QuickRef.ME

YAML

This is a quick reference cheat sheet for understanding and writing YAML format configuration files.

Getting Started

Introduction

YAML is a data serialisation language designed to be directly writable and readable by humans

- YAML does not allow the use of tabs
- Must be space between the element parts
- YAML is CASE sensitive
- End your YAML file with the `.yaml` or `.yml` extension
- YAML is a superset of JSON
- Ansible playbooks are YAML files

Scalar types

```yaml
n1: 1            # integer
n2: 1.234        # float

s1: 'abc'        # string
s2: "abc"        # string
s3: abc          # string

b: false         # boolean type
```

```yaml
d: 2015-04-05    # date type
```

↓ Equivalent JSON

```json
{
  "n1": 1,
  "n2": 1.234,
  "s1": "abc",
  "s2": "abc",
  "s3": "abc",
  "b": false,
  "d": "2015-04-05"
}
```

Variables

```yaml
some_thing: &VAR_NAME foobar
other_thing: *VAR_NAME
```

↓ Equivalent JSON

```json
{
  "some_thing": "foobar",
  "other_thing": "foobar"
}
```

Comments

```yaml
# A single line comment example

# block level comment example
# comment line 1
# comment line 2
# comment line 3
```

Multiline strings

```yaml
description: |
  hello
  world
```

↓ Equivalent JSON

```json
{"description": "hello\nworld\n"}
```

Inheritance

```yaml
parent: &defaults
  a: 2
  b: 3

child:
  <<: *defaults
  b: 4
```

↓ Equivalent JSON

```json
{
  "parent": {
    "a": 2,
    "b": 3
  },
  "child": {
    "a": 2,
    "b": 4
  }
}
```

Reference

```yaml
values: &ref
  - Will be
  - reused below

other_values:
  i_am_ref: *ref
```

↓ Equivalent JSON

```json
{
  "values": [
    "Will be",
    "reused below"
  ],
  "other_values": {
    "i_am_ref": [
      "Will be",
      "reused below"
    ]
  }
}
```

Folded strings

```yaml
description: >
  hello
  world
```

↓ Equivalent JSON

```json
{"description": "hello world\n"}
```

Two Documents

```yaml
---
document: this is doc 1
---
document: this is doc 2
```

YAML uses `---` to separate directives from document content.

YAML Collections

Sequence

```yaml
- Mark McGwire
- Sammy Sosa
- Ken Griffey
```

↓ Equivalent JSON

```json
[
  "Mark McGwire",
  "Sammy Sosa",
  "Ken Griffey"
]
```

Mapping

```yaml
hr:  65       # Home runs
avg: 0.278    # Batting average
rbi: 147      # Runs Batted In
```

↓ Equivalent JSON

```
{
  "hr": 65,
  "avg": 0.278,
```

Mapping to Sequences

```
attributes:
  - a1
  - a2
methods: [getter, setter]
```

↓ Equivalent JSON

```
{
  "attributes": ["a1", "a2"],
  "methods": ["getter", "setter"]
}
```

Sequence of Mappings

```
children:
  - name: Jimmy Smith
    age: 15
  - name: Jimmy Smith
    age: 15
  -
    name: Sammy Sosa
    age: 12
```

↓ Equivalent JSON

```
{
  "children": [
    {"name": "Jimmy Smith", "age": 15},
    {"name": "Jimmy Smith", "age": 15},
    {"name": "Sammy Sosa", "age": 12}
  ]
}
```

Sequence of Sequences

```
my_sequences:
  - [1, 2, 3]
  - [4, 5, 6]
  -
    - 7
```

```yaml
    - 8
    - 9
    - 0
```

↓ Equivalent JSON

```json
{
  "my_sequences": [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9, 0]
  ]
}
```

Mapping of Mappings

```yaml
Mark McGwire: {hr: 65, avg: 0.278}
Sammy Sosa: {
    hr: 63,
    avg: 0.288
  }
```

↓ Equivalent JSON

```json
{
  "Mark McGwire": {
    "hr": 65,
    "avg": 0.278
  },
  "Sammy Sosa": {
    "hr": 63,
    "avg": 0.288
  }
}
```

Nested Collections

```yaml
Jack:
  id: 1
  name: Franc
  salary: 25000
  hobby:
    - a
    - b
  location: {country: "A", city: "A-A"}
```

