Discover how to use jq, a JSON manipulation command line, with GeoJSON

webgeodatavore.com/jq-json-manipulation-command-line-with-geojson.html

Mostly, when we need to manipulate GeoJSON, we do it using Python or Node.js/io.js.

Sometimes, it can be interesting to manipulate GeoJSON from command line. We will see how along the blog post.

Do not hesitate to make us feedback about content.

The shorter description of jq is as below :

> jq is a lightweight and flexible command-line JSON processor.

If you are a command line addict, you will like the official description

> jq is like sed for JSON data – you can use it to slice and filter and map and transform structured data with the same ease that sed, awk, grep and friends let you play with text.

As an addict to geo data, we frequently need to use GeoJSON, a subset of JSON for geographic objects.

So, let's try using a GeoJSON file `countries.geojson`. Here, we will only play with `FeatureCollection`.

For all commands, we consider you are using a *Unix-like* system. If you are on Windows, you will have to deal on your own with characters escaping.

Let's start!

Get the data

```
wget https://raw.github.com/datasets/geo-boundaries-world-
110m/master/countries.geojson
```

Count GeoJSON features

```
jq '.features |length' countries.geojson
```

Return

```
177
```

Combine both result from two keys without filtering features

```
jq '{type: .type , features: .features}' countries.geojson
```

Return the same content as countries.geojson

Filter on countries

```
jq '{type: .type , features: [ .features[]| select( .properties.sovereignt ==
"France") ] }' countries.geojson
```

Return a GeoJSON string where only features with `sovereignt` property equal "France" are kept.

Get properties keys

```
jq '.features[0:1][0].properties |keys' countries.geojson
```

Return

```
[
  "abbrev",
  "abbrev_len",
  "adm0_a3",
  "adm0_a3_is",
  "adm0_a3_un",
  "adm0_a3_us",
  "adm0_a3_wb",
  "adm0_dif",
  "admin",
  "brk_a3",
  "brk_diff",
  "brk_group",
  "brk_name",
  "continent",
  "economy",
  "featurecla",
  "fips_10",
  "formal_en",
  "formal_fr",
  "gdp_md_est",
  "gdp_year",
  "geou_dif",
  "geounit",
  "gu_a3",
  "homepart",
  "income_grp",
  "iso_a2",
  "iso_a3",
  "iso_n3",
  "labelrank",
  "lastcensus",
  "level",
  "long_len",
  "mapcolor13",
  "mapcolor7",
  "mapcolor8",
  "mapcolor9",
  "name",
  "name_alt",
  "name_len",
  "name_long",
  "name_sort",
  "note_adm0",
  "note_brk",
  "pop_est",
  "pop_year",
  "postal",
  "region_un",
  "region_wb",
  "scalerank",
  "sov_a3",
```

```
  "sovereignt",
  "su_a3",
  "su_dif",
  "subregion",
  "subunit",
  "tiny",
  "type",
  "un_a3",
  "wb_a2",
  "wb_a3",
  "wikipedia",
  "woe_id"
]
```

When you may need to add CRS

```
jq --arg crs '{"type": "name", "properties": {"name":
"urn:ogc:def:crs:OGC:1.3:CRS84"}}'  '{type: .type , crs: $crs|fromjson, features: [
.features[]| select( .properties.sovereignt == "France") ] }' countries.geojson
```

Return same content as `countries.geojson` but with additional CRS

Get min value (if numeric)

```
jq '[.features[].properties.pop_est] | min' countries.geojson
```

Return

```
-99
```

In fact, -99 means "no data"

Get max value

```
jq '[.features[].properties.pop_est] | max' countries.geojson
```

Return

```
1338612970
```

Get unique values (return array)

```
jq '[.features[].properties.economy] | unique' countries.geojson
```

Return

```
[
  "1. Developed region: G7",
  "2. Developed region: nonG7",
  "3. Emerging region: BRIC",
  "4. Emerging region: MIKT",
  "5. Emerging region: G20",
  "6. Developing region",
  "7. Least developed region"
]
```

Get unique (return list, useful to pipe with other unix command)

```
jq '[.features[].properties.economy] | unique[]' countries.geojson
```

Return

```
"1. Developed region: G7"
"2. Developed region: nonG7"
"3. Emerging region: BRIC"
"4. Emerging region: MIKT"
"5. Emerging region: G20"
"6. Developing region"
"7. Least developed region"
```

Sort only value, not the object (return array)

```
jq '[.features[].properties.pop_est] | sort' countries.geojson
```

Return

```
[
  -99,
  140,
  3140,
  3802,
  57600,
  …
  …
  198739269,
  240271522,
  313973000,
  1166079220,
  1338612970
]
```

Sort only value, not the object (return list, useful to pipe with other unix commands)

Ten minimum values with :

```
jq '[.features[].properties.pop_est] | sort[]' countries.geojson | head
```

Return

```
-99
140
3140
3802
57600
218519
227436
265100
306694
307899
```

Ten maximum values with :

```
jq '[.features[].properties.pop_est] | sort[]' countries.geojson | tail
```

Return

```
127078679
140041247
149229090
156050883
176242949
198739269
240271522
313973000
1166079220
1338612970
```

Alternatives to manipulate JSON with command line

> Underscore-Cli

You can see this list of alternatives and also this blog post

As you may need to beautify JSON output, you will see below some command line tools for this purpose

- Built-in module in Python with `python -mjson.tool file.json`
- jsbeautifier
- Underscore-Cli

Ressources

- https://github.com/stedolan/jq/issues/610
- https://doublebyteblog.wordpress.com/2014/12/03/json-to-geojson-with-jq/

- http://blog.mapillary.com/technology/2014/08/12/jq-power.html
- https://stedolan.github.io/jq/manual/

Comments, improvements

We have done some mistakes. you see some improvements to add, feel free to comment or contact us.