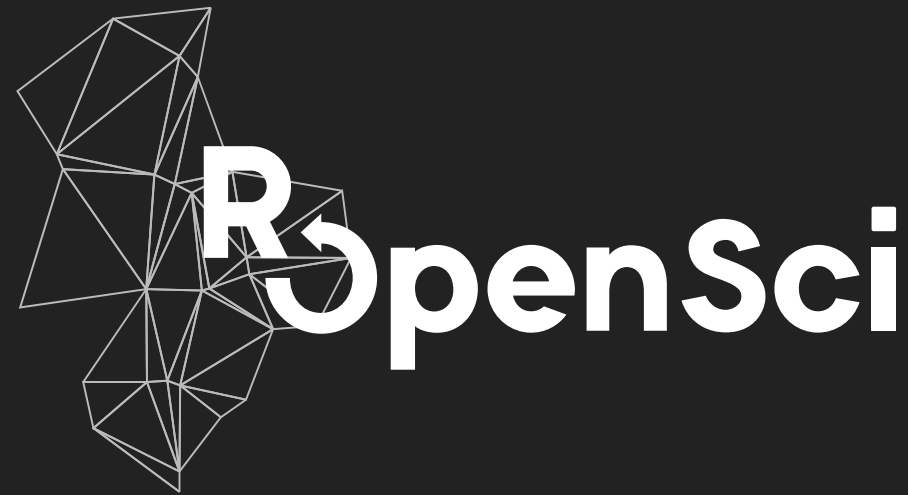


HTTP Requests for Users & Package Developers

Scott Chamberlain (🐦 [@sckottie](https://twitter.com/sckottie))



THE LEONA M. AND HARRY B.

HELMSLEY

CHARITABLE TRUST



3 packages: crul, webmockr, vcr

rOpenSci has a lot of pkgs that do http requests
giving rise to the tools presented here

curl - a new http client

 [ropensci/curl](https://github.com/ropensci/curl)

curl - features

- asynchronous requests
- pagination
- supports mocking and caching
- writing to disk + streaming
- request + response hooks

- does not have: OAuth

crul - lots of example usage

Reverse imports: [bold](#), [brranching](#), [ccafs](#), [codemetar](#), [crminer](#), [discgolf](#), [duckduckr](#), [elastic](#), [fulltext](#), [geojsonlint](#), [getlandsat](#), [handlr](#), [jaod](#), [microdemic](#), [nasapower](#), [natserv](#), [nsapi](#), [openadds](#), [originr](#), [pangaeear](#), [pleiades](#), [postlightmercury](#), [rbhl](#), [rbison](#), [rbraries](#), [rcitoid](#), [rcoreoa](#), [rcrossref](#), [rdatacite](#), [rdpla](#), [rdryad](#), [rerddap](#), [rgbif](#), [rif](#), [ritis](#), [rjsonapi](#), [rnoaa](#), [ropenaq](#), [rorcid](#), [rphylopic](#), [rplos](#), [rredlist](#), [rsnps](#), [rsunlight](#), [rtimes](#), [rvertnet](#), [searoundus](#), [sofa](#), [solrium](#), [spocc](#), [taxize](#), [tradestatistics](#), [traits](#), [vcr](#), [webmockr](#), [wikitaxa](#), [worrms](#), [zbank](#)

Reverse suggests: [fauxpas](#), [finch](#)

crul demo

```
con <- crul::HttpClient$new(url = "https://httpbin.org")
con$get(path = "get")
```

```
<crul response>
url: https://httpbin.org/get
request_headers:
  User-Agent: libcurl/7.54.0 r-curl/3.3 crul/0.7.4

  Accept-Encoding: gzip, deflate
  Accept: application/json, text/xml, application/xml, */*
response_headers:
  status: HTTP/1.1 200 OK
  access-control-allow-credentials: true
  access-control-allow-origin: *
  content-encoding: gzip
  content-type: application/json
  date: Wed, 12 Jun 2019 23:21:09 GMT
  referrer-policy: no-referrer-when-downgrade
  server: nginx
  x-content-type-options: nosniff
  x-frame-options: DENY
  x-xss-protection: 1; mode=block
  content-length: 218
  connection: keep-alive
status: 200
```

