yamllint

Release 1.2.1

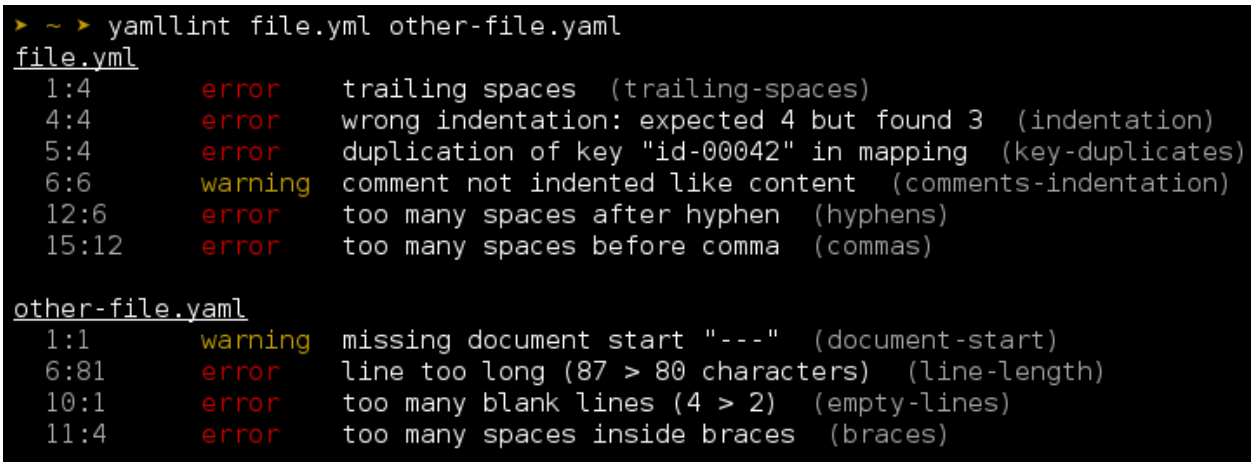
March 25, 2016

Contents

A linter for YAML files.

yamllint does not only check for syntax validity, but for weirdnesses like key repetition and cosmetic problems such as lines length, trailing spaces, indentation, etc.

Screenshot



Note: The default output format is inspired by eslint, a great linting tool for Javascript.

CHAPTER 2

Table of contents

2.1 Quickstart

2.1.1 Installing yamllint

On Fedora / CentOS:

```
sudo dnf install yamllint
```

On Debian 9+ / Ubuntu 16.04+:

```
sudo apt-get install yamllint
```

On older Debian / Ubuntu versions:

```
sudo add-apt-repository -y ppa:adrienverge/ppa && sudo apt-get update
sudo apt-get install yamllint
```

Alternatively using pip, the Python package manager:

```
sudo pip install yamllint
```

If you prefer installing from source, you can run, from the source directory:

```
python setup.py sdist
sudo pip install dist/yamllint-*.tar.gz
```

2.1.2 Running yamllint

Basic usage:

```
yamllint file.yml other-file.yaml
```

You can also lint all YAML files in a whole directory:

```
yamllint .
```

The output will look like (colors are not displayed here):

```
file.yml
  1:4       error    trailing spaces  (trailing-spaces)
  4:4       error    wrong indentation: expected 4 but found 3  (indentation)
  5:4       error    duplication of key "id-00042" in mapping  (key-duplicates)
```

```
  6:6       warning  comment not indented like content  (comments-indentation)
  12:6      error    too many spaces after hyphen  (hyphens)
  15:12     error    too many spaces before comma  (commas)

other-file.yaml
  1:1       warning  missing document start "---"  (document-start)
  6:81      error    line too long (87 > 80 characters)  (line-length)
  10:1      error    too many blank lines (4 > 2)  (empty-lines)
  11:4      error    too many spaces inside braces  (braces)
```

Add the `-f parsable` arguments if you need an output format parsable by a machine (for instance for syntax highlighting in text editors). The output will then look like:

```
file.yml:6:2: [warning] missing starting space in comment (comments)
file.yml:57:1: [error] trailing spaces (trailing-spaces)
file.yml:60:3: [error] wrong indentation: expected 4 but found 2 (indentation)
```

If you have a custom linting configuration file (see how to configure yamllint), it can be passed to yamllint using the `-c` option:

```
yamllint -c ~/myconfig file.yaml
```

Note: If you have a `.yamllint` file in your working directory, it will be automatically loaded as configuration by yamllint.

