

# jq(JSON Processing Tool)

1: What is Jq:

- Jq is a command-line tool to parse JSON Data on Linux Command Line
- Parsing JSON Data means
  - Reading JSON Data
  - Writing JSON Data
  - Modifying (Inserting/Deleting) JSON Data

2: Jq command syntax:

- Syntax:
  - `jq [options] 'LogicToParseJsonData' [input]` -> It gives Output
  - Note:
    - **Options** are always optional but based on requirement we can also pass options
    - **LogicToParseJsonData** can be framed with filters, conditions, functions etc ...
    - **Input** -> It can be file/directly a JSON String/JSON Data From Unix/Linux Command
  - `jq --help` command will give all info

## Filters:

- Filter are used to filter the data
  - **Identity Filter: [ . (DOT) is called Identity filter ]**
    - It is used or useful to validate the JSON data
    - It is called identity filter because its input and output both are identical
    - Syntax
      - `jq 'LogicToParseJsonData' [input]`
      - `jq . [input]`
      - `jq '' [input]`--> Prefer to use this one
      - `jq ":" [Input]`
  - **Field Filter:**
    - It is useful to get key/property value from a JSON Dta
    - Syntax:
      - `jq .key [input]`
      - `jq '.key' [input]`
      - `jq '.key' [input]`
    - We can also Chain keys/properties together to get nested objects.

- Syntax:
  - `jq '.key1.key2.key3' fileName`
- Note: Filters are also called as operators.

## **Jq Options to work with Raw String (-R and -r Options):**

What is RAW String:

- It is nothing but NON JSON DATA
- Option\_1
  - `-R` -> To Read NON Json Data (Read Raw String)
- Option\_2
  - `-r` -> To Get RAW String (print Output)
  - It will remove the the crotations like from "King" to King
  - eg: `jq -r ' LogicToParseJsonData ' [input]`

## **Jq Command with Field Filter & Comma:**

Comma(,):

- It is Useful to apply multiple Field Filters in one single jq command, so that we can get multiple key values.
  - eg: `jq ' LogicToParseJsonData , LogicToParseJsonData ' [Input]`

## **Jq Command with -S(Upper case 'S') Option to Sort Keys:**

- Sort the JSON file Keys in to alphabetical order
- Note: Only we can do in the output only not in the actual file.
  - eg: `jq -S ' LogicToParseJsonData ' [input]`

## **Jq Command with -s(Lower Case 's') option to combine Multiple JSON Files:**

- Use to create new output or file using 2 or more JSON files then we use - s(Lower Case) option
- Note: Result is an Array, where values are from input files.

## **Different Ways to pass JSON Data as an Input to Jq Command:**

There are 3 ways to pass JSON Data as input to Jq Command

- Syntax:
  - `jq [options] 'LogicToParseJsonData' [input]` → It Gives Output
  - `jq :[input]`
- Note:
  - `Input JSON Data Could be from a file or commandOutput or even directly JSON Array or Object`
  - `jq [options] 'LogicToParseJsonData' fileName`
  - `jq [options] 'LogicToParseJsonData' <<< "$(LinuxCommand Which produces output as a json)"`
  - `LinuxCommand Which produces output as a json | jq [options] 'LogicToParseJsonData'`
  - `jq [options] 'LogicToParseJsonData' <<< 'JsonArray/JsonObject'`
  - `echo 'JsonArray/JsonObject' | jq [options] 'LogicToParseJsonData'`

## Parsing API Response with Jq Command

pass API to jq in different ways

our api is : <https://randomuser.me/api/>

- `jq '! <<< "$(curl -s https://randomuser.me/api/)"`
- `curl -s https://randomuser.me/api/ | jq !'`

## Parsing Cloud CLI and Kubectl Command's Output with Jq

- eg: `jq '! <<< "$(aws iam list-users --output json)"`

### Iterator Filter or Array Filter for Arrays:

- Iterator filter or Array Filter is like a loop, which is useful to take and print one by one values from a JSON array.
  - `eg: jq '.[]' [Input File]`

