`jq` : The JSON Swiss Army Knife

by @colinfdrake

What is `jq`?

- A command line utility to filter, manipulate, and query JSON
- Follows the pipeline/filter model of `sed`
  - Use it with network utilities like `curl`, `httpie`, etc.
  - Or a cached response in a `.json` file
- Useful for exploration, inspection, and scripting alike

Concepts

- Operates with a composable and sequencable set of *filters*
  - Select the first node from this array
  - Select only nodes with the `foo` key equal to `3`
  - Remove `foo` and `bar` keys from the first three nodes
  - Map the `foo` key, assuming an integer, to `foo` plus four

Concepts

- Filters may be chained in a sequence: `.foo.bar`
- Or fed into one another with operators: `map(.foo)`
- Or piped: `.foo|map(.bar)`
- Provided via command line `jq '.foo|map(.bar)'`
    - *Don't forget quotes!*

Prerequisites

I really, really hate using cURL, so we'll use `httpie` instead.

- `brew install httpie jq`

Demo Data Set (IP Address Querying)

```
$ http api.ipify.org format==json
{
    "ip": "12.34.56.78"
}
```

Demo Data Set (Songsterr Song Search)

```
$ http --follow songsterr.com/a/ra/songs.json \
                pattern==Bowie
[
  {
    "id": 3649,
    "type": "Song",
    "title": "Rebel Rebel",
    "artist": {
      "id": 60,
      "type": "Artist",
      ...
      "name": "David Bowie"
    },
    "chordsPresent": true,
    "tabTypes": [
      ...
    ]
  },
  ...
]
```

Basic Filters

`.` : Passthrough Filter

A filter that returns the input, pretty-printed.

Passthrough Filter: Example

```
$ http <songsterr> | jq '.'
[
  {
    "id": 3649,
    "type": "Song",
    "title": "Rebel Rebel",
    "artist": {
      "id": 60,
      "type": "Artist",
      ...
      "name": "David Bowie"
    },
    "chordsPresent": true,
    "tabTypes": [
      ...
    ]
  },
  ...
]
```

`.<index>`, `.[<index>]`: Key indexing

A filter that returns the value of the `<index>` key in a JSON dictionary (or the `<index>` th element of an array, if using bracket notation).

`.<key>` : Key Selection

```
$ http <ipaddress> | jq '.'
{
    "ip": "12.34.56.78"
}

$ http <ipaddress> | jq '.ip'
"12.34.56.78"
```

`.<key>` : Key Selection

```
# [
#   {
#     "id": 3649,
#     "type": "Song",
#     "title": "Rebel Rebel",
#     "artist": {
#       ....
#       "name": "David Bowie"
#     },
#     "chordsPresent": true,
#     "tabTypes": [...]
#   }
#   ...
# ]

$ http <songsterr> | jq '.[0].artist.name'
"Rebel Rebel"
```

Object and Array Construction

JSON literal syntaxes (`{...}` and `[...]`) allow you to build arbitrary outputs from your inputs.

Object Construction

Simply write a JSON literal, dropping in filters where you want to build off of the original input.

```
$ http <ipaddress> | jq \
          '{my_ip: .ip, original: ., other: "foo"}'
{
  "my_ip": "70.113.86.150",
  "original": {
    "ip": "70.113.86.150"
  },
  "other": "foo"
}
```

Array Construction

```
$ http <songsterr> | jq '[.[0], .[5]]'
[
  {
    "id": 3649,
    "type": "Song",
    "title": "Rebel Rebel",
    ...
  },
  {
    "id": 29023,
    "type": "Song",
    "title": "China Girl",
    ...
  }
]
```

Built-in Operators

`jq` comes with mathematical operators, comparison operators, simple functions (`length` , `keys`), higher order filters (`map` , `select`), and even some meta-functions (`$__loc__`).

`map(<filter>)`

Applies a filter to each element in an array.

```
$ http <songsterr> | jq \
        'map({title: .title, chords: .chordsPresent})'
[
  {
    "title": "Rebel Rebel",
    "chords": true
  },
  {
    "title": "Ziggy Stardust",
    "chords": true
  },
  {
    "title": "Starman",
    "chords": true
  },
  ...
]
```

`select(<boolean_expression>)`

Returns the original input if `<boolean_expression>` is true, otherwise returns nothing.

```
$ http <songsterr> | jq 'select(.[0].title | length > 10)
[...original input...]

$ http <songsterr> | jq 'select(arrays)'
[...original input...]

$ http <songsterr> | jq 'select(objects)'
...nothing!...
```

Common Pattern: `map(select(...))`

```
$ http ... | jq  'map(select(.chordsPresent == false)) |
                  map({id, title})'
[
  {
    "title": "Wild Is The Wind",
    "id": 407353
  },
  {
    "title": "The Jean Genie",
    "id": 407347
  },
  {
    "title": "Modern Love",
    "id": 407356
  },
  ...
]
```

Note: `{title}` is short-hand for `{title: .title}`.

Other Helpful Built-ins

- `keys` : returns an array of keys in an object.
- `has(<key>)` : returns whether an object has a key.
- `del(<path_expression>)` : removes a key from an object.
- `any(<condition>)` : returns true if at least one element in an array matches a condtion. Also see `all(<condition>)` .
- `flatten` recursively flattens an array.
- `sort` .

Questions?

Here's some links:

- Try jq online!
- Tutorial covering the basics with the Github API.
- jq Manual, *extremely* helpful documentation with examples for every filter