2.2 Configuration

yamllint uses a set of *rules* to check sources files for problems. Each rule is independent from the others, and can be enabled, disabled or tweaked. All these settings can be gathered in a configuration file.

To use a custom configuration file, either name it `.yamllint` in your working directory, or use the `-c` option:

```
yamllint -c ~/myconfig file.yaml
```

2.2.1 Default configuration

Unless told otherwise, yamllint uses its `default` configuration:

```
---

rules:
  braces:
    min-spaces-inside: 0
    max-spaces-inside: 0
  brackets:
    min-spaces-inside: 0
    max-spaces-inside: 0
  colons:
    max-spaces-before: 0
    max-spaces-after: 1
  commas:
    max-spaces-before: 0
    min-spaces-after: 1
    max-spaces-after: 1
```

```
  comments:
    level: warning
    require-starting-space: yes
    min-spaces-from-content: 2
  comments-indentation:
    level: warning
  document-end: disable
  document-start:
    level: warning
    present: yes
  empty-lines:
    max: 2
    max-start: 0
    max-end: 0
  hyphens:
    max-spaces-after: 1
  indentation:
    spaces: consistent
    indent-sequences: yes
    check-multi-line-strings: no
  key-duplicates: enable
  line-length:
    max: 80
    allow-non-breakable-words: yes
  new-line-at-end-of-file: enable
  new-lines:
    type: unix
  trailing-spaces: enable
```

Details on rules can be found on the rules page.

There is another pre-defined configuration named `relaxed`. As its name suggests, it is more tolerant.

It can be chosen using:

```
yamllint -d relaxed file.yml
```

2.2.2 Extending the default configuration

When writing a custom configuration file, you don't need to redefine every rule. Just extend the `default` configuration (or any already-existing configuration file).

For instance, if you just want to disable the `comments-indentation` rule, your file could look like this:

```
# This is my first, very own configuration file for yamllint!
# It extends the default conf by adjusting some options.

extends: default

rules:
  comments-indentation: disable  # don't bother me with this rule
```

Similarly, if you want to set the `line-length` rule as a warning and be less strict on block sequences indentation:

```
extends: default

rules:
  # 80 chars should be enough, but don't fail if a line is longer
  line-length:
```

```
    max: 80
    level: warning

  # accept both      key:
  #                    - item
  #
  # and              key:
  #                  - item
  indentation:
    indent-sequences: whatever
```

2.2.3 Custom configuration without a config file

It is possible – although not recommended – to pass custom configuration options to yamllint with the `-d` (short for `--config-data`) option.

Its content can either be the name of a pre-defined conf (example: `default` or `relaxed`) or a serialized YAML object describing the configuration.

For instance:

```
yamllint -d "{extends: relaxed, rules: {line-length: {max: 120}}}" file.yaml
```

2.2.4 Errors and warnings

Problems detected by yamllint can be raised either as errors or as warnings.

In both cases, the script will output them (with different colors when using the `standard` output format), but the exit code can be different. More precisely, the script will exit will a failure code *only when* there is one or more error(s).

2.3 Rules

When linting a document with yamllint, a series of rules (such as `line-length`, `trailing-spaces`, etc.) are checked against.

A configuration file can be used to enable or disable these rules, to set their level (*error* or *warning*), but also to tweak their options.

This page describes the rules and their options.

List of rules

2.3.1 braces

Use this rule to control the number of spaces inside braces (`{` and `}`).

Options

- `min-spaces-inside` defines the minimal number of spaces required inside braces.
- `max-spaces-inside` defines the maximal number of spaces allowed inside braces.

