jq Cheatsheet

By Dejan Panovski • Updated on Nov 6, 2026

jq filter reference for selecting fields, filtering arrays with select, building objects, CSV and TSV output, shell variables, and editing JSON files.

The `jq` command reads JSON, runs a filter on it, and prints the result. Wrap every filter in single quotes so the shell passes it through unchanged. The examples were tested with jq 1.8.1 on Ubuntu 26.04. Ubuntu 24.04 and Debian 13 ship jq 1.7.1, so entries marked jq 1.8+ need a newer release there.

Basic Syntax

jq reads the files you name, or standard input when there are none.

`jq '.' file.json`	Pretty-print JSON (colored in a terminal)
`command | jq '.'`	Pretty-print the JSON output of another command
`jq '.name' file.json`	Print one field
`jq -r '.name' file.json`	Print a string without quotes
`jq -c '.' file.json`	Print compact JSON on one line
`jq 'f1 | f2' file.json`	Pass each result of `f1` into `f2`
`jq 'f1, f2' file.json`	Print the results of both filters
`jq '.name' a.json b.json`	Run the filter on each file in turn
`jq -n 'FILTER'`	Run the filter without reading input
`jq empty file.json`	Check that a file is valid JSON (silent on success)

Selecting Fields

A missing key returns `null` instead of an error.

`.name`	Value of a top-level key
`.a.b.c`	Nested value
`."my-key"`	Key with a dash, a space, or a leading digit
`.["my-key"]`	Same, with bracket syntax
`.name, .version`	Two fields as separate outputs
`.key?`	Skip the error when the input is not an object
`getpath(["a", "b", "c"])`	Value at a path given as an array
`.. | objects | select(has("id")) | .id`	Every `id` value at any depth
`keys`	Sorted array of an object's keys
`has("email")`	`true` when the object has the key

Arrays

Indexes start at 0, and negative indexes count from the end.

`.[0]`	First element
`.[-1]`	Last element
`.[2:4]`	Elements 2 and 3 (the end index is excluded)
`.[:2]`	First two elements
`.[-2:]`	Last two elements
`.[]`	Every element, one at a time
`.[]?`	Iterate without an error on non-array input
`.items[].name`	One field from every element
`length`	Number of elements (keys for an object, characters for a string)
`first, last`	First or last element
`[limit(3; .[])]`	First three results of a stream
`flatten`	Flatten nested arrays
`reverse`	Reverse the order
`index("x")`	Position of a value, or `null` when it is absent
`[range(5)]`	Build `[0,1,2,3,4]`

Filtering with select

`select(COND)` passes its input through when the condition produces any value except `false` or `null`.

`.[] | select(.age > 30)`	Elements where a number is greater than 30
`.[] | select(.role == "admin")`	Elements with an exact string value
`.[] | select(.role != "admin")`	Elements without that value
`select(.a == 1 and .b == 2)`	Both conditions must match
`select(.a == 1 or .b == 2)`	Either condition matches
`select(.active | not)`	Negate a condition
`select(has("email"))`	Objects that have the key
`select(.email != null)`	Objects where the key is present and not `null`
`select(.role | IN("admin", "owner"))`	Value is one of several
`select(.name | test("^a"))`	Regex match
`select(.name | test("alice"; "i"))`	Case-insensitive regex match
`select(.name | startswith("a"))`	Prefix match (also `endswith`)
`select(any(.tags[]; . == "dev"))`	Array field contains an exact value
`map(select(.active))`	Keep the matches as an array
`first(.[] | select(.active))`	First match only
`any(.active), all(.active)`	`true` when any or all elements match
`.[] | numbers`	Only numbers (also `strings`, `objects`, `arrays`, `booleans`, `nulls`)

Building Objects and Arrays

Braces build a new object and square brackets collect results into an array.

`{user: .name, id: .id}`	New object from chosen fields
`{id, name}`	Shorthand for `{id: .id, name: .name}`
`{(.name): .age}`	Object with a key taken from the data
`[.[] | .name]`	Collect results into an array
`map(.name)`	Same, applied to an array
`map({id, name})`	Keep only some keys in every element
`pick(.name, .a.b)`	Keep chosen paths with their nesting (jq 1.7+)
`. + {status: "ok"}`	Add or replace a key
`to_entries`	`{"a":1}` becomes `[{"key":"a","value":1}]`
`from_entries`	Turn key and value pairs back into an object
`with_entries(.key |= ascii_upcase)`	Change every key
`with_entries(select(.value != null))`	Drop keys with `null` values
`map({(.name): .id}) | add`	Array of objects to one lookup object
`INDEX(.name)`	Object of elements keyed by a field

Strings and Dates

Regex functions use Perl-style syntax; double the backslash inside a jq string (`"\\s"`).

