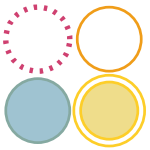
latexindent.pl

Version 4.0



Chris Hughes *

2026-03-15

`latexindent.pl` is a `Perl` script that indents `.tex` (and other) files according to an indentation scheme that the user can modify to suit their taste. Environments, including those with alignment delimiters (such as `tabular`), and commands, including those that can split braces and brackets across lines, are *usually* handled correctly by the script. Options for `verbatim`-like environments and commands, together with indentation after headings (such as `chapter`, `section`, etc) are also available. The script also has the ability to modify line breaks, and to add comment symbols and blank lines; furthermore, it permits string or regex-based substitutions. All user options are customisable via the switches and the YAML interface.

tl;dr, a quick start guide is given in Section 1.2 on page 5.

Contents

*and contributors! See Section 9.5 on page 143. For all communication, please visit [36].

SECTION 1

Introduction

1.1 License

`latexindent.pl` is free and open source, and it always will be; it is released under the GNU General Public License v3.0.

Before you start using it on any important files, bear in mind that `latexindent.pl` has the option to overwrite your `.tex` files. It will always make at least one backup (you can choose how many it makes, see page 22) but you should still be careful when using it. The script has been tested on many files, but there are some known limitations (see Section 8). You, the user, are responsible for ensuring that you maintain backups of your files before running `latexindent.pl` on them. I think it is important at this stage to restate an important part of the license here:

> *This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.*

There is certainly no malicious intent in releasing this script, and I do hope that it works as you expect it to; if it does not, please first of all make sure that you have the correct settings, and then feel free to let me know at [36] with a complete minimum working example as I would like to improve the code as much as possible.

Warning!

Before you try the script on anything important (like your thesis), test it out on the sample files in the `test-case` directory [36].

1.2 Quick start

When `latexindent.pl` reads and writes files, the files are read and written in UTF-8 format by default. That is to say, the encoding format for tex and yaml files needs to be in UTF-8 format.

If you'd like to get started with `latexindent.pl` then simply type

```
cmh:~$ latexindent.pl myfile.tex
```

from the command line.

We give an introduction to `latexindent.pl` using Listing 1; there is no explanation in this section, which is deliberate for a quick start. The rest of the manual is more verbose.

LISTING 1: quick-start.tex

```
\documentclass{article}
\usepackage[
inner=2.5cm,
]{geometry}
\begin{document}
A quick start
    demonstration for latexindent.pl.
    \begin{myenv}
      The body of environments and
      other code blocks
        receive indentation.
    \end{myenv}
\end{document}
```

Running

```
cmh:~$ latexindent.pl quick-start.tex
```

gives Listing 2.

LISTING 2: quick-start-default.tex

```
\documentclass{article}
\usepackage[
inner=2.5cm,
]{geometry}
\begin{document}
A quick start
demonstration for latexindent.pl.
\begin{myenv}
   The body of environments and
   other code blocks
   receive indentation.
\end{myenv}
\end{document}
```

example 1 Running

```
cmh:~$ latexindent.pl -l quick-start1.yaml quick-start.tex
```

gives Listing 3.

LISTING 3: quick-start-mod1.tex

```
\documentclass{article}
\usepackage[
inner=2.5cm,
]{geometry}
\begin{document}
A␣quick␣start
demonstration␣for␣latexindent.pl.
\begin{myenv}
␣The␣body␣of␣environments␣and
␣other␣code␣blocks
␣receive␣indentation.
\end{myenv}
\end{document}
```

LISTING 4: quick-start1.yaml

```
defaultIndent: " "
```

See Section 3.4. ■

example 2 Running

```
cmh:~$ latexindent.pl -l quick-start2.yaml quick-start.tex
```

gives Listing 5.

LISTING 5: quick-start-mod2.tex

```
\documentclass{article}
\usepackage[
inner=2.5cm,
]{geometry}
\begin{document}
A␣quick␣start
demonstration␣for␣latexindent.pl.
\begin{myenv}
␣␣␣The␣body␣of␣environments␣and
␣␣␣other␣code␣blocks
␣␣␣receive␣indentation.
\end{myenv}
\end{document}
```

LISTING 6: quick-start2.yaml

```
indentRules:
  myenv: "    "
```

See Section 3.9. ■

example 3 Running

```
cmh:~$ latexindent.pl -l quick-start3.yaml quick-start.tex
```

gives Listing 7.

LISTING 7: quick-start-mod3.tex

```
\documentclass{article}
\usepackage[
inner=2.5cm,
]{geometry}
\begin{document}
A quick start
demonstration for latexindent.pl.
\begin{myenv}
The body of environments and
other code blocks
receive indentation.
\end{myenv}
\end{document}
```

LISTING 8: quick-start3.yaml

```
noAdditionalIndent:
  myenv: 1
```

See Section 3.9. ■

example 4 Running

```
cmh:~$ latexindent.pl -m -l quick-start4.yaml quick-start.tex
```

gives Listing 9. ■

LISTING 9: quick-start-mod4.tex

```
\documentclass{article}
\usepackage[
inner=2.5cm,
]{geometry}
\begin{document}
A quick start
demonstration for latexindent.pl.
\begin{myenv}
   The body of environments and
   other code blocks
   receive indentation.
\end{myenv}
\end{document}
```

LISTING 10: quick-start4.yaml -m

```
modifyLineBreaks:
    textWrapOptions:
        columns: 20
```

Full details of text wrapping in Section 4.1. ■

example 5 Running

```
cmh:~$ latexindent.pl -m -l quick-start5.yaml quick-start.tex
```

gives Listing 11.

LISTING 11: quick-start-mod5.tex

```
\documentclass{article}
\usepackage[
inner=2.5cm,
]{geometry}
\begin{document}
A quick start
demonstration for latexindent.pl.
\begin{myenv}
   The body of
   environments and
   other code blocks
   receive
   indentation.
\end{myenv}
\end{document}
```

LISTING 12: quick-start5.yaml -m

```
modifyLineBreaks:
    textWrapOptions:
        columns: 20
        blocksFollow:
           other: '\\begin\{myenv\}'
```

Full details of text wrapping in Section 4.1. ■

example 6 Running

```
cmh:~$ latexindent.pl -m -l quick-start6.yaml quick-start.tex
```

gives Listing 13. ■

LISTING 13: quick-start-mod6.tex

```
\documentclass{article}
\usepackage[
inner=2.5cm,
]{geometry}\begin{document}
A quick start
demonstration for
    latexindent.pl.\begin{myenv}
   The body of environments and
   other code blocks
   receive indentation.
\end{myenv}
\end{document}
```

LISTING 14: quick-start6.yaml -m

```
modifyLineBreaks:
    environments:
        BeginStartsOnOwnLine: -1
```

This is an example of a *poly-switch*; full details of *poly-switches* are covered in Section 4.3. ■

example 7 Running

```
cmh:~$ latexindent.pl -m -l quick-start7.yaml quick-start.tex
```

gives Listing 15.

LISTING 15: quick-start-mod7.tex

```
\documentclass{article}
\usepackage[
inner=2.5cm,
]{geometry}
\begin{document}
A quick start
demonstration for latexindent.pl.
\begin{myenv}
   The body of environments and
   other code blocks
   receive
    indentation.\end{myenv}\end{document}
```

LISTING 16: quick-start7.yaml -m

```
modifyLineBreaks:
    environments:
        EndStartsOnOwnLine: -1
```

Full details of *poly-switches* are covered in Section 4.3. ■

example 8 Running

```
cmh:~$ latexindent.pl -l quick-start8.yaml quick-start.tex
```

gives Listing 17; note that the *preamble* has been indented. ■

LISTING 17: quick-start-mod8.tex

```
\documentclass{article}
\usepackage[
   inner=2.5cm,
]{geometry}
\begin{document}
A quick start
demonstration for latexindent.pl.
\begin{myenv}
   The body of environments and
   other code blocks
   receive indentation.
\end{myenv}
\end{document}
```

LISTING 18: quick-start8.yaml

```
indentPreamble: 1
```

See Section 3.3. ■

example 9 Running

```
cmh:~$ latexindent.pl -l quick-start9.yaml quick-start.tex
```

gives Listing 19.

LISTING 19: quick-start-mod9.tex

```
\documentclass{article}
\usepackage[
inner=2.5cm,
]{geometry}
\begin{document}
   A quick start
   demonstration for latexindent.pl.
   \begin{myenv}
      The body of environments and
      other code blocks
      receive indentation.
   \end{myenv}
\end{document}
```

LISTING 20: quick-start9.yaml

```
noAdditionalIndent:
  document: 0
```

See Section 3.9. ■

1.3 Requirements

1.3.1 Perl users

N: 2018-01-13

Perl users will need a few standard Perl modules – see Section A on page 145 for details; in particular, note that a module installer helper script is shipped with `latexindent.pl`. You might also like to see https://stackoverflow.com/questions/19590042/error-cant-locate-file-homedir-pm-in-inc.

1.3.2 Windows users without perl

`latexindent.pl` ships with `latexindent.exe` for Windows users, so that you can use the script with or without a Perl distribution.

`latexindent.exe` is available from [36].

MiKTeX users on Windows may like to see [39] for details of how to use `latexindent.exe` without a Perl installation.

1.3.3 Ubuntu Linux users without perl

`latexindent.pl` ships with `latexindent-linux` for Ubuntu Linux users, so that you can use the script with or without a Perl distribution.

N: 2022-10-30

`latexindent-linux` is available from [36].

1.3.4 macOS users without perl

`latexindent.pl` ships with `latexindent-macos` for macOS users, so that you can use the script with or without a Perl distribution.

N: 2022-10-30

`latexindent-macOS` is available from [36].

1.3.5 conda users

Users of `conda` should see the details given in Section E.

1.3.6 docker users

Users of `docker` should see the details given in Section F.

1.4 About this documentation

As you read through this documentation, you will see many listings; in this version of the documentation, there are a total of 596. This may seem a lot, but I deem it necessary in presenting the various different options of `latexindent.pl` and the associated output that they are capable of producing.

The different listings are presented using different styles:

LISTING 21: demo-tex.tex

```
demonstration .tex file
```

This type of listing is a `.tex` file.

LISTING 22: fileExtensionPreference

```
31 fileExtensionPreference:
32   .tex: 1
33   .sty: 2
34   .cls: 3
35   .bib: 4
```

This type of listing is a `.yaml` file; when you see line numbers given (as here) it means that the snippet is taken directly from `defaultSettings.yaml`, discussed in detail in Section 3 on page 21.

LISTING 23: modifyLineBreaks -m

```
289 modifyLineBreaks:
290   preserveBlankLines: 1                # 0/1
291   condenseMultipleBlankLinesInto: 1    # 0/1
```

This type of listing is a `.yaml` file, but it will only be relevant when the `-m` switch is active; see Section 4 on page 70 for more details.

LISTING 24: replacements -r

```
419 replacements:
420   - amalgamate: 1
421   - this: latexindent.pl
422     that: pl.latexindent
423     lookForThis: 0
424     when: before
```

This type of listing is a `.yaml` file, but it will only be relevant when the `-r` switch is active; see Section 5 on page 119 for more details.

N: 2017-06-25

You will occasionally see dates shown in the margin (for example, next to this paragraph!) which detail the date of the version in which the feature was implemented; the 'N' stands for 'new as of the date shown' and 'U' stands for 'updated as of the date shown'. If you see ✨, it means that the feature is either new (N) or updated (U) as of the release of the current version; if you see ✨ attached to a listing, then it means that listing is new (N) or updated (U) as of the current version. If you have not read this document before (and even if you have!), then you can ignore every occurrence of the ✨; they are simply there to highlight new and updated features. The new and updated features in this documentation (V4.0) are on the following pages:

1.5 A word about regular expressions

As you read this documentation, you may encounter the term *regular expressions*. I've tried to write this documentation in such a way so as to allow you to engage with them or not, as you prefer. This documentation is not designed to be a guide to regular expressions, and if you'd like to read about them, I recommend [35].

SECTION 2

How to use the script

`latexindent.pl` ships as part of the TEXLive distribution for Linux and Mac users; `latexindent.exe` ships as part of the TEXLive for Windows users. These files are also available from github [36] should you wish to use them without a TEX distribution; in this case, you may like to read Section B on page 149 which details how the `path` variable can be updated.

In what follows, we will always refer to `latexindent.pl`, but depending on your operating system and preference, you might substitute `latexindent.exe` or simply `latexindent`.

There are two ways to use `latexindent.pl`: from the command line, and using `arara`; we discuss these in Section 2.1 and Section 2.2 respectively. We will discuss how to change the settings and behaviour of the script in Section 3 on page 21.

2.1 From the command line

`latexindent.pl` has a number of different switches/flags/options, which can be combined in any way that you like, either in short or long form as detailed below. `latexindent.pl` produces a `.log` file, `indent.log`, every time it is run; the name of the log file can be customised, but we will refer to the log file as `indent.log` throughout this document. There is a base of information that is written to `indent.log`, but other additional information will be written depending on which of the following options are used.

When using `latexindent.pl` in different ways on different systems, the range of characters supported by its switches/flags/options may vary. We discuss these in Section Section K.

N: 2017-06-25

`-v, -version`

```
cmh:~$ latexindent.pl -v
cmh:~$ latexindent.pl --version
```

This will output only the version number to the terminal.

N: 2022-01-08

`-vv, -vversion`

```
cmh:~$ latexindent.pl -vv
cmh:~$ latexindent.pl --vversion
```

This will output *verbose* version details to the terminal, including the location of `latexindent.pl` and `defaultSettings.yaml`.

`-h, -help`

```
cmh:~$ latexindent.pl -h
cmh:~$ latexindent.pl --help
```

As above this will output a welcome message to the terminal, including the version number and available options.

```
cmh:~$ latexindent.pl myfile.tex
```

This will operate on `myfile.tex`, but will simply output to your terminal; `myfile.tex` will not be changed by `latexindent.pl` in any way using this command.

N: 2022-03-25

You can instruct `latexindent.pl` to operate on multiple (batches) of files, for example

```
cmh:~$ latexindent.pl myfile1.tex myfile2.tex
```

Full details are given in Section C on page 151.

`-w, -overwrite`

```
cmh:~$ latexindent.pl -w myfile.tex
cmh:~$ latexindent.pl --overwrite myfile.tex
cmh:~$ latexindent.pl myfile.tex --overwrite
```

This *will* overwrite `myfile.tex`, but it will make a copy of `myfile.tex` first. You can control the name of the extension (default is `.bak`), and how many different backups are made – more on this in Section 3, and in particular see `backupExtension` and `onlyOneBackUp`.

Note that if `latexindent.pl` can not create the backup, then it will exit without touching your original file; an error message will be given asking you to check the permissions of the backup file.

N: 2022-03-25

`-wd, -overwriteIfDifferent`

```
cmh:~$ latexindent.pl -wd myfile.tex
cmh:~$ latexindent.pl --overwriteIfDifferent myfile.tex
cmh:~$ latexindent.pl myfile.tex --overwriteIfDifferent
```

This *will* overwrite `myfile.tex` but only *if the indented text is different from the original*. If the indented text is *not* different from the original, then `myfile.tex` will *not* be overwritten.

All other details from the `-w` switch are relevant here. If you call `latexindent.pl` with both the `-wd` and the `-w` switch, then the `-w` switch will be deactivated and the `-wd` switch takes priority.

`-o=output.tex,-outputfile=output.tex`

```
cmh:~$ latexindent.pl -o=output.tex myfile.tex
cmh:~$ latexindent.pl myfile.tex -o=output.tex
cmh:~$ latexindent.pl --outputfile=output.tex myfile.tex
cmh:~$ latexindent.pl --outputfile output.tex myfile.tex
```

This will indent `myfile.tex` and output it to `output.tex`, overwriting it (`output.tex`) if it already exists[1].

Note that if `latexindent.pl` is called with both the `-w` and `-o` switches, then `-w` will be ignored and `-o` will take priority (this seems safer than the other way round). The same is true for the `-wd` switch, and the `-o` switch takes priority over it.

Note that using `-o` as above is equivalent to using

```
cmh:~$ latexindent.pl myfile.tex > output.tex
```

N: 2017-06-25

You can call the `-o` switch with the name of the output file *without* an extension; in this case, `latexindent.pl` will use the extension from the original file. For example, the following two calls to `latexindent.pl` are equivalent:

[1] Users of version 2.* should note the subtle change in syntax

```
cmh:~$ latexindent.pl myfile.tex -o=output
cmh:~$ latexindent.pl myfile.tex -o=output.tex
```

N: 2017-06-25

You can call the `-o` switch using a `+` symbol at the beginning; this will concatenate the name of the input file and the text given to the `-o` switch. For example, the following two calls to `latexindent.pl` are equivalent:

```
cmh:~$ latexindent.pl myfile.tex -o=+new
cmh:~$ latexindent.pl myfile.tex -o=myfilenew.tex
```

N: 2017-06-25

You can call the `-o` switch using a `++` symbol at the end of the name of your output file; this tells `latexindent.pl` to search successively for the name of your output file concatenated with $0, 1, \ldots$ while the name of the output file exists. For example,

```
cmh:~$ latexindent.pl myfile.tex -o=output++
```

tells `latexindent.pl` to output to `output0.tex`, but if it exists then output to `output1.tex`, and so on.

Calling `latexindent.pl` with simply

```
cmh:~$ latexindent.pl myfile.tex -o=++
```

tells it to output to `myfile0.tex`, but if it exists then output to `myfile1.tex` and so on.

The `+` and `++` feature of the `-o` switch can be combined; for example, calling

```
cmh:~$ latexindent.pl myfile.tex -o=+out++
```

tells `latexindent.pl` to output to `myfileout0.tex`, but if it exists, then try `myfileout1.tex`, and so on.

There is no need to specify a file extension when using the `++` feature, but if you wish to, then you should include it *after* the `++` symbols, for example

```
cmh:~$ latexindent.pl myfile.tex -o=+out++.tex
```

`-s, -silent`

```
cmh:~$ latexindent.pl -s myfile.tex
cmh:~$ latexindent.pl myfile.tex -s
```

Silent mode: no output will be given to the terminal.

`-t, -trace`

```
cmh:~$ latexindent.pl -t myfile.tex
cmh:~$ latexindent.pl myfile.tex -t
```

Tracing mode: verbose output will be given to `indent.log`. This is useful if `latexindent.pl` has made a mistake and you're trying to find out where and why. You might also be interested in learning about `latexindent.pl`'s thought process – if so, this switch is for you, although it should be noted that, especially for large files, this does affect performance of the script.

`-tt, -ttrace`

```
cmh:~$ latexindent.pl -tt myfile.tex
cmh:~$ latexindent.pl myfile.tex -tt
```

More detailed tracing mode: this option gives more details to `indent.log` than the standard `trace` option (note that, even more so than with `-t`, especially for large files, performance of the script will be affected).

`-l, -local[=myyaml.yaml,other.yaml,...]`

```
cmh:~$ latexindent.pl -l myfile.tex
cmh:~$ latexindent.pl -l=myyaml.yaml myfile.tex
cmh:~$ latexindent.pl -l myyaml.yaml myfile.tex
cmh:~$ latexindent.pl -l first.yaml,second.yaml,third.yaml myfile.tex
cmh:~$ latexindent.pl -l=first.yaml,second.yaml,third.yaml myfile.tex
cmh:~$ latexindent.pl myfile.tex -l=first.yaml,second.yaml,third.yaml
```

`latexindent.pl` will always load `defaultSettings.yaml` (rhymes with camel) and if it is called with the `-l` switch and it finds `localSettings.yaml` in the same directory as `myfile.tex`, then, if not found, it looks for `localSettings.yaml` (and friends, see Section H.2 on page 161) in the current working directory, then these settings will be added to the indentation scheme. Information will be given in `indent.log` on the success or failure of loading `localSettings.yaml`.

U: 2021-03-14

The `-l` flag can take an *optional* parameter which details the name (or names separated by commas) of a YAML file(s) that resides in the same directory as `myfile.tex`; you can use this option if you would like to load a settings file in the current working directory that is *not* called `localSettings.yaml`. In fact, you can specify both *relative* and *absolute paths* for your YAML files; for example

U: 2017-08-21

```
cmh:~$ latexindent.pl -l=../../myyaml.yaml myfile.tex
cmh:~$ latexindent.pl -l=/home/cmhughes/Desktop/myyaml.yaml myfile.tex
cmh:~$ latexindent.pl -l=C:\Users\cmhughes\Desktop\myyaml.yaml myfile.tex
```

You will find a lot of other explicit demonstrations of how to use the `-l` switch throughout this documentation,

N: 2017-06-25

You can call the `-l` switch with a '+' symbol either before or after another YAML file; for example:

```
cmh:~$ latexindent.pl -l=+myyaml.yaml myfile.tex
cmh:~$ latexindent.pl -l "+ myyaml.yaml" myfile.tex
cmh:~$ latexindent.pl -l=myyaml.yaml+  myfile.tex
```

which translate, respectively, to

```
cmh:~$ latexindent.pl -l=localSettings.yaml,myyaml.yaml myfile.tex
cmh:~$ latexindent.pl -l=localSettings.yaml,myyaml.yaml myfile.tex
cmh:~$ latexindent.pl -l=myyaml.yaml,localSettings.yaml myfile.tex
```

Note that the following is *not* allowed:

```
cmh:~$ latexindent.pl -l+myyaml.yaml myfile.tex
```

and

```
cmh:~$ latexindent.pl -l + myyaml.yaml myfile.tex
```

will *only* load `localSettings.yaml`, and `myyaml.yaml` will be ignored. If you wish to use spaces between any of the YAML settings, then you must wrap the entire list of YAML files in quotes, as demonstrated above.

N: 2017-06-25

You may also choose to omit the `yaml` extension, such as

```
cmh:~$ latexindent.pl -l=localSettings,myyaml myfile.tex
```

`-y, -yaml=yaml settings`

```
cmh:~$ latexindent.pl myfile.tex -y="defaultIndent:␣'␣'"
cmh:~$ latexindent.pl myfile.tex -y="defaultIndent:␣'␣',maximumIndentation:'␣'"
cmh:~$ latexindent.pl myfile.tex -y="indentRules:␣one:␣'\t\t\t\t'"
cmh:~$ latexindent.pl myfile.tex
    -y='modifyLineBreaks:environments:EndStartsOnOwnLine:3' -m
cmh:~$ latexindent.pl myfile.tex
    -y='modifyLineBreaks:environments:one:EndStartsOnOwnLine:3' -m
```

N: 2017-08-21

You can specify YAML settings from the command line using the `-y` or `-yaml` switch; sample demonstrations are given above. Note, in particular, that multiple settings can be specified by separating them via commas. There is a further option to use a `;` to separate fields, which is demonstrated in Section H.3 on page 162.

Any settings specified via this switch will be loaded *after* any specified using the `-l` switch. This is discussed further in Section H.4 on page 162.

`-d, -onlydefault`

```
cmh:~$ latexindent.pl -d myfile.tex
```

Only `defaultSettings.yaml`: you might like to read Section 3 before using this switch. By default, `latexindent.pl` will always search for `indentconfig.yaml` or `.indentconfig.yaml` in your home directory. If you would prefer it not to do so then (instead of deleting or renaming `indentconfig.yaml` or `.indentconfig.yaml`) you can simply call the script with the `-d` switch; note that this will also tell the script to ignore `localSettings.yaml` even if it has been called with the `-l` switch; `latexindent.pl` will also ignore any settings specified from the `-y` switch.

U: 2017-08-21

`-c, -cruft=<directory>`

```
cmh:~$ latexindent.pl -c=/path/to/directory/ myfile.tex
```

If you wish to have backup files and `indent.log` written to a directory other than the current working directory, then you can send these 'cruft' files to another directory. Note the use of a trailing forward slash.

If the cruft directory does not exist, `latexindent.pl` will attempt to create it.

`-g, -logfile=<name of log file>`

```
cmh:~$ latexindent.pl -g=other.log myfile.tex
cmh:~$ latexindent.pl -g other.log myfile.tex
cmh:~$ latexindent.pl --logfile other.log myfile.tex
cmh:~$ latexindent.pl myfile.tex -g other.log
```

By default, `latexindent.pl` reports information to `indent.log`, but if you wish to change the name of this file, simply call the script with your chosen name after the `-g` switch as demonstrated above.

N: 2021-05-07

If `latexindent.pl` can not open the log file that you specify, then the script will operate, and no log file will be produced; this might be helpful to users who wish to specify the following, for example

```
cmh:~$ latexindent.pl -g /dev/null myfile.tex
```

`-sl, -screenlog`

```
cmh:~$ latexindent.pl -sl myfile.tex
cmh:~$ latexindent.pl -screenlog myfile.tex
```

N: 2018-01-13

Using this option tells `latexindent.pl` to output the log file to the screen, as well as to your chosen log file.

`-m, -modifylinebreaks`

```
cmh:~$ latexindent.pl -m myfile.tex
cmh:~$ latexindent.pl -modifylinebreaks myfile.tex
```

One of the most exciting developments in Version 3.0 is the ability to modify line breaks; for full details see Section 4 on page 70

`latexindent.pl` can also be called on a file without the file extension, for example

```
cmh:~$ latexindent.pl myfile
```

and in which case, you can specify the order in which extensions are searched for; see Listing 26 on page 21 for full details.

`STDIN`

```
cmh:~$ cat myfile.tex | latexindent.pl
cmh:~$ cat myfile.tex | latexindent.pl -
```

N: 2018-01-13

`latexindent.pl` will allow input from STDIN, which means that you can pipe output from other commands directly into the script. For example assuming that you have content in `myfile.tex`, then the above command will output the results of operating upon `myfile.tex`.

If you wish to use this feature with your own local settings, via the `-l` switch, then you should finish your call to `latexindent.pl` with a `-` sign:

```
cmh:~$ cat myfile.tex | latexindent.pl -l=mysettings.yaml -
```

U: 2018-01-13

Similarly, if you simply type `latexindent.pl` at the command line, then it will expect (STDIN) input from the command line.

```
cmh:~$ latexindent.pl
```

Once you have finished typing your input, you can press

- `CTRL+D` on Linux
- `CTRL+Z` followed by `ENTER` on Windows

to signify that your input has finished. Thanks to [9] for an update to this feature.

`-r, -replacement`

```
cmh:~$ latexindent.pl -r myfile.tex
cmh:~$ latexindent.pl -replacement myfile.tex
```

N: 2019-07-13

You can call `latexindent.pl` with the `-r` switch to instruct it to perform replacements/substitutions on your file; full details and examples are given in Section 5 on page 119.

`-rv, -replacementrespectverb`

```
cmh:~$ latexindent.pl -rv myfile.tex
cmh:~$ latexindent.pl -replacementrespectverb myfile.tex
```

N: 2019-07-13

You can instruct `latexindent.pl` to perform replacements/substitutions by using the `-rv` switch, but will *respect verbatim code blocks*; full details and examples are given in Section 5 on page 119.

`-rr, -onlyreplacement`

```
cmh:~$ latexindent.pl -rr myfile.tex
cmh:~$ latexindent.pl -onlyreplacement myfile.tex
```

N: 2019-07-13

You can instruct `latexindent.pl` to skip all of its other indentation operations and *only* perform replacements/substitutions by using the `-rr` switch; full details and examples are given in Section 5 on page 119.

`-k, -check`

```
cmh:~$ latexindent.pl -k myfile.tex
cmh:~$ latexindent.pl -check myfile.tex
```

N: 2021-09-16

You can instruct `latexindent.pl` to check if the text after indentation matches that given in the original file.

The `exit` code of `latexindent.pl` is 0 by default. If you use the `-k` switch then

- if the text after indentation matches that given in the original file, then the exit code is 0;
- if the text after indentation does *not* match that given in the original file, then the exit code is 1.

The value of the exit code may be important to those wishing to, for example, check the status of the indentation in continuous integration tools such as GitHub Actions. Full details of the exit codes of `latexindent.pl` are given in Table 1.

A simple `diff` will be given in `indent.log`.

`-kv, -checkv`

```
cmh:~$ latexindent.pl -kv myfile.tex
cmh:~$ latexindent.pl -checkv myfile.tex
```

N: 2021-09-16

The `check verbose` switch is exactly the same as the `-k` switch, except that it is *verbose*, and it will output the (simple) diff to the terminal, as well as to `indent.log`.

`-n, -lines=MIN-MAX`

```
cmh:~$ latexindent.pl -n 5-8 myfile.tex
cmh:~$ latexindent.pl -lines 5-8 myfile.tex
```

N: 2021-09-16

The `lines` switch instructs `latexindent.pl` to operate only on specific line ranges within `myfile.tex`.

Complete demonstrations are given in Section 6.

`-GCString`

```
cmh:~$ latexindent.pl --GCString myfile.tex
```

N: 2022-03-25

instructs `latexindent.pl` to load the `Unicode::GCString` module. This should only be necessary if you find that the alignment at ampersand routine (described in Section 3.5) does not work for your language. Further details are given in Section A.3.

2.2 From arara

Using `latexindent.pl` from the command line is fine for some folks, but others may find it easier to use from `arara`; you can find the arara rule for `latexindent.pl` and its associated documentation at [1].

2.3 switches via yaml

N: 2026-03-15

For some of the switches, you can specify them by the YAML interface; the defaults are shown in Listing 25.

Note: switches can only be turned *on* using this field, they will not be turned *off*.

LISTING 25: `switchesViaYaml`

```
switchesViaYaml:
  kSwitch:  0 # -k,  check mode
  kvSwitch: 0 # -kv, verbose check mode
  mSwitch:  0 # -m,  modify line breaks
  rSwitch:  0 # -r,  replacement
  rrSwitch: 0 # -rr, only replacements
  rvSwitch: 0 # -rv, replacement respect verbatim
  sSwitch:  0 # -s,  silent mode
  slSwitch: 0 # -sl, screen log
  tSwitch:  0 # -t,  trace mode
  ttSwitch: 0 # -tt, ttrace mode
```

2.4 Summary of exit codes

Assuming that you call `latexindent.pl` on `myfile.tex`

```
cmh:~$ latexindent.pl myfile.tex
```

then `latexindent.pl` can exit with the exit codes given in Table 1.

TABLE 1: Exit codes for `latexindent.pl`

exit code	indentation	status
0	✔	success; if `-k` or `-kv` active, indented text matches original
0	✖	success; if `-version`, `-vversion` or `-help`, no indentation performed
1	✔	success, and `-k` or `-kv` active; indented text *different* from original
2	✖	failure, `defaultSettings.yaml` could not be read
3	✖	failure, myfile.tex not found
4	✖	failure, myfile.tex exists but cannot be read
5	✖	failure, `-w` active, and back-up file cannot be written
6	✖	failure, `-c` active, and cruft directory could not be created

SECTION 3

Config

`latexindent.pl` loads its default settings from `defaultSettings.yaml`. The idea is to separate the behaviour of the script from the internal working – this is very similar to the way that we separate content from form when writing our documents in LaTeX. *All of these settings can be changed by you, the user; demonstrations are given throughout this documentation using the -l switch*.

If you look in `defaultSettings.yaml` you'll find the settings that govern the behaviour of `latexindent.pl`. If you're not sure where `defaultSettings.yaml` resides on your computer, don't worry as `indent.log` will tell you where to find it. `defaultSettings.yaml` is commented, but here is a description of what each switch is designed to do. The default value is given in each case; whenever you see *integer* in *this* section, assume that it must be greater than or equal to `0` unless otherwise stated.

For most of the settings in `defaultSettings.yaml` that are specified as integers, then we understand `0` to represent 'off' and `1` to represent 'on'. For fields that allow values other than `0` or `1`, it is hoped that the specific context and associated commentary should make it clear which values are allowed.

`fileExtensionPreference`: ⟨*fields*⟩

`latexindent.pl` can be called to act on a file without specifying the file extension. For example we can call

```
cmh:~$ latexindent.pl myfile
```

in which case the script will look for `myfile` with the extensions specified in `fileExtensionPreference` in their numeric order. If no match is found, the script will exit. As with all of the fields, you should change and/or add to this as necessary.

LISTING 26: `fileExtensionPreference`

```
31 fileExtensionPreference:
32   .tex: 1
33   .sty: 2
34   .cls: 3
35   .bib: 4
```

Calling `latexindent.pl myfile` with the (default) settings specified in Listing 26 means that the script will first look for `myfile.tex`, then `myfile.sty`, `myfile.cls`, and finally `myfile.bib` in order[2].

3.1 Backup and log file preferences

`backupExtension`: ⟨*extension name*⟩

If you call `latexindent.pl` with the `-w` switch (to overwrite `myfile.tex`) then it will create a backup file before doing any indentation; the default extension is `.bak`, so, for example, `myfile.bak0` would be created when calling `latexindent.pl myfile.tex` for the first time.

By default, every time you subsequently call `latexindent.pl` with the `-w` to act upon `myfile.tex`, it will create successive back up files: `myfile.bak1`, `myfile.bak2`, etc.

[2] Throughout this manual, listings shown with line numbers represent code taken directly from `defaultSettings.yaml`.

`onlyOneBackUp`: ⟨*integer*⟩

If you don't want a backup for every time that you call `latexindent.pl` (so you don't want `myfile.bak1`, `myfile.bak2`, etc) and you simply want `myfile.bak` (or whatever you chose `backupExtension` to be) then change `onlyOneBackUp` to `1`; the default value of `onlyOneBackUp` is `0`.

`maxNumberOfBackUps`: ⟨*integer*⟩

Some users may only want a finite number of backup files, say at most 3, in which case, they can change this switch. The smallest value of `maxNumberOfBackUps` is 0 which will *not* prevent backup files being made; in this case, the behaviour will be dictated entirely by `onlyOneBackUp`. The default value of `maxNumberOfBackUps` is `0`.

`cycleThroughBackUps`: ⟨*integer*⟩

Some users may wish to cycle through backup files, by deleting the oldest backup file and keeping only the most recent; for example, with `maxNumberOfBackUps: 4`, and `cycleThroughBackUps` set to `1` then the `copy` procedure given below would be obeyed.

```
cmh:~$ copy myfile.bak1 to myfile.bak0
cmh:~$ copy myfile.bak2 to myfile.bak1
cmh:~$ copy myfile.bak3 to myfile.bak2
cmh:~$ copy myfile.bak4 to myfile.bak3
```

The default value of `cycleThroughBackUps` is `0`.

3.2 Verbatim code blocks

`verbatimEnvironments`: ⟨*fields*⟩

A field that contains a list of environments that you would like left completely alone – no indentation will be performed on environments that you have specified in this field, see Listing 27.

LISTING 27: `verbatimEnvironments`

```
68 verbatimEnvironments:
69   verbatim: 1
70   lstlisting: 1
71   minted: 1
```

LISTING 28: `verbatimCommands`

```
74 verbatimCommands:
75   verb: 1
76   lstinline: 1
```

Note that if you put an environment in `verbatimEnvironments` and in other fields such as `lookForAlignDelims` or `noAdditionalIndent` then `latexindent.pl` will *always* prioritize `verbatimEnvironments`.

N: 2021-10-30

You can, optionally, specify the `verbatim` field using the `name` field which takes a regular expression as its argument; thank you to [18] for contributing this feature.

example 10 For demonstration, then assuming that your file contains the environments `latexcode`, `latexcode*`, `pythoncode` and `pythoncode*`, then the listings given in Listings 29 and 30 are equivalent.

LISTING 29: `nameAsRegex1.yaml`

```
verbatimEnvironments:
    latexcode: 1
    latexcode*: 1
    pythoncode: 1
    pythoncode*: 1
```

LISTING 30: `nameAsRegex2.yaml`

```
verbatimEnvironments:
    nameAsRegex:
      name: '\w+code\*?'
      lookForThis: 1
```

With reference to Listing 30: ■

- the `name` field as specified here means *any word followed by the word code, optionally followed by **;
- we have used `nameAsRegex` to identify this field, but you can use any description you like;
- the `lookForThis` field is optional, and can take the values 0 (off) or 1 (on); by default, it is assumed to be 1 (on).

■

`verbatimCommands`: ⟨*fields*⟩

A field that contains a list of commands that are verbatim commands, for example `\lstinline`; any commands populated in this field are protected from line breaking routines (only relevant if the `-m` is active, see Section 4 on page 70).

With reference to Listing 28, by default `latexindent.pl` looks for `\verb` immediately followed by another character, and then it takes the body as anything up to the next occurrence of the character; this means that, for example, `\verb!x+3!` is treated as a `verbatimCommands`.

N: 2021-10-30

You can, optionally, specify the `verbatimCommands` field using the `name` field which takes a regular expression as its argument; thank you to [18] for contributing this feature.

example 11 For demonstration, then assuming that your file contains the commands `verbinline`, `myinline` then the listings given in Listings 31 and 32 are equivalent.

LISTING 31: `nameAsRegex3.yaml`

```
verbatimCommands:
    verbinline: 1
    myinline: 1
```

LISTING 32: `nameAsRegex4.yaml`

```
verbatimCommands:
    nameAsRegex:
      name: '\w+inline'
      lookForThis: 1
```

With reference to Listing 32:

- the `name` field as specified here means *any word followed by the word inline*;
- we have used `nameAsRegex` to identify this field, but you can use any description you like;
- the `lookForThis` field is optional, and can take the values 0 (off) or 1 (on); by default, it is assumed to be 1 (on).

■

`noIndentBlock`: ⟨*fields*⟩

If you have a block of code that you don't want `latexindent.pl` to touch (even if it is *not* a verbatim-like environment) then you can wrap it in an environment from `noIndentBlock`; you can use any name you like for this, provided you populate it as demonstrate in Listing 33.

LISTING 33: `noIndentBlock`

```
81 noIndentBlock:
82   noindent: 1
83   cmhtest: 1
```

Of course, you don't want to have to specify these as null environments in your code, so you use them with a comment symbol, %, followed by as many spaces (possibly none) as you like; see Listing 34 for example.

LISTING 34: noIndentBlock.tex

```
% \begin{noindent}
some before text
        this code
                 won't
     be touched
                    by
              latexindent.pl!
some after text
% \end{noindent}
```

Important note: it is assumed that the noindent block statements specified in this way appear on their own line.

example 12

N: 2021-06-19

The noIndentBlock fields can also be specified in terms of begin and end fields. We use the code in Listing 35 to demonstrate this feature.

LISTING 35: noIndentBlock1.tex

```
some before text
        this code
                 won't
     be touched
                    by
              latexindent.pl!
some after text
```

The settings given in Listings 36 and 37 are equivalent:

LISTING 36: noindent1.yaml

```
noIndentBlock:
    demo:
        begin: 'some\hbefore'
        body: '.*?'
        end: 'some\hafter\htext'
        lookForThis: 1
```

LISTING 37: noindent2.yaml

```
noIndentBlock:
    demo:
        begin: 'some\hbefore'
        end: 'some\hafter\htext'
```

LISTING 38: noindent3.yaml

```
noIndentBlock:
    demo:
        begin: 'some\hbefore'
        body: '.*?'
        end: 'some\hafter\htext'
        lookForThis: 0
```

Upon running the commands

```
cmh:~$ latexindent.pl -l noindent1.yaml noindent1
cmh:~$ latexindent.pl -l noindent2.yaml noindent1
```

then we receive the output given in Listing 39.

LISTING 39: noIndentBlock1.tex using Listing 36 or Listing 37

```
some before text
        this code
                 won't
     be touched
                    by
              latexindent.pl!
some after text
```

■

The begin, body and end fields for noIndentBlock are all *regular expressions*. If the body field is not specified, then it takes a default value of .*? which is written explicitly in Listing 36. In this context, we interpret .*? in words as *the fewest number of characters (possibly none) until the 'end' field is reached*.

The lookForThis field is optional, and can take the values 0 (off) or 1 (on); by default, it is assumed to be 1 (on).

example 13 Using Listing 38 demonstrates setting `lookForThis` to 0 (off); running the command

```
cmh:~$ latexindent.pl -l noindent3.yaml noindent1
```

gives the output in Listing 40.

LISTING 40: `noIndentBlock1.tex` using Listing 38

```
some before text
this code
won't
be touched
by
latexindent.pl!
some after text
```

■

We will demonstrate this feature later in the documentation in Listing 548.

N: 2021-10-30

You can, optionally, specify the `noIndentBlock` field using the `name` field which takes a regular expression as its argument; thank you to [18] for contributing this feature.

example 14 For demonstration, then assuming that your file contains the environments `testnoindent`, `testnoindent*` then the listings given in Listings 41 and 42 are equivalent.

LISTING 41: `nameAsRegex5.yaml`

```
noIndentBlock:
    mytest:
      begin: '\\begin\{testnoindent\*?\}'
      end: '\\end\{testnoindent\*?\}'
```

LISTING 42: `nameAsRegex6.yaml`

```
noIndentBlock:
    nameAsRegex:
      name: '\w+noindent\*?'
      lookForThis: 1
```

With reference to Listing 42:

- the `name` field as specified here means *any word followed by the word noindent, optionally followed by **;
- we have used `nameAsRegex` to identify this field, but you can use any description you like;
- the `lookForThis` field is optional, and can take the values 0 (off) or 1 (on); by default, it is assumed to be 1 (on).

