./jq

{ JSON: Like a Boss }

Jan 31
11:30am - 1pm
West Tower Rm. 5-20

By Bob Tiernay

HELLO!

I’m Bob

A web developer who frequently navigates *very* large and deeply nested JSON documents.

I also ♥ jq :)

Nested!

MOTIVATION

JSON

You just queried a new REST API....

... a wall of text descends upon you ...

You ask: "How can I get this?!?!"

← ...Hundreds of lines this way

```
.....00001000448","projectCodes":[],"files":[],"metadata":{"d
atasetId":"EGAD00001000448","mappings":{"Analysis_Sample_meta
_info":[],"Sample_File":[{"SAMPLE_ALIAS":"BN03T","SAMPLE_ACCE
SSION":"EGAN00001085931","FILE_NAME":"111201_SN545_0167_AC05P
LACXX/BN03T_CGATGT_L003_R1_001.fastq.bz2.gpg","FILE_ACCESSION
":"EGAF00000193585"}sapiens","COMMON_NAME":"human"},"broker_n
ame":"EGA","alias":"BN03T","IDENTIFIERS":{"PRIMARY_ID":"ERS18
4758","SUBMITTER_ID":{"namespace":"NCCRI","content":"BN03T"}}
,"TITLE":"Hep,"accession":"ERS184758","SAMPLE_ATTRIBUTES":{"S
AMPLE_ATTRIBUTE":[{"TAG":"gender","VALUE":"male"},{"TAG":"ENA
-SUBMISSION-TOOL","VALUE":"SRA-Webin"}]}}}}},"experiments":{"
EGAX00001103159":{"EXPERIMENT_SET":{"EXPERIMENT":{"PLATFORM":
{"ILLUMINA":{"INSTRUMENT_MODEL":"Illumina HiSeq
2000"}},"DESIGN":{"DESIGN_DESCRIPTION":{},"SPOT_DESCRIPTOR":{
"SPOT_DECODE_SPEC":{"READ":[{"READ_CLASS":"Application
0f0e2da588f55845c0d9d78d331"}]}},"broker_name":"EGA","alias":
"ena-RUN-NCCRI-29-05-2013-08:52:43:023-74","RUN_ATTRIBUTES":{
"RUN_ATTRIBUTE":{"TAG":"ENA-SUBMISSION-TOOL","VALUE":"SRA-Web
in"}},"IDENTIFIERS":{"PRIMARY_ID":"ERR280119","SUBMITTER_ID":
{"namespace":"NCCRI","content":"ena-RUN-NCCRI-29.....
```

Thousands of lines that way... →

Say hello to my little friend:

./jq

Hi!

THE FAMILY

- jq → {JSON}
- xmlstarlet → <XML/>
- pup → <HTML>
- yq → YAML:

Siblings

Cousins: sed, awk, grep

“ jq is a lightweight and flexible command-line **JSON processor**

— Stephen Dolan

Much more powerful than Json Path!

- Written in C
- No runtime dependencies
- **Functional** programming language
- [Turing complete](#) ← Brainf*ck proof!
- Terse and expressive
- Slice, filter, map and transform data with ease

SOME

USECASES

- Exploring JSON APIs
  - Elasticsearch
  - GitHub
  - Docker

- Exploring JSONL dumps
- Lightweight integration
- One-off mini-ETLs
- Analytics / aggregations

* Also, just a darn handy JSON formatter!

Links:

https://stedolan.github.io/jq/

- Homepage
- Installation instructions
- Documentation and resources

https://jqplay.org/

- Interactive **jq** playground
- Great for learning
- Share snippets

jq tag on Stackoverflow

#jq channel on Freenode

PLATFORMS

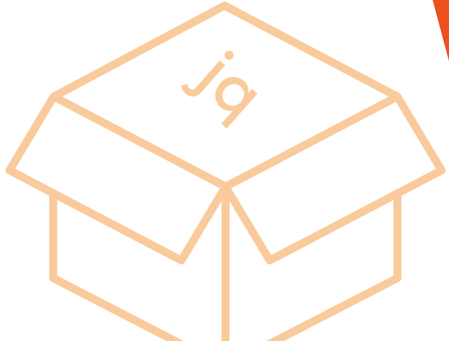
- Linux
- OSX
- FreeBSD
- Solaris
- Windows