`"\(.name) is \(.age)"`	String interpolation
`.first + " " + .last`	Join strings with +
`ascii_downcase, ascii_upcase`	Change case
`split(",")`	String to array
`join(", ")`	Array to string (`null` becomes an empty string)
`ltrimstr("v"), rtrimstr(".log")`	Remove a prefix or suffix when present
`trim, ltrim, rtrim`	Remove surrounding whitespace (jq 1.8+)
`sub("-"; "_")`	Replace the first regex match
`gsub("\\s+"; " ")`	Replace every regex match
`test("^v[0-9]")`	`true` when the regex matches
`capture("(?<user>[^@]+)@(?<host>.+)")`	Named regex groups to an object
`[scan("[0-9]+")]`	Every regex match as an array
`tostring, tonumber`	Convert between strings and numbers
`todate`	Unix time to an ISO 8601 string
`fromdate`	ISO 8601 string (`2023-11-14T22:13:20Z`) to Unix time
`strftime("%Y-%m-%d")`	Format Unix time

Output Formats

Add `-r` so the formatted result prints as plain text.

`@csv`	Array to a CSV row (strings quoted)
`@tsv`	Array to a tab-separated row
`.[] | [.id, .name] | @csv`	One CSV row per element
`@base64, @base64d`	Encode or decode Base64
`@uri`	Percent-encode for a URL
`@sh`	Quote for safe use in a shell command
`@sh "echo \(.name)"`	Apply the format to each interpolated value
`@html`	Escape HTML special characters
`@json, tojson`	Value as a JSON string
`fromjson`	Parse a JSON string into a value

Shell Variables

Pass shell values with options instead of building the filter inside double quotes.

`jq --arg r "$role" '.[] | select(.role == $r)' f.json`	Filter by a shell variable
`--arg name "$value"`	Shell value as the string `$name`
`--argjson n 5`	JSON value (number, boolean, array, object) as `$n`
`--slurpfile ids ids.json`	Contents of a JSON file as an array in `$ids`
`--rawfile text notes.txt`	Contents of any file as one string in `$text`
`$ENV.HOME, env.HOME`	Read an environment variable
`jq -n '$ARGS.positional' --args a b`	Remaining arguments as an array of strings
`jq -n '$ARGS.positional' --jsonargs 1 true`	Remaining arguments parsed as JSON
`$ARGS.named`	Every `--arg` and `--argjson` value as an object
`.name as $n | ...`	Bind a value to a variable inside the filter
`name=$(jq -r '.name' f.json)`	Store a result in a shell variable
`while IFS= read -r n; do printf '%s\n' "$n"; done < <(jq -r '.[].name' f.json)`	Loop over results in Bash

Defaults and Errors

jq exits with 2 for a bad option, 3 for a filter syntax error, and 5 for a runtime or JSON parse error.

`.key // "default"`	Fallback when the value is `null`, `false`, or missing
`(.list // [])[]`	Iterate an array that may be missing
`.list[]?`	Iterate and suppress errors
`try .a catch "fallback"`	Replace an error with a value
`empty`	Produce no output
`error("message")`	Stop with a custom error
`type`	`"object"`, `"array"`, `"string"`, `"number"`, `"boolean"`, or `"null"`
`jq -e '.key' f.json`	Exit 1 when the last output is `false` or `null`
`jq -e 'has("key")' f.json`	Test for a key in a shell `if`
`jq -e '.[] | select(.x)' f.json`	Exit 4 when the filter produces no output

Sorting and Grouping

These functions take an array and return a new one or a single value.

`sort`	Sort an array of values
`sort_by(.age)`	Sort objects by a field
`sort_by(-.age)`	Descending order for numbers
`sort_by(.age) | reverse`	Descending order for any type
`sort_by(.role, .age)`	Sort by a second field when the first ties
`unique`	Sorted array without duplicates
`unique_by(.role)`	One element for each distinct value
`group_by(.role)`	Arrays of elements that share a value
`group_by(.role) | map({role: .[0].role, count: length})`	Count per group
`min_by(.age), max_by(.age)`	Element with the smallest or largest value
`min, max`	Smallest or largest value in an array
`map(.size) | add`	Sum of a field
`add / length`	Average of an array of numbers
`add(.[].size)`	Sum without building an array first (jq 1.8+)
`[.[] | select(.active)] | length`	Count matching elements
`map(.tags) | add | unique`	Merge arrays and remove duplicates
`reduce .[] as $u ({}; .[$u.role] += 1)`	Count elements by value

Modifying JSON

jq prints a modified copy and never changes the input file.