■

3.3 filecontents and preamble

`fileContentsEnvironments`: ⟨*field*⟩

Before `latexindent.pl` determines the difference between preamble (if any) and the main document, it first searches for any of the environments specified in `fileContentsEnvironments`, see Listing 43. The behaviour of `latexindent.pl` on these environments is determined by their location (preamble or not), and the value `indentPreamble`, discussed next.

LISTING 43: `fileContentsEnvironments`

```
87 fileContentsEnvironments:
88   filecontents: 1
89   filecontents*: 1
```

`indentPreamble`: **0**|**1**

The preamble of a document can sometimes contain some trickier code for `latexindent.pl` to operate upon. By default, `latexindent.pl` won't try to operate on the preamble (as `indentPreamble`

is set to 0, by default), but if you'd like `latexindent.pl` to try then change `indentPreamble` to 1.

`lookForPreamble`: ⟨*fields*⟩

Not all files contain preamble; for example, `sty`, `cls` and `bib` files typically do *not*. Referencing Listing 44, if you set, for example, `.tex` to 0, then regardless of the setting of the value of `indentPreamble`, preamble will not be assumed when operating upon `.tex` files.

LISTING 44: lookForPreamble

```
95  lookForPreamble:
96    .tex: 1
97    .sty: 0
98    .cls: 0
99    .bib: 0
100   STDIN: 1
```

3.4 Indentation and horizontal space

`defaultIndent`: ⟨*horizontal space*⟩

This is the default indentation used in the absence of other details for the code block with which we are working. The default value is `\t` which means a tab; we will explore customisation beyond `defaultIndent` in Section 3.9 on page 50.

If you're interested in experimenting with `latexindent.pl` then you can *remove* all indentation by setting `defaultIndent: ""`.

`removeTrailingWhitespace`: ⟨*fields*⟩

Trailing white space can be removed both *before* and *after* processing the document, as detailed in Listing 45; each of the fields can take the values 0 or 1. See Listings 443 to 445 on page 107 for before and after results. Thanks to [3] for providing this feature.

LISTING 45: removeTrailingWhitespace

```
106 removeTrailingWhitespace:
107   beforeProcessing: 0
108   afterProcessing: 1
```

LISTING 46: removeTrailingWhitespace (alt)

```
removeTrailingWhitespace: 1
```

N: 2017-06-28

You can specify `removeTrailingWhitespace` simply as 0 or 1, if you wish; in this case, `latexindent.pl` will set both `beforeProcessing` and `afterProcessing` to the value you specify; see Listing 46.

3.5 Aligning at delimiters

`lookForAlignDelims`: ⟨*fields*⟩

This contains a list of code blocks that are operated upon in a special way by `latexindent.pl` (see Listing 47). In fact, the fields in `lookForAlignDelims` can actually take two different forms: the *basic* version is shown in Listing 47 and the *advanced* version in Listing 50; we will discuss each in turn.

LISTING 47: lookForAlignDelims (basic)

```
111 lookForAlignDelims:
112   tabular: 1
113   tabularx: 1
114   longtable: 1
```

Specifying code blocks in this field instructs `latexindent.pl` to try and align each column by its alignment delimiters. It does have some limitations (discussed further in Section 8), but in many cases it will produce results such as those in Listings 48 and 49; running the command

```
cmh:~$ latexindent.pl tabular1.tex
```

gives the output given in Listing 49.

LISTING 48: `tabular1.tex`

```
\begin{tabular}{cccc}
1& 2 &3        &4\\
5& &6        &\\
\end{tabular}
```

LISTING 49: `tabular1.tex` default output

```
\begin{tabular}{cccc}
   1 & 2 & 3 & 4 \\
   5 &   & 6 &   \\
\end{tabular}
```

If you find that `latexindent.pl` does not perform satisfactorily on such environments then you can set the relevant key to 0, for example `tabular: 0`; alternatively, if you just want to ignore *specific* instances of the environment, you could wrap them in something from `noIndentBlock` (see Listing 33 on page 23).

If, for example, you wish to remove the alignment of the \\ within a delimiter-aligned block, then the advanced form of `lookForAlignDelims` shown in Listing 50 is for you.

LISTING 50: `lookForAlignDelims` (advanced)

```
tabular:
  delims: 1
  alignDoubleBackSlash: 1
  spacesBeforeDoubleBackSlash: 1
  multiColumnGrouping: 0
  alignRowsWithoutMaxDelims: 1
  spacesBeforeAmpersand: 1
  spacesAfterAmpersand: 1
  justification: left
  alignFinalDoubleBackSlash: 0
  dontMeasure: 0
  delimiterRegEx: (?<!\\)(&)
  delimiterJustification: left
  lookForChildCodeBlocks: 1
  alignContentAfterDoubleBackSlash: 0
  spacesAfterDoubleBackSlash: 1
  doubleBackSlash: \\\\(?:\h*\[\h*[0-9.]+\h*[a-zA-Z]+\h*\])?
```

N: 2026-03-15

Note that you can use a mixture of the basic and advanced form: in Listing 47 the entries are basic; in Listing 50 the entry for `tabular` is advanced. The default values for the advanced form are controlled in `fineTuning`, see Section 7.6 on page 140. When using the advanced form, each field should receive at least 1 sub-field, and *can* (but does not have to) receive any of the following fields:

- `delims`: binary switch (0 or 1) equivalent to simply specifying, for example, `tabular: 1` in the basic version shown in Listing 47. If `delims` is set to 0 then the align at ampersand routine will not be called for this code block (default: 1);
- `alignDoubleBackSlash`: binary switch (0 or 1) to determine if \\ should be aligned (default: 1);

U: 2018-01-13

- `spacesBeforeDoubleBackSlash`: optionally, specifies the number (integer ≥ 0) of spaces to be inserted before \\ (default: 1);

N: 2017-06-19

- `multiColumnGrouping`: binary switch (0 or 1) that details if `latexindent.pl` should group columns above and below a `\multicolumn` command (default: 0);

N: 2017-06-19

- `alignRowsWithoutMaxDelims`: binary switch (0 or 1) that details if rows that do not contain the maximum number of delimiters should be formatted so as to have the ampersands aligned (default: 1);

N: 2018-01-13
- `spacesBeforeAmpersand`: optionally specifies the number (integer ≥ 0) of spaces to be placed *before* ampersands (default: 1);

N: 2018-01-13
- `spacesAfterAmpersand`: optionally specifies the number (integer ≥ 0) of spaces to be placed *After* ampersands (default: 1);

N: 2018-01-13
- `justification`: optionally specifies the justification of each cell as either *left* or *right* (default: left);

N: 2020-03-21
- alignFinalDoubleBackSlash optionally specifies if the *final* double backslash should be used for alignment (default: 0);

N: 2020-03-21
- dontMeasure optionally specifies if user-specified cells, rows or the largest entries should *not* be measured (default: 0);

N: 2020-03-21
- delimiterRegEx optionally specifies the pattern matching to be used for the alignment delimiter (default: `'(?<!\\)(&)'`);

N: 2020-03-21
- delimiterJustification optionally specifies the justification for the alignment delimiters (default: left); note that this feature is only useful if you have delimiters of different lengths in the same column, discussed in Section 3.5.3;

N: 2021-12-13
- lookForChildCodeBlocks optionally instructs `latexindent.pl` to search for child code blocks or not (default: 1), discussed in Section 3.5.4;

N: 2023-05-01
- alignContentAfterDoubleBackSlash optionally instructs `latexindent.pl` to align content *after* double back slash (default: 0), discussed in Section 3.5.5;

N: 2023-05-01
- spacesAfterDoubleBackSlash optionally specifies the number (integer ≥ 0) of spaces to be placed *after* the double back slash *when alignContentAfterDoubleBackSlash is active*; demonstrated in Section 3.5.5.

example 15 We will explore most of these features using the file `tabular2.tex` in Listing 51 (which contains a `\multicolumn` command), and the YAML files in Listings 52 to 58; the `dontMeasure` feature will be described in Section 3.5.2, and `delimiterRegEx` in Section 3.5.3.

LISTING 51: tabular2.tex

```
\begin{tabular}{cccc}
A&    B & C       &D\\
AAA&    BBB & CCC       &DDD\\
  \multicolumn{2}{c}{first heading} & \multicolumn{2}{c}{second heading}\\
one&    two & three       &four\\
five& &six      &\\
seven & \\
\end{tabular}
```

LISTING 52: tabular2.yaml

```
lookForAlignDelims:
   tabular:
      multiColumnGrouping: 1
```

LISTING 53: tabular3.yaml

```
lookForAlignDelims:
   tabular:
      alignRowsWithoutMaxDelims: 0
```

LISTING 54: tabular4.yaml

```
lookForAlignDelims:
   tabular:
      spacesBeforeAmpersand: 4
```

LISTING 55: tabular5.yaml

```
lookForAlignDelims:
   tabular:
      spacesAfterAmpersand: 4
```

LISTING 56: tabular6.yaml

```
lookForAlignDelims:
   tabular:
      alignDoubleBackSlash: 0
```

LISTING 57: tabular7.yaml

```
lookForAlignDelims:
   tabular:
      spacesBeforeDoubleBackSlash: 0
```

■

LISTING 58: tabular8.yaml

```
lookForAlignDelims:
   tabular:
      justification: "right"
```

On running the commands

```
cmh:~$ latexindent.pl tabular2.tex
cmh:~$ latexindent.pl tabular2.tex -l tabular2.yaml
cmh:~$ latexindent.pl tabular2.tex -l tabular3.yaml
cmh:~$ latexindent.pl tabular2.tex -l tabular2.yaml,tabular4.yaml
cmh:~$ latexindent.pl tabular2.tex -l tabular2.yaml,tabular5.yaml
cmh:~$ latexindent.pl tabular2.tex -l tabular2.yaml,tabular6.yaml
cmh:~$ latexindent.pl tabular2.tex -l tabular2.yaml,tabular7.yaml
cmh:~$ latexindent.pl tabular2.tex -l tabular2.yaml,tabular8.yaml
```

we obtain the respective outputs given in Listings 59 to 66.

LISTING 59: tabular2.tex default output

```
\begin{tabular}{cccc}
   A                                & B                                & C     & D    \\
   AAA                              & BBB                              & CCC   & DDD  \\
   \multicolumn{2}{c}{first heading} & \multicolumn{2}{c}{second heading}                \\
   one                              & two                              & three & four \\
   five                             &                                  & six   &      \\
   seven                            &                                  &       &      \\
\end{tabular}
```

LISTING 60: tabular2.tex using Listing 52

```
\begin{tabular}{cccc}
   A     & B                        & C     & D                        \\
   AAA   & BBB                      & CCC   & DDD                      \\
   \multicolumn{2}{c}{first heading} & \multicolumn{2}{c}{second heading} \\
   one   & two                      & three & four                     \\
   five  &                          & six   &                          \\
   seven &                                                             \\
\end{tabular}
```

LISTING 61: tabular2.tex using Listing 53

```
\begin{tabular}{cccc}
   A    & B   & C     & D                                              \\
   AAA  & BBB & CCC   & DDD                                            \\
   \multicolumn{2}{c}{first heading} & \multicolumn{2}{c}{second heading} \\
   one  & two & three & four                                           \\
   five &     & six   &                                                \\
   seven &                                                            \\
\end{tabular}
```

LISTING 62: tabular2.tex using Listings 52 and 54

```
\begin{tabular}{cccc}
   A        & B                          & C        & D                          \\
   AAA      & BBB                        & CCC      & DDD                        \\
   \multicolumn{2}{c}{first heading}     & \multicolumn{2}{c}{second heading} \\
   one      & two                        & three    & four                       \\
   five     &                            & six      &                            \\
   seven    &                                                                    \\
\end{tabular}
```

LISTING 63: tabular2.tex using Listings 52 and 55

```
\begin{tabular}{cccc}
   A     &    B                       &    C     &    D                       \\
   AAA   &    BBB                     &    CCC   &    DDD                     \\
   \multicolumn{2}{c}{first heading} &    \multicolumn{2}{c}{second heading} \\
   one   &    two                     &    three &    four                    \\
   five  &                            &    six   &                            \\
   seven &                                                                    \\
\end{tabular}
```

LISTING 64: tabular2.tex using Listings 52 and 56

```
\begin{tabular}{cccc}
   A     & B                          & C     & D \\
   AAA   & BBB                        & CCC   & DDD \\
   \multicolumn{2}{c}{first heading} & \multicolumn{2}{c}{second heading} \\
   one   & two                        & three & four \\
   five  &                            & six   & \\
   seven & \\
\end{tabular}
```

LISTING 65: tabular2.tex using Listings 52 and 57

```
\begin{tabular}{cccc}
   A     & B                          & C     & D                          \\
   AAA   & BBB                        & CCC   & DDD                        \\
   \multicolumn{2}{c}{first heading} & \multicolumn{2}{c}{second heading}\\
   one   & two                        & three & four                       \\
   five  &                            & six   &                            \\
   seven &                                                                 \\
\end{tabular}
```

LISTING 66: tabular2.tex using Listings 52 and 58

```
\begin{tabular}{cccc}
                              A &   B &                              C &    D \\
                            AAA & BBB &                            CCC &  DDD \\
   \multicolumn{2}{c}{first heading} & \multicolumn{2}{c}{second heading} \\
                            one & two &                          three & four \\
                           five &     &                            six &      \\
                          seven &                                        \\
\end{tabular}
```

Notice in particular:

- in both Listings 59 and 60 all rows have been aligned at the ampersand, even those that do not contain the maximum number of ampersands (3 ampersands, in this case);
- in Listing 59 the columns have been aligned at the ampersand;
- in Listing 60 the \multicolumn command has grouped the 2 columns beneath *and* above it,

because `multiColumnGrouping` is set to 1 in Listing 52;

- in Listing 61 rows 3 and 6 have *not* been aligned at the ampersand, because `alignRowsWithoutMaxDelims` has been to set to 0 in Listing 53; however, the `\\` *have* still been aligned;
- in Listing 62 the columns beneath and above the `\multicolumn` commands have been grouped (because `multiColumnGrouping` is set to 1), and there are at least 4 spaces *before* each aligned ampersand because `spacesBeforeAmpersand` is set to 4;
- in Listing 63 the columns beneath and above the `\multicolumn` commands have been grouped (because `multiColumnGrouping` is set to 1), and there are at least 4 spaces *after* each aligned ampersand because `spacesAfterAmpersand` is set to 4;
- in Listing 64 the `\\` have *not* been aligned, because `alignDoubleBackSlash` is set to 0, otherwise the output is the same as Listing 60;
- in Listing 65 the `\\` *have* been aligned, and because `spacesBeforeDoubleBackSlash` is set to 0, there are no spaces ahead of them; the output is otherwise the same as Listing 60;
- in Listing 66 the cells have been *right*-justified; note that cells above and below the `\multicol` statements have still been group correctly, because of the settings in Listing 52.

■

3.5.1 lookForAlignDelims: spacesBeforeAmpersand

U: 2021-06-19

The `spacesBeforeAmpersand` can be specified in a few different ways. The *basic* form is demonstrated in Listing 54, but we can customise the behaviour further by specifying if we would like this value to change if it encounters a *leading blank column*; that is, when the first column contains only zero-width entries. We refer to this as the *advanced* form.

example 16 We demonstrate this feature in relation to Listing 67; upon running the following command

```
cmh:~$ latexindent.pl aligned1.tex -o=+-default
```

then we receive the default output given in Listing 68.

LISTING 67: `aligned1.tex`

```
\begin{aligned}
& a & b, \\
& c & d.
\end{aligned}
```

LISTING 68: `aligned1-default.tex`

```
\begin{aligned}
    & a & b, \\
    & c & d.
\end{aligned}
```

The settings in Listings 69 to 72 are all equivlanent; we have used the not-yet discussed `noAdditionalIndent` field (see Section 3.9 on page 50) which will assist in the demonstration in what follows.

LISTING 69: `sba1.yaml`

```
noAdditionalIndent:
  aligned: 1
lookForAlignDelims:
   aligned: 1
```

LISTING 70: `sba2.yaml`

```
noAdditionalIndent:
  aligned: 1
lookForAlignDelims:
   aligned:
      spacesBeforeAmpersand: 1
```

LISTING 71: `sba3.yaml`

```
noAdditionalIndent:
  aligned: 1
lookForAlignDelims:
   aligned:
      spacesBeforeAmpersand:
        default: 1
```

LISTING 72: `sba4.yaml`

```
noAdditionalIndent:
  aligned: 1
lookForAlignDelims:
   aligned:
      spacesBeforeAmpersand:
        leadingBlankColumn: 1
```

Upon running the following commands

■

```
cmh:~$ latexindent.pl aligned1.tex -l sba1.yaml
cmh:~$ latexindent.pl aligned1.tex -l sba2.yaml
cmh:~$ latexindent.pl aligned1.tex -l sba3.yaml
cmh:~$ latexindent.pl aligned1.tex -l sba4.yaml
```

then we receive the (same) output given in Listing 73; we note that there is *one space* before each ampersand.

LISTING 73: aligned1-mod1.tex

```
\begin{aligned}
 & a & b, \\
 & c & d.
\end{aligned}
```

We note in particular:

- Listing 69 demonstrates the *basic* form for `lookForAlignDelims`; in this case, the default values are specified as in Listing 50 on page 27;
- Listing 70 demonstrates the *advanced* form for `lookForAlignDelims` and specified `spacesBeforeAmpersand`. The default value is `1`;
- Listing 71 demonstrates the new *advanced* way to specify `spacesBeforeAmpersand`, and for us to set the `default` value that sets the number of spaces before ampersands which are *not* in leading blank columns. The default value is `1`.

  We note that `leadingBlankColumn` has not been specified in Listing 71, and it will inherit the value from `default`;
- Listing 72 demonstrates spaces to be used before amperands for *leading blank columns*. We note that *default* has not been specified, and it will be set to `1` by default.

■

example 17 We can customise the space before the ampersand in the *leading blank column* of Listing 73 by using either of Listings 74 and 75, which are equivalent.

LISTING 74: sba5.yaml

```
noAdditionalIndent:
  aligned: 1
lookForAlignDelims:
   aligned:
      spacesBeforeAmpersand:
        leadingBlankColumn: 0
```

LISTING 75: sba6.yaml

```
noAdditionalIndent:
  aligned: 1
lookForAlignDelims:
   aligned:
      spacesBeforeAmpersand:
        leadingBlankColumn: 0
        default: 1
```

Upon running

```
cmh:~$ latexindent.pl aligned1.tex -l sba5.yaml
cmh:~$ latexindent.pl aligned1.tex -l sba6.yaml
```

then we receive the (same) output given in Listing 76. We note that the space before the ampersand in the *leading blank column* has been set to `0` by Listing 75.

We can demonstrated this feature further using the settings in Listing 78 which give the output in Listing 77.

■

LISTING 76: aligned1-mod5.tex

```
\begin{aligned}
& a & b, \\
& c & d.
\end{aligned}
```

LISTING 77: aligned1.tex using Listing 78

```
\begin{aligned}
   & a& b, \\
   & c& d.
\end{aligned}
```

LISTING 78: sba7.yaml

```
noAdditionalIndent:
  aligned: 1
lookForAlignDelims:
   aligned:
      spacesBeforeAmpersand:
        leadingBlankColumn: 3
        default: 0
```

3.5.2 lookForAlignDelims: the dontMeasure feature

N: 2020-03-21

The `lookForAlignDelims` field can, optionally, receive the `dontMeasure` option which can be specified in a few different ways.

example 18 We will explore this feature in relation to the code given in Listing 79; the default output is shown in Listing 80.

LISTING 79: tabular-DM.tex

```
\begin{tabular}{cccc}
  aaaaaa&bbbbb&ccc&dd\\
  11&2&33&4\\
  5&66&7&8
\end{tabular}
```

LISTING 80: tabular-DM.tex default output

```
\begin{tabular}{cccc}
   aaaaaa & bbbbb & ccc & dd \\
   11     & 2     & 33  & 4  \\
   5      & 66    & 7   & 8
\end{tabular}
```

The `dontMeasure` field can be specified as `largest`, and in which case, the largest element will not be measured; with reference to the YAML file given in Listing 82, we can run the command

```
cmh:~$ latexindent.pl tabular-DM.tex -l=dontMeasure1.yaml
```

and receive the output given in Listing 81.

LISTING 81: tabular-DM.tex using Listing 82

```
\begin{tabular}{cccc}
   aaaaaa & bbbbb & ccc & dd \\
   11 & 2  & 33 & 4          \\
   5  & 66 & 7  & 8
\end{tabular}
```

LISTING 82: dontMeasure1.yaml

```
lookForAlignDelims:
   tabular:
      dontMeasure: largest
      lookForChildCodeBlocks: 0
```

We note that the *largest* column entries have not contributed to the measuring routine.

example 19 The `dontMeasure` field can also be specified in the form demonstrated in Listing 84. On running the following commands,

```
cmh:~$ latexindent.pl tabular-DM.tex -l=dontMeasure2.yaml
```

we receive the output in Listing 83.

LISTING 83: tabular-DM.tex using Listing 84 or Listing 86

```
\begin{tabular}{cccc}
   aaaaaa & bbbbb & ccc & dd \\
   11 & 2  & 33 & 4          \\
   5  & 66 & 7  & 8
\end{tabular}
```

LISTING 84: dontMeasure2.yaml

```
lookForAlignDelims:
   tabular:
      dontMeasure:
        - aaaaaa
        - bbbbb
        - ccc
        - dd
```

We note that in Listing 84 we have specified entries not to be measured, one entry per line. ■

example 20 The `dontMeasure` field can also be specified in the forms demonstrated in Listing 86 and Listing 87. Upon running the commands

```
cmh:~$ latexindent.pl tabular-DM.tex -l=dontMeasure3.yaml
cmh:~$ latexindent.pl tabular-DM.tex -l=dontMeasure4.yaml
```

we receive the output given in Listing 85

LISTING 85: tabular-DM.tex using Listing 86 or Listing 86

```
\begin{tabular}{cccc}
   aaaaaa & bbbbb & ccc & dd \\
   11 & 2  & 33 & 4          \\
   5  & 66 & 7  & 8
\end{tabular}
```

LISTING 86: dontMeasure3.yaml

```
lookForAlignDelims:
   tabular:
      dontMeasure:
        -
          this: aaaaaa
          applyTo: cell
        -
          this: bbbbb
        - ccc
        - dd
```

LISTING 87: dontMeasure4.yaml

```
lookForAlignDelims:
   tabular:
      dontMeasure:
        -
          regex: [a-z]
          applyTo: cell
```

We note that in:

- Listing 86 we have specified entries not to be measured, each one has a *string* in the `this` field, together with an optional specification of `applyTo` as `cell`;
- Listing 87 we have specified entries not to be measured as a *regular expression* using the `regex` field, together with an optional specification of `applyTo` as `cell` field, together with an optional specification of `applyTo` as `cell`.

In both cases, the default value of `applyTo` is `cell`, and does not need to be specified. ■

example 21 We may also specify the `applyTo` field as `row`, a demonstration of which is given in Listing 89; upon running

```
cmh:~$ latexindent.pl tabular-DM.tex -l=dontMeasure5.yaml
```

we receive the output in Listing 88.

LISTING 88: tabular-DM.tex using Listing 89

```
\begin{tabular}{cccc}
   aaaaaa & bbbbb & ccc & dd \\
   11 & 2  & 33 & 4          \\
   5  & 66 & 7  & 8
\end{tabular}
```

LISTING 89: dontMeasure5.yaml

```
lookForAlignDelims:
   tabular:
      dontMeasure:
        -
          this: aaaaaa&bbbbb&ccc&dd\\
          applyTo: row
```

■

example 22 Finally, the `applyTo` field can be specified as `row`, together with a `regex` expression. For example, for the settings given in Listing 91, upon running

```
cmh:~$ latexindent.pl tabular-DM.tex -l=dontMeasure6.yaml
```

we receive the output in Listing 90. ■

LISTING 90: tabular-DM.tex using Listing 91

```
\begin{tabular}{cccc}
   aaaaaa & bbbbb & ccc & dd \\
   11 & 2  & 33 & 4          \\
   5  & 66 & 7  & 8
\end{tabular}
```

LISTING 91: dontMeasure6.yaml

```
lookForAlignDelims:
   tabular:
      dontMeasure:
        -
          regex: [a-z]
          applyTo: row
```

■

3.5.3 lookForAlignDelims: the delimiterRegEx and delimiterJustification feature

N: 2020-03-21

The delimiter alignment will, by default, align code blocks at the ampersand character. The behaviour is controlled by the `delimiterRegEx` field within `lookForAlignDelims`; the default value is `'(?<!\\)(&)'`, which can be read as: *an ampersand, as long as it is not immediately preceded by a backslash*.

Warning!

Important: note the 'capturing' parenthesis in the `(&)` which are necessary; if you intend to customise this field, then be sure to include them appropriately.

example 23 We demonstrate how to customise this with respect to the code given in Listing 92; the default output from `latexindent.pl` is given in Listing 93.

LISTING 92: tabbing.tex

```
\begin{tabbing}
    aa \=   bb \= cc \= dd \= ee \\
    \>2\> 1 \> 7 \> 3 \\
    \>3 \> 2\>8\> 3 \\
    \>4 \>2 \\
\end{tabbing}
```

LISTING 93: tabbing.tex default output

```
\begin{tabbing}
   aa \=   bb \= cc \= dd \= ee \\
   \>2\> 1 \> 7 \> 3 \\
   \>3 \> 2\>8\> 3 \\
   \>4 \>2 \\
\end{tabbing}
```

Let's say that we wish to align the code at either the `\=` or `\>`. We employ the settings given in Listing 95 and run the command

```
cmh:~$ latexindent.pl tabbing.tex -l=delimiterRegEx1.yaml
```

to receive the output given in Listing 94.

LISTING 94: tabbing.tex using Listing 95

```
\begin{tabbing}
   aa \= bb \= cc \= dd \= ee \\
      \> 2  \> 1  \> 7  \> 3  \\
      \> 3  \> 2  \> 8  \> 3  \\
      \> 4  \> 2              \\
\end{tabbing}
```

LISTING 95: delimiterRegEx1.yaml

```
lookForAlignDelims:
   tabbing:
    delimiterRegEx: '(\\(?:=|>))'
```

We note that:

- in Listing 94 the code has been aligned, as intended, at both the `\=` and `\>`;
- in Listing 95 we have heeded the warning and captured the expression using grouping parenthesis, specified a backslash using `\\` and said that it must be followed by either `=` or `>`.

■

example 24 We can explore `delimiterRegEx` a little further using the settings in Listing 97 and run the command

■

```
cmh:~$ latexindent.pl tabbing.tex -l=delimiterRegEx2.yaml
```

to receive the output given in Listing 96.

LISTING 96: tabbing.tex using Listing 97

```
\begin{tabbing}
   aa \=   bb \= cc \= dd \= ee \\
    \> 2 \> 1 \> 7 \> 3         \\
    \> 3 \> 2 \> 8 \> 3         \\
    \> 4 \> 2                   \\
\end{tabbing}
```

LISTING 97: delimiterRegEx2.yaml

```
lookForAlignDelims:
   tabbing:
    delimiterRegEx: '(\\>)'
```

We note that only the \> have been aligned. ■

example 25 Of course, the other lookForAlignDelims options can be used alongside the `delimiterRegEx`; regardless of the type of delimiter being used (ampersand or anything else), the fields from Listing 50 on page 27 remain the same; for example, using the settings in Listing 99, and running

```
cmh:~$ latexindent.pl tabbing.tex -l=delimiterRegEx3.yaml
```

to receive the output given in Listing 98.

LISTING 98: tabbing.tex using Listing 99

```
\begin{tabbing}
   aa\=bb\=cc\=dd\=ee \\
     \>2 \>1 \>7 \>3  \\
     \>3 \>2 \>8 \>3  \\
     \>4 \>2          \\
\end{tabbing}
```

LISTING 99: delimiterRegEx3.yaml

```
lookForAlignDelims:
   tabbing:
    delimiterRegEx: '(\\(?:=|>))'
    spacesBeforeAmpersand: 0
    spacesAfterAmpersand: 0
```

■

example 26 It is possible that delimiters specified within `delimiterRegEx` can be of different lengths. Consider the file in Listing 100, and associated YAML in Listing 102. Note that the Listing 102 specifies the option for the delimiter to be either `#` or `\>`, *which are different lengths*. Upon running the command

```
cmh:~$ latexindent.pl tabbing1.tex -l=delimiterRegEx4.yaml -o=+-mod4
```

we receive the output in Listing 101.

LISTING 100: tabbing1.tex

```
\begin{tabbing}
    1#22\>333\\
    xxx#aaa#yyyyy\\
    .##&\\
\end{tabbing}
```

LISTING 101: tabbing1-mod4.tex

```
\begin{tabbing}
   1   # 22  \> 333   \\
   xxx # aaa #  yyyyy \\
   .   #     #  &     \\
\end{tabbing}
```

LISTING 102: delimiterRegEx4.yaml

```
lookForAlignDelims:
   tabbing:
    delimiterRegEx: '(#|\\>)'
```

■

example 27 You can set the *delimiter* justification as either `left` (default) or `right`, which will only have effect when delimiters in the same column have different lengths. Using the settings in Listing 104 and running the command

```
cmh:~$ latexindent.pl tabbing1.tex -l=delimiterRegEx5.yaml -o=+-mod5
```

■

gives the output in Listing 103.

LISTING 103: tabbing1-mod5.tex

```
\begin{tabbing}
   1   # 22  \> 333   \\
   xxx # aaa  # yyyyy \\
   .   #      # &     \\
\end{tabbing}
```

LISTING 104: delimiterRegEx5.yaml

```
lookForAlignDelims:
   tabbing:
    delimiterRegEx: '(#|\\>)'
    delimiterJustification: right
```

Note that in Listing 103 the second set of delimiters have been *right aligned* – it is quite subtle! ■

3.5.4 lookForAlignDelims: lookForChildCodeBlocks

N: 2021-12-13

There may be scenarios in which you would prefer to instruct `latexindent.pl` *not* to search for child blocks; in which case setting `lookForChildCodeBlocks` to 0 may be a good way to proceed.

example 28 Using the settings from Listing 82 on page 33 on the file in Listing 105 and running the command

```
cmh:~$ latexindent.pl tabular-DM-1.tex -l=dontMeasure1.yaml -o=+-mod1
```

gives the output in Listing 106.

LISTING 105: tabular-DM-1.tex

```
\begin{tabular}{cc}
1&2\only<2->{\\
3&4}
\end{tabular}
```

LISTING 106: tabular-DM-1-mod1.tex

```
\begin{tabular}{cc}
   1 & 2\only<2->{ \\
   3 & 4}
\end{tabular}
```

We can improve the output from Listing 106 by employing the settings in Listing 108

```
cmh:~$ latexindent.pl tabular-DM-1.tex -l=dontMeasure1a.yaml -o=+-mod1a
```

which gives the output in Listing 108.

LISTING 107: tabular-DM-1-mod1a.tex

```
\begin{tabular}{cc}
   1 & 2\only<2->{ \\
   3 & 4}
\end{tabular}
```

LISTING 108: dontMeasure1a.yaml

```
lookForAlignDelims:
   tabular:
      dontMeasure: largest
      lookForChildCodeBlocks: 0
```

■

3.5.5 lookForAlignDelims: alignContentAfterDoubleBackSlash

N: 2023-05-01

You can instruct `latexindent` to align content after the double back slash. See also Section 4.3.2 on page 109.

example 29 We consider the file in Listing 109, and the default output given in Listing 110.

LISTING 109: tabular5.tex

```
\begin{tabular}{cc}
  1 & 2
  \\ aa & bbb
  \\ ccc&ddd
\end{tabular}
```

LISTING 110: tabular5-default.tex

```
\begin{tabular}{cc}
   1 & 2
   \\ aa & bbb
   \\ ccc&ddd
\end{tabular}
```

Using the settings given in Listing 112 and running

```
cmh:~$ latexindent.pl -s tabular5.tex -l alignContentAfterDBS1 -o=+-mod1
```

■

gives the output in Listing 111.

LISTING 111: tabular5-mod1.tex

```
\begin{tabular}{cc}
      1   & 2
   \\ aa  & bbb
   \\ ccc & ddd
\end{tabular}
```

LISTING 112: alignContentAfterDBS1.yaml

```
lookForAlignDelims:
   tabular:
      alignContentAfterDoubleBackSlash: 1
```

■

example 30 When using the `alignContentAfterDoubleBackSlash` feature, then you can also specify how many spaces to insert after the double backslash; the default is 1.

N: 2023-05-01

Starting from Listing 109 and using the the settings given in Listing 114

```
cmh:~$ latexindent.pl -s tabular5.tex -l alignContentAfterDBS2 -o=+-mod2
```

gives the output in Listing 113.

LISTING 113: tabular5-mod2.tex

```
\begin{tabular}{cc}
        1   & 2
   \\   aa  & bbb
   \\   ccc & ddd
\end{tabular}
```

LISTING 114: alignContentAfterDBS2.yaml

```
lookForAlignDelims:
   tabular:
      alignContentAfterDoubleBackSlash: 1
      spacesAfterDoubleBackSlash: 3
```

■

3.5.6 lookForAlignDelims: other

As of Version 3.0, the alignment routine works on mandatory and optional arguments within commands, and also within 'special' code blocks (see `specialBeginEnd` on page 39).

example 31 Assuming that you have a command called `\matrix` and that it is populated within `lookForAlignDelims` (which it is, by default), and that you run the command

```
cmh:~$ latexindent.pl matrix1.tex
```

then the before-and-after results shown in Listings 115 and 116 are achievable by default.

LISTING 115: matrix1.tex

```
\matrix [
   1&2   &3\\
4&5&6]{
7&8   &9\\
10&11&12
}
```

LISTING 116: matrix1.tex default output

```
\matrix [
   1 & 2 & 3 \\
   4 & 5 & 6]{
   7  & 8  & 9  \\
   10 & 11 & 12
}
```

■

If you have blocks of code that you wish to align at the & character that are *not* wrapped in, for example, `\begin{tabular}` ... `\end{tabular}`, then you can use the mark up illustrated in Listing 117; the default output is shown in Listing 118. Note that the `%*` must be next to each other, but that there can be any number of spaces (possibly none) between the `*` and `\begin{tabular}`; note also that you may use any environment name that you have specified in `lookForAlignDelims`.

LISTING 117: align-block.tex

```
%* \begin{tabular}
   1 & 2 & 3 & 4 \\
   5 &   & 6 &   \\
  %* \end{tabular}
```

LISTING 118: align-block.tex default output

```
%* \begin{tabular}
   1 & 2 & 3 & 4 \\
   5 &   & 6 &   \\
%* \end{tabular}
```

With reference to Table 2 on page 49 and the, yet undiscussed, fields of `noAdditionalIndent`

and indentRules (see Section 3.9 on page 50), these comment-marked blocks are considered environments.

3.6 Indent after items, specials and headings

indentAfterItems: ⟨*fields*⟩

The environment names specified in indentAfterItems tell latexindent.pl to look for \item commands; if these switches are set to 1 then indentation will be performed so as indent the code after each item. A demonstration is given in Listings 120 and 121

LISTING 119: indentAfterItems

```
indentAfterItems:
  itemize: 1
  itemize*: 1
  enumerate: 1
  enumerate*: 1
  description: 1
  description*: 1
  list: 1
```

LISTING 120: items1.tex

```
\begin{itemize}
\item some text here
some more text here
some more text here
\item another item
some more text here
\end{itemize}
```

LISTING 121: items1.tex default output

```
\begin{itemize}
   \item some text here
         some more text here
         some more text here
   \item another item
         some more text here
\end{itemize}
```

N: 2026-03-15

If you have your own item commands (perhaps you prefer to use myitem, for example) then you can put adjust itemRegEx in the fineTuning field, discussed in Section 7.3 on page 139.

specialBeginEnd: ⟨*fields*⟩

U: 2017-08-21

N: 2026-03-15

The fields specified in specialBeginEnd are, in their default state, focused on math mode begin and end statements, but there is no requirement for this to be the case; Listing 122 shows the default settings of specialBeginEnd.

LISTING 122: specialBeginEnd

```
specialBeginEnd:
  - amalgamate: 1
  - name: displayMath
    begin: (?<!\\)\\\[            # \[ but *not* \\[
    end: \\\]                     # \]
  - name: inlineMath
    begin: (?<!\$)(?<!\\)\$(?!\$) # $ but *not* \$ or $$
    end: (?<!\\)\$(?!\$)          # $ but *not* \$ or $$
  - name: displayMathTeX
    begin: \$\$                   # $$
    end: \$\$                     # $$
    lookForThis: 1                # 0/1, default 1
    nested: 0                     # 0/1, default 0
```

The field displayMath represents \[...\], inlineMath represents $...$ and displayMathTex represents $$...$$. You can, of course, rename these in your own YAML files (see Section H.2 on page 161); indeed, you might like to set up your own special begin and end statements.

example 32 A demonstration of the before-and-after results are shown in Listings 123 and 124; explicitly, running the command

```
cmh:~$ latexindent.pl special1.tex -o=+-default
```

gives the output given in Listing 124. ■

LISTING 123: special1.tex before

```
The function $f$ has formula
\[
f(x)=x^2.
\]
If you like splitting dollars,
$
g(x)=f(2x)
$
```

LISTING 124: special1.tex default output

```
The function $f$ has formula
\[
   f(x)=x^2.
\]
If you like splitting dollars,
$
   g(x)=f(2x)
$
```

■

For each field, `lookForThis` is set to `1` by default, which means that `latexindent.pl` will look for this pattern; you can tell `latexindent.pl` not to look for the pattern, by setting `lookForThis` to `0`.

example 33 For example, consider the file shown in Listing 125.

LISTING 125: specialLR.tex

```
\begin{equation}
\left[
\sqrt{
a+b
}
\right]
\end{equation}
```

Now consider the YAML file shown in Listing 126

LISTING 126: specialsLeftRight.yaml

```
specialBeginEnd:
  - name: leftRightSquare
    begin: \\left\[
    end: \\right\]

fineTuning:
  commands:
    name: |-
      (?x)
      (?!         #  NOT
        (?:       #
          begin   #
          |       #
          end     #
          |       #
          left    #
          |       #
          right   #
          )       #
      )[+a-zA-Z@*0-9_:]+
```

Upon running the following commands

```
cmh:~$ latexindent.pl specialLR.tex -l=specialsLeftRight.yaml
```

we receive the output in Listing 127.

■

LISTING 127: specialLR.tex using Listing 126

```
\begin{equation}
   \left[
      \sqrt{
         a+b
      }
   \right]
\end{equation}
```

N: 2018-04-27

You can, optionally, specify the `middle` field for anything that you specify in `specialBeginEnd`.

example 34 For example, let's consider the `.tex` file in Listing 128.

LISTING 128: special2.tex

```
\If
something 0
\ElsIf
something 1
\ElsIf
something 2
\ElsIf
something 3
\Else
something 4
\EndIf
```

Upon saving the YAML settings in Listings 130 and 132 and running the commands

```
cmh:~$ latexindent.pl special2.tex -l=middle
cmh:~$ latexindent.pl special2.tex -l=middle1
```

then we obtain the output given in Listings 129 and 131.

LISTING 129: special2.tex using Listing 130

```
\If
   something 0
\ElsIf
   something 1
\ElsIf
   something 2
\ElsIf
   something 3
   \Else
   something 4
\EndIf
```

LISTING 130: middle.yaml

```
specialBeginEnd:
  - name: If
    begin: '\\If'
    middle: '\\ElsIf'
    end: '\\EndIf'
    lookForThis: 1
```

LISTING 131: special2.tex using Listing 132

```
\If
   something 0
\ElsIf
   something 1
\ElsIf
   something 2
\ElsIf
   something 3
\Else
   something 4
\EndIf
```

LISTING 132: middle1.yaml

```
specialBeginEnd:
  - name: If
    begin: '\\If'
    middle:
      - '\\ElsIf'
      - '\\Else'
    end: '\\EndIf'
    lookForThis: 1
```

We note that:

- in Listing 129 the bodies of each of the `Elsif` statements have been indented appropriately;
- the `Else` statement has *not* been indented appropriately in Listing 129 – read on!
- we have specified multiple settings for the `middle` field using the syntax demonstrated in Listing 132 so that the body of the `Else` statement has been indented appropriately in Listing 131.

■

You may need these fields in your own YAML files (see Section H.2 on page 161), if you use popular algorithm packages such as algorithms, algorithm2e or algpseudocode, etc.

example 35 For example, let's consider the `.tex` file in Listing 133.

LISTING 133: specialAlgo.tex

```
\For{$n = 1, \dots, 10$}
\State body
\EndFor
\FOR{for 1}
\FOR{for 2}
\FOR{for 3}
\STATE{some statement.}
\ENDFOR
\ENDFOR
\ENDFOR
\If{$quality\ge 9$}
\State $a\gets perfect$
\ElsIf{$quality\ge 7$}
\State $a\gets good$
\Else
\While{$i\le n$}
\State $i\gets i+1$
\EndWhile
\EndIf
\ForAll{$n \in \{1, \dots, 10\}$}
\State body
\Loop
\State body
\EndLoop
\State $i\gets 1$
\Repeat
\State $i\gets i+1$
\Until{$i>n$}
\EndFor
\Function{Euclid}{$a,b$}\Comment{The g.c.d. of a and b}
\While{$r\not=0$}\Comment{We have the answer if r is 0}
\State $r\gets a\bmod b$
\EndWhile
\State \textbf{return} $b$\Comment{The gcd is b}
\EndFunction
```

Upon saving the YAML settings in Listing 135 and running the command

```
cmh:~$ latexindent.pl -l=algo.yaml specialAlgo.tex
```

then we obtain the output given in Listing 134.

