latexindent.pl

Version 3.0

Chris Hughes *

January 17, 2017

Abstract

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 indentation after headings (such as chapter, section, etc) are also available. The script also has the ability to modifiy line breaks, and add comment symbols.

Contents

Listings

*and contributors! (See **??** on page ??.) For all communication, please visit [**latexindent-home**].

1 Introduction

1.1 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 [**cmhblog**] I received some positive feedback and follow-up feature requests. A big thank you to Harish Kumar 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.

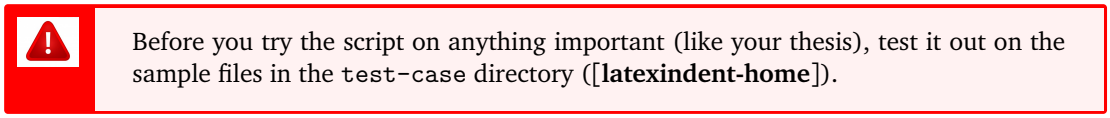
There have been several contributors to the project so far (and hopefully more in the future!); thank you very much to the people detailed in **??** on page ?? for their valued contributions.

1.2 License

`latexindent.pl` is free and open source, and it always will be. 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 8) but you should still be careful when using it. The script has been tested on many files, but there are some known limitations (see **??**). 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 ([**latexindent-home**]) with a complete minimum working example as I would like to improve the code as much as possible.



If you have used any version 2. of* `latexindent.pl`*, there are a few changes to the interface; see* **??** *on page ?? and the comments throughout this document for details*

2 Demonstration: before and after

Let's give a demonstration of some before and after code–after all, you probably won't want to try the script if you don't much like the results. You might also like to watch the video demonstration I made on youtube [**cmh:videodemo**]

FIX

As you look at Listings 1 to 6, remember that `latexindent.pl` is just following its rules, and there is nothing particular about these code snippets. All of the rules can be modified so that each user can personalize their indentation scheme.

In each of the samples given in Listings 1 to 6 the 'before' case is a 'worst case scenario' with no effort to make indentation. The 'after' result would be the same, regardless of the leading white space at the beginning of each line which is stripped by `latexindent.pl` (unless a `verbatim`-like environment or `noIndentBlock` is specified – more on this in Section 4).

LISTING 1: `filecontents` before

```
\begin{filecontents}{mybib.bib}
@online{strawberryperl,
title="Strawberry Perl",
url="http://strawberryperl.com/"}
@online{cmhblog,
title="A Perl script ...
url="...
}
\end{filecontents}
```

LISTING 2: `filecontents` after

```
\begin{filecontents}{mybib.bib}
  @online{strawberryperl,
    title="Strawberry Perl",
    url="http://strawberryperl.com/"}
  @online{cmhblog,
    title="A Perl script ...
    url="...
  }
\end{filecontents}
```

LISTING 3: `tikzset` before

```
\tikzset{
shrink inner sep/.code={
\pgfkeysgetvalue...
\pgfkeysgetvalue...
}
}
```

LISTING 4: `tikzset` after

```
\tikzset{
  shrink inner sep/.code={
      \pgfkeysgetvalue...
      \pgfkeysgetvalue...
    }
}
```

LISTING 5: `pstricks` before

```
\def\Picture#1{%
\def\stripH{#1}%
\begin{pspicture}[showgrid...
\psforeach{\row}{%
{{3,2.8,2.7,3,3.1}},%
{2.8,1,1.2,2,3},%
...
}{%
\expandafter...
}
\end{pspicture}}
```

LISTING 6: `pstricks` after

```
\def\Picture#1{%
  \def\stripH{#1}%
  \begin{pspicture}[showgrid...
    \psforeach{\row}{%
        {{3,2.8,2.7,3,3.1}},%
        {2.8,1,1.2,2,3},%
      ...
    }{%
      \expandafter...
    }
  \end{pspicture}}
```

3 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 and MiKTEXdistributions for Windows users. These files are also available from github [**latexindent-home**] should you wish to use them without a TEX distribution; in this case, you may like to read **??** on page ?? 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 3.1 and Section 3.2 respectively. We will discuss how to change the settings and behaviour of the script in Section 4 on page 7.

`latexindent.pl` ships with `latexindent.exe` for Windows users, so that you can use the script with or without a Perl distribution. If you plan to use `latexindent.pl` (i.e, the original Perl script) then you will need a few standard Perl modules–see **??** on page ?? for details.

