

latexindent.pl

Version 3.0

Chris Hughes *

January 25, 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 modify line breaks, and add comment symbols.

Contents

1	Introduction	5
1.1	Thanks	5
1.2	License	5
2	Demonstration: before and after	5
3	How to use the script	6
3.1	From the command line	6
3.2	From <code>arara</code>	9
4	default, user, and local settings	10
4.1	<code>defaultSettings.yaml</code>	10
4.1.1	The code blocks known <code>latexindent.pl</code>	17
4.2	<code>noAdditionalIndent</code> and <code>indentRules</code>	18
4.2.1	Environments and their arguments	19
4.2.2	Environments with items	25
4.2.3	Commands with arguments	26
4.2.4	<code>ifelsefi</code> code blocks	28
4.2.5	<code>specialBeginEnd</code> code blocks	30
4.2.6	<code>afterHeading</code> code blocks	31
5	The <code>-m</code> (modifylinebreaks) switch	33
5.1	Poly-switches	34
5.2	<code>modifyLineBreaks</code> for environments	34
5.2.1	Adding line breaks (poly-switches set to 1 or 2)	34
5.2.2	Removing line breaks (poly-switches set to -1)	37
5.3	Poly-switches for other code blocks	39
6	<code>indentconfig.yaml</code> and <code>.indentconfig.yaml</code> (for user settings)	40
6.1	<code>localSettings.yaml</code>	42
6.2	Settings load order	43

*and contributors! (See Section 7.2 on page 43.) For all communication, please visit [6].



7	References	43
7.1	External links	43
7.2	Contributors	43
A	Required Perl modules	44
B	The arara rule	45
C	Updating the path variable	45
C.1	Add to path for Linux	45
C.2	Add to path for Windows	46
D	Differences from Version 2.2 to 3.0	46

Listings

LISTING 1:	filecontents before	6
LISTING 2:	filecontents after	6
LISTING 3:	tikzset before	6
LISTING 4:	tikzset after	6
LISTING 5:	pstricks before	6
LISTING 6:	pstricks after	6
LISTING 7:	arara sample usage	9
LISTING 8:	fileExtensionPreference	10
LISTING 9:	logFilePreferences	11
LISTING 10:	verbatimEnvironments	11
LISTING 11:	verbatimCommands	11
LISTING 12:	noIndentBlock	12
LISTING 13:	noIndentBlock demonstration	12
LISTING 14:	removeTrailingWhitespace	12
LISTING 15:	fileContentsEnvironments	12
LISTING 16:	lookForPreamble	13
LISTING 17:	Motivating preambleCommandsBeforeEnvironments	13
LISTING 19:	tabular before	14
LISTING 20:	tabular after (basic)	14
LISTING 21:	lookForAlignDelims (advanced)	14
LISTING 22:	tabular before	14
LISTING 23:	tabular after (advanced)	14
LISTING 24:	tabular before	14
LISTING 25:	tabular after (spacing)	14
LISTING 26:	matrix before	15
LISTING 27:	matrix after	15
LISTING 28:	indentAfterItems	15
LISTING 29:	items before	15
LISTING 30:	items after	15
LISTING 31:	itemName	15
LISTING 32:	specialBeginEnd	15
LISTING 33:	special1.tex before	16
LISTING 34:	special1.tex after	16
LISTING 35:	indentAfterHeadings	16
LISTING 36:	headings1.yaml	17
LISTING 37:	headings1.tex	17
LISTING 38:	headings1.tex using Listing 36	17
LISTING 39:	headings1.tex second modification	17
LISTING 48:	myenv.tex	19
LISTING 49:	myenv-noAdd1.yaml	19
LISTING 50:	myenv-noAdd2.yaml	19
LISTING 51:	myenv.tex output (using either Listings 49 and 50)	20
LISTING 52:	myenv-noAdd3.yaml	20



LISTING 53: myenv-noAdd4.yaml	20
LISTING 54: myenv.tex output (using either Listings 52 and 53)	20
LISTING 55: myenv-args.tex	20
LISTING 56: myenv-args.tex using Listing 49	21
LISTING 57: myenv-noAdd5.yaml	21
LISTING 58: myenv-noAdd6.yaml	21
LISTING 59: myenv-args.tex using Listing 57	21
LISTING 60: myenv-args.tex using Listing 58	21
LISTING 61: myenv-rules1.yaml	22
LISTING 62: myenv-rules2.yaml	22
LISTING 63: myenv.tex output (using either Listings 61 and 62)	22
LISTING 64: myenv-args.tex using Listing 61	22
LISTING 65: myenv-rules3.yaml	22
LISTING 66: myenv-rules4.yaml	22
LISTING 67: myenv-args.tex using Listing 65	23
LISTING 68: myenv-args.tex using Listing 66	23
LISTING 69: env-noAdditionalGlobal.yaml	23
LISTING 70: myenv-args.tex using Listing 69	24
LISTING 71: myenv-args.tex using Listings 61 and 69	24
LISTING 72: opt-args-no-add-glob.yaml	24
LISTING 73: mand-args-no-add-glob.yaml	24
LISTING 74: myenv-args.tex using Listing 72	24
LISTING 75: myenv-args.tex using Listing 73	24
LISTING 76: env-indentRulesGlobal.yaml	24
LISTING 77: myenv-args.tex using Listing 76	25
LISTING 78: myenv-args.tex using Listings 61 and 76	25
LISTING 79: opt-args-indent-rules-glob.yaml	25
LISTING 80: mand-args-indent-rules-glob.yaml	25
LISTING 81: myenv-args.tex using Listing 79	25
LISTING 82: myenv-args.tex using Listing 80	25
LISTING 83: item-noAdd1.yaml	26
LISTING 84: item-rules1.yaml	26
LISTING 85: items1.tex using Listing 83	26
LISTING 86: items1.tex using Listing 84	26
LISTING 87: items-noAdditionalGlobal.yaml	26
LISTING 88: items-indentRulesGlobal.yaml	26
LISTING 89: mycommand.tex	27
LISTING 90: mycommand.tex default output	27
LISTING 91: mycommand-noAdd1.yaml	27
LISTING 92: mycommand-noAdd2.yaml	27
LISTING 93: mycommand.tex using Listing 91	27
LISTING 94: mycommand.tex using Listing 92	27
LISTING 95: mycommand-noAdd3.yaml	27
LISTING 96: mycommand-noAdd4.yaml	27
LISTING 97: mycommand.tex using Listing 95	28
LISTING 98: mycommand.tex using Listing 96	28
LISTING 99: mycommand-noAdd5.yaml	28
LISTING 100: mycommand-noAdd6.yaml	28
LISTING 101: mycommand.tex using Listing 99	28
LISTING 102: mycommand.tex using Listing 100	28
LISTING 103: ifelsefi1.tex	29
LISTING 104: ifelsefi1.tex default output	29
LISTING 105: ifnum-noAdd.yaml	29
LISTING 106: ifnum-indent-rules.yaml	29
LISTING 107: ifelsefi1.tex using Listing 105	29
LISTING 108: ifelsefi1.tex using Listing 106	29
LISTING 109: ifelsefi-noAdd-glob.yaml	29
LISTING 110: ifelsefi-indent-rules-global.yaml	29



