yamllint

Release 1.21.0

Mar 24, 2020

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

```
➤ ~ ➤ yamllint file.yml other-file.yaml
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)
```

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

Table of contents

2.1 Quickstart

2.1.1 Installing yamllint

On Fedora / CentOS (note: EPEL is required on CentOS):

```
sudo dnf install yamllint
```

On Debian 8+ / Ubuntu 16.04+:

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

On Mac OS 10.11+:

```
brew install yamllint
```

On FreeBSD:

```
pkg install py36-yamllint
```

On OpenBSD:

```
doas pkg_add py3-yamllint
```

Alternatively using pip, the Python package manager:

```
pip install --user yamllint
```

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

```
python setup.py sdist
pip install --user 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 .
```

Or lint a YAML stream from standard input:

```
echo -e 'this: is\nvalid: YAML' | 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)
```

By default, the output of yamllint is colored when run from a terminal, and pure text in other cases. Add the `-f standard` arguments to force non-colored output. Use the `-f colored` arguments to force colored output.

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 source 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, use the `-c` option:

```
yamllint -c /path/to/myconfig file-to-lint.yaml
```

If `-c` is not provided, yamllint will look for a configuration file in the following locations (by order of preference):

- `.yamllint`, `.yamllint.yaml` or `.yamllint.yml` in the current working directory
- `$XDG_CONFIG_HOME/yamllint/config`
- `~/.config/yamllint/config`

Finally if no config file is found, the default configuration is applied.

2.2.1 Default configuration

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

```yaml
---

yaml-files:
  - '*.yaml'
  - '*.yml'
  - '.yamllint'

rules:
  braces: enable
  brackets: enable
  colons: enable
  commas: enable
  comments:
    level: warning
  comments-indentation:
    level: warning
  document-end: disable
  document-start:
    level: warning
  empty-lines: enable
  empty-values: disable
  hyphens: enable
  indentation: enable
  key-duplicates: enable
  key-ordering: disable
  line-length: enable
  new-line-at-end-of-file: enable
  new-lines: enable
  octal-values: disable
  quoted-strings: disable
  trailing-spaces: enable
  truthy:
    level: warning
```

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:

```yaml
---

extends: default
```

(continues on next page)

```yaml
rules:
  braces:
    level: warning
    max-spaces-inside: 1
  brackets:
    level: warning
    max-spaces-inside: 1
  colons:
    level: warning
  commas:
    level: warning
  comments: disable
  comments-indentation: disable
  document-start: disable
  empty-lines:
    level: warning
  hyphens:
    level: warning
  indentation:
    level: warning
    indent-sequences: consistent
  line-length:
    level: warning
    allow-non-breakable-inline-mappings: true
  truthy: disable
```

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:

```yaml
# 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:

```yaml
extends: default

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

Chapter 2. Table of contents

```
  # 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. The CLI will output them (with different colors when using the `colored` output format, or `auto` when run from a terminal).

By default the script will exit with a return code `1` *only when* there is one or more error(s).

However if strict mode is enabled with the `-s` (or `--strict`) option, the return code will be:

- `0` if no errors or warnings occur
- `1` if one or more errors occur
- `2` if no errors occur, but one or more warnings occur

If the script is invoked with the `--no-warnings` option, it won't output warning level problems, only error level ones.

2.2.5 YAML files extensions

To configure what yamllint should consider as YAML files, set `yaml-files` configuration option. The default is:

```
yaml-files:
  - '*.yaml'
  - '*.yml'
  - '.yamllint'
```

The same rules as for ignoring paths apply (`.gitignore`-style path pattern, see below).

2.2.6 Ignoring paths

It is possible to exclude specific files or directories, so that the linter doesn't process them.

You can either totally ignore files (they won't be looked at):

```
extends: default

ignore: |
  /this/specific/file.yaml
  all/this/directory/
  *.template.yaml
```

or ignore paths only for specific rules:

```
extends: default

rules:
  trailing-spaces:
    ignore: |
      /this-file-has-trailing-spaces-but-it-is-OK.yaml
      /generated/*.yaml
```

Note that this `.gitignore`-style path pattern allows complex path exclusion/inclusion, see the pathspec README file for more details. Here is a more complex example:

