YAML Syntax

This page provides a basic overview of correct YAML syntax, which is how Ansible playbooks (our configuration management language) are expressed.

We use YAML because it is easier for humans to read and write than other common data formats like XML or JSON. Further, there are libraries available in most programming languages for working with YAML.

You may also wish to read :ref:`working_with_playbooks` at the same time to see how this is used in practice.

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\reference_appendices\[ansible-devel][docs][docsite][rst][reference_appendices]YAMLSyntax.rst`, **line 14**); *backlink*
>
> Unknown interpreted text role "ref".

YAML Basics

For Ansible, nearly every YAML file starts with a list. Each item in the list is a list of key/value pairs, commonly called a "hash" or a "dictionary". So, we need to know how to write lists and dictionaries in YAML.

There's another small quirk to YAML. All YAML files (regardless of their association with Ansible or not) can optionally begin with `---` and end with `...`. This is part of the YAML format and indicates the start and end of a document.

All members of a list are lines beginning at the same indentation level starting with a `"- "` (a dash and a space):

```
---
# A list of tasty fruits
- Apple
- Orange
- Strawberry
- Mango
...
```

A dictionary is represented in a simple `key: value` form (the colon must be followed by a space):

```
# An employee record
martin:
  name: Martin D'vloper
  job: Developer
  skill: Elite
```

More complicated data structures are possible, such as lists of dictionaries, dictionaries whose values are lists or a mix of both:

```
# Employee records
- martin:
    name: Martin D'vloper
    job: Developer
    skills:
      - python
      - perl
      - pascal
- tabitha:
    name: Tabitha Bitumen
    job: Developer
    skills:
      - lisp
      - fortran
      - erlang
```

Dictionaries and lists can also be represented in an abbreviated form if you really want to:

```
---
martin: {name: Martin D'vloper, job: Developer, skill: Elite}
fruits: ['Apple', 'Orange', 'Strawberry', 'Mango']
```

These are called "Flow collections".

Ansible doesn't really use these too much, but you can also specify a boolean value (true/false) in several forms:

```
create_key: yes
needs_agent: no
knows_oop: True
likes_emacs: TRUE
uses_cvs: false
```

Use lowercase 'true' or 'false' for boolean values in dictionaries if you want to be compatible with default yamllint options.

Values can span multiple lines using | or >. Spanning multiple lines using a "Literal Block Scalar" | will include the newlines and any trailing spaces. Using a "Folded Block Scalar" > will fold newlines to spaces; it's used to make what would otherwise be a very long line easier to read and edit. In either case the indentation will be ignored. Examples are:

```
include_newlines: |
            exactly as you see
            will appear these three
            lines of poetry

fold_newlines: >
            this is really a
            single line of text
            despite appearances
```

While in the above > example all newlines are folded into spaces, there are two ways to enforce a newline to be kept:

```
fold_some_newlines: >
    a
    b

    c
    d
      e
    f
```

Alternatively, it can be enforced by including newline `\n` characters:

```
fold_same_newlines: "a b\nc d\n  e\nf\n"
```

Let's combine what we learned so far in an arbitrary YAML example. This really has nothing to do with Ansible, but will give you a feel for the format:

```
---
# An employee record
name: Martin D'vloper
job: Developer
skill: Elite
employed: True
foods:
  - Apple
  - Orange
  - Strawberry
  - Mango
languages:
  perl: Elite
  python: Elite
  pascal: Lame
education: |
  4 GCSEs
  3 A-Levels
  BSc in the Internet of Things
```

That's all you really need to know about YAML to start writing *Ansible* playbooks.

Gotchas

While you can put just about anything into an unquoted scalar, there are some exceptions. A colon followed by a space (or newline) `": "` is an indicator for a mapping. A space followed by the pound sign `" #"` starts a comment.

Because of this, the following is going to result in a YAML syntax error:

```
foo: somebody said I should put a colon here: so I did

windows_drive: c:
```

...but this will work:

```
windows_path: c:\windows
```

You will want to quote hash values using colons followed by a space or the end of the line:

```
foo: 'somebody said I should put a colon here: so I did'

windows_drive: 'c:'
```

...and then the colon will be preserved.

Alternatively, you can use double quotes:

```
foo: "somebody said I should put a colon here: so I did"

windows_drive: "c:"
```

The difference between single quotes and double quotes is that in double quotes you can use escapes:

```
foo: "a \t TAB and a \n NEWLINE"
```

The list of allowed escapes can be found in the YAML Specification under "Escape Sequences" (YAML 1.1) or "Escape Characters" (YAML 1.2).

The following is invalid YAML:

```
foo: "an escaped \' single quote"
```

Further, Ansible uses "{{ var }}" for variables. If a value after a colon starts with a "{", YAML will think it is a dictionary, so you must quote it, like so:

```
foo: "{{ variable }}"
```

If your value starts with a quote the entire value must be quoted, not just part of it. Here are some additional examples of how to properly quote things:

```
foo: "{{ variable }}/additional/string/literal"
foo2: "{{ variable }}\\backslashes\\are\\also\\special\\characters"
foo3: "even if it's just a string literal it must all be quoted"
```

Not valid:

```
foo: "E:\\path\\"rest\\of\\path
```

In addition to `'` and `"` there are a number of characters that are special (or reserved) and cannot be used as the first character of an unquoted scalar: `[] {} > | * & ! % # ` @ ,.`

You should also be aware of `? : -`. In YAML, they are allowed at the beginning of a string if a non-space character follows, but YAML processor implementations differ, so it's better to use quotes.

In Flow Collections, the rules are a bit more strict:

```
a scalar in block mapping: this } is [ all , valid

flow mapping: { key: "you { should [ use , quotes here" }
```

Boolean conversion is helpful, but this can be a problem when you want a literal *yes* or other boolean values as a string. In these cases just use quotes:

```
non_boolean: "yes"
other_string: "False"
```

YAML converts certain strings into floating-point values, such as the string *1.0*. If you need to specify a version number (in a requirements.yml file, for example), you will need to quote the value if it looks like a floating-point value:

```
version: "1.0"
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\reference_appendices\[ansible-devel][docs][docsite][rst][reference_appendices]YAMLSyntax.rst, line 227)

Unknown directive type "seealso".

```
.. seealso::

   :ref:`working_with_playbooks`
       Learn what playbooks can do and how to write/run them.
   `YAMLLint <http://yamllint.com/>`_
       YAML Lint (online) helps you debug YAML syntax if you are having problems
   `GitHub examples directory <https://github.com/ansible/ansible-examples>`_
       Complete playbook files from the github project source
   `Wikipedia YAML syntax reference <https://en.wikipedia.org/wiki/YAML>`_
       A good guide to YAML syntax
   `Mailing List <https://groups.google.com/group/ansible-project>`_
       Questions? Help? Ideas?  Stop by the list on Google Groups
   :ref:`communication_irc`
       How to join Ansible chat channels (join #yaml for yaml-specific questions)
   `YAML 1.1 Specification <https://yaml.org/spec/1.1/>`_
       The Specification for YAML 1.1, which PyYAML and libyaml are currently
       implementing
   `YAML 1.2 Specification <https://yaml.org/spec/1.2/spec.html>`_
       For completeness, YAML 1.2 is the successor of 1.1
```