■

LISTING 134: specialAlgo.tex using Listing 135

```
\For{$n = 1, \dots, 10$}
  \State body
\EndFor
\FOR{for 1}
  \FOR{for 2}
    \FOR{for 3}
      \STATE{some statement.}
    \ENDFOR
  \ENDFOR
\ENDFOR
\If{$quality\ge 9$}
  \State $a\gets perfect$
\ElsIf{$quality\ge 7$}
  \State $a\gets good$
\Else
  \While{$i\le n$}
    \State $i\gets i+1$
  \EndWhile
\EndIf
\ForAll{$n \in \{1, \dots, 10\}$}
  \State body
  \Loop
    \State body
  \EndLoop
  \State $i\gets 1$
  \Repeat
    \State $i\gets i+1$
  \Until{$i>n$}
\EndFor
\Function{Euclid}{$a,b$}\Comment{The g.c.d. of a and b}
  \While{$r\not=0$}\Comment{We have the answer if r is 0}
    \State $r\gets a\bmod b$
  \EndWhile
  \State \textbf{return} $b$\Comment{The gcd is b}
\EndFunction
```

LISTING 135: algo.yaml

```
specialBeginEnd:
 - name: ForStatement
  begin: \\For\{[^}]+?\}
  end: \\EndFor
 - name: FORStatement
  begin: \\FOR\{[^}]+?\}
  end: \\ENDFOR
  nested: 1
 - name: WhileStatement
  begin: \\While\{[^}]+?\}
  end: \\EndWhile
  nested: 1
 - name: WHILEStatement
  begin: \\WHILE\{[^}]+?\}
  end: \\ENDWHILE
 - name: ForAllStatement
  begin: \\ForAll\{[^}]+?\}
  end: \\EndFor
  nested: 1
 - name: LoopStatement
  begin: \\Loop
  end: \\EndLoop
 - name: RepeatStatement
  begin: \\Repeat
  end: \\Until\{[^}]+?\}
 - name: ProcedureStatement
  begin: \\Procedure\{[^}]+?\}\{[^}]+?\}
  end: \\EndProcedure
 - name: FunctionStatement
  begin: \\Function\{[^}]+?\}\{[^}]+?\}
  end: \\EndFunction
 - name: IfStatement
  begin: \\If\{[^}]+?\}
  middle:
   - \\Else
   - \\ElsIf\{[^}]+?\}
  end: \\EndIf
 - name: IFStatement
  begin: \\IF\{[^}]+?\}
  middle:
   - \\ELSE
   - \\ELSIF\{[^}]+?\}
  end: \\ENDIF
 - name: inlineMath
  lookForThis: 0

fineTuning:
 commands:
  name: |-
   (?x)
   (?!
    (?:
     begin
     |
     end
     |
     ElsIf
     |
     EndFor
     |
     ENDFOR
     |
     EndWhile
     |
     ENDWHILE
     |
     Function
     |
     For
     |
     FOR
     |
     If
     |
     IF
     |
     Loop
     |
     Procedure
     |
     Repeat
     |
     While
    )
   )[+a-zA-Z@*0-9_:]+
```

N: 2018-08-13

You may specify fields in `specialBeginEnd` to be treated as verbatim code blocks by changing `lookForThis` to be `verbatim`.

example 36 For example, beginning with the code in Listing 136 and the YAML in Listing 137, and running

```
cmh:~$ latexindent.pl special3.tex -l=special-verb1
```

then the output in Listing 136 is unchanged.

LISTING 136: special3.tex and output using Listing 137

```
\[
  special code
blocks
    can be
  treated
    as verbatim\]
```

LISTING 137: special-verb1.yaml

```
specialBeginEnd:
  - name: displayMath
    lookForThis: verbatim
```

We can combine the specialBeginEnd with the lookForAlignDelims feature.

example 37 We begin with the code in Listing 138.

LISTING 138: special-align.tex

```
\begin{tikzpicture}
  \path (A) edge node {0,1,L}(B)
  edge node {1,1,R} (C)
  (B) edge [loop above]node {1,1,L}(B)
  edge node {0,1,L}(C)
  (C) edge node {0,1,L}(D)
  edge [bend left]node {1,0,R}(E)
  (D) edge[loop below] node {1,1,R}(D)
  edge node {0,1,R}(A)
  (E) edge[bend left] node {1,0,R} (A);
\end{tikzpicture}
```

Let's assume that our goal is to align the code at the edge and node text; we employ the code given in Listing 140 and run the command

```
cmh:~$ latexindent.pl special-align.tex -l edge-node1.yaml -o=+-mod1
```

to receive the output in Listing 139.

LISTING 139: special-align.tex using Listing 140

```
\begin{tikzpicture}
   \path (A) edge               node {0,1,L}(B)
             edge               node {1,1,R} (C)
         (B) edge [loop above] node {1,1,L}(B)
             edge               node {0,1,L}(C)
         (C) edge               node {0,1,L}(D)
             edge [bend left]   node {1,0,R}(E)
         (D) edge [loop below] node {1,1,R}(D)
             edge               node {0,1,R}(A)
         (E) edge [bend left]   node {1,0,R} (A);
\end{tikzpicture}
```

LISTING 140: edge-node1.yaml

```
specialBeginEnd:
  - name: path
    begin: '\\path'
    end: ';'
lookForAlignDelims:
   path:
      delimiterRegEx: '(edge|node)'
      lookForChildCodeBlocks: 0
```

The output in Listing 139 is not quite ideal. We can tweak the settings within Listing 140 in order to improve the output; in particular, we employ the code in Listing 142 and run the command

```
cmh:~$ latexindent.pl special-align.tex -l edge-node2.yaml -o=+-mod2
```

to receive the output in Listing 141.

LISTING 141: special-align.tex using Listing 142

```
\begin{tikzpicture}
   \path (A) edge               node {0,1,L} (B)
             edge               node {1,1,R} (C)
         (B) edge [loop above]  node {1,1,L} (B)
             edge               node {0,1,L} (C)
         (C) edge               node {0,1,L} (D)
             edge [bend left]   node {1,0,R} (E)
         (D) edge [loop below]  node {1,1,R} (D)
             edge               node {0,1,R} (A)
         (E) edge [bend left]   node {1,0,R} (A);
\end{tikzpicture}
```

LISTING 142: edge-node2.yaml

```
specialBeginEnd:
  -
    name: path
    begin: '\\path'
    end: ';'

lookForAlignDelims:
    path:
       delimiterRegEx:
     '(edge|node\h*\{[0-9,A-Z]+\})'
       lookForChildCodeBlocks: 0
```

U: 2021-06-19

The `lookForThis` field can be considered optional; by default, it is assumed to be 1, which is demonstrated in Listing 142.

N: 2023-09-23

Referencing Listing 122 on page 39 we see that each of the `specialBeginEnd` fields can *optionally* accept the `body` field. If the `body` field is omitted, then `latexindent.pl` uses a value that means

> anything except one of the begin statements from `specialBeginEnd`.

In general, it is usually *not* necessary to specify the `body` field, but let's detail an example just for reference.

example 38 We begin with the example in Listing 143

LISTING 143: special-body.tex

```
$
a
+
(
b + c
-
(
    d
)
)
=
e
$
and
$
f + g = h
$
```

Using the settings in Listing 145 and running the command

```
cmh:~$ latexindent.pl special-body.tex -l=special-nested1.yaml
```

gives the output in Listing 144.

LISTING 144: special-body.tex using Listing 145

```
$
  a
  +
  (
    b + c
    -
    (
      d
    )
  )
  =
  e
$
and
$
  f + g = h
$
```

LISTING 145: special-nested1.yaml

```
defaultIndent: "  "
specialBeginEnd:
  - name: parentheses
    begin: \(
    end: \)
    nested: 1
```

N: 2026-03-15

We note that the output in Listing 144 is as we would expect, as the `nested` field has been specified in Listing 145. ■

`indentAfterHeadings`: ⟨*fields*⟩

This field enables the user to specify indentation rules that take effect after heading commands such as `\part`, `\chapter`, `\section`, `\subsection*`, or indeed any user-specified command written in this field.

LISTING 146: indentAfterHeadings

```
211 indentAfterHeadings:
212   part:
213     lookForThis: 0
214     level: 1
215   chapter:
216     lookForThis: 0
217     level: 2
218   section:
219     lookForThis: 0
220     level: 3
```

The default settings do *not* place indentation after a heading, but you can easily switch them on by changing `lookForThis` from 0 to 1. The `level` field tells `latexindent.pl` the hierarchy of the heading structure in your document. You might, for example, like to have both `section` and `subsection` set with `level:  3` because you do not want the indentation to go too deep.

You can add any of your own custom heading commands to this field, specifying the `level` as appropriate. You can also specify your own indentation in `indentRules` (see Section 3.9 on page 50); you will find the default `indentRules` contains `chapter: " "` which tells `latexindent.pl` simply to use a space character after `chapter` headings (once `indent` is set to 1 for `chapter`).

example 39 For example, assuming that you have the code in Listing 148 saved into `headings1.yaml`, and that you have the text from Listing 147 saved into `headings1.tex`. ■

LISTING 147: headings1.tex

```
\subsection{subsection title}
subsection text
subsection text
\paragraph{paragraph title}
paragraph text
paragraph text
\paragraph{paragraph title}
paragraph text
paragraph text
```

LISTING 148: headings1.yaml

```
indentAfterHeadings:
    subsection:
       lookForThis: 1
       level: 1
    paragraph:
       lookForThis: 1
       level: 2
```

If you run the command

```
cmh:~$ latexindent.pl headings1.tex -l=headings1.yaml
```

then you should receive the output given in Listing 149.

LISTING 149: headings1.tex using Listing 148

```
\subsection{subsection title}
    subsection text
    subsection text
    \paragraph{paragraph title}
        paragraph text
        paragraph text
    \paragraph{paragraph title}
        paragraph text
        paragraph text
```

LISTING 150: headings1.tex second modification

```
\subsection{subsection title}
    subsection text
    subsection text
\paragraph{paragraph title}
    paragraph text
    paragraph text
\paragraph{paragraph title}
    paragraph text
    paragraph text
```

Now say that you modify the YAML from Listing 148 so that the paragraph level is 1; after running

```
cmh:~$ latexindent.pl headings1.tex -l=headings1.yaml
```

you should receive the code given in Listing 150; notice that the paragraph and subsection are at the same indentation level.

maximumIndentation: ⟨*horizontal space*⟩

N: 2017-08-21

You can control the maximum indentation given to your file by specifying the maximumIndentation field as horizontal space (but *not* including tabs). This feature uses the Text::Tabs module [47], and is *off* by default.

example 40 For example, consider the example shown in Listing 151 together with the default output shown in Listing 152.

LISTING 151: mult-nested.tex

```
\begin{one}
one
\begin{two}
    two
\begin{three}
     three
\begin{four}
       four
\end{four}
\end{three}
\end{two}
\end{one}
```

LISTING 152: mult-nested.tex default output

```
\begin{one}
   one
   \begin{two}
      two
      \begin{three}
         three
         \begin{four}
            four
         \end{four}
      \end{three}
   \end{two}
\end{one}
```

■

example 41 Now say that, for example, you have the `max-indentation1.yaml` from Listing 154 and that you run the following command:

```
cmh:~$ latexindent.pl mult-nested.tex -l=max-indentation1
```

You should receive the output shown in Listing 153.

LISTING 153: mult-nested.tex using Listing 154

```
\begin{one}
␣one
␣\begin{two}
␣two
␣\begin{three}
␣three
␣\begin{four}
␣four
␣\end{four}
␣\end{three}
␣\end{two}
\end{one}
```

LISTING 154: max-indentation1.yaml

```
maximumIndentation: " "
```

■

Comparing the output in Listings 152 and 153 we notice that the (default) tabs of indentation have been replaced by a single space.

In general, when using the `maximumIndentation` feature, any leading tabs will be replaced by equivalent spaces except, of course, those found in `verbatimEnvironments` (see Listing 27 on page 22) or `noIndentBlock` (see Listing 33 on page 23).

3.7 Arguments and the strings allowed between them

The strings allowed between arguments for `commands`, `namedGroupingBracesBrackets`, `keyEqualsValuesBracesB` and `UnNamedGroupingBracesBrackets` are controlled by `fineTuning`; specific demonstrations are given in Section 7.4 on page 139.

3.8 The code blocks known to latexindent.pl

As of Version 3.0, `latexindent.pl` processes documents using code blocks; each of these are shown in Table 2.

N: 2019-07-13

We will refer to these code blocks in what follows. Note that the fine tuning of the definition of the code blocks detailed in Table 2 is discussed in Section 7 on page 134.

TABLE 2: Code blocks known to `latexindent.pl`

Code block	characters allowed in name	example

environments	`a-zA-Z@\*0-9_\\`	`\begin{myenv}`
		`body of myenv`
		`\end{myenv}`

optionalArguments	*inherits* name from parent (e.g environment name)	`[`
		`opt arg text`
		`]`

mandatoryArguments	*inherits* name from parent (e.g environment name)	`{`
		`mand arg text`
		`}`

commands	`+a-zA-Z@\*0-9_\:`	`\mycommand`⟨*arguments*⟩
keyEqualsValuesBracesBrackets	`a-zA-Z@\*0-9_\/.\h\{\}:\#-`	`my key/.style=`⟨*arguments*⟩
namedGroupingBracesBrackets	`0-9\.a-zA-Z@\*><`	`in`⟨*arguments*⟩
UnNamedGroupingBracesBrackets	*No name!*	`{` or `[` or `,` or `\&` or `)` or `(` or `$` followed by ⟨*arguments*⟩

ifElseFi	`@a-zA-Z` but must begin with either `\if` of `\@if`	`\ifnum...`
		`...`
		`\else`
		`...`
		`\fi`

items	User specified, see Listing 119 on page 39	`\begin{enumerate}`
		`  \item ...`
		`\end{enumerate}`

specialBeginEnd	User specified, see Listing 122 on page 39	`\[`
		`  ...`
		`\]`

afterHeading	User specified, see Listing 146 on page 46	`\chapter{title}`
		`  ...`
		`\section{title}`

filecontents	User specified, see Listing 43 on page 25	`\begin{filecontents}`
		`...`
		`\end{filecontents}`

3.9 noAdditionalIndent and indentRules

`latexindent.pl` operates on files by looking for code blocks, as detailed in Section 3.8 on page 48; for each type of code block in Table 2 on the previous page (which we will call a ⟨*thing*⟩ in what follows) it searches YAML fields for information in the following order:

1. `noAdditionalIndent` for the *name* of the current ⟨*thing*⟩;
2. `indentRules` for the *name* of the current ⟨*thing*⟩;
3. `noAdditionalIndentGlobal` for the *type* of the current ⟨*thing*⟩;
4. `indentRulesGlobal` for the *type* of the current ⟨*thing*⟩.

Using the above list, the first piece of information to be found will be used; failing that, the value of `defaultIndent` is used. If information is found in multiple fields, the first one according to the list above will be used; for example, if information is present in both `indentRules` and in `noAdditionalIndentGlobal`, then the information from `indentRules` takes priority.

We now present details for the different type of code blocks known to `latexindent.pl`, as detailed in Table 2 on the preceding page; for reference, there follows a list of the code blocks covered.

3.9.1 Environments and their arguments

There are a few different YAML switches governing the indentation of environments; let's start with the code shown in Listing 155.

LISTING 155: `myenv.tex`

```
\begin{outer}
\begin{myenv}
  body of environment
body of environment
     body of environment
\end{myenv}
\end{outer}
```

`noAdditionalIndent`: ⟨*fields*⟩

example 42 If we do not wish `myenv` to receive any additional indentation, we have a few choices available to us, as demonstrated in Listings 156 and 157. ■

LISTING 156: myenv-noAdd1.yaml

```
noAdditionalIndent:
    myenv: 1
```

LISTING 157: myenv-noAdd2.yaml

```
noAdditionalIndent:
    myenv:
        body: 1
```

On applying either of the following commands,

```
cmh:~$ latexindent.pl myenv.tex -l myenv-noAdd1.yaml
cmh:~$ latexindent.pl myenv.tex -l myenv-noAdd2.yaml
```

we obtain the output given in Listing 158; note in particular that the environment myenv has not received any *additional* indentation, but that the outer environment *has* still received indentation.

LISTING 158: myenv.tex output (using either Listing 156 or Listing 157)

```
\begin{outer}
   \begin{myenv}
   body of environment
   body of environment
   body of environment
   \end{myenv}
\end{outer}
```

■

example 43 Upon changing the YAML files to those shown in Listings 159 and 160, and running either

```
cmh:~$ latexindent.pl myenv.tex -l myenv-noAdd3.yaml
cmh:~$ latexindent.pl myenv.tex -l myenv-noAdd4.yaml
```

we obtain the output given in Listing 161.

LISTING 159: myenv-noAdd3.yaml

```
noAdditionalIndent:
    myenv: 0
```

LISTING 160: myenv-noAdd4.yaml

```
noAdditionalIndent:
    myenv:
        body: 0
```

LISTING 161: myenv.tex output (using either Listing 159 or Listing 160)

```
\begin{outer}
   \begin{myenv}
      body of environment
      body of environment
      body of environment
   \end{myenv}
\end{outer}
```

■

example 44 Let's now allow myenv to have some optional and mandatory arguments, as in Listing 162. ■

LISTING 162: myenv-args.tex

```
\begin{outer}
\begin{myenv}[%
  optional argument text
        optional argument text]%
  { mandatory argument text
 mandatory argument text}
  body of environment
body of environment
     body of environment
\end{myenv}
\end{outer}
```

Upon running

```
cmh:~$ latexindent.pl -l=myenv-noAdd1.yaml myenv-args.tex
```

we obtain the output shown in Listing 163; note that the optional argument, mandatory argument and body *all* have received no additional indent. This is because, when `noAdditionalIndent` is specified in 'scalar' form (as in Listing 156), then *all* parts of the environment (body, optional and mandatory arguments) are assumed to want no additional indent.

LISTING 163: myenv-args.tex using Listing 156

```
\begin{outer}
   \begin{myenv}[%
   optional argument text
   optional argument text]%
   { mandatory argument text
   mandatory argument text}
   body of environment
   body of environment
   body of environment
   \end{myenv}
\end{outer}
```

■

example 45 We may customise `noAdditionalIndent` for optional and mandatory arguments of the `myenv` environment, as shown in, for example, Listings 164 and 165.

LISTING 164: myenv-noAdd5.yaml

```
noAdditionalIndent:
    myenv:
        body: 0
        optionalArguments: 1
        mandatoryArguments: 0
```

LISTING 165: myenv-noAdd6.yaml

```
noAdditionalIndent:
    myenv:
        body: 0
        optionalArguments: 0
        mandatoryArguments: 1
```

Upon running

```
cmh:~$ latexindent.pl myenv.tex -l myenv-noAdd5.yaml
cmh:~$ latexindent.pl myenv.tex -l myenv-noAdd6.yaml
```

we obtain the respective outputs given in Listings 166 and 167. Note that in Listing 166 the text for the *optional* argument has not received any additional indentation, and that in Listing 167 the *mandatory* argument has not received any additional indentation; in both cases, the *body* has not received any additional indentation.

■

LISTING 166: myenv-args.tex using Listing 164

```
\begin{outer}
   \begin{myenv}[%
      optional argument text
      optional argument text]%
      { mandatory argument text
         mandatory argument text}
      body of environment
      body of environment
      body of environment
   \end{myenv}
\end{outer}
```

LISTING 167: myenv-args.tex using Listing 165

```
\begin{outer}
   \begin{myenv}[%
         optional argument text
         optional argument text]%
      { mandatory argument text
      mandatory argument text}
      body of environment
      body of environment
      body of environment
   \end{myenv}
\end{outer}
```

■

indentRules: ⟨*fields*⟩

example 46 We may also specify indentation rules for environment code blocks using the `indentRules` field; see, for example, Listings 168 and 169.

LISTING 168: myenv-rules1.yaml

```
indentRules:
    myenv: "    "
```

LISTING 169: myenv-rules2.yaml

```
indentRules:
    myenv:
        body: "    "
```

On applying either of the following commands,

```
cmh:~$ latexindent.pl myenv.tex -l myenv-rules1.yaml
cmh:~$ latexindent.pl myenv.tex -l myenv-rules2.yaml
```

we obtain the output given in Listing 170; note in particular that the environment `myenv` has received one tab (from the `outer` environment) plus three spaces from Listing 168 or 169.

LISTING 170: myenv.tex output (using either Listing 168 or Listing 169)

```
\begin{outer}
___\begin{myenv}
___␣␣␣␣body␣of␣environment
___␣␣␣␣body␣of␣environment
___␣␣␣␣body␣of␣environment
___\end{myenv}
\end{outer}
```

■

If you specify a field in `indentRules` using anything other than horizontal space, it will be ignored.

example 47 Returning to the example in Listing 162 that contains optional and mandatory arguments. Upon using Listing 168 as in

```
cmh:~$ latexindent.pl myenv-args.tex -l=myenv-rules1.yaml
```

we obtain the output in Listing 171; note that the body, optional argument and mandatory argument of `myenv` have *all* received the same customised indentation.

■

LISTING 171: myenv-args.tex using Listing 168

```
\begin{outer}
    \begin{myenv}[%
        optional argument text
        optional argument text]%
        { mandatory argument text
        mandatory argument text}
        body of environment
        body of environment
        body of environment
    \end{myenv}
\end{outer}
```

■

example 48 You can specify different indentation rules for the different features using, for example, Listings 172 and 173

LISTING 172: myenv-rules3.yaml

```
indentRules:
    myenv:
        body: "    "
        optionalArguments: " "
```

LISTING 173: myenv-rules4.yaml

```
indentRules:
    myenv:
        body: "    "
        mandatoryArguments:
    "\t\t"
```

After running

```
cmh:~$ latexindent.pl myenv-args.tex -l myenv-rules3.yaml
cmh:~$ latexindent.pl myenv-args.tex -l myenv-rules4.yaml
```

then we obtain the respective outputs given in Listings 174 and 175.

LISTING 174: myenv-args.tex using Listing 172

```
\begin{outer}
    \begin{myenv}[%
        optional argument text
        optional argument text]%
        { mandatory argument text
            mandatory argument text}
        body of environment
        body of environment
        body of environment
    \end{myenv}
\end{outer}
```

LISTING 175: myenv-args.tex using Listing 173

```
\begin{outer}
    \begin{myenv}[%
        optional argument text
        optional argument text]%
        { mandatory argument text
            mandatory argument text}
        body of environment
        body of environment
        body of environment
    \end{myenv}
\end{outer}
```

Note that in Listing 174, the optional argument has only received a single space of indentation, while the mandatory argument has received the default (tab) indentation; the environment body has received three spaces of indentation.

In Listing 175, the optional argument has received the default (tab) indentation, the mandatory argument has received two tabs of indentation, and the body has received three spaces of indentation.

■

`noAdditionalIndentGlobal`: ⟨*fields*⟩

Assuming that your environment name is not found within neither `noAdditionalIndent` nor `indentRules`, the next place that `latexindent.pl` will look is `noAdditionalIndentGlobal`, and in particular *for the environments* key (see Listing 176).

LISTING 176: noAdditionalIndentGlobal

```
260 noAdditionalIndentGlobal:
261   environments: 0                    # 0/1
```

example 49 Let's say that you change the value of `environments` to `1` in Listing 176, and that you run

```
cmh:~$ latexindent.pl myenv-args.tex -l env-noAdditionalGlobal.yaml
cmh:~$ latexindent.pl myenv-args.tex -l myenv-rules1.yaml,env-noAdditionalGlobal.yaml
```

The respective output from these two commands are in Listings 177 and 178; in Listing 177 notice that *both* environments receive no additional indentation but that the arguments of `myenv` still *do* receive indentation. In Listing 178 notice that the *outer* environment does not receive additional indentation, but because of the settings from `myenv-rules1.yaml` (in Listing 168 on page 53), the `myenv` environment still *does* receive indentation.

LISTING 177: myenv-args.tex using Listing 176

```
\begin{outer}
\begin{myenv}[%
   optional argument text
   optional argument text]%
{ mandatory argument text
   mandatory argument text}
body of environment
body of environment
body of environment
\end{myenv}
\end{outer}
```

LISTING 178: myenv-args.tex using Listings 168 and 176

```
\begin{outer}
\begin{myenv}[%
      optional argument text
      optional argument text]%
   { mandatory argument text
      mandatory argument text}
   body of environment
   body of environment
   body of environment
\end{myenv}
\end{outer}
```

■

example 50 In fact, `noAdditionalIndentGlobal` also contains keys that control the indentation of optional and mandatory arguments; on referencing Listings 179 and 180

LISTING 179: opt-args-no-add-glob.yaml

```
noAdditionalIndentGlobal:
    optionalArguments: 1
```

LISTING 180: mand-args-no-add-glob.yaml

```
noAdditionalIndentGlobal:
    mandatoryArguments: 1
```

we may run the commands

```
cmh:~$ latexindent.pl myenv-args.tex -local opt-args-no-add-glob.yaml
cmh:~$ latexindent.pl myenv-args.tex -local mand-args-no-add-glob.yaml
```

which produces the respective outputs given in Listings 181 and 182. Notice that in Listing 181 the *optional* argument has not received any additional indentation, and in Listing 182 the *mandatory* argument has not received any additional indentation.

■

LISTING 181: myenv-args.tex using Listing 179

```
\begin{outer}
   \begin{myenv}[%
      optional argument text
      optional argument text]%
      { mandatory argument text
         mandatory argument text}
      body of environment
      body of environment
      body of environment
   \end{myenv}
\end{outer}
```

LISTING 182: myenv-args.tex using Listing 180

```
\begin{outer}
   \begin{myenv}[%
         optional argument text
         optional argument text]%
      { mandatory argument text
      mandatory argument text}
      body of environment
      body of environment
      body of environment
   \end{myenv}
\end{outer}
```

■

`indentRulesGlobal`: ⟨*fields*⟩

The final check that `latexindent.pl` will make is to look for `indentRulesGlobal` as detailed in Listing 183.

LISTING 183: indentRulesGlobal

```
275 indentRulesGlobal:
276   environments: 0                  # 0/h-space
```

example 51 If you change the `environments` field to anything involving horizontal space, say `" "`, and then run the following commands

```
cmh:~$ latexindent.pl myenv-args.tex -l env-indentRules.yaml
cmh:~$ latexindent.pl myenv-args.tex -l myenv-rules1.yaml,env-indentRules.yaml
```

then the respective output is shown in Listings 184 and 185. Note that in Listing 184, both the environment blocks have received a single-space indentation, whereas in Listing 185 the `outer` environment has received single-space indentation (specified by `indentRulesGlobal`), but `myenv` has received `"  "`, as specified by the particular `indentRules` for `myenv` Listing 168 on page 53.

LISTING 184: myenv-args.tex using Listing 183

```
\begin{outer}
␣\begin{myenv}[%
␣␣␣␣␣optional␣argument␣text
␣␣␣␣␣optional␣argument␣text]%
␣␣{␣mandatory␣argument␣text
␣␣␣␣␣mandatory␣argument␣text}
␣␣body␣of␣environment
␣␣body␣of␣environment
␣␣body␣of␣environment
␣\end{myenv}
\end{outer}
```

LISTING 185: myenv-args.tex using Listings 168 and 183

```
\begin{outer}
␣\begin{myenv}[%
␣␣␣␣␣␣␣optional␣argument␣text
␣␣␣␣␣␣␣optional␣argument␣text]%
␣␣␣␣{␣mandatory␣argument␣text
␣␣␣␣␣␣␣mandatory␣argument␣text}
␣␣␣␣body␣of␣environment
␣␣␣␣body␣of␣environment
␣␣␣␣body␣of␣environment
␣\end{myenv}
\end{outer}
```

■

example 52 You can specify `indentRulesGlobal` for both optional and mandatory arguments, as detailed in Listings 186 and 187

■

LISTING 186: opt-args-indent-rules-glob.yaml

```
indentRulesGlobal:
    optionalArguments: "\t\t"
```

LISTING 187: mand-args-indent-rules-glob.yaml

```
indentRulesGlobal:
    mandatoryArguments: "\t\t"
```

Upon running the following commands

```
cmh:~$ latexindent.pl myenv-args.tex -local opt-args-indent-rules-glob.yaml
cmh:~$ latexindent.pl myenv-args.tex -local mand-args-indent-rules-glob.yaml
```

we obtain the respective outputs in Listings 188 and 189. Note that the *optional* argument in Listing 188 has received two tabs worth of indentation, while the *mandatory* argument has done so in Listing 189.

LISTING 188: myenv-args.tex using Listing 186

```
\begin{outer}
    \begin{myenv}[%
        optional argument text
        optional argument text]%
    { mandatory argument text
        mandatory argument text}
    body of environment
    body of environment
    body of environment
    \end{myenv}
\end{outer}
```

LISTING 189: myenv-args.tex using Listing 187

```
\begin{outer}
    \begin{myenv}[%
        optional argument text
        optional argument text]%
    { mandatory argument text
        mandatory argument text}
    body of environment
    body of environment
    body of environment
    \end{myenv}
\end{outer}
```

■

3.9.2 Environments with items

With reference to Listing 119 on page 39, some commands may contain `item` commands; for the purposes of this discussion, we will use the code from Listing 120 on page 39.

Assuming that you've populated `itemNames` with the name of your `item`, you can put the item name into `noAdditionalIndent` as in Listing 190, although a more efficient approach may be to change the relevant field in `itemNames` to `0`.

example 53 Similarly, you can customise the indentation that your `item` receives using `indentRules`, as in Listing 191

LISTING 190: item-noAdd1.yaml

```
noAdditionalIndent:
    item: 1
```

LISTING 191: item-rules1.yaml

```
indentRules:
    item: " "
```

Upon running the following commands

```
cmh:~$ latexindent.pl items1.tex -local item-noAdd1.yaml
cmh:~$ latexindent.pl items1.tex -local item-rules1.yaml
```

the respective outputs are given in Listings 192 and 193; note that in Listing 192 that the text after each `item` has not received any additional indentation, and in Listing 193, the text after each `item` has received a single space of indentation, specified by Listing 191.

■

LISTING 192: items1.tex using Listing 190

```
\begin{itemize}
   \item some text here
   some more text here
   some more text here
   \item another item
   some more text here
\end{itemize}
```

LISTING 193: items1.tex using Listing 191

```
\begin{itemize}
___\item␣some␣text␣here
___␣some␣more␣text␣here
___␣some␣more␣text␣here
___\item␣another␣item
___␣some␣more␣text␣here
\end{itemize}
```

■

example 54 Alternatively, you might like to populate `noAdditionalIndentGlobal` or `indentRulesGlobal` using the `items` key, as demonstrated in Listings 194 and 195. Note that there is a need to 'reset/remove' the `item` field from `indentRules` in both cases (see the hierarchy description given on page 50) as the `item` command is a member of `indentRules` by default.

LISTING 194: items-noAdditionalGlobal.yaml

```
indentRules:
    item: 0
noAdditionalIndentGlobal:
    items: 1
```

LISTING 195: items-indentRulesGlobal.yaml

```
indentRules:
    item: 0
indentRulesGlobal:
    items: " "
```

Upon running the following commands,

```
cmh:~$ latexindent.pl items1.tex -local items-noAdditionalGlobal.yaml
cmh:~$ latexindent.pl items1.tex -local items-indentRulesGlobal.yaml
```

the respective outputs from Listings 192 and 193 are obtained; note, however, that *all* such `item` commands without their own individual `noAdditionalIndent` or `indentRules` settings would behave as in these listings.

■

3.9.3 Commands with arguments

example 55 Let's begin with the simple example in Listing 196; when `latexindent.pl` operates on this file, the default output is shown in Listing 197.

LISTING 196: mycommand.tex

```
\mycommand
{
mand arg text
mand arg text}
[
opt arg text
opt arg text
]
```

LISTING 197: mycommand.tex default output

```
\mycommand
{
   mand arg text
   mand arg text}
[
   opt arg text
   opt arg text
]
```

As in the environment-based case (see Listings 156 and 157 on page 51) we may specify `noAdditionalIndent` either in 'scalar' form, or in 'field' form, as shown in Listings 198 and 199

LISTING 198: mycommand-noAdd1.yaml

```
noAdditionalIndent:
    mycommand: 1
```

LISTING 199: mycommand-noAdd2.yaml

```
noAdditionalIndent:
    mycommand:
        body: 1
```

After running the following commands,

■

```
cmh:~$ latexindent.pl mycommand.tex -l mycommand-noAdd1.yaml
cmh:~$ latexindent.pl mycommand.tex -l mycommand-noAdd2.yaml
```

we receive the respective output given in Listings 200 and 201

LISTING 200: mycommand.tex using Listing 198

```
\mycommand
{
mand arg text
mand arg text}
[
opt arg text
opt arg text
]
```

LISTING 201: mycommand.tex using Listing 199

```
\mycommand
{
   mand arg text
   mand arg text}
[
   opt arg text
   opt arg text
]
```

Note that in Listing 200 that the 'body', optional argument *and* mandatory argument have *all* received no additional indentation, while in Listing 201, only the 'body' has not received any additional indentation. We define the 'body' of a command as any lines following the command name that include its optional or mandatory arguments. ■

example 56 We may further customise `noAdditionalIndent` for `mycommand` as we did in Listings 164 and 165 on page 52; explicit examples are given in Listings 202 and 203.

LISTING 202: mycommand-noAdd3.yaml

```
noAdditionalIndent:
    mycommand:
        body: 0
        optionalArguments: 1
        mandatoryArguments: 0
```

LISTING 203: mycommand-noAdd4.yaml

```
noAdditionalIndent:
    mycommand:
        body: 0
        optionalArguments: 0
        mandatoryArguments: 1
```

After running the following commands,

```
cmh:~$ latexindent.pl mycommand.tex -l mycommand-noAdd3.yaml
cmh:~$ latexindent.pl mycommand.tex -l mycommand-noAdd4.yaml
```

we receive the respective output given in Listings 204 and 205.

LISTING 204: mycommand.tex using Listing 202

```
\mycommand
{
   mand arg text
   mand arg text}
[
opt arg text
opt arg text
]
```

LISTING 205: mycommand.tex using Listing 203

```
\mycommand
{
mand arg text
mand arg text}
[
   opt arg text
   opt arg text
]
```

■

example 57 Attentive readers will note that the body of `mycommand` in both Listings 204 and 205 has received no additional indent, even though `body` is explicitly set to `0` in both Listings 202 and 203. This is because, by default, `noAdditionalIndentGlobal` for `commands` is set to `1` by default; this can be easily fixed as in Listings 206 and 207. ■

LISTING 206: mycommand-noAdd5.yaml

```
noAdditionalIndent:
    mycommand:
        body: 0
        optionalArguments: 1
        mandatoryArguments: 0
noAdditionalIndentGlobal:
    commands: 0
```

LISTING 207: mycommand-noAdd6.yaml

```
noAdditionalIndent:
    mycommand:
        body: 0
        optionalArguments: 0
        mandatoryArguments: 1
noAdditionalIndentGlobal:
    commands: 0
```

After running the following commands,

```
cmh:~$ latexindent.pl mycommand.tex -l mycommand-noAdd5.yaml
cmh:~$ latexindent.pl mycommand.tex -l mycommand-noAdd6.yaml
```

we receive the respective output given in Listings 208 and 209.

LISTING 208: mycommand.tex using Listing 206

```
\mycommand
   {
      mand arg text
      mand arg text}
   [
   opt arg text
   opt arg text
   ]
```

LISTING 209: mycommand.tex using Listing 207

```
\mycommand
   {
   mand arg text
   mand arg text}
   [
      opt arg text
      opt arg text
   ]
```

Both `indentRules` and `indentRulesGlobal` can be adjusted as they were for *environment* code blocks, as in Listings 172 and 173 on page 54 and Listings 183, 186 and 187 on pages 56–57.

3.9.4 ifelsefi code blocks

example 58 Let's use the simple example shown in Listing 210; when `latexindent.pl` operates on this file, the output as in Listing 211; note that the body of each of the `\if` statements have been indented, and that the `\else` statement has been accounted for correctly.

LISTING 210: ifelsefi1.tex

```
\ifodd\radius
\ifnum\radius<14
\pgfmathparse{100-(\radius)*4};
\else
\pgfmathparse{200-(\radius)*3};
\fi\fi
```

LISTING 211: ifelsefi1.tex default output

```
\ifodd\radius
   \ifnum\radius<14
      \pgfmathparse{100-(\radius)*4};
   \else
      \pgfmathparse{200-(\radius)*3};
   \fi\fi
```

It is recommended to specify `noAdditionalIndent` and `indentRules` in the 'scalar' form only for these type of code blocks, although the 'field' form would work, assuming that `body` was specified. Examples are shown in Listings 212 and 213.

LISTING 212: ifnum-noAdd.yaml

```
noAdditionalIndent:
    ifnum: 1
```

LISTING 213: ifnum-indent-rules.yaml

```
indentRules:
    ifnum: "  "
```

After running the following commands,

```
cmh:~$ latexindent.pl ifelsefi1.tex -local ifnum-noAdd.yaml
cmh:~$ latexindent.pl ifelsefi1.tex -l ifnum-indent-rules.yaml
```

we receive the respective output given in Listings 214 and 215; note that in Listing 214, the `ifnum` code block has *not* received any additional indentation, while in Listing 215, the `ifnum` code block has received one tab and two spaces of indentation.

LISTING 214: `ifelsefi1.tex` using Listing 212

```
\ifodd\radius
   \ifnum\radius<14
   \pgfmathparse{100-(\radius)*4};
   \else
   \pgfmathparse{200-(\radius)*3};
   \fi\fi
```

LISTING 215: `ifelsefi1.tex` using Listing 213

```
\ifodd\radius
___\ifnum\radius<14
___␣␣\pgfmathparse{100-(\radius)*4};
___\else
___␣␣\pgfmathparse{200-(\radius)*3};
___\fi\fi
```

example 59 We may specify `noAdditionalIndentGlobal` and `indentRulesGlobal` as in Listings 216 and 217.

LISTING 216: `ifelsefi-noAdd-glob.yaml`

```
noAdditionalIndentGlobal:
    ifElseFi: 1
```

LISTING 217: `ifelsefi-indent-rules-global.yaml`

```
indentRulesGlobal:
    ifElseFi: " "
```

Upon running the following commands

```
cmh:~$ latexindent.pl ifelsefi1.tex -local ifelsefi-noAdd-glob.yaml
cmh:~$ latexindent.pl ifelsefi1.tex -l ifelsefi-indent-rules-global.yaml
```

we receive the outputs in Listings 218 and 219; notice that in Listing 218 neither of the `ifelsefi` code blocks have received indentation, while in Listing 219 both code blocks have received a single space of indentation.

LISTING 218: `ifelsefi1.tex` using Listing 216

```
\ifodd\radius
\ifnum\radius<14
\pgfmathparse{100-(\radius)*4};
\else
\pgfmathparse{200-(\radius)*3};
\fi\fi
```

LISTING 219: `ifelsefi1.tex` using Listing 217

```
\ifodd\radius
␣\ifnum\radius<14
␣␣\pgfmathparse{100-(\radius)*4};
␣\else
␣␣\pgfmathparse{200-(\radius)*3};
␣\fi\fi
```

example 60 We can further explore the treatment of `ifElseFi` code blocks in Listing 220, and the associated default output given in Listing 221; note, in particular, that the bodies of each of the 'or statements' have been indented.

U: 2018-04-27

LISTING 220: ifelsefi2.tex

```
\ifcase#1
zero%
\or
one%
\or
two%
\or
three%
\else
default
\fi
```

LISTING 221: ifelsefi2.tex default output

```
\ifcase#1
   zero%
\or
   one%
\or
   two%
\or
   three%
\else
   default
\fi
```

Fine tuning of `ifElseFi` is demonstrated in Section 7.5 on page 139.

3.9.5 specialBeginEnd code blocks

Let's use the example from Listing 123 on page 40 which has default output shown in Listing 124 on page 40.

example 61 It is recommended to specify `noAdditionalIndent` and `indentRules` in the 'scalar' form for these type of code blocks, although the 'field' form would work, assuming that `body` was specified. Examples are shown in Listings 222 and 223.

LISTING 222: displayMath-noAdd.yaml

```
noAdditionalIndent:
    displayMath: 1
```

LISTING 223: displayMath-indent-rules.yaml

```
indentRules:
    displayMath: "\t\t\t"
```

After running the following commands,

```
cmh:~$ latexindent.pl special1.tex -local displayMath-noAdd.yaml
cmh:~$ latexindent.pl special1.tex -l displayMath-indent-rules.yaml
```

we receive the respective output given in Listings 224 and 225; note that in Listing 224, the `displayMath` code block has *not* received any additional indentation, while in Listing 225, the `displayMath` code block has received three tabs worth of indentation.