LISTING 111: ifelsefi1.tex using Listing 109	30
LISTING 112: ifelsefi1.tex using Listing 110	30
LISTING 113: displayMath-noAdd.yaml	30
LISTING 114: displayMath-indent-rules.yaml	30
LISTING 115: special1.tex using Listing 113	30
LISTING 116: special1.tex using Listing 114	30
LISTING 117: special-noAdd-glob.yaml	30
LISTING 118: special-indent-rules-global.yaml	30
LISTING 119: special1.tex using Listing 117	31
LISTING 120: special1.tex using Listing 118	31
LISTING 121: headings2.tex	31
LISTING 122: headings2.tex using Listing 123	31
LISTING 123: headings3.yaml	31
LISTING 124: headings2.tex using Listing 125	31
LISTING 125: headings4.yaml	31
LISTING 126: headings2.tex using Listing 127	32
LISTING 127: headings5.yaml	32
LISTING 128: headings2.tex using Listing 129	32
LISTING 129: headings6.yaml	32
LISTING 130: headings2.tex using Listing 131	32
LISTING 131: headings7.yaml	32
LISTING 132: headings2.tex using Listing 133	32
LISTING 133: headings8.yaml	32
LISTING 134: headings2.tex using Listing 135	33
LISTING 135: headings9.yaml	33
LISTING 137: mlb1.tex	34
LISTING 138: mlb1.tex out output	34
LISTING 139: environments	34
LISTING 140: env-mlb1.tex	34
LISTING 141: env-mlb1.yaml	35
LISTING 142: env-mlb2.yaml	35
LISTING 143: env-mlb.tex using Listing 141	35
LISTING 144: env-mlb.tex using Listing 142	35
LISTING 145: env-mlb3.yaml	35
LISTING 146: env-mlb4.yaml	35
LISTING 147: env-mlb.tex using Listing 145	35
LISTING 148: env-mlb.tex using Listing 146	35
LISTING 149: env-mlb5.yaml	35
LISTING 150: env-mlb6.yaml	35
LISTING 151: env-mlb.tex using Listing 149	36
LISTING 152: env-mlb.tex using Listing 150	36
LISTING 153: env-mlb7.yaml	36
LISTING 154: env-mlb8.yaml	36
LISTING 155: env-mlb.tex using Listing 153	36
LISTING 156: env-mlb.tex using Listing 154	36
LISTING 157: env-mlb2.tex	36
LISTING 158: env-mlb3.tex	36
LISTING 159: env-mlb3.tex using Listing 142 on page 35	36
LISTING 160: env-mlb3.tex using Listing 146 on page 35	36
LISTING 161: env-mlb4.tex	37
LISTING 162: env-mlb9.yaml	37
LISTING 163: env-mlb10.yaml	37
LISTING 164: env-mlb11.yaml	37
LISTING 165: env-mlb12.yaml	37
LISTING 166: env-mlb4.tex using Listing 162	37
LISTING 167: env-mlb4.tex using Listing 163	37
LISTING 168: env-mlb4.tex using Listing 164	37
LISTING 169: env-mlb4.tex using Listing 165	37



LISTING 170: <code>env-mlb5.tex</code>	38
LISTING 171: <code>removeTWS-before.yaml</code>	38
LISTING 172: <code>env-mlb5.tex</code> using Listings 166 to 169	38
LISTING 173: <code>env-mlb5.tex</code> using Listings 166 to 169 and Listing 171	38
LISTING 174: <code>env-mlb6.tex</code>	38
LISTING 175: <code>UnpreserveBlankLines.yaml</code>	38
LISTING 176: <code>env-mlb6.tex</code> using Listings 166 to 169	39
LISTING 177: <code>env-mlb6.tex</code> using Listings 166 to 169 and Listing 175	39
LISTING 190: <code>helloworld.pl</code>	44

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 T_EX stack exchange [1] 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 Section 7.2 on page 43 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 11) 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 ([6]) with a complete minimum working example as I would like to improve the code as much as possible.



Before you try the script on anything important (like your thesis), test it out on the sample files in the `test-case` directory ([6]).

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 [10]

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.

FIX



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="StrawberryPerl",
url="http://strawberryperl.com/"}
@online{cmhblog,
title="APerlscript...
url="...
}
\end{filecontents}
```

LISTING 2: filecontents after

```
\begin{filecontents}{mybib.bib}
    \@online{strawberryperl,
        \title="StrawberryPerl",
        \url="http://strawberryperl.com/"}
    \@online{cmhblog,
        \title="APerlscript...
        \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 T_EXLive distribution for Linux and Mac users; `latexindent.exe` ships as part of the T_EXLive and MiK_TE_Xdistributions for Windows users. These files are also available from github [6] should you wish to use them without a T_EX distribution; in this case, you may like to read appendix C on page 45 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 10.

`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 appendix A on page 44 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.

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

FIX

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 exists¹. 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
```

See appendix D on page 46 for details of how the interface has changed from Version 2.2 to Version 3.0 for this flag.

-s, -silent

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



```
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`

```
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>`

FIX



```
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 Section 5 on page 33

`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 the following page 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.

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

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

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 \TeX .

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 Section 6 if you like.

`fileExtensionPreference:` $\langle fields \rangle$

`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

```
fileExtensionPreference:
  .tex: 1
  .sty: 2
  .cls: 3
  .bib: 4
```

`backupExtension:` $\langle extension\ name \rangle$

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.

logFilePreferences: *<fields>*

`latexindent.pl` writes information to `indent.log`, some of which can be customised by changing `logFilePreferences`; see Listing 9. If you load your own user settings (see Section 6 on page 40) 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. 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.

LISTING 9: `logFilePreferences`

```
logFilePreferences:
  showEveryYamlRead: 1
  showAmalgamatedSettings: 0
  endLogFileWith: '-----'
  showGitHubInfoFooter: 1
```

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 10.

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`.

LISTING 10:
`verbatimEnvironments`

```
verbatimEnvironments:
  verbatim: 1
  lstlisting: 1
```

LISTING 11:
`verbatimCommands`

```
verbatimCommands:
  verb: 1
  lstinline: 1
```



`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 12.

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 13 for example.

LISTING 12:
`noIndentBlock`

```
84 noIndentBlock:
85     noindent: 1
86     cmhtest: 1
```

LISTING 13: `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 14; each of the fields can take the values 0 or 1. See Listings 171 to 173 on page 38 for before and after results. Thanks to [11] for providing this feature.

LISTING 14:
`removeTrailingWhitespace`

```
89 removeTrailingWhitespace:
90     beforeProcessing: 0
91     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 15. The behaviour of `latexindent.pl` on these environments is determined by their location (preamble or not), and the value `indentPreamble`, discussed next.

LISTING 15:
`fileContentsEnvironments`

```
95 fileContentsEnvironments:
96     filecontents: 1
97     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 16, 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.

`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 17.

LISTING 17: 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 18). In fact, the fields in `lookForAlignDelims` can actually take two different forms: the *basic* version is shown in Listing 18 and the *advanced* version in Listing 21; we will discuss each in turn.

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 19 and 20.

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 12).

LISTING 16:
`lookForPreamble`

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

LISTING 18:
`lookForAlignDelims`
(basic)

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



LISTING 19: tabular before

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

LISTING 20: 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 21 is for you.

