

latexindent.pl

Version 3.1



Chris Hughes *

May 13, 2017

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

Contents

1	Introduction	4
1.1	Thanks	4
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	User, local settings, <code>indentconfig.yaml</code> and <code>.indentconfig.yaml</code>	10
4.1	<code>localSettings.yaml</code>	11
4.2	Settings load order	11
5	<code>defaultSettings.yaml</code>	11
5.1	The code blocks known <code>latexindent.pl</code>	19
5.2	<code>noAdditionalIndent</code> and <code>indentRules</code>	21
5.2.1	Environments and their arguments	21
5.2.2	Environments with items	28
5.2.3	Commands with arguments	29
5.2.4	<code>ifelsefi</code> code blocks	31
5.2.5	<code>specialBeginEnd</code> code blocks	32
5.2.6	<code>afterHeading</code> code blocks	33
5.2.7	The remaining code blocks	35
5.2.8	Summary	37
5.3	Commands and the strings between their arguments	37
6	The <code>-m (modifylinebreaks)</code> switch	40
6.1	Poly-switches	48
6.2	<code>modifyLineBreaks</code> for environments	48
6.2.1	Adding line breaks (poly-switches set to 1 or 2)	48

*and contributors! See Section 8.2 on page 59. For all communication, please visit [6].



6.2.2 Removing line breaks (poly-switches set to <code>-1</code>)	51
6.3 Poly-switches for other code blocks	53
6.4 Partnering <code>BodyStartsOnOwnLine</code> with argument-based poly-switches	55
6.5 Conflicting poly-switches: sequential code blocks	56
6.6 Conflicting poly-switches: nested code blocks	57
7 Conclusions and known limitations	58
8 References	59
8.1 External links	59
8.2 Contributors	59
A Required Perl modules	59
B Updating the path variable	60
B.1 Add to path for Linux	61
B.2 Add to path for Windows	61
C Differences from Version 2.2 to 3.0	62

Listings

LISTING 1: <code>filecontents1.tex</code>	5	LISTING 35: <code>items1.tex</code> default output	17
LISTING 2: <code>filecontents1.tex</code> default output	5	LISTING 36: <code>itemNames</code>	17
LISTING 3: <code>tikzset.tex</code>	6	LISTING 37: <code>specialBeginEnd</code>	18
LISTING 4: <code>tikzset.tex</code> default output	6	LISTING 38: <code>special1.tex</code> before	18
LISTING 5: <code>pstricks.tex</code>	6	LISTING 39: <code>special1.tex</code> default output	18
LISTING 6: <code>pstricks.tex</code> default output	6	LISTING 40: <code>indentAfterHeadings</code>	18
LISTING 7: <code>arara</code> sample usage	9	LISTING 41: <code>headings1.yaml</code>	19
LISTING 11: <code>fileExtensionPreference</code>	12	LISTING 42: <code>headings1.tex</code>	19
LISTING 12: <code>logFilePreferences</code>	13	LISTING 43: <code>headings1.tex</code> using Listing 41	19
LISTING 13: <code>verbatimEnvironments</code>	13	LISTING 44: <code>headings1.tex</code> second modification	19
LISTING 14: <code>verbatimCommands</code>	13	LISTING 53: <code>myenv.tex</code>	21
LISTING 15: <code>noIndentBlock</code>	14	LISTING 54: <code>myenv-noAdd1.yaml</code>	22
LISTING 16: <code>noIndentBlock</code> demonstration	14	LISTING 55: <code>myenv-noAdd2.yaml</code>	22
LISTING 17: <code>removeTrailingWhitespace</code>	14	LISTING 56: <code>myenv.tex</code> output (using either Listing 54 or Listing 55)	22
LISTING 18: <code>fileContentsEnvironments</code>	14	LISTING 57: <code>myenv-noAdd3.yaml</code>	22
LISTING 19: <code>lookForPreamble</code>	14	LISTING 58: <code>myenv-noAdd4.yaml</code>	22
LISTING 20: <code>Motivating preambleCommandsBeforeEnvironments</code>	15	LISTING 59: <code>myenv.tex</code> output (using either Listing 57 or Listing 58)	22
LISTING 22: <code>tabular1.tex</code>	15	LISTING 60: <code>myenv-args.tex</code>	23
LISTING 23: <code>tabular1.tex</code> default output	15	LISTING 61: <code>myenv-args.tex</code> using Listing 54	23
LISTING 24: <code>tabular.yaml</code>	16	LISTING 62: <code>myenv-noAdd5.yaml</code>	23
LISTING 25: <code>tabular1.tex</code>	16	LISTING 63: <code>myenv-noAdd6.yaml</code>	23
LISTING 26: <code>tabular1.tex</code> using Listing 24	16	LISTING 64: <code>myenv-args.tex</code> using Listing 62	24
LISTING 27: <code>tabular1.tex</code> using Listing 28	16	LISTING 65: <code>myenv-args.tex</code> using Listing 63	24
LISTING 28: <code>tabular1.yaml</code>	16	LISTING 66: <code>myenv-rules1.yaml</code>	24
LISTING 29: <code>matrix1.tex</code>	17	LISTING 67: <code>myenv-rules2.yaml</code>	24
LISTING 30: <code>matrix1.tex</code> default output	17	LISTING 68: <code>myenv.tex</code> output (using either Listing 66 or Listing 67)	24
LISTING 31: <code>align-block.tex</code>	17	LISTING 69: <code>myenv-args.tex</code> using Listing 66	25
LISTING 32: <code>align-block.tex</code> default output	17	LISTING 70: <code>myenv-rules3.yaml</code>	25
LISTING 33: <code>indentAfterItems</code>	17	LISTING 71: <code>myenv-rules4.yaml</code>	25
LISTING 34: <code>items1.tex</code>	17		