LISTING 224: special1.tex using Listing 222

```
The function $f$ has formula
\[
f(x)=x^2.
\]
If you like splitting dollars,
$
   g(x)=f(2x)
$
```

LISTING 225: special1.tex using Listing 223

```
The function $f$ has formula
\[
		f(x)=x^2.
\]
If you like splitting dollars,
$
	g(x)=f(2x)
$
```

example 62 We may specify `noAdditionalIndentGlobal` and `indentRulesGlobal` as in Listings 226 and 227.

LISTING 226: special-noAdd-glob.yaml

```
noAdditionalIndentGlobal:
    specialBeginEnd: 1
```

LISTING 227: special-indent-rules-global.yaml

```
indentRulesGlobal:
    specialBeginEnd: " "
```

Upon running the following commands

```
cmh:~$ latexindent.pl special1.tex -local special-noAdd-glob.yaml
cmh:~$ latexindent.pl special1.tex -l special-indent-rules-global.yaml
```

we receive the outputs in Listings 228 and 229; notice that in Listing 228 neither of the `special` code blocks have received indentation, while in Listing 229 both code blocks have received a single space of indentation.

LISTING 228: special1.tex using Listing 226

```
The function $f$ has formula
\[
f(x)=x^2.
\]
If you like splitting dollars,
$
g(x)=f(2x)
$
```

LISTING 229: special1.tex using Listing 227

```
␣The␣function␣$f$␣has␣formula
\[
␣␣f(x)=x^2.
\]
If␣you␣like␣splitting␣dollars,
$
␣␣g(x)=f(2x)
$
```

■

3.9.6 afterHeading code blocks

Let's use the example Listing 230 for demonstration throughout this Section. As discussed on page 47, by default `latexindent.pl` will not add indentation after headings.

LISTING 230: headings2.tex

```
\paragraph{paragraph
title}
paragraph text
paragraph text
```

example 63 On using the YAML file in Listing 232 by running the command

```
cmh:~$ latexindent.pl headings2.tex -l headings3.yaml
```

we obtain the output in Listing 231. Note that the argument of `paragraph` has received (default) indentation, and that the body after the heading statement has received (default) indentation.

LISTING 231: headings2.tex using Listing 232

```
\paragraph{paragraph
      title}
   paragraph text
   paragraph text
```

LISTING 232: headings3.yaml

```
indentAfterHeadings:
    paragraph:
       lookForThis: 1
       level: 1
```

If we specify `noAdditionalIndent` as in Listing 234 and run the command

```
cmh:~$ latexindent.pl headings2.tex -l headings4.yaml
```

then we receive the output in Listing 233. Note that the arguments *and* the body after the heading of `paragraph` has received no additional indentation, because we have specified `noAdditionalIndent` in scalar form.

■

LISTING 233: headings2.tex using Listing 234

```
\paragraph{paragraph
title}
paragraph text
paragraph text
```

LISTING 234: headings4.yaml

```
indentAfterHeadings:
    paragraph:
       lookForThis: 1
       level: 1
noAdditionalIndent:
    paragraph: 1
```

example 64 Similarly, if we specify `indentRules` as in Listing 236 and run analogous commands to those above, we receive the output in Listing 235; note that the *body*, *mandatory argument* and content *after the heading* of `paragraph` have *all* received three tabs worth of indentation.

LISTING 235: headings2.tex using Listing 236

```
\paragraph{paragraph
                    title}
            paragraph text
            paragraph text
```

LISTING 236: headings5.yaml

```
indentAfterHeadings:
    paragraph:
       lookForThis: 1
       level: 1
indentRules:
    paragraph: "\t\t\t"
```

example 65 We may, instead, specify `noAdditionalIndent` in 'field' form, as in Listing 238 which gives the output in Listing 237.

LISTING 237: headings2.tex using Listing 238

```
\paragraph{paragraph
   title}
paragraph text
paragraph text
```

LISTING 238: headings6.yaml

```
indentAfterHeadings:
    paragraph:
       lookForThis: 1
       level: 1
noAdditionalIndent:
    paragraph:
        body: 0
        mandatoryArguments: 0
        afterHeading: 1
```

example 66 Analogously, we may specify `indentRules` as in Listing 240 which gives the output in Listing 239; note that mandatory argument text has only received a single space of indentation, while the body after the heading has received three tabs worth of indentation.

LISTING 239: headings2.tex using Listing 240

```
\paragraph{paragraph
          title}
            paragraph text
            paragraph text
```

LISTING 240: headings7.yaml

```
indentAfterHeadings:
    paragraph:
       lookForThis: 1
       level: 1
indentRules:
    paragraph:
        mandatoryArguments: " "
        afterHeading: "\t\t\t"
```

example 67 Finally, let's consider `noAdditionalIndentGlobal` and `indentRulesGlobal` shown in Listings 242 and 244 respectively, with respective output in Listings 241 and 243. Note that in Listing 242 the *mandatory argument* of `paragraph` has received a (default) tab's worth of indentation, while the body after the heading has received *no additional indentation*. Similarly, in Listing 243, the *argument* has received both a (default) tab plus two spaces of indentation (from the global rule specified in Listing 244), and the remaining body after `paragraph` has received just two spaces

of indentation.

LISTING 241: headings2.tex using Listing 242

```
\paragraph{paragraph
   title}
paragraph text
paragraph text
```

LISTING 242: headings8.yaml

```
indentAfterHeadings:
    paragraph:
       lookForThis: 1
       level: 1
noAdditionalIndentGlobal:
    afterHeading: 1
```

LISTING 243: headings2.tex using Listing 244

```
\paragraph{paragraph
␣␣␣␣title}
␣␣paragraph␣text
␣␣paragraph␣text
```

LISTING 244: headings9.yaml

```
indentAfterHeadings:
    paragraph:
       lookForThis: 1
       level: 1
indentRulesGlobal:
    afterHeading: "  "
```

■

3.9.7 The remaining code blocks

Referencing the different types of code blocks in Table 2 on page 49, we have a few code blocks yet to cover; these are very similar to the `commands` code block type covered comprehensively in Section 3.9.3 on page 58, but a small discussion defining these remaining code blocks is necessary.

3.9.7.1 keyEqualsValuesBracesBrackets

`latexindent.pl` defines this type of code block by the following criteria:

- it has a name made up of the characters detailed in Table 2 on page 49;
- then an = symbol;
- then at least one set of curly braces or square brackets (comments and line breaks allowed throughout).

See the `keyEqualsValuesBracesBrackets:` `name` field of the fine tuning section in Listing 542 on page 134

N: 2019-07-13

example 68 An example is shown in Listing 245, with the default output given in Listing 246.

LISTING 245: pgfkeys1.tex

```
\pgfkeys{/tikz/.cd,
start coordinate/.initial={0,
\vertfactor},
}
```

LISTING 246: pgfkeys1.tex default output

```
\pgfkeys{/tikz/.cd,
␣␣␣start coordinate/.initial={0,
␣␣␣␣␣␣␣␣␣\vertfactor},
}
```

In Listing 246, note that the maximum indentation is three tabs, and these come from:

- the `\pgfkeys` command's mandatory argument;
- the `start coordinate/.initial` key's mandatory argument;
- the `start coordinate/.initial` key's body, which is defined as any lines following the name of the key that include its arguments. This is the part controlled by the *body* field for `noAdditionalIndent` and friends from page 50.

■

3.9.7.2 namedGroupingBracesBrackets

This type of code block is mostly motivated by tikz-based code; we define this code block as follows:

- the name may contain the characters detailed in Table 2 on page 49;

- then at least one set of curly braces or square brackets (comments and line breaks allowed throughout).

N: 2019-07-13

See the `NamedGroupingBracesBrackets: name` field of the fine tuning section in Listing 542 on page 134

example 69 A simple example is given in Listing 247, with default output in Listing 248.

LISTING 247: `child1.tex`

```
\coordinate
child[grow=down]{
edge from parent [antiparticle]
node [above=3pt] {$C$}
}
```

LISTING 248: `child1.tex` default output

```
\coordinate
child[grow=down]{
    edge from parent [antiparticle]
        node [above=3pt] {$C$}
    }
```

In particular, `latexindent.pl` considers `child` and `parent` to be `namedGroupingBracesBrackets`[a]. Referencing Listing 248, note that the maximum indentation is three tabs, and these come from:

- the `child`'s mandatory argument;
- the `child`'s body, which is defined as any lines following the name of the `namedGroupingBracesBrackets` that include its arguments. This is the part controlled by the *body* field for `noAdditionalIndent` and friends from page 50;
- the `parent`'s body, which is defined as any lines following the name that include its arguments.

[a] You may like to verify this by using the `-t` option and checking `indent.log`!

■

example 70 Consider the file given in Listing 249, together with its default output using the command

```
cmh:~$ latexindent.pl named1.tex
```

is given in Listing 250.

LISTING 249: `named1.tex`

```
\mycommand{
    \rule{G -> +H[-G]CL}
    \rule{H -> -G[+H]CL}
    \rule{g -> +h[-g]cL}
    \rule{h -> -g[+h]cL}
}
```

LISTING 250: `named1.tex` default

```
\mycommand{
   \rule{G -> +H[-G]CL}
   \rule{H -> -G[+H]CL}
   \rule{g -> +h[-g]cL}
   \rule{h -> -g[+h]cL}
}
```

■

example 71 Consider the file given in Listing 251, together with its default output using the command

```
cmh:~$ latexindent.pl finetuning2.tex
```

is given in Listing 252.

LISTING 251: `finetuning2.tex`

```
@misc{ wikilatex,
author = "{Wikipedia contributors}",
title = "LaTeX --- {Wikipedia}{,}",
note = "[Online; accessed 3-March-2020]"
}
```

LISTING 252: `finetuning2.tex` default

```
@misc{ wikilatex,
      author = "{Wikipedia contributors}",
      title = "LaTeX --- {Wikipedia}{,}",
      note = "[Online; accessed 3-March-2020]"
   }
```

■

example 72 Starting with the file in Listing 253 and running the command

■

```
cmh:~$ latexindent.pl bib1.tex -o=+-mod1
```

gives the output in Listing 254.

LISTING 253: bib1.bib

```
@online{paulo,
title="arararule,indent.yaml",
author="PauloCereda",
date={2013-05-23},
urldate={2021-03-19},
keywords={contributor},}
```

LISTING 254: bib1-mod1.bib

```
@online{paulo,
   title="arararule,indent.yaml",
   author="PauloCereda",
   date={2013-05-23},
   urldate={2021-03-19},
   keywords={contributor},}
```

Let's assume that we would like to format the output so as to align the = symbols. Using the settings in Listing 256 and running the command

```
cmh:~$ latexindent.pl bib1.bib -l bibsettings1.yaml -o=+-mod2
```

gives the output in Listing 255.

LISTING 255: bib1.bib using Listing 256

```
@online{paulo,
   title    = "arararule,indent.yaml",
   author   = "PauloCereda",
   date     = {2013-05-23},
   urldate  = {2021-03-19},
   keywords = {contributor},}
```

LISTING 256: bibsettings1.yaml

```
lookForAlignDelims:
   @online:
      delimiterRegEx: '(=)'
      lookForChildCodeBlocks: 0
noAdditionalIndentGlobal:
  namedGroupingBracesBrackets: 1
```

we have populated the `lookForAlignDelims` field with the `online` command, and have used the `delimiterRegEx`, discussed in Section 3.5.3 on page 35. ■

example 73 We can build upon Listing 256 for slightly more complicated bibliography files.

Starting with the file in Listing 257 and running the command

```
cmh:~$ latexindent.pl bib2.bib -l bibsettings1.yaml -o=+-mod1
```

gives the output in Listing 258.

LISTING 257: bib2.bib

```
@online{cmh:videodemo,
title="Videodemonstrationofpl.latexindentonyoutube",
url="https://www.youtube.com/watch?v=wo38aaH2F4E&spfreload=10",
urldate={2017-02-21},
}
```

LISTING 258: bib2-mod1.bib

```
@online{cmh:videodemo,
   title   = "Videodemonstrationofpl.latexindentonyoutube",
   url     = "https://www.youtube.com/watch?v                = wo38aaH2F4E&spfreload = 10",
   urldate = {2017-02-21},
}
```

The output in Listing 258 is not ideal, as the = symbol within the url field has been incorrectly used as an alignment delimiter. ■

We address this by tweaking the `delimiterRegEx` field in Listing 259.

LISTING 259: `bibsettings2.yaml`

```
lookForAlignDelims:
   @online:
      delimiterRegEx: '(?<!v)(?<!spfreload)(=)'
```

Upon running the command

```
cmh:~$ latexindent.pl bib2.bib -l bibsettings1.yaml,bibsettings2.yaml -o=+-mod2
```

we receive the *desired* output in Listing 260.

LISTING 260: `bib2-mod2.bib`

```
@online{cmh:videodemo,
   title   = "Videodemonstrationofpl.latexindentonyoutube",
   url     = "https://www.youtube.com/watch?v=wo38aaH2F4E&spfreload=10",
   urldate = {2017-02-21},
}
```

With reference to Listing 259 we note that the `delimiterRegEx` has been adjusted so that `=` symbols are used as the delimiter, but only when they are *not preceded* by either `v` or `spfreload`. ■

3.9.7.3 UnNamedGroupingBracesBrackets

occur in a variety of situations; specifically, we define this type of code block as satisfying the following criteria:

- it isn't a `command`, `keyEqualsValuesBracesBrackets` or `namedGroupingBracesBrackets`
- and it has at least one set of curly braces or square brackets (comments and line breaks allowed throughout).

example 74 An example is shown in Listing 261 with default output give in Listing 262.

LISTING 261: `psforeach1.tex`

```
\psforeach{\row}{%
{
{3,2.8,2.7,3,3.1}},%
{2.8,1,1.2,2,3},%
}
```

LISTING 262: `psforeach1.tex` default output

```
\psforeach{\row}{%
    {
        {3,2.8,2.7,3,3.1}},%
    {2.8,1,1.2,2,3},%
}
```

Referencing Listing 262, there are *two* sets of unnamed braces. Note also that the maximum value of indentation is two tabs, and these come from:

- the `\psforeach` command's mandatory argument;
- the *first* un-named braces mandatory argument. ■

Users wishing to customise the mandatory and/or optional arguments on a *per-name* basis for the `UnNamedGroupingBracesBrackets` should use `always-un-named`.

example 75 Starting with the file in Listing 263 we receive the default output in Listing 264. ■

LISTING 263: unnamed1.tex

```
 {%
{1,2,3,4}
}[%
{1,2,3,4}
]
```

LISTING 264: unnamed1.tex default output

```
{%
    {1,2,3,4}
}[%
    {1,2,3,4}
]
```

We can customise the output using, for example, Listing 265 and running

```
cmh:~$ latexindent.pl unnamed1 -l unnamed1.yaml unnamed1
```

gives the output in Listing 264

LISTING 265: unnamed1.yaml

```
indentRules:
  always-un-named:
    body: ""
    mandatoryArguments: " "
    optionalArguments: "   "
```

LISTING 266: unnamed1.tex mod1 output

```
{%
 {1,2,3,4}
}[%
   {1,2,3,4}
]
```

■

3.9.8 Summary

Having considered all of the different types of code blocks, the functions of the fields given in Listings 267 and 268 should now make sense.

LISTING 267: noAdditionalIndentGlobal

```
noAdditionalIndentGlobal:
  environments: 0                    # 0/1
  commands: 1                        # 0/1
  optionalArguments: 0               # 0/1
  mandatoryArguments: 0              # 0/1
  ifElseFi: 0                        # 0/1
  items: 0                           # 0/1
  keyEqualsValuesBracesBrackets: 0   # 0/1
  namedGroupingBracesBrackets: 0     # 0/1
  UnNamedGroupingBracesBrackets: 1   # 0/1
  specialBeginEnd: 0                 # 0/1
  afterHeading: 0                    # 0/1
```

LISTING 268: indentRulesGlobal

```
indentRulesGlobal:
  environments: 0                    # 0/h-space
  commands: 0                        # 0/h-space
  optionalArguments: 0               # 0/h-space
  mandatoryArguments: 0              # 0/h-space
  ifElseFi: 0                        # 0/h-space
  items: 0                           # 0/h-space
  keyEqualsValuesBracesBrackets: 0   # 0/h-space
  namedGroupingBracesBrackets: 0     # 0/h-space
  UnNamedGroupingBracesBrackets: 0   # 0/h-space
  specialBeginEnd: 0                 # 0/h-space
  afterHeading: 0                    # 0/h-space
```

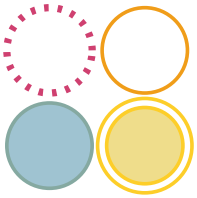
SECTION 4

The -m (modifylinebreaks) switch

All features described in this section will only be relevant if the `-m` switch is used.

`modifylinebreaks`: ⟨*fields*⟩

As of Version 3.0, `latexindent.pl` has the `-m` switch, which permits `latexindent.pl` to modify line breaks, according to the specifications in the `modifyLineBreaks` field. *The settings in this field will only be considered if the `-m` switch has been used.* A snippet of the default settings of this field is shown in Listing 269.



LISTING 269: `modifyLineBreaks` `-m`

```
289 modifyLineBreaks:
290   preserveBlankLines: 1                # 0/1
291   condenseMultipleBlankLinesInto: 1    # 0/1
```

Having read the previous paragraph, it should sound reasonable that, if you call `latexindent.pl` using the `-m` switch, then you give it permission to modify line breaks in your file, but let's be clear:

Warning!

If you call `latexindent.pl` with the `-m` switch, then you are giving it permission to modify line breaks. By default, the only thing that will happen is that multiple blank lines will be condensed into one blank line; many other settings are possible, discussed next.

`preserveBlankLines`: **0**|**1**

This field is directly related to *poly-switches*, discussed in Section 4.3. By default, it is set to `1`, which means that blank lines will be *protected* from removal; however, regardless of this setting, multiple blank lines can be condensed if `condenseMultipleBlankLinesInto` is greater than `0`, discussed next.

`condenseMultipleBlankLinesInto`: ⟨*positive integer*⟩

Assuming that this switch takes an integer value greater than `0`, `latexindent.pl` will condense multiple blank lines into the number of blank lines illustrated by this switch.

example 76 As an example, Listing 270 shows a sample file with blank lines; upon running

```
cmh:~$ latexindent.pl myfile.tex -m -o=+-mod1
```

the output is shown in Listing 271; note that the multiple blank lines have been condensed into ■

one blank line, and note also that we have used the `-m` switch!

LISTING 270: `mlb1.tex`

```
before blank line


after blank line


after blank line
```

LISTING 271: `mlb1-mod1.tex`

```
before blank line

after blank line

after blank line
```

■

4.1 Text Wrapping

N: 2022-03-13

The text wrapping routine has been over-hauled as of V3.16; I hope that the interface is simpler, and most importantly, the results are better.

The complete settings for this feature are given in Listing 272.

LISTING 272: `textWrapOptions` `-m`

```
textWrapOptions:
  columns: 0
  multipleSpacesToSingle: 1
  removeBlockLineBreaks: 1
  when: before                        # before/after
  comments:
    wrap: 0                           # 0/1
    inheritLeadingSpace: 0            # 0/1
  blocksFollow:
    headings: 1                       # 0/1
    commentOnPreviousLine: 1          # 0/1
    par: 1                            # 0/1
    blankLine: 1                      # 0/1
    verbatim: 1                       # 0/1
    other: \\\]|\\item(?:\h|\[)       # regex
  blocksBeginWith:
    A-Z: 1                            # 0/1
    a-z: 1                            # 0/1
    0-9: 0                            # 0/1
    other: 0                          # regex
  blocksEndBefore:
    commentOnOwnLine: 1               # 0/1
    verbatim: 1                       # 0/1
    other: \\begin\{|\\\[|\\end\{     # regex
  huge: overflow                      # forbid mid-word line breaks
  separator: ""
```

4.1.1 Text wrap: overview

An overview of how the text wrapping feature works:

1. the default value of `columns` is 0, which means that text wrapping will *not* happen by default;
2. it happens *after* verbatim blocks have been found;
3. it happens *after* the oneSentencePerLine routine (see Section 4.2);
4. N: 2023-01-01 it can happen *before* or *after* all of the other code blocks are found and does *not* operate on a per-code-block basis; when using `before` this means that, including indentation, you may receive a column width wider than that which you specify in `columns`, and in which case you probably wish to explore `after` in Section 4.1.7;
5. code blocks to be text wrapped will:

(a) *follow* the fields specified in `blocksFollow`

(b) *begin* with the fields specified in `blocksBeginWith`

(c) *end* before the fields specified in `blocksEndBefore`

6. setting `columns` to a value > 0 will text wrap blocks by first removing line breaks, and then wrapping according to the specified value of `columns`;
7. setting `columns` to -1 will *only* remove line breaks within the text wrap block;
8. by default, the text wrapping routine will remove line breaks within text blocks because `removeBlockLineBreak` is set to 1; switch it to 0 if you wish to change this;
9. about trailing comments within text wrap blocks:
   (a) trailing comments that do *not* have leading space instruct the text wrap routine to connect the lines *without* space (see Listing 310);
   (b) multiple trailing comments will be connected at the end of the text wrap block (see Listing 314);
   (c) the number of spaces between the end of the text wrap block and the (possibly combined) trailing comments is determined by the spaces (if any) at the end of the text wrap block (see Listing 316);

N: 2023-01-01

10. trailing comments can receive text wrapping ; examples are shown in Section 4.1.8 and Section 4.2.9.

We demonstrate this feature using a series of examples.

4.1.2 Text wrap: simple examples

example 77 Let's use the sample text given in Listing 273.

LISTING 273: `textwrap1.tex`

```
Here  is a line of text that will be wrapped by latexindent.pl.

Here is  a line of text that will be wrapped by latexindent.pl.
```

We will change the value of `columns` in Listing 275 and then run the command

```
cmh:~$ latexindent.pl -m -l textwrap1.yaml textwrap1.tex
```

then we receive the output given in Listing 274.

LISTING 274: `textwrap1-mod1.tex`

```
Here is a line of
text that will be
wrapped by
latexindent.pl.

Here is a line of
text that will be
wrapped by
latexindent.pl.
```

LISTING 275: `textwrap1.yaml` `-m`

```
modifyLineBreaks:
    textWrapOptions:
        columns: 20
```

■

example 78 If we set `columns` to -1 then `latexindent.pl` remove line breaks within the text wrap block, and will *not* perform text wrapping. We can use this to undo text wrapping.

Starting from the file in Listing 274 and using the settings in Listing 276

■

LISTING 276: textwrap1A.yaml -m

```
modifyLineBreaks:
    textWrapOptions:
        columns: -1
```

and running

```
cmh:~$ latexindent.pl -m -l textwrap1A.yaml textwrap1-mod1.tex
```

gives the output in Listing 277.

LISTING 277: textwrap1-mod1A.tex

```
Here is a line of text that will be wrapped by latexindent.pl.

Here is a line of text that will be wrapped by latexindent.pl.
```

■

example 79 By default, the text wrapping routine will convert multiple spaces into single spaces. You can change this behaviour by flicking the switch `multipleSpacesToSingle` which we have done in Listing 279

Using the settings in Listing 279 and running

```
cmh:~$ latexindent.pl -m -l textwrap1B.yaml textwrap1-mod1.tex
```

gives the output in Listing 278.

LISTING 278: textwrap1-mod1B.tex

```
Here␣␣is␣a␣line␣of
text␣that␣will␣be
wrapped␣by
latexindent.pl.

Here␣is␣␣␣a␣line␣of
text␣that␣will␣be
wrapped␣by
latexindent.pl.
```

LISTING 279: textwrap1B.yaml -m

```
modifyLineBreaks:
    textWrapOptions:
        columns: 20
        multipleSpacesToSingle: 0
```

We note that in Listing 278 the multiple spaces have *not* been condensed into single spaces. ■

4.1.3 Text wrap: `blocksFollow` examples

We examine the `blocksFollow` field of Listing 272.

example 80 Let's use the sample text given in Listing 280.

LISTING 280: tw-headings1.tex

```
\section{my heading}\label{mylabel1}
text to
    be
 wrapped from the first section
\subsection{subheading}
text to
    be
 wrapped from the first section
```

We note that Listing 280 contains the heading commands `section` and `subsection`. Upon running the command ■

```
cmh:~$ latexindent.pl -m -l textwrap1.yaml tw-headings1.tex
```

then we receive the output given in Listing 281.

LISTING 281: tw-headings1-mod1.tex

```
\section{my heading}\label{mylabel1}
text to be wrapped
from the first
section
\subsection{subheading}
text to be wrapped
from the first
section
```

We reference Listing 272 on page 72 and also Listing 146 on page 46:

- in Listing 272 the `headings` field is set to `1`, which instructs `latexindent.pl` to read the fields from Listing 146 on page 46, *regardless of the value of lookForThis or level*;
- the default is to assume that the heading command can, optionally, be followed by a `label` command.

If you find scenarios in which the default value of `headings` does not work, then you can explore the `other` field.

We can turn off `headings` as in Listing 283 and then run

```
cmh:~$ latexindent.pl -m -l textwrap1.yaml,bf-no-headings.yaml tw-headings1.tex
```

gives the output in Listing 282, in which text wrapping has been instructed *not to happen* following headings.

LISTING 282: tw-headings1-mod2.tex

```
\section{my heading}\label{mylabel1}
text to
be
wrapped from the first section
\subsection{subheading}
text to
be
wrapped from the first section
```

LISTING 283: bf-no-headings.yaml -m

```
modifyLineBreaks:
    textWrapOptions:
        blocksFollow:
            headings: 0
```

■

example 81 Let's use the sample text given in Listing 284.

LISTING 284: tw-comments1.tex

```
% trailing comment
text to
    be
 wrapped following first comment
% another comment
text to
    be
 wrapped following second comment
```

We note that Listing 284 contains trailing comments. Upon running the command

```
cmh:~$ latexindent.pl -m -l textwrap1.yaml tw-comments1.tex
```

■

then we receive the output given in Listing 285.

LISTING 285: tw-comments1-mod1.tex

```
% trailing comment
text to be wrapped
following first
comment
% another comment
text to be wrapped
following second
comment
```

With reference to Listing 272 on page 72 the `commentOnPreviousLine` field is set to `1`, which instructs `latexindent.pl` to find text wrap blocks after a comment on its own line.

We can turn off `comments` as in Listing 287 and then run

```
cmh:~$ latexindent.pl -m -l textwrap1.yaml,bf-no-comments.yaml tw-comments1.tex
```

gives the output in Listing 286, in which text wrapping has been instructed *not to happen* following comments on their own line.

LISTING 286: tw-comments1-mod2.tex

```
% trailing comment
text to
be
wrapped following first comment
% another comment
text to
be
wrapped following second comment
```

LISTING 287: bf-no-comments.yaml -m

```
modifyLineBreaks:
    textWrapOptions:
        blocksFollow:
           commentOnPreviousLine: 0
```

■

Referencing Listing 272 on page 72 the `blocksFollow` fields `par`, `blankline`, `verbatim` and `filecontents` fields operate in analogous ways to those demonstrated in the above.

The `other` field of the `blocksFollow` can either be `0` (turned off) or set as a regular expression. The default value is set to `\\\]|\\item(?:\h|\[)` which can be translated to *backslash followed by a square bracket* or *backslash item followed by horizontal space or a square bracket*, or in other words, *end of display math* or an item command.

example 82 Let's use the sample text given in Listing 288.

LISTING 288: tw-disp-math1.tex

```
text to
    be
 wrapped before display math
 \[ y = x\]
text to
    be
 wrapped after display math
```

We note that Listing 288 contains display math. Upon running the command

```
cmh:~$ latexindent.pl -m -l textwrap1.yaml tw-disp-math1.tex
```

then we receive the output given in Listing 289.

■

LISTING 289: tw-disp-math1-mod1.tex

```
text to be wrapped
before display math
\[ y = x\]
text to be wrapped
after display math
```

With reference to Listing 272 on page 72 the `other` field is set to `\\\]`, which instructs `latexindent.pl` to find text wrap blocks after the end of display math.

We can turn off this switch as in Listing 291 and then run

```
cmh:~$ latexindent.pl -m -l textwrap1.yaml,bf-no-disp-math.yaml tw-disp-math1.tex
```

gives the output in Listing 290, in which text wrapping has been instructed *not to happen* following display math.

LISTING 290: tw-disp-math1-mod2.tex

```
text to be wrapped
before display math
\[ y = x\]
text to
be
wrapped after display math
```

LISTING 291: bf-no-disp-math.yaml -m

```
modifyLineBreaks:
    textWrapOptions:
        blocksFollow:
            other: 0
```

Naturally, you should feel encouraged to customise this as you see fit. ■

The `blocksFollow` field *deliberately* does not default to allowing text wrapping to occur after `begin environment` statements. You are encouraged to customize the `other` field to accommodate the environments that you would like to text wrap individually, as in the next example.

example 83 Let's use the sample text given in Listing 292.

LISTING 292: tw-bf-myenv1.tex

```
text to
    be
 wrapped before myenv environment
 \begin{myenv}
text to
    be
 wrapped within myenv environment
 \end{myenv}
text to
    be
 wrapped after myenv environment
```

We note that Listing 292 contains `myenv` environment. Upon running the command

```
cmh:~$ latexindent.pl -m -l textwrap1.yaml tw-bf-myenv1.tex
```

then we receive the output given in Listing 293. ■

LISTING 293: tw-bf-myenv1-mod1.tex

```
text to be wrapped
before myenv
environment
\begin{myenv}
   text to
   be
   wrapped within myenv environment
\end{myenv}
text to
be
wrapped after myenv environment
```

We note that we have *not* received much text wrapping. We can turn do better by employing Listing 295 and then run

```
cmh:~$ latexindent.pl -m -l textwrap1.yaml,tw-bf-myenv.yaml tw-bf-myenv1.tex
```

which gives the output in Listing 294, in which text wrapping has been implemented across the file.

LISTING 294: tw-bf-myenv1-mod2.tex

```
text to be wrapped
before myenv
environment
\begin{myenv}
   text to be wrapped
   within myenv
   environment
\end{myenv}
text to be wrapped
after myenv
environment
```

LISTING 295: tw-bf-myenv.yaml -m

```
modifyLineBreaks:
    textWrapOptions:
        blocksFollow:
           other: |-
             (?x)
                \\\]
                |
                \\item(?:\h|\[)
                |
                \\begin\{myenv\} # <--- new bit
                |                # <--- new bit
                \\end\{myenv\}   # <--- new bit
```

4.1.4 Text wrap: blocksBeginWith examples

We examine the blocksBeginWith field of Listing 272 with a series of examples.

example 84 By default, text wrap blocks can begin with the characters a-z and A-Z.

If we start with the file given in Listing 296

LISTING 296: tw-0-9.tex

```
123 text to
    be
 wrapped before display math
 \[ y = x\]
456 text to
    be
 wrapped after display math
```

and run the command

```
cmh:~$ latexindent.pl -m -l textwrap1.yaml tw-0-9.tex
```

then we receive the output given in Listing 297 in which text wrapping has *not* occurred.

LISTING 297: tw-0-9-mod1.tex

```
123 text to
be
wrapped before display math
\[ y = x\]
456 text to
be
wrapped after display math
```

We can allow paragraphs to begin with 0-9 characters by using the settings in Listing 299 and running

```
cmh:~$ latexindent.pl -m -l textwrap1.yaml,bb-0-9-yaml tw-0-9.tex
```

gives the output in Listing 298, in which text wrapping *has* happened.

LISTING 298: tw-0-9-mod2.tex

```
123 text to be
wrapped before
display math
\[ y = x\]
456 text to be
wrapped after
display math
```

LISTING 299: bb-0-9.yaml.yaml -m

```
modifyLineBreaks:
    textWrapOptions:
        blocksBeginWith:
            0-9: 1
```

■

example 85 Let's now use the file given in Listing 300

LISTING 300: tw-bb-announce1.tex

```
% trailing comment
\announce{announce text}
    and text
    to be
 wrapped before
  goes here
```

and run the command

```
cmh:~$ latexindent.pl -m -l textwrap1.yaml tw-bb-announce1.tex
```

then we receive the output given in Listing 301 in which text wrapping has *not* occurred.

LISTING 301: tw-bb-announce1-mod1.tex

```
% trailing comment
\announce{announce text}
and text
to be
wrapped before
goes here
```

We can allow \announce to be at the beginning of paragraphs by using the settings in Listing 303 and running

```
cmh:~$ latexindent.pl -m -l textwrap1.yaml,tw-bb-announce.yaml tw-bb-announce1.tex
```

gives the output in Listing 302, in which text wrapping *has* happened.

■

LISTING 302: tw-bb-announce1-mod2.tex

```
% trailing comment
\announce{announce
   text} and text to
be wrapped before
goes here
```

LISTING 303: tw-bb-announce.yaml -m

```
modifyLineBreaks:
    textWrapOptions:
        blocksBeginWith:
            other: '\\announce'
```

4.1.5 Text wrap: blocksEndBefore examples

We examine the blocksEndBefore field of Listing 272 with a series of examples.

example 86 Let's use the sample text given in Listing 304.

LISTING 304: tw-be-equation.tex

```
before
equation
text
\begin{align}
  1 & 2 \\
  3 & 4
\end{align}
after
equation
text
```

We note that Listing 304 contains an environment. Upon running the command

```
cmh:~$ latexindent.pl -m -l textwrap1A.yaml tw-be-equation.tex
```

then we receive the output given in Listing 305.

LISTING 305: tw-be-equation-mod1.tex

```
before equation text
\begin{align}
   1 & 2 \\
   3 & 4
\end{align}
after
equation
text
```

With reference to Listing 272 on page 72 the other field is set to \\begin\{|\\\[|\\end\{, which instructs latexindent.pl to *stop* text wrap blocks before begin statements, display math, and end statements.

We can turn off this switch as in Listing 306 and then run

```
cmh:~$ latexindent.pl -m -l textwrap1A.yaml,tw-be-equation.yaml tw-be-equation.tex
```

gives the output in Listing 307, in which text wrapping has been instructed *not* to stop at these statements.

LISTING 306: tw-be-equation.yaml -m

```
modifyLineBreaks:
    textWrapOptions:
        blocksEndBefore:
           other: 0
```

LISTING 307: tw-be-equation-mod2.tex

```
before equation text \begin{align} 1 & 2 \\ 3 & 4 \end{align} after equation text
```

Naturally, you should feel encouraged to customise this as you see fit. ■

4.1.6 Text wrap: trailing comments and spaces

We explore the behaviour of the text wrap routine in relation to trailing comments using the following examples.

example 87 The file in Listing 308 contains a trailing comment which *does* have a space infront of it.

Running the command

```
cmh:~$ latexindent.pl -m tw-tc1.tex -l textwrap1A.yaml -o=+-mod1
```

gives the output given in Listing 309.

LISTING 308: tw-tc1.tex

```
foo␣%
bar
```

LISTING 309: tw-tc1-mod1.tex

```
foo bar%
```

■

example 88 The file in Listing 310 contains a trailing comment which does *not* have a space infront of it.

Running the command

```
cmh:~$ latexindent.pl -m tw-tc2.tex -l textwrap1A.yaml -o=+-mod1
```

gives the output in Listing 311.

LISTING 310: tw-tc2.tex

```
foo%
bar
```

LISTING 311: tw-tc2-mod1.tex

```
foobar%
```

We note that, because there is *not* a space before the trailing comment, that the lines have been joined *without* a space. ■

example 89 The file in Listing 312 contains multiple trailing comments.

Running the command

```
cmh:~$ latexindent.pl -m tw-tc3.tex -l textwrap1A.yaml -o=+-mod1
```

gives the output in Listing 313.

LISTING 312: tw-tc3.tex

```
foo %1
bar%2
three
```

LISTING 313: tw-tc3-mod1.tex

```
foo barthree%1%2
```

■

example 90 The file in Listing 314 contains multiple trailing comments.

Running the command

```
cmh:~$ latexindent.pl -m tw-tc4.tex -l textwrap1A.yaml -o=+-mod1
```

gives the output in Listing 315.

LISTING 314: tw-tc4.tex

```
foo %1
bar%2
three%3
```

LISTING 315: tw-tc4-mod1.tex

```
foo barthree%1%2%3
```

■

example 91 The file in Listing 316 contains multiple trailing comments.

Running the command

```
cmh:~$ latexindent.pl -m tw-tc5.tex -l textwrap1A.yaml -o=+-mod1
```

gives the output in Listing 317.

LISTING 316: tw-tc5.tex

```
foo%1
bar%2
three␣%3
```

LISTING 317: tw-tc5-mod1.tex

```
foobarthree␣%1%2%3
```

The space at the end of the text block has been preserved. ■

example 92 The file in Listing 318 contains multiple trailing comments.

Running the command

```
cmh:~$ latexindent.pl -m tw-tc6.tex -l textwrap1A.yaml -o=+-mod1
```

gives the output in Listing 319.

LISTING 318: tw-tc6.tex

```
foo%1
bar
```

LISTING 319: tw-tc6-mod1.tex

```
foobar␣%1
```

The space at the end of the text block has been preserved. ■

4.1.7 Text wrap: when before/after

N: 2023-01-01

The text wrapping routine operates, by default, `before` the code blocks have been found, but this can be changed to `after`:

- `before` means it is likely that the columns of wrapped text may *exceed* the value specified in `columns`;
- `after` means it columns of wrapped text should *not* exceed the value specified in `columns`.

We demonstrate this in the following examples. See also Section 4.2.8.

example 93 Let's begin with the file in Listing 320. ■

LISTING 320: textwrap8.tex

```
This paragraph
has line breaks throughout its paragraph;
we would like to combine
the textwrapping
and paragraph removal routine.
\begin{myenv}
This paragraph
has line breaks throughout its paragraph;
we would like to combine
the textwrapping
and paragraph removal routine.
\end{myenv}
```

Using the settings given in Listing 322 and running the command

```
cmh:~$ latexindent.pl textwrap8.tex -o=+-mod1.tex -l=tw-before1.yaml -m
```

gives the output given in Listing 321.

LISTING 321: textwrap8-mod1.tex

```
This paragraph has line breaks
throughout its paragraph; we would
like to combine the textwrapping
and paragraph removal routine.
\begin{myenv}
  This paragraph has line breaks
  throughout its paragraph; we would
  like to combine the textwrapping
  and paragraph removal routine.
\end{myenv}
----|----|----|----|----|----|----|----|
   5    10    15    20    25    30    35    40
```

LISTING 322: tw-before1.yaml -m

```
defaultIndent: '  '

modifyLineBreaks:
    textWrapOptions:
        columns: 35
        when: before  # <!-------
        blocksFollow:
          other: \\begin\{myenv\}
```

We note that, in Listing 321, that the wrapped text has *exceeded* the specified value of `columns` (35) given in Listing 322. We can affect this by changing `when`; we explore this next. ■

example 94 We continue working with Listing 320.

Using the settings given in Listing 324 and running the command

```
cmh:~$ latexindent.pl textwrap8.tex -o=+-mod2.tex -l=tw-after1.yaml -m
```

gives the output given in Listing 323. ■

LISTING 323: textwrap8-mod2.tex

```
This paragraph has line breaks
throughout its paragraph; we would
like to combine the textwrapping
and paragraph removal routine.
\begin{myenv}
  This paragraph has line breaks
  throughout its paragraph; we
  would like to combine the
  textwrapping and paragraph
  removal routine.
\end{myenv}
----|----|----|----|----|----|----|----|
    5    10    15    20    25    30    35    40
```

LISTING 324: tw-after1.yaml -m

```
defaultIndent: '  '

modifyLineBreaks:
    textWrapOptions:
        columns: 35
        when: after   # <!-------
        blocksFollow:
          other: \\begin\{myenv\}
```

We note that, in Listing 323, that the wrapped text has *obeyed* the specified value of `columns` (35) given in Listing 324. ■

4.1.8 Text wrap: wrapping comments

N: 2023-01-01

You can instruct `latexindent.pl` to apply text wrapping to comments ; we demonstrate this with examples, see also Section 4.2.9.

example 95 We use the file in Listing 325 which contains a trailing comment block.

LISTING 325: textwrap9.tex

```
My first sentence
% first comment
%   second
%third comment
%     fourth
```

Using the settings given in Listing 327 and running the command

```
cmh:~$ latexindent.pl textwrap9.tex -o=+-mod1.tex -l=wrap-comments1.yaml -m
```

gives the output given in Listing 326.

LISTING 326: textwrap9-mod1.tex

```
My first sentence
% first comment second third
% comment fourth
----|----|----|----|----|----|----|----|
    5    10    15    20    25    30    35    40
```

LISTING 327: wrap-comments1.yaml -m

```
modifyLineBreaks:
  textWrapOptions:
    columns: 35
    comments:
      wrap: 1  #<!-------
```

We note that, in Listing 326, that the comments have been *combined and wrapped* because of the annotated line specified in Listing 327. ■

example 96 We use the file in Listing 328 which contains a trailing comment block.