LISTING 21: lookForAlignDelims (advanced)

```
121 lookForAlignDelims:
122   tabular:
123     delims: 1
124     alignDoubleBackSlash: 1
125     spacesBeforeDoubleBackSlash: 2
126   tabularx:
127     delims: 1
128   longtable: 1
```

Note that you can use a mixture of the basic and advanced form: in Listing 21 `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 18 (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 21 saved in `mysettings.yaml`, and the code from Listing 19 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 22 and 23; note that the ampersands have been aligned, but the `\\` have not (compare the alignment of `\\` in Listings 20 and 23).

LISTING 22: tabular before

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

LISTING 23: tabular after (advanced)

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

Using `spacesBeforeDoubleBackSlash: 3` gives Listings 24 and 25, note the spacing before the `\\` in Listing 25.

LISTING 24: tabular before

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

LISTING 25: 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 26 and 27 are achievable by default.

FIX



LISTING 26: matrix before

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

LISTING 27: 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 29 and 30

LISTING 28: indentAfterItems

```
155 indentAfterItems:
156   itemize: 1
157   enumerate: 1
158   list: 1
```

LISTING 29: 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 30: 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 Section 6 on page 40 for details of how to configure user settings, and Listing 188 on page 42 in particular.

LISTING 31:
itemNames

```
164 itemNames:
165   item: 1
166   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 32 shows the default settings of `specialBeginEnd`.

LISTING 32: specialBeginEnd

```
170 specialBeginEnd:
171   displayMath:
172     begin: '\\\[
173     end: '\\\[
174     lookForThis: 1
175   inlineMath:
176     begin: '(?!\$)(?!\$)\$(?!\$)'
177     end: '(?!\$)\$(?!\$)'
178     lookForThis: 1
179   displayMathTeX:
180     begin: '\$ \$'
181     end: '\$ \$'
182     lookForThis: 1
```



The field `displayMath` represents `\[...\]`, `inlineMath` represents `...\$` and `displayMathTex` represents `$$...$$`. You can, of course, rename these in your own YAML files (see Section 6.1 on page 42); 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 33 and 34.

LISTING 33: <code>special1.tex</code> before	LISTING 34: <code>special1.tex</code> after
<pre>The\function\ \$ f \$ \has\formula \[f(x)=x^2. \] If\you\like\splitting\dollars, \$ g(x)=f(2x) \$</pre>	<pre>The\function\ \$ f \$ \has\formula \[(x)=x^2. \] If\you\like\splitting\dollars, \$ (x)=f(2x) \$</pre>

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.²

The default settings do *not* place indentation after a heading, but you can easily switch them on by changing `indentAfterThisHeading` 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`; 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`).

For example, assuming that you have read Section 6.1 on page 42, say that you have the code in Listing 36 saved into `headings1.yaml`, and that you have the text from Listing 37 saved into `headings1.tex`.

LISTING 35: <code>indentAfterHeadings</code>
<pre>indentAfterHeadings: part: indentAfterThisHeading: 0 level: 1 chapter: indentAfterThisHeading: 0 level: 2 section: indentAfterThisHeading: 0 level: 3</pre>

FIX

²There is a slight difference in interface for this field when comparing Version 2.2 to Version 3.0; see appendix D on page 46 for details.



LISTING 36: headings1.yaml

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

LISTING 37: 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 38.

LISTING 38: headings1.tex using Listing 36

```
\subsection{subsection_title}
  \subsection_text
  \subsection_text
  \paragraph{paragraph_title}
  \paragraph_text
  \paragraph_text
  \paragraph{paragraph_title}
  \paragraph_text
  \paragraph_text
```

LISTING 39: 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 39; notice that the paragraph and subsection are at the same indentation level.

4.1.1 The code blocks known latexindent.pl

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

TABLE 2: Code blocks known to latexindent.pl

Code block	characters allowed in name	example
environments	a-zA-Z@*0-9_\\	<pre>\begin{myenv} body of myenv \end{myenv}</pre>
optionalArguments	inherits name from parent (e.g environment name)	<pre>[opt arg text]</pre>



mandatoryArguments	<i>inherits name from parent (e.g environment name)</i>	<code>{ mand arg text }</code>
commands	<code>a-zA-Z@*0-9_\:</code>	<code>\mycommand{arguments}</code>
keyEqualsValuesBraces	<code>a-zA-Z@*0-9_\/. \h\{\}:\#-</code>	<code>my key/.style={arguments}</code>
namedGroupingBracesBrackets	<code>a-zA-Z@* > <</code>	<code>in{arguments}</code>
UnNamedGroupingBracesBrackets	<i>No name!</i>	<code>{ or [or , or & followed by {arguments}</code>
ifElseFi	<code>@a-zA-Z but must begin with either \if of \@if</code>	<code>\ifnum \else ... \fi</code>
items	User specified, see Listings 28 and 31 on page 15	<code>\begin{enumerate} \item ... \end{enumerate}</code>
specialBeginEnd	User specified, see Listing 32 on page 15	<code>\[... \]</code>
afterHeading	User specified, see Listing 35 on page 16	<code>\chapter{title} ... \section{title}</code>
filecontents	User specified, see Listing 15 on page 12	<code>\begin{filecontents} ... \end{filecontents}</code>

We will refer to these code blocks in what follows.

4.2 noAdditionalIndent and indentRules

`latexindent.pl` operates on files by looking for code blocks, as detailed in Section 4.1.1 on page 17; for each type of code block in Table 2 on page 17 (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 page 17; for reference, there follows a list of the code blocks covered.

4.2.1	Environments and their arguments	19
4.2.2	Environments with items	25
4.2.3	Commands with arguments	26
4.2.4	<code>ifelsefi</code> code blocks	28
4.2.5	<code>specialBeginEnd</code> code blocks	30
4.2.6	<code>afterHeading</code> code blocks	31

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 48.

LISTING 48: `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 49 and 50.

LISTING 49:

`myenv-noAdd1.yaml`

```
noAdditionalIndent:
  myenv: 1
```

LISTING 50:

`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 51; note in particular that the environment `myenv` has not received any *additional* indentation, but that the outer environment *has* still received indentation.



LISTING 51: myenv.tex output (using either Listings 49 and 50)

```

\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 52 and 53, 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 54.

LISTING 52: myenv-noAdd3.yaml
noAdditionalIndent: myenv: 0

LISTING 53: myenv-noAdd4.yaml
noAdditionalIndent: myenv: body: 0

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}

```

Let's now allow myenv to have some optional and mandatory arguments, as in Listing 55.

LISTING 55: 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 56; 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 49), then *all* parts of the environment (body, optional and mandatory arguments) are assumed to want no additional indent.



LISTING 56: 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}

```

We may customise noAdditionalIndent for optional and mandatory arguments of the myenv environment, as shown in, for example, Listings 57 and 58.

LISTING 57: myenv-noAdd5.yaml

```

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

```

LISTING 58: 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 59 and 60. Note that in Listing 59 the text for the *optional* argument has not received any additional indentation, and that in Listing 60 the *mandatory* argument has not received any additional indentation; in both cases, the *body* has not received any additional indentation.

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}

```

LISTING 60: myenv-args.tex using Listing 58

```

\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 61 and 62.



LISTING 61:
myenv-rules1.yaml

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

LISTING 62:
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 63; note in particular that the environment myenv has not received any *additional* indentation, but that the outer environment *has* still received indentation.