LISTING 72: myenv-args.tex using Listing 70	25	LISTING 119: displayMath-indent-rules.yaml	32
LISTING 73: myenv-args.tex using Listing 71	25	LISTING 120: special1.tex using Listing 118	33
LISTING 74: noAdditionalIndentGlobal	25	LISTING 121: special1.tex using Listing 119	33
LISTING 75: myenv-args.tex using Listing 74	26	LISTING 122: special-noAdd-glob.yaml	33
LISTING 76: myenv-args.tex using Listings 66 and 74	26	LISTING 123: special-indent-rules-global.yaml	33
LISTING 77: opt-args-no-add-glob.yaml	26	LISTING 124: special1.tex using Listing 122	33
LISTING 78: mand-args-no-add-glob.yaml	26	LISTING 125: special1.tex using Listing 123	33
LISTING 79: myenv-args.tex using Listing 77	26	LISTING 126: headings2.tex	33
LISTING 80: myenv-args.tex using Listing 78	26	LISTING 127: headings2.tex using Listing 128	34
LISTING 81: indentRulesGlobal	27	LISTING 128: headings3.yaml	34
LISTING 82: myenv-args.tex using Listing 81	27	LISTING 129: headings2.tex using Listing 130	34
LISTING 83: myenv-args.tex using Listings 66 and 81	27	LISTING 130: headings4.yaml	34
LISTING 84: opt-args-indent-rules-glob.yaml	27	LISTING 131: headings2.tex using Listing 132	34
LISTING 85: mand-args-indent-rules-glob.yaml	27	LISTING 132: headings5.yaml	34
LISTING 86: myenv-args.tex using Listing 84	28	LISTING 133: headings2.tex using Listing 134	34
LISTING 87: myenv-args.tex using Listing 85	28	LISTING 134: headings6.yaml	34
LISTING 88: item-noAdd1.yaml	28	LISTING 135: headings2.tex using Listing 136	35
LISTING 89: item-rules1.yaml	28	LISTING 136: headings7.yaml	35
LISTING 90: items1.tex using Listing 88	28	LISTING 137: headings2.tex using Listing 138	35
LISTING 91: items1.tex using Listing 89	28	LISTING 138: headings8.yaml	35
LISTING 92: items-noAdditionalGlobal.yaml	29	LISTING 139: headings2.tex using Listing 140	35
LISTING 93: items-indentRulesGlobal.yaml	29	LISTING 140: headings9.yaml	35
LISTING 94: mycommand.tex	29	LISTING 141: pgfkeys1.tex	35
LISTING 95: mycommand.tex default output	29	LISTING 142: pgfkeys1.tex default output	35
LISTING 96: mycommand-noAdd1.yaml	29	LISTING 143: child1.tex	36
LISTING 97: mycommand-noAdd2.yaml	29	LISTING 144: child1.tex default output	36
LISTING 98: mycommand.tex using Listing 96	30	LISTING 145: psforeach1.tex	36
LISTING 99: mycommand.tex using Listing 97	30	LISTING 146: psforeach1.tex default output	36
LISTING 100: mycommand-noAdd3.yaml	30	LISTING 147: noAdditionalIndentGlobal	37
LISTING 101: mycommand-noAdd4.yaml	30	LISTING 148: indentRulesGlobal	37
LISTING 102: mycommand.tex using Listing 100	30	LISTING 149: commandCodeBlocks	37
LISTING 103: mycommand.tex using Listing 101	30	LISTING 150: pstricks1.tex	38
LISTING 104: mycommand-noAdd5.yaml	30	LISTING 151: pstricks1 default output	38
LISTING 105: mycommand-noAdd6.yaml	30	LISTING 152: pstricks1.tex using Listing 153	38
LISTING 106: mycommand.tex using Listing 104	31	LISTING 153: noRoundParentheses.yaml	38
LISTING 107: mycommand.tex using Listing 105	31	LISTING 154: pstricks1.tex using Listing 155	38
LISTING 108: ifelsefi1.tex	31	LISTING 155: defFunction.yaml	38
LISTING 109: ifelsefi1.tex default output	31	LISTING 156: tikz-node1.tex	39
LISTING 110: ifnum-noAdd.yaml	31	LISTING 157: tikz-node1 default output	39
LISTING 111: ifnum-indent-rules.yaml	31	LISTING 158: tikz-node1.tex using Listing 159	39
LISTING 112: ifelsefi1.tex using Listing 110	32	LISTING 159: draw.yaml	39
LISTING 113: ifelsefi1.tex using Listing 111	32	LISTING 160: tikz-node1.tex using Listing 161	39
LISTING 114: ifelsefi-noAdd-glob.yaml	32	LISTING 161: no-to.yaml	39
LISTING 115: ifelsefi-indent-rules-global.yaml	32	LISTING 162: modifyLineBreaks	40
LISTING 116: ifelsefi1.tex using Listing 114	32	LISTING 163: mlb1.tex	41
LISTING 117: ifelsefi1.tex using Listing 115	32	LISTING 164: mlb1.tex out output	41
LISTING 118: displayMath-noAdd.yaml	32	LISTING 165: textWrapOptions	41
		LISTING 166: textwrap1.tex	41
		LISTING 167: textwrap1-mod1.tex	41



