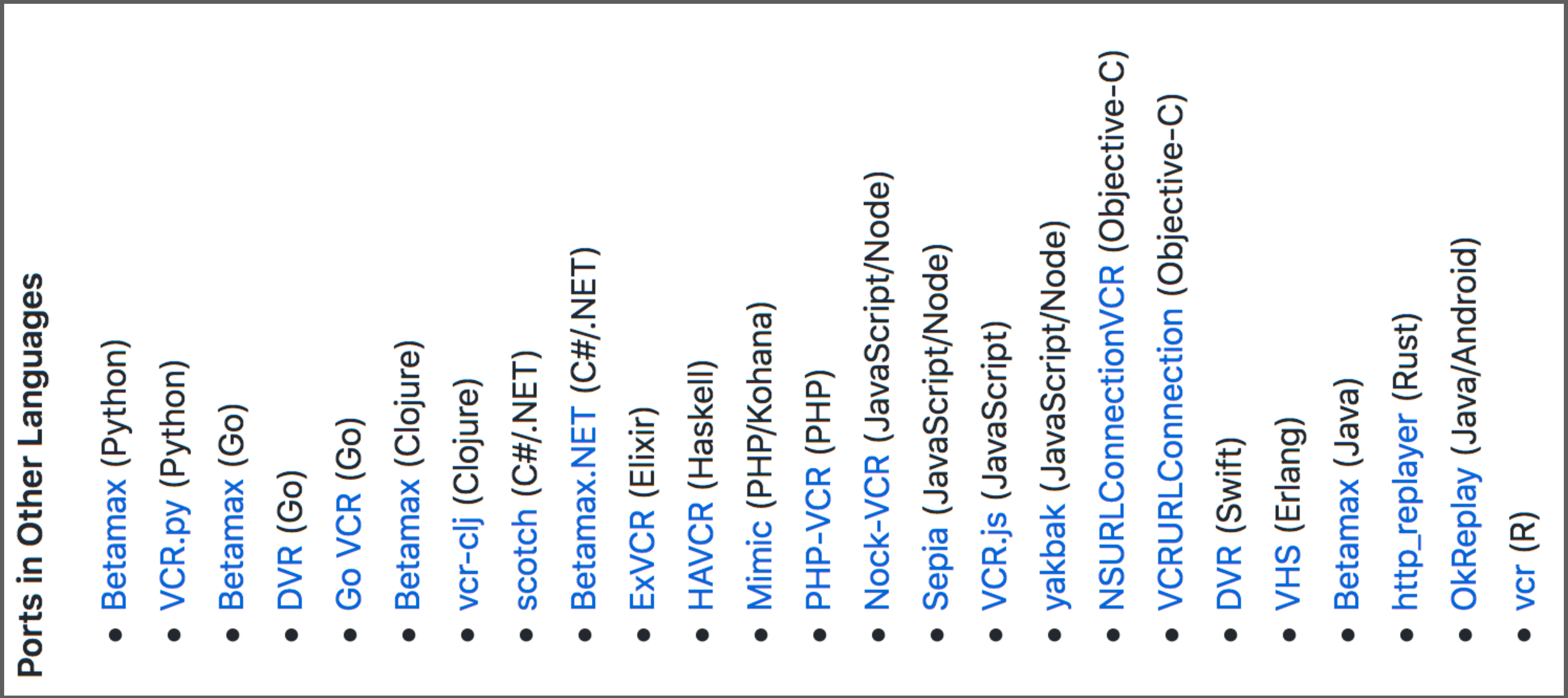
Record your test suite's HTTP interactions and replay them during future test runs for fast, deterministic, accurate tests.
http://relishapp.com/vcr/vcr