3.1 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.

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

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

`-h, -help`

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

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

FIX

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

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

`-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 4, 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.

`-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).

Note that using `-o` is equivalent to using

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

Local settings: you might like to read Section 4 before using this switch. `latexindent.pl` will always load `defaultSettings.yaml` and if it is called with the `-l` switch and it finds `localSettings.yaml` in the same directory as `myfile.tex` 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`.

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`.

`-d, -onlydefault`

FIX

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

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

Only `defaultSettings.yaml`: you might like to read Section 4 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`/`.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.

`-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.

`-g, -logfile`

```
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 this, simply call the script with your chosen name after the `-g` switch.

`-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 **??** on page ??

`latexindent.pl` can also be called on a file without the file extension, for example `latexindent.pl myfile` and in which case, you can specify the order in which extensions are searched for; see Listing 8 on page 8 for full details.

3.2 From `arara`

Using `latexindent.pl` from the command line is fine for some folks, but others may find it easier to use from `arara`. `latexindent.pl` ships with an `arara` rule, `indent.yaml`, which can be copied to the directory of your other `arara` rules; otherwise you can add the directory in which `latexindent.pl` resides to your `araraconfig.yaml` file.

Once you have told `arara` where to find your `indent` rule, you can use it any of the ways described in Listing 7 (or combinations thereof). In fact, `arara` allows yet greater flexibility–you can use `yes/no`, `true/false`, or `on/off` to toggle the various options.

LISTING 7: `arara` sample usage

```
% arara: indent
% arara: indent: {overwrite: yes}
% arara: indent: {output: myfile.tex}
% arara: indent: {silent: yes}
% arara: indent: {trace: yes}
% arara: indent: {localSettings: yes}
% arara: indent: {onlyDefault: on}
% arara: indent: { cruft: /home/cmhughes/Desktop }
% arara: indent: { modifylinebreaks: yes }
\documentclass{article}
...
```

Hopefully the use of these rules is fairly self-explanatory, but for completeness Table 1 shows the relationship between `arara` directive arguments and the switches given in Section 3.1.

FIX

TABLE 1: `arara` directive arguments and corresponding switches

`arara` directive argument	switch
`overwrite`	`-w`
`output`	`-o`
`silent`	`-s`
`trace`	`-t`
`localSettings`	`-l`
`onlyDefault`	`-d`
`cruft`	`-c`
`modifylinebreaks`	`-m`

The `cruft` directive does not work well when used with directories that contain spaces.

4 default, user, and local settings

`latexindent.pl` loads its settings from `defaultSettings.yaml` (rhymes with camel). 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.

4.1 defaultSettings.yaml

If you look in `defaultSettings.yaml` you'll find the switches 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.

You can certainly feel free to edit `defaultSettings.yaml`, but this is not ideal as it may be overwritten when you update your TeX distribution – all of your hard work tweaking the script would be undone! Don't worry, there's a solution, feel free to peek ahead to **??** if you like.

`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.

Calling `latexindent.pl myfile` with the (default) settings specified in Listing 8 means that the script will first look for `myfile.tex`, then `myfile.sty`, `myfile.cls`, and finally `myfile.bib` in order.

LISTING 8: `fileExtensionPreference`

```
22 fileExtensionPreference:
23     .tex: 1
24     .sty: 2
25     .cls: 3
26     .bib: 4
```

`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`.

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.

`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`.

`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 9.

LISTING 9: `verbatimEnvironments`

```
64 verbatimEnvironments:
65     verbatim: 1
66     lstlisting: 1
```

Note that if you put an environment in `verbatimEnvironments` and in other fields such as `lookForAlignDelims` or

LISTING 10: `verbatimCommands`

```
69 verbatimCommands:
70     verb: 1
```

noAdditionalIndent then latexindent.pl will *always* prioritize verbatimEnvironments.

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 **??** on page ??).

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 11.

LISTING 11: noIndentBlock

```
77 noIndentBlock:
78     noindent: 1
79     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 12 for example.

LISTING 12: noIndentBlock demonstration

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

removeTrailingWhitespace: ⟨*fields*⟩

Trailing white space can be removed both *before* and *after* processing the document, as detailed in Listing 13; each of the fields can take the values 0 or 1. Thanks to [**vosskuhle**] for providing this feature.

