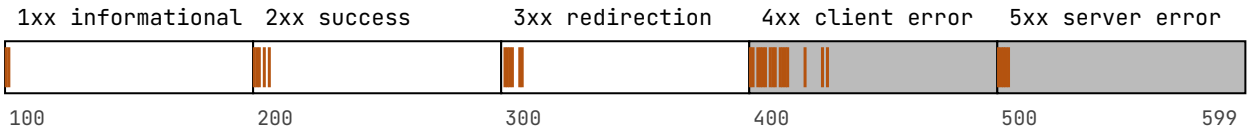
$ cd reference / HTTP status codes

HTTP status codes and when you meet them

Not every registered code, only those that show up in browsers, logs and API clients. For each one: what usually causes it, and what the client should do next.

[01] The five classes



Each tick is one code listed below. 4xx means the request should change; 5xx means retrying later may help.

Clients that do not know a code must treat it as the x00 code of its class, so an unknown 4xx is handled as 400 and an unknown 5xx as 500.

[02] 1xx: informational

CODE	NAME	WHEN YOU MEET IT
100	Continue	A client sent `Expect: 100-continue` before a large upload, and the other side agrees to receive the body. curl sends this header on its own for larger uploads.
101	Switching Protocols	The response to a WebSocket upgrade request over HTTP/1.1. Over HTTP/2 a WebSocket opens with an extended CONNECT answered by 200 (RFC 8441).

[03] 2xx: success

CODE	NAME	WHEN YOU MEET IT
200	OK	The normal success response with a body.
201	Created	A POST or PUT created a resource. A `Location` header should point to it.
202	Accepted	The request was queued for later processing; nothing has happened yet.
204	No Content	Success with no body, typical for DELETE and for PUT that returns nothing. A 204 response cannot carry content (RFC 9110).
206	Partial Content	The answer to a `Range` request: resumed downloads, video seeking.

[04] 3xx: redirection

CODE	NAME	WHEN YOU MEET IT
301	Moved Permanently	The URL changed for good. Browsers cache it hard, so test a new 301 carefully. Clients may turn a POST into a GET.
302	Found	A temporary redirect. Like 301, clients usually switch POST to GET.
303	See Other	After a form POST, send the browser to a result page with GET. This is the redirect to use after a POST.
304	Not Modified	The cached copy is still valid after a conditional request with `If-None-Match` or `If-Modified-Since`. No body.
307	Temporary Redirect	Like 302, but the method and body must be kept. Chrome's developer tools also show the browser's own HSTS upgrade from http to https as a 307 "Internal Redirect"; that one never came from the server.
308	Permanent Redirect	Like 301, but the method and body must be kept.

Choosing a redirect: permanent and safe for any method, 308. Temporary and method-preserving, 307. After a form submission, 303.

[05] 4xx: client errors

CODE	NAME	WHEN YOU MEET IT
400	Bad Request	The request is malformed: broken JSON, a missing required field, a header that cannot be parsed.
401	Unauthorized	Means unauthenticated: no credentials, or expired ones. Must include `WWW-Authenticate`.
403	Forbidden	Credentials are known but not sufficient, or access is refused for another reason, such as file permissions or an access rule.
404	Not Found	Nothing at this URL. Also used deliberately instead of 403 to avoid confirming that a resource exists.
405	Method Not Allowed	The URL exists but not for this method, for example POST to a static file. Must include `Allow`, listing the methods that do work.
406	Not Acceptable	No representation matches the `Accept` headers. Rare; most sites ignore Accept and return what they have.
408	Request Timeout	The client opened a connection but did not finish sending a request in time. Browsers that open connections in advance and leave them unused can cause it.

CODE	NAME	WHEN YOU MEET IT
409	Conflict	The request conflicts with current state: a duplicate unique key, an edit to an outdated version.
410	Gone	Removed on purpose and not coming back. It states the intent more clearly than 404, but Google documents that its crawler treats every 4xx code except 429 the same way.
412	Precondition Failed	An `If-Match` check failed: someone else changed the resource first. This is the basis of optimistic locking over HTTP.
413	Content Too Large	The body exceeds a size limit. An upload limit configured in front of the application is the usual cause.
414	URI Too Long	Usually a GET with a huge query string that should have been a POST body.
415	Unsupported Media Type	Wrong or missing `Content-Type`, for example JSON sent as `text/plain`.
422	Unprocessable Content	Syntactically valid but semantically wrong input; many APIs use it for validation errors.
429	Too Many Requests	Rate limited. If `Retry-After` is present, it gives a number of seconds or a date to wait for; respect it, and without it back off exponentially.
431	Request Header Fields Too Large	Usually an oversized cookie or a very long `Authorization` header. Clearing cookies for the site fixes it for users.

401 versus 403: if logging in (again) could help, it is 401. If the user is known and still not allowed, it is 403.

[06] 5xx: server errors

CODE	NAME	WHEN YOU MEET IT
500	Internal Server Error	An unhandled exception in the application. The response says nothing useful; the application log does.
501	Not Implemented	The method is not supported at all. Rare in practice.
502	Bad Gateway	A gateway or load balancer in front of the application got no valid response from it: the application crashed, is restarting, or closed the connection.
503	Service Unavailable	Overloaded or in maintenance; temporary by definition. `Retry-After` may say when to try again.
504	Gateway Timeout	The gateway waited for the application longer than its timeout. The request may still be running and may still complete.

A 502 or 504 is produced by whatever sits in front of the application, so the useful log is often there rather than in the application.

[07] Checking a status quickly

TASK	COMMAND
Status line and headers only	`curl -sI "$URL"`
Just the code	`curl -s -o /dev/null -w "%{http_code}\n" "$URL"`
Follow redirects and show each hop	`curl -sIL "$URL" | grep -iE "^(HTTP|location)"`

[08] Sources

[1] RFC 9110, section 15: status codes

[2] RFC 6585: 429 and 431

[3] RFC 8441: WebSockets over HTTP/2

[4] IANA HTTP status code registry

[5] Google Search Central: how its crawler handles status codes