Examples

1. With `braces: {min-spaces-inside: 0, max-spaces-inside: 0}`

   the following code snippet would **PASS**:

```
object: {key1: 4, key2: 8}
```

   the following code snippet would **FAIL**:

```
object: { key1: 4, key2: 8 }
```

2. With `braces: {min-spaces-inside: 1, max-spaces-inside: 3}`

   the following code snippet would **PASS**:

```
object: { key1: 4, key2: 8 }
```

   the following code snippet would **PASS**:

```
object: { key1: 4, key2: 8   }
```

   the following code snippet would **FAIL**:

```
object: {    key1: 4, key2: 8   }
```

the following code snippet would **FAIL**:

```
object: {key1: 4, key2: 8 }
```

2.3.2 brackets

Use this rule to control the number of spaces inside brackets (`[` and `]`).

Options

- `min-spaces-inside` defines the minimal number of spaces required inside brackets.
- `max-spaces-inside` defines the maximal number of spaces allowed inside brackets.

Examples

1. With `brackets: {min-spaces-inside: 0, max-spaces-inside: 0}`

   the following code snippet would **PASS**:

```
object: [1, 2, abc]
```

   the following code snippet would **FAIL**:

```
object: [ 1, 2, abc ]
```

2. With `brackets: {min-spaces-inside: 1, max-spaces-inside: 3}`

   the following code snippet would **PASS**:

```
object: [ 1, 2, abc ]
```

   the following code snippet would **PASS**:

```
object: [ 1, 2, abc   ]
```

   the following code snippet would **FAIL**:

```
object: [    1, 2, abc   ]
```

   the following code snippet would **FAIL**:

```
object: [1, 2, abc ]
```

2.3.3 colons

Use this rule to control the number of spaces before and after colons (`:`).

Options

- `max-spaces-before` defines the maximal number of spaces allowed before colons (use `-1` to disable).
- `max-spaces-after` defines the maximal number of spaces allowed after colons (use `-1` to disable).

Examples

1. With `colons: {max-spaces-before: 0, max-spaces-after: 1}`

   the following code snippet would **PASS**:

```
object:
  - a
  - b
key: value
```

2. With `colons: {max-spaces-before: 1}`

   the following code snippet would **PASS**:

```
object :
  - a
  - b
```

   the following code snippet would **FAIL**:

```
object  :
  - a
  - b
```

3. With `colons: {max-spaces-after: 2}`

   the following code snippet would **PASS**:

```
first:  1
second: 2
third:  3
```

   the following code snippet would **FAIL**:

```
first: 1
2nd:   2
third: 3
```

2.3.4 commas

Use this rule to control the number of spaces before and after commas (`,`).

Options

- `max-spaces-before` defines the maximal number of spaces allowed before commas (use `-1` to disable).
- `min-spaces-after` defines the minimal number of spaces required after commas.
- `max-spaces-after` defines the maximal number of spaces allowed after commas (use `-1` to disable).

Examples

1. With `commas: {max-spaces-before: 0}`

   the following code snippet would **PASS**:

```
strange var:
  [10, 20, 30, {x: 1, y: 2}]
```

the following code snippet would **FAIL**:

```
strange var:
  [10, 20 , 30, {x: 1, y: 2}]
```

2. With `commas: {max-spaces-before: 2}`

   the following code snippet would **PASS**:

```
strange var:
  [10  , 20 , 30,  {x: 1  , y: 2}]
```

3. With `commas: {max-spaces-before: -1}`

   the following code snippet would **PASS**:

```
strange var:
  [10,
   20   , 30
   ,   {x: 1, y: 2}]
```

4. With `commas: {min-spaces-after: 1, max-spaces-after: 1}`

   the following code snippet would **PASS**:

```
strange var:
  [10, 20,30, {x: 1, y: 2}]
```

   the following code snippet would **FAIL**:

```
strange var:
  [10, 20,30,   {x: 1,   y: 2}]
```

5. With `commas: {min-spaces-after: 1, max-spaces-after: 3}`

   the following code snippet would **PASS**:

```
strange var:
  [10, 20,  30,  {x: 1,   y: 2}]
```

6. With `commas: {min-spaces-after: 0, max-spaces-after: 1}`

   the following code snippet would **PASS**:

```
strange var:
  [10, 20,30, {x: 1, y: 2}]
```

2.3.5 comments

Use this rule to control the position and formatting of comments.