LISTING 63: myenv.tex output (using either Listings 61 and 62)

```
\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 55 that contains optional and mandatory arguments. Upon using Listing 61 as in

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

we obtain the output in Listing 64; note that the body, optional argument and mandatory argument have *all* received the same customised indentation.

LISTING 64: myenv-args.tex using Listing 61

```
\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 65 and 66

LISTING 65: myenv-rules3.yaml

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

LISTING 66: 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 67 and 68.

LISTING 67: myenv-args.tex using Listing 65

```
\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 68: myenv-args.tex using Listing 66

```
\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 67, 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 68, 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 69). Let's say that you change the value of `environments` to 1 in Listing 69, and that you run

LISTING 69:
env-noAdditionalGlobal.yaml

```
noAdditionalIndentGlobal:
  environments: 0
```

```
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 70 and 71; in Listing 70 notice that *both* environments receive no additional indentation but that the arguments of `myenv` still *do* receive indentation. In Listing 71 notice that the *outer* environment does not receive additional indentation, but because of the settings from `myenv-rules1.yaml` (in Listing 61 on page 22), the `myenv` environment still *does* receive indentation.



LISTING 70: myenv-args.tex using Listing 69

```
\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 71: myenv-args.tex using Listings 61 and 69

```
\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 72 and 73

LISTING 72:

opt-args-no-add-glob.yaml

```
noAdditionalIndentGlobal:
  optionalArguments: 1
```

LISTING 73:

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 74 and 75. Notice that in Listing 74 the *optional* argument has not received any additional indentation, and in Listing 75 the *mandatory* argument has not received any additional indentation.

LISTING 74: myenv-args.tex using Listing 72

```
\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 75: myenv-args.tex using Listing 73

```
\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 76; if you change the environments field to anything involving horizontal space, say " ", and then run the following commands

LISTING 76:

env-indentRulesGlobal.yaml

```
indentRulesGlobal:
  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 77 and 78. Note that in Listing 77, both the environment blocks have received a single-space indentation, whereas in Listing 78 the outer environment has received single-space indentation (specified by `indentRulesGlobal`), but `myenv` has received " " , as specified by the particular `indentRules` for `myenv` Listing 61 on page 22.

LISTING 77: `myenv-args.tex` using Listing 76

```
\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 78: `myenv-args.tex` using Listings 61 and 76

```
\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 79 and 80

LISTING 79:
`opt-args-indent-rules-glob.yaml`

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

LISTING 80:
`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 81 and 82. Note that the *optional* argument in Listing 81 has received two tabs worth of indentation, while the *mandatory* argument has done so in Listing 82.

LISTING 81: `myenv-args.tex` using Listing 79

```
\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 82: `myenv-args.tex` using Listing 80

```
\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 28 and 31 on page 15, some commands may contain `item` commands; for the purposes of this discussion, we will use the code from Listing 29 on page 15.

Assuming that you've populated `itemNames` with the name of your `item`, you can put the item name into `noAdditionalIndent` as in Listing 83, 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 84



LISTING 83: item-noAdd1.yaml

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

LISTING 84: 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 85 and 86; note that in Listing 85 that the text after each item has not received any additional indentation, and in Listing 86, the text after each item has received a single space of indentation, specified by Listing 84.

LISTING 85: items1.tex using Listing 83

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

LISTING 86: items1.tex using Listing 84

```
\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 87 and 88. Note that there is a need to ‘reset/remove’ the `item` field from `indentRules` in both cases (see the hierarchy description given on page 18) as the `item` command is a member of `indentRules` by default.

LISTING 87:

items-noAdditionalGlobal.yaml

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

LISTING 88:

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 85 and 86 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 89; when `latexindent.pl` operates on this file, the default output is shown in Listing 90.



LISTING 89: mycommand.tex

```
\mycommand
{
  mand_arg_text
  mand_arg_text}
[
  opt_arg_text
  opt_arg_text
]
```

LISTING 90: 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 49 and 50 on page 19) we may specify `noAdditionalIndent` either in ‘scalar’ form, or in ‘field’ form, as shown in Listings 91 and 92

LISTING 91:

mycommand-noAdd1.yaml

```
noAdditionalIndent:
  mycommand: 1
```

LISTING 92:

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 93 and 94

LISTING 93: mycommand.tex using Listing 91

```
\mycommand
{
  mand_arg_text
  mand_arg_text}
[
  opt_arg_text
  opt_arg_text
]
```

LISTING 94: mycommand.tex using Listing 92

```
\mycommand
{
  \mand_arg_text
  \mand_arg_text}
[
  \opt_arg_text
  \opt_arg_text
]
```

Note that in Listing 93 that the ‘body’, optional argument *and* mandatory argument have *all* received no additional indentation, while in Listing 94, 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 57 and 58 on page 21; explicit examples are given in Listings 95 and 96.

LISTING 95:

mycommand-noAdd3.yaml

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

LISTING 96:

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 97 and 98.

LISTING 97: mycommand.tex using Listing 95

```
\mycommand
{
  \mand_arg_text
  \mand_arg_text}
[
opt_arg_text
opt_arg_text
]
```

LISTING 98: mycommand.tex using Listing 96

```
\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 97 and 98 has received no additional indent, even though body is explicitly set to 0 in both Listings 95 and 96. This is because, by default, noAdditionalIndentGlobal for commands is set to 1 by default; this can be easily fixed as in Listings 99 and 100.

LISTING 99:
mycommand-noAdd5.yaml

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

LISTING 100:
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 101 and 102.

LISTING 101: mycommand.tex using Listing 99

```
\mycommand
\{
\  \mand_arg_text
\  \mand_arg_text}
\[[
\opt_arg_text
\opt_arg_text
\]
```

LISTING 102: mycommand.tex using Listing 100

```
\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 65 and 66 on page 22 and Listings 76, 79 and 80 on page 24 and on page 25.

4.2.4 ifelsefi code blocks

Let's use the simple example shown in Listing 103; when latexindent.pl operates on this file, the output as in Listing 104; note that the body of each of the \if statements have been indented, and that the \else statement has been accounted for correctly.



LISTING 103: ifelsefi1.tex

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

LISTING 104: 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 105 and 106.

LISTING 105:

ifnum-noAdd.yaml

```
noAdditionalIndent:
  ifnum: 1
```

LISTING 106:

ifnum-indent-rules.yaml

```
indentRules:
  ifnum: "\t\t"
```

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 107 and 108; note that in Listing 107, the `ifnum` code block has not received any additional indentation, while in Listing 108, the `ifnum` code block has received three tabs worth of indentation.

LISTING 107: ifelsefi1.tex using Listing 105

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

LISTING 108: ifelsefi1.tex using Listing 106

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

We may specify `noAdditionalIndentGlobal` and `indentRulesGlobal` as in Listings 109 and 110.

LISTING 109:

ifelsefi-noAdd-glob.yaml

```
noAdditionalIndentGlobal:
  ifElseFi: 1
```

LISTING 110:

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 111 and 112; notice that in Listing 111 neither of the `ifelsefi` code blocks have received indentation, while in Listing 112 both code blocks have received a single space of indentation.



LISTING 111: ifelsefi1.tex using Listing 109

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

LISTING 112: ifelsefi1.tex using Listing 110

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

4.2.5 specialBeginEnd code blocks

Let's use the example from Listing 33 on page 16 which has default output shown in Listing 34 on page 16.

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 113 and 114.

LISTING 113:
displayMath-noAdd.yaml