Mac

```
brew install jq
```

Linux

```
apt-get install jq
```



jq INVOCATION₁

`stdin ➜ jq <filter> ➜ stdout` `jq <filter> file ➜ stdout`

Most common

Inline Input

- Read from stdin
- Write to stdout

Try httpie instead!

```
curl -s … | jq …
```

```
echo … | jq …
```

```
cat … | jq …
```

File Input

- Read from file
- Write to stdout

```
jq … file.json
```

> jq --help

```
jq - commandline JSON processor [version 1.5]
Usage: jq [options] <jq filter> [file...]

    jq is a tool for processing JSON inputs [...]

    Some of the options include:
```

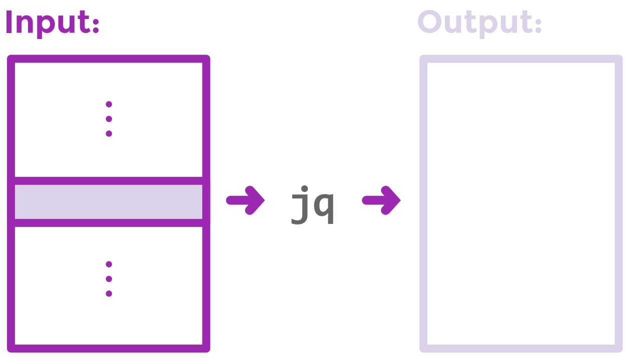
-c	compact instead of pretty-printed output;
-n	use `null` as the single input value;
-e	set the exit status code based on the output;
-s	read (slurp) all inputs into an array; apply
-r	output raw strings, not JSON texts;
-R	read raw strings, not JSON texts;
-C	colorize JSON;
-M	monochrome (don't colorize JSON);
-S	sort keys of objects on output;
--tab	use tabs for indentation;
--arg a v	set variable $a to value <v>;
--argjson a v	set variable $a to JSON value <v>;
--slurpfile a f	set variable $a to an array of JSON texts read from <f>;

jq INVOCATION$_2$

Modes

Line by Line

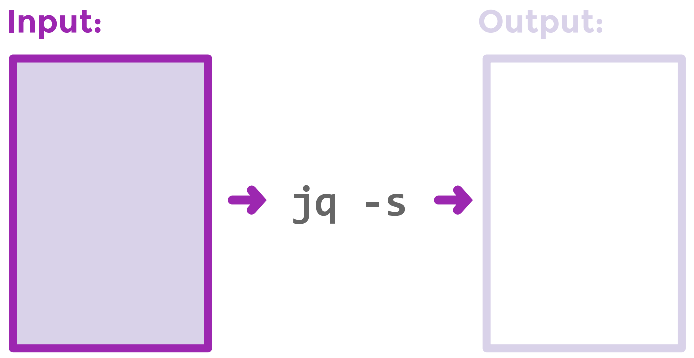
The default



Read line by line

"Slurp Mode"

```
--slurp / -s
```



Reads as a single array!

KEY IDEA

FILTERS:

```
jq <filter>
```

BASIC FILTERS$_1$

```
jq .
```

Identity filter

```
> echo '{"x":1,"y":2}' | jq .
```

```
{
    "x": 1,
    "y": 2
}
```

Very useful for pretty printing / colourizing input!

BASIC FILTERS$_2$

```
jq .property
```

Projection

```
> echo '{"x":1,"y":2}' | jq .x
```

```
1
```

Removes outer object

PRO-TIP

```
jq .property?
```

- Just like `.property` but does not output an error when “.” is not an array or an object.

BASIC FILTERS$_3$

```
jq .nested.property
```

Nested projection

```
> echo '{"x":{"y":"z"}}' | jq .x.y
```

```
"z"
```

← Removes both parents

BASIC FILTERS$_4$

```
jq .[]
```

Flatmap

```
> echo '[{"x":1},{"x":2}]' | jq .[]
```

```
{
    "x": 1
}
{
    "x": 2
}
```

Removes the outer array

KEY IDEA

OPERATORS:

```
jq f₁ <op> f₂
```

BASIC OPERATORS$_1$

```
jq filter₁ | filter₂
```

Pipe

```
> echo ‘{“x”:{“y”:“z”}}’ | jq ‘.x | .y’
```

```
“z”
```

Result of pipelining. Can be done many, many times!