Options

- Use `require-starting-space` to require a space character right after the `#`. Set to `yes` to enable, `no` to disable.

- `min-spaces-from-content` is used to visually separate inline comments from content. It defines the minimal required number of spaces between a comment and its preceding content.

Examples

1. With `comments: {require-starting-space: yes}`

   the following code snippet would **PASS**:

```
# This sentence
# is a block comment
```

   the following code snippet would **FAIL**:

```
#This sentence
#is a block comment
```

2. With `comments: {min-spaces-from-content: 2}`

   the following code snippet would **PASS**:

```
x = 2 ^ 127 - 1  # Mersenne prime number
```

   the following code snippet would **FAIL**:

```
x = 2 ^ 127 - 1 # Mersenne prime number
```

2.3.6 comments-indentation

Use this rule to force comments to be indented like content.

Examples

1. With `comments-indentation: {}`

   the following code snippet would **PASS**:

```
# Fibonacci
[0, 1, 1, 2, 3, 5]
```

   the following code snippet would **FAIL**:

```
  # Fibonacci
[0, 1, 1, 2, 3, 5]
```

   the following code snippet would **PASS**:

```
list:
    - 2
    - 3
    # - 4
    - 5
```

   the following code snippet would **FAIL**:

```
list:
    - 2
    - 3
#    - 4
    - 5
```

the following code snippet would **PASS**:

```
# This is the first object
obj1:
  - item A
  # - item B
# This is the second object
obj2: []
```

the following code snippet would **PASS**:

```
# This sentence
# is a block comment
```

the following code snippet would **FAIL**:

```
# This sentence
 # is a block comment
```

2.3.7 document-end

Use this rule to require or forbid the use of document end marker (`...`).

Options

- Set `present` to `yes` when the document end marker is required, or to `no` when it is forbidden.

Examples

1. With `document-end: {present: yes}`

   the following code snippet would **PASS**:

```
---
this:
  is: [a, document]
...
---
- this
- is: another one
...
```

   the following code snippet would **FAIL**:

```
---
this:
  is: [a, document]
---
- this
- is: another one
...
```

2. With `document-end: {present: no}`

   the following code snippet would **PASS**:

```
---
this:
  is: [a, document]
---
- this
- is: another one
```

   the following code snippet would **FAIL**:

```
---
this:
  is: [a, document]
...
---
- this
- is: another one
```

2.3.8 document-start

Use this rule to require or forbid the use of document start marker (`---`).

Options

- Set `present` to `yes` when the document start marker is required, or to `no` when it is forbidden.

Examples

1. With `document-start: {present: yes}`

   the following code snippet would **PASS**:

```
---
this:
  is: [a, document]
---
- this
- is: another one
```

   the following code snippet would **FAIL**:

```
this:
  is: [a, document]
---
- this
- is: another one
```

2. With `document-start: {present: no}`

   the following code snippet would **PASS**:

```
this:
  is: [a, document]
...
```

the following code snippet would **FAIL**:

```
---
this:
  is: [a, document]
...
```

2.3.9 empty-lines

Use this rule to set a maximal number of allowed consecutive blank lines.

Options

- `max` defines the maximal number of empty lines allowed in the document.
- `max-start` defines the maximal number of empty lines allowed at the beginning of the file. This option takes precedence over `max`.
- `max-end` defines the maximal number of empty lines allowed at the end of the file. This option takes precedence over `max`.

Examples

1. With `empty-lines: {max: 1}`

   the following code snippet would **PASS**:

```
- foo:
    - 1
    - 2

- bar: [3, 4]
```

   the following code snippet would **FAIL**:

```
- foo:
    - 1
    - 2


- bar: [3, 4]
```

2.3.10 hyphens

Use this rule to control the number of spaces after hyphens (`-`).

Options

- `max-spaces-after` defines the maximal number of spaces allowed after hyphens.

