Hurl HTTP requests as tests

START HERE

`hurl <file.hurl>`
Run a file, print last body

`hurl --test <file.hurl>`
Run as test, show summary

`hurl --variable <name>=<value> <file>`
Inject a variable

`hurl -v <file>`
Show request and response

`hurl --very-verbose <file>`
Debug incl. bodies

`hurl --test --report-html <dir> <files>`
HTML test report

`hurl --retry <n> <file>`
Retry failed entries

`hurl --glob "<pattern>"`
Run files by pattern

Request METHOD URL, headers, body → `HTTP` → **Response** HTTP 200 → `[Captures]` → **Variables** {{name}} → `next entry` → **[Asserts]** pass or fail, exit code

CONCEPTS

A .hurl file is a list of entries: a request, then the response you expect.

Entries run in order, so a capture in one is usable as {{name}} in the next.

Without --test, only the last response body is printed; asserts still run and set the exit code.

Comments start with #. To send a literal # in a header value, write \#.

RUN

`hurl <file.hurl>`
Run a file

`hurl <a.hurl> <b.hurl>`
Run several files in order

`hurl --test <files>`
Test mode: progress and summary

`hurl --glob "<pattern>"`
Run files matching a glob

`hurl --continue-on-error <file>`
Do not stop at the first failed assert

`hurl --retry <n> --retry-interval <ms> <file>`
Retry failed entries

REQUEST

`GET <url>`
First line: method and URL

`<Name>: <value>`
Header lines straight after the URL

`[Query] <key>: <value>`
Query string params, unencoded

`[Form] <key>: <value>`
URL-encoded form fields

`[BasicAuth] <user>: <password>`
Basic authentication

`[Options] <option>: <value>`
Per-request options

`{ "key": "value" }`
JSON body, Content-Type set for you

REQUEST (cont.)

```` ```json ... ``` ````
Multiline body, with templates

RESPONSE & ASSERTS

`HTTP 200`
Expect this status

`[Asserts]`
Section of checks, one per line

`status < 300`
Status check

`header "<Name>" contains "<text>"`
Header check

`jsonpath "$.a.b" == "<value>"`
Check a JSON value

`jsonpath "$.items" count == <n>`
Check list length

`xpath "//h1" exists`
Check HTML or XML

`body contains "<text>"`
Substring of the body

`jsonpath "$.x" not exists`
Negate any predicate with not

PREDICATES

`== != > >= < <=`
Compare values

`startsWith endsWith contains`
String and list checks

`matches /<re>/`
Regex match

`exists isEmpty`
Presence checks

`isString isInteger isBoolean`
Type checks (also isFloat, isList...)

Prefix any predicate with not, e.g. jsonpath "$.x" not exists.

CAPTURES & VARIABLES

`[Captures]`
Section that stores response values

`id: jsonpath "$.id"`
Capture a JSON value

`tok: header "X-Token"`
Capture a header

CAPTURES & VARIABLES (cont.)

`pass: header "token" redact`
Hide a captured value in logs

`{{name}}`
Use a variable in URL, header, body

`hurl --variable <name>=<value>`
Set a variable

`hurl --variables-file <file>`
Variables from name=value lines

`hurl --secret <name>=<value>`
Variable redacted from logs

FILTERS

Filters chain left to right: header "x" split "," nth 0 == "a".

HTTP OPTIONS

`-L, --location`
Follow redirects

`-H, --header <Name:value>`
Add a header to every request

`-u, --user <user:password>`
Basic authentication

`-k, --insecure`
Skip TLS verification

`--compressed`
Ask for and decode compression

`--resolve <host:port:addr>`
Override DNS for a host

`-b <file> / -c <file>`
Read or write cookies (Netscape)

OUTPUT & REPORTS

`-v, --verbose`
Request and response details on stderr

`--very-verbose`
Debug output with bodies

`-i, --include`
Print response headers too

`-o, --output <file>`
Write the body to a file

`--json`
Output results as JSON

OUTPUT & REPORTS (cont.)

`--error-format long`
Detailed error messages

`-h, --help / -V, --version`
Help and version

`--report-html <dir>`
HTML report

`--report-junit <file>`
JUnit XML for CI

Exit codes

Code	Meaning
0	Success
1	Bad command-line options
2	Hurl file parse error
3	Runtime error (e.g. connection)
4	Assert failed

A secret is redacted from logs and reports, but not from response bodies written to stdout.

RECIPES

Login, capture a token, call the API

```
POST https://api.example.com/login
{ "user": "{{user}}", "pass": "{{pass}}" }
HTTP 200
[Captures]
token: jsonpath "$.token" redact

GET https://api.example.com/me
Authorization: Bearer {{token}}
HTTP 200
[Asserts]
jsonpath "$.name" == "{{user}}"
```

Poll until a job is done

```
GET https://api.example.com/jobs/{{id}}
[Options]
retry: 10
retry-interval: 2000
HTTP 200
[Asserts]
jsonpath "$.status" == "done"
```

CI run with a JUnit report

```
hurl --test --variable host=https://staging.example.com \
  --report-junit build/hurl.xml tests/*.hurl
```

Upload a file with a query parameter

```
POST https://api.example.com/upload
[Query]
folder: docs
[Multipart]
file: file,report.pdf; application/pdf
HTTP 201
```