Returns an R6 object

curl demo

Index to results and methods with \$

```
res$request  
res$content  
res$times  
res$modified  
res$response_headers_all  
res$response_headers  
res$request_headers  
res$status_code  
res$handle  
res$opts  
res$url  
res$method  
res$clone()  
res$raise_for_status()  
res$status_http()  
res$success()  
res$parse()  
res$initialize()  
res$print()
```

crul asynchronous

Same http options for every URL

```
cc <- Async$new(  
  urls = c(  
    'https://httpbin.org/get',  
    'https://httpbin.org/get?a=5',  
    'https://httpbin.org/get?foo=bar'  
  )  
)  
res <- cc$get()  
vapply(res, function(z) z$parse("UTF-8"), "")  
#> [1] "{\n  \"args\": {}, \n  \"headers\": {\n    \"Accept\": \"application/json  
#> [2] "{\n  \"args\": {\n    \"a\": \"5\"\n  }, \n  \"headers\": {\n    \"Accept  
#> [3] "{\n  \"args\": {\n    \"foo\": \"bar\"\n  }, \n  \"headers\": {\n    \"Ac
```

Async varied: custom http options for every request

```
req1 <- HttpRequest$new("https://httpbin.org/get", headers = list(a="b"))$get()  
req2 <- HttpRequest$new("https://httpbin.org/post")$post()  
out <- AsyncVaried$new(req1, req2)  
out$parse()  
#> [1] "{\n  \"args\": {}, \n  \"headers\": {\n    \"Accept\": \"application/json  
#> [2] "{\n  \"args\": {}, \n  \"data\": \"\", \n  \"files\": {}, \n  \"form\": {
```

crul pagination

```
cli <- HttpClient$new(url = "https://api.crossref.org")
cc <- Paginator$new(client = cli, limit_param = "rows",
  offset_param = "offset", limit = 50, limit_chunk = 10)
cc$get('works')
cc
#> <crul paginator>
#> base url: https://api.crossref.org
#> by: query_params
#> limit_chunk: 10
#> limit_param: rows
#> offset_param: offset
#> limit: 50
#> progress: FALSE
#> status: 5 requests done
```

```
cc$status_code()
#> [1] 200 200 200 200 200
```

```
cc$responses()
cc$parse()
etc ...
```

Only supports pagination done via query parameters
Link headers and cursors to come

curl request/response hooks

- request hook: run *before* the request occurs
- response hook: run *once* the request is done

request and response hooks example

```
fun_req <- function(request) {  
  cat(paste0("Requesting: ", request$url$url, " at ", as.character(Sys.time())),  
      sep = "\n")  
}  
fun_res <- function(response) {  
  cat(paste0("status_code: ", response$status_code), sep = "\n")  
}  
x <- HttpClient$new(url = "https://httpbin.org",  
  hooks = list(request = fun_req, response = fun_res))
```

```
invisible(x$get('get'))  
#> Requesting: https://httpbin.org/get at 2019-07-06 02:10:38  
#> status_code: 200
```

Mocking/caching

webmockr & vcr:

forked ↻ from another language (Ruby)

we can take advantage of all they've learned

& both general purpose

work with current and future http pkgs

Other langs

keep an eye  out for other languages

what good ideas can we adopt in R land

webmockr - mock http requests

arose: because needed to make vcr

 [ropensci/webmockr](https://github.com/ropensci/webmockr)

webmockr - what does it do?

set what you want to **match against** & what to **return**

make a request

if it matches you get what you set to **return**

if it doesn't match: **error**

webmockr - huh?

webmockr hooks into crul, **hijacking** the normal request

constructing a response that matches a real response

based on what you told webmockr to respond with

& vcr builds on webmockr ...

webmockr - example

```
library(crul)
library(webmockr)
```

```
stub_request("get", "https://httpbin.org/get") %>%
  with(query = list(hello = "world")) %>% to_return(status = 418)
#> <webmockr stub>
#>   method: get
#>   uri: https://httpbin.org/get
#>   with:
#>     query: hello=world
#>     body:
#>     request_headers:
#>     to_return:
#>       status: 418
#>       body:
#>     response_headers:
#>     should_timeout: FALSE
#>     should_raise: FALSE
```

```
HttpClient$new()$get(path = 'get', query = list(hello = "world"))
#> <crul response>
#>   url: https://httpbin.org/get?hello=world
#>   request_headers:
#>     User-Agent: libcurl/7.54.0 r-curl/3.3 crul/0.7.0.9310
#>     Accept-Encoding: gzip, deflate
#>     Accept: application/json, text/xml, application/xml, */*
#>   response_headers:
#>   params:
#>     hello: world
#>   status: 418
```