LISTING 13: removeTrailingWhitespace

```
82 removeTrailingWhitespace:
83     beforeProcessing: 0
84     afterProcessing: 1
```

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 14. The behaviour of latexindent.pl on these environments is determined by their location (preamble or not), and the value indentPreamble, discussed next.

LISTING 14: fileContentsEnvironments

```
88 fileContentsEnvironments:
89     filecontents: 1
90     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 15, 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 15: lookForPreamble

```
lookForPreamble:
    .tex: 1
    .sty: 0
    .cls: 0
    .bib: 0
```

`preambleCommandsBeforeEnvironments`: **0**|**1**

Assuming that `latexindent.pl` is asked to operate upon the preamble of a document, when this switch is set to `0` then environment code blocks will be sought first, and then command code blocks. When this switch is set to `1`, commands will be sought first. The example that first motivated this switch contained the code given in Listing 16.

LISTING 16: Motivating `preambleCommandsBeforeEnvironments`

```
...
preheadhook={\begin{mdframed}[style=myframedstyle]},
postfoothook=\end{mdframed},
...
```

`defaultIndent`: ⟨*horizontal space*⟩

This is the default indentation (`\t` means a tab, and is the default value) used in the absence of other details for the command or environment we are working with; see `indentRules` for more details (**??**).

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

`lookForAlignDelims`: ⟨*fields*⟩

This contains a list of environments and/or commands that are operated upon in a special way by `latexindent.pl` (see Listing 17). In fact, the fields in `lookForAlignDelims` can actually take two different forms: the *basic* version is shown in Listing 17 and the *advanced* version in Listing 20; we will discuss each in turn.

LISTING 17: `lookForAlignDelims` (basic)

```
lookForAlignDelims:
   tabular: 1
   tabularx: 1
   longtable: 1
   array: 1
   matrix: 1
   ...
```

The environments specified in this field will be operated on in a special way by `latexindent.pl`. In particular, it will try and align each column by its alignment tabs. It does have some limitations (discussed further in **??**), but in many cases it will produce results such as those in Listings 18 and 19.

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 11).

LISTING 18: `tabular` before

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

LISTING 19: `tabular` after (basic)

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

If you wish to remove the alignment of the \\ within a delimiter-aligned block, then the advanced form of `lookForAlignDelims` shown in Listing 20 is for you.

LISTING 20: `lookForAlignDelims` (advanced)

```
114 lookForAlignDelims:
115    tabular:
116       delims: 1
117       alignDoubleBackSlash: 1
118       spacesBeforeDoubleBackSlash: 2
119    tabularx:
120       delims: 1
121    longtable: 1
```

Note that you can use a mixture of the basic and advanced form: in Listing 20 `tabular` and `tabularx` are advanced and `longtable` is basic. When using the advanced form, each field should receive at least 1 sub-field, and *can* (but does not have to) receive up to 3 fields:

- `delims`: switch equivalent to simply specifying, for example, `tabular: 1` in the basic version shown in Listing 17 (default: 1);
- `alignDoubleBackSlash`: switch to determine if \\ should be aligned (default: 1);
- `spacesBeforeDoubleBackSlash`: optionally, specifies the number of spaces to be inserted before (non-aligned) \\. In order to use this field, `alignDoubleBackSlash` needs to be set to 0 (default: 0).

Assuming that you have the settings in Listing 20 saved in `mysettings.yaml`, and the code from Listing 18 in `myfile.tex` and you run

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

then you should receive the before-and-after results shown in Listings 21 and 22; note that the ampersands have been aligned, but the \\ have not (compare the alignment of \\ in Listings 19 and 22).

LISTING 21: `tabular` before

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

LISTING 22: `tabular` after (advanced)

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

Using `spacesBeforeDoubleBackSlash: 3` gives Listings 23 and 24, note the spacing before the \\ in Listing 24.

LISTING 23: `tabular` before

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

LISTING 24: `tabular` after (spacing)

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

As of Version 3.0, the alignment routine works on mandatory and optional arguments within commands, and also within 'special' code blocks (see); for example, assuming that you have a command called `\matrix` and that it is populated within `lookForAlignDelims` (which it is, by default), then the before-and-after results shown in Listings 25 and 26 are achievable by default.

FIX

Listing 25: matrix before

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

Listing 26: matrix after

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

FIX

FIX

`indentAfterItems`: ⟨*fields*⟩

The environments 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 28 and 29

Listing 27: indentAfterItems