1,994 commits | 25 branches | 67 releases | 105 contributors | MIT

Record/Replay HTTP Requests

Ported from Ruby's vcr

Take advantage of prior art!
Build on hard-earned knowledge



Ports in Other Languages

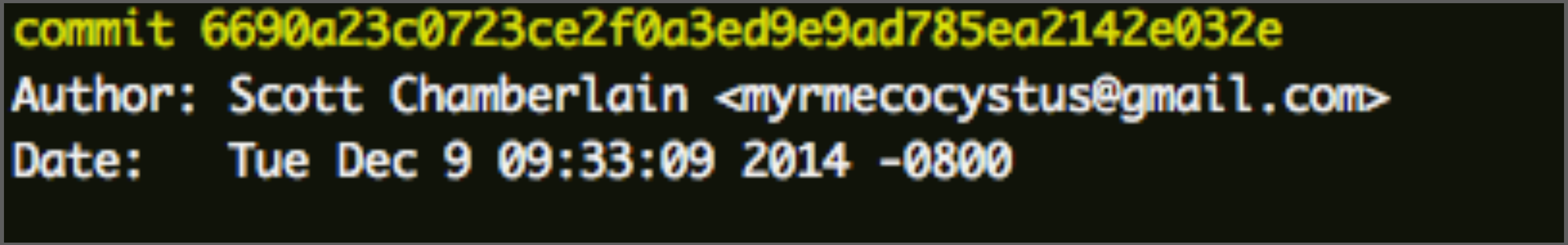
- Betamax (Python)
- VCR.py (Python)
- Betamax (Go)
- DVR (Go)
- Go VCR (Go)
- Betamax (Clojure)
- vcr-clj (Clojure)
- scotch (C#/.NET)
- Betamax.NET (C#/.NET)
- ExVCR (Elixir)
- HAVCR (Haskell)
- Mimic (PHP/Kohana)
- PHP-VCR (PHP)
- Nock-VCR (JavaScript/Node)
- Sepia (JavaScript/Node)
- VCR.js (JavaScript)
- yakbak (JavaScript/Node)
- NSURLConnectionVCR (Objective-C)
- VCRURLConnection (Objective-C)
- DVR (Swift)
- VHS (Erlang)
- Betamax (Java)
- http_replayer (Rust)
- OkReplay (Java/Android)
- vcr (R)

vcr R package

github.com/ropensci/vcr

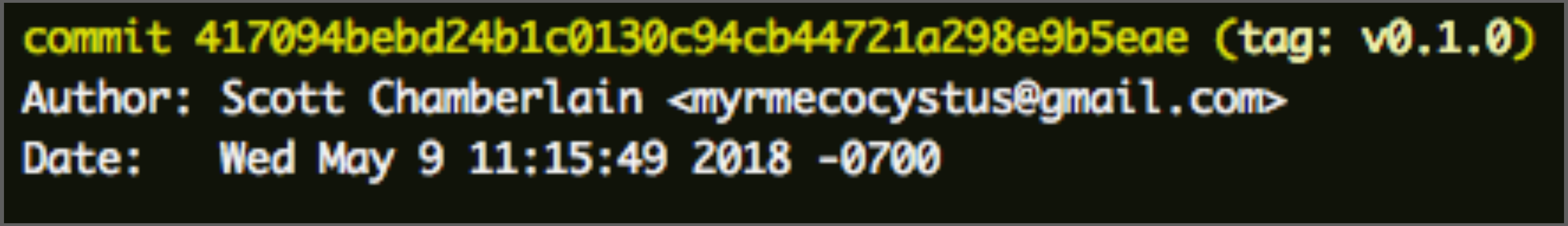
Record/Replay HTTP requests

- *webmockr* does not cache
- *vcr*, which does cache, leverages *webmockr* for the matching
- *vcr* can be used to:
    - Cache individual/one off HTTP requests
    - Unit tests in packages/projects



```
commit 6690a23c0723ce2f0a3ed9e9ad785ea2142e032e
Author: Scott Chamberlain <myrmecocystus@gmail.com>
Date:   Tue Dec 9 09:33:09 2014 -0800
```