webmockr - no matching stub

```
library(httr)
GET("https://httpbin.org/get")
#> Error: Real HTTP connections are disabled.
#> Unregistered request:
#>   GET https://httpbin.org/get   with headers
#>     {Accept: application/json, text/xml, application/xml, */*}
#>
#> You can stub this request with the following snippet:
#>
#>   stub_request('get', uri = 'https://httpbin.org/get') %>%
#>     wi_th(
#>       headers = list(
#>         'Accept' = 'application/json, text/xml, application/xml, */*'
#>       )
#>     )
```

usage in the wild

```
upload_file_job_json <- jsonlite::read_json("upload-file-job-2.json")
mockery::stub(upload_forecast, 'httr::upload_file', NULL)
stub_request('post', uri='http://example.com/api/model/1/forecasts/') %>%
  to_return(
    body=upload_file_job_json,
    status=200,
    headers=list('Content-Type'='application/json; charset=utf-8')
  )
```

src: <https://github.com/reichlab/zoltr>

```
test_that('create_database works with mock', {
  stub_request("post", "https://api.treasuredata.com/v3/database/create/test") %>%
    to_return(body = "{}", status = 200)
  expect_true(create_database(conn, "test"))
})
```

src: <https://github.com/cran/RTD>

Note - mocking requests with crul/httr inside of other fxns

expect failures?!

Expectation to timeout

```
library(crul)
library(webmockr)
crul::mock()

stub_request("get", "https://httpbin.org/get") %>% to_timeout()
x <- HttpClient$new(url = "https://httpbin.org")

x$get('get')
#> Error: Request Timeout (HTTP 408).
#> - The client did not produce a request within the time that the server
#>   was prepared to wait. The client MAY repeat the request without
#>   modifications at any later time.
```

Expectation to raise exception

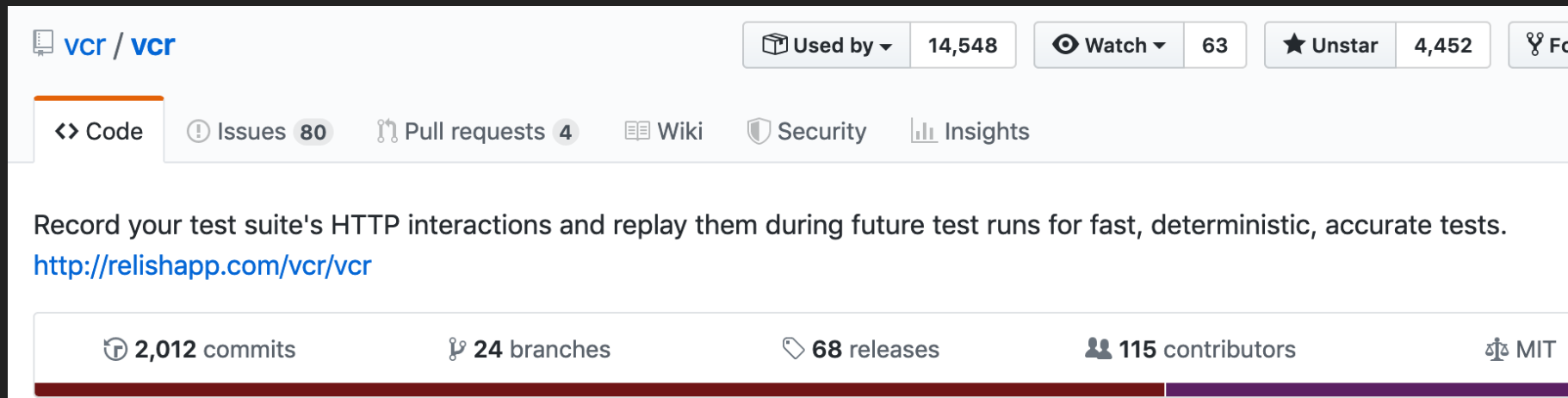
```
library(fauxpas)
stub_request("get", "https://httpbin.org/get") %>%
  to_raise(fauxpas::HTTPBadGateway)
HttpClient$new(url = "https://httpbin.org")$get("get")
#> Error: Bad Gateway (HTTP 502).
#> - The server, while acting as a gateway or proxy, received an invalid
#>   response from the upstream server it accessed in attempting to
#>   fulfill the request.
```

vcr - record and replay HTTP requests/responses

arose: observing other language communities &
need to improve testing in many API clients



vcr - hardest software project I've worked on



The screenshot shows the GitHub repository page for `vcr / vcr`. The repository is owned by MIT. The page displays the following statistics:

- Used by: 14,548
- Watch: 63
- Unstar: 4,452
- Issues: 80
- Pull requests: 4
- Wiki
- Security
- Insights

The repository description is: "Record your test suite's HTTP interactions and replay them during future test runs for fast, deterministic, accurate tests." The URL <http://relishapp.com/vcr/vcr> is provided. The repository has 2,012 commits, 24 branches, 68 releases, and 115 contributors.