```
# For all rules
ignore: |
  *.dont-lint-me.yaml
  /bin/
  !/bin/*.lint-me-anyway.yaml

extends: default

rules:
  key-duplicates:
    ignore: |
      generated
      *.template.yaml
  trailing-spaces:
    ignore: |
      *.ignore-trailing-spaces.yaml
      ascii-art/*
```

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

- *braces*
- *brackets*
- *colons*

- *commas*
- *comments*
- *comments-indentation*
- *document-end*
- *document-start*
- *empty-lines*
- *empty-values*
- *hyphens*
- *indentation*
- *key-duplicates*
- *key-ordering*
- *line-length*
- *new-line-at-end-of-file*
- *new-lines*
- *octal-values*
- *quoted-strings*
- *trailing-spaces*
- *truthy*

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

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

   the following code snippet would **PASS**:

   ```
   object: {}
   ```

   the following code snippet would **FAIL**:

   ```
   object: { }
   ```

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

   the following code snippet would **PASS**:

   ```
   object: {          }
   ```

   the following code snippet would **FAIL**:

   ```
   object: {}
   ```

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

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

   the following code snippet would **PASS**:

```
object: []
```

   the following code snippet would **FAIL**:

```
object: [ ]
```

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

   the following code snippet would **PASS**:

```
object: [         ]
```

   the following code snippet would **FAIL**:

```
object: []
```

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 `true` to enable, `false` to disable.

- Use `ignore-shebangs` to ignore a [shebang](#) at the beginning of the file when `require-starting-space` is set.

- `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: true}`

   the following code snippet would **PASS**:

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

   the following code snippet would **PASS**:

   ```
   ############################
   ## This is some documentation
   ```

   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 `true` when the document end marker is required, or to `false` when it is forbidden.

Examples

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

   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:  false}`

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 `true` when the document start marker is required, or to `false` when it is forbidden.

Examples

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

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:  false}`

   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 empty-values

Use this rule to prevent nodes with empty content, that implicitly result in `null` values.

Options

- Use `forbid-in-block-mappings` to prevent empty values in block mappings.
- Use `forbid-in-flow-mappings` to prevent empty values in flow mappings.

Examples

1. With `empty-values:  {forbid-in-block-mappings:  true}`

   the following code snippets would **PASS**:

   ```
   some-mapping:
     sub-element: correctly indented
   ```

   ```
   explicitly-null: null
   ```

   the following code snippets would **FAIL**:

   ```
   some-mapping:
   sub-element: incorrectly indented
   ```

   ```
   implicitly-null:
   ```

2. With `empty-values:  {forbid-in-flow-mappings:  true}`

   the following code snippet would **PASS**:

   ```
   {prop: null}
   {a: 1, b: 2, c: 3}
   ```

   the following code snippets would **FAIL**:

   ```
   {prop: }
   ```

   ```
   {a: 1, b:, c: 3}
   ```

2.3.11 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
   ```

2.3.12 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: `true`, `false`, `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 `true` to enable, `false` 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: false}`

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: true}`

   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");
    }
```

2.3.13 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
```

2.3.14 key-ordering

Use this rule to enforce alphabetical ordering of keys in mappings. The sorting order uses the Unicode code point number. As a result, the ordering is case-sensitive and not accent-friendly (see examples below).

Examples

1. With `key-ordering:   {}`

   the following code snippet would **PASS**:

```
- key 1: v
  key 2: val
  key 3: value
- {a: 1, b: 2, c: 3}
- T-shirt: 1
  T-shirts: 2
  t-shirt: 3
  t-shirts: 4
- hair: true
  hais: true
  haïr: true
  haïssable: true
```

   the following code snippet would **FAIL**:

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

   the following code snippet would **FAIL**:

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

   the following code snippet would **FAIL**:

```
- T-shirt: 1
  t-shirt: 2
  T-shirts: 3
  t-shirts: 4
```

   the following code snippet would **FAIL**:

```
- haïr: true
  hais: true
```

2.3.15 line-length

Use this rule to set a limit to lines length.