LISTING 328: textwrap10.tex

```
My first sentence
%    first comment
%   second
%third comment
%     fourth
```

Using the settings given in Listing 330 and running the command ■

```
cmh:~$ latexindent.pl textwrap10.tex -o=+-mod1.tex -l=wrap-comments1.yaml -m
```

gives the output given in Listing 329.

LISTING 329: textwrap10-mod1.tex

```
My first sentence
% first comment second third
% comment fourth
----|----|----|----|----|----|----|----|
   5    10   15   20   25   30   35   40
```

LISTING 330: wrap-comments1.yaml -m

```
modifyLineBreaks:
  textWrapOptions:
    columns: 35
    comments:
      wrap: 1  #<!-------
```

We note that, in Listing 329, that the comments have been *combined and wrapped* because of the annotated line specified in Listing 330, and that the space from the leading comment has not been inherited; we will explore this further in the next example. ■

example 97 We continue to use the file in Listing 328.

Using the settings given in Listing 332 and running the command

```
cmh:~$ latexindent.pl textwrap10.tex -o=+-mod2.tex -l=wrap-comments2.yaml -m
```

gives the output given in Listing 331.

LISTING 331: textwrap10-mod2.tex

```
My first sentence
%    first comment second third
%    comment fourth
----|----|----|----|----|----|----|----|
   5    10   15   20   25   30   35   40
```

LISTING 332: wrap-comments2.yaml -m

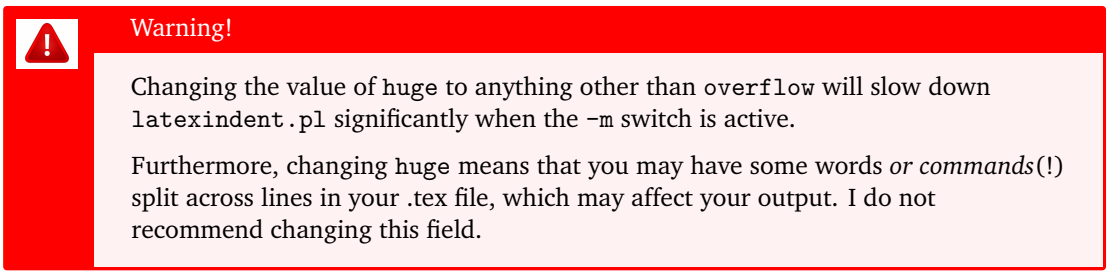
```
modifyLineBreaks:
  textWrapOptions:
    columns: 35
    comments:
      wrap: 1                  #<!-------
      inheritLeadingSpace: 1 #<!-------
```

We note that, in Listing 331, that the comments have been *combined and wrapped* and that the leading space has been inherited because of the annotated lines specified in Listing 332. ■

4.1.9 Text wrap: huge, tabstop and separator

U: 2021-07-23

The default value of `huge` is `overflow`, which means that words will *not* be broken by the text wrapping routine, implemented by the `Text::Wrap` [48]. There are options to change the `huge` option for the `Text::Wrap` module to either `wrap` or `die`. Before modifying the value of `huge`, please bear in mind the following warning:



example 98 For example, using the settings in Listings 334 and 336 and running the commands

```
cmh:~$    latexindent.pl -m textwrap4.tex -o=+-mod2A -l textwrap2A.yaml
cmh:~$    latexindent.pl -m textwrap4.tex -o=+-mod2B -l textwrap2B.yaml
```

gives the respective output in Listings 333 and 335. ■

LISTING 333: textwrap4-mod2A.tex

```
He
re
is
a
li
ne
of
te
xt
.
```

LISTING 334: textwrap2A.yaml -m

```
modifyLineBreaks:
  textWrapOptions:
    columns: 3
    huge: wrap
```

LISTING 335: textwrap4-mod2B.tex

```
Here
is
a
line
of
text.
```

LISTING 336: textwrap2B.yaml -m

```
modifyLineBreaks:
  textWrapOptions:
    columns: 3
```

■

N: 2020-11-06

You can also specify the `tabstop` field as an integer value, which is passed to the text wrap module; see [48] for details.

example 99 Starting with the code in Listing 337 with settings in Listing 338, and running the command

```
cmh:~$     latexindent.pl -m textwrap-ts.tex -o=+-mod1 -l tabstop.yaml
```

gives the code given in Listing 339.

LISTING 337: textwrap-ts.tex

```
x       y
```

LISTING 338: tabstop.yaml -m

```
modifyLineBreaks:
    textWrapOptions:
        columns: 80
        tabstop: 9
        multipleSpacesToSingle: 0
```

LISTING 339: textwrap-ts-mod1.tex

```
x       y
```

■

You can specify `separator`, `break` and `unexpand` options in your settings in analogous ways to those demonstrated in Listings 336 and 338, and they will be passed to the `Text::Wrap` module. I have not found a useful reason to do this; see [48] for more details.

4.2 oneSentencePerLine: modifying line breaks for sentences

N: 2018-01-13

You can instruct `latexindent.pl` to format your file so that it puts one sentence per line. Thank you to [7] for helping to shape and test this feature. The behaviour of this part of the script is controlled by the switches detailed in Listing 340, all of which we discuss next.

LISTING 340: oneSentencePerLine -m

```
295  oneSentencePerLine:
296    manipulateSentences: 0              # 0/1
297    removeSentenceLineBreaks: 1         # 0/1
298    multipleSpacesToSingle: 1           # 0/1
299    textWrapSentences: 0                # 1 disables main textWrap
300    sentenceIndent: ""
301    sentencesFollow:
302      par: 1                            # 0/1
303      blankLine: 1                      # 0/1
304      fullStop: 1                       # 0/1
305      exclamationMark: 1                # 0/1
306      questionMark: 1                   # 0/1
307      rightBrace: 1                     # 0/1
308      commentOnPreviousLine: 1          # 0/1
309      other: 0                          # regex
310    sentencesBeginWith:
311      A-Z: 1                            # 0/1
312      a-z: 0                            # 0/1
313      other: 0                          # regex
314    sentencesEndWith:
315      basicFullStop: 0                  # 0/1
316      betterFullStop: 1                 # 0/1
317      exclamationMark: 1                # 0/1
318      questionMark: 1                   # 0/1
319      other: 0                          # regex
320    sentencesDoNOTcontain:
321      other: \\begin                    # regex
```

4.2.1 oneSentencePerLine: overview

An overview of how the oneSentencePerLine routine feature works:

1. the default value of `manipulateSentences` is 0, which means that oneSentencePerLine will *not* happen by default;
2. it happens *after* verbatim blocks have been found;
3. it happens *before* the text wrapping routine (see Section 4.1);
4. it happens *before* the main code blocks have been found;
5. sentences to be found:
   (a) *follow* the fields specified in `sentencesFollow`
   (b) *begin* with the fields specified in `sentencesBeginWith`
   (c) *end* with the fields specified in `sentencesEndWith`
6. by default, the oneSentencePerLine routine will remove line breaks within sentences because `removeSentenceLineBreaks` is set to 1; switch it to 0 if you wish to change this;
7. sentences can be text wrapped according to `textWrapSentences`, and will be done either `before` or `after` the main indentation routine (see Section 4.2.8);
8. about trailing comments within text wrap blocks:
   (a) multiple trailing comments will be connected at the end of the sentence;
   (b) the number of spaces between the end of the sentence and the (possibly combined) trailing comments is determined by the spaces (if any) at the end of the sentence.

We demonstrate this feature using a series of examples.

`manipulateSentences`: **0|1**

This is a binary switch that details if `latexindent.pl` should perform the sentence manipulation routine; it is *off* (set to 0) by default, and you will need to turn it on (by setting it to 1) if you want the script to modify line breaks surrounding and within sentences.

`removeSentenceLineBreaks`: **0|1**

When operating upon sentences `latexindent.pl` will, by default, remove internal line breaks as `removeSentenceLineBreaks` is set to 1. Setting this switch to 0 instructs `latexindent.pl` not to do so.

example 100 For example, consider `multiple-sentences.tex` shown in Listing 341.

LISTING 341: `multiple-sentences.tex`

```
This is the first
sentence. This is the; second, sentence. This is the
third sentence.

This is the fourth
sentence! This is the fifth sentence? This is the
sixth sentence.
```

If we use the YAML files in Listings 343 and 345, and run the commands

```
cmh:~$ latexindent.pl multiple-sentences -m -l=manipulate-sentences.yaml
cmh:~$ latexindent.pl multiple-sentences -m -l=keep-sen-line-breaks.yaml
```

then we obtain the respective output given in Listings 342 and 344.

LISTING 342: `multiple-sentences.tex` using Listing 343

```
This is the first sentence.
This is the; second, sentence.
This is the third sentence.

This is the fourth sentence!
This is the fifth sentence?
This is the sixth sentence.
```

LISTING 343: `manipulate-sentences.yaml` `-m`

```
modifyLineBreaks:
    oneSentencePerLine:
        manipulateSentences: 1
```

LISTING 344: `multiple-sentences.tex` using Listing 345

```
This is the first
sentence.
This is the; second, sentence.
This is the
third sentence.

This is the fourth
sentence!
This is the fifth sentence?
This is the
sixth sentence.
```

LISTING 345: `keep-sen-line-breaks.yaml` `-m`

```
modifyLineBreaks:
    oneSentencePerLine:
        manipulateSentences: 1
        removeSentenceLineBreaks: 0
```

Notice, in particular, that the 'internal' sentence line breaks in Listing 341 have been removed in Listing 342, but have not been removed in Listing 344. ■

`multipleSpacesToSingle`: **0|1**

U: 2022-03-25

By default, the one-sentence-per-line routine will convert multiple spaces into single spaces. You can change this behaviour by changing the switch `multipleSpacesToSingle` to a value of `0`.

The remainder of the settings displayed in Listing 340 on page 87 instruct `latexindent.pl` on how to define a sentence. From the perspective of `latexindent.pl` a sentence must:

- *follow* a certain character or set of characters (see Listing 346); by default, this is either `\par`, a blank line, a full stop/period (.), exclamation mark (!), question mark (?) right brace (}) or a comment on the previous line;
- *begin* with a character type (see Listing 347); by default, this is only capital letters;
- *end* with a character (see Listing 348); by default, these are full stop/period (.), exclamation mark (!) and question mark (?).

In each case, you can specify the `other` field to include any pattern that you would like; you can specify anything in this field using the language of regular expressions.

LISTING 346: `sentencesFollow` -m

```
sentencesFollow:
  par: 1                          # 0/1
  blankLine: 1                    # 0/1
  fullStop: 1                     # 0/1
  exclamationMark: 1              # 0/1
  questionMark: 1                 # 0/1
  rightBrace: 1                   # 0/1
  commentOnPreviousLine: 1        # 0/1
  other: 0                        # regex
```

LISTING 347: `sentencesBeginWith` -m

```
sentencesBeginWith:
  A-Z: 1                          # 0/1
  a-z: 0                          # 0/1
  other: 0                        # regex
```

LISTING 348: `sentencesEndWith` -m

```
sentencesEndWith:
  basicFullStop: 0                # 0/1
  betterFullStop: 1               # 0/1
  exclamationMark: 1              # 0/1
  questionMark: 1                 # 0/1
  other: 0                        # regex
```

4.2.2 oneSentencePerLine: sentencesFollow

Let's explore a few of the switches in `sentencesFollow`.

example 101 We start with Listing 341 on the preceding page, and use the YAML settings given in Listing 350. Using the command

```
cmh:~$ latexindent.pl multiple-sentences -m -l=sentences-follow1.yaml
```

we obtain the output given in Listing 349.

LISTING 349: `multiple-sentences.tex` using Listing 350

```
This is the first sentence.
This is the; second, sentence.
This is the third sentence.

This is the fourth
sentence!
This is the fifth sentence?
This is the sixth sentence.
```

LISTING 350: `sentences-follow1.yaml` -m

```
modifyLineBreaks:
    oneSentencePerLine:
        manipulateSentences: 1
        sentencesFollow:
            blankLine: 0
```

■

Notice that, because `blankLine` is set to `0`, `latexindent.pl` will not seek sentences following a blank line, and so the fourth sentence has not been accounted for. ■

example 102 We can explore the `other` field in Listing 346 with the `.tex` file detailed in Listing 351.

LISTING 351: `multiple-sentences1.tex`

```
(Some sentences stand alone in brackets.) This is the first
sentence. This is the; second, sentence. This is the
third sentence.
```

Upon running the following commands

```
cmh:~$ latexindent.pl multiple-sentences1 -m -l=manipulate-sentences.yaml
cmh:~$ latexindent.pl multiple-sentences1 -m -l=manipulate-sentences.yaml,sentences-follow2.yaml
```

then we obtain the respective output given in Listings 352 and 353.

LISTING 352: `multiple-sentences1.tex` using Listing 343 on page 88

```
(Some sentences stand alone in brackets.) This is the first
sentence.
This is the; second, sentence.
This is the third sentence.
```

LISTING 353: `multiple-sentences1.tex` using Listing 354

```
(Some sentences stand alone in brackets.)
This is the first sentence.
This is the; second, sentence.
This is the third sentence.
```

LISTING 354: `sentences-follow2.yaml` `-m`

```
modifyLineBreaks:
    oneSentencePerLine:
        manipulateSentences: 1
        sentencesFollow:
            other: "\)"
```

Notice that in Listing 352 the first sentence after the) has not been accounted for, but that following the inclusion of Listing 354, the output given in Listing 353 demonstrates that the sentence *has* been accounted for correctly. ■

4.2.3 oneSentencePerLine: sentencesBeginWith

By default, `latexindent.pl` will only assume that sentences begin with the upper case letters `A-Z`; you can instruct the script to define sentences to begin with lower case letters (see Listing 347), and we can use the `other` field to define sentences to begin with other characters.

example 103 We use the file in Listing 355.

LISTING 355: `multiple-sentences2.tex`

```
This is the first
sentence.

$a$ can
represent a
number. 7 is
at the beginning of this sentence.
```

Upon running the following commands

```
cmh:~$ latexindent.pl multiple-sentences2 -m -l=manipulate-sentences.yaml
cmh:~$ latexindent.pl multiple-sentences2 -m -l=manipulate-sentences.yaml,sentences-begin1.yaml
```

■

then we obtain the respective output given in Listings 356 and 357.

LISTING 356: multiple-sentences2.tex using Listing 343 on page 88

```
This is the first sentence.

$a$ can
represent a
number. 7 is
at the beginning of this sentence.
```

LISTING 357: multiple-sentences2.tex using Listing 358

```
This is the first sentence.

$a$ can represent a number.
7 is at the beginning of this sentence.
```

LISTING 358: sentences-begin1.yaml -m

```
modifyLineBreaks:
    oneSentencePerLine:
        manipulateSentences: 1
        sentencesBeginWith:
            other: "\$|[0-9]"
```

Notice that in Listing 356, the first sentence has been accounted for but that the subsequent sentences have not. In Listing 357, all of the sentences have been accounted for, because the `other` field in Listing 358 has defined sentences to begin with either `$` or any numeric digit, `0` to `9`. ■

4.2.4 oneSentencePerLine: sentencesEndWith

example 104 Let's return to Listing 341 on page 88; we have already seen the default way in which `latexindent.pl` will operate on the sentences in this file in Listing 342 on page 88. We can populate the `other` field with any character that we wish; for example, using the YAML specified in Listing 360 and the command

```
cmh:~$ latexindent.pl multiple-sentences -m -l=sentences-end1.yaml
cmh:~$ latexindent.pl multiple-sentences -m -l=sentences-end2.yaml
```

then we obtain the output in Listing 359.

LISTING 359: multiple-sentences.tex using Listing 360

```
This is the first sentence.
This is the;
second, sentence.
This is the third sentence.

This is the fourth sentence!
This is the fifth sentence?
This is the sixth sentence.
```

LISTING 360: sentences-end1.yaml -m

```
modifyLineBreaks:
    oneSentencePerLine:
        manipulateSentences: 1
        sentencesEndWith:
          other: "\:|\;|\,"
```

LISTING 361: multiple-sentences.tex using Listing 362

```
This is the first sentence.
This is the;
second,
sentence.
This is the third sentence.

This is the fourth sentence!
This is the fifth sentence?
This is the sixth sentence.
```

LISTING 362: sentences-end2.yaml -m

```
modifyLineBreaks:
    oneSentencePerLine:
        manipulateSentences: 1
        sentencesEndWith:
          other: "\:|\;|\,"
        sentencesBeginWith:
          a-z: 1
```

■

There is a subtle difference between the output in Listings 359 and 361; in particular, in Listing 359 the word `sentence` has not been defined as a sentence, because we have not instructed `latexindent.pl` to begin sentences with lower case letters. We have changed this by using the settings in Listing 362, and the associated output in Listing 361 reflects this. ■

Referencing Listing 348 on page 89, you'll notice that there is a field called `basicFullStop`, which is set to 0, and that the `betterFullStop` is set to 1 by default.

example 105 Let's consider the file shown in Listing 363.

LISTING 363: `url.tex`

```
This sentence, \url{tex.stackexchange.com/} finishes here. Second sentence.
```

Upon running the following commands

```
cmh:~$ latexindent.pl url -m -l=manipulate-sentences.yaml
```

we obtain the output given in Listing 364.

LISTING 364: `url.tex` using Listing 343 on page 88

```
This sentence, \url{tex.stackexchange.com/} finishes here.
Second sentence.
```

Notice that the full stop within the url has been interpreted correctly. This is because, within the `betterFullStop`, full stops at the end of sentences have the following properties:

- they are ignored within `e.g.` and `i.e.`;
- they can not be immediately followed by a lower case or upper case letter;
- they can not be immediately followed by a hyphen, comma, or number. ■

N: 2019-07-13

If you find that the `betterFullStop` does not work for your purposes, then you can switch it off by setting it to 0, and you can experiment with the `other` field. You can also seek to customise the `betterFullStop` routine by using the *fine tuning*, detailed in Listing 542 on page 134.

The `basicFullStop` routine should probably be avoided in most situations, as it does not accommodate the specifications above.

example 106 For example, using the following command

```
cmh:~$ latexindent.pl url -m -l=alt-full-stop1.yaml
```

and the YAML in Listing 366 gives the output in Listing 365.

LISTING 365: `url.tex` using Listing 366

```
This sentence, \url{tex.
   stackexchange.com/} finishes here.
Second sentence.
```

LISTING 366: `alt-full-stop1.yaml` `-m`

```
modifyLineBreaks:
    oneSentencePerLine:
        manipulateSentences: 1
        sentencesEndWith:
            basicFullStop: 1
            betterFullStop: 0
```

Notice that the full stop within the URL has not been accommodated correctly because of the non-default settings in Listing 366. ■

4.2.5 oneSentencePerLine: sentencesDoNOTcontain

N: 2023-09-09

You can specify patterns that sentences do *not* contain using the field in Listing 367.

LISTING 367: sentencesDoNOTcontain -m

```
320     sentencesDoNOTcontain:
321       other: \\begin                    # regex
```

If sentences run across environments then, by default, they will *not* be considered a sentence by `latexindent.pl`.

U: 2023-09-09

example 107 For example, if we use the `.tex` file in Listing 368

LISTING 368: multiple-sentences4.tex

```
This sentence
\begin{itemize}
  \item continues
\end{itemize}
across itemize
and finishes here.
```

and run the command

```
cmh:~$ latexindent.pl multiple-sentences4 -m -l=manipulate-sentences.yaml
```

then the output is unchanged, because the default value of `sentencesDoNOTcontain` says, *sentences do NOT contain*

This means that, by default, `latexindent.pl` does *not* consider the file in Listing 368 to have a sentence. `\\begin`

■

example 108 We can customise the `sentencesDoNOTcontain` field with anything that we do *not* want sentences to contain.

We begin with the file in Listing 369.

LISTING 369: sentence-dnc1.tex

```
This should not be a sentence \cmh{?} and should not change.
But this
one should.
```

Upon running the following commands

```
cmh:~$ latexindent.pl sentence-dnc1.tex -m -l=dnc1.yaml
```

then we obtain the output given in Listing 370.

LISTING 370: sentence-dnc1-mod1.tex

```
This should not be a sentence \cmh{?} and should not change.
But this one should.
```

LISTING 371: dnc1.yaml -m

```
modifyLineBreaks:
    oneSentencePerLine:
        manipulateSentences: 1
        sentencesDoNOTcontain:
            other: |-
              (?x)
              \\begin
              |
              \\cmh
```

The settings in Listing 371 say that sentences do *not* contain `\begin` and that they do not contain `\cmh`

■

example 109 We can implement case insensitivity for the `sentencesDoNOTcontain` field.

We begin with the file in Listing 372.

LISTING 372: sentence-dnc2.tex

```
This should not be a sentence \cmh{?} and should not change.
This should not be a sentence \CMH{?} and should not change.
But this
one should.
```

Upon running the following commands

```
cmh:~$ latexindent.pl sentence-dnc2.tex -m -l=dnc2.yaml
```

then we obtain the output given in Listing 373.

LISTING 373: sentence-dnc2-mod2.tex

```
This should not be a sentence \cmh{?} and should not change.
This should not be a sentence \CMH{?} and should not change.
But this one should.
```

LISTING 374: dnc2.yaml -m

```
modifyLineBreaks:
    oneSentencePerLine:
        manipulateSentences: 1
        sentencesDoNOTcontain:
            other: |-
              (?xi)    #<!----
              \\begin
              |
              \\cmh
```

The settings in Listing 374 say that sentences do *not* contain `\begin` and that they do not contain *case insensitive* versions of `\cmh` ■

example 110 We can turn off `sentenceDoNOTcontain` by setting it to `0` as in Listing 375.

LISTING 375: dnc-off.yaml -m

```
modifyLineBreaks:
    oneSentencePerLine:
        manipulateSentences: 1
        sentencesDoNOTcontain: 0
```

The settings in Listing 375 mean that sentences can contain any character. ■

4.2.6 Features of the oneSentencePerLine routine

The sentence manipulation routine takes place *after* verbatim environments, preamble and trailing comments have been accounted for; this means that any characters within these types of code blocks will not be part of the sentence manipulation routine.

example 111 For example, if we begin with the `.tex` file in Listing 376, and run the command

```
cmh:~$ latexindent.pl multiple-sentences3 -m -l=manipulate-sentences.yaml
```

then we obtain the output in Listing 377. ■

LISTING 376: multiple-sentences3.tex

```
The first sentence continues after the verbatim
\begin{verbatim}
  there are sentences within this. These
  will not be operated
  upon by latexindent.pl.
\end{verbatim}
and finishes here. Second sentence % a commented full stop.
contains trailing comments,
which are ignored.
```

LISTING 377: multiple-sentences3.tex using Listing 343 on page 88

```
The first sentence continues after the verbatim \begin{verbatim}
  there are sentences within this. These
  will not be operated
  upon by latexindent.pl.
\end{verbatim} and finishes here.
Second sentence contains trailing comments, which are ignored.
% a commented full stop.
```

■

4.2.7 oneSentencePerLine: text wrapping and indenting sentences

N: 2018-08-13

The oneSentencePerLine can be instructed to perform text wrapping and indentation upon sentences.

example 112 Let's use the code in Listing 378.

LISTING 378: multiple-sentences5.tex

```
A distincao entre conteudo \emph{real} e conteudo \emph{intencional} esta
relacionada, ainda, a distincao entre o conceito husserliano de
\emph{experiencia} e o uso popular desse termo. No sentido comum,
o \term{experimentado} e um complexo de eventos exteriores,
e o \term{experimentar} consiste em percepcoes (alem de julgamentos e outros
atos) nas quais tais eventos aparecem como objetos, e objetos frequentemente
to the end.
```

Referencing Listing 380, and running the following command

```
cmh:~$ latexindent.pl multiple-sentences5 -m -l=sentence-wrap1.yaml
```

we receive the output given in Listing 379.

LISTING 379: multiple-sentences5.tex using Listing 380

```
A distincao entre conteudo \emph{real} e conteudo
  \emph{intencional} esta relacionada, ainda, a
  distincao entre o conceito husserliano de
  \emph{experiencia} e o uso popular desse termo.
No sentido comum, o \term{experimentado} e um
  complexo de eventos exteriores, e o
  \term{experimentar} consiste em percepcoes (alem
  de julgamentos e outros atos) nas quais tais
  eventos aparecem como objetos, e objetos
  frequentemente to the end.
```

LISTING 380: sentence-wrap1.yaml -m

```
modifyLineBreaks:
    oneSentencePerLine:
        manipulateSentences: 1
        removeSentenceLineBreaks: 1
        textWrapSentences: 1
        sentenceIndent: "  "
    textWrapOptions:
        columns: 50
```

■

If you specify `textWrapSentences` as 1, but do *not* specify a value for `columns` then the text wrapping will *not* operate on sentences, and you will see a warning in `indent.log`.

example 113 The indentation of sentences requires that sentences are stored as code blocks. This means that you may need to tweak Listing 348 on page 89. Let's explore this in relation to Listing 381.

LISTING 381: `multiple-sentences6.tex`

```
Consider the following:
\begin{itemize}
        \item firstly.
        \item secondly.
\end{itemize}
```

By default, `latexindent.pl` will find the full-stop within the first `item`, which means that, upon running the following commands

```
cmh:~$ latexindent.pl multiple-sentences6 -m -l=sentence-wrap1.yaml
cmh:~$ latexindent.pl multiple-sentences6 -m -l=sentence-wrap1.yaml
    -y="modifyLineBreaks:oneSentencePerLine:sentenceIndent:''"
```

we receive the respective output in Listing 382 and Listing 383.

LISTING 382: `multiple-sentences6-mod1.tex` using Listing 380

```
Consider the following:
\begin{itemize}
   \item firstly.
   \item secondly.
\end{itemize}
```

LISTING 383: `multiple-sentences6-mod2.tex` using Listing 380 and no sentence indentation

```
Consider the following:
\begin{itemize}
   \item firstly.
   \item secondly.
\end{itemize}
```

We note that Listing 382 the `itemize` code block has *not* been indented appropriately. This is because the oneSentencePerLine has been instructed to store sentences (because Listing 380); each sentence is then searched for code blocks. ■

example 114 We can tweak the settings in Listing 348 on page 89 to ensure that full stops are not followed by `item` commands, and that the end of sentences contains `\end{itemize}` as in Listing 384. This setting is actually an appended version of the `betterFullStop` from the `fineTuning`, detailed in Listing 542 on page 134. ■

LISTING 384: itemize.yaml -m

```
modifyLineBreaks:
    textWrapOptions:
        columns: 45
    oneSentencePerLine:
        sentencesEndWith:
            betterFullStop: 0
            other: |-
                (?x)
                (?:                                 # new
                 (?:\R|\h)*\\item                   # new
                )                                   # new
                |
                (?:
                  \.\)
                  (?!\h*[a-z])
                )
                |
                (?:
                  (?<!
                    (?:
                      (?:[eE]\.[gG])
                      |
                      (?:[iI]\.[eE])
                      |
                      (?:etc)
                    )
                  )
                )
                \.
                (?:\h*\R*(?:\\end\{itemize\})?) # new
                (?!
                  (?:
                    [a-zA-Z0-9-~,]
                    |
                    \),
                    |
                    \)\.
                  )
                )
```

Upon running

```
cmh:~$ latexindent.pl multiple-sentences6 -m -l=sentence-wrap1.yaml,itemize.yaml
```

we receive the output in Listing 385.

LISTING 385: multiple-sentences6-mod3.tex using Listing 380 and Listing 384

```
Consider the following:
\begin{itemize}
   \item firstly.
   \item secondly.
\end{itemize}
```

Notice that the sentence has received indentation, and that the `itemize` code block has been found and indented correctly. ■

U: 2022-04-04

Text wrapping when using the `oneSentencePerLine` routine determines if it will remove line breaks while text wrapping, from the value of `removeSentenceLineBreaks`.

4.2.8 oneSentencePerLine: text wrapping and indenting sentences, when before/after

N: 2023-01-01

The text wrapping routine operates, by default, `before` the code blocks have been found, but this can be changed to `after`:

- `before` means it is likely that the columns of wrapped text may *exceed* the value specified in `columns`;
- `after` means it columns of wrapped text should *not* exceed the value specified in `columns`.

We demonstrate this in the following examples. See also Section 4.1.7.

example 115 Let's begin with the file in Listing 386.

LISTING 386: multiple-sentences8.tex

```
This paragraph
has line breaks throughout its paragraph;
we would like to combine
the textwrapping
and paragraph removal routine.
\begin{myenv}
This paragraph
has line breaks throughout its paragraph;
we would like to combine
the textwrapping
and paragraph removal routine.
\end{myenv}
```

Using the settings given in Listing 388 and running the command

```
cmh:~$ latexindent.pl multiple-sentences8 -o=+-mod1.tex -l=sentence-wrap2 -m
```

gives the output given in Listing 387.

LISTING 387: multiple-sentences8-mod1.tex

```
This paragraph has line breaks
throughout its paragraph; we would
like to combine the textwrapping
and paragraph removal routine.
\begin{myenv}
   This paragraph has line breaks
   throughout its paragraph; we would
   like to combine the textwrapping
   and paragraph removal routine.
\end{myenv}
----|----|----|----|----|----|----|----|
   5    10   15   20   25   30   35   40
```

LISTING 388: sentence-wrap2.yaml -m

```
defaultIndent: '   '
modifyLineBreaks:
    oneSentencePerLine:
        manipulateSentences: 1
        textWrapSentences: 1
    textWrapOptions:
        columns: 35
        when: before  # <!-------
```

We note that, in Listing 387, that the wrapped text has *exceeded* the specified value of `columns` (35) given in Listing 388. We can affect this by changing `when`; we explore this next. ■

example 116 We continue working with Listing 386.

Using the settings given in Listing 390 and running the command

```
cmh:~$ latexindent.pl multiple-sentences8.tex -o=+-mod2.tex -l=sentence-wrap3 -m
```

gives the output given in Listing 389. ■

LISTING 389: multiple-sentences8-mod2.tex

```
This paragraph has line breaks
throughout its paragraph; we would
like to combine the textwrapping
and paragraph removal routine.
\begin{myenv}
   This paragraph has line breaks
   throughout its paragraph; we
   would like to combine the
   textwrapping and paragraph
   removal routine.
\end{myenv}
----|----|----|----|----|----|----|----|
   5    10   15   20   25   30   35   40
```

LISTING 390: sentence-wrap3.yaml -m

```
defaultIndent: '   '
modifyLineBreaks:
    oneSentencePerLine:
        manipulateSentences: 1
        textWrapSentences: 1
    textWrapOptions:
        columns: 35
        when: after  # <!-------
```

We note that, in Listing 389, that the wrapped text has *obeyed* the specified value of `columns` (35) given in Listing 390. ■

4.2.9 oneSentencePerLine: text wrapping sentences and comments

We demonstrate the one sentence per line routine with respect to text wrapping *comments*. See also Section 4.1.8.

example 117 Let's begin with the file in Listing 391.

LISTING 391: multiple-sentences9.tex

```
This paragraph% first comment
has line breaks throughout its paragraph;% second comment
we would like to combine% third comment
the textwrapping% fourth comment
and paragraph removal routine. % fifth comment
```

Using the settings given in Listing 393 and running the command

```
cmh:~$ latexindent.pl multiple-sentences9 -o=+-mod1.tex -l=sentence-wrap4 -m
```

gives the output given in Listing 392.

LISTING 392: multiple-sentences9-mod1.tex

```
This paragraph has line breaks
throughout its paragraph; we would
like to combine the textwrapping
and paragraph removal routine.
% first comment second comment
% third comment fourth comment
% fifth comment
----|----|----|----|----|----|----|----|
   5    10   15   20   25   30   35   40
```

LISTING 393: sentence-wrap4.yaml -m

```
defaultIndent: '   '
modifyLineBreaks:
    oneSentencePerLine:
        manipulateSentences: 1
        textWrapSentences: 1
    textWrapOptions:
        columns: 35
        comments:
          wrap: 1  #<!-------
```

We note that, in Listing 392, that the sentences have been wrapped, and so too have the comments because of the annotated line in Listing 393. ■

4.3 Poly-switches

U: 2017-08-21

Every other field in the `modifyLineBreaks` field uses poly-switches, and can take one of the following integer values:

−1 *remove mode*: line breaks before or after the *<part of thing>* can be removed (assuming that `preserveBlankLines` is set to 0);

0 *off mode*: line breaks will not be modified for the *<part of thing>* under consideration;

1 *add mode*: a line break will be added before or after the *<part of thing>* under consideration, assuming that there is not already a line break before or after the *<part of thing>*;

2 *comment then add mode*: a comment symbol will be added, followed by a line break before or after the *<part of thing>* under consideration, assuming that there is not already a comment and line break before or after the *<part of thing>*;

N: 2017-08-21

3 *add then blank line mode*: a line break will be added before or after the *<part of thing>* under consideration, assuming that there is not already a line break before or after the *<part of thing>*, followed by a blank line;

N: 2019-07-13

4 *add blank line mode*; a blank line will be added before or after the *<part of thing>* under consideration, even if the *<part of thing>* is already on its own line.

In the above, *<part of thing>* refers to either the *begin statement*, *body* or *end statement* of the code blocks detailed in Table 2 on page 49. All poly-switches are *off* by default; `latexindent.pl` searches first of all for per-name settings, and then followed by global per-thing settings.

4.3.1 Poly-switches for environments

We start by viewing a snippet of `defaultSettings.yaml` in Listing 394; note that it contains *global* settings (immediately after the `environments` field) and that *per-name* settings are also allowed – in the case of Listing 394, settings for `equation*` have been specified for demonstration. Note that all poly-switches are *off* (set to 0) by default.

LISTING 394: `environments` -m

```
360  environments:
361    BeginStartsOnOwnLine: 0            # -1,0,1,2,3,4
362    BodyStartsOnOwnLine: 0             # -1,0,1,2,3,4
363    EndStartsOnOwnLine: 0              # -1,0,1,2,3,4
364    EndFinishesWithLineBreak: 0        # -1,0,1,2,3,4
365    # equation*:
366    #     BeginStartsOnOwnLine: 0      # -1,0,1,2,3,4
367    #     BodyStartsOnOwnLine: 0       # -1,0,1,2,3,4
368    #     EndStartsOnOwnLine: 0        # -1,0,1,2,3,4
369    #     EndFinishesWithLineBreak: 0  # -1,0,1,2,3,4
```

Let's begin with the simple example given in Listing 395; note that we have annotated key parts of the file using ♠, ♡, ♢ and ♣, these will be related to fields specified in Listing 394.

LISTING 395: `env-mlb1.tex`

```
before words♠ \begin{myenv}♡body of myenv♢\end{myenv}♣ after words
```

4.3.1.1 Adding line breaks: BeginStartsOnOwnLine and BodyStartsOnOwnLine

example 118 Let's explore `BeginStartsOnOwnLine` and `BodyStartsOnOwnLine` in Listings 396 and 397, and in particular, let's allow each of them in turn to take a value of 1.

LISTING 396: `env-mlb1.yaml` -m

```
modifyLineBreaks:
    environments:
        BeginStartsOnOwnLine: 1
```

LISTING 397: `env-mlb2.yaml` -m

```
modifyLineBreaks:
    environments:
        BodyStartsOnOwnLine: 1
```

After running the following commands, ■

```
cmh:~$ latexindent.pl -m env-mlb.tex -l env-mlb1.yaml
cmh:~$ latexindent.pl -m env-mlb.tex -l env-mlb2.yaml
```

the output is as in Listings 398 and 399 respectively.

LISTING 398: env-mlb.tex using Listing 396

```
before words
\begin{myenv}body of myenv\end{myenv} after words
```

LISTING 399: env-mlb.tex using Listing 397

```
before words \begin{myenv}
    body of myenv\end{myenv} after words
```

There are a couple of points to note:

- in Listing 398 a line break has been added at the point denoted by ♠ in Listing 395; no other line breaks have been changed;
- in Listing 399 a line break has been added at the point denoted by ♡ in Listing 395; furthermore, note that the *body* of myenv has received the appropriate (default) indentation.

■

example 119 Let's now change each of the 1 values in Listings 396 and 397 so that they are 2 and save them into env-mlb3.yaml and env-mlb4.yaml respectively (see Listings 400 and 401).

LISTING 400: env-mlb3.yaml -m

```
modifyLineBreaks:
    environments:
        BeginStartsOnOwnLine: 2
```

LISTING 401: env-mlb4.yaml -m

```
modifyLineBreaks:
    environments:
        BodyStartsOnOwnLine: 2
```

Upon running the commands

```
cmh:~$ latexindent.pl -m env-mlb.tex -l env-mlb3.yaml
cmh:~$ latexindent.pl -m env-mlb.tex -l env-mlb4.yaml
```

we obtain Listings 402 and 403.

LISTING 402: env-mlb.tex using Listing 400

```
before words%
\begin{myenv}body of myenv\end{myenv} after words
```

LISTING 403: env-mlb.tex using Listing 401

```
before words \begin{myenv}%
    body of myenv\end{myenv} after words
```

Note that line breaks have been added as in Listings 398 and 399, but this time a comment symbol has been added before adding the line break; in both cases, trailing horizontal space has been stripped before doing so.

■

example 120 Let's now change each of the 1 values in Listings 396 and 397 so that they are 3 and save them into env-mlb5.yaml and env-mlb6.yaml respectively (see Listings 404 and 405).

N: 2017-08-21

LISTING 404: env-mlb5.yaml -m

```
modifyLineBreaks:
    environments:
        BeginStartsOnOwnLine: 3
```

LISTING 405: env-mlb6.yaml -m

```
modifyLineBreaks:
    environments:
        BodyStartsOnOwnLine: 3
```

Upon running the commands

```
cmh:~$ latexindent.pl -m env-mlb.tex -l env-mlb5.yaml
cmh:~$ latexindent.pl -m env-mlb.tex -l env-mlb6.yaml
```

we obtain Listings 406 and 407.

■

LISTING 406: env-mlb.tex using Listing 404

```
before words

\begin{myenv}body of myenv\end{myenv} after words
```

LISTING 407: env-mlb.tex using Listing 405

```
before words \begin{myenv}

    body of myenv\end{myenv} after words
```

Note that line breaks have been added as in Listings 398 and 399, but this time a *blank line* has been added after adding the line break. ■

example 121 Let's now change each of the 1 values in Listings 404 and 405 so that they are 4 and save them into env-beg4.yaml and env-body4.yaml respectively (see Listings 408 and 409).

N: 2019-07-13

LISTING 408: env-beg4.yaml -m

```
modifyLineBreaks:
    environments:
        BeginStartsOnOwnLine: 4
```

LISTING 409: env-body4.yaml -m

```
modifyLineBreaks:
    environments:
        BodyStartsOnOwnLine: 4
```

We will demonstrate this poly-switch value using the code in Listing 410.

LISTING 410: env-mlb1.tex

```
before words
\begin{myenv}
body of myenv
\end{myenv}
after words
```

Upon running the commands

```
cmh:~$ latexindent.pl -m env-mlb1.tex -l env-beg4.yaml
cmh:~$ latexindent.pl -m env-mlb.1tex -l env-body4.yaml
```

then we receive the respective outputs in Listings 411 and 412.

LISTING 411: env-mlb1.tex using Listing 408

```
before words

\begin{myenv}
   body of myenv
\end{myenv}
after words
```

LISTING 412: env-mlb1.tex using Listing 409

```
before words
\begin{myenv}

   body of myenv
\end{myenv}
after words
```

We note in particular that, by design, for this value of the poly-switches:

1. in Listing 411 a blank line has been inserted before the \begin statement, even though the \begin statement was already on its own line;
2. in Listing 412 a blank line has been inserted before the beginning of the *body*, even though it already began on its own line. ■

4.3.1.2 Adding line breaks: EndStartsOnOwnLine and EndFinishesWithLineBreak

example 122 Let's explore EndStartsOnOwnLine and EndFinishesWithLineBreak in Listings 413 and 414, and in particular, let's allow each of them in turn to take a value of 1. ■

LISTING 413: env-mlb7.yaml -m

```
modifyLineBreaks:
    environments:
        EndStartsOnOwnLine: 1
```

LISTING 414: env-mlb8.yaml -m

```
modifyLineBreaks:
    environments:
        EndFinishesWithLineBreak: 1
```

After running the following commands,

```
cmh:~$ latexindent.pl -m env-mlb.tex -l env-mlb7.yaml
cmh:~$ latexindent.pl -m env-mlb.tex -l env-mlb8.yaml
```

the output is as in Listings 415 and 416.

LISTING 415: env-mlb.tex using Listing 413

```
before words \begin{myenv}body
    of myenv
\end{myenv} after words
```

LISTING 416: env-mlb.tex using Listing 414

```
before words \begin{myenv}body
    of myenv\end{myenv}
after words
```

There are a couple of points to note:

- in Listing 415 a line break has been added at the point denoted by ♢ in Listing 395 on page 100; no other line breaks have been changed and the \end{myenv} statement has *not* received indentation (as intended);
- in Listing 416 a line break has been added at the point denoted by ♣ in Listing 395 on page 100.

■

example 123 Let's now change each of the 1 values in Listings 413 and 414 so that they are 2 and save them into env-mlb9.yaml and env-mlb10.yaml respectively (see Listings 417 and 418).