Examples

1. With `hyphens: {max-spaces-after: 1}`

   the following code snippet would **PASS**:

```
- first list:
    - a
    - b
- - 1
  - 2
  - 3
```

   the following code snippet would **FAIL**:

```
-  first list:
     - a
     - b
```

   the following code snippet would **FAIL**:

```
- - 1
  -  2
  - 3
```

2. With `hyphens: {max-spaces-after: 3}`

   the following code snippet would **PASS**:

```
-   key
-  key2
- key42
```

   the following code snippet would **FAIL**:

```
-    key
-   key2
-  key42
```

2.3.11 indentation

Use this rule to control the indentation.

Options

- `spaces` defines the indentation width, in spaces. Set either to an integer (e.g. `2` or `4`, representing the number of spaces in an indentation level) or to `consistent` to allow any number, as long as it remains the same within the file.
- `indent-sequences` defines whether block sequences should be indented or not (when in a mapping, this indentation is not mandatory – some people perceive the `-` as part of the indentation). Possible values: `yes`, `no`, `whatever` and `consistent`. `consistent` requires either all block sequences to be indented, or none to be. `whatever` means either indenting or not indenting individual block sequences is OK.
- `check-multi-line-strings` defines whether to lint indentation in multi-line strings. Set to `yes` to enable, `no` to disable.

Examples

1. With `indentation: {spaces: 1}`

   the following code snippet would **PASS**:

```
history:
 - name: Unix
   date: 1969
 - name: Linux
   date: 1991
nest:
 recurse:
  - haystack:
     needle
```

2. With `indentation: {spaces: 4}`

   the following code snippet would **PASS**:

```
history:
    - name: Unix
      date: 1969
    - name: Linux
      date: 1991
nest:
    recurse:
        - haystack:
              needle
```

   the following code snippet would **FAIL**:

```
history:
  - name: Unix
    date: 1969
  - name: Linux
    date: 1991
nest:
  recurse:
    - haystack:
        needle
```

3. With `indentation: {spaces: consistent}`

   the following code snippet would **PASS**:

```
history:
   - name: Unix
     date: 1969
   - name: Linux
     date: 1991
nest:
   recurse:
      - haystack:
           needle
```

   the following code snippet would **FAIL**:

```
some:
  Russian:
      dolls
```

4. With `indentation: {spaces: 2, indent-sequences: no}`

   the following code snippet would **PASS**:

```
list:
- flying
- spaghetti
- monster
```

   the following code snippet would **FAIL**:

```
list:
  - flying
  - spaghetti
  - monster
```

5. With `indentation: {spaces: 2, indent-sequences: whatever}`

   the following code snippet would **PASS**:

```
list:
- flying:
  - spaghetti
  - monster
- not flying:
    - spaghetti
    - sauce
```

6. With `indentation: {spaces: 2, indent-sequences: consistent}`

   the following code snippet would **PASS**:

```
- flying:
  - spaghetti
  - monster
- not flying:
  - spaghetti
  - sauce
```

   the following code snippet would **FAIL**:

```
- flying:
    - spaghetti
    - monster
- not flying:
  - spaghetti
  - sauce
```

7. With `indentation: {spaces: 4, check-multi-line-strings: yes}`

   the following code snippet would **PASS**:

```
Blaise Pascal:
    Je vous écris une longue lettre parce que
    je n'ai pas le temps d'en écrire une courte.
```

   the following code snippet would **PASS**:

```
Blaise Pascal: Je vous écris une longue lettre parce que
               je n'ai pas le temps d'en écrire une courte.
```

   the following code snippet would **FAIL**:

```
Blaise Pascal: Je vous écris une longue lettre parce que
  je n'ai pas le temps d'en écrire une courte.
```

the following code snippet would **FAIL**:

```
C code:
    void main() {
        printf("foo");
    }
```

the following code snippet would **PASS**:

```
C code:
    void main() {
    printf("bar");
    }
```

2.3.12 key-duplicates

Use this rule to prevent multiple entries with the same key in mappings.

