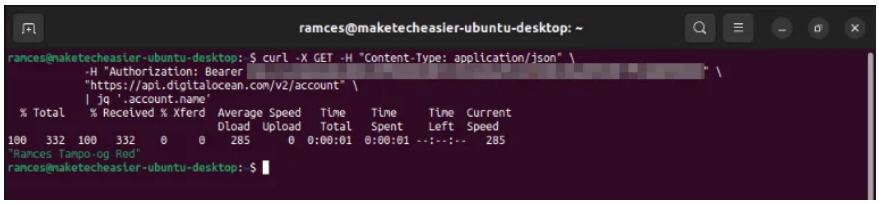
How to use jq command to process JSON in Linux

Jq is a powerful and highly flexible parser program that can stream and filter JSON data out of UNIX files and paths.

Jq is a powerful and highly flexible parser program that can stream and filter JSON data out of UNIX files and paths. This article will walk you through the basics of jq, presenting examples as well as some alternative implementations you can apply today.

What is jq used for?

The most common use of jq is to process and manipulate JSON responses from Software-as-a-Service (SaaS) APIs. For example, you can use jq with cURL to exploit Digitalocean API endpoints to get account details.



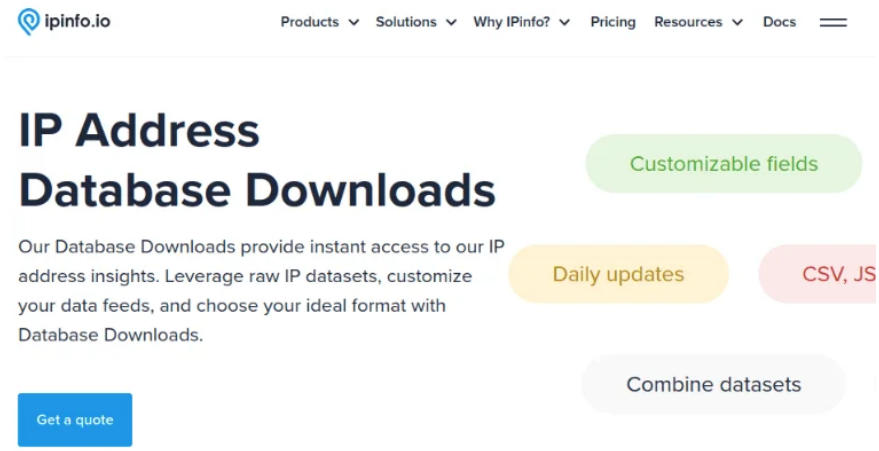
In addition, jq is also a powerful utility for managing large JSON files. Some of today's most popular database programs such as MongoDB, PostgreSQL, and MySQL support JSON as a way to store data. As such, learning jq gives an advantage in understanding how those database systems work.

Install and use jq

To get started with jq, install its binary package to the system:

```
sudo apt install jq
```

Find an open API endpoint where you can test jq. The example scenario will use the ipinfo.io IP check API.



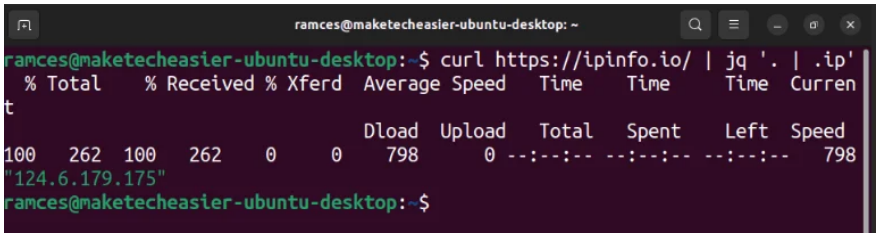
The most basic filter for jq is the dot (.) filter. The following command will output a JSON response as jq receives it from standard input:

```
curl https://ipinfo.io/ | jq '.'
```

Another basic filter is the symbol (|). Here is a special filter that passes the output of one filter as the input of another:

```
curl https://ipinfo.io/ | jq '. | .ip'
```

The value after the | operator is 'Object Identifier-Index'. This will search the JSON input to detect any variable that matches its text and print its value on the terminal. The following example is looking for the value of the key 'ip:'.



With the basics done, the following sections will show you some tricks you can do using jq.

1. Create a basic feed reader with jq

Most modern websites today provide open API endpoints to read data within their platform. For example, every Github repository has its own API URL to retrieve the latest commits and issues for that project.