LISTING 417: env-mlb9.yaml -m

```
modifyLineBreaks:
    environments:
        EndStartsOnOwnLine: 2
```

LISTING 418: env-mlb10.yaml -m

```
modifyLineBreaks:
    environments:
        EndFinishesWithLineBreak: 2
```

Upon running the commands

```
cmh:~$ latexindent.pl -m env-mlb.tex -l env-mlb9.yaml
cmh:~$ latexindent.pl -m env-mlb.tex -l env-mlb10.yaml
```

we obtain Listings 419 and 420.

LISTING 419: env-mlb.tex using Listing 417

```
before words \begin{myenv}body of myenv%
\end{myenv} after words
```

LISTING 420: env-mlb.tex using Listing 418

```
before words \begin{myenv}body of myenv\end{myenv}%
after words
```

Note that line breaks have been added as in Listings 415 and 416, but this time a comment symbol has been added before adding the line break; in both cases, trailing horizontal space has been stripped before doing so.

■

example 124 N: 2017-08-21 Let's now change each of the 1 values in Listings 413 and 414 so that they are 3 and save them into env-mlb11.yaml and env-mlb12.yaml respectively (see Listings 421 and 422).

■

LISTING 421: env-mlb11.yaml -m

```
modifyLineBreaks:
    environments:
        EndStartsOnOwnLine: 3
```

LISTING 422: env-mlb12.yaml -m

```
modifyLineBreaks:
    environments:
        EndFinishesWithLineBreak: 3
```

Upon running the commands

```
cmh:~$ latexindent.pl -m env-mlb.tex -l env-mlb11.yaml
cmh:~$ latexindent.pl -m env-mlb.tex -l env-mlb12.yaml
```

we obtain Listings 423 and 424.

LISTING 423: env-mlb.tex using Listing 421

```
before words \begin{myenv}body of myenv

\end{myenv} after words
```

LISTING 424: env-mlb.tex using Listing 422

```
before words \begin{myenv}body of myenv\end{myenv}

after words
```

Note that line breaks have been added as in Listings 415 and 416, and that a *blank line* has been added after the line break. ■

example 125

N: 2019-07-13

Let's now change each of the 1 values in Listings 421 and 422 so that they are 4 and save them into env-end4.yaml and env-end-f4.yaml respectively (see Listings 425 and 426).

LISTING 425: env-end4.yaml -m

```
modifyLineBreaks:
    environments:
        EndStartsOnOwnLine: 4
```

LISTING 426: env-end-f4.yaml -m

```
modifyLineBreaks:
    environments:
        EndFinishesWithLineBreak: 4
```

We will demonstrate this poly-switch value using the code from Listing 410 on page 102.

Upon running the commands

```
cmh:~$ latexindent.pl -m env-mlb1.tex -l env-end4.yaml
cmh:~$ latexindent.pl -m env-mlb.1tex -l env-end-f4.yaml
```

then we receive the respective outputs in Listings 427 and 428.

LISTING 427: env-mlb1.tex using Listing 425

```
before words
\begin{myenv}
   body of myenv

\end{myenv}
after words
```

LISTING 428: env-mlb1.tex using Listing 426

```
before words
\begin{myenv}
   body of myenv
\end{myenv}

after words
```

We note in particular that, by design, for this value of the poly-switches:

1. in Listing 427 a blank line has been inserted before the \end statement, even though the \end statement was already on its own line;
2. in Listing 428 a blank line has been inserted after the \end statement, even though it already began on its own line.

■

4.3.1.3 poly-switches 1, 2, and 3 only add line breaks when necessary

If you ask latexindent.pl to add a line break (possibly with a comment) using a poly-switch value of 1 (or 2 or 3), it will only do so if necessary.

example 126 For example, if you process the file in Listing 429 using poly-switch values of 1, 2, or 3, it will be left unchanged.

LISTING 429: `env-mlb2.tex`

```
before words
\begin{myenv}
  body of myenv
\end{myenv}
after words
```

LISTING 430: `env-mlb3.tex`

```
before words
\begin{myenv}  %
  body of myenv%
\end{myenv}%
after words
```

■

Setting the poly-switches to a value of 4 instructs `latexindent.pl` to add a line break even if the *<part of thing>* is already on its own line; see Listings 411 and 412 and Listings 427 and 428.

example 127 In contrast, the output from processing the file in Listing 430 will vary depending on the poly-switches used; in Listing 431 you'll see that the comment symbol after the `\begin{myenv}` has been moved to the next line, as `BodyStartsOnOwnLine` is set to 1. In Listing 432 you'll see that the comment has been accounted for correctly because `BodyStartsOnOwnLine` has been set to 2, and the comment symbol has *not* been moved to its own line. You're encouraged to experiment with Listing 430 and by setting the other poly-switches considered so far to 2 in turn.

LISTING 431: `env-mlb3.tex` using Listing 397 on page 100

```
before words
\begin{myenv}
   %
   body of myenv%
\end{myenv}%
after words
```

LISTING 432: `env-mlb3.tex` using Listing 401 on page 101

```
before words
\begin{myenv}  %
   body of myenv%
\end{myenv}%
after words
```

■

The details of the discussion in this section have concerned *global* poly-switches in the `environments` field; each switch can also be specified on a *per-name* basis, which would take priority over the global values; with reference to Listing 394 on page 100, an example is shown for the `equation*` environment.

4.3.1.4 Removing line breaks (poly-switches set to −1)

Setting poly-switches to −1 tells `latexindent.pl` to remove line breaks of the *<part of the thing>*, if necessary.

example 128 We will consider the example code given in Listing 433, noting in particular the positions of the line break highlighters, ♠, ♡, ♢ and ♣, together with the associated YAML files in Listings 434 to 437.

■

LISTING 433: env-mlb4.tex

```
before words♠
\begin{myenv}♡
body of myenv♢
\end{myenv}♣
after words
```

LISTING 434: env-mlb13.yaml -m

```
modifyLineBreaks:
    environments:
        BeginStartsOnOwnLine: -1
```

LISTING 435: env-mlb14.yaml -m

```
modifyLineBreaks:
    environments:
        BodyStartsOnOwnLine: -1
```

LISTING 436: env-mlb15.yaml -m

```
modifyLineBreaks:
    environments:
        EndStartsOnOwnLine: -1
```

LISTING 437: env-mlb16.yaml -m

```
modifyLineBreaks:
    environments:
        EndFinishesWithLineBreak:
    -1
```

After running the commands

```
cmh:~$ latexindent.pl -m env-mlb4.tex -l env-mlb13.yaml
cmh:~$ latexindent.pl -m env-mlb4.tex -l env-mlb14.yaml
cmh:~$ latexindent.pl -m env-mlb4.tex -l env-mlb15.yaml
cmh:~$ latexindent.pl -m env-mlb4.tex -l env-mlb16.yaml
```

we obtain the respective output in Listings 438 to 441.

LISTING 438: env-mlb4.tex using Listing 434

```
before words\begin{myenv}
   body of myenv
\end{myenv}
after words
```

LISTING 439: env-mlb4.tex using Listing 435

```
before words
\begin{myenv}body of myenv
\end{myenv}
after words
```

LISTING 440: env-mlb4.tex using Listing 436

```
before words
\begin{myenv}
   body of myenv\end{myenv}
after words
```

LISTING 441: env-mlb4.tex using Listing 437

```
before words
\begin{myenv}
   body of myenv
\end{myenv}after words
```

Notice that in:

- Listing 438 the line break denoted by ♠ in Listing 433 has been removed;
- Listing 439 the line break denoted by ♡ in Listing 433 has been removed;
- Listing 440 the line break denoted by ♢ in Listing 433 has been removed;
- Listing 441 the line break denoted by ♣ in Listing 433 has been removed.

We examined each of these cases separately for clarity of explanation, but you can combine all of the YAML settings in Listings 434 to 437 into one file; alternatively, you could tell `latexindent.pl` to load them all by using the following command, for example

```
cmh:~$
    latexindent.pl -m env-mlb4.tex -l env-mlb13.yaml,env-mlb14.yaml,env-mlb15.yaml,env-mlb16.yaml
```

which gives the output in Listing 395 on page 100.

4.3.1.5 About trailing horizontal space

Recall that on page 26 we discussed the YAML field `removeTrailingWhitespace`, and that it has two (binary) switches to determine if horizontal space should be removed `beforeProcessing` and `afterProcessing`. The `beforeProcessing` is particularly relevant when considering the `-m` switch.

example 129 We consider the file shown in Listing 442, which highlights trailing spaces.

LISTING 442: env-mlb5.tex

```
before␣words␣␣␣♠
\begin{myenv}␣␣␣␣␣␣␣␣␣␣␣♡
body␣of␣myenv␣␣␣␣␣␣◇
\end{myenv}␣␣␣␣␣♣
after␣words
```

LISTING 443: removeTWS-before.yaml

```
removeTrailingWhitespace:
    beforeProcessing: 1
```

The output from the following commands

```
cmh:~$ latexindent.pl -m env-mlb5.tex -l env-mlb13,env-mlb14,env-mlb15,env-mlb16
cmh:~$ latexindent.pl -m env-mlb5.tex -l env-mlb13,env-mlb14,env-mlb15,env-mlb16,removeTWS-before
```

is shown, respectively, in Listings 444 and 445; note that the trailing horizontal white space has been preserved (by default) in Listing 444, while in Listing 445, it has been removed using the switch specified in Listing 443.

LISTING 444: env-mlb5.tex using Listings 438 to 441

```
before␣words␣␣␣\begin{myenv}␣␣␣␣␣␣␣␣␣␣␣body␣of␣myenv␣␣␣␣␣␣\end{myenv}␣␣␣␣␣after␣words
```

LISTING 445: env-mlb5.tex using Listings 438 to 441 *and* Listing 443

```
before␣words\begin{myenv}body␣of␣myenv\end{myenv}after␣words
```

4.3.1.6 poly-switch line break removal and blank lines

example 130 Now let's consider the file in Listing 446, which contains blank lines.

LISTING 446: env-mlb6.tex

```
before words♠


\begin{myenv}♡


body of myenv◇


\end{myenv}♣

after words
```

LISTING 447: UnpreserveBlankLines.yaml `-m`

```
modifyLineBreaks:
    preserveBlankLines: 0
```

Upon running the following commands

```
cmh:~$ latexindent.pl -m env-mlb6.tex -l env-mlb13,env-mlb14,env-mlb15,env-mlb16
cmh:~$
    latexindent.pl -m env-mlb6.tex -l env-mlb13,env-mlb14,env-mlb15,env-mlb16,UnpreserveBlankLines
```

we receive the respective outputs in Listings 448 and 449. In Listing 448 we see that the multiple blank lines have each been condensed into one blank line, but that blank lines have *not* been removed by the poly-switches – this is because, by default, `preserveBlankLines` is set to 1. By contrast, in Listing 449, we have allowed the poly-switches to remove blank lines because, in Listing 447, we have set `preserveBlankLines` to 0.

LISTING 448: env-mlb6.tex using Listings 438 to 441

```
before words

\begin{myenv}

   body of myenv

\end{myenv}

after words
```

LISTING 449: env-mlb6.tex using Listings 438 to 441 *and* Listing 447

```
before words\begin{myenv}body of myenv\end{myenv}after words
```

■

example 131 We can explore this further using the blank-line poly-switch value of 3; let's use the file given in Listing 450.

LISTING 450: env-mlb7.tex

```
\begin{one} one text \end{one} \begin{two} two text \end{two}
```

Upon running the following commands

```
cmh:~$ latexindent.pl -m env-mlb7.tex -l env-mlb12.yaml,env-mlb13.yaml
cmh:~$
    latexindent.pl -m env-mlb7.tex -l env-mlb13,env-mlb14,UnpreserveBlankLines
```

we receive the outputs given in Listings 451 and 452.

LISTING 451: env-mlb7-preserve.tex

```
\begin{one} one text \end{one}

\begin{two} two text \end{two}
```

LISTING 452: env-mlb7-no-preserve.tex

```
\begin{one} one text \end{one} \begin{two} two text \end{two}
```

Notice that in:

- Listing 451 that `\end{one}` has added a blank line, because of the value of `EndFinishesWithLineBreak` in Listing 422 on page 104, and even though the line break ahead of `\begin{two}` should have been removed (because of `BeginStartsOnOwnLine` in Listing 434 on page 106), the blank line has been preserved by default;
- Listing 452, by contrast, has had the additional line-break removed, because of the settings in Listing 447.

■

4.3.2 Poly-switches for double backslash

N: 2019-07-13

With reference to `lookForAlignDelims` (see Listing 47 on page 26) you can specify poly-switches to dictate the line-break behaviour of double backslashes in environments (Listing 49 on page 27), commands (Listing 116 on page 38), or special code blocks (Listing 124 on page 40). [3]

Consider the code given in Listing 453.

LISTING 453: `tabular3.tex`

```
\begin{tabular}{cc}
 1 & 2 ★\\□ 3 & 4 ★\\□
\end{tabular}
```

Referencing Listing 453:

- DBS stands for *double backslash*;
- line breaks ahead of the double backslash are annotated by ★, and are controlled by `DBSStartsOnOwnLine`;
- line breaks after the double backslash are annotated by □, and are controlled by `DBSFinishesWithLineBreak`.

Let's explore each of these in turn.

4.3.2.1 Double backslash starts on own line

example 132 We explore `DBSStartsOnOwnLine` (★ in Listing 453); starting with the code in Listing 453, together with the YAML files given in Listing 455 and Listing 457 and running the following commands

```
cmh:~$ latexindent.pl -m tabular3.tex -l DBS1.yaml
cmh:~$ latexindent.pl -m tabular3.tex -l DBS2.yaml
```

then we receive the respective output given in Listing 454 and Listing 456.

LISTING 454: `tabular3.tex` using Listing 455

```
\begin{tabular}{cc}
   1 & 2
   \\ 3 & 4
   \\
\end{tabular}
```

LISTING 455: `DBS1.yaml` `-m`

```
modifyLineBreaks:
    environments:
        DBSStartsOnOwnLine: 1
```

LISTING 456: `tabular3.tex` using Listing 457

```
\begin{tabular}{cc}
   1 & 2 %
   \\ 3 & 4 %
   \\
\end{tabular}
```

LISTING 457: `DBS2.yaml` `-m`

```
modifyLineBreaks:
    environments:
        tabular:
            DBSStartsOnOwnLine: 2
```

We note that

- Listing 455 specifies `DBSStartsOnOwnLine` for *every* environment (that is within `lookForAlignDelims`, Listing 50 on page 27); the double backslashes from Listing 453 have been moved to their own line in Listing 454;
- Listing 457 specifies `DBSStartsOnOwnLine` on a *per-name* basis for `tabular` (that is within `lookForAlignDelims`, Listing 50 on page 27); the double backslashes from Listing 453 have been moved to their own line in Listing 456, having added comment symbols before moving them.

■

[3] There is no longer any need for the code block to be specified within `lookForAlignDelims` for DBS poly-switches to activate.

U: 2023-01-01

example 133 We can combine DBS poly-switches with, for example, the `alignContentAfterDoubleBackSlash` in Section 3.5.5 on page 37.

For example, starting with the file Listing 458, and using the settings in Listings 112 and 114 on page 38 and running

```
cmh:~$ latexindent.pl -s -m -l alignContentAfterDBS1.yaml,DBS1.yaml tabular6.tex -o=+-mod1
cmh:~$ latexindent.pl -s -m -l alignContentAfterDBS2.yaml,DBS1.yaml tabular6.tex -o=+-mod2
```

gives the respective outputs shown in Listings 459 and 460.

LISTING 458: tabular6.tex

```
\begin{tabular}{cc}
 1&22\\333&4444\\55555&666666
\end{tabular}
```

LISTING 459: tabular6-mod1.tex

```
\begin{tabular}{cc}
      1     & 22
   \\ 333   & 4444
   \\ 55555 & 666666
\end{tabular}
```

LISTING 460: tabular6-mod2.tex

```
\begin{tabular}{cc}
      1     & 22
   \\ 333   & 4444
   \\ 55555 & 666666
\end{tabular}
```

We note that:

- in Listing 459 the content *after* the double back slash has been aligned;
- in Listing 460 we see that 3 spaces have been added after the double back slash. ■

4.3.2.2 Double backslash finishes with line break

example 134 Let's now explore `DBSFinishesWithLineBreak` (□ in Listing 453); starting with the code in Listing 453, together with the YAML files given in Listing 462 and Listing 464 and running the following commands

```
cmh:~$ latexindent.pl -m tabular3.tex -l DBS3.yaml
cmh:~$ latexindent.pl -m tabular3.tex -l DBS4.yaml
```

then we receive the respective output given in Listing 461 and Listing 463.

LISTING 461: tabular3.tex using Listing 462

```
\begin{tabular}{cc}
   1 & 2 \\
   3 & 4 \\
\end{tabular}
```

LISTING 462: DBS3.yaml -m

```
modifyLineBreaks:
    environments:
        DBSFinishesWithLineBreak: 1
```

LISTING 463: tabular3.tex using Listing 464

```
\begin{tabular}{cc}
   1 & 2 \\%
   3 & 4 \\
\end{tabular}
```

LISTING 464: DBS4.yaml -m

```
modifyLineBreaks:
 environments:
   tabular:
     DBSFinishesWithLineBreak: 2
```

We note that

- Listing 462 specifies `DBSFinishesWithLineBreak` for *every* environment (that is within `lookForAlignDelims`, Listing 50 on page 27); the code following the double backslashes from Listing 453 has been moved to their own line in Listing 461;
- Listing 464 specifies `DBSFinishesWithLineBreak` on a *per-name* basis for `tabular` (that is within `lookForAlignDelims`, Listing 50 on page 27); the first double backslashes from Listing 453 have moved code following them to their own line in Listing 463, having added comment symbols before moving them; the final double backslashes have *not* added a line ■

break as they are at the end of the body within the code block.

4.3.2.3 Double backslash poly-switches for specialBeginEnd

example 135 Let's explore the double backslash poly-switches for code blocks within `specialBeginEnd` code blocks (Listing 122 on page 39); we begin with the code within Listing 465.

LISTING 465: special4.tex

```
\< a& =b  \\ & =c\\ & =d\\ & =e \>
```

Upon using the YAML settings in Listing 467, and running the command

```
cmh:~$ latexindent.pl -m special4.tex -l DBS5.yaml
```

then we receive the output given in Listing 466.

LISTING 466: special4.tex using Listing 467

```
\<
   a & =b \\
     & =c \\
     & =d \\
     & =e %
\>
```

LISTING 467: DBS5.yaml -m

```
specialBeginEnd:
  - name: cmhMath
    begin: '\\<'
    end: '\\>'
lookForAlignDelims:
    cmhMath: 1
modifyLineBreaks:
 specialBeginEnd:
  cmhMath:
   DBSFinishesWithLineBreak: 1
   SpecialBodyStartsOnOwnLine: 1
   SpecialEndStartsOnOwnLine: 2
```

There are a few things to note:

- in Listing 467 we have specified `cmhMath` within `lookForAlignDelims`; without this, the double backslash poly-switches would be ignored for this code block;
- the `DBSFinishesWithLineBreak` poly-switch has controlled the line breaks following the double backslashes;
- the `SpecialEndStartsOnOwnLine` poly-switch has controlled the addition of a comment symbol, followed by a line break, as it is set to a value of 2.

4.3.2.4 Double backslash poly-switches for optional and mandatory arguments

For clarity, we provide a demonstration of controlling the double backslash poly-switches for optional and mandatory arguments.

example 136 We use with the code in Listing 468.

LISTING 468: mycommand2.tex

```
\mycommand [
   1&2   &3\\ 4&5&6]{
7&8   &9\\ 10&11&12
}
```

Upon using the YAML settings in Listings 470 and 472, and running the command

```
cmh:~$ latexindent.pl -m mycommand2.tex -l DBS6.yaml
cmh:~$ latexindent.pl -m mycommand2.tex -l DBS7.yaml
```

then we receive the output given in Listings 469 and 471.

LISTING 469: mycommand2.tex using Listing 470

```
\mycommand [
   1 & 2 & 3 %
   \\%
   4 & 5 & 6]{
   7 & 8 & 9 \\ 10&11&12
}
```

LISTING 470: DBS6.yaml -m

```
lookForAlignDelims:
    mycommand: 1
modifyLineBreaks:
    optionalArguments:
        DBSStartsOnOwnLine: 2
        DBSFinishesWithLineBreak: 2
```

LISTING 471: mycommand2.tex using Listing 472

```
\mycommand [
   1 & 2 & 3 \\ 4&5&6]{
   7  & 8  & 9  %
   \\%
   10 & 11 & 12
}
```

LISTING 472: DBS7.yaml -m

```
lookForAlignDelims:
    mycommand: 1
modifyLineBreaks:
    mandatoryArguments:
        DBSStartsOnOwnLine: 2
        DBSFinishesWithLineBreak: 2
```

■

4.3.2.5 Double backslash optional square brackets

The pattern matching for the double backslash will also, optionally, allow trailing square brackets that contain a measurement of vertical spacing, for example `\\[3pt]`.

example 137 For example, beginning with the code in Listing 473

LISTING 473: pmatrix3.tex

```
\begin{pmatrix}
1 & 2 \\[2pt] 3 & 4 \\ [ 3 ex] 5&6\\[    4  pt   ] 7 & 8
\end{pmatrix}
```

and running the following command, using Listing 462,

```
cmh:~$ latexindent.pl -m pmatrix3.tex -l DBS3.yaml
```

then we receive the output given in Listing 474.

LISTING 474: pmatrix3.tex using Listing 462

```
\begin{pmatrix}
   1 & 2 \\[2pt]
   3 & 4 \\ [ 3 ex]
   5 & 6 \\[    4  pt   ]
   7 & 8
\end{pmatrix}
```

■

You can customise the pattern for the double backslash by exploring the *fine tuning* field detailed in Listing 542 on page 134.

4.3.3 Poly-switches for commas

With reference to Table 3 you can specify poly-switches to dictate the line-break behaviour of commas in arguments for code blocks that have arguments: `commands`, `namedGroupingBracesBrackets`, `keyEqualsValuesBracesBrackets` and `UnNamedGroupingBracesBrackets`.

Consider the code given in Listing 475.

LISTING 475: comma1.tex

```
\mycommand{1★,□2★,□3}[4★,□5★,□6]
```

Referencing Listing 475:

- line breaks ahead of the comma are annotated by ★, and are controlled by `CommaStartsOnOwnLine`;
- line breaks after the comma are annotated by □, and are controlled by `CommaFinishesWithLineBreak`.

Let's explore each of these in turn.

4.3.3.1 Comma starts on own line

example 138 We explore `CommaStartsOnOwnLine` (★ in Listing 475); starting with the code in Listing 475, together with the YAML files given in Listing 477 and Listing 479 and running the following commands

```
cmh:~$ latexindent.pl comma1.tex -l comma1.yaml
cmh:~$ latexindent.pl comma1.tex -l comma2.yaml
```

then we receive the respective output given in Listing 476 and Listing 478.

LISTING 476: comma1-mod1.tex

```
\mycommand{1
    ,2
    ,3}[4
    ,5
    ,6]
```

LISTING 477: comma1.yaml -m

```
switchesViaYaml:
  mSwitch: 1
modifyLineBreaks:
  optionalArguments:
    CommaStartsOnOwnLine: 1
  mandatoryArguments:
    CommaStartsOnOwnLine: 1
```

LISTING 478: comma1-mod2.tex

```
\mycommand{1%
    ,2%
    ,3}[4%
    ,5%
    ,6]
```

LISTING 479: comma2.yaml -m

```
switchesViaYaml:
  mSwitch: 1
modifyLineBreaks:
  optionalArguments:
    mycommand:
      CommaStartsOnOwnLine: 2
  mandatoryArguments:
    mycommand:
      CommaStartsOnOwnLine: 2
```

We note that

- Listing 477 specifies `CommaStartsOnOwnLine` for *every* mandatory and optional argument; the commas Listing 475 have been moved to their own line in Listing 476;
- Listing 479 specifies `CommaStartsOnOwnLine` on a *per-name* basis for `mycommand`; the commas from Listing 475 have been moved to their own line in Listing 478, having added comment symbols before moving them.

■

4.3.3.2 Comma finishes with line break

example 139 Let's now explore `CommaFinishesWithLineBreak` (□ in Listing 475); starting with the code in Listing 475, together with the YAML files given in Listing 481 and Listing 483 and running the following commands

```
cmh:~$ latexindent.pl comma1.tex -l comma3.yaml
cmh:~$ latexindent.pl comma1.tex -l comma4.yaml
```

then we receive the respective output given in Listing 480 and Listing 482.

■

LISTING 480: comma1-mod3.tex

```
\mycommand{1,
   2,
   3}[4,
   5,
   6]
```

LISTING 481: comma3.yaml -m

```
switchesViaYaml:
  mSwitch: 1
modifyLineBreaks:
  optionalArguments:
    CommaFinishesWithLineBreak: 1
  mandatoryArguments:
    CommaFinishesWithLineBreak: 1
```

LISTING 482: comma1-mod4.tex

```
\mycommand{1,%
   2,%
   3}[4,%
   5,%
   6]
```

LISTING 483: comma4.yaml -m

```
switchesViaYaml:
  mSwitch: 1
modifyLineBreaks:
  optionalArguments:
    mycommand:
      CommaFinishesWithLineBreak: 2
  mandatoryArguments:
    mycommand:
      CommaFinishesWithLineBreak: 2
```

We note that

- Listing 481 specifies `CommaFinishesWithLineBreak` for *every* mandatory and optional argument; the code following the commas from Listing 475 has been moved to their own line in Listing 480;
- Listing 483 specifies `CommaFinishesWithLineBreak` on a *per-name* basis for `mycommand`; the code following the commas has been moved to their own line in Listing 482, having added comment symbols before moving them.

■

4.3.4 Poly-switches for other code blocks

Rather than repeat the examples shown for the environment code blocks (in Section 4.3.1 on page 100), we choose to detail the poly-switches for all other code blocks in Table 3; note that each and every one of these poly-switches is *off by default*, i.e, set to `0`.

U: 2019-05-05

Note also that, by design, line breaks involving, `filecontents` and 'comment-marked' code blocks (Listing 117 on page 38) can *not* be modified using `latexindent.pl`. However, there are two poly-switches available for `verbatim` code blocks: environments (Listing 27 on page 22), commands (Listing 28 on page 22) and `specialBeginEnd` (Listing 137 on page 44).

TABLE 3: Poly-switch mappings for all code-block types

Code block	Sample		Poly-switch mapping
environment	before words♠ \begin{myenv}♡ body of myenv♢ \end{myenv}♣ after words	♠ ♡ ♢ ♣	BeginStartsOnOwnLine BodyStartsOnOwnLine EndStartsOnOwnLine EndFinishesWithLineBreak
ifelsefi	before words♠ \if...♡ body of if/or statement▲ \or▼ body of if/or statement★ \else□ body of else statement♢ \fi♣ after words	♠ ♡ ▲ ▼ ★ □ ♢ ♣	IfStartsOnOwnLine BodyStartsOnOwnLine OrStartsOnOwnLine OrFinishesWithLineBreak ElseStartsOnOwnLine ElseFinishesWithLineBreak FiStartsOnOwnLine FiFinishesWithLineBreak
optionalArguments	...♠ [♡ value before comma★, □ end of body of opt arg♢]♣ ...	♠ ♡ ★ □ ♢ ♣	LSqBStartsOnOwnLine[4] OptArgBodyStartsOnOwnLine CommaStartsOnOwnLine CommaFinishesWithLineBreak RSqBStartsOnOwnLine RSqBFinishesWithLineBreak
mandatoryArguments	...♠ {♡ value before comma★, □ end of body of mand arg♢ }♣ ...	♠ ♡ ★ □ ♢ ♣	LCuBStartsOnOwnLine[5] MandArgBodyStartsOnOwnLine CommaStartsOnOwnLine CommaFinishesWithLineBreak RCuBStartsOnOwnLine RCuBFinishesWithLineBreak
commands	before words♠ \mycommand ⟨*arguments*⟩	♠	CommandStartsOnOwnLine
namedGroupingBracesBrackets	before words♠ myname ⟨*braces/brackets*⟩	♠	NameStartsOnOwnLine
keyEqualsValuesBracesBrackets	before words♠ key•=♡ ⟨*braces/brackets*⟩	♠ • ♡	KeyStartsOnOwnLine EqualsStartsOnOwnLine EqualsFinishesWithLineBreak
items	before words♠ \item♡ ...	♠ ♡	ItemStartsOnOwnLine ItemFinishesWithLineBreak
specialBeginEnd	before words♠ \[♡ body of special/middle★ \middle□ body of special/middle ♢ \]♣ after words	♠ ♡ ★ □ ♢ ♣	SpecialBeginStartsOnOwnLine SpecialBodyStartsOnOwnLine SpecialMiddleStartsOnOwnLine SpecialMiddleFinishesWithLineBrea SpecialEndStartsOnOwnLine SpecialEndFinishesWithLineBreak
verbatim	before words♠\begin{verbatim}	♠	VerbatimBeginStartsOnOwnLine

[4]LSqB stands for Left Square Bracket
[5]LCuB stands for Left Curly Brace

body of verbatim	\end{verbatim}♣ ♣ VerbatimEndFinishesWithLineBreak
after words	

4.3.5 Conflicting poly-switches: sequential code blocks

It is very easy to have conflicting poly-switches. The most poly-switch corresponding to the most-recently-found code block will be respected.

example 140 We use the Listing 484

LISTING 484: mycommand1.tex

```
\mycommand
{
mand arg text
mand arg text}
{
mand arg text
mand arg text}
```

and consider the YAML settings given in Listing 486. The output from running

```
cmh:~$ latexindent.pl -m -l=mycom-mlb4.yaml mycommand1.tex
```

is given in Listing 486.

LISTING 485: mycommand1.tex using Listing 486

```
\mycommand{
   mand arg text
   mand arg text}{
   mand arg text
   mand arg text}
```

LISTING 486: mycom-mlb4.yaml -m

```
modifyLineBreaks:
    mandatoryArguments:
        LCuBStartsOnOwnLine: -1
        RCuBFinishesWithLineBreak: 1
```

Studying Listing 486, we see that the two poly-switches are at opposition with one another:

- on the one hand, LCuBStartsOnOwnLine should *not* start on its own line (as poly-switch is set to −1);
- on the other hand, RCuBFinishesWithLineBreak *should* finish with a line break.

So, which should win the conflict? As demonstrated in Listing 485, it is clear that LCuBStartsOnOwnLine won this conflict, and the reason is that *the second argument was processed after the first* – in general, the most recently-processed code block and associated poly-switch takes priority. ■

example 141 We can explore this further by considering the YAML settings in Listing 488; upon running the command

```
cmh:~$ latexindent.pl -m -l=mycom-mlb5.yaml mycommand1.tex
```

we obtain the output given in Listing 487. ■

LISTING 487: mycommand1.tex using Listing 488

```
\mycommand
{
   mand arg text
   mand arg text}
{
   mand arg text
   mand arg text}
```

LISTING 488: mycom-mlb5.yaml -m

```
modifyLineBreaks:
  mandatoryArguments:
    LCuBStartsOnOwnLine: 1
    RCuBFinishesWithLineBreak: -1
```

As previously, the most-recently-processed code block takes priority – as before, the second (i.e, *last*) argument.

Exploring this further, we consider the YAML settings in Listing 490, and run the command

```
cmh:~$ latexindent.pl -m -l=mycom-mlb6.yaml mycommand1.tex
```

which gives the output in Listing 489.

LISTING 489: mycommand1.tex using Listing 490

```
\mycommand
{
   mand arg text
   mand arg text}%
{
   mand arg text
   mand arg text}
```

LISTING 490: mycom-mlb6.yaml -m

```
modifyLineBreaks:
  mandatoryArguments:
    LCuBStartsOnOwnLine: 2
    RCuBFinishesWithLineBreak: -1
```

Note that a % *has* been added to the trailing first }; this is because:

- while processing the *first* argument, the trailing line break has been removed (RCuBFinishesWithLineBreak set to −1);
- while processing the *second* argument, latexindent.pl finds that it does *not* begin on its own line, and so because LCuBStartsOnOwnLine is set to 2, it adds a comment, followed by a line break.

■

4.3.6 Conflicting poly-switches: nested code blocks

example 142 Now let's consider an example when nested code blocks have conflicting poly-switches; we'll use the code in Listing 491, noting that it contains nested environments.

LISTING 491: nested-env.tex

```
\begin{one}
one text
\begin{two}
two text
\end{two}
\end{one}
```

Let's use the YAML settings given in Listing 493, which upon running the command

```
cmh:~$ latexindent.pl -m -l=nested-env-mlb1.yaml nested-env.tex
```

gives the output in Listing 492.

■

LISTING 492: nested-env.tex using Listing 493

```
\begin{one}
   one text
   \begin{two}
      two text\end{two}\end{one}
```

LISTING 493: nested-env-mlb1.yaml -m

```
modifyLineBreaks:
    environments:
        EndStartsOnOwnLine: -1
        EndFinishesWithLineBreak: 1
```

In Listing 492, let's first of all note that both environments have received the appropriate (default) indentation; secondly, note that the poly-switch `EndStartsOnOwnLine` appears to have won the conflict, as `\end{one}` has had its leading line break removed. ■

To understand it, let's talk about how `latexindent.pl` has processed Listing 491; specifically `latexindent.pl`:

1. finds the `environment` named `one`
2. searches `one` for code blocks
   (a) finds the `environment` named `two`
   (b) searches `two` for code blocks, finds none
   (c) does the poly-switch work for `two` instructed by Listing 493: remove the line break before the end statement, and adds a line break *after* the end statement.
3. does the poly-switch work for `one` instructed by Listing 492 which is: remove line breaks before the end statement, and add a line break *after* the end statement;

example 143 We can explore this further using the poly-switches in Listing 495; upon running the command

```
cmh:~$ latexindent.pl -m -l=nested-env-mlb2.yaml nested-env.tex
```

we obtain the output given in Listing 494.

LISTING 494: nested-env.tex using Listing 495

```
\begin{one}
   one text
   \begin{two}
      two text
   \end{two}
\end{one}
```

LISTING 495: nested-env-mlb2.yaml -m

```
modifyLineBreaks:
    environments:
        EndStartsOnOwnLine: 1
        EndFinishesWithLineBreak: -1
```

To understand it, let's talk about how `latexindent.pl` has processed Listing 494; specifically `latexindent.pl`:

1. finds the `environment` named `one`
2. searches `one` for code blocks
   (a) finds the `environment` named `two`
   (b) searches `two` for code blocks, finds none
   (c) does the poly-switch work for `two` instructed by Listing 493: add a line break before the end statement, and remove line breaks *after* the end statement.
3. does the poly-switch work for `one` instructed by Listing 495 which is: add a line break before the end statement, and remove line breaks *after* the end statement. ■

SECTION 5

The -r, -rv and -rr switches

N: 2019-07-13

You can instruct `latexindent.pl` to perform replacements/substitutions on your file by using any of the `-r`, `-rv` or `-rr` switches:

- the `-r` switch will perform indentation and replacements, not respecting verbatim code blocks;
- the `-rv` switch will perform indentation and replacements, and *will* respect verbatim code blocks;
- the `-rr` switch will *not* perform indentation, and will perform replacements not respecting verbatim code blocks.

We will demonstrate each of the `-r`, `-rv` and `-rr` switches, but a summary is given in Table 4.

TABLE 4: The replacement mode switches

switch	indentation?	respect verbatim?
`-r`	✔	✖
`-rv`	✔	✔
`-rr`	✖	✖

The default value of the `replacements` field is shown in Listing 496; as with all of the other fields, you are encouraged to customise and change this as you see fit. The options in this field will *only* be considered if the `-r`, `-rv` or `-rr` switches are active; when discussing YAML settings related to the replacement-mode switches, we will use the style given in Listing 496.

LISTING 496: `replacements` -r

```
419 replacements:
420   - amalgamate: 1
421   - this: latexindent.pl
422     that: pl.latexindent
423     lookForThis: 0
424     when: before
```

The first entry within the `replacements` field is `amalgamate`, and is *optional*; by default it is set to 1, so that replacements will be amalgamated from each settings file that you specify. As you'll see in the demonstrations that follow, there is no need to specify this field.

You'll notice that, by default, there is only *one* entry in the `replacements` field, but it can take as many entries as you would like; each one needs to begin with a `-` on its own line.

5.1 Introduction to replacements

Let's explore the action of the default settings, and then we'll demonstrate the feature with further examples.

example 144 Beginning with the code in Listing 497 and running the command

```
cmh:~$ latexindent.pl -r replace1.tex
```

gives the output given in Listing 498. ■

LISTING 497: replace1.tex

```
Before text, latexindent.pl,
after text.
```

LISTING 498: replace1.tex default

```
Before text, latexindent.pl,
after text.
```

We note that in Listing 496, because lookForThis is set to 0, the specified replacement has *not* been made, and there is no difference between Listings 497 and 498.

If we *do* wish to perform this replacement, then we can tweak the default settings of Listing 496 on the preceding page by changing lookForThis to 1; we perform this action in Listing 500, and run the command

```
cmh:~$ latexindent.pl -r replace1.tex -l=replace1.yaml
```

which gives the output in Listing 499.

LISTING 499: replace1.tex using Listing 500

```
Before text, pl.latexindent,
after text.
```

LISTING 500: replace1.yaml -r

```
replacements:
  -
    amalgamate: 0
  -
    this: latexindent.pl
    that: pl.latexindent
    lookForThis: 1
```

Note that in Listing 500 we have specified amalgamate as 0 so that the default replacements are overwritten. ■

We haven't yet discussed the when field; don't worry, we'll get to it as part of the discussion in what follows.

5.2 The two types of replacements

There are two types of replacements:

1. *string*-based replacements, which replace the string in *this* with the string in *that*. If you specify this and you do not specify that, then the that field will be assumed to be empty.
2. *regex*-based replacements, which use the substitution field.

We will demonstrate both in the examples that follow.

latexindent.pl chooses which type of replacement to make based on which fields have been specified; if the this field is specified, then it will make *string*-based replacements, regardless of if substitution is present or not.

5.3 Examples of replacements

example 145 We begin with code given in Listing 501

LISTING 501: colsep.tex

```
\begin{env}
1 2 3\arraycolsep=3pt
4 5 6\arraycolsep=5pt
\end{env}
```

Let's assume that our goal is to remove both of the arraycolsep statements; we can achieve this in a few different ways.

Using the YAML in Listing 503, and running the command ■

```
cmh:~$ latexindent.pl -r colsep.tex -l=colsep.yaml
```

then we achieve the output in Listing 502.

LISTING 502: colsep.tex using Listing 503

```
\begin{env}
   1 2 3
   4 5 6
\end{env}
```

LISTING 503: colsep.yaml -r

```
replacements:
  -
    this: \arraycolsep=3pt
  -
    this: \arraycolsep=5pt
```

Note that in Listing 503, we have specified *two* separate fields, each with their own '*this*' field; furthermore, for both of the separate fields, we have not specified 'that', so the that field is assumed to be blank by latexindent.pl;

We can make the YAML in Listing 503 more concise by exploring the substitution field. Using the settings in Listing 505 and running the command

```
cmh:~$ latexindent.pl -r colsep.tex -l=colsep1.yaml
```

then we achieve the output in Listing 504.

LISTING 504: colsep.tex using Listing 505

```
\begin{env}
   1 2 3
   4 5 6
\end{env}
```

LISTING 505: colsep1.yaml -r

```
replacements:
  -
    substitution:
    s/\\arraycolsep=\d+pt//sg
```

The code given in Listing 505 is an example of a *regular expression*, which we may abbreviate to *regex* in what follows. This manual is not intended to be a tutorial on regular expressions; you might like to read, for example, [35] for a detailed covering of the topic. With reference to Listing 505, we do note the following:

- the general form of the substitution field is s/regex/replacement/modifiers. You can place any regular expression you like within this;
- we have 'escaped' the backslash by using \\
- we have used \d+ to represent *at least* one digit
- the s *modifier* (in the sg at the end of the line) instructs latexindent.pl to treat your file as one single line;
- the g *modifier* (in the sg at the end of the line) instructs latexindent.pl to make the substitution *globally* throughout your file; you might try removing the g modifier from Listing 505 and observing the difference in output.

You might like to see https://perldoc.perl.org/perlre.html#Modifiers for details of modifiers; in general, I recommend starting with the sg modifiers for this feature. ■

example 146 We'll keep working with the file in Listing 501 on the previous page for this example.

Using the YAML in Listing 507, and running the command

```
cmh:~$ latexindent.pl -r colsep.tex -l=multi-line.yaml
```

then we achieve the output in Listing 506. ■

LISTING 506: colsep.tex using Listing 507

```
multi-line!
```

LISTING 507: multi-line.yaml -r

```
replacements:
  -
    this: |-
     \begin{env}
     1 2 3\arraycolsep=3pt
     4 5 6\arraycolsep=5pt
     \end{env}
    that: 'multi-line!'
```

With reference to Listing 507, we have specified a *multi-line* version of `this` by employing the *literal* YAML style `|-`. See, for example, https://stackoverflow.com/questions/3790454/in-yaml-how-do-i-break-a-string-over-multiple-lines for further options, all of which can be used in your YAML file.