Note: with Python 2, the `line-length` rule may not work properly with unicode characters because of the way strings are represented in bytes. We recommend running yamllint with Python 3.

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 `true` to allow, `false` to forbid.

- `allow-non-breakable-inline-mappings` implies `allow-non-breakable-words` and extends it to also allow non-breakable words in inline mappings.

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:   true}`

   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 split...
   ```

   and the following code snippet would also **FAIL**:

```
- foobar: http://localhost/very/very/very/very/very/very/very/very/long/url
```

3. With `line-length: {max: 60, allow-non-breakable-words: true, allow-non-breakable-inline-mappings: true}`

   the following code snippet would **PASS**:

```
- foobar: http://localhost/very/very/very/very/very/very/very/very/long/url
```

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

   the following code snippet would **FAIL**:

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

2.3.16 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.

2.3.17 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`).

2.3.18 octal-values

Use this rule to prevent values with octal numbers. In YAML, numbers that start with `0` are interpreted as octal, but this is not always wanted. For instance `010` is the city code of Beijing, and should not be converted to `8`.

Examples

1. With `octal-values: {forbid-implicit-octal: true}`

   the following code snippets would **PASS**:

```
user:
  city-code: '010'
```

   the following code snippets would **PASS**:

```
user:
  city-code: 010,021
```

the following code snippets would **FAIL**:

```
user:
  city-code: 010
```

2. With `octal-values:  {forbid-explicit-octal:  true}`

the following code snippets would **PASS**:

```
user:
  city-code: '0o10'
```

the following code snippets would **FAIL**:

```
user:
  city-code: 0o10
```

2.3.19 quoted-strings

Use this rule to forbid any string values that are not quoted, or to prevent quoted strings without needing it. You can also enforce the type of the quote used.

Options

- `quote-type` defines allowed quotes: `single`, `double` or `any` (default).
- `required` defines whether using quotes in string values is required (`true`, default) or not (`false`), or only allowed when really needed (`only-when-needed`).

Note: Multi-line strings (with `|` or `>`) will not be checked.

Examples

1. With `quoted-strings:  {quote-type:  any, required:  true}`

the following code snippet would **PASS**:

```
foo: "bar"
bar: 'foo'
number: 123
boolean: true
```

the following code snippet would **FAIL**:

```
foo: bar
```

2. With `quoted-strings:  {quote-type:  single, required:  only-when-needed}`

the following code snippet would **PASS**:

```
foo: bar
bar: foo
not_number: '123'
not_boolean: 'true'
not_comment: '# comment'
```

```
not_list: '[1, 2, 3]'
not_map: '{a: 1, b: 2}'
```

the following code snippet would **FAIL**:

```
foo: 'bar'
```

2.3.20 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
   ```

2.3.21 truthy

Use this rule to forbid non-explictly typed truthy values other than allowed ones (by default: `true` and `false`), for example `YES` or `off`.

This can be useful to prevent surprises from YAML parsers transforming `[yes, FALSE, Off]` into `[true, false, false]` or `{y:  1, yes:  2, on:  3, true:  4, True:  5}` into `{y:  1, true: 5}`.

Options

- `allowed-values` defines the list of truthy values which will be ignored during linting. The default is `['true',  'false']`, but can be changed to any list containing: `'TRUE'`, `'True'`, `'true'`, `'FALSE'`, `'False'`, `'false'`, `'YES'`, `'Yes'`, `'yes'`, `'NO'`, `'No'`, `'no'`, `'ON'`, `'On'`, `'on'`, `'OFF'`, `'Off'`, `'off'`.