You can use an API endpoint like this with jq to create your own simple 'RSS-like' feed. To start, use cURL to test if the endpoint is working fine:

```
curl https://api.github.com/repos/bitcoin/bitcoin/issues
```

Run the following command to print the first entry in the feed:

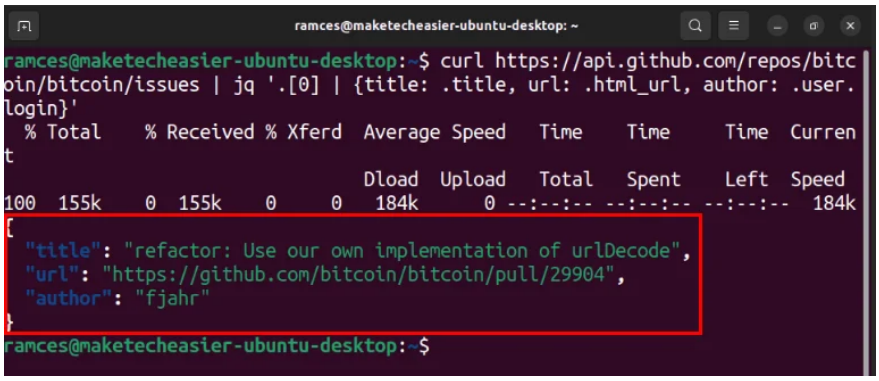
```
curl https://api.github.com/repos/bitcoin/bitcoin/issues | jq '.[0]'
```

This will show the different fields that the Github API sends to jq. You can use these to create your own custom JSON objects by passing the input to a curly braces filter ({}):

```
curl https://api.github.com/repos/bitcoin/bitcoin/issues | jq '.[0] | { title: .
```

Adding a comma (,) filter inside curly braces allows adding more fields to the custom object:

```
curl https://api.github.com/repos/bitcoin/bitcoin/issues | jq '.[0] | {title: .t
```



Removing **the '0'** inside the square brackets will apply the jq filter to the entire feed:

```
curl https://api.github.com/repos/bitcoin/bitcoin/issues | jq '.[] | {title: .ti
```

You can also create a small Bash script to display the latest issues from your favorite Github project. Paste the following block of code into the empty shell script file:

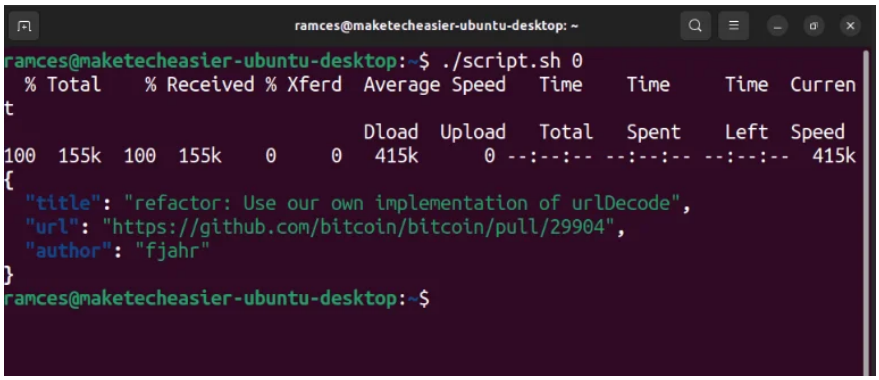
```
#!/bin/bash

# usage: ./script.sh [0 . 29]

REPO="https://api.github.com/repos/bitcoin/bitcoin/issues"

curl $REPO | jq ".[$1] | {title: .title, url: .html_url, author: .user.login}"
```

Save the file, then run the following command to make it executable:

```
chmod u+x ./script.sh
```

Check out the new feed reader by listing the latest issue in your favorite Github repo:

```
./script.sh 0
```



2. Read and search through the JSON database

In addition to reading data from the API, you can also use jq to manage JSON files in your local machine. Start by creating a simple JSON database file using your favorite text editor:

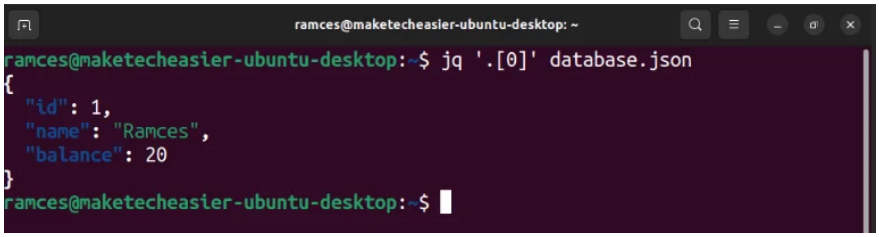
```
nano ./database.json
```

Paste the following block of data into the file, then save it:

```
[
 {"id": 1, "name": "Ramces", "balance": 20},
 {"id": 2, "name": "Alice", "balance": 30},
 {"id": 3, "name": "Bob", "balance": 10},
 {"id": 4, "name": "Charlie", "balance": 20},
 {"id": 5, "name": "Maria", "balance": 50}
]
```

