YAML

- YAML Formatierung mehrzeiliger Strings

YAML Formatierung mehrzeiliger Strings

Quelle: https://stackoverflow.com/a/21699210

There are ~~5~~ ~~6~~ ***NINE*** (or 63*, depending how you count) different ways to write multi-line strings in YAML.

TL;DR

- Use `>` most of the time: interior line breaks are stripped out, although you get one at the end:

```
key: >
  Your long
  string here.
```

- Use `|` if you want those linebreaks to be preserved as `\n` (for instance, embedded markdown with paragraphs).

```
key: |
  ### Heading

  * Bullet
  * Points
```

- Use `>-` or `|-` instead if you don't want a linebreak appended at the end.
- Use `"..."` if you need to split lines in the middle of words or want to literally type linebreaks as `\n`:

```
key: "Antidisestab\
 lishmentarianism.\n\nGet on it."
```

- YAML is crazy.

Block scalar styles (`>`, `|`)

These allow characters such as `\` and `"` without escaping, and add a new line (`\n`) to the end of your string.

`>` *Folded style* removes single newlines within the string (but adds one at the end, and converts double newlines to singles):

```
Key: >
  this is my very very very
  long string
```

? `this is my very very very long string\n`

`|` *Literal style* turns every newline within the string into a literal newline, and adds one at the end:

```
Key: |
  this is my very very very
  long string
```

? `this is my very very very\nlong string\n`

Here's the official definition from the YAML Spec 1.2

> Scalar content can be written in block notation, using a literal style (indicated by “|”) where all line breaks are significant. Alternatively, they can be written with the folded style (denoted by “>”) where each line break is folded to a space unless it ends an empty or a more-indented line.

Block styles with block chomping indicator (`>-`, `|-`, `>+`, `|+`)

You can control the handling of the final new line in the string, and any trailing blank lines (`\n\n`) by adding a block chomping indicator character:

- `>`, `|`: "clip": keep the line feed, remove the trailing blank lines.
- `>-`, `|-`: "strip": remove the line feed, remove the trailing blank lines.
- `>+`, `|+`: "keep": keep the line feed, keep trailing blank lines.

"Flow" scalar styles (, `"`, `'`)

These have limited escaping, and construct a single-line string with no new line characters. They can begin on the same line as the key, or with additional newlines first.

plain style (no escaping, no `#` or `:` combinations, limits on first character):

```
Key: this is my very very very
  long string
```

double-quoted style (`\` and `"` must be escaped by `\`, newlines can be inserted with a literal `\n` sequence, lines can be concatenated without spaces with trailing `\`):

```
Key: "this is my very very \"very\" loooo\
  ng string.\n\nLove, YAML."
```

? `"this is my very very \"very\" loooong string.\n\nLove, YAML."`

single-quoted style (literal `'` must be doubled, no special characters, possibly useful for expressing strings starting with double quotes):

```
Key: 'this is my very very "very"
  long string, isn''t it.'
```

? `"this is my very very \"very\" long string, isn't it."`

Summary

In this table, `_` means `space character`. `\n` means "newline character" (`\n` in JavaScript), except for the "in-line newlines" row, where it means literally a backslash and an n).

```
                      >     |            "     '     >-     >+     |-     |+
-------------------------|------|-----|-----|-----|------|------|------|------
Trailing spaces   | Kept | Kept |     |     |     | Kept | Kept | Kept | Kept
Single newline => | _    | \n   | _   | _   | _   | _    |  _   | \n   | \n
Double newline => | \n   | \n\n | \n  | \n  | \n  | \n   |  \n  | \n\n | \n\n
Final newline  => | \n   | \n   |     |     |     |      |  \n  |      | \n
Final dbl nl's => |      |      |     |     |     |      | Kept |      | Kept
In-line newlines  | No   | No   | No  | \n  | No  | No   | No   | No   | No
Spaceless newlines| No   | No   | No  | \   | No  | No   | No   | No   | No
Single quote      | '    | '    | '   | '   | ''  | '    | '    | '    | '
Double quote      | "    | "    | "   | \"  | "   | "    | "    | "    | "
Backslash         | \    | \    | \   | \\  | \   | \    | \    | \    | \
" #", ": "        | Ok   | Ok   | No  | Ok  | Ok  | Ok   | Ok   | Ok   | Ok
Can start on same | No   | No   | Yes | Yes | Yes | No   | No   | No   | No
line as key       |
```

Examples

Note the trailing spaces on the line before "spaces."

```
- >
  very "long"
  'string' with

  paragraph gap, \n and
  spaces.
- |
  very "long"
  'string' with

  paragraph gap, \n and
  spaces.
- very "long"
  'string' with

  paragraph gap, \n and
  spaces.
- "very \"long\"
  'string' with

  paragraph gap, \n and
  s\
  p\
  a\
  c\
  e\
  s."
- 'very "long"
  ''string'' with

  paragraph gap, \n and
  spaces.'
- >-
  very "long"
  'string' with

  paragraph gap, \n and
  spaces.
```

```
[
  "very \"long\" 'string' with\nparagraph gap, \\n and         spaces.\n",
  "very \"long\"\n'string' with\n\nparagraph gap, \\n and        \nspaces.\n",
  "very \"long\" 'string' with\nparagraph gap, \\n and spaces.",
  "very \"long\" 'string' with\nparagraph gap, \n and spaces.",
  "very \"long\" 'string' with\nparagraph gap, \\n and spaces.",
  "very \"long\" 'string' with\nparagraph gap, \\n and         spaces."
]
```

Block styles with indentation indicators

Just in case the above isn't enough for you, you can add a "block indentation indicator" (after your block chomping indicator, if you have one):

```
- >8
        My long string
        starts over here
- |+1
 This one
 starts here
```

Addendum

If you insert extra spaces at the start of not-the-first lines in Folded style, they will be kept, with a bonus newline. This doesn't happen with flow styles:

```
- >
    my long
      string
- my long
    string
```

? `["my long\n  string\n", "my long string"]`

I can't even.

`*` *2 block styles, each with 2 possible block chomping indicators (or none), and with 9 possible indentation indicators (or none), 1 plain style and 2 quoted styles: 2 x (2 + 1) x (9 + 1) + 1 + 2 = 63*

Some of this information has also been summarised here.