```
indentAfterItems:
    itemize: 1
    enumerate: 1
    list: 1
```

Listing 28: items before

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

Listing 29: items after

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

`itemNames`: ⟨*fields*⟩

If you have your own `item` commands (perhaps you prefer to use `myitem`, for example) then you can put populate them in `itemNames`. For example, users of the `exam` document class might like to add `parts` to `indentAfterItems` and `part` to `itemNames` to their user settings–see **??** on page ?? for details of how to configure user settings, and **??** on page ?? in particular.

Listing 30: itemNames

```
itemNames:
    item: 1
    myitem: 1
```

`specialBeginEnd`: ⟨*fields*⟩

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 31 shows the default settings of `specialBeginEnd`.

Listing 31: specialBeginEnd

```
specialBeginEnd:
    displayMath:
        begin: '\\\['
        end: '\\\]'
        lookForThis: 1
    inlineMath:
        begin: '(?<!\$)(?<!\\)\$(?!\$)'
        end: '(?<!\\)\$(?!\$)'
        lookForThis: 1
    displayMathTeX:
        begin: '\$\$'
        end: '\$\$'
        lookForThis: 1
```

The field `displayMath` represents `\[...\]`, `inlineMath` represents `$...$` and `displayMathTex`

represents `$$...$$`. You can, of course, rename these in your own YAML files (see **??** on page ??); indeed, you might like to set up your own specil begin and end statements.

A demonstration of the before-and-after results are shown in Listings 32 and 33.

LISTING 32: `special1.tex` before

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

LISTING 33: `special1.tex` after

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

For each field, the `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`.

`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 34: `indentAfterHeadings`

```
185 indentAfterHeadings:
186     part:
187         indentAfterThisHeading: 0
188         level: 1
189     chapter:
190         indentAfterThisHeading: 0
191         level: 2
192     section:
193         indentAfterThisHeading: 0
194         level: 3
```

This field is slightly different from most of the fields that we have considered previously, because each element is itself a field which has two elements: `indent` and `level`. (Similar in structure to the advanced form of `lookForAlignDelims` in Listing 20.)

The default settings do *not* place indentation after a heading, but you can easily switch them on by changing `indentAfterThisHeading: 0` to `indentAfterThisHeading: 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` ; you will find the default `indentRules` contains `chapter: " "` which tells `latexindent.pl` simply to use a space character after headings (once `indent` is set to `1` for `chapter`).

FIX

For example, assuming that you have read **??** on page ??, say that you have the code in Listing 36 saved into `headings1.yaml`, and that you have the text from Listing 36 saved into `headings1.tex`.

LISTING 35: headings1.yaml

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

LISTING 36: headings1.tex

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

If you run the command

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

then you should receive the output given in Listing 37.

LISTING 37: headings1.tex first modification

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

LISTING 38: 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 36 so that the paragraph level is 1; after running

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

you should now receive the code given in Listing 38; notice that the paragraph and subsection are at the same indentation level.

4.2 noAdditionalIndent and indentRules

latexindent.pl 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.

FIX

4.2.1 Environments and their arguments

There are a few different YAML switches governing the indentation of environments; let's start with the simple sample code shown in Listing 39.

LISTING 39: myenv.tex

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

noAdditionalIndent: ⟨*fields*⟩

If we do not wish myenv to receive any additional indentation, we have a few choices available to us, as demonstrated in Listings 40 and 41.

LISTING 40: myenv-noAdd1.yaml

```
1 noAdditionalIndent:
2     myenv: 1
```

LISTING 41: myenv-noAdd2.yaml

```
1 noAdditionalIndent:
2     myenv:
3         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 42; note in particular that the environment myenv has not received any *additional* indentation, but that the outer environment *has* still received indentation.

LISTING 42: myenv.tex output (using either Listings 40 and 41)

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

Upon changing the YAML files to those shown in Listings 43 and 44, 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 45.

LISTING 43: myenv-noAdd3.yaml

```
1 noAdditionalIndent:
2     myenv: 0
```

LISTING 44: myenv-noAdd4.yaml

```
1 noAdditionalIndent:
2     myenv:
3         body: 0
```

LISTING 45: myenv.tex output (using either Listings 43 and 44)

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

Let's now allow myenv to have some optional and mandatory arguments, as in Listing 46.