Check if jq reads the JSON file correctly by printing the first object in the database array:

```
jq '.[0]' database.json
```

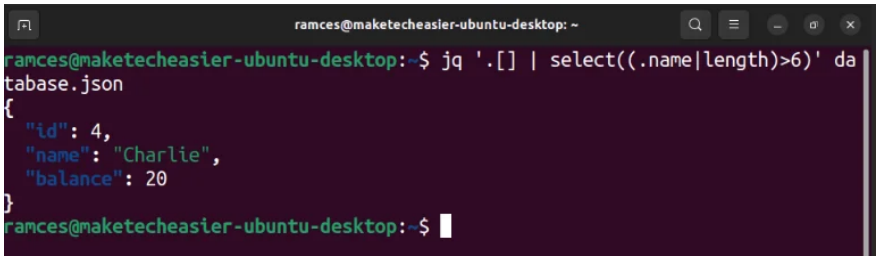


Execute a query on the JSON database using the 'Object Identifier-Index' filter. In this case, the author is looking for the value of the key '.name' on every entry in the database:

```
jq '.[] | .name' database.json
```

You can also use some of jq's built-in functions to filter queries based on certain characteristics. For example, it is possible to search and print all JSON objects with a '.name' value of more than 6 characters:

```
jq '.[] | select((.name|length)>6)' database.json
```



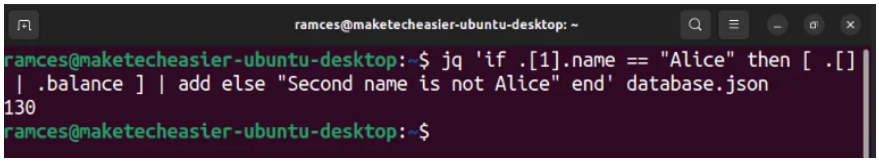
Works on JSON database with jq

Additionally, jq can operate on a JSON database similar to a basic spreadsheet. For example, the following command prints the sum of the '.balance' key for every object in the database:

```
jq '[.[] | .balance] | add' database.json
```

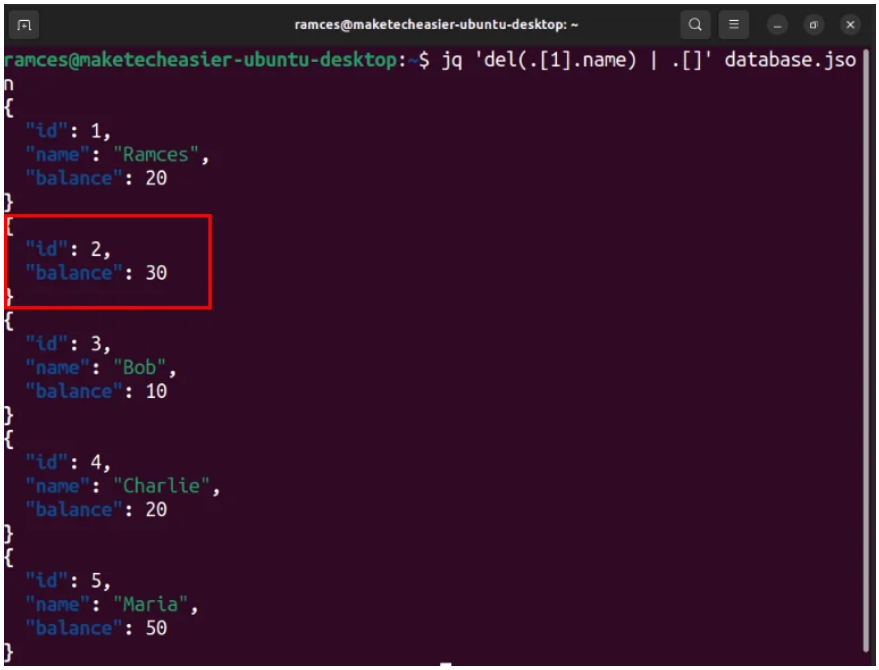
You can even extend this by adding conditional statements to your query. The following will only add '.balance' if the second object's '.name' value is 'Alice':

```
jq 'if .[1].name == "Alice" then [ .[] | .balance ] | add else "Second name is no
```



