YAML Essentials Cheatsheet

A concise guide to the YAML data serialization format, covering syntax, data types, collections, and advanced features for configuration files and data exchange.

YAML Basics & Syntax

Fundamental Syntax Rules

Indentation
- Uses spaces for nesting, **no tabs**.
- Consistent indentation is crucial.

Comments
- Start with `#`.
- Can be on their own line or at the end of a line.

Key-Value Pairs
- Syntax: `key: value`
- Key must be unique within a map.
- Space after the colon is required.

Lists (Sequences)
- Start with `-` followed by a space.
- Items are at the same indentation level.

Maps (Mappings)
- Represent key-value pairs.
- Keys are unique strings by default.
- Nested structures are created via indentation.

Case Sensitivity
- Keys and scalar values are case-sensitive.

Document Separators
- `---` Separates directives from the document.
- `...` Indicates the end of a document.

Root Element
- A YAML file can be a single scalar, list, or map.

Reserved Characters
- `:` `#` `-` `?` `,` `[` `]` `{` `}` `@` `*` `&` `!` `|` `>` `'` `"` `%`
- These may need quoting or escaping depending on context.

Whitespace
- Significant for indentation.
- Trailing whitespace should be avoided.

Example Structure:

```yaml
person:
  name: Alice # This is a comment
  age: 30
  city: New York
```

Example List:

```yaml
- item 1
- item 2
- item 3
```

YAML Data Types (Scalars)

Scalar Types & Notation

Plain Scalars (Most Common)
- Strings without quotes.
- Numbers, booleans, null, dates/times are often parsed automatically.

Quoted Scalars
- Single (`'...'`) or Double (`"..."`).
- Used for strings containing special characters or for explicit string typing.
- Double quotes allow escape sequences (`\n`, `\t`, etc.).

Numeric Types
- Integers: `123`, `+45`, `-67`
- Floats: `1.23`, `-4.5e+6`, `.inf`, `-.inf`, `.nan`

Boolean Types
- Represent truth values.
- Common representations: `true`, `false`, `True`, `False`, `on`, `off`, `yes`, `no`.

Null Type
- Represents a null or empty value.
- Common representations: `null`, `Null`, `NULL`, `~`, `''` (empty string can often be interpreted as null depending on context/loader).

String Examples:

```yaml
plain: This is a plain string
single_quoted: 'This is a string with spaces'
double_quoted: "This string includes a newline\nand quotes \""
empty_string: ""
```

Multi-line Strings

- Literal Block (`|`): Preserves newlines and leading indentation of subsequent lines.
- Folded Block (`>`): Folds newlines into spaces, preserving blank lines.
- Indicator (`-` or `+`): Controls whether trailing newlines are kept (`+`) or stripped (`-`). Default is stripped.

Literal Block Example:

```yaml
poem: |
  This is the first line.
  This is the second line.

  This is the third line.
```

Folded Block Example:

```yaml
quote: >
  This is a very long
  sentence that should
  be folded into a
  single line.
```

Explicit Typing (Tags)

- Force a scalar to be a specific type.
- Syntax: `!!type value`

```yaml
string_int: !!str 123 # Forces '123' to be a string
int_string: !!int "123" # Forces "123" to be an integer
```

Date & Time Types

- YAML has standard representations for dates and times.
- `!!timestamp` is the common tag, often inferred.

```yaml
date: 2023-10-27
datetime: 2023-10-27T10:00:00Z
datetime_offset: 2023-10-27 10:00:00-05:00
```

Binary Data

- Represented using base64 encoding.
- Requires the `!!binary` tag.

```yaml
data: !!binary |
  R0lGODlhDAAMAKIFAOCwsP////8yMj
  IyAAAAAAACwAAAAADAAMAAACDpSP
  aLnjjmoCNloAAKeWwO
```

YAML Collections (Lists & Maps)

Lists (Sequences)

Block Style List

- Each item starts with `-` followed by a space.
- Items are at the same indentation level.
- Items can be scalars, maps, or other lists.