Ouch!
3 years & 5 months to 1st CRAN version



```
commit 417094bebd24b1c0130c94cb44721a298e9b5eae (tag: v0.1.0)
Author: Scott Chamberlain <myrmecocystus@gmail.com>
Date:   Wed May 9 11:15:49 2018 -0700
```

vcr

- Hooks into an http package (e.g., *crul*) by using *webmockr*. When *vcr* is enabled, real requests are allowed, and *vcr* caches real responses
- *vcr* works no matter how many layers down the HTTP request is made

vcr

- Adapters:
  - crul - github.com/ropensci/crul ✅ (on CRAN)

```
install.packages("vcr")
```

  - httr - github.com/r-lib/httr ✅ (on GitHub)

```
remotes::install_github("ropensci/vcr@httr-integration")
```

  - curl - github.com/jeroen/curl (not ready yet 😞)

vcr: requirements

- DESCRIPTION file: add *vcr* to Suggests (don’t need to include *webmockr*)
- Add *vcr::vcr_configure()* call in tests/ dir
- Wherever needed, wrap requests that make HTTP requests in *vcr::use_cassette()*

vcr: setup

basic setup

```r
# tests/testthat/helper-packagename.R
library("vcr")
vcr_configure(dir = "../fixtures")
```

secure strings

```r
# tests/testthat/helper-packagename.R
library("vcr")
vcr_configure(
    dir = "../fixtures",
    filter_sensitive_data = list("<<rredlist_api_token>>" = Sys.getenv('IUCN_REDLIST_KEY'))
)
```

vcr: usage

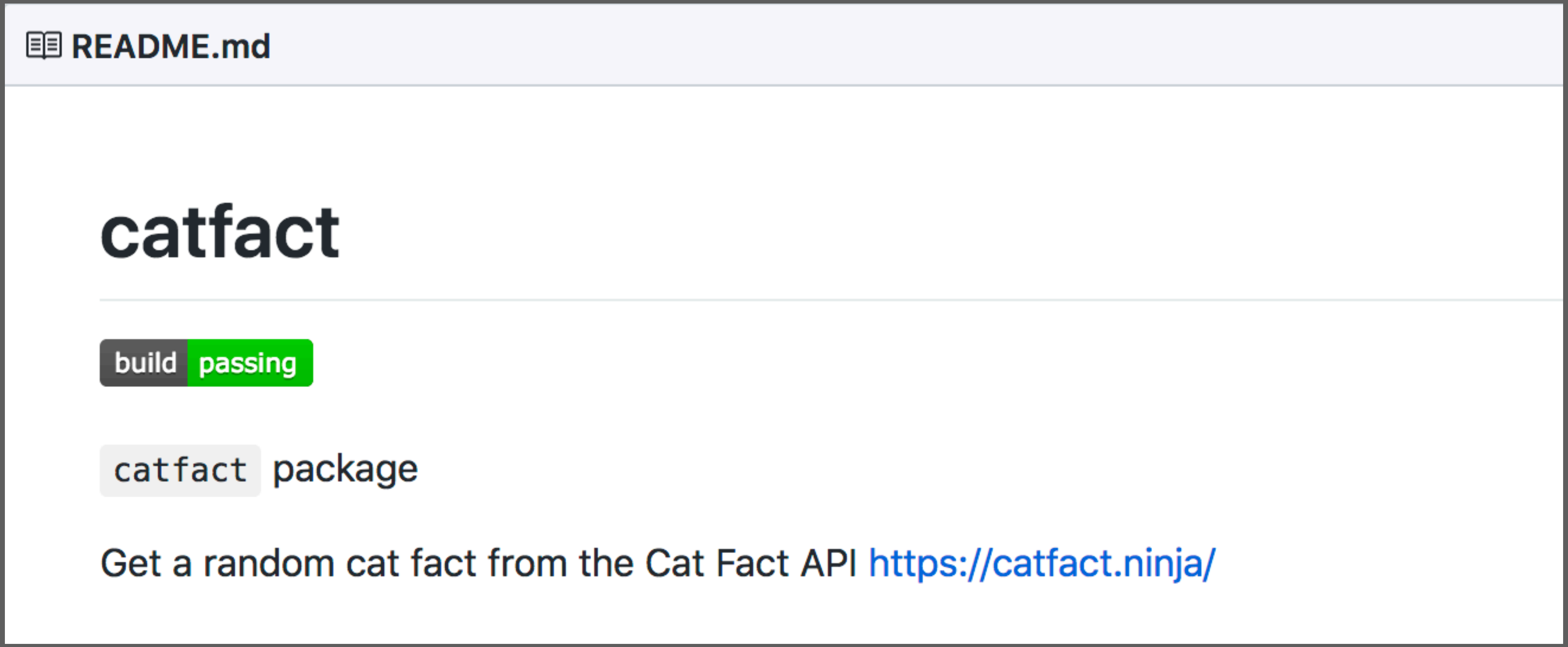
```r
# tests/testthat/test-rl_history.R
vcr::use_cassette("rl_history", {
  aa <- rl_history('Loxodonta africana')

  expect_is(aa, "list")
  expect_is(aa$name, "character")
  expect_is(aa$result, "data.frame")
  expect_true(any(grepl("Vulnerable", aa$result$category, ignore.case = TRUE)))
})
```