LISTING 46: 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 47; 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 40), then *all* parts of the environment (body, optional and mandatory arguments) are assumed to want no additional indent.

LISTING 47: myenv-args.tex using Listing 40

```
\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}
```

We may customise noAdditionalIndent for optional and mandatory arguments of the myenv environment, as shown in, for example, Listings 48 and 49.

LISTING 48: myenv-noAdd5.yaml

```
1 noAdditionalIndent:
2     myenv:
3         body: 0
4         optionalArguments: 1
5         mandatoryArguments: 0
```

LISTING 49: myenv-noAdd6.yaml

```
1 noAdditionalIndent:
2     myenv:
3         body: 0
4         optionalArguments: 0
5         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 50 and 51. Note that in Listing 50 the text for the *optional* argument has not received any additional indentation, and that in Listing 51 the *mandatory* argument has not received any additional indentation; in both cases, the *body* has not received any additional indentation.

LISTING 50: myenv-args.tex using Listing 48

```
\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 51: myenv-args.tex using Listing 49

```
\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*⟩

We may also specify indentation rules for environment code blocks using the `indentRules` field; see, for example, Listings 52 and 53.

LISTING 52: myenv-rules1.yaml

```
1 indentRules:
2     myenv: "    "
```

LISTING 53: myenv-rules2.yaml

```
1 indentRules:
2     myenv:
3         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 54; note in particular that the environment `myenv` has not received any *additional* indentation, but that the `outer` environment *has* still received indentation.

LISTING 54: myenv.tex output (using either Listings 52 and 53)

```
\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.

Let's now return to the example in Listing 46 that contains optional and mandatory arguments. Upon using Listing 52 as in

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

we obtain the output in Listing 55; note that the body, optional argument and mandatory argument have *all* received the same customised indentation.

LISTING 55: myenv-args.tex using Listing 52

```
\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}
```

You can specify different indentation rules for the different features using, for example, Listings 56 and 57

LISTING 56: myenv-rules3.yaml

```
1 indentRules:
2     myenv:
3         body: "    "
4         optionalArguments: " "
```

LISTING 57: myenv-rules4.yaml

```
1 indentRules:
2     myenv:
3         body: "    "
4         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 58 and 59.

LISTING 58: myenv-args.tex using Listing 56

```
\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 59: myenv-args.tex using Listing 57

```
\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 58, 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 59, 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

LISTING 60: env-noAdditionalGlobal.yaml

```
240 noAdditionalIndentGlobal:
241     environments: 0
```

in particular *for the environments* key (see Listing 60). Let's say that you change the value of `environments` to 1 in Listing 60, 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 61 and 62; in Listing 61 notice that *both* environments receive no additional indentation but that the arguments of `myenv` still *do* receive indentation. In Listing 62 notice that the *outer* environment does not receive additional indentation, but because of the settings from `myenv-rules1.yaml` (in Listing 52 on page 17), the `myenv` environment still *does* receive indentation.

LISTING 61: `myenv-args.tex` using Listing 60

```
\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 62: `myenv-args.tex` using Listings 52 and 60

```
\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}
```

In fact, `noAdditionalIndentGlobal` also contains keys that control the indentation of optional and mandatory arguments; on referencing Listings 63 and 64

LISTING 63: `opt-args-no-add-glob.yaml`

```
1 noAdditionalIndentGlobal:
2     optionalArguments: 1
```

LISTING 64: `mand-args-no-add-glob.yaml`

```
1 noAdditionalIndentGlobal:
2     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 65 and 66. Notice that in Listing 65 the *optional* argument has not received any additional indentation, and in Listing 66 the *mandatory* argument has not received any additional indentation.