`.version = "2.0"`	Set a value (the key is created when missing)
`.a.b.c = 1`	Set a nested value, creating parent objects
`.count += 1`	Update with arithmetic (also `-=`, `*=`, `/=`)
`.name |= ascii_upcase`	Update a value with a filter
`.list += ["new"]`	Append to an array
`map(.age += 1)`	Update a field in every element
`(.[] | select(.name == "bob") | .active) = true`	Update only matching elements
`del(.password)`	Remove a key
`del(.a, .b)`	Remove several keys
`del(.[] | select(.active == false))`	Remove matching elements
`. * {"a": {"b": 9}}`	Deep merge (+ replaces nested objects instead)
`walk(if type == "string" then ascii_downcase else . end)`	Change every value at any depth
`jq '.v = 2' f.json > tmp.json && mv tmp.json f.json`	Save the result back to the file

Common Options

Short options can be combined, as in `-rc` or `-Rs`.

`-r`	Print strings without quotes
`-j`	Like `-r`, with no newline after each result
`-c`	One compact line per result
`-n`	Do not read input; start from `null`
`-s`	Read all inputs into one array
`-R`	Read each input line as a string
`-Rs`	Read the whole input as one string
`-e`	Set the exit status from the last output
`-S`	Sort object keys
`--tab`	Indent with tabs
`--indent 4`	Indent with 4 spaces (up to 7; the default is 2)
`-C, -M`	Force or disable color
`-a`	Escape non-ASCII characters
`-f file.jq`	Read the filter from a file
`--raw-output0`	Raw output with a NUL byte after each result (jq 1.7+)

Recipes

Common pipelines for API responses, logs, and configuration files.

`curl -fsS URL | jq -r '.name'`	Read a field from an API response
`jq -c '.[]' list.json > list.ndjson`	Array to JSON Lines (one object per line)
`jq -s '.' list.ndjson`	JSON Lines back to one array
`jq -c 'select(.level == "error")' app.log`	Filter JSON log lines
`jq -r '.[] | [.id, .name] | @tsv' f.json | column -t`	Aligned table
`jq -r '(["ID","NAME"] | @tsv), (.[] | [.id, .name] | @tsv)' f.json`	TSV with a header row
`jq -s '.[0] * .[1]' base.json override.json`	Merge two configuration files
`jq -r 'to_entries[] | "\(.key)=\(.value)"' f.json`	Object to `KEY=value` lines
`jq -r 'paths(scalars | true) | map(tostring) | join(".")' f.json`	List every scalar leaf path, including `false` and `null`
`diff <(jq -S . a.json) <(jq -S . b.json)`	Compare two JSON files, ignoring key order
`ip -j addr | jq -r '.[].ifname'`	Network interface names
`jq --raw-output0 '.[].path' f.json | xargs -0 ls -l --`	Pass values to `xargs` safely (jq 1.7+)

Related Guides

Use these guides for full JSON and API workflows.

jq Command	Full `jq` guide with examples
curl REST API Requests	Send API requests and read the responses
curl Cheatsheet	Request options for API calls
GitHub CLI	Filter `gh api` output with `--jq`
Parsing JSON in Python	Handle JSON in a script
awk Cheatsheet	Field processing for plain text

Troubleshooting

Quick checks for common jq errors and surprises.

`Cannot index array with string "name"`	The input is an array; iterate first with `.[].name`
`Cannot index string with string "key"`	The input is a string, not an object; check the structure with `jq .`
`Cannot iterate over null`	The key is missing or `null`; use `(.key // [])[]` or `.key[]?`
`key/0 is not defined for .my-key`	Quote keys that contain a dash: `."my-key"`
`admin/0 is not defined`	String values need double quotes: `select(.role == "admin")`
`$role is not defined`	Pass the value with `--arg role "$role"`
`parse error: Invalid numeric literal`	The input is not JSON (an HTML error page, YAML, or plain text)
`parse error: Unfinished JSON term`	The input is truncated or missing a closing bracket
`Filter breaks inside double quotes`	The shell expands `$var` first; use single quotes and `--arg`
`contains(["dev"]) also matches "devrel"`	Strings match as substrings; use `any(.tags[]; . == "dev")`
`trim/0 is not defined`	jq is older than 1.8; use `gsub("^\\s+|\\s+$"; "")`
`Strings print with quotes`	Add `-r`
`File is empty after jq ... f.json > f.json`	The shell truncates it first; write to a temp file, then `mv` it