LISTING 168: <code>textwrap1.yaml</code>	41	LISTING 212: <code>env-mlb2.tex</code>	50
LISTING 169: <code>textwrap2.tex</code>	42	LISTING 213: <code>env-mlb3.tex</code>	50
LISTING 170: <code>textwrap2-mod1.tex</code>	42	LISTING 214: <code>env-mlb3.tex</code> using Listing 197 on page 49	51
LISTING 171: <code>textwrap3.tex</code>	42	LISTING 215: <code>env-mlb3.tex</code> using Listing 201 on page 49	51
LISTING 172: <code>textwrap3-mod1.tex</code>	42	LISTING 216: <code>env-mlb4.tex</code>	51
LISTING 173: <code>textWrapOptions</code>	43	LISTING 217: <code>env-mlb9.yaml</code>	51
LISTING 174: <code>textwrap4.tex</code>	43	LISTING 218: <code>env-mlb10.yaml</code>	51
LISTING 175: <code>textwrap4-mod2.tex</code>	43	LISTING 219: <code>env-mlb11.yaml</code>	51
LISTING 176: <code>textwrap2.yaml</code>	43	LISTING 220: <code>env-mlb12.yaml</code>	51
LISTING 177: <code>removeParagraphLineBreaks</code>	43	LISTING 221: <code>env-mlb4.tex</code> using Listing 217	51
LISTING 178: <code>shortlines.tex</code>	44	LISTING 222: <code>env-mlb4.tex</code> using Listing 218	51
LISTING 179: <code>remove-para1.yaml</code>	44	LISTING 223: <code>env-mlb4.tex</code> using Listing 219	52
LISTING 180: <code>shortlines1.tex</code>	44	LISTING 224: <code>env-mlb4.tex</code> using Listing 220	52
LISTING 181: <code>shortlines1-tws.tex</code>	44	LISTING 225: <code>env-mlb5.tex</code>	52
LISTING 182: <code>shortlines-mand.tex</code>	45	LISTING 226: <code>removeTWS-before.yaml</code>	52
LISTING 183: <code>shortlines-opt.tex</code>	45	LISTING 227: <code>env-mlb5.tex</code> using Listings 221 to 224 ..	52
LISTING 184: <code>shortlines-mand1.tex</code>	45	LISTING 228: <code>env-mlb5.tex</code> using Listings 221 to 224 and Listing 226	52
LISTING 185: <code>shortlines-opt1.tex</code>	45	LISTING 229: <code>env-mlb6.tex</code>	53
LISTING 186: <code>shortlines-envs.tex</code>	46	LISTING 230: <code>UnpreserveBlankLines.yaml</code>	53
LISTING 187: <code>remove-para2.yaml</code>	46	LISTING 231: <code>env-mlb6.tex</code> using Listings 221 to 224 ..	53
LISTING 188: <code>remove-para3.yaml</code>	46	LISTING 232: <code>env-mlb6.tex</code> using Listings 221 to 224 and Listing 230	53
LISTING 189: <code>shortlines-envs2.tex</code>	46	LISTING 242: <code>mycommand1.tex</code>	55
LISTING 190: <code>shortlines-envs3.tex</code>	47	LISTING 243: <code>mycommand1.tex</code> using Listing 244	55
LISTING 191: <code>shortlines-md.tex</code>	47	LISTING 244: <code>mycom-mlb1.yaml</code>	55
LISTING 192: <code>remove-para4.yaml</code>	47	LISTING 245: <code>mycommand1.tex</code> using Listing 246	55
LISTING 193: <code>shortlines-md4.tex</code>	48	LISTING 246: <code>mycom-mlb2.yaml</code>	55
LISTING 194: <code>environments</code>	48	LISTING 247: <code>mycommand1.tex</code> using Listing 248	56
LISTING 195: <code>env-mlb1.tex</code>	49	LISTING 248: <code>mycom-mlb3.yaml</code>	56
LISTING 196: <code>env-mlb1.yaml</code>	49	LISTING 249: <code>mycommand1.tex</code> using Listing 250	56
LISTING 197: <code>env-mlb2.yaml</code>	49	LISTING 250: <code>mycom-mlb4.yaml</code>	56
LISTING 198: <code>env-mlb.tex</code> using Listing 196	49	LISTING 251: <code>mycommand1.tex</code> using Listing 252	56
LISTING 199: <code>env-mlb.tex</code> using Listing 197	49	LISTING 252: <code>mycom-mlb5.yaml</code>	56
LISTING 200: <code>env-mlb3.yaml</code>	49	LISTING 253: <code>mycommand1.tex</code> using Listing 254	57
LISTING 201: <code>env-mlb4.yaml</code>	49	LISTING 254: <code>mycom-mlb6.yaml</code>	57
LISTING 202: <code>env-mlb.tex</code> using Listing 200	49	LISTING 255: <code>nested-env.tex</code>	57
LISTING 203: <code>env-mlb.tex</code> using Listing 201	49	LISTING 256: <code>nested-env.tex</code> using Listing 256	57
LISTING 204: <code>env-mlb5.yaml</code>	49	LISTING 257: <code>nested-env-mlb1.yaml</code>	57
LISTING 205: <code>env-mlb6.yaml</code>	49	LISTING 258: <code>nested-env.tex</code> using Listing 259	58
LISTING 206: <code>env-mlb.tex</code> using Listing 204	50	LISTING 259: <code>nested-env-mlb2.yaml</code>	58
LISTING 207: <code>env-mlb.tex</code> using Listing 205	50	LISTING 260: <code>helloworld.pl</code>	60
LISTING 208: <code>env-mlb7.yaml</code>	50		
LISTING 209: <code>env-mlb8.yaml</code>	50		
LISTING 210: <code>env-mlb.tex</code> using Listing 208	50		
LISTING 211: <code>env-mlb.tex</code> using Listing 209	50		