Category	Count
Used by	14,548
Watch	63
Unstar	4,452
Issues	80
Pull requests	4
Commits	2,012
Branches	24
Releases	68
Contributors	115

vcr - hardest software project I've worked on

Ruby

```
def has_interaction_matching?(request)
  !!matching_interaction_index_for(request) ||
  !!matching_used_interaction_for(request) ||
  @parent_list.has_interaction_matching?(request)
end
```

R

```
has_interaction_matching = function(request) {
  private$matching_interaction_bool(request) ||
  private$matching_used_interaction_for(request) ||
  self$parent_list$has_interaction_matching()
}
```


vcr - no monkey patching in R!

Allowed in Ruby, but not in R

in R we can do

```
assignInNamespace("some_object", value = function(e) e, ns = "some_other_pkg")
```

But not allowed on CRAN

vcr - how does it work?



vcr - how does it work?

I thought vcr worked by listening  for requests in R

realized it most definitely did not

it modifies an HTTP request & looks for a match

so had to make `webmockr` first

vcr - what does it do?

HTTP requests in a test suite as usual

w/o making real HTTP requests

so you test your package

not the remote service

(p.s. great for rate-limited services)

what is a cassette?

```
http_interactions:  
- request:  
  method: get  
  uri: http://www.marinespecies.org/rest/AphiaExternalIDByAphiaID/1080?type=tsn  
  body:  
    encoding: ''  
    string: ''  
  headers:  
    User-Agent: libcurl/7.54.0 r-curl/3.3 crul/0.8.0  
  
    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  
    date: Fri, 28 Jun 2019 16:55:51 GMT  
    server: Apache/2.4.25 (Win32) PHP/5.6.29  
    x-powered-by: PHP/5.6.29  
    access-control-allow-origin: '*'  
    access-control-allow-headers: X-Requested-With, Content-Type, Accept, Origin  
    Authorization  
    access-control-allow-methods: GET, POST, OPTIONS  
    content-length: '9'  
    content-type: application/json  
  body:  
    encoding: UTF-8  
    string: '["85257"]'  
recorded_at: 2019-06-28 16:55:51 GMT
```

vcr - a brief example

```
library(vcr)
library(crul)

cli <- crul::HttpClient$new(url = "https://api.crossref.org")
use_cassette(name = "helloworld", {
  res1 <- cli$get("works", query = list(rows = 3))
})
```

Do the request again

```
use_cassette(name = "helloworld", {
  res2 <- cli$get("works", query = list(rows = 3))
})
```

Identical responses

```
identical(res1$parse(), res2$parse())
#> [1] TRUE
```

speeds up your tests

w/o vcr

```
→ Rscript -e 'devtools::test()'
Testing worrms
✓ | OK F W S | Context
✓ | 15       | wm_children [10.0 s]
✓ | 6        | wm_classification [1.4 s]
✓ | ..      | ...
== Results ==
Duration: 141.0 s
```

w/ vcr

```
→ Rscript -e 'devtools::test()'
Testing worrms
✓ | OK F W S | Context
✓ | 15       | wm_children [3.8 s]
✓ | 6        | wm_classification [0.5 s]
✓ | ..      | ...
== Results ==
Duration: 35.6 s
```

vcr - in the works

- JSON cassettes
- `testthat` reporter for cassette usage
- dates 🗓️ 🗓️
- data security 🔒, always more to do
- responses written to disk
- docs: [http testing book - bit.ly/http-testing](http://testingbook.com)
- many more

further reading



HTTP Testing Book:
bit.ly/http-testing

crul/webmockr/vcr in detail
w/ caveats/edge cases/etc.

slides: scotttalks.info/user-http

Made w/: [reveal.js v3.7.0](#), [FontAwesome v5.7.2](#)