Example Block List:

```yaml
fruits:
  - Apple
  - Banana
  - Orange
```

Nested Block Lists

- Achieved through consistent indentation.

Example Nested Lists:

```yaml
matrix:
  - - 1
    - 2
  - - 3
    - 4
```

Flow Style List

- Similar to JSON arrays.
- Uses `[]` with items separated by `,`.

Example Flow List:

```yaml
colors: [red, green, blue]
```

List Containing Maps

- Common structure for lists of objects.

Example List of Maps:

```yaml
people:
  - name: Alice
    age: 30
  - name: Bob
    age: 25
```

Empty List

- Represented by just `[]`.

Example Empty List:

```yaml
empty_items: []
```

Using `?` and `:` in Lists (Less Common)

- YAML allows complex keys using `?`.

```
- ? key1
  : value1
  ? key2
  : value2
```

Combined List/Map Structure:

```
config:
  users:
    - id: 1
      name: UserA
    - id: 2
      name: UserB
  settings:
    theme: dark
    language: en
```

Maps (Mappings / Dictionaries)

Block Style Map

- Each key-value pair is on a new line, indented under the parent map.
- Syntax: `key: value`

Example Block Map:

```
user:
  name: Alice
  age: 30
  isStudent: false
```

Nested Block Maps

- Achieved by indenting child maps under their parent keys.

Example Nested Maps:

```
company:
  name: Example Corp
  address:
    street: 123 Main St
    city: Anytown
```

Flow Style Map

- Similar to JSON objects.
- Uses `{}` with key-value pairs separated by `,`.
- Syntax: `{ key1: value1, key2: value2 }`

Example Flow Map:

```
settings: { theme: dark, language: en }
```

Maps Containing Lists

- A common pattern to group related items under a key.

Example Map with List:

```
config:
  servers:
    - prod.example.com
    - dev.example.com
  ports: [80, 443]
```

Complex Keys (`?`)

- Any value can be a map key if denoted with `?`.

```
? - address
  - shipping
: This is the shipping address
```

Empty Map

- Represented by just `{}`.

Example Empty Map:

```
empty_settings: {}
```

Combining Styles

- Block and Flow styles can be mixed within a document.
- Flow style can be useful for short collections.

Advanced YAML Features

Anchors and Aliases

Anchors (`&`)

- Mark a node (scalar, list, map) for future reference.
- Syntax: `&anchor_name value`

Aliases (`*`)

- Reference a previously defined anchor.
- Syntax: `*anchor_name`
- The alias takes on the value/structure of the anchored node.

Usage:

- Avoid repetition of data.
- Define templates or common configurations.

Example Basic Usage:

```yaml
default_settings: &defaults
  timeout: 30
  retries: 3

service1:
  <<: *defaults # Merge defaults into service1
  url: http://svc1.example.com

service2:
  <<: *defaults # Merge defaults into service2
  url: http://svc2.example.com
  timeout: 60 # Override default timeout
```

Merging (`<<`)

- Special syntax used with aliases to merge the contents of a map anchor into the current map.
- Aliases are processed first, then subsequent keys in the current map override keys from the alias.

Anchors for Lists/Scalars:

- Can anchor non-map nodes too.

```yaml
common_list: &list_items
  - item A
  - item B

list1: *list_items
list2:
  - item C
  - *list_items # Adds the list as a sub-list
```

Best Practice:

- Place anchors near the top of the document or in a logical section.

Caution:

- Circular references using anchors/aliases are usually disallowed by parsers.

Tags

Purpose:

- Explicitly define the data type or structure of a node.
- Overrides the default type inferred by the parser.

Syntax:

- `!tag_name value`
- Can be applied to any node (scalar, list, map).

Standard YAML Tags:

- Prefixed with `!!` (e.g., `!!str`, `!!int`, `!!map`, `!!seq`, `!!bool`, `!!null`, `!!float`, `!!timestamp`, `!!binary`).
- These are usually inferred, but can be explicit.

Example Standard Tag:

```yaml
price: !!float "10.99" # Ensures it's a float, even if quoted
is_valid: !!bool "no" # Ensures it's a boolean
```

Custom Tags:

- Define application-specific types or objects.
- Syntax: `!your_tag_name value`
- The parser needs to know how to handle the custom tag.