Examples

1. With `truthy:  {}`

   the following code snippet would **PASS**:

```
boolean: true

object: {"True": 1, 1: "True"}

"yes":  1
"on":   2
"True": 3

 explicit:
   string1: !!str True
   string2: !!str yes
   string3: !!str off
   encoded: !!binary |
              True
              OFF
              pad==  # this decodes as 'N»8Qii'
   boolean1: !!bool true
   boolean2: !!bool "false"
   boolean3: !!bool FALSE
   boolean4: !!bool True
   boolean5: !!bool off
   boolean6: !!bool NO
```

the following code snippet would **FAIL**:

```
object: {True: 1, 1: True}
```

the following code snippet would **FAIL**:

```
yes:  1
on:   2
True: 3
```

2. With `truthy:  {allowed-values:  ["yes", "no"]}`

   the following code snippet would **PASS**:

```
- yes
- no
- "true"
- 'false'
- foo
- bar
```

the following code snippet would **FAIL**:

```
- true
- false
- on
- off
```

2.4 Disable with comments

2.4.1 Disabling checks for a specific line

To prevent yamllint from reporting problems for a specific line, add a directive comment (`# yamllint disable-line ...`) on that line, or on the line above. For instance:

```
# The following mapping contains the same key twice,
# but I know what I'm doing:
key: value 1
key: value 2  # yamllint disable-line rule:key-duplicates

- This line is waaaaaaaaaay too long but yamllint will not report anything about it.
 ↪# yamllint disable-line rule:line-length
  This line will be checked by yamllint.
```

or:

```
# The following mapping contains the same key twice,
# but I know what I'm doing:
key: value 1
# yamllint disable-line rule:key-duplicates
key: value 2

# yamllint disable-line rule:line-length
- This line is waaaaaaaaaay too long but yamllint will not report anything about it.
  This line will be checked by yamllint.
```

It is possible, although not recommend, to disabled **all** rules for a specific line:

```
# yamllint disable-line
-   {    all : rules ,are disabled   for this line}
```

If you need to disable multiple rules, it is allowed to chain rules like this: `# yamllint disable-line rule:hyphens rule:commas rule:indentation`.

2.4.2 Disabling checks for all (or part of) the file

To prevent yamllint from reporting problems for the whole file, or for a block of lines within the file, use `# yamllint disable ...` and `# yamllint enable ...` directive comments. For instance:

```
# yamllint disable rule:colons
- Lorem       : ipsum
  dolor       : sit amet,
  consectetur : adipiscing elit
# yamllint enable rule:colons

- rest of the document...
```

It is possible, although not recommend, to disabled **all** rules:

```
# yamllint disable
- Lorem       :
      ipsum:
        dolor : [   sit,amet]
```

```
-         consectetur : adipiscing elit
# yamllint enable
```

If you need to disable multiple rules, it is allowed to chain rules like this: `# yamllint disable rule:hyphens rule:commas rule:indentation`.

2.4.3 Disabling all checks for a file

To prevent yamllint from reporting problems for a specific file, add the directive comment `# yamllint disable-file` as the first line of the file. For instance:

```
# yamllint disable-file
# The following mapping contains the same key twice, but I know what I'm doing:
key: value 1
key: value 2

- This line is waaaaaaaaaay too long but yamllint will not report anything about it.
  This line will be checked by yamllint.
```

or:

```
# yamllint disable-file
# This file is not valid YAML because it is a Jinja template
{% if extra_info %}
key1: value1
{% endif %}
key2: value2
```

2.5 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*, *filepath=None*)
    Lints a YAML source.

    Returns a generator of LintProblem objects.

    Parameters

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

- **conf** – yamllint configuration object

2.6 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.

2.6.1 Vim

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

If you instead use the syntastic plugin, add this to your `.vimrc`:

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

2.6.2 Neovim

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

2.6.3 Emacs

If you are flycheck user, you can use flycheck-yamllint integration.

2.6.4 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).

2.7 Integration with other software

2.7.1 Integration with pre-commit

You can integrate yamllint in pre-commit tool. Here is an example, to add in your .pre-commit-config.yaml

```
---
# Update the rev variable with the release version that you want, from the yamllint␣
↪repo
# You can pass your custom .yamllint with args attribute.
- repo: https://github.com/adrienverge/yamllint.git
  rev: v1.17.0
  hooks:
    - id: yamllint
      args: [-c=/path/to/.yamllint]
```

Python Module Index

y