### Iterator Filter with Index of Array Values:

#### Iterator Filter With Index Of Array Values

- There is an index number or position for each value in an array
- Using Index Value(s):
  - We can get required Array value
  - We can create a sub array
    - (It is also called as Slicing Operations)
- Note:
  - Array Values are also called as Items or Elements
  - We can use +ve or -ve index numbers to access or get Array Values

## Constructing JSON null, number, string and Boolean data's with jq without any input Json data

### Constructing JSON null, number, string and Boolean data's with jq without any input Json data

- We can create all kind of JSON data's using jq command with or without some input
- Syntax:

```
> jq -n "    --> null
> jq -n '    --> null
> jq -n 'null' --> null
<   > jq -n '5'  --> number
   > jq -n "I AM JQ" --> string
   > jq -n 'true/false' --> boolean
```

## Constructing JSON Array with Jq without Any Input JSON Data

### Constructing JSON Array with Jq without Any Input JSON Data

- Syntax:
- jq -n '[]'
- jq -n '[ 2,true,4, "value4"]'

## Constructing/Creating JSON Object with Jq without Any Input JSON Data

### Constructing JSON Object with Jq without Any Input JSON Data

- Syntax:
- jq -n '{}'
- jq -n '{"User": "VRTech"}'
- Note:
  - JSON Object with Input jq '{"age": .age}' fileName
- Create/ Construct JSON object with one of the input VALUE
  - eg: jq '{"new\_key": .inputfile\_key}' inputfile

## --tab and -c Options for Jq Command

- --tab
  - used for Indentation
  - eg: jq --tab '{"devops\_tools": .tools}' devopsinfo.json
- -c
  - Compact instead of pretty-printed output (in single line)
  - eg: jq -c '{"devops\_tools": .tools}' devopsinfo.json
- We can use both --tab and -c in jq command which ever mentioned last jq will take that(having heigh priority)

# Adding or Modifying a Key Value for a given JSON Object

## Adding or Modifying a Key Value for a given JSON Object

➤ There are two ways to Add or Modify a Key Value for a given Object

➤ First Way:

➤ Syntax:

➤ `jq '.key = Value' [input]`

➤ Value can be null, number, string, array, object

➤ Second Way:

➤ Syntax:

➤ `jq '.+= { "key": "Value" }' [input]`

➤ `jq '.+= { "key": "Value" }' [input]`

eg: `jq '.members[0].friends={"Vamsi": "Masters Friend", "Balakrishna": "Room Friend", "Anil": "PG Friend"}' family.json`

# Adding and Modifying a Value for a given JSON Array

## Adding or Modifying a Value for a given JSON Array

➤ We have two ways to Add and one way to Modify a Value for a given Array

➤ First Way:

➤ Syntax:

➤ `jq '.[index] = Value' [input]`

➤ `jq '.[.length]= Value' [input]` → It always add a Value

➤ Value can be null, number, string, array, object

➤ Second Way: (We can not modify the value, but we can Add a Value )

➤ Syntax:

➤ `jq '.= . + [value]' [input]`

➤ `jq '.+= [value]' [input]`

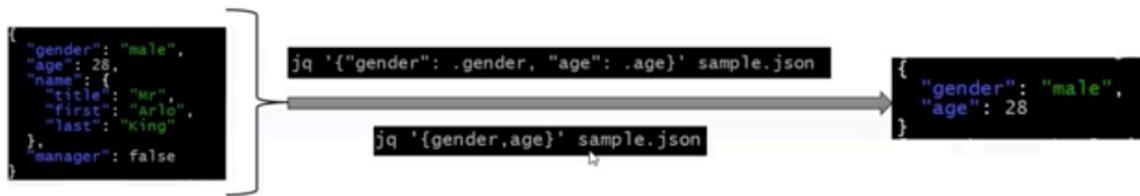
eg:

- `jq '[-1]="test'" family.json`
- `jq '.members.[-1]="test'" family.json`
- `jq '.members[length]={ "a": "b" }' family.json`
- `jq '.members.[length]=9999' family.json`

# Recreating an Object with shortcut

## Recreating an Object with shortcut

- Re-Creating an Object is nothing but creating a new object with same keys & values of input object



