Command line parsing of JSON with jq

Michelle L. Gill
Metis, DS 8, Investigation 1
2016/07/07

What is jq?

- Parses JSON data from the command line – like `sed`
- `sed` works line-by-line while `jq` is heirarchical
- Integrates with command line tools (`awk` , `grep` , `paste`)
- Fast, powerful, easy to install

Why use jq?

- Faster than launching Jupyter notebook
- No GUI – ideal for remote/server work
- Pre-process large files before loading

How does it work?

- Traverses JSON keys (dictionary) and operates on data
- Jupyter notebooks are JSON

```
! head -n 20 project_luther.ipynb
```

```
{
 "cells": [
  {
```

```
   "cell_type": "code",
   "execution_count": 98,
   "metadata": {
    "ExecuteTime": {
     "end_time": "2016-07-03T06:16:52.895215",
     "start_time": "2016-07-03T06:16:52.875746"
    },
    "collapsed": true
   },
   "outputs": [],
   "source": [
    "import csv\n",
    "from collections import defaultdict\n",
    "import pprint\n",
    "from dateutil import parser as dtparser\n",
    "import datetime as dt\n",
    "\n",
```


Parsing Jupyter notebooks *is* useful

- Like running "`grep`" from command line – except better!
- Jupyter stores plots as base–64 encoded strings (messy)
- `jq` can clean output cells

```
img_text = ! sed -ne '948p' project_luther.ipynb
print(len(img_text[0]), img_text[0][:1000].strip())
```


```
(48518, '"image/png": "iVBORw0KGgoAAAANSUhEUgAAAscAAAFhCAYAAAACcWwdAAAAABHNCSVQI
```


Demo

Get the *input* code of the first cell

```
! head -n 18 project_luther.ipynb
```

```
{
 "cells": [
  {
   "cell_type": "code",
   "execution_count": 98,
   "metadata": {
    "ExecuteTime": {
     "end_time": "2016-07-03T06:16:52.895215",
     "start_time": "2016-07-03T06:16:52.875746"
    },
    "collapsed": true
   },
   "outputs": [],
   "source": [
    "import csv\n",
    "from collections import defaultdict\n",
    "import pprint\n",
    "from dateutil import parser as dtparser\n",
```

```
! jq '.cells[0].source' project_luther.ipynb
```

```
[
  "import csv\n",
  "from collections import defaultdict\n",
  "import pprint\n",
  "from dateutil import parser as dtparser\n",
  "import datetime as dt\n",
  "\n",
  "import pandas as pd\n",
  "\n",
  "import matplotlib.pyplot as plt\n",
  "from matplotlib.ticker import FuncFormatter\n",
  "import seaborn as sns\n",
  "\n",
  "pp = pprint.PrettyPrinter(indent=2)\n",
  "\n",
```

```
  "sns.set_context('talk')\n",
  "sns.set_style('ticks')\n",
  "sns.set_palette('dark')\n",
  "\n",
  "%matplotlib inline"
]
```

```
! jq '.cells[0].source' project_luther.ipynb | sed 's/^  "//g;s/\\n\",//g;s/^\[
```

```
import csv
from collections import defaultdict
import pprint
from dateutil import parser as dtparser
import datetime as dt

import pandas as pd

import matplotlib.pyplot as plt
from matplotlib.ticker import FuncFormatter
import seaborn as sns

pp = pprint.PrettyPrinter(indent=2)

sns.set_context('talk')
sns.set_style('ticks')
sns.set_palette('dark')

%matplotlib inline"
```

Demo

Select only code from two cells

Compare to actual notebook

```
! jq '.cells[1,2] | select(.cell_type=="code") | .source[]' project_luther.ipyn
```

```
filelist = ['turnstile_160430.txt',
            'turnstile_160507.txt',
            'turnstile_160514.txt',
            'turnstile_160521.txt',
            'turnstile_160528.txt',
            'turnstile_160604.txt',
            'turnstile_160611.txt',
            'turnstile_160618.txt',
            'turnstile_160625.txt'
            ]

data = defaultdict(list)

for fil in filelist:
    with open(fil, 'r') as fh:

        reader = csv.reader(fh, delimiter=',')

        for row in reader:
            if 'C/A' not in row:
                row_str = map(lambda x: x.strip(), row)
                data[tuple(row_str[:4])].append(row_str[4:])

key = data.keys()[0]
print(key)
for i in range(10):
    print(data[key][i])"
```