vcr: cassette

```yaml
# tests/fixtures/rl_history.yml
http_interactions:
- request:
    method: get
    uri: http://apiv3.iucnredlist.org/api/v3/species/history/name/Loxodonta%20africana?token=<<rredlist_api_token>>
    body:
      encoding: ''
      string: ''
    headers:
      User-Agent: r-curl/3.2 crul/0.5.2 rOpenSci(rredlist/0.4.1.9120)
      Accept-Encoding: gzip, deflate
      Accept: application/json, text/xml, application/xml, */*
  response:
    status:
      status_code: '200'
      message: OK
      explanation: Request fulfilled, document follows
    headers:
      status: HTTP/1.1 200 OK
      server: nginx/1.1.19
      date: Thu, 17 May 2018 18:36:03 GMT
      content-type: application/json; charset=utf-8
      content-length: '400'
      connection: keep-alive
      x-powered-by: Sails <sailsjs.org>
      vary: Accept-Encoding
    body:
      encoding: UTF-8
      string: '{"name":"Loxodonta africana","result":[{"year":"2008","code":"VU","category":"Vulnerable"},
        {"year":"2004","code":"VU","category":"Vulnerable"},{"year":"1996","code":"EN","category":"Endangered"},
        {"year":"1994","code":"V","category":"Vulnerable"},{"year":"1990","code":"V","category":"Vulnerable"},
        {"year":"1988","code":"V","category":"Vulnerable"},{"year":"1986","code":"V","category":"Vulnerable"}]}'
  recorded_at: 2018-05-17 18:36:03 GMT
recorded_with: vcr/0.1.0, webmockr/0.2.6, crul/0.5.2
```