## Jq Command with Exit Status

### Jq Command with Exit Status

- Exit Status of any command can be find using \$? on Unix/Linux System's and it is useful to validate the execution status of the command
- The default exit status from Jq command is zero – irrespective of the operation status
- Use -e option to get exact status of the Jq Command
- Syntax of Jq Command with exit status:
  - `jq -e '.key' sample.json`

## Storing the Jq Command Output into a Shell Variable

### Storing the Jq Command Output into a Shell Variable

- Jq is like a Unix/Linux Command, So, we can store this command output also into a Shell Variable
- Syntax:
  - `result=$(jq [options] 'LogicToParseJsonData' [input])`
  - Or
  - `result=`jq [options] 'LogicToParseJsonData' [input]``
- Note: We must include -e option to know the exact status of Jq command

## Understanding About Jq Pipes

### Understanding About Jq Pipes

- Linux Pipes are useful to pass one command output as an input to another command
- Example: `cat abc.json | jq '.'`
- Jq Pipes are useful to combine multiple jq commands logics with one jq command

**eg:**

- `jq '.members[]' family.json | jq '.firstname'`
- `jq '.members[]' family.json | jq '.firstname' | jq 'length'`
- `jq '.members[].firstname | length' family.json`
-

## Creating JSON array from Input file

eg:

- jq '[.members[].firstname]' family.json

## FUNCTIONS (Jq Functions)

### Introduction to Jq Functions

- jq has many powerful built-in functions, which are useful to execute some logic on input JSON Data
- Functions are:
  - keys
  - length
  - min, max & add
  - reverse & sort
  - unique
  - del
  - env
  - join & split
  - has
  - map & reduce
  - select
  - match
  - type conversion functions (tonumber, tostring, uppercase,lowercase) etc...
- key and length Functions:**

- Jq Command Syntax with Function(s):
  - jq [options] 'LogicToParseJsonData' [input]
  - jq [options] 'functionName' [input]
- Note:
  - Functions are case-sensitive
  - Don't Apply . before any function

### keys & length Functions

- Syntax:
  - jq [options] 'keys' [input]
    - keys function gets the keys of an object as an array
  - jq [options] 'length' [input]
    - To find the
      - number of key value pairs for an object
      - length of an array
      - length of a string
- min, max, add, sort, reverse & unique Functions for array of NUMBERS
- eg: array = [1,3,4,5,2,5,6,7]

## min, max, add, sort, reverse & unique Functions

- Syntax:
  - Jq Command Syntax with Function(s):
    - jq [options] 'LogicToParseJsonData' [input]
    - jq [options] 'functionName' [input]
  - min: To find the minimum value from a given array
  - max: To find the maximum value from a given array
  - add: To find the sum of the values from a given array
  - sort: To get the array values in ascending order
  - reverse: To reverse the array values
  - unique: To remove duplicate values from an array
  - Note: We can apply all these functions on all data types, but most useful case is for numbers only

## **min\_by, max\_by, sort\_by , group\_by & unique\_by Functions for Array of Objects**

- **has Function:**

### has Function

- has function is used to validate :
    - Given object has a required key or not
    - Syntax:
      - `jq 'has("key")' filename` → input is object
    - Given array has a value at required index or not
    - Syntax:
      - `jq 'has(indexPosition)' filename` → input is array
  - Note: Output of the has function is always either true or false
  - **map & map\_values Functions**

## map & map\_values Functions

- Syntax:
  - `jq 'map(LogicToParseJsonData Array)`' [input] (input must be an array )
  - Note:
    - We must provide input as an array for map function
    - map applies the given logic (filter or function) on each element from input and it returns output as an array
    - So, input & outputs both are arrays for map function
  - `jq 'map_value(LogicToParseJsonData Array or Object)`' [input]
  - map\_values is same as map if input is an array, but it applies the given logic on each value of a key if input is an object
- **Select Function**

## select Function

- The select function is used to select value(s)/element(s)/item(s) from an object or from an array based on given test condition
- Syntax:
  - `jq 'select(test condition)' [input]`
  - Note:
    - test condition result must be a Boolean (either true or false)
    - It can be framed using comparison operators, logical operators, has function etc...