jq JSON processor

START HERE

`jq . <file>`
Pretty-print JSON

`jq '.<key>'`
Pick a field

`jq '.[]'`
Iterate an array

`jq '.[] | select(.<key> == <v>)'`
Filter items

`jq -r '.<key>'`
Raw string, no quotes

`jq -c .`
One line per value

`jq '{<a>, <b>: .<c>}'`
Build an object

`jq -n '<expr>'`
Run without input

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 -S . <file>`
Sort object keys

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

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

`jq -f <script.jq> <file>`
Load the filter from a file

`jq -j '<filter>'`
Raw output with no newline after each value

`jq --raw-output0 '<filter>'`
NUL-separated raw output
Pairs with xargs -0.

`jq --rawfile <name> <file> '$<name>'`
Read a file into a string variable

`jq -n '$ARGS.positional' --jsonargs <json>...`
Pass positional JSON values

`jq --tab . / jq --indent <n> .`
Change indentation

`jq --slurpfile <name> <file> '$<name>'`
Load a file as an array variable

`jq --args '<filter>' <a> <b>`
Positional strings in $ARGS.positional

COMMAND-LINE FLAGS (cont.)

`jq -C . <file> | less -R`
Keep colours in a pager

`jq --stream . <file>`
Emit [path,leaf] events for huge files

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

`.<key>?`
Field, suppress errors

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

`..`
Recurse into every value

`paths / paths(<cond>)`
List the path to every value

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

`getpath(["a",0])`
Value at a path array

`pick(.a, .b.c)`
Keep only the given paths

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
else is optional; without it the input passes through.

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

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

FILTERING AND LOGIC (cont.)

`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
The variable is visible to the right of the pipe, e.g. .a[] as $x | $x.

`try <f> catch <g>`
Handle errors instead of aborting

CONSTRUCTING

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

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

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

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

`.a + .b`
Add numbers, join strings, arrays, objects

`$ENV.<NAME> / env.<NAME>`
Environment variable

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

`.<key> ⊨ <f>`
Update a field with a filter

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

`del(.[] | select(<cond>))`
Remove matching elements

`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

ARRAYS AND AGGREGATION (cont.)

`add`
Sum numbers or concatenate

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

`limit(<n>; <f>)`
Take first n outputs

`flatten`
Flatten nested arrays

`reduce <src> as $x (<init>; <update>)`
Fold into one value

STRINGS AND FORMATS

`tostring / tonumber`
Convert type

`type`
Name of the value type

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

`ltrimstr("<s>") / rtrimstr("<s>")`
Strip a prefix or suffix

`startswith("<s>") / endswith("<s>")`
Prefix or suffix test

`sub("<re>"; "<new>")`
Replace first regex match

`gsub("<re>"; "<new>")`
Replace all matches

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

`tojson / fromjson`
Encode to or parse a JSON string

`@sh`
Quote for shell

`@base64 / @base64d`
Encode or decode base64

RECIPES

JSON array to CSV (headers from first object)

```
jq -r '(.[0] | keys_unsorted) as $k
  | $k, (.[] | [.[$k[]]]) | @csv' <file>
```

Count items per group

```
jq 'group_by(.<key>)
  | map({<key>: .[0].<key>, n: length})' <file>
```

Edit a file in place

```
jq '.<key> = "<new>"' <file> > tmp.json \
  && mv tmp.json <file>
```

Feed values to a shell loop

```
jq -r '.[].<key>' <file> |
while read -r n; do
  echo "$n"
done
```