BASIC OPERATORS$_2$

$\boxed{\texttt{jq filter}_1\ \texttt{,}\ \texttt{filter}_2}$

Tee

```
> echo '{"x":1}' | jq '. , .'
```

```
{
    "x": 1
}
{
    "x": 1
}
```

← One record becomes two

BASIC OPERATORS$_2$

jq (expressions)

Grouping

```
> echo '{"x":1}' | jq '. , (. | .x)'
```

Useful for complex sub-processing

```
{
    "x": 1
}
1
```

BASIC OPERATORS$_3$

```
> echo \
[2,4,6,8] |
```

```
jq . + 2 → [4, 6, 8, 10]
```
Addition

```
jq . - 2 → [0, 2, 4, 8]
```
Subtraction

```
jq . * 2 → [4, 8, 12, 16]
```
Multiplication

```
jq . / 2 → [1, 2, 3, 4]
```
Division

```
jq . % 2 → [0, 0, 0, 0]
```
Modulus

KEY IDEA

CONSTRUCTORS:

```
jq <...>
```

CONSRUCTORS, 1

```
jq [ ... ]
```

Array Constructor

```
> echo ‘{“x”:1}’ | jq ‘[.]’
```

```
[
    {
        “x”: 1
    }
]
```

← Puts each record into an array

CONSRUCTORS$_2$

```
jq {…}
```

Object Constructor

```
> echo ‘1’ | jq ‘{“x”:.}’
```

```
{
    “x”: 1
}
```

Creates a new object

VARIABLES

Ok, ok, "bindings" for the language theorists

$ VARIABLES

- Usually not needed
- Can help to cut down on noise or repetition
- Must start with a **$**
- Scoped over the expression that defines them

```
expression as $x
```

$$\text{expression as }\underbrace{\$x}_{\text{Variable definition}}$$

$ VARIABLES

- Can use multiple declarations via *"Destructuring"*

```
. as {a: $x, b: [$y, $z]}
```

Variable definitions

$f(x)$

FUNCTIONS

in, map(x), map_
del(path_expression), g
VALUE), to_entries, from_entrie
select(boolean_expression), arrays, obj
booleans, numbers, normals, finites, strings, n
values, scalars, empty, error(message), paths,
paths(node_filter), leaf_paths, add, any, any(co
any(generator; condition), all, all(condition),
all(generator; condition), flatten, flatten(dep
range(upto), range(from;upto) range(from;upto;b
sqrt, tonumber, tostring, type, infinite, nan,
isinfinite, isnan, isfinite, isnormal, sort,
sort_by(path_expression), group_by(path_expre
max, min_by(path_exp), max_by(path_exp), uni
unique_by(path_exp), reverse, contains(eleme
indices(s), index(s), rindex(s), inside, st
ndswith(str), combinations, combinations(
rtrimstr(str), explode, imp
ascii_downcase, as
(cond; next),
recurse_

FUNCTIONS: DEFINITIONS

- Unit of reuse and encapsulation
- Introduced with the **def** keyword
- Can take arguments
- . is an implicit arg!

Function definition:

Optional!

```
def name[(args)]: body;
```

Examples:

```
def increment: . + 1;
```

```
def map(f): [.[] | f];
```

FUNCTIONS: ARRAY BUILTINS

Input:

```
> echo \
[2,4,6,8] |
```

`jq length`	→	`4`
`jq indices(8)`	→	`[3]`
`jq contains([2])`	→	`true`
`jq reverse`	→	`[8,6,4,2]`
`jq min`	→	`2`
`jq max`	→	`8`

FUNCTIONS: STRING BUILTINS

Input:

```
> echo \
'"Hello!"' |
```

`jq split("l")`	→	`["He","","o"]`
`jq test("He.*")`	→	`true`
`jq length`	→	`6`
`jq contains("!")`	→	`true`
`jq startswith("!")`	→	`false`
`jq ascii_downcase`	→	`"hello!"`