```
noAdditionalIndent:
  displayMath: 1
```

LISTING 114:
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 115 and 116; note that in Listing 115, the `displayMath` code block has *not* received any additional indentation, while in Listing 116, the `displayMath` code block has received three tabs worth of indentation.

LISTING 115: special1.tex using Listing 113

```
The_\function_\$ f \$ _has_\formula
\[
f(x)=x^2.
\]
If_\you_\like_\splitting_\dollars,
$
(x)=f(2x)
$
```

LISTING 116: special1.tex using Listing 114

```
The_\function_\$ f \$ _has_\formula
\[
(x)=x^2.
\]
If_\you_\like_\splitting_\dollars,
$
(x)=f(2x)
$
```

We may specify `noAdditionalIndentGlobal` and `indentRulesGlobal` as in Listings 117 and 118.

LISTING 117:
special-noAdd-glob.yaml

```
noAdditionalIndentGlobal:
  specialBeginEnd: 1
```

LISTING 118:
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 119 and 120; notice that in Listing 119 neither of the special code blocks have received indentation, while in Listing 120 both code blocks have received a single space of indentation.

LISTING 119: special1.tex using Listing 117

```
The_function $ f $ _has_formula
\[
f(x)=x^2.
\]
If_you_like_splitting_dollars,
$
g(x)=f(2x)
$
```

LISTING 120: special1.tex using Listing 118

```
The_function $ f $ _has_formula
\[
 f(x)=x^2.
\]
If_you_like_splitting_dollars,
 $
 g(x)=f(2x)
 $
```

4.2.6 afterHeading code blocks

Let's use the example Listing 121 for demonstration throughout this Section. As discussed on page 17, by default latexindent.pl will not add indentation after headings.

LISTING 121: headings2.tex

```
\paragraph{paragraph
title}
paragraph_text
paragraph_text
```

On using the YAML file in Listing 123 by running the command

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

we obtain the output in Listing 122. Note that the argument of paragraph has received (default) indentation, and that the body after the heading statement has received (default) indentation.

LISTING 122: headings2.tex using Listing 123

```
\paragraph{paragraph
  ¶ title}
  ¶paragraph_text
  ¶paragraph_text
```

LISTING 123: headings3.yaml

```
indentAfterHeadings:
  paragraph:
    indentAfterThisHeading: 1
    level: 1
```

If we specify noAdditionalIndent as in Listing 125 and run the command

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

then we receive the output in Listing 124. 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 124: headings2.tex using Listing 125

```
\paragraph{paragraph
title}
paragraph_text
paragraph_text
```

LISTING 125: headings4.yaml

```
indentAfterHeadings:
  paragraph:
    indentAfterThisHeading: 1
    level: 1
noAdditionalIndent:
  paragraph: 1
```




LISTING 134: headings2.tex using Listing 135

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

LISTING 135: headings9.yaml

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

5 The -m (modifylinebreaks) switch

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

`modifylinebreaks: {fields}`

One of the most exciting features of Version 3.0 is 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 136.

LISTING 136: modifyLineBreaks

```
modifyLineBreaks:
  preserveBlankLines: 1
  condenseMultipleBlankLinesInto: 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:



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.

All YAML-based details in this section only apply if the -m switch is active.

`preserveBlankLines: 0|1`

This field is directly related to *poly-switches*, discussed below. 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: {integer ≥ 0}`

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. As an example, Listing 137 shows a sample file with blank lines; upon running

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

the output is shown in Listing 138; note that the multiple blank lines have been condensed into one blank line, and note also that we have used the -m switch!



LISTING 137: mlb1.tex

```
before_blank_line

after_blank_line

after_blank_line
```

LISTING 138: mlb1.tex out output

```
before_blank_line

after_blank_line

after_blank_line
```

5.1 Poly-switches

Every other field in the `modifyLineBreaks` field uses poly-switch, and can take one of four integer values³:

- 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>*.

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.

5.2 modifyLineBreaks for environments

We start by viewing a snippet of `defaultSettings.yaml` in Listing 139; note that it contains *global* settings (immediately after the `environments` field) and that *per-name* settings are also allowed – in the case of Listing 139, settings for `equation*` have been specified. Note that all poly-switches are *off* by default.

LISTING 139: environments

```
348 environments:
349     BeginStartsOnOwnLine: 0
350     BodyStartsOnOwnLine: 0
351     EndStartsOnOwnLine: 0
352     EndFinishesWithLineBreak: 0
353 equation*:
354     BeginStartsOnOwnLine: 0
355     BodyStartsOnOwnLine: 0
356     EndStartsOnOwnLine: 0
357     EndFinishesWithLineBreak: 0
```

5.2.1 Adding line breaks (poly-switches set to 1 or 2)

Let's begin with the simple example given in Listing 140; note that we have annotated key parts of the file using ♠, ♥, ♦ and ♣, these will be related to fields specified in Listing 139.

LISTING 140: env-mlb1.tex

```
before words ♠ \begin{myenv} ♥ body of myenv ♦ \end{myenv} ♣ after words
```

Let's explore `BeginStartsOnOwnLine` and `BodyStartsOnOwnLine` in Listings 141 and 142, and in particular, let's allow each of them in turn to take a value of 1.

³visual learners might like to associate one of the four circles in the logo with one of the four given values



LISTING 141: env-mlb1.yaml

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

LISTING 142: env-mlb2.yaml

```
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 143 and 144.

LISTING 143: env-mlb.tex using Listing 141

```
before_words
\begin{myenv}body_of_myenv\end{myenv}after_words
```

LISTING 144: env-mlb.tex using Listing 142

```
before_words \begin{myenv}
  body_of_myenv\end{myenv}after_words
```

There are a couple of points to note:

- in Listing 143 a line break has been added at the point denoted by ♠ in Listing 140; no other line breaks have been changed;
- in Listing 144 a line break has been added at the point denoted by ♥ in Listing 140; furthermore, note that the *body* of *myenv* has received the appropriate (default) indentation.

Let's now change each of the 1 values in Listings 141 and 142 so that they are 2 and save them into env-mlb3.yaml and env-mlb4.yaml respectively (see Listings 145 and 146).

LISTING 145: env-mlb3.yaml

```
modifyLineBreaks:
  environments:
    BeginStartsOnOwnLine: 2
```

LISTING 146: env-mlb4.yaml

```
modifyLineBreaks:
  environments:
    BodyStartsOnOwnLine: 2
```

Upon running commands analogous to the above, we obtain Listings 147 and 148.

LISTING 147: env-mlb.tex using Listing 145

```
before_words%
\begin{myenv}body_of_myenv\end{myenv}after_words
```

LISTING 148: env-mlb.tex using Listing 146

```
before_words \begin{myenv}%
  body_of_myenv\end{myenv}after_words
```

Note that line breaks have been added as in Listings 143 and 144, 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.

Let's explore *EndStartsOnOwnLine* and *EndFinishesWithLineBreak* in Listings 149 and 150, and in particular, let's allow each of them in turn to take a value of 1.

LISTING 149: env-mlb5.yaml

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

LISTING 150: env-mlb6.yaml

```
modifyLineBreaks:
  environments:
    EndFinishesWithLineBreak: 1
```

After running the following commands,