LISTING 65: `myenv-args.tex` using Listing 63

```
\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 66: `myenv-args.tex` using Listing 64

```
\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 `indentRules` as detailed in Listing 67; if you change the `environments` field to anything involving horizontal space, say `" "`, and then run the following commands

LISTING 67: `env-indentRulesGlobal.yaml`

```
256 indentRulesGlobal:
257     environments: 0
```

```
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 68 and 69. Note that in Listing 68, both the environment blocks have received a single-space indentation, whereas in Listing 69 the `outer` environment has received single-space indentation (specified by `indentRulesGlobal`), but `myenv` has received `"  "`, as specified by the particular `indentRules` for `myenv` Listing 52 on page 17.

LISTING 68: `myenv-args.tex` using Listing 67

```
\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 69: `myenv-args.tex` using Listings 52 and 67

```
\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}
```

You can specify `indentRulesGlobal` for both optional and mandatory arguments, as detailed in Listings 70 and 71

LISTING 70: `opt-args-indent-rules-glob.yaml`

```
1 indentRulesGlobal:
2     optionalArguments: "\t\t"
```

LISTING 71: `mand-args-indent-rules-glob.yaml`

```
1 indentRulesGlobal:
2     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 72 and 73. Note that the *optional* argument in Listing 72 has received two tabs worth of indentation, while the *mandatory* argument has done so in Listing 73.

LISTING 72: myenv-args.tex using Listing 70

```
\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 73: myenv-args.tex using Listing 71

```
\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}
```

4.2.2 Environments with items

With reference to Listings 27 and 30 on page 12, some commands may contain `item` commands; for the purposes of this discussion, we will use the code from Listing 28 on page 12.

Assuming that you've populated `itemNames` with the name of your `item`, you can put the item name into `noAdditionalIndent` as in Listing 74, although a more efficient approach may be to change the relevant field in `itemNames` to `0`. Similarly, you can customise the indentation that your `item` receives using `indentRules`, as in Listing 75

LISTING 74: item-noAdd1.yaml

```
noAdditionalIndent:
    item: 1
# itemNames:
#   item: 0
```

LISTING 75: 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 76 and 77; note that in Listing 76 that the text after each `item` has not received any additional identation, and in Listing 77, the text after each `item` has received a single space of indentation, specified by Listing 75.

LISTING 76: items1.tex using Listing 74

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

LISTING 77: items1.tex using Listing 75

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

Alternatively, you might like to populate `noAdditionalIndentGlobal` or `indentRulesGlobal` using the `items` key, as demonstrated in Listings 78 and 79. Note that there is a need to 'reset/remove' the `item` field from `indentRules` in both cases (see the hierarchy description given on page 14) as the `item` command is a member of `indentRules` by default.

LISTING 78: items-noAdditionalGlobal.yaml

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

LISTING 79: 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 76 and 77 are obtained; note, however, that *all* such `item` commands without their own individual `noAdditionalIndent` or `indentRules` settings would behave as in these listings.

4.2.3 Commands with arguments

Let's begin with the simple example Listing 80; when `latexindent.pl` operates on this file, the default output is shown in Listing 81.

LISTING 80: `mycommand.tex`

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

LISTING 81: `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 40 and 41 on page 15) we may specify `noAdditionalIndent` either in 'scalar' form, or in 'field' form, as shown in Listings 82 and 83

LISTING 82: `mycommand-noAdd1.yaml`

```
1 noAdditionalIndent:
2     mycommand: 1
```

LISTING 83: `mycommand-noAdd2.yaml`

```
1 noAdditionalIndent:
2     mycommand:
3         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 84 and 85

LISTING 84: `mycommand.tex` using Listing 82

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

LISTING 85: `mycommand.tex` using Listing 83

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

Note that in Listing 84 that the 'body', optional argument *and* mandatory argument have *all* received no additional indentation, while in Listing 85, 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.

We may further customise *noAdditionalIndent* for `mycommand` as we did in Listings 48 and 49 on page 16; explicit examples are given in Listings 86 and 87.

LISTING 86: mycommand-noAdd3.yaml

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

LISTING 87: 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 88 and 89.

LISTING 88: mycommand.tex using Listing 86

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

LISTING 89: mycommand.tex using Listing 87

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

Attentive readers will note that the body of `mycommand` in both Listings 88 and 89 has received no additional indent, even though `body` is explicitly set to `0` in both Listings 86 and 87. This is because, by default, `noAdditionalIndentGlobal` for `commands` is set to `1` by default; this can be easily fixed as in Listings 90 and 91.

LISTING 90: mycommand-noAdd5.yaml

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

LISTING 91: 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 92 and 93.

LISTING 92: mycommand.tex using Listing 90

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

LISTING 93: mycommand.tex using Listing 91

```
\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 56 and 57 on page 18 and Listings 67, 70 and 71 on page 19 and on page 20.