FUNCTIONS: OBJECT BUILTINS

Input:

```
> echo     \
'{         \
  “a”:1,   \
  “b”:2,   \
  “c”:3    \
}'         |
```

jq keys	→ [“a”,“b”,“c”]
jq has(“a”)	→ true
jq del(“a”)	→ {"b":2,"c":3}
jq add	→ 6
jq to_entries	→ [{“key”:”a”, “Value”:”1}, ...]
jq flatten	→ [1,2,3]

FUNCTIONS: SELECT

```
select(boolean_expresion)
```

Input:

```
> echo          \
'{"x":1}'       \
'{"x":2}'       \
'{"x":3}'       \
'{"x":4}'       |
```

Expression:

```
jq select(. > 2)
```

Only pass through values that match the boolean expression

Output:

```
{"x": 3}
{"x": 4}
```

FUNCTIONS: PATHS

```
paths(node_filter)
```

Input:

```
> echo \ '{
    "a":{
        "b":{
            "c":1,
            "d":2
        }
    }
}' |
```

Expression:

```
jq paths(scalars)
```

Only give leaf paths

Output:

```
[“a”]
[“a”,“b”]
[“a”,“b”,“c”]
[“a”,“b”,“d”]
```

FUNCTIONS: RECURSE

```
recurse(f;condition)
```

Input:

```
> echo \ '{
    "a":{
        "b":{
            "c":1,
            "d":2
        }
    }
}' |
```

Expression:

```
jq recurse
```

- Or -

```
jq ..
```

Recursively emit all sub-values

Output:

```
{"a":{"b":{"c":1,"d":2}}}
{"b":{"c":1,"d":2}}
{"c":1,"d":2}
1
2
```

FUNCTIONS: GROUP_BY

```
group_by(path_expression)
```

Input:

```
> echo \ '[
    {"x":1, "y":1},
    {"x":1, "y":2},
    {"x":2, "y":1},
    {"x":2, "y":2}
]' |
```

Expression:

```
jq group_by(.x)
```

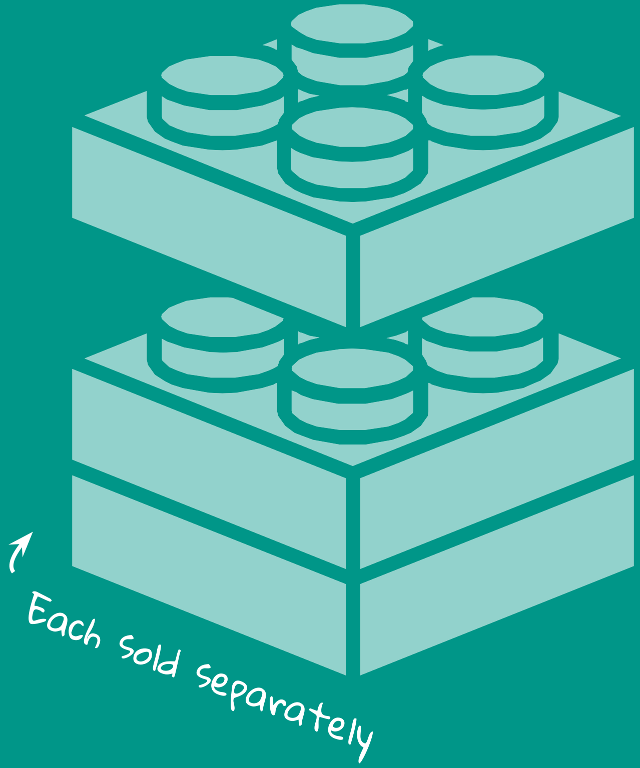
Groups by path expression.

Requires array as input.