```
cmh:~$ latexindent.pl -m env-mlb.tex -l env-mlb5.yaml
cmh:~$ latexindent.pl -m env-mlb.tex -l env-mlb6.yaml
```

the output is as in Listings 151 and 152.



LISTING 151: env-mlb.tex using Listing 149

```
before_words\begin{myenv}body_of_myenv
\end{myenv}after_words
```

LISTING 152: env-mlb.tex using Listing 150

```
before_words\begin{myenv}body_of_myenv\end{myenv}
after_words
```

There are a couple of points to note:

- in Listing 151 a line break has been added at the point denoted by ♦ in Listing 140 on page 34; no other line breaks have been changed and the `\end{myenv}` statement has *not* received indentation (as intended);
- in Listing 152 a line break has been added at the point denoted by ♣ in Listing 140 on page 34.

Let's now change each of the 1 values in Listings 149 and 150 so that they are 2 and save them into `env-mlb7.yaml` and `env-mlb8.yaml` respectively (see Listings 153 and 154).

LISTING 153: env-mlb7.yaml

```
modifyLineBreaks:
  environments:
    EndStartsOnOwnLine: 2
```

LISTING 154: env-mlb8.yaml

```
modifyLineBreaks:
  environments:
    EndFinishesWithLineBreak: 2
```

Upon running commands analogous to the above, we obtain Listings 155 and 156.

LISTING 155: env-mlb.tex using Listing 153

```
before_words\begin{myenv}body_of_myenv%
\end{myenv}after_words
```

LISTING 156: env-mlb.tex using Listing 154

```
before_words\begin{myenv}body_of_myenv\end{myenv}%
after_words
```

Note that line breaks have been added as in Listings 151 and 152, 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.

If you ask `latexindent.pl` to add a line break (possibly with a comment) using a poly-switch value of 1 (or 2), it will only do so if necessary. For example, if you process the file in Listing 142 on page 35 using any of the YAML files presented so far in this section, it will be left unchanged.

LISTING 157: env-mlb2.tex

```
before_words
\begin{myenv}
body_of_myenv
\end{myenv}
after_words
```

LISTING 158: env-mlb3.tex

```
before_words
\begin{myenv}%
body_of_myenv%
\end{myenv}%
after_words
```

In contrast, the output from processing the file in Listing 158 will vary depending on the poly-switches used; in Listing 159 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 160 you'll see that the comment has been accounted for correctly, and that, because `BodyStartsOnOwnLine` has been set to 2, the comment symbol has *not* been moved to its own line. You're encouraged to experiment with Listing 158 and by setting the other poly-switches considered so far to 2 in turn.

LISTING 159: env-mlb3.tex using Listing 142 on page 35

```
before_words
\begin{myenv}
%
body_of_myenv%
\end{myenv}%
after_words
```

LISTING 160: env-mlb3.tex using Listing 146 on page 35

```
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 139 on page 34, an example is shown for the `equation*` environment.



5.2.2 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. We will consider the example code given in Listing 161, noting in particular the positions of the line break highlighters, ♠, ♥, ♦ and ♣, together with the associated YAML files in Listings 162 to 165.

LISTING 161: env-mlb4.tex

```
before words ♠
\begin{myenv} ♥
body of myenv ♦
\end{myenv} ♣
after words
```

LISTING 162: env-mlb9.yaml

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

LISTING 163: env-mlb10.yaml

```
modifyLineBreaks:
  environments:
    BodyStartsOnOwnLine: -1
```

LISTING 164: env-mlb11.yaml

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

LISTING 165: env-mlb12.yaml

```
modifyLineBreaks:
  environments:
    EndFinishesWithLineBreak: -1
```

After running the commands

```
cmh:~$ latexindent.pl -m env-mlb4.tex -l env-mlb9.yaml
cmh:~$ latexindent.pl -m env-mlb4.tex -l env-mlb10.yaml
cmh:~$ latexindent.pl -m env-mlb4.tex -l env-mlb11.yaml
cmh:~$ latexindent.pl -m env-mlb4.tex -l env-mlb12.yaml
```

we obtain the output in Listings 166 to 169.

LISTING 166: env-mlb4.tex using Listing 162

```
before_words\begin{myenv}
  #body_of_myenv
\end{myenv}
after_words
```

LISTING 167: env-mlb4.tex using Listing 163

```
before_words
\begin{myenv}body_of_myenv
\end{myenv}
after_words
```

LISTING 168: env-mlb4.tex using Listing 164

```
before_words
\begin{myenv}
  #body_of_myenv\end{myenv}
after_words
```

LISTING 169: env-mlb4.tex using Listing 165

```
before_words
\begin{myenv}
  #body_of_myenv
\end{myenv}after_words
```

Notice that in

- Listing 166 the line break denoted by ♠ has been removed;
- Listing 167 the line break denoted by ♥ has been removed;
- Listing 168 the line break denoted by ♦ has been removed;
- Listing 169 the line break denoted by ♣ 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 162 to 165 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-mlb9.yaml,env-mlb10.yaml,env-mlb11.yaml,env-mlb12.yaml
```

which gives the output in Listing 140 on page 34.

About trailing horizontal space Recall that on page 12 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; let's consider the file shown in Listing 170, which highlights trailing spaces.

LISTING 170: env-mlb5.tex

```
before_words   ♠
\begin{myenv}  ♥
body_of_myenv  ♦
\end{myenv}    ♣
after_words
```

LISTING 171:

removeTWS-before.yaml

```
removeTrailingWhitespace:
  beforeProcessing: 1
```

The output from the following commands

```
cmh:~$ latexindent.pl -m env-mlb5.tex -l env-mlb9.yaml,env-mlb10.yaml,env-mlb11.yaml,env-mlb12.yaml
cmh:~$ latexindent.pl -m env-mlb5.tex -l
      env-mlb9.yaml,env-mlb10.yaml,env-mlb11.yaml,env-mlb12.yaml,removeTWS-before.yaml
```

is shown, respectively, in Listings 172 and 173; note that the trailing horizontal white space has been preserved (by default) in Listing 172, while in Listing 173, it has been removed using the switch specified in Listing 171.

LISTING 172: env-mlb5.tex using Listings 166 to 169

```
before_words   \begin{myenv}body_of_myenv\end{myenv}after_words
```

LISTING 173: env-mlb5.tex using Listings 166 to 169 and Listing 171

```
before_words\begin{myenv}body_of_myenv\end{myenv}after_words
```

Blank lines Now let's consider the file in Listing 174, which contains blank lines.

LISTING 174: env-mlb6.tex

```
before words ♠
```

```
\begin{myenv} ♥
```

```
body of myenv ♦
```

```
\end{myenv} ♣
```

```
after words
```

LISTING 175:

UnpreserveBlankLines.yaml

```
modifyLineBreaks:
  preserveBlankLines: 0
```

Upon running the following commands