This is a natural point to explore the `when` field, specified in Listing 496 on page 119. This field can take two values: *before* and *after*, which respectively instruct `latexindent.pl` to perform the replacements *before* indentation or *after* it. The default value is `before`.

Using the YAML in Listing 509, and running the command

```
cmh:~$ latexindent.pl -r colsep.tex -l=multi-line1.yaml
```

then we achieve the output in Listing 508.

LISTING 508: colsep.tex using Listing 509

```
\begin{env}
   1 2 3\arraycolsep=3pt
   4 5 6\arraycolsep=5pt
\end{env}
```

LISTING 509: multi-line1.yaml -r

```
replacements:
  -
    this: |-
     \begin{env}
     1 2 3\arraycolsep=3pt
     4 5 6\arraycolsep=5pt
     \end{env}
    that: 'multi-line!'
    when: after
```

We note that, because we have specified `when: after`, that `latexindent.pl` has not found the string specified in Listing 509 within the file in Listing 501 on page 120. As it has looked for the string within Listing 509 *after* the indentation has been performed. After indentation, the string as written in Listing 509 is no longer part of the file, and has therefore not been replaced.

As a final note on this example, if you use the `-rr` switch, as follows,

```
cmh:~$ latexindent.pl -rr colsep.tex -l=multi-line1.yaml
```

then the `when` field is ignored, no indentation is done, and the output is as in Listing 506. ■

example 147 An important part of the substitution routine is in *capture groups*.

Assuming that we start with the code in Listing 510, let's assume that our goal is to replace each occurrence of `$$...$$` with `\begin{equation*}...\end{equation*}`. This example is partly motivated by tex stackexchange question 242150. ■

LISTING 510: displaymath.tex

```
before text $$a^2+b^2=4$$ and $$c^2$$

$$
d^2+e^2 = f^2
$$
and also $$ g^2
$$ and some inline math: $h^2$
```

We use the settings in Listing 512 and run the command

```
cmh:~$ latexindent.pl -r displaymath.tex -l=displaymath1.yaml
```

to receive the output given in Listing 511.

LISTING 511: displaymath.tex using Listing 512

```
before text \begin{equation*}a^2+b^2=4\end{equation*}
    and \begin{equation*}c^2\end{equation*}

\begin{equation*}
   d^2+e^2 = f^2
\end{equation*}
and also \begin{equation*} g^2
\end{equation*} and some inline math: $h^2$
```

LISTING 512: displaymath1.yaml

-r

```
replacements:
  -
    substitution: |-
     s/\$\$
       (.*?)
       \$\$/\\begin{equation*}$1\\end{equation*}/sgx
```

A few notes about Listing 512:

1. we have used the x modifier, which allows us to have white space within the regex;
2. we have used a capture group, `(.*?)` which captures the content between the `$$...$$` into the special variable, `$1`;
3. we have used the content of the capture group, `$1`, in the replacement text.

See https://perldoc.perl.org/perlre.html#Capture-groups for a discussion of capture groups.

The features of the replacement switches can, of course, be combined with others from the toolkit of `latexindent.pl`. For example, we can combine the poly-switches of Section 4.3 on page 99, which we do in Listing 514; upon running the command

```
cmh:~$ latexindent.pl -r -m displaymath.tex -l=displaymath1.yaml,equation.yaml
```

then we receive the output in Listing 513.

LISTING 513: displaymath.tex using Listings 512 and 514

```
before text%
\begin{equation*}%
   a^2+b^2=4%
\end{equation*}%
and%
\begin{equation*}%
   c^2%
\end{equation*}

\begin{equation*}
   d^2+e^2 = f^2
\end{equation*}
and also%
\begin{equation*}%
   g^2
\end{equation*}%
and some inline math: $h^2$
```

LISTING 514: equation.yaml -m

```
modifyLineBreaks:
    environments:
        equation*:
            BeginStartsOnOwnLine: 2
            BodyStartsOnOwnLine: 2
            EndStartsOnOwnLine: 2
            EndFinishesWithLineBreak: 2
```

■

example 148 This example is motivated by tex stackexchange question 490086. We begin with the code in Listing 515.

LISTING 515: phrase.tex

```
phrase 1       phrase 2 phrase 3            phrase 100

phrase 1       phrase 2 phrase 3            phrase 100

phrase 1       phrase 2 phrase 3            phrase 100

phrase 1       phrase 2 phrase 3            phrase 100
```

Our goal is to make the spacing uniform between the phrases. To achieve this, we employ the settings in Listing 517, and run the command

```
cmh:~$ latexindent.pl -r phrase.tex -l=hspace.yaml
```

which gives the output in Listing 516.

LISTING 516: phrase.tex using Listing 517

```
phrase 1 phrase 2 phrase 3 phrase 100

phrase 1 phrase 2 phrase 3 phrase 100

phrase 1 phrase 2 phrase 3 phrase 100

phrase 1 phrase 2 phrase 3 phrase 100
```

LISTING 517: hspace.yaml -r

```
replacements:
  -
    substitution: s/\h+/ /sg
```

The \h+ setting in Listing 517 say to replace *at least one horizontal space* with a single space. ■

example 149 We begin with the code in Listing 518. ■

LISTING 518: references.tex

```
equation \eqref{eq:aa} and Figure  \ref{fig:bb}
and table~\ref{tab:cc}
```

Our goal is to change each reference so that both the text and the reference are contained within one hyperlink. We achieve this by employing Listing 520 and running the command

```
cmh:~$ latexindent.pl -r references.tex -l=reference.yaml
```

which gives the output in Listing 519.

LISTING 519: references.tex using Listing 520

```
\hyperref{equation \ref*{eq:aa}} and \hyperref{Figure \ref*{fig:bb}}
and \hyperref{table \ref*{tab:cc}}
```

LISTING 520: reference.yaml -r

```
replacements:
  -
    substitution: |-
      s/(
        equation
        |
        table
        |
        figure
        |
        section
      )
      (\h|~)*
      \\(?:eq)?
      ref\{(.*?)\}/\\hyperref{$1 \\ref\*{$3}}/sgxi
```

Referencing Listing 520, the | means *or*, we have used *capture groups*, together with an example of an *optional* pattern, `(?:eq)?`. ■

example 150 Let's explore the three replacement mode switches (see Table 4 on page 119) in the context of an example that contains a verbatim code block, Listing 521; we will use the settings in Listing 522.

LISTING 521: verb1.tex

```
\begin{myenv}
body of verbatim
\end{myenv}
some verbatim
\begin{verbatim}
    body
        of
      verbatim
 text
\end{verbatim}
text
```

LISTING 522: verbatim1.yaml -r

```
replacements:
  -
    this: 'body'
    that: 'head'
```

Upon running the following commands,

```
cmh:~$ latexindent.pl -r verb1.tex -l=verbatim1.yaml -o=+mod1
cmh:~$ latexindent.pl -rv verb1.tex -l=verbatim1.yaml -o=+-rv-mod1
cmh:~$ latexindent.pl -rr verb1.tex -l=verbatim1.yaml -o=+-rr-mod1
```

■

we receive the respective output in Listings 523 to 525

LISTING 523: verb1-mod1.tex

```
\begin{myenv}
   head of verbatim
\end{myenv}
some verbatim
\begin{verbatim}
    head
        of
      verbatim
 text
\end{verbatim}
text
```

LISTING 524: verb1-rv-mod1.tex

```
\begin{myenv}
   head of verbatim
\end{myenv}
some verbatim
\begin{verbatim}
    body
        of
      verbatim
 text
\end{verbatim}
text
```

LISTING 525: verb1-rr-mod1.tex

```
\begin{myenv}
head of verbatim
\end{myenv}
some verbatim
\begin{verbatim}
    head
        of
      verbatim
 text
\end{verbatim}
text
```

We note that:

1. in Listing 523 indentation has been performed, and that the replacements specified in Listing 522 have been performed, even within the verbatim code block;
2. in Listing 524 indentation has been performed, but that the replacements have *not* been performed within the verbatim environment, because the `rv` switch is active;
3. in Listing 525 indentation has *not* been performed, but that replacements have been performed, not respecting the verbatim code block.

See the summary within Table 4 on page 119.

example 151 Let's explore the `amalgamate` field from Listing 496 on page 119 in the context of the file specified in Listing 526.

LISTING 526: amalg1.tex

```
one two three
```

Let's consider the YAML files given in Listings 527 to 529.

LISTING 527: amalg1-yaml.yaml -r

```
replacements:
  -
    this: one
    that: 1
```

LISTING 528: amalg2-yaml.yaml -r

```
replacements:
  -
    this: two
    that: 2
```

LISTING 529: amalg3-yaml.yaml -r

```
replacements:
  -
    amalgamate: 0
  -
    this: three
    that: 3
```

Upon running the following commands,

```
cmh:~$ latexindent.pl -r amalg1.tex -l=amalg1-yaml
cmh:~$ latexindent.pl -r amalg1.tex -l=amalg1-yaml,amalg2-yaml
cmh:~$ latexindent.pl -r amalg1.tex -l=amalg1-yaml,amalg2-yaml,amalg3-yaml
```

we receive the respective output in Listings 530 to 532.

LISTING 530: amalg1.tex using Listing 527

```
1 two three
```

LISTING 531: amalg1.tex using Listings 527 and 528

```
1 2 three
```

LISTING 532: amalg1.tex using Listings 527 to 529

```
one two 3
```

We note that:

1. in Listing 530 the replacements from Listing 527 have been used;
2. in Listing 531 the replacements from Listings 527 and 528 have *both* been used, because

the default value of `amalgamate` is 1;

3. in Listing 532 *only* the replacements from Listing 529 have been used, because the value of `amalgamate` has been set to 0.

■

SECTION 6

The –lines switch

N: 2021-09-16

`latexindent.pl` can operate on a *selection* of lines of the file using the `-lines` or `-n` switch.

The basic syntax is `-lines MIN-MAX`, so for example

```
cmh:~$ latexindent.pl --lines 3-7 myfile.tex
cmh:~$ latexindent.pl -n 3-7 myfile.tex
```

will only operate upon lines 3 to 7 in `myfile.tex`. All of the other lines will *not* be operated upon by `latexindent.pl`.

The options for the `lines` switch are:

- line range, as in `-lines 3-7`
- single line, as in `-lines 5`
- multiple line ranges separated by commas, as in `-lines 3-5,8-10`
- negated line ranges, as in `-lines !3-5` which translates to `-lines 1-2,6-N`, where N is the number of lines in your file.

We demonstrate this feature, and the available variations in what follows. We will use the file in Listing 533.

LISTING 533: `myfile.tex`

```
1  Before the environments
2  \begin{one}
3     first block, first line
4     first block, second line
5     first block, third line
6     \begin{two}
7        second block, first line
8        second block, second line
9        second block, third line
10       second block, fourth line
11    \end{two}
12 \end{one}
```

example 152 We demonstrate the basic usage using the command

```
cmh:~$ latexindent.pl --lines 3-7 myfile.tex -o=+-mod1
```

which instructs `latexindent.pl` to only operate on lines 3 to 7; the output is given in Listing 534. ■

LISTING 534: myfile-mod1.tex

```
Before the environments
\begin{one}
first block, first line
first block, second line
first block, third line
\begin{two}
second block, first line
      second block, second line
      second block, third line
      second block, fourth line
   \end{two}
\end{one}
```

The following two calls to `latexindent.pl` are equivalent

```
cmh:~$ latexindent.pl --lines 3-7 myfile.tex -o=+-mod1
cmh:~$ latexindent.pl --lines 7-3 myfile.tex -o=+-mod1
```

as `latexindent.pl` performs a check to put the lowest number first. ■

example 153 You can call the `lines` switch with only *one number* and in which case only that line will be operated upon. For example

```
cmh:~$ latexindent.pl --lines 5 myfile.tex -o=+-mod2
```

instructs `latexindent.pl` to only operate on line 5; the output is given in Listing 535.

LISTING 535: myfile-mod2.tex

```
Before the environments
\begin{one}
   first block, first line
   first block, second line
first block, third line
   \begin{two}
      second block, first line
      second block, second line
      second block, third line
      second block, fourth line
   \end{two}
\end{one}
```

The following two calls are equivalent:

```
cmh:~$ latexindent.pl --lines 5 myfile.tex
cmh:~$ latexindent.pl --lines 5-5 myfile.tex
```

■

example 154 If you specify a value outside of the line range of the file then `latexindent.pl` will ignore the `lines` argument, detail as such in the log file, and proceed to operate on the entire file.

For example, in the following call

```
cmh:~$ latexindent.pl --lines 11-13 myfile.tex
```

`latexindent.pl` will ignore the `lines` argument, and *operate on the entire file* because List- ■

ing 533 only has 12 lines.

Similarly, in the call

```
cmh:~$ latexindent.pl --lines -1-3 myfile.tex
```

`latexindent.pl` will ignore the `lines` argument, and *operate on the entire file* because we assume that negatively numbered lines in a file do not exist. ■

example 155 You can specify *multiple line ranges* as in the following

```
cmh:~$ latexindent.pl --lines 3-5,8-10 myfile.tex -o=+-mod3
```

which instructs `latexindent.pl` to operate upon lines 3 to 5 and lines 8 to 10; the output is given in Listing 536.

LISTING 536: `myfile-mod3.tex`

```
1  Before the environments
2  \begin{one}
3  first block, first line
4  first block, second line
5  first block, third line
6     \begin{two}
7        second block, first line
8  second block, second line
9  second block, third line
10 second block, fourth line
11    \end{two}
12 \end{one}
```

The following calls to `latexindent.pl` are all equivalent

```
cmh:~$ latexindent.pl --lines 3-5,8-10 myfile.tex
cmh:~$ latexindent.pl --lines 8-10,3-5 myfile.tex
cmh:~$ latexindent.pl --lines 10-8,3-5 myfile.tex
cmh:~$ latexindent.pl --lines 10-8,5-3 myfile.tex
```

as `latexindent.pl` performs a check to put the lowest line ranges first, and within each line range, it puts the lowest number first. ■

example 156 There's no limit to the number of line ranges that you can specify, they just need to be separated by commas. For example

```
cmh:~$ latexindent.pl --lines 1-2,4-5,9-10,12 myfile.tex -o=+-mod4
```

has four line ranges: lines 1 to 2, lines 4 to 5, lines 9 to 10 and line 12. The output is given in Listing 537. ■

LISTING 537: myfile-mod4.tex

```
Before the environments
\begin{one}
   first block, first line
   first block, second line
   first block, third line
   \begin{two}
      second block, first line
      second block, second line
   second block, third line
   second block, fourth line
   \end{two}
\end{one}
```

As previously, the ordering does not matter, and the following calls to `latexindent.pl` are all equivalent

```
cmh:~$ latexindent.pl --lines 1-2,4-5,9-10,12 myfile.tex
cmh:~$ latexindent.pl --lines 2-1,4-5,9-10,12 myfile.tex
cmh:~$ latexindent.pl --lines 4-5,1-2,9-10,12 myfile.tex
cmh:~$ latexindent.pl --lines 12,4-5,1-2,9-10 myfile.tex
```

as `latexindent.pl` performs a check to put the lowest line ranges first, and within each line range, it puts the lowest number first. ■

example 157 You can specify *negated line ranges* by using `!` as in

```
cmh:~$ latexindent.pl --lines !5-7 myfile.tex -o=+-mod5
```

which instructs `latexindent.pl` to operate upon all of the lines *except* lines 5 to 7.

In other words, `latexindent.pl` *will* operate on lines 1 to 4, and 8 to 12, so the following two calls are equivalent:

```
cmh:~$ latexindent.pl --lines !5-7 myfile.tex
cmh:~$ latexindent.pl --lines 1-4,8-12 myfile.tex
```

The output is given in Listing 538.

LISTING 538: myfile-mod5.tex

```
Before the environments
\begin{one}
first block, first line
first block, second line
   first block, third line
   \begin{two}
      second block, first line
second block, second line
second block, third line
second block, fourth line
\end{two}
\end{one}
```

■

example 158 You can specify *multiple negated line ranges* such as ■

```
cmh:~$ latexindent.pl --lines !5-7,!9-10 myfile.tex -o=+-mod6
```

which is equivalent to:

```
cmh:~$ latexindent.pl --lines 1-4,8,11-12 myfile.tex -o=+-mod6
```

The output is given in Listing 539.

LISTING 539: myfile-mod6.tex

```
1  Before the environments
2  \begin{one}
3  first block, first line
4  first block, second line
5     first block, third line
6     \begin{two}
7        second block, first line
8  second block, second line
9        second block, third line
10       second block, fourth line
11 \end{two}
12 \end{one}
```

example 159 If you specify a line range with anything other than an integer, then `latexindent.pl` will ignore the `lines` argument, and *operate on the entire file*.

Sample calls that result in the `lines` argument being ignored include the following:

```
cmh:~$ latexindent.pl --lines 1-x myfile.tex
cmh:~$ latexindent.pl --lines !y-3 myfile.tex
```

example 160 We can, of course, use the `lines` switch in combination with other switches.

For example, let's use with the file in Listing 540.

LISTING 540: myfile1.tex

```
1  Before the environments
2  \begin{one}
3     first block, first line
4     first block, second line
5     first block, third line
6     \begin{two} body \end{two}
7  \end{one}
```

We can demonstrate interaction with the `-m` switch (see Section 4 on page 70); in particular, if we use Listing 429 on page 105, Listing 413 on page 103 and Listing 414 on page 103 and run

```
cmh:~$ latexindent.pl --lines 6 myfile1.tex -o=+-mod1 -m -l env-mlb2,env-mlb7,env-mlb8 -o=+-mod1
```

then we receive the output in Listing 541.

LISTING 541: myfile1-mod1.tex

```
Before the environments
\begin{one}
   first block, first line
   first block, second line
   first block, third line
\begin{two}
   body
\end{two}
\end{one}
```

■

SECTION 7

Fine tuning

N: 2019-07-13

`latexindent.pl` operates by looking for the code blocks detailed in Table 2 on page 49. The fine tuning of the details of such code blocks is controlled by the `fineTuning` field, detailed in Listing 542.

This field is for those that would like to peek under the bonnet/hood and make some fine tuning to `latexindent.pl`'s operating.

Warning!

Making changes to the fine tuning may have significant consequences for your indentation scheme, proceed with caution!

LISTING 542: `fineTuning`

```
fineTuning:
  environments:
    name: [a-zA-Z@\*0-9_\\]+
  items:
    itemRegEx: \\(?:item|myitem)
  ifElseFi:
    name: (?!@?if[a-zA-Z@]*?\{)@?if[a-zA-Z@]*
    elseOrRegEx: \\(?:else|or)
  commands:
    name: [+a-zA-Z@*0-9_:]+
  namedGroupingBracesBrackets:
    name: [0-9\.a-zA-Z@\*><]+
  keyEqualsValuesBracesBrackets:
    name: [a-zA-Z@\*0-9_\/.:\#-]+[a-zA-Z@\*0-9_\/.\h:\#-]*
  arguments:
    between: |-
      (?x)
      (?:
        [_^*#0-9(),+-]
        |
        (?:node|at|to|decoration)
        |
        (?:\([^)]*\))
      )
  trailingComments:
    notPrecededBy: (?<!\\)
    afterComment: .*?
  lookForAlignDelimsDefaults:
    delims: 1
    alignDoubleBackSlash: 1
    spacesBeforeDoubleBackSlash: 1
    multiColumnGrouping: 0
    alignRowsWithoutMaxDelims: 1
    spacesBeforeAmpersand: 1
    spacesAfterAmpersand: 1
    justification: left
    alignFinalDoubleBackSlash: 0
    dontMeasure: 0
```

```
465     delimiterRegEx: (?<!\\)(&)
466     delimiterJustification: left
467     lookForChildCodeBlocks: 1
468     alignContentAfterDoubleBackSlash: 0
469     spacesAfterDoubleBackSlash: 1
470     doubleBackSlash: \\\\(?:\h*\[\h*[0-9.]+\h*[a-zA-Z]+\h*\])?
471   modifyLineBreaks:
472     comma: ','
473     betterFullStop: |-
474       (?x)                                # ignore spaces in the below
475       (?:                                 #
476         \.\)                              # .)
477         (?!\h*[a-z])                      # not *followed by* a-z
478       )                                   #
479       |                                   # OR
480       (?:                                 #
481         (?<!                              # not *preceded by*
482           (?:                             #
483             (?:[eE]\.[gG])                # e.g OR E.g OR e.G OR E.G
484             |                             #
485             (?:[iI]\.[eE])                # i.e OR I.e OR i.E OR I.E
486             |                             #
487             (?:etc)                       # etc
488             |                             #
489             (?:[wW]\.[rR]\.[tT])          # w.r.t OR W.r.t OR w.R.t OR w.r.T OR W.R.t OR W.r.T
    OR w.R.T OR W.R.T
490           )                               #
491         )                                 #
492       )                                   #
493       \.                                  # .
494       (?!                                 # not *followed by*
495         (?:                               #
496           [a-zA-Z0-9-~,]                  #
497           |                               #
498           \),                             # ),
499           |                               #
500           \)\.                            # ).
501         )                                 #
502       )                                   #
```

The fields given in Listing 542 are all *regular expressions*. This manual is not intended to be a tutorial on regular expressions; you might like to read, for example, [35] for a detailed covering of the topic.

We make the following comments with reference to Listing 542:

1. the `environments:name` field details that the *name* of an environment can contain:
   (a) `a-z` lower case letters
   (b) `A-Z` upper case letters
   (c) `@` the `@` 'letter'
   (d) `\*` stars
   (e) `0-9` numbers
   (f) `_` underscores
   (g) `\` backslashes

   The `+` at the end means *at least one* of the above characters.
2. the `ifElseFi:name` field:
   (a) `@?` means that it *can possibly* begin with `@`

(b) followed by `if`

(c) followed by 0 or more characters from `a-z`, `A-Z` and `@`

(d) the `?` the end means *non-greedy*, which means 'stop the match as soon as possible'

3. the `modifyLineBreaks` field refers to fine tuning settings detailed in Section 4 on page 70. In particular:

   (a) `betterFullStop` is in relation to the one sentence per line routine, detailed in Section 4.2 on page 86

   (b) `comma` is in relation to the `CommaStartsOnOwnLine` and `CommaFinishesWithLineBreak` polyswitches surrounding commas in optional and mandatory arguments; see Table 3 on page 115

Warning!

For the `fineTuning` feature you should usually use *non*-capturing groups, such as `(?:...)` and *not* capturing groups, which are `(...)`

7.1 fineTuning trailing comments

example 161 We can tweak the `fineTuning` for how trailing comments are classified. For motivation, let's consider the code given in Listing 543

LISTING 543: `finetuning4.tex`

```
some before text
 \href{Handbook%20for%30Spoken%40document.pdf}{my document}
some after text
```

We will compare the settings given in Listings 544 and 545.

LISTING 544: `href1.yaml` `-m`

```
modifyLineBreaks:
    textWrapOptions:
        columns: -1
        blocksEndBefore:
           verbatim: 0
        blocksFollow:
           verbatim: 0

removeTrailingWhitespace:
    beforeProcessing: 1
```

LISTING 545: `href2.yaml` `-m`

```
fineTuning:
    trailingComments:
      notPrecededBy:
    '(?:(?<!Handbook)(?<!for)(?<!Spoken))'

modifyLineBreaks:
    textWrapOptions:
        columns: -1
        blocksEndBefore:
           verbatim: 0
        blocksFollow:
           verbatim: 0

removeTrailingWhitespace:
    beforeProcessing: 1
```

Upon running the following commands

```
cmh:~$ latexindent.pl -m finetuning4.tex -o=+-mod1 -l=href1
cmh:~$ latexindent.pl -m finetuning4.tex -o=+-mod2 -l=href2
```

we receive the respective output in Listings 546 and 547.

LISTING 546: `finetuning4.tex` using Listing 544

```
some before text \href{Handbooksome after text%20for%30Spoken%40document.pdf}{my document}
```

■

LISTING 547: finetuning4.tex using Listing 545

```
some before text \href{Handbook%20for%30Spoken%40document.pdf}{my document} some after text
```

We note that in:

- Listing 546 the trailing comments are assumed to be everything following the first comment symbol, which has meant that everything following it has been moved to the end of the line; this is undesirable, clearly!
- Listing 547 has fine-tuned the trailing comment matching, and says that % cannot be immediately preceded by the words 'Handbook', 'for' or 'Spoken', which means that none of the % symbols have been treated as trailing comments, and the output is desirable.

example 162 Another approach to this situation, which does not use `fineTuning`, is to use `noIndentBlock` which we discussed in Listing 33 on page 23; using the settings in Listing 548 and running the command

```
cmh:~$ latexindent.pl -m finetuning4.tex -o=+-mod3 -l=href3
```

then we receive the same output given in Listing 547.

LISTING 548: href3.yaml -m

```
modifyLineBreaks:
    textWrapOptions:
        columns: -1
        blocksEndBefore:
           verbatim: 0
        blocksFollow:
           verbatim: 0
noIndentBlock:
    href:
        begin: \\href\{[^}]*?\}\{
        body: [^}]*?
        end: \}
```

With reference to the `body` field in Listing 548, we note that the `body` field can be interpreted as: the fewest number of zero or more characters that are not right braces. This is an example of character class.

N: 2023-06-01 **ample 163** We can use the `fineTuning` settings to tweak how `latexindent.pl` finds trailing comments.

We begin with the file in Listing 549

LISTING 549: finetuning5.tex

```
\chapter{chapter text} % 123
chapter text
\section{section text} % 456
section text
% end
% end
```

Using the settings in Listing 551 and running the command

```
cmh:~$ latexindent.pl finetuning5.tex -l=fine-tuning3.yaml
```

gives the output in Listing 550.

LISTING 550: finetuning5-mod1.tex

```
\chapter{chapter text} % 123
   chapter text
   \section{section text} % 456
      section text
   % end
% end
```

LISTING 551: fine-tuning3.yaml

```
fineTuning:
  trailingComments:
    notPrecededBy: (?<!\\)
    afterComment: (?!(?:\hend)).*?
  commands:
    name: |-
      (?x)
       (?!(?:begin|end|chapter|section))
       [+a-zA-Z@*0-9_:]+

specialBeginEnd:
 - name: customHeading
   begin: \\(?:chapter|section)
   end: \%\h+end
   nested: 1
```

The settings in Listing 551 detail that trailing comments can *not* be followed by a single space, and then the text 'end'. This means that the `specialBeginEnd` routine will be able to find the pattern `% end` as the `end` part. The trailing comments `123` and `456` are still treated as trailing comments. ■

7.2 fineTuning environments

example 164 We can use the `fineTuning` settings to tweak how `latexindent.pl` finds environments.

N: 2023-10-13

We begin with the file in Listing 552.

LISTING 552: finetuning6.tex

```
\begin{myenv}\label{mylabel}The body of my environment...\end{myenv}
```

Using the settings in Listing 554 and running the command

```
cmh:~$ latexindent.pl finetuning6.tex -m -l=fine-tuning4.yaml
```

gives the output in Listing 553.

LISTING 553: finetuning6-mod1.tex

```
\begin{myenv}\label{mylabel}
   The body of my environment...
\end{myenv}
```

LISTING 554: fine-tuning4.yaml -m

```
modifyLineBreaks:
   environments:
       BodyStartsOnOwnLine: 1
       EndStartsOnOwnLine: 1

fineTuning:
  environments:
    begin: |-
      (?x)                        # environment
        \\begin\{                 # regex
          (?<ENVNAME>             # NON default
           [a-zA-Z@\*0-9_\\]+     #
          )                       #
        \}\s*\\label\{[^}]+?\}    #
    end: \\end\{\g{ENVNAME}\}     # note the capture group name
```

By using the settings in Listing 554 it means that the default poly-switch location of `BodyStartsOnOwnLine` for environments (denoted ♡ in Table 3) has been overwritten so that it is *after* the `label` command.

Referencing Listing 554, unless both `begin` and `end` are specified, then the default value of `name` ■

will be used. ■

7.3 fineTuning itemsRegex

In Listing 121 on page 39 we demonstrated the default indentation after items. You can customise the `itemsRegex` in the `fineTuning`, which we demonstrate next.

example 165 We begin with the code in Listing 555 and use the settings in Listing 556. Upon running

```
cmh:~$ latexindent.pl -l ft-itemsRegex items2-o=+-mod1
```

we receive the output in Listing 557.

LISTING 555: items2.tex

```
\begin{myenv}
\cmh custom item
and after text
\cmh custom item
and after text
\end{myenv}
```

LISTING 556: ft-itemsRegex.yaml

```
indentAfterItems:
  myenv: 1
fineTuning:
  items:
    itemRegEx: \\(?:item|cmh)
indentRules:
  cmh: "     "
```

LISTING 557: items2-mod1.tex

```
\begin{myenv}
   \cmh custom item
        and after text
   \cmh custom item
        and after text
\end{myenv}
```

■

7.4 fineTuning arguments

The strings allowed between arguments for `commands`, `namedGroupingBracesBrackets`, `keyEqualsValuesBracesB` and `UnNamedGroupingBracesBrackets` are controlled by `fineTuning`. We demonstrate with an example next.

N: 2026-03-15

example 166 We begin with the code in Listing 558 and use the settings in Listing 559. Upon running

```
cmh:~$ latexindent.pl -l ft-args beamer1-o=+-mod1
```

we receive the output in Listing 560.

LISTING 558: beamer1.tex

```
\cmh<+->{
first arg}
{
second arg}
```

LISTING 559: ft-args.yaml

```
fineTuning:
  arguments:
    between: |-
      (?x)
      (?:
        [_^*#0-9(),+-]
        |
        (?:node|at|to|decoration)
        |
        (?:\([^)]*\))
        |             # NEW BIT
        (?:<[^>]+>)   # NEW BIT
      )
```

LISTING 560: beamer1-mod1.tex

```
\cmh<+->{
   first arg}
{
   second arg}
```

■

7.5 fineTuning ifElseFi

We can fine tune the `elseOrRegEx` for `ifElseFi` blocks as we demonstrate next.

example 167 We begin with the code in Listing 561 and use the settings in Listing 562. Upon running

```
cmh:~$ latexindent.pl -l ft-ifelsefi ft-ifelsefi-o=+-mod1
```

■

we receive the output in Listing 563.

LISTING 561: ft-ifelsefi.tex

```
\if
if body
\cmh
other body
\fi
```

LISTING 562: ft-ifelsefi.yaml

```
fineTuning:
  ifElseFi:
    elseOrRegEx: \\(?:else|or|cmh)
```

LISTING 563: ft-ifelsefi-mod1.tex

```
\if
   if body
\cmh
   other body
\fi
```

7.6 fineTuning lookForAlignDelims

As detailed in Section 3.5 on page 26 there is a basic and an advanced way to specify the entries in `lookForAlignDelims`; any entries not specified explicitly, will be read from the default values specified in `fineTuning`.

As an example, the settings in Listings 564 to 566 are equivalent.

LISTING 564: align1.yaml

```
lookForAlignDelims:
  align: 1
```

LISTING 565: align2.yaml

```
lookForAlignDelims:
  align:
    spacesBeforeDoubleBackSlash: 1
```

LISTING 566: align3.yaml

```
lookForAlignDelims:
  align: 1

fineTuning:
  lookForAlignDelimsDefaults:
    alignDoubleBackSlash: 1
```

7.7 fineTuning using -y switch

example 168 You can tweak the `fineTuning` using the `-y` switch, but to be sure to use quotes appropriately. For example, starting with the code in Listing 567 and running the following command

```
cmh:~$ latexindent.pl -m
    -y='modifyLineBreaks:oneSentencePerLine:manipulateSentences:␣1,␣
    modifyLineBreaks:oneSentencePerLine:sentencesBeginWith:a-z:␣1,␣
    fineTuning:modifyLineBreaks:betterFullStop:␣
    "(?:\.|;|:(?![a-z]))|(?:(?<!(?:(?:e\.g)|(?:i\.e)|(?:etc))))\.(?!(?:[a-z]|[A-Z]|\
    issue-243.tex -o=+-mod1
```

gives the output shown in Listing 568.

LISTING 567: finetuning3.tex

```
We go; you see: this sentence \cite{tex:stackexchange} finishes here.
```

LISTING 568: finetuning3.tex using -y switch

```
We go;
you see:
this sentence \cite{tex:stackexchange} finishes here.
```

SECTION 8

Conclusions and known limitations

There are a number of known limitations of the script, and almost certainly quite a few that are *unknown*! The known issues include:

multicolumn alignment when working with code blocks in which multicolumn commands overlap, the algorithm can fail; see Listing 60 on page 29.

textWrap after when operating with `indentRules` (see Section 3.9 on page 50) may not always cooperate with one another; if you have a specific example that does not work, please report it to [36].

U: 2019-07-13

You can run `latexindent` on any file; if you don't specify an extension, then the extensions that you specify in `fileExtensionPreference` (see Listing 26 on page 21) will be consulted. If you find a case in which the script struggles, please feel free to report it at [36], and in the meantime, consider using a `noIndentBlock` (see page 23).

I hope that this script is useful to some; if you find an example where the script does not behave as you think it should, the best way to contact me is to report an issue on [36]; otherwise, feel free to find me on the `http://tex.stackexchange.com/users/6621/cmhughes`.

SECTION 9

References

9.1 perl-related links

[32] *CPAN: Comprehensive Perl Archive Network*. URL: http://www.cpan.org/ (visited on 01/23/2017).

[33] *Data Dumper demonstration*. URL: https://stackoverflow.com/questions/7466825/how-do-you-sort-the-output-of-datadumper (visited on 06/18/2021).

[34] *Data::Dumper module*. URL: https://perldoc.perl.org/Data::Dumper (visited on 06/18/2021).

[35] Jeffrey E. F. Friedl. *Mastering Regular Expressions*. ISBN: 0596002890.

[41] *Log4perl Perl module*. URL: http://search.cpan.org/~mschilli/Log-Log4perl-1.49/lib/Log/Log4perl.pm (visited on 09/24/2017).

[42] *Perlbrew*. URL: http://perlbrew.pl/ (visited on 01/23/2017).

[43] *perldoc Encode::Supported*. URL: https://perldoc.perl.org/Encode::Supported (visited on 05/06/2021).

[46] *Strawberry Perl*. URL: http://strawberryperl.com/ (visited on 01/23/2017).

[47] *Text::Tabs Perl module*. URL: http://search.cpan.org/~muir/Text-Tabs+Wrap-2013.0523/lib.old/Text/Tabs.pm (visited on 07/06/2017).

[48] *Text::Wrap Perl module*. URL: http://perldoc.perl.org/Text/Wrap.html (visited on 05/01/2017).

9.2 conda-related links

[30] *anacoda*. URL: https://www.anaconda.com/products/individual (visited on 12/22/2021).

[31] *conda forge*. URL: https://github.com/conda-forge/miniforge (visited on 12/22/2021).

[38] *How to install Anaconda on Ubuntu?* URL: https://askubuntu.com/questions/505919/how-to-install-anaconda-on-ubuntu (visited on 01/21/2022).

[45] *Solving environment: failed with initial frozen solve. Retrying with flexible solve.* URL: https://github.com/conda/conda/issues/9367#issuecomment-558863143 (visited on 01/21/2022).

9.3 VScode-related links

[37] *How to create your own auto-completion for JSON and YAML files on VS Code with the help of JSON Schema*. URL: https://dev.to/brpaz/how-to-create-your-own-auto-completion-for-json-and-yaml-files-on-vs-code-with-the-help-of-json-schema-k1i (visited on 01/01/2022).

[50] *VSCode YAML extension*. URL: https://marketplace.visualstudio.com/items?itemName=redhat.vscode-yaml (visited on 01/01/2022).

9.4 Other links

[29] *A Perl script for indenting tex files*. URL: http://tex.blogoverflow.com/2012/08/a-perl-script-for-indenting-tex-files/ (visited on 01/23/2017).

[36] *Home of latexindent.pl*. URL: https://github.com/cmhughes/latexindent.pl (visited on 01/23/2017).

[39] *How to use latexindent on Windows?* URL: https://tex.stackexchange.com/questions/577250/how-to-use-latexindent-on-windows (visited on 01/08/2022).

[40] *latexindent.pl ghcr (GitHub Container Repository) location*. URL: https://github.com/cmhughes?tab=packages (visited on 06/12/2022).

[44] *pre-commit: A framework for managing and maintaining multi-language pre-commit hooks.* URL: https://pre-commit.com/ (visited on 01/08/2022).

[49] *Video demonstration of latexindent.pl on youtube*. URL: https://www.youtube.com/watch?v=wo38aaH2F4E&spfreload=10 (visited on 02/21/2017).
[51] *Windows line breaks on Linux prevent removal of white space from end of line*. URL: https://github.com/cmhughes/latexindent.pl/issues/256 (visited on 06/18/2021).

9.5 Contributors (in chronological order)

[1] Paulo Cereda. *arara rule, indent.yaml*. May 23, 2013. URL: https://github.com/islandoftex/arara/blob/master/rules/arara-rule-indent.yaml (visited on 03/19/2021).
[2] Harish Kumar. *Early version testing*. Nov. 10, 2013. URL: https://github.com/harishkumarholla (visited on 06/30/2017).
[3] Michel Voßkuhle. *Remove trailing white space*. Nov. 10, 2013. URL: https://github.com/cmhughes/latexindent.pl/pull/12 (visited on 01/23/2017).
[4] Jacobo Diaz. *Changed shebang to make the script more portable*. July 23, 2014. URL: https://github.com/cmhughes/latexindent.pl/pull/17 (visited on 01/23/2017).
[5] Jacobo Diaz. *Hiddenconfig*. July 21, 2014. URL: https://github.com/cmhughes/latexindent.pl/pull/18 (visited on 01/23/2017).
[6] Jason Juang. *add in PATH installation*. Nov. 24, 2015. URL: https://github.com/cmhughes/latexindent.pl/pull/38 (visited on 01/23/2017).
[7] mlep. *One sentence per line*. Aug. 16, 2017. URL: https://github.com/cmhughes/latexindent.pl/issues/81 (visited on 01/08/2018).
[8] John Owens. *Paragraph line break routine removal*. May 27, 2017. URL: https://github.com/cmhughes/latexindent.pl/issues/33 (visited on 05/27/2017).
[9] Cheng Xu (xu cheng). *always output log/help text to STDERR*. July 13, 2018. URL: https://github.com/cmhughes/latexindent.pl/pull/121 (visited on 08/05/2018).
[10] Tom Zöhner (zoehneto). *Improving text wrap*. Feb. 4, 2018. URL: https://github.com/cmhughes/latexindent.pl/issues/103 (visited on 08/04/2018).
[11] Sam Abey. *Print usage tip to STDERR only if STDIN is TTY*. Sept. 17, 2019. URL: https://github.com/cmhughes/latexindent.pl/pull/174 (visited on 03/19/2021).
[12] Randolf J. *Alpine-linux instructions*. Aug. 10, 2020. URL: https://github.com/cmhughes/latexindent.pl/pull/214 (visited on 08/10/2020).
[13] jeanlego. *Search localSettings in CWD as well*. Sept. 15, 2020. URL: https://github.com/cmhughes/latexindent.pl/pull/221 (visited on 03/19/2021).
[14] newptcai. *Update appendices.tex*. Feb. 16, 2021. URL: https://github.com/cmhughes/latexindent.pl/pull/221 (visited on 03/19/2021).
[15] qiancy98. *Locale encoding of file system*. May 6, 2021. URL: https://github.com/cmhughes/latexindent.pl/pull/273 (visited on 05/06/2021).
[16] Alexander Regueiro. *Update help screen*. Mar. 18, 2021. URL: https://github.com/cmhughes/latexindent.pl/pull/261 (visited on 03/19/2021).
[17] XuehaiPan. *-y switch upgrade*. Nov. 12, 2021. URL: https://github.com/cmhughes/latexindent.pl/pull/297 (visited on 11/12/2021).
[18] XuehaiPan. *Verbatim block upgrade*. Oct. 3, 2021. URL: https://github.com/cmhughes/latexindent.pl/pull/290 (visited on 10/03/2021).
[19] eggplants. *Add Dockerfile and its updater/releaser*. June 12, 2022. URL: https://github.com/cmhughes/latexindent.pl/pull/370 (visited on 06/12/2022).
[20] Tom de Geus. *Adding Perl installation + pre-commit hook*. Jan. 21, 2022. URL: https://github.com/cmhughes/latexindent.pl/pull/322 (visited on 01/21/2022).
[21] Jan Holthuis. *Fix pre-commit usage*. Mar. 31, 2022. URL: https://github.com/cmhughes/latexindent.pl/pull/354 (visited on 04/02/2022).
[22] Nehctargl. *Added support for the XDG specification*. Dec. 23, 2022. URL: https://github.com/cmhughes/latexindent.pl/pull/397 (visited on 12/23/2022).
[23] Junfeng Qiao. *Add w.r.t to betterFullStop*. May 25, 2023. URL: https://github.com/cmhughes/latexindent.pl/pull/447 (visited on 05/25/2023).
[24] Henrik Sloot. *feat: add devcontainer configuration*. May 20, 2023. URL: https://github.com/cmhughes/latexindent.pl/pull/443 (visited on 05/20/2023).
[25] Henrik Sloot. *fix: find local settings when working file dir is not working dir*. Feb. 15, 2023. URL: https://github.com/cmhughes/latexindent.pl/pull/422 (visited on 02/15/2023).