Output:

```
[
    [
        {"x":1,"y":1},
        {"x":1,"y":2}],
    [
        {"x":2,"y":1},
        {"x":2,"y":2}
    ]
]
```

MODULES



MODULES

- Break larger scripts into files
- Reuse functions across sessions, modules, etc.
- Searches `"~/.jq"`, `"$ORIGIN/../lib"`, ... for `*.jq` files

```
import MyModule as MY_MODULE;
```

Relative path string. Could be at ~/.jq/MyModule.jq

```
...
```

Imported prefix

```
. | MY_MODULE::my_function
```

ESCAPE

a.k.a String Sauce

```
… | jq @name
```

Apply formatter

`@text`	Just calls tostring
`@json`	Serializes input as JSON
`@html`	Applies HTML/XML escaping
`@uri`	Applies percent encoding
`@csv`	Rendered as CSV with double quotes
`@tsv`	Rendered as TSV (tab-separated values)
`@sh`	Escaped suitable for use in a POSIX shell
`@base64`	Converted to base64 as specified by RFC 4648

ESCAPE

Example: Format Output as CSV

Input: Expression:

```
> echo [1,2,3,4] | jq @csv
```

Output in CSV format:

```
"1","2","3","4"
```

@csv

OTHER

STUFF (that I don't have time for)

- Dates
- Control flow
- Assignment
- Generators
- Parsers
- Streaming
- I/O

```
if, while,
--stream,
input,
|=,
etc.
```

LIKE A BOSS

Tips and tricks from the field

MISSION:

“Create a TCGA barcode-to-UUID mapping in TSV format.”

INPUT:

```
curl -s 'https://gdc-api.nci.nih.gov/legacy/cases?...
```

```
{"data": {"hits": [{"case_id": "eb7c3b35-7a5e-4621-b31f-9775c51f9a23", "samples":
[{"sample_id": "f57d3d51-3754-44b7-88f9-b5b1eaa534c5", "portions": [{"analytes":
[{"aliquots": [{"aliquot_id": "48be744a-1f5d-4e70-9bb2-c30a131d8679", "submitter_id":
"TCGA-61-2098-11A-01W-0721-10"}, {"aliquot_id": "5a568ebf-3329-4d21-be35-e7578c526c30",
"submitter_id": "TCGA-61-2098-11A-01W-0725-09"}, {"aliquot_id":
"7c7d78f2-df0f-4a03-9e42-cf83f20949ae", "submitter_id": "TCGA-61-2098-11A-01W-1217-08"},
{"aliquot_id": "65621d9f-8a77-4643-9b82-5b3d01f19ca6", "submitter_id":
"TCGA-61-2098-11A-01W-0723-08"}]}, {"aliquots": [{"aliquot_id":
"de14acff-b622-48c1-94c7-4105a3a6fa92", "submitter_id": "TCGA-61-2098-11A-01D-0664-04"},
{"aliquot_id": "5cb6df63-9901-483a-8a0b-7a67c49caab3", "submitter_id":
"TCGA-61-2098-11A-01D-0665-06"}, {"aliquot_id": "220f1a3b-49f1-4686-b2a9-0d6087262998",
"submitter_id": "TCGA-61-2098-01A-01D-0665-06"}, {"aliquot_id":
"89f4b51d-2c3d-4a66-8580-cb1edb8bb072", "submitter_id": "TCGA-61-2098-01A-01D-0663-01"},
{"aliquot_id": "8b8a5622-5638-4d0e-b034-64739cfc678b", "submitter_id":
"TCGA-61-2098-01A-01D-0667-05"}]}]}, {"analytes": []}], "submitter_id": "TCGA-61-2098-01A"}],
"submitter_id": "TCGA-61-2098"}], "pagination": {"count": 1, "sort": "", "from": 1, "page": 1,
"total": 17177, "pages": 17177, "size": 1}}, "warnings": {}}
```

FORMAT:

```
… | jq .
```

```
{
  "data": {
    "hits": [
      {
        "case_id": "eb7c3b35-7a5e-4621-b31f-9775c51f9a23",
        "samples": [
          {
            "sample_id": "f57d3d51-3754-44b7-88f9-b5b1eaa534c5",
            "portions": [
              {
                "analytes": [
                  {
                    "aliquots": [
                      {
                        "aliquot_id": "48be744a-1f5d-4e70-9bb2-c30a131d8679",
                        "submitter_id": "TCGA-61-2098-11A-01W-0721-10" ...
```

PRO-TIP

This JSON is too big for my console!

```
curl … | jq -C . | less -R
```

Colorize

Interpret colors

BOSS IDEA:

INSTANT SCHEMA

```
jq -r 'path(..) | map(tostring) | join("/")'
```

```
data
data/hits
data/hits/0
data/hits/0/case_id
data/hits/0/samples
data/hits/0/samples/0
data/hits/0/samples/0/sample_id
data/hits/0/samples/0/portions
data/hits/0/samples/0/portions/0
data/hits/0/samples/0/portions/0/analytes
data/hits/0/samples/0/portions/0/analytes/0 ...
```

TRANSFORM:

```
jq -r '.data.hits[] |
[.case_id, .submitter_id],
(.samples[]? | [.sample_id, .submitter_id]),
(.samples[]?.portions[]?.analytes[]?.aliquots[]
  | [.aliquot_id, .submitter_id]) | @tsv'
```

```
eb7c3b35-7a5e-4621-b31f-9775c51f9a23  TCGA-61-2098
f57d3d51-3754-44b7-88f9-b5b1eaa534c5  TCGA-61-2098-11A
60b275ef-91db-4572-a187-74fea2507bb8  TCGA-61-2098-01A
48be744a-1f5d-4e70-9bb2-c30a131d8679  TCGA-61-2098-11A-01W-0721-10
5a568ebf-3329-4d21-be35-e7578c526c30  TCGA-61-2098-11A-01W-0725-09
7c7d78f2-df0f-4a03-9e42-cf83f20949ae  TCGA-61-2098-11A-01W-1217-08
65621d9f-8a77-4643-9b82-5b3d01f19ca6  TCGA-61-2098-11A-01W-0723-08
de14acff-b622-48c1-94c7-4105a3a6fa92  TCGA-61-2098-11A-01D-0664-04
5cb6df63-9901-483a-8a0b-7a67c49caab3  TCGA-61-2098-11A-01D-0665-06 ...
```

PRO-TIP

```
echo 'def schema: path(..) | map(tostring) | join("/");' >> ~/.jq
```

```
curl … | jq schema
```

GOTCHA!

Just don't do this stuff!

GOTCHA: Shell Quoting

PROBLEM:

```
>jq ...|...|...|...
```

Shell interprets these as pipes!

SOLUTION:

```
>jq '...|...|...|...'
```

Add single quotes FTW!

Prefer single over double

GOTCHA: Property vs Call

PROBLEM:

```
>jq 'x.y'
```

Expects x to be function!

```
jq: error: x/0 is not defined
at <top-level>, line 1:
```

SOLUTION:

```
>jq '.x.y'
```

Add a dot in front to access property

GOTCHA: Property w/ ‘-’

PROBLEM:

```
>jq ‘.file-id’
```

Thinks '-' is subtraction!

```
jq: error: id/0 is not defined
at <top-level>, line 1:
```

SOLUTION:

```
>jq ‘.[“file-id”]’
```

Need to use array-style access

GOTCHA: **Recurse**

PROBLEM:

```
>jq '..a'
```

Won't work!

```
jq: error: syntax error, unexpected
IDENT, expecting $end (Unix shell
quoting issues?) at <top-level>,
line 1:
```

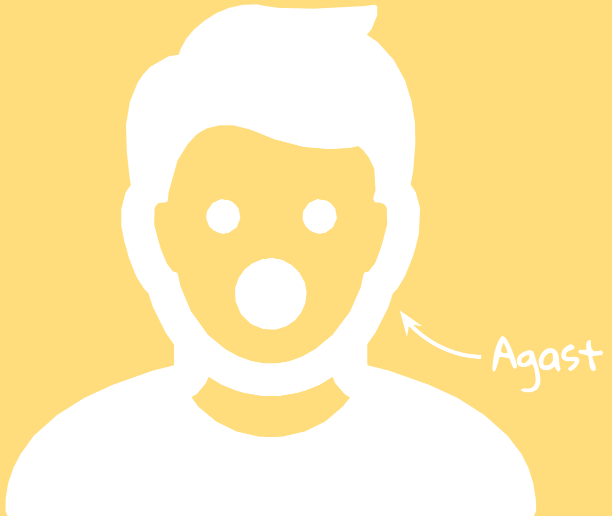
SOLUTION:

```
>jq '.. | a'
```

Add a pipe instead

BUT WAIT...

... THERE'S MORE!