↓ Equivalent JSON

```
{
  "Jack": {
    "id": 1,
    "name": "Franc",
    "salary": 25000,
    "hobby": ["a", "b"],
    "location": {
        "country": "A", "city": "A-A"
    }
  }
}
```

Unordered Sets

```
set1: !!set
  ? one
  ? two
set2: !!set {'one', "two"}
```

↓ Equivalent JSON

```
{
  "set1": {"one": null, "two": null},
  "set2": {"one": null, "two": null}
}
```

Sets are represented as a Mapping where each key is associated with a null value

Ordered Mappings

```
ordered: !!omap
- Mark McGwire: 65
- Sammy Sosa: 63
- Ken Griffy: 58
```

↓ Equivalent JSON

```
{
  "ordered": [
     {"Mark McGwire": 65},
     {"Sammy Sosa": 63},
     {"Ken Griffy": 58}
  ]
}
```

YAML Reference

Terms

- Sequences aka arrays or lists
- Scalars aka strings or numbers
- Mappings aka hashes or dictionaries

Based on the YAML.org refcard.

Document indicators

`%`	Directive indicator
`---`	Document header
`...`	Document terminator

Collection indicators

`?`	Key indicator
`:`	Value indicator
`-`	Nested series entry indicator
`,`	Separate in-line branch entries
`[]`	Surround in-line series branch
`{}`	Surround in-line keyed branch

Alias indicators

`&`	Anchor property
`*`	Alias indicator

Special keys

`=`	Default "value" mapping key
`<<`	Merge keys from another mapping

Scalar indicators

`''`	Surround in-line unescaped scalar
`"`	Surround in-line escaped scalar
`|`	Block scalar indicator
`>`	Folded scalar indicator
`-`	Strip chomp modifier (`|-` or `>-`)
`+`	Keep chomp modifier (`|+` or `>+`)
`1-9`	Explicit indentation modifier (`|1` or `>2`). Modifiers can be combined (`|2-`, `>+1`)

Tag Property (usually unspecified)

`none`	Unspecified tag (automatically resolved by application)
`!`	Non-specific tag (by default, `!!map`/`!!seq`/`!!str`)
`!foo`	Primary (by convention, means a local `!foo` tag)
`!!foo`	Secondary (by convention, means `tag:yaml.org,2002:foo`)
`!h!foo`	Requires `%TAG !h! <prefix>` (and then means `<prefix>foo`)
`!<foo>`	Verbatim tag (always means `foo`)

Misc indicators

`#`	Throwaway comment indicator
`` `@ ``	Both reserved for future use

Core types (default automatic tags)

`!!map`	`{Hash table, dictionary, mapping}`
`!!seq`	`{List, array, tuple, vector, sequence}`
`!!str`	Unicode string

Escape Codes

-	
Numeric	
`\x12` (8-bit)	`\u1234` (16-bit)

`\U00102030` (32-bit)		
Protective		
`\\` (\)	`\"` (")	`\ ` ( )
`\<TAB>` (TAB)		
C		
`\0` (NUL)	`\a` (BEL)	`\b` (BS)
`\f` (FF)	`\n` (LF)	`\r` (CR)
`\t` (TAB)	`\v` (VTAB)	
Additional		
`\e` (ESC)	`\_` (NBSP)	`\N` (NEL)
`\L` (LS)	`\P` (PS)	

More types

`!!set`	`{cherries, plums, apples}`
`!!omap`	`[one: 1, two: 2]`

Language Independent Scalar Types

`{~, null}`	Null (no value).
`[1234, 0x4D2, 02333]`	[Decimal int, Hexadecimal int, Octal int]
`[1_230.15, 12.3015e+02]`	[Fixed float, Exponential float]
`[.inf, -.Inf, .NAN]`	[Infinity (float), Negative, Not a number]
`{Y, true, Yes, ON}`	Boolean true
`{n, FALSE, No, off}`	Boolean false

Also see

YAML Reference Card (yaml.org)
Learn X in Y minutes (learnxinyminutes.com)
YAML lint online (yamllint.com)

Related Cheatsheet

ES6 Cheatsheet
Quick Reference

Express Cheatsheet
Quick Reference

JSON Cheatsheet
Quick Reference

Kubernetes Cheatsheet
Quick Reference

Recent Cheatsheet

Remote Work Revolution Cheatsheet
Quick Reference

Homebrew Cheatsheet
Quick Reference

PyTorch Cheatsheet
Quick Reference

Taskset Cheatsheet
Quick Reference