[26] Jesse Stricker. *Create cruft directory if it does not exist*. July 12, 2023. URL: https://github.com/cmhughes/latexindent.pl/pull/453 (visited on 07/12/2023).
[27] valtterikantanen. *fix: decode the name of the backup file*. Apr. 7, 2023. URL: https://github.com/cmhughes/latexindent.pl/pull/439 (visited on 04/07/2023).
[28] fengzyf. *Encoding work*. June 15, 2024. URL: https://github.com/cmhughes/latexindent.pl/pull/548 (visited on 06/15/2024).

SECTION A

Required Perl modules

If you intend to use `latexindent.pl` and *not* one of the supplied standalone executable files (`latexindent.exe` is available for Windows users without Perl, see Section 1.3.2), then you will need a few standard Perl modules.

If you can run the minimum code in Listing 569 as in

```
cmh:~$ perl helloworld.pl
```

then you will be able to run `latexindent.pl`, otherwise you may need to install the missing modules; see Sections A.1 and A.2.

LISTING 569: `helloworld.pl`

```
#!/usr/bin/perl

use strict;                          #     |
use warnings;                        #     |
use Encode;                          #     |
use Getopt::Long;                    #     |
use Data::Dumper;                    #  these modules are
use List::Util qw(max);              #  generally part
use PerlIO::encoding;                #  of a default perl distribution
use open ':std', ':encoding(UTF-8)';#     |
use Text::Wrap;                      #     |
use Text::Tabs;                      #     |
use FindBin;                         #     |
use File::Copy;                      #     |
use File::Basename;                  #     |
use File::Path;                      #     |
use File::HomeDir;                   # <--- typically requires install via cpanm
use YAML::Tiny;                      # <--- typically requires install via cpanm

print "hello world";
exit;
```

A.1 Module installer script

N: 2018-01-13

`latexindent.pl` ships with a helper script that will install any missing `perl` modules on your system; if you run

```
cmh:~$ perl latexindent-module-installer.pl
```

or

```
C:\Users\cmh>perl latexindent-module-installer.pl
```

then, once you have answered `Y`, the appropriate modules will be installed onto your distribution.

A.2 Manually installing modules

Manually installing the modules given in Listing 569 will vary depending on your operating system and `Perl` distribution.

A.2.1 Linux

A.2.1.1 perlbrew

Linux users may be interested in exploring Perlbrew [42]; an example installation would be:

```
cmh:~$ sudo apt-get install perlbrew
cmh:~$ perlbrew init
cmh:~$ perlbrew install perl-5.42.0
cmh:~$ perlbrew switch perl-5.42.0
cmh:~$ sudo apt-get install curl
cmh:~$ curl -L http://cpanmin.us | perl - App::cpanminus
cmh:~$ cpanm YAML::Tiny
cmh:~$ cpanm File::HomeDir
```

A.2.1.2 Ubuntu/Debian

For other distributions, the Ubuntu/Debian approach may work as follows

```
cmh:~$ sudo apt install perl
cmh:~$ sudo cpan -i App::cpanminus
cmh:~$ sudo cpanm YAML::Tiny
cmh:~$ sudo cpanm File::HomeDir
```

or else by running, for example,

```
cmh:~$ sudo perl -MCPAN -e'install␣"File::HomeDir"'
```

A.2.1.3 Ubuntu: using the texlive from apt-get

Ubuntu users that install texlive using `apt-get` as in the following

```
cmh:~$ sudo apt install texlive
cmh:~$ sudo apt install texlive-latex-recommended
```

may need the following additional command to work with `latexindent.pl`

```
cmh:~$ sudo apt install texlive-extra-utils
```

A.2.1.4 Ubuntu: users without perl

N: 2022-10-30

`latexindent-linux` is a standalone executable for Ubuntu Linux (and therefore does not require a Perl distribution) and caches copies of the Perl modules onto your system. It is available from [36].

A.2.1.5 Arch-based distributions

`latexindent` is included in Arch-packaged TeX Live, and can be installed by:

```
cmh:~$ sudo pacman -S texlive-binextra perl-yaml-tiny perl-file-homedir
```

To enable optional `-GCString` switch, install `perl-unicode-linebreak`:

```
cmh:~$ sudo pacman -S perl-unicode-linebreak
```

A.2.1.6 Alpine

If you are using Alpine, some `Perl` modules are not build-compatible with Alpine, but replacements are available through `apk`. For example, you might use the commands given in Listing 570; thanks to [12] for providing these details.

LISTING 570: `alpine-install.sh`

```
# Installing perl
apk --no-cache add miniperl perl-utils

# Installing incompatible latexindent perl dependencies via apk
apk --no-cache add \
    perl-log-dispatch \
    perl-namespace-autoclean \
    perl-specio \
    perl-unicode-linebreak

# Installing remaining latexindent perl dependencies via cpan
apk --no-cache add curl wget make
ls /usr/share/texmf-dist/scripts/latexindent
cd /usr/local/bin && \
    curl -L https://cpanmin.us/ -o cpanm && \
    chmod +x cpanm
cpanm -n App::cpanminus
cpanm -n File::HomeDir
cpanm -n Params::ValidationCompiler
cpanm -n YAML::Tiny
```

Users of NixOS might like to see https://github.com/cmhughes/latexindent.pl/issues/222 for tips.

A.2.2 Mac

Users of the Macintosh operating system might like to explore the following commands, for example:

```
cmh:~$ brew install perl
cmh:~$ brew install cpanm
cmh:~$
cmh:~$ cpanm YAML::Tiny
cmh:~$ cpanm File::HomeDir
```

Alternatively,

```
cmh:~$ brew install latexindent
```

N: 2022-10-30

`latexindent-macos` is a standalone executable for macOS (and therefore does not require a Perl distribution) and caches copies of the Perl modules onto your system. It is available from [36].

A.2.3 Windows

Strawberry Perl users on Windows might use `CPAN client`. All of the modules are readily available on CPAN [32]. `indent.log` will contain details of the location of the Perl modules on your system.

`latexindent.exe` is a standalone executable for Windows (and therefore does not require a Perl distribution) and caches copies of the Perl modules onto your system; if you wish to see where they are cached, use the `trace` option, e.g

```
C:\Users\cmh>latexindent.exe -t myfile.tex
```

A.3 The GCString switch

N: 2022-03-25

If you find that the `lookForAlignDelims` (as in Section 3.5) does not work correctly for your language, then you may wish to use the `Unicode::GCString` module .

This can be loaded by calling `latexindent.pl` with the `GCString` switch as in

```
cmh:~$ latexindent.pl --GCString myfile.tex
```

In this case, you will need to have the `Unicode::GCString` installed in your `perl` distribution by using, for example,

```
cmh:~$ cpanm Unicode::GCString
```

Note: this switch does *nothing* for `latexindent.exe` which loads the module by default. Users of `latexindent.exe` should not see any difference in behaviour whether they use this switch or not, as `latexindent.exe` loads the `Unicode::GCString` module.

SECTION B

Updating the path variable

`latexindent.pl` has a few scripts (available at [36]) that can update the `path` variables. Thank you to [6] for this feature. If you're on a Linux or Mac machine, then you'll want `CMakeLists.txt` from [36].

B.1 Add to path for Linux

To add `latexindent.pl` to the path for Linux, follow these steps:

1. download `latexindent.pl` and its associated modules, `defaultSettings.yaml`, to your chosen directory from [36] ;
2. within your directory, create a directory called `path-helper-files` and download `CMakeLists.txt` and `cmake_uninstall.cmake.in` from [36]/path-helper-files to this directory;
3. run

   ```
   cmh:~$ ls /usr/local/bin
   ```

   to see what is *currently* in there;
4. run the following commands

   ```
   cmh:~$ sudo apt-get update
   cmh:~$ sudo apt-get install --no-install-recommends cmake make # or any
       other generator
   cmh:~$ mkdir build && cd build
   cmh:~$ cmake ../path-helper-files
   cmh:~$ sudo make install
   ```
5. run

   ```
   cmh:~$ ls /usr/local/bin
   ```

   again to check that `latexindent.pl`, its modules and `defaultSettings.yaml` have been added.

To *remove* the files, run

```
cmh:~$ sudo make uninstall
```

B.2 Add to path for Windows

To add `latexindent.exe` to the path for Windows, follow these steps:

1. download `latexindent.exe`, `defaultSettings.yaml`, `add-to-path.bat` from [36] to your chosen directory;
2. open a command prompt and run the following command to see what is *currently* in your `%path%` variable;

```
C:\Users\cmh>echo %path%
```

3. right click on `add-to-path.bat` and *Run as administrator*;
4. log out, and log back in;
5. open a command prompt and run

   ```
   C:\Users\cmh>echo %path%
   ```

   to check that the appropriate directory has been added to your `%path%`.

To *remove* the directory from your `%path%`, run `remove-from-path.bat` as administrator.

SECTION C

Batches of files

N: 2022-03-25

You can instruct `latexindent.pl` to operate on multiple files. For example, the following calls are all valid

```
cmh:~$ latexindent.pl myfile1.tex
cmh:~$ latexindent.pl myfile1.tex myfile2.tex
cmh:~$ latexindent.pl myfile*.tex
```

We note the following features of the script in relation to the switches detailed in Section 2.

C.1 location of indent.log

If the `-c` switch is *not* active, then `indent.log` goes to the directory of the final file called.

If the `-c` switch *is* active, then `indent.log` goes to the specified directory.

C.2 interaction with -w switch

If the `-w` switch is active, as in

```
cmh:~$ latexindent.pl -w myfile*.tex
```

then files will be overwritten individually. Back-up files can be re-directed via the `-c` switch.

C.3 interaction with -o switch

If `latexindent.pl` is called using the `-o` switch as in

```
cmh:~$ latexindent.pl myfile*.tex -o=my-output-file.tex
```

and there are multiple files to operate upon, then the `-o` switch is ignored because there is only *one* output file specified.

More generally, if the `-o` switch does *not* have a `+` symbol at the beginning, then the `-o` switch will be ignored, and is turned it off.

For example

```
cmh:~$ latexindent.pl myfile*.tex -o=+myfile
```

will work fine because *each* `.tex` file will output to `<basename>myfile.tex`

Similarly,

```
cmh:~$ latexindent.pl myfile*.tex -o=++
```

will work because the 'existence check/incrementation' routine will be applied.

C.4 interaction with lines switch

This behaves as expected by attempting to operate on the line numbers specified for each file. See the examples in Section 6.

C.5 interaction with check switches

The exit codes for `latexindent.pl` are given in Table 1 on page 20.

When operating on multiple files with the check switch active, as in

```
cmh:~$ latexindent.pl myfile*.tex --check
```

then

- exit code 0 means that the text from *none* of the files has been changed;
- exit code 1 means that the text from *at least one* of the files been file changed.

The interaction with `checkv` switch is as in the check switch, but with verbose output.

C.6 when a file does not exist

What happens if one of the files can not be operated upon?

- if at least one of the files does not exist and `latexindent.pl` has been called to act upon multiple files, then the exit code is 3; note that `latexindent.pl` will try to operate on each file that it is called upon, and will not exit with a fatal message in this case;
- if at least one of the files can not be read and `latexindent.pl` has been called to act upon multiple files, then the exit code is 4; note that `latexindent.pl` will try to operate on each file that it is called upon, and will not exit with a fatal message in this case;
- if `latexindent.pl` has been told to operate on multiple files, and some do not exist and some cannot be read, then the exit code will be either 3 or 4, depending upon which it scenario it encountered most recently.

SECTION D

latexindent-yaml-schema.json

N: 2022-01-02

`latexindent.pl` ships with `latexindent-yaml-schema.json` which might help you when constructing your YAML files.

D.1 VSCode demonstration

To use `latexindent-yaml-schema.json` with `VSCode`, you can use the following steps:

1. download `latexindent-yaml-schema.json` from the `documentation` folder of [36], save it in whichever directory you would like, noting it for reference;
2. following the instructions from [37], for example, you should install the VSCode YAML extension [50];
3. set up your `settings.json` file using the directory you saved the file by adapting Listing 571; on my Ubuntu laptop this file lives at `/home/cmhughes/.config/Code/User/settings.json`.

LISTING 571: `settings.json`

```
{
  "yaml.schemas": {
    "/home/cmhughes/projects/latexindent/documentation/latexindent-yaml-schema.json":
    "/home/cmhughes/projects/latexindent/defaultSettings.yaml"
  },
  "redhat.telemetry.enabled": true
}
```

Alternatively, if you would prefer not to download the json file, you might be able to use an adapted version of Listing 572.

LISTING 572: `settings-alt.json`

```
{
  "yaml.schemas": {

    "https://raw.githubusercontent.com/cmhughes/latexindent.pl/main/documentation/latexindent-yaml-schema.json":
    "/home/cmhughes/projects/latexindent/defaultSettings.yaml"
  }
}
```

Finally, if your TeX distribution is up to date, then `latexindent-yaml-schema.json` *should* be in the documentation folder of your installation, so an adapted version of Listing 573 may work.

LISTING 573: `settings-alt1.json`

```
{
  "yaml.schemas": {
    "/usr/local/texlive/2021/texmf-dist/doc/support/latexindent/latexindent-yaml-schema.json":
    "/home/cmhughes/projects/latexindent/defaultSettings.yaml"
  }
}
```

If you have details of how to implement this schema in other editors, please feel encouraged to contribute to this documentation.

SECTION E

Using conda

If you use conda you'll only need

```
cmh:~$ conda install latexindent.pl -c conda-forge
```

This will install the executable and all its dependencies (including perl) in the activate environment. You don't even have to worry about `defaultSettings.yaml` as it included too, you can thus skip Sections A and B.

You can get a conda installation for example from [31] or from [30].

E.1 Sample conda installation on Ubuntu

On Ubuntu I followed the 64-bit installation instructions at [38] and then I ran the following commands:

```
cmh:~$ conda create -n latexindent.pl
cmh:~$ conda activate latexindent.pl
cmh:~$ conda install latexindent.pl -c conda-forge
cmh:~$ conda info --envs
cmh:~$ conda list
cmh:~$ conda run latexindent.pl -vv
```

I found the details given at [45] to be helpful.

SECTION F

Using docker

N: 2022-06-12

If you use docker you'll only need

```
cmh:~$ docker pull ghcr.io/cmhughes/latexindent.pl
```

This will download the image packed `latexindent`'s executable and its all dependencies. Thank you to [19] for contributing this feature; see also [40]. For reference, *ghcr* stands for *GitHub Container Repository*.

F.1 Sample docker installation on Ubuntu

To pull the image and show `latexindent`'s help on Ubuntu:

LISTING 574: `docker-install.sh`

```
# setup docker if not already installed
if ! command -v docker &> /dev/null; then
  sudo apt install docker.io -y
  sudo groupadd docker
  sudo gpasswd -a "$USER" docker
  sudo systemctl restart docker
  newgrp docker
fi

# download image and execute
docker pull ghcr.io/cmhughes/latexindent.pl
docker run ghcr.io/cmhughes/latexindent.pl -h
```

Once I have run the above, on subsequent logins I run

LISTING 575: `docker-install.sh`

```
newgrp docker
docker run ghcr.io/cmhughes/latexindent.pl -h
```

F.2 How to format on Docker

When you use `latexindent` with the docker image, you have to mount target `tex` file like this:

```
cmh:~$ docker run -v /path/to/local/myfile.tex:/myfile.tex
    ghcr.io/cmhughes/latexindent.pl -s -w myfile.tex
```

SECTION G

pre-commit

N: 2022-01-21

Users of `.git` may be interested in exploring the `pre-commit` tool [44], which is supported by `latexindent.pl`. Thank you to [20] for contributing this feature, and to [21] for their contribution to it.

To use the `pre-commit` tool, you will need to install `pre-commit`; sample instructions for Ubuntu are given in Section G.1. Once installed, there are two ways to use `pre-commit`: using `CPAN` or using `conda`, detailed in Section G.3 and Section G.4 respectively.

G.1 Sample pre-commit installation on Ubuntu

On Ubuntu I ran the following command:

```
cmh:~$ python3 -m pip install pre-commit
```

I then updated my path via .bashrc so that it includes the line in Listing 576.

LISTING 576: `.bashrc` update

```
...
export PATH=$PATH:/home/cmhughes/.local/bin
```

G.2 pre-commit defaults

The default values that are employed by `pre-commit` are shown in Listing 577.

LISTING 577: `.pre-commit-hooks.yaml` (default)

```
- id: latexindent
  name: latexindent.pl
  description: Run latexindent.pl (get dependencies using CPAN)
  minimum_pre_commit_version: 2.1.0
  entry: latexindent.pl
  args: ["--overwriteIfDifferent", "--silent", "--local"]
  language: perl
  types: [tex]
- id: latexindent-conda
  name: latexindent.pl
  description: Run latexindent.pl (get dependencies using Conda)
  minimum_pre_commit_version: 2.1.0
  entry: latexindent.pl
  args: ["--overwriteIfDifferent", "--silent", "--local"]
  language: conda
  types: [tex]
- id: latexindent-docker
  name: latexindent.pl
  description: Run latexindent.pl (get dependencies using Docker)
  minimum_pre_commit_version: 2.1.0
  entry: ghcr.io/cmhughes/latexindent.pl:4.0
  language: docker_image
  types: [tex]
  args: ["--overwriteIfDifferent", "--silent", "--local"]
```

In particular, the decision has deliberately been made (in collaboration with [21]) to have the default to employ the following switches: `overwriteIfDifferent`, `silent`, `local`; this is detailed in the lines that specify `args` in Listing 577.

Warning!

Users of `pre-commit` will, by default, have the `overwriteIfDifferent` switch employed. It is assumed that such users have version control in place, and are intending to overwrite their files.

G.3 pre-commit using CPAN

To use `latexindent.pl` with `pre-commit`, create the file `.pre-commit-config.yaml` given in Listing 578 in your git-repository.

LISTING 578: `.pre-commit-config.yaml` (cpan)

```
- repo: https://github.com/cmhughes/latexindent.pl
  rev: V4.0
  hooks:
  - id: latexindent
    args: [-s]
```

Once created, you should then be able to run the following command:

```
cmh:~$ pre-commit run --all-files
```

A few notes about Listing 578:

- the settings given in Listing 578 instruct `pre-commit` to use `CPAN` to get dependencies;
- this requires `pre-commit` and `perl` to be installed on your system;
- the `args` lists selected command-line options; the settings in Listing 578 are equivalent to calling

  ```
  cmh:~$ latexindent.pl -s myfile.tex
  ```

  for each `.tex` file in your repository;
- to instruct `latexindent.pl` to overwrite the files in your repository, then you can update Listing 578 so that `args: [-s, -w]`.

Naturally you can add options, or omit `-s` and `-w`, according to your preference.

G.4 pre-commit using conda

You can also rely on `conda` (detailed in Section E) instead of `CPAN` for all dependencies, including `latexindent.pl` itself.

LISTING 579: `.pre-commit-config.yaml` (conda)

```
- repo: https://github.com/cmhughes/latexindent.pl
  rev: V4.0
  hooks:
  - id: latexindent-conda
    args: [-s]
```

Once created, you should then be able to run the following command:

```
cmh:~$ pre-commit run --all-files
```

A few notes about Listing 578:

- the settings given in Listing 579 instruct `pre-commit` to use `conda` to get dependencies;
- this requires `pre-commit` and `conda` to be installed on your system;
- the `args` lists selected command-line options; the settings in Listing 578 are equivalent to calling

```
cmh:~$ conda run latexindent.pl -s myfile.tex
```

  for each `.tex` file in your repository;
- to instruct `latexindent.pl` to overwrite the files in your repository, then you can update Listing 578 so that `args: [-s, -w]`.

G.5 pre-commit using docker

You can also rely on `docker` (detailed in Section F) instead of `CPAN` for all dependencies, including `latexindent.pl` itself.

LISTING 580: `.pre-commit-config.yaml` (docker)

```
- repo: https://github.com/cmhughes/latexindent.pl
  rev: V4.0
  hooks:
    - id: latexindent-docker
      args: [-s]
```

Once created, you should then be able to run the following command:

```
cmh:~$ pre-commit run --all-files
```

A few notes about Listing 578:

- the settings given in Listing 580 instruct `pre-commit` to use `docker` to get dependencies;
- this requires `pre-commit` and `docker` to be installed on your system;
- the `args` lists selected command-line options; the settings in Listing 578 are equivalent to calling

```
cmh:~$ docker run -v /path/to/myfile.tex:/myfile.tex
    ghcr.io/cmhughes/latexindent.pl -s myfile.tex
```

  for each `.tex` file in your repository;
- to instruct `latexindent.pl` to overwrite the files in your repository, then you can update Listing 578 so that `args: [-s, -w]`.

G.6 pre-commit example using -l, -m switches

Let's consider a small example, with local `latexindent.pl` settings in `.latexindent.yaml`.

example 169 We use the local settings given in Listing 581.

LISTING 581: `.latexindent.yaml`

```
onlyOneBackUp: 1

modifyLineBreaks:
 oneSentencePerLine:
   manipulateSentences: 1
```

and `.pre-commit-config.yaml` as in Listing 582: ■

LISTING 582: `.pre-commit-config.yaml` (demo)

```
- repo: https://github.com/cmhughes/latexindent.pl
  rev: V4.0
  hooks:
  - id: latexindent
    args: [-l, -m, -s, -w]
```

Now running

```
cmh:~$ pre-commit run --all-files
```

is equivalent to running

```
cmh:~$ latexindent.pl -l -m -s -w myfile.tex
```

for each `.tex` file in your repository.

A few notes about Listing 582:

- the `-l` option was added to use the local `.latexindent.yaml` (where it was specified to only create one back-up file, as `git` typically takes care of this when you use `pre-commit`);
- `-m` to modify line breaks; in addition to `-s` to suppress command-line output, and `-w` to format files in place.

■

SECTION H

indentconfig options

The behaviour of `latexindent.pl` is controlled from the settings specified in any of the YAML files that you tell it to load. By default, `latexindent.pl` will only load `defaultSettings.yaml`, but there are a few ways that you can tell it to load your own settings files.

We focus our discussion on `indentconfig.yaml`, but there are other options which we also detail here.

N: 2023-01-01

H.1 indentconfig.yaml and .indentconfig.yaml

`latexindent.pl` will always check your home directory for `indentconfig.yaml` and `.indentconfig.yaml` (unless it is called with the `-d` switch), which is a plain text file you can create that contains the *absolute* paths for any settings files that you wish `latexindent.pl` to load. There is no difference between `indentconfig.yaml` and `.indentconfig.yaml`, other than the fact that `.indentconfig.yaml` is a 'hidden' file; thank you to [5] for providing this feature. In what follows, we will use `indentconfig.yaml`, but it is understood that this could equally represent `.indentconfig.yaml`. If you have both files in existence then `indentconfig.yaml` takes priority.

For Mac and Linux users, their home directory is `/username` while Windows (Vista onwards) is `C:\Users\username`[6] Listing 583 shows a sample `indentconfig.yaml` file.

LISTING 583: `indentconfig.yaml` (sample)

```
# Paths to user settings for latexindent.pl
#
# Note that the settings will be read in the order you
# specify here- each successive settings file will overwrite
# the variables that you specify

paths:
- /home/cmhughes/Documents/yamlfiles/mysettings.yaml
- /home/cmhughes/folder/othersettings.yaml
- /some/other/folder/anynameyouwant.yaml
- C:\Users\chughes\Documents\mysettings.yaml
- C:\Users\chughes\Desktop\test spaces\more spaces.yaml
```

Note that the `.yaml` files you specify in `indentconfig.yaml` will be loaded in the order in which you write them. Each file doesn't have to have every switch from `defaultSettings.yaml`; in fact, I recommend that you only keep the switches that you want to *change* in these settings files.

To get started with your own settings file, you might like to save a copy of `defaultSettings.yaml` in another directory and call it, for example, `mysettings.yaml`. Once you have added the path to `indentconfig.yaml` you can change the switches and add more code-block names to it as you see fit – have a look at Listing 584 for an example that uses four tabs for the default indent, adds the `tabbing` environment/command to the list of environments that contains alignment delimiters; you might also like to refer to the many YAML files detailed throughout the rest of this documentation.

[6] If you're not sure where to put `indentconfig.yaml`, don't worry `latexindent.pl` will tell you in the log file exactly where to put it assuming it doesn't exist already.

LISTING 584: `mysettings.yaml` (example)

```
# Default value of indentation
defaultIndent: "\t\t\t\t"

# environments that have tab delimiters, add more
# as needed
lookForAlignDelims:
    tabbing: 1
```

You can make sure that your settings are loaded by checking `indent.log` for details – if you have specified a path that `latexindent.pl` doesn't recognise then you'll get a warning, otherwise you'll get confirmation that `latexindent.pl` has read your settings file [7].

Warning!

When editing `.yaml` files it is *extremely* important to remember how sensitive they are to spaces. I highly recommend copying and pasting from `defaultSettings.yaml` when you create your first `whatevernameyoulike.yaml` file.

If `latexindent.pl` can not read your `.yaml` file it will tell you so in `indent.log`.

N: 2024-04-28

As of you can specify the `paths` field from Listing 583 within any of your `latexindent.yaml` and friends settings files. This can lead to creative nesting of configuration files; a demonstration is given in Section I on page 166.

H.2 localSettings.yaml and friends

U: 2021-03-14

The `-l` switch tells `latexindent.pl` to look for `localSettings.yaml` and/or friends in the *same directory* as `myfile.tex`. For example, if you use the following command

```
cmh:~$ latexindent.pl -l myfile.tex
```

then `latexindent.pl` will search for and then, assuming they exist, load each of the following files in the following order:

1. localSettings.yaml
2. latexindent.yaml
3. .localSettings.yaml
4. .latexindent.yaml

These files will be assumed to be in the same directory as `myfile.tex`, or otherwise in the current working directory. You do not need to have all of the above files, usually just one will be sufficient. In what follows, whenever we refer to `localSettings.yaml` it is assumed that it can mean any of the four named options listed above.

If you'd prefer to name your `localSettings.yaml` file something different, (say, `mysettings.yaml` as in Listing 584) then you can call `latexindent.pl` using, for example,

```
cmh:~$ latexindent.pl -l=mysettings.yaml myfile.tex
```

Any settings file(s) specified using the `-l` switch will be read *after* `defaultSettings.yaml` and, assuming they exist, any user setting files specified in `indentconfig.yaml`.

Your settings file can contain any switches that you'd like to change; a sample is shown in Listing 585, and you'll find plenty of further examples throughout this manual.

[7] Windows users may find that they have to end `.yaml` files with a blank line

LISTING 585: localSettings.yaml (example)

```
#  verbatim environments - environments specified
#  here will not be changed at all!
verbatimEnvironments:
    cmhenvironment: 0
    myenv: 1
```

You can make sure that your settings file has been loaded by checking `indent.log` for details; if it can not be read then you receive a warning, otherwise you'll get confirmation that `latexindent.pl` has read your settings file.

H.3 The -y|yaml switch

N: 2017-08-21

You may use the `-y` switch to load your settings; for example, if you wished to specify the settings from Listing 585 using the `-y` switch, then you could use the following command:

```
cmh:~$ latexindent.pl -y="verbatimEnvironments:cmhenvironment:0;myenv:1" myfile.tex
```

Note the use of `;` to specify another field within `verbatimEnvironments`. This is shorthand, and equivalent, to using the following command:

```
cmh:~$ latexindent.pl
    -y="verbatimEnvironments:cmhenvironment:0,verbatimEnvironments:myenv:1"
    myfile.tex
```

You may, of course, specify settings using the `-y` switch as well as, for example, settings loaded using the `-l` switch; for example,

```
cmh:~$ latexindent.pl -l=mysettings.yaml
    -y="verbatimEnvironments:cmhenvironment:0;myenv:1" myfile.tex
```

Any settings specified using the `-y` switch will be loaded *after* any specified using `indentconfig.yaml` and the `-l` switch.

If you wish to specify any regex-based settings using the `-y` switch, it is important not to use quotes surrounding the regex; for example, with reference to the 'one sentence per line' feature (Section 4.2 on page 86) and the listings within Listing 348 on page 89, the following settings give the option to have sentences end with a semicolon

```
cmh:~$ latexindent.pl -m
    --yaml='modifyLineBreaks:oneSentencePerLine:sentencesEndWith:other:\;'
```

Note that the `paths` settings (see Section I on page 166) can *not* be specified using the `-y` switch.

H.4 Settings load order

`latexindent.pl` loads the settings files in the following order:

1. `defaultSettings.yaml` is always loaded, and can not be renamed;
2. `anyUserSettings.yaml` and any other arbitrarily-named files specified in `indentconfig.yaml`;
3. `localSettings.yaml` but only if found in the same directory as `myfile.tex` and called with `-l` switch; this file can be renamed, provided that the call to `latexindent.pl` is adjusted accordingly (see Section H.2). You may specify both relative and absolute paths to other YAML files using the `-l` switch, separating multiple files using commas;

U: 2017-08-21

4. any settings specified in the `-y` switch.

N: 2017-08-21

A visual representation of this is given in Figure 1.

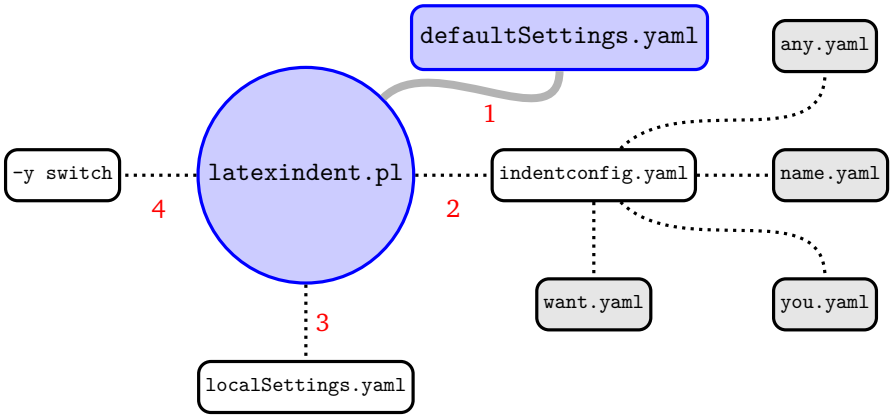


FIGURE 1: Schematic of the load order described in Section H.4; solid lines represent mandatory files, dotted lines represent optional files. `indentconfig.yaml` can contain as many files as you like. The files will be loaded in order; if you specify settings for the same field in more than one file, the most recent takes priority.

H.5 indentconfig options

N: 2023-01-01

This section describes the possible locations for the main configuration file, discussed in Section H.1. Thank you to [22] for this contribution.

The possible locations of `indentconfig.yaml` are read one after the other, and reading stops when a valid file is found in one of the paths.

Before stating the list, we give summarise in Table 5.

TABLE 5: indentconfig environment variable summaries

environment variable	type	Linux	macOS	Windows
LATEXINDENT_CONFIG	full path to file	✔	✔	✔
XDG_CONFIG_HOME	directory path	✔	✖	✖
LOCALAPPDATA	directory path	✖	✖	✔

The following list shows the checked options and is sorted by their respective priority. It uses capitalized and with a dollar symbol prefixed names (e.g. `$LATEXINDENT_CONFIG`) to symbolize environment variables. In addition to that the variable name `$homeDir` is used to symbolize your home directory.

1. The value of the environment variable `$LATEXINDENT_CONFIG` is treated as highest priority source for the path to the configuration file.
2. The next options are dependent on your operating system:
   - Linux:
     (a) The file at `$XDG_CONFIG_HOME/latexindent/indentconfig.yaml`
     (b) The file at `$homeDir/.config/latexindent/indentconfig.yaml`
   - Windows:
     (a) The file at `$LOCALAPPDATA\latexindent\indentconfig.yaml`
     (b) The file at `$homeDir\AppData\Local\latexindent\indentconfig.yaml`
   - Mac:
     (a) The file at `$homeDir/Library/Preferences/latexindent/indentconfig.yaml`
3. The file at `$homeDir/indentconfig.yaml`

4. The file at `$homeDir/.indentconfig.yaml`

H.6 Why to change the configuration location

This is mostly a question about what you prefer, some like to put all their configuration files in their home directory (see `$homeDir` above), whilst some like to sort their configuration. And if you don't care about it, you can just continue using the same defaults.

H.7 How to change the configuration location

This depends on your preferred location, if, for example, you would like to set a custom location, you would have to change the `$LATEXINDENT_CONFIG` environment variable.

Although the following example only covers `$LATEXINDENT_CONFIG`, the same process can be applied to `$XDG_CONFIG_HOME` or `$LOCALAPPDATA` because both are environment variables. You just have to change the path to your chosen configuration directory (e.g. `$homeDir/.config` or `$homeDir\AppData\Loca` on Linux or Windows respectively)

H.7.1 Linux

To change `$LATEXINDENT_CONFIG` on Linux you can run the following command in a terminal after changing the path:

```
cmh:~$ echo 'export␣LATEXINDENT_CONFIG="/home/cmh/latexindent-config.yaml"' >> ~/.profile
```

Context: This command adds the given line to your `.profile` file (which is commonly stored in `$HOME/.profile`). All commands in this file a run after login, so the environment variable will be set after your next login.

You can check the value of `$LATEXINDENT_CONFIG` by typing

```
cmh:~$ echo $LATEXINDENT_CONFIG
cmh:~$ /home/cmh/latexindent-config.yaml
```

Linux users interested in `$XDG_CONFIG_HOME` can explore variations of the following commands

```
cmh:~$ echo $XDG_CONFIG_HOME
cmh:~$ echo ${XDG_CONFIG_HOME:=$HOME/.config}
cmh:~$ echo $XDG_CONFIG_HOME
cmh:~$ mkdir /home/cmh/.config/latexindent
cmh:~$ touch /home/cmh/.config/latexindent/indentconfig.yaml
```

H.7.2 Windows

To change `$LATEXINDENT_CONFIG` on Windows you can run the following command in `powershell.exe` after changing the path:

```
C:\Users\cmh>[Environment]::SetEnvironmentVariable
C:\Users\cmh>    ("LATEXINDENT_CONFIG", "\your\config\path", "User")
```

This sets the environment variable for every user session.

H.7.3 Mac

To change `$LATEXINDENT_CONFIG` on macOS you can run the following command in a terminal after changing the path:

```
cmh:~$ echo 'export␣LATEXINDENT_CONFIG="/your/config/path"' >> ~/.profile
```

Context: This command adds the line to your `.profile` file (which is commonly stored in `$HOME/.profile`). All commands in this file a run after login, so the environment variable will be set after your next login.

SECTION I

paths demonstration

N: 2024-04-28

As detailed in Section H on page 160 , the `paths` field can be specified in any of your `YAML` files.

We will use the file in Listing 586 for demonstration in what follows.

LISTING 586: `paths-demo.tex`

```
\pathdemo[
opt arg
]{
mand arg
}
```

example 170 Consider the settings given in Listing 587 and Listing 588.

LISTING 587: `path1.yaml`

```
defaultIndent: ''
paths:
- path2.yaml
```

LISTING 588: `path2.yaml`

```
defaultIndent:␣'␣␣␣'
```

Upon calling

```
cmh:~$ latexindent.pl -l=path1.yaml paths-demo.tex
```

then we will receive the output given in Listing 589.

LISTING 589: `paths-demo-mod1.tex`

```
\pathdemo[
␣␣␣opt␣arg
]{
␣␣␣mand␣arg
}
```

We note that the settings from Listing 588 have been called from Listing 587.

On inspection of `indent.log` from the above call, we see the details of this part of the process given in Listing 590. ■

LISTING 590: path-test1.txt

```
YAML settings, reading from the following files:
       Reading USER settings from path1.yaml
       Reading path information from path1.yaml
       ---
       defaultIndent: ''
       paths:
         - path2.yaml

       ---
       defaultIndent: ''
       paths:
         - path2.yaml

       Reading USER settings from path2.yaml
       ---
       defaultIndent: '   '
```

■

example 171 Consider the settings given in Listing 591 to Listing 593.

LISTING 591: path3.yaml

```
defaultIndent: ''
paths:
- path4.yaml
```

LISTING 592: path4.yaml

```
defaultIndent:␣'␣␣␣'
paths:
-␣path5.yaml
```

LISTING 593: path5.yaml

```
defaultIndent:␣'␣'
```

Upon calling

```
cmh:~$ latexindent.pl -l=path3.yaml paths-demo.tex
```

then we will receive the output given in Listing 594.

LISTING 594: paths-demo-mod3.tex

```
\pathdemo[
␣opt␣arg
]{
␣mand␣arg
}
```

We see that `path3.yaml` calls `path4.yaml` which in turn calls `path5.yaml`.

On inspection of `indent.log` from the above call, we see the details of this part of the process given in Listing 595.

■

LISTING 595: path-test3.txt

```
YAML settings, reading from the following files:
       Reading USER settings from path3.yaml
       Reading path information from path3.yaml
       ---
       defaultIndent: ''
       paths:
         - path4.yaml

       ---
       defaultIndent: ''
       paths:
         - path4.yaml

       Reading USER settings from path4.yaml
       Reading path information from path4.yaml
       ---
       defaultIndent: '   '
       paths:
         - path5.yaml

       ---
       defaultIndent: '   '
       paths:
         - path5.yaml

       Reading USER settings from path5.yaml
       ---
       defaultIndent: ' '
```

SECTION J

logFilePreferences

`latexindent.pl` writes information to `indent.log`, some of which can be customized by changing `logFilePreferences`; see Listing 596. If you load your own user settings (see Section H.1 on page 160) then `latexindent.pl` will detail them in `indent.log`; you can choose not to have the details logged by switching `showEveryYamlRead` to `0`. Once all of your settings have been loaded, you can see the amalgamated settings in the log file by switching `showAmalgamatedSettings` to `1`, if you wish.

LISTING 596: `logFilePreferences`

```
logFilePreferences:
  showEveryYamlRead: 1
  showAmalgamatedSettings: 0
  endLogFileWith: '--------------'
  showGitHubInfoFooter: 1
  Dumper:
    Terse: 1
    Indent: 1
    Useqq: 1
    Deparse: 1
    Quotekeys: 0
    Sortkeys: 1
    Pair: " => "
```

The log file will end with the characters given in `endLogFileWith`, and will report the `GitHub` address of `latexindent.pl` to the log file if `showGitHubInfoFooter` is set to `1`.

U: 2021-03-14

Note: `latexindent.pl` no longer uses the `log4perl` module to handle the creation of the logfile.

U: 2021-06-19

Some of the options for Perl's `Dumper` module can be specified in Listing 596; see [34] and [33] for more information. These options will mostly be helpful for those calling `latexindent.pl` with the `-tt` option described in Section 2.1.

Listing 596 describes the options for customising the information given to the log file, and we provide a few demonstrations here.

SECTION K

Encoding

When using latexindent in different ways on different systems, the range of characters supported by its switches/flags/options (see Section 2.1 on page 13) may vary.

For the Windows executable file `latexindent.exe`, its options support UTF-8 characters.

For the Windows Perl script `latexindent.pl`, its option switch supports the characters supported by the encoding corresponding to the system code page. You can check the system code page by running the following command in either `cmd.exe` or `powershell.exe`:

```
C:\Users\cmh>  chcp
```

which may receive the following result

```
C:\Users\cmh>  Active code page: 936
```

and then the characters supported by the code page can be found in https://learn.microsoft.com/en-us/windows/win32/intl/code-page-identifiers. For example, the characters supported by the encoding corresponding to code page 936 are: ANSI/OEM Simplified Chinese (PRC, Singapore); Chinese Simplified (GB2312).

For Ubuntu Linux and macOS users, whether using the Perl script or the executable file, the options support UTF-8 characters.

Section L

dos2unix linebreak adjustment

`dos2unixlinebreaks`: ⟨*integer*⟩

N: 2021-06-19

If you use `latexindent.pl` on a dos-based Windows file on Linux then you may find that trailing horizontal space is not removed as you hope.

In such a case, you may wish to try setting `dos2unixlinebreaks` to 1 and employing, for example, the following command.

```
cmh:~$ latexindent.pl -y="dos2unixlinebreaks:1" myfile.tex
```

See [51] for further dertails.

SECTION M

Thanks

I first created `latexindent.pl` to help me format chapter files in a big project. After I blogged about it on the TEX stack exchange [29] I received some positive feedback and follow-up feature requests. A big thank you to Harish Kumar [2] who helped to develop and test the initial versions of the script.

The `YAML`-based interface of `latexindent.pl` was inspired by the wonderful `arara` tool; any similarities are deliberate, and I hope that it is perceived as the compliment that it is. Thank you to Paulo Cereda and the team for releasing this awesome tool; I initially worried that I was going to have to make a GUI for `latexindent.pl`, but the release of `arara` has meant there is no need.

There have been several contributors to the project so far (and hopefully more in the future!); thank you very much to the people detailed in Section 9.5 on page 143 for their valued contributions, and thank you to those who report bugs and request features at [36].

End

Listings

Index