Demo

Remove output from last cell

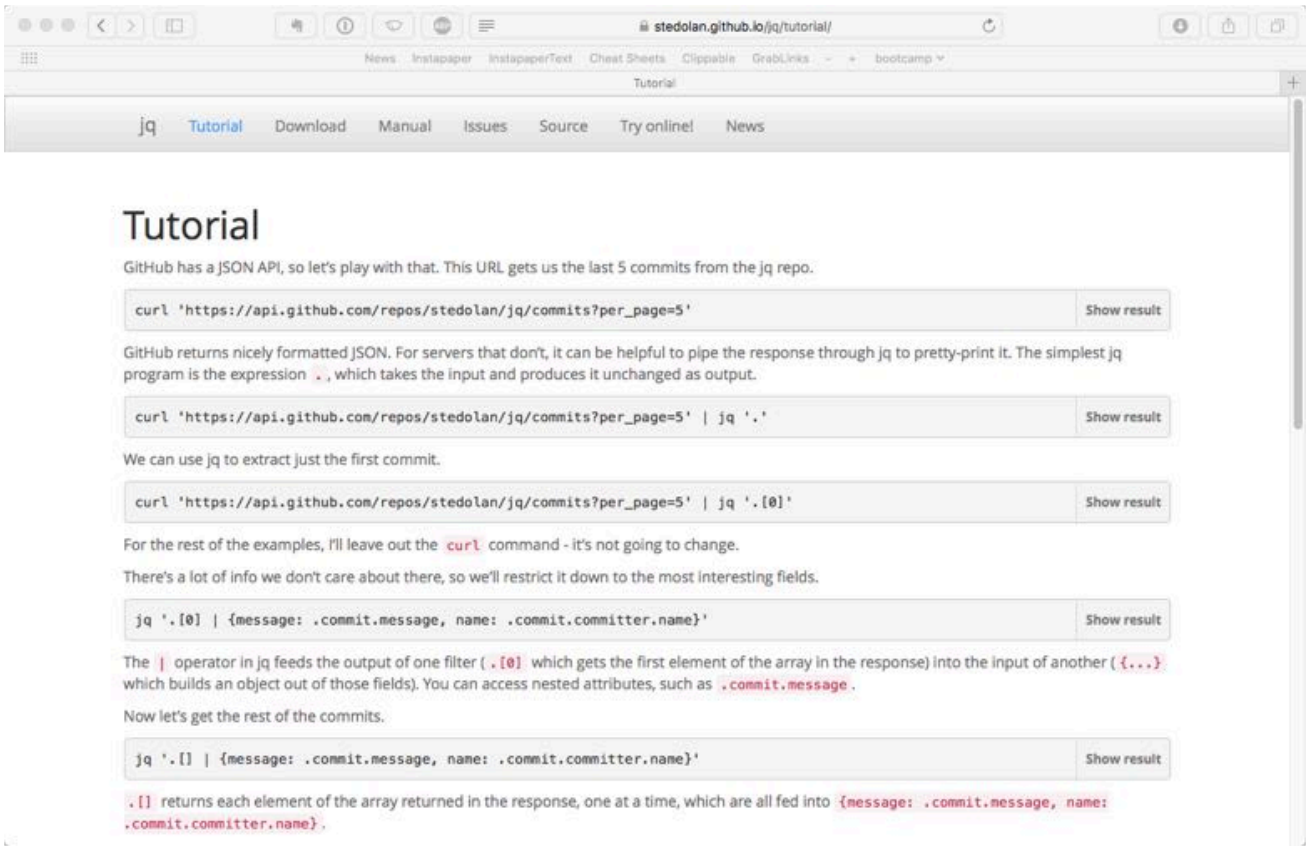
```
! jq '.cells[-1] | .outputs=[] ' project_luther.ipynb
```

```
{
  "cell_type": "code",
```

```
    "execution_count": 150,
    "metadata": {
      "ExecuteTime": {
        "end_time": "2016-07-03T06:33:46.120895",
        "start_time": "2016-07-03T06:33:45.736118"
      },
      "collapsed": false
    },
    "outputs": [],
    "source": [
      "n_stations = 10\n",
      "\n",
      "ax = ( pd_timeseries_by_station\n",
      "       .head(n=n_stations)\n",
      "       .reset_index()\n",
      "       .plot('STATION', 'ENTRIES',\n",
      "             kind='bar', figsize=(7,6), \n",
      "             legend=False)\n",
      "      )\n",
      "\n",
      "ax.set_xlabel('Station')\n",
      "\n",
      "scale_pow = 6\n",
      "ax.get_yaxis().set_major_formatter(FuncFormatter(scale_ticklabels))\n",
      "ax.set_ylabel('Total Ridership (Millions)')\n",
      "\n",
      "ax.set_title('Top {} Stations by Ridership \\nBetween {} and {}'\n",
      "             .format(n_stations, min_date, max_date))\n",
      "plt.xticks(rotation=30, ha='right')\n",
      "\n",
      "ax.set_xticklabels(map(lambda x: x.get_text().title(), ax.get_xticklabels(\n",
      "\n",
      "plt.tight_layout()\n",
      "sns.despine()"
    ]
  }
```


More information

- Manual: https://stedolan.github.io/jq/manual
- Tutorial: https://stedolan.github.io/jq/tutorial


stedolan.github.io/jq/tutorial/
News Instapaper InstapaperText Cheat Sheets Clippable GrabLinks bootcamp
Tutorial
jq Tutorial Download Manual Issues Source Try online! News
Tutorial
GitHub has a JSON API, so let's play with that. This URL gets us the last 5 commits from the jq repo.
curl 'https://api.github.com/repos/stedolan/jq/commits?per_page=5'
Show result
GitHub returns nicely formatted JSON. For servers that don't, it can be helpful to pipe the response through jq to pretty-print it. The simplest jq program is the expression ., which takes the input and produces it unchanged as output.
curl 'https://api.github.com/repos/stedolan/jq/commits?per_page=5' | jq '.'
Show result
We can use jq to extract just the first commit.
curl 'https://api.github.com/repos/stedolan/jq/commits?per_page=5' | jq '.[0]'
Show result
For the rest of the examples, I'll leave out the curl command - it's not going to change.
There's a lot of info we don't care about there, so we'll restrict it down to the most interesting fields.
jq '.[0] | {message: .commit.message, name: .commit.committer.name}'
Show result
The | operator in jq feeds the output of one filter (.[0] which gets the first element of the array in the response) into the input of another ({...} which builds an object out of those fields). You can access nested attributes, such as .commit.message.
Now let's get the rest of the commits.
jq '.[] | {message: .commit.message, name: .commit.committer.name}'
Show result
.[] returns each element of the array returned in the response, one at a time, which are all fed into {message: .commit.message, name: .commit.committer.name}.