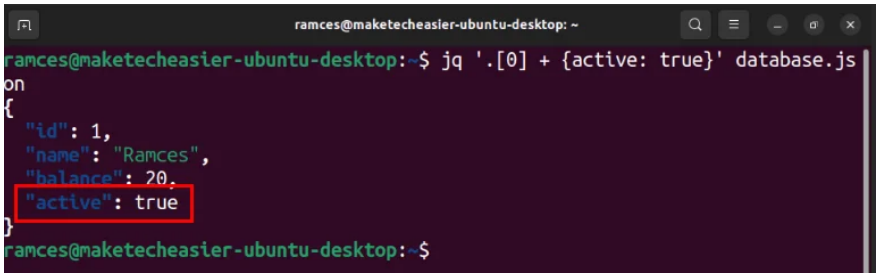
Variables can be temporarily removed from a JSON database. This can be useful if you are testing a filter and want to ensure that the filter can still process the data set:

```
jq 'del(.[1].name) | .[]' database.json
```



You can also insert new variables into the database using the '+' operator. For example, the following line adds the variable 'active: true' to the first object in the database:

```
jq '.[0] + {active: true}' database.json
```



Note: You can make the changes permanent by converting the output of the jq command to the original database file:

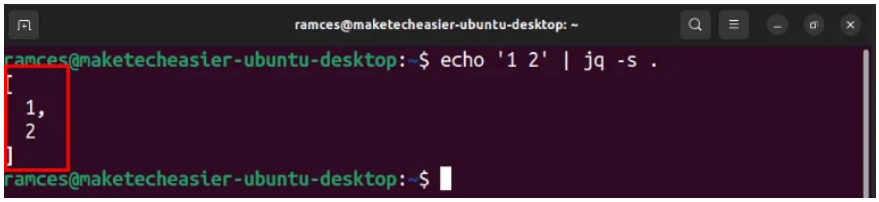
```
jq '.[0] + {active: true}' database.json > database.json
```

3. Convert non-JSON data in jq

Another great feature of jq is that it can accept and work with non-JSON data. To achieve that, the program uses an alternate 'slurp mode, in which it converts any space- and newline-separated data into a JSON array.

You can enable this feature by passing data into jq using the -s flag:

```
echo '1 2' | jq -s .
```

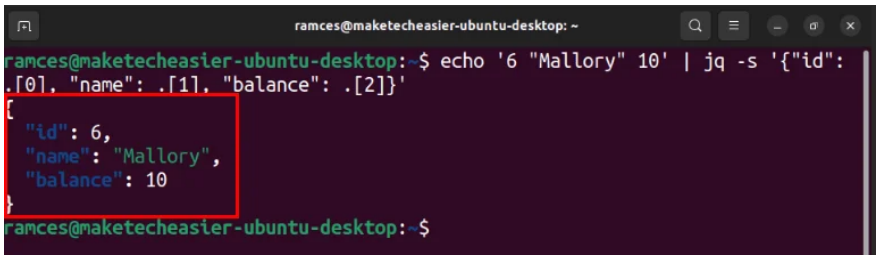


One advantage of converting raw data to an array is that you can process them using array index numbers. The following command adds two values ??by reference to their converted array location:

```
echo '1 2' | jq -s '.[0] + .[1]'
```

You can develop this array position further and build new JSON code around it. For example, this code converts the text from the echo command into a JSON object through a curly braces filter:

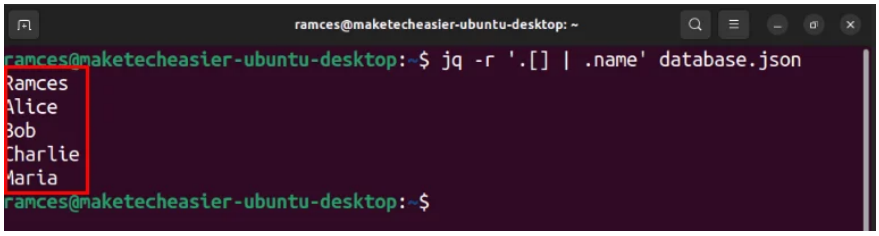
```
echo '6 "Mallory" 10' | jq -s '{"id": .[0], "name": .[1], "balance": .[2]}'
```



In addition to getting raw data, jq can also return non-JSON data as output. This is useful if you are using jq as part of a larger shell script and only need the results from its filters.

To do that, run jq followed by the -r flag. For example, the following command reads all names from the database file and returns it as plain text data:

```
jq -r '.[] | .name' database.json
```



You finished reading the article "**How to use jq command to process JSON in Linux**" edited by the **TipsMake** team. We hope this article has provided you with many useful tech tips and tricks. You can search for similar articles on tips and guides. Thank you for reading and for following us regularly.