```
cmh:~$ latexindent.pl -m env-mlb6.tex -l env-mlb9.yaml,env-mlb10.yaml,env-mlb11.yaml,env-mlb12.yaml
cmh:~$ latexindent.pl -m env-mlb6.tex -l
env-mlb9.yaml,env-mlb10.yaml,env-mlb11.yaml,env-mlb12.yaml,UnpreserveBlankLines.yaml
```

we receive the respective outputs in Listings 176 and 177. In Listing 176 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 177, we have allowed the poly-switches to remove blank lines because, in Listing 175, we have set `preserveBlankLines` to 0.

LISTING 176:

env-mlb6.tex using
Listings 166 to 169

before_␣words

`\begin{myenv}`

`␣body␣of␣myenv`

`\end{myenv}`

after_␣words

LISTING 177: env-mlb6.tex using Listings 166 to 169 and
Listing 175

before_␣words`\begin{myenv}`body_␣of_␣myenv`\end{myenv}`after_␣words

5.3 Poly-switches for other code blocks

Rather than repeat the examples shown for the environment code blocks (in Section 5.2 on page 34), 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*.

TABLE 3: Poly-switch mappings for all code-block types

Code block	Sample	Poly-switch mapping
environment	before words♠ <code>\begin{myenv}</code> ♥ body of myenv◇ <code>\end{myenv}</code> ♣ after words	♠ BeginStartsOnOwnLine ♥ BodyStartsOnOwnLine ◇ EndStartsOnOwnLine ♣ EndFinishesWithLineBreak
ifelsefi	before words♠ <code>\if...</code> ♥ body of if statement★ <code>\else</code> □ body of else statement◇ <code>\fi</code> ♣ after words	♠ IfStartsOnOwnLine ♥ BodyStartsOnOwnLine ★ ElseStartsOnOwnLine □ ElseFinishesWithLineBreak ◇ FiStartsOnOwnLine ♣ FiFinishesWithLineBreak



optionalArguments	<pre>...♠ [♥ body of opt arg◇]♣ ...</pre>	<ul style="list-style-type: none"> ♠ LSqBStartsOnOwnLine⁴ ♥ OptArgBodyStartsOnOwnLine ◇ RSqBStartsOnOwnLine ♣ RSqBFinishesWithLineBreak
mandatoryArguments	<pre>...♠ {♥ body of mand arg◇ }♣ ...</pre>	<ul style="list-style-type: none"> ♠ LCuBStartsOnOwnLine⁵ ♥ MandArgBodyStartsOnOwnLine ◇ RCuBStartsOnOwnLine ♣ RCuBFinishesWithLineBreak
commands	<pre>before words♠ \mycommand♥ ⟨arguments⟩</pre>	<ul style="list-style-type: none"> ♠ CommandStartsOnOwnLine ♥ CommandNameFinishesWithLineBreak
namedGroupingBraces Brackets	<pre>before words♠ myname♥ ⟨braces/brackets⟩</pre>	<ul style="list-style-type: none"> ♠ NameStartsOnOwnLine ♥ NameFinishesWithLineBreak
keyEqualsValuesBraces	<pre>before words♠ key•=♥ ⟨braces/brackets⟩</pre>	<ul style="list-style-type: none"> ♠ KeyStartsOnOwnLine • EqualsStartsOnOwnLine ♥ EqualsFinishesWithLineBreak
items	<pre>before words♠ \item♥ ...</pre>	<ul style="list-style-type: none"> ♠ ItemStartsOnOwnLine ♥ ItemFinishesWithLineBreak
specialBeginEnd	<pre>before words♠ \[♥ body of special◇ \]♣ after words</pre>	<ul style="list-style-type: none"> ♠ SpecialBeginStartsOnOwnLine ♥ SpecialBodyStartsOnOwnLine ◇ SpecialEndStartsOnOwnLine ♣ SpecialEndFinishesWithLineBreak

6 indentconfig.yaml and .indentconfig.yaml (for user settings)

Editing defaultSettings.yaml is not ideal as it may be overwritten when updating your distribution—a better way to customize the settings to your liking is to set up your own settings file, mysettings.yaml

⁴LSqB stands for Left Square Bracket

⁵LCuB stands for Left Curly Brace



(or any name you like, provided it ends with `.yaml`). The only thing you have to do is tell `latexindent.pl` where to find it.

`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 equally represents `.indentconfig.yaml` as well. If you have both files in existence, `indentconfig.yaml` takes priority.

For Mac and Linux users, their home directory is `/username` while Windows (Vista onwards) is `C:\Users\username` ⁶ Listing 187 shows a sample `indentconfig.yaml` file.

LISTING 187: `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 that you write them in. 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 188 for an example that uses four tabs for the default indent, adds the `tabbing` environment to the list of environments that contains alignment delimiters, and adds the changes we described on page 15; you might also like to refer to the many YAML files detailed throughout the rest of this documentation.

⁶If 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 188: mysettings.yaml (example)

```
# Default value of indentation
defaultIndent: "\t\t\t\t"

# environments that have tab delimiters, add more
# as needed
lookForAlignDelims:
  tabbing: 1

# If you use the exam documentclass, you might
# like the following settings
# environments that have \item commands
indentAfterItems:
  parts: 1

# commands to be treated like \item
itemNames:
  part: 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 recognize then you'll get a warning, otherwise you'll get confirmation that `latexindent.pl` has read your settings file ⁷.



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`.

6.1 localSettings.yaml

Throughout this manual, we have discussed the `-l` switch that tells `latexindent.pl` either to look for `localSettings.yaml` in the *same directory* as `myfile.tex`; alternatively, it may look for any other specified YAML file. Any settings file(s) specified in this way will be read *after* `defaultSettings.yaml` and, assuming they exist, user settings from `indentconfig.yaml`.

The *local* settings file may be called `localSettings.yaml`, and it can contain any switches that you'd like to change; a sample is shown in Listing 189.

LISTING 189: localSettings.yaml (example)

```
# Default value of indentation
defaultIndent: " "

# environments that have tab delimiters, add more
# as needed
lookForAlignDelims:
  tabbing: 0

# verbatim environments- environments specified
# in this hash table will not be changed at all!
verbatimEnvironments:
  cmhenvironment: 0
```

You can make sure that your local settings are loaded by checking `indent.log` for details; if `localSettings.yaml` can not be read then you will get a warning, otherwise you'll get confirmation that `latexindent.pl` has read `localSettings.yaml`.

FIX

⁷Windows users may find that they have to end `.yaml` files with a blank line

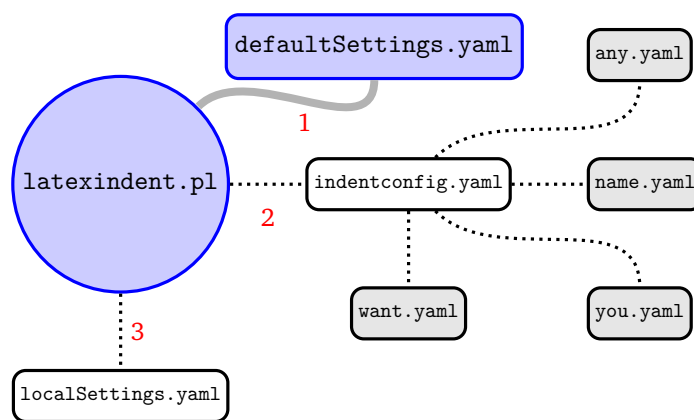


FIGURE 1: Schematic of the load order described in Section 6.2; solid lines represent mandatory files, dotted lines represent optional files. `indentconfig.yaml` can contain as many files as you like. The files will be loaded in order; if you specify settings for the same field in more than one file, the most recent takes priority.

If you'd prefer to name your `localSettings.yaml` file something different, (say, `myyaml.yaml`) then you can call `latexindent.pl` using, for example,

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

6.2 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 6.1). You may specify relative paths to other YAML files using the `-l` switch, separating files using commas.