minimal eg pkg. w/ httr+vcr

github.com/sckott/catfact

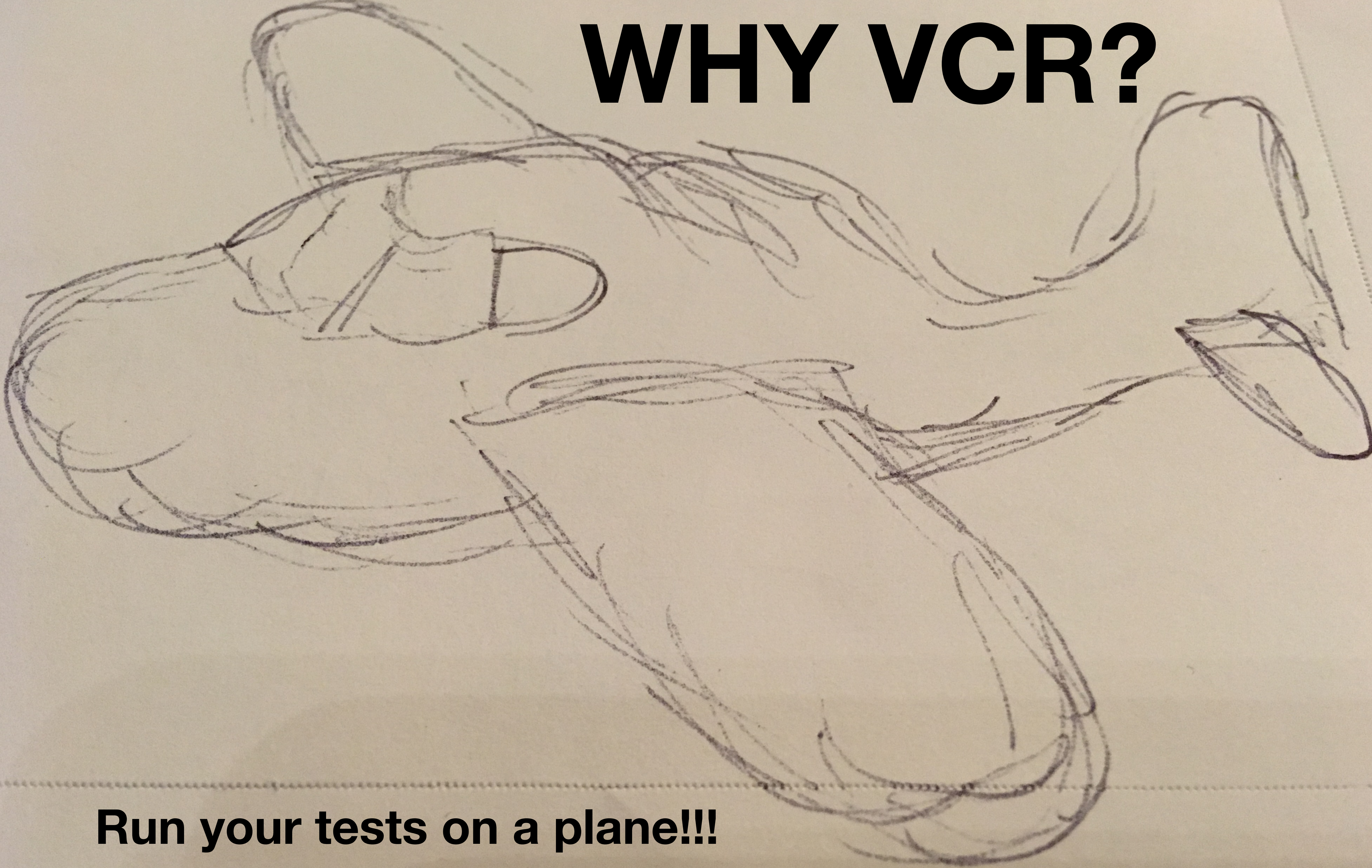

README.md

catfact

`catfact` package

Get a random cat fact from the Cat Fact API https://catfact.ninja/

WHY VCR?



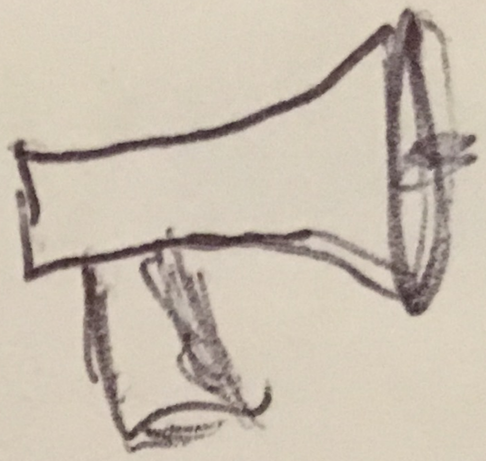
Run your tests on a plane!!!

src/

github.com/ropensci/vcr

github.com/ropensci/webmockr

talk to us



forum

discuss.ropensci.org

submit/review package

onboarding.ropensci.org

scotttalks.info/jsm2018