BEYOND THE BOX

Java: jq-jackson

https://github.com/eiiches/jackson-jq

- 100% Java
- Embeddable
- Works with Jackson’s JsonNode
- Code functions in Java

```
val query = JsonQuery.compile(".name|repeat(3)");
val record = mapper.readTree("{\"name\":\"a\"}");
List<JsonNode> result = query.apply(scope, record);
```

Node: node-jq

https://github.com/sanack/node-jq

- npm module
- jq wrapper (not native js)
- Goal was to make jq syntax available in Atom with atom-jq.

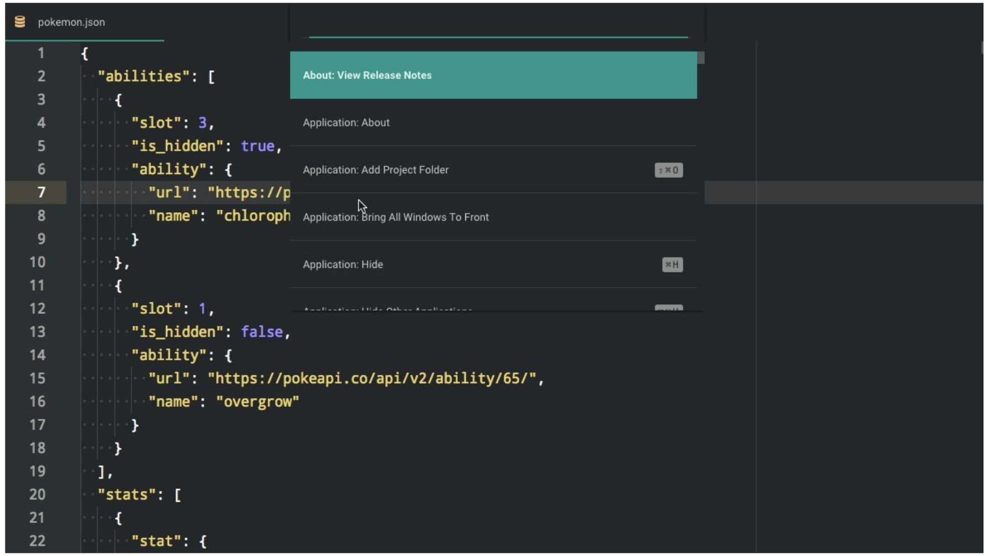
```
import { run } from 'node-jq'
const filter = '. | map(select(.foo > 10))'
const jsonPath = '/path/to/json'
const options = {}

run(filter, jsonPath, options).then(console.log)
```

Atom: atom-jq

https://github.com/sanack/atom-jq

- atom module
- Run jq inside Atom!


pokemon.json
About: View Release Notes
Application: About
Application: Add Project Folder
Application: Bring All Windows To Front
Application: Hide

```
{
  "abilities": [
    {
      "slot": 3,
      "is_hidden": true,
      "ability": {
        "url": "https://p
        "name": "chloroph
      }
    },
    {
      "slot": 1,
      "is_hidden": false,
      "ability": {
        "url": "https://pokeapi.co/api/v2/ability/65/",
        "name": "overgrow"
      }
    }
  ],
  "stats": [
    {
      "stat": {
```

OTHER

LANGUAGE

jq-go	Go
jq-shell	jq based shell
jr, jqr, rbq	Ruby
jq-r	R
lq	Lua

SNIPPETS

- jq-hopkok
- jq-cookbook

ALTERNATIVES

- jq.node jq like

TOOLS

jq-httpd	jq HTTP server
jq-parser	Scala jq parser
jqnpm	jq package manager
jqui	UI
show-struct	Print filters

THANKS!

Any questions?

CREDITS

Images by:

The Noun Project

JSON File
By Oliviu Stoian, RO

Family
By Abhiraami Thangavel, IN

Web API
By Ian Kirkland, US

Filter
By Alex Koev, BG

Calculator
By Creative Stall, PK

Carpenter
By Gan Khoon Lay

Modules
By mikicon, CZ

Presentation
By Creative Stall, PK

Explosion
By Aldric Rodríguez

Gift
By Chanut is Industries, TH

Surprised
By Gregor Črešnar