FIX

A visual representation of this is given in Figure 1.

7 References

7.1 External links

- [1] 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).
- [3] CPAN: Comprehensive Perl Archive Network. URL: <http://www.cpan.org/> (visited on 01/23/2017).
- [6] Home of `latexindent.pl`. URL: <https://github.com/cmhughes/latexindent.pl> (visited on 01/23/2017).
- [8] Perlbrew. URL: <http://perlbrew.pl/> (visited on 01/23/2017).
- [9] Strawberry Perl. URL: <http://strawberryperl.com/> (visited on 01/23/2017).
- [10] Video demonstration of `latexindent.pl` on youtube. URL: http://www.youtube.com/watch?v=s_AMmNVg5WM (visited on 01/23/2017).

7.2 Contributors

- [2] Paulo Cereda. *arara rule, indent.yaml*. May 23, 2013. URL: <https://github.com/cereda/arara/blob/master/rules/indent.yaml> (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).
- [7] Jason Juang. *add in PATH installation*. Nov. 24, 2015. URL: <https://github.com/cmhughes/latexindent.pl/pull/38> (visited on 01/23/2017).
- [11] Michel Voßkuhle. *Remove trailing white space*. Nov. 10, 2013. URL: <https://github.com/cmhughes/latexindent.pl/pull/12> (visited on 01/23/2017).

A Required Perl modules

If you intend to use `latexindent.pl` and *not* one of the supplied standalone executable files, then you will need a few standard Perl modules—if you can run the minimum code in Listing 190 (`perl helloworld.pl`) then you will be able to run `latexindent.pl`, otherwise you may need to install the missing modules.

LISTING 190: `helloworld.pl`

```
#!/usr/bin/perl

use strict;
use warnings;
use FindBin;
use YAML::Tiny;
use File::Copy;
use File::Basename;
use Getopt::Long;
use File::HomeDir;

print "hello_world";
exit;
```

My default installation on Ubuntu 12.04 did *not* come with all of these modules as standard, but Strawberry Perl for Windows [9] did.

Installing the modules given in Listing 190 will vary depending on your operating system and Perl distribution. For example, Ubuntu users might visit the software center, or else run

```
cmh:~$ sudo perl -MCPAN -e 'install "File::HomeDir"'
```

Linux users may be interested in exploring Perlbrew [8]; possible installation and setup options follow for Ubuntu (other distributions will need slightly different commands).

```
cmh:~$ sudo apt-get install perlbrew
cmh:~$ perlbrew install perl-5.20.1
cmh:~$ perlbrew switch perl-5.20.1
cmh:~$ sudo apt-get install curl
cmh:~$ curl -L http://cpanmin.us | perl - App::cpanminus
cmh:~$ cpanm YAML::Tiny
cmh:~$ cpanm File::HomeDir
```

Strawberry Perl users on Windows might use `CPAN client`. All of the modules are readily available on CPAN [3].

`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
```

B The arara rule

The arara rule (`indent.yaml`) contains lines such as those given in Listing 191. With this setup, the user *always* has to specify whether or not they want (in this example) to use the trace identifier.

LISTING 191: The arara rule

```
...
arguments:
- identifier: trace
flag: <arara> @{ isTrue( parameters.trace, "-t" ) }
...
```

If you would like to have the trace option on by default every time you call `latexindent.pl` from arara (without having to write `% arara: indent: {trace: yes}`), then simply amend Listing 191 so that it looks like Listing 192.

LISTING 192: The arara rule (modified)

```
...
arguments:
- identifier: trace
flag: <arara> @{ isTrue( parameters.trace, "-t" ) }
default: "-t"
...
```

With this modification in place, you now simply to write `% arara: indent` and trace mode will be activated by default. If you wish to turn off trace mode then you can write `% arara: indent: {trace: off}`.

Of course, you can apply these types of modifications to *any* of the identifiers, but proceed with caution if you intend to do this for overwrite.

C Updating the path variable

`latexindent.pl` ships with a few scripts that can update the path variables ⁸. If you're on a Linux or Mac machine, then you'll want `CMakeLists.txt` from [6].

C.1 Add to path for Linux

To add `latexindent.pl` to the path for Linux, follow these steps:

1. download `latexindent.pl`, `defaultSettings.yaml` and its associated modules, to your chosen directory from [6] ;
2. within your directory, create a directory called `path-helper-files` and download `CMakeLists.txt` and `cmake_uninstall.cmake.in` from [6]/`path-helper-files` to this directory;
3. run

```
cmh:~$ ls /usr/local/bin
```

to see what is *currently* in there;

⁸Thanks to [7] for this feature!



- run the following commands

```
cmh:~$ sudo apt-get install cmake
cmh:~$ sudo apt-get update && sudo apt-get install build-essential
cmh:~$ mkdir build && cd build
cmh:~$ cmake ../path-helper-files
cmh:~$ sudo make install
```

- run

```
cmh:~$ ls /usr/local/bin
```

again to check that `latexindent.pl` and `defaultSettings.yaml` have been added.

To remove the files, run

```
cmh:~$ sudo make uninstall}.
```

C.2 Add to path for Windows

To add `latexindent.exe` to the path for Windows, follow these steps:

- download `latexindent.exe`, `defaultSettings.yaml`, `add-to-path.bat` from [6] to your chosen directory;
- open a command prompt and run to see what is *currently* in your `%path%` variable;

```
C:\Users\cmh>echo %path%
```

- right click on `add-to-path.bat` and *Run as administrator*;
- log out, and log back in;
- 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.

D Differences from Version 2.2 to 3.0

There are a few (small) changes to the interface when comparing Version 2.2 to Version 3.0. Explicitly, in previous versions you might have run, for example,

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

whereas in Version 3.0 you would run

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



noting that the *output* file is given *next to* the `-o` switch.

The fields given in Listing 193 are *obsolete* from Version 3.0 onwards.

LISTING 193: Obsolete YAML fields from Version 3.0

```
alwaysLookforSplitBrackets
alwaysLookforSplitBrackets
checkunmatched
checkunmatchedELSE
checkunmatchedbracket
constructIfElseFi
```

There is a slight difference when specifying indentation after headings; specifically, we now write `indentAfterThisHeading` instead of `indent`. See Listings 194 and 195

LISTING 194:
`indentAfterThisHeading` in Version 2.2

```
indentAfterHeadings:
  part:
    indent: 0
    level: 1
```

LISTING 195:
`indentAfterThisHeading` in Version 3.0

```
indentAfterHeadings:
  part:
    indentAfterThisHeading: 0
    level: 1
```

To specify `noAdditionalIndent` for `display-math` environments in Version 2.2, you would write YAML as in Listing 196; as of Version 3.0, you would write YAML as in Listing 197 or, if you're using `-m` switch, Listing 198.

LISTING 196: `noAdditionalIndent` in Version 2.2

```
noAdditionalIndent:
  \[: 0
  \]: 0
```

LISTING 197: `noAdditionalIndent` for `displayMath` in Version 3.0

```
specialBeginEnd:
  displayMath:
    begin: '\\\['
    end: '\\\]'
    lookForThis: 0
```

LISTING 198: `noAdditionalIndent` for `displayMath` in Version 3.0

```
noAdditionalIndent:
  displayMath: 1
```