Examples

1. With `key-duplicates: {}`

   the following code snippet would **PASS**:

```
- key 1: v
  key 2: val
  key 3: value
- {a: 1, b: 2, c: 3}
```

   the following code snippet would **FAIL**:

```
- key 1: v
  key 2: val
  key 1: value
```

   the following code snippet would **FAIL**:

```
- {a: 1, b: 2, b: 3}
```

   the following code snippet would **FAIL**:

```
duplicated key: 1
"duplicated key": 2

other duplication: 1
? >-
    other
    duplication
: 2
```

2.3.13 line-length

Use this rule to set a limit to lines length.

Options

- `max` defines the maximal (inclusive) length of lines.
- `allow-non-breakable-words` is used to allow non breakable words (without spaces inside) to overflow the limit. This is useful for long URLs, for instance. Use `yes` to allow, `no` to forbid.

Examples

1. With `line-length: {max: 70}`

   the following code snippet would **PASS**:

```
long sentence:
  Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do
  eiusmod tempor incididunt ut labore et dolore magna aliqua.
```

   the following code snippet would **FAIL**:

```
long sentence:
  Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod
  tempor incididunt ut labore et dolore magna aliqua.
```

2. With `line-length: {max: 60, allow-non-breakable-words: yes}`

   the following code snippet would **PASS**:

```
this:
  is:
    - a:
        http://localhost/very/very/very/very/very/very/very/very/long/url

# this comment is too long,
# but hard to split:
# http://localhost/another/very/very/very/very/very/very/very/very/long/url
```

   the following code snippet would **FAIL**:

```
- this line is waaaaaaaaaaaaaay too long but could be easily splitted...
```

3. With `line-length: {max: 60, allow-non-breakable-words: no}`

   the following code snippet would **FAIL**:

```
this:
  is:
    - a:
        http://localhost/very/very/very/very/very/very/very/very/long/url
```

2.3.14 new-line-at-end-of-file

Use this rule to require a new line character (`\n`) at the end of files.

The POSIX standard requires the last line to end with a new line character. All UNIX tools expect a new line at the end of files. Most text editors use this convention too.

2.3.15 new-lines

Use this rule to force the type of new line characters.

Options

- Set `type` to `unix` to use UNIX-typed new line characters (`\n`), or `dos` to use DOS-typed new line characters (`\r\n`).

2.3.16 trailing-spaces

Use this rule to forbid trailing spaces at the end of lines.

Examples

1. With `trailing-spaces: {}`

   the following code snippet would **PASS**:

```
this document doesn't contain
any trailing
spaces
```

   the following code snippet would **FAIL**:

```
this document contains
trailing spaces
on lines 1 and 3
```

2.4 Development

yamllint provides both a script and a Python module. The latter can be used to write your own linting tools:

class `yamllint.linter.`**`LintProblem`**(*line*, *column*, *desc='<no description>'*, *rule=None*)

Represents a linting problem found by yamllint.

`column` = None

Column on which the problem was found (starting at 1)

`desc` = None

Human-readable description of the problem

`line` = None

Line on which the problem was found (starting at 1)

`rule` = None

Identifier of the rule that detected the problem

`yamllint.linter.`**`run`**(*input*, *conf*)

Lints a YAML source.

Returns a generator of LintProblem objects.

Parameters

- **`input`** – buffer, string or stream to read from

- **conf** – yamllint configuration object

2.5 Integration with text editors

Most text editors support syntax checking and highlighting, to visually report syntax errors and warnings to the user. yamllint can be used to syntax-check YAML source, but a bit of configuration is required depending on your favorite text editor.

2.5.1 Vim

Assuming that the syntastic plugin is installed, add to your `.vimrc`:

```
let g:syntastic_yaml_checkers = ['yamllint']
```

2.5.2 Neovim

Assuming that the neomake plugin is installed, yamllint is supported by default. It is automatically enabled when editing YAML files.

2.5.3 Other text editors

Help wanted!

Your favorite text editor is not listed here? Help us improve by adding a section (by opening a pull-request or issue on GitHub).

y

C

D

L

R

Y