jq JSON processor

CONCEPTS

Everything is a filter: it takes one input and emits zero or more outputs. The pipe | feeds each output of the left side into the right side. Quote the program in single quotes.

Input JSON file or stdin
↓ parse
Filter .a | .b[] | f
↓ 0..n values
Output pretty by default

Each stage runs once per value it receives

COMMAND-LINE FLAGS

`jq -r '<filter>'`
Print strings without quotes

`jq -c '<filter>'`
Compact, one value per line

`jq -n '<filter>'`
Use null as input, read nothing

`jq -s '<filter>' <file>`
Slurp all inputs into one array

`jq -R '<filter>' <file>`
Read each line as a string

`jq -e '<filter>'`
Exit 1 if last output is false/null, 4 if none

`jq --arg <name> <value> '$<name>'`
Pass a string variable

`jq --argjson <name> '<json>' '$<name>'`
Pass a JSON variable

SELECTING

`.`
The whole input

`.<key>`
Object field

`.<key>.<sub>`
Nested field

`.[<n>]`
Array element, negative counts from end

`.[<a>:<b>]`
Array or string slice

`.[]`
Every element or value

SELECTING (cont.)

`.<key>?`
Field, suppress errors

`.["<key>"]`
Field with odd characters

`..`
Recurse into every value

`.a, .b`
Several outputs from one input

FILTERING AND LOGIC

`select(<cond>)`
Keep inputs where cond is true

`map(<f>)`
Apply f to each array element

`map_values(<f>)`
Apply f to each object value

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

`and / or / not`
Combine conditions

`<a> // <default>`
Use default if a is null or false

`if <cond> then <a> else <b> end`
Conditional

`test("<regex>")`
Regex match, returns boolean

`has("<key>")`
Object contains key

`values / strings / numbers / objects / arrays`
Keep only values of that type, e.g. .. | strings

`any(<f>) / all(<f>)`
Does some or every element match

`<f> as $<x> | <expr>`
Bind each output of f to a variable

CONSTRUCTING

`{<a>, <b>: .<c>}`
Build an object, shorthand for {a: .a}

`[<f>]`
Collect all outputs into an array

`{(.<key>): .<v>}`
Computed key

CONSTRUCTING (cont.)

`"<text> \(.<key>)"`
String interpolation

`.<key> = <v>`
Set a field

`.<key> |= <f>`
Update a field with a filter

`del(.<key>)`
Remove a field or element

`to_entries / from_entries / with_entries(<f>)`
Object to key-value pairs and back

ARRAYS AND AGGREGATION

`length`
Size of array, object, string

`keys`
Sorted object keys or array indexes

`sort / sort_by(.<key>)`
Sort array

`group_by(.<key>)`
Group into sub-arrays

`unique / unique_by(.<key>)`
Remove duplicates

`min_by(.<key>) / max_by(.<key>)`
Smallest or largest element

`add`
Sum numbers or concatenate

`first / last / nth(<n>)`
Pick by position

`flatten`
Flatten nested arrays

STRINGS AND FORMATS

`tostring / tonumber`
Convert type

`type`
Name of the value type

`split("<sep>") / join("<sep>")`
Split and join strings

`@csv / @tsv`
Array to CSV or TSV row (use with -r)