Example Custom Tag:

```yaml
!!person
name: Alice
age: 30
```

Local Tags:

- Start with `!` followed by non-punctuation, typically `!tag` or `!prefix!tag`.
- Example: `!MyObject { key: value }`

Global Tags:

- Start with `!!` (standard) or a URI prefix (e.g., `!yaml!tag:yaml.org,2002:str`).

Directives (less common for users):

- `%YAML` - specifies YAML version.
- `%TAG` - associates a URI prefix with a handle.

```yaml
%TAG !ex! tag:example.com,2023:
--- # start of document
user: !ex!User # refers to tag:example.com,2023:User
  id: 123
```

Tagging Anchored Nodes:

- The tag applies to the node *before* the anchor or alias.

Multiple Documents

Purpose:
- Store multiple distinct YAML documents within a single file.
- Useful for config files, log streams, etc.

Separator (`---`)
- Marks the beginning of a new document.
- Must be on a line by itself.

End Marker (`...`)
- Optionally marks the end of a document.
- Useful to signal the end of the last document without a subsequent `---`.

Example Multiple Documents:

```yaml
# Document 1: User Config
user:
  name: Bob
  id: 456
---
# Document 2: App Settings
app:
  theme: light
  version: 1.0
...
```

Reading Multiple Documents
- YAML parsers typically offer functions to load all documents from a stream or file.

Common Use Cases:
- Configuration files for complex systems (e.g., Kubernetes).
- Data exchange protocols.

Each document is independent
- Anchors and aliases defined in one document are typically *not* accessible in subsequent documents (parser dependent, but standard behavior).

Directives
- Directives (`%YAML`, `%TAG`) apply only to the next document, unless the `%YAML` directive changes the version rules.

Tips, Tricks, and Best Practices

General Guidelines

Use Spaces, Not Tabs
- Absolutely critical for correct parsing. Most editors can be configured to insert spaces for tabs.

Consistent Indentation
- Stick to 2 or 4 spaces for indentation throughout your file.

Quote Strings Wisely
- Quote strings if they:
  - Start with a special character (`-`, `:`, `?`, etc.).
  - Contain internal special characters.
  - Look like numbers, booleans, or null (`'123'`, `'yes'`, `'null'`).

Keep Lines Readable
- Avoid overly long lines.
- Use multi-line string syntax (`|`, `>`) for longer text blocks.

Comments are Your Friend
- Explain complex structures, default values, or the purpose of sections.

Avoid Trailing Whitespace
- Can sometimes interfere with parsing, especially with multi-line strings.

Use Anchors/Aliases for Repetition
- Increases readability and reduces file size for repeated blocks of data.

Prefer Block Style for Structure
- Block style (indented) is generally more readable for complex nested structures than flow style.

Use Flow Style for Simple Collections
- Short lists (`[a, b, c]`) or maps (`{key: value}`) can be more concise in flow style.

Validate Your YAML
- Use online validators or command-line tools (`yamllint`, `yq`) to check syntax.

Common Pitfalls

Using Tabs for Indentation: Leads to parsing errors. Always use spaces.

Inconsistent Indentation: Mixing space counts (e.g., 2 spaces here, 4 spaces there) breaks structure.

Forgetting Space After Colon/Dash: `key:value` or `-item` is invalid. Needs space: `key: value`, `- item`.

Unquoted Strings: Values like `yes`, `no`, `on`, `off`, numbers, and dates can be auto-converted unexpectedly if not quoted when intended as strings.

Special Characters: Forgetting to quote or escape strings containing `:`, `-`, `*`, `&`, `?`, etc.

Complex Keys: Using non-string keys in maps without the `?` explicit notation (though many parsers handle simple non-string keys).

Unexpected Type Coercion: YAML's flexibility in type inference can sometimes lead to values being interpreted differently than intended (e.g., `1e2` as a float, `010` as an octal integer). Use explicit tags (`!!str`, `!!int`) if needed.