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 8.2 on page 59 for their valued contributions, and thank you to those who report bugs and request features at [6].

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 12) but you should still be careful when using it. The script has been tested on many files, but there are some known limitations (see Section 7). You, the user, are responsible for ensuring that you maintain backups of your files before running `latexindent.pl` on them. I think it is important at this stage to restate an important part of the license here:

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

There is certainly no malicious intent in releasing this script, and I do hope that it works as you expect it to; if it does not, please first of all make sure that you have the correct settings, and then feel free to let me know at [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 appendix C on page 62 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 [11]

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 you can personalize your indentation scheme.

In each of the samples given in Listings 1 to 6 the 'before' case is a 'worst case scenario' with no effort to make indentation. The 'after' result would be the same, regardless of the leading white space at the beginning of each line which is stripped by `latexindent.pl` (unless a `verbatim-like` environment or `noIndentBlock` is specified – more on this in Section 5).

LISTING 1: `filecontents1.tex`

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

LISTING 2: `filecontents1.tex` default output

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



LISTING 3: tikzset.tex

```
\tikzset{
shrink_inner_sep/.code={
\pgfkeysgetvalue...
\pgfkeysgetvalue...
}
}
```

LISTING 4: tikzset.tex default output

```
\tikzset{
  \shrink_inner_sep/.code={
    \pgfkeysgetvalue...
    \pgfkeysgetvalue...
  }
}
```

LISTING 5: pstricks.tex

```
\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.tex default output

```
\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_X distributions 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 B on page 60 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 5 on page 11.

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 59 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
```

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

-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 5, 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 C on page 62 for details of how the interface has changed from Version 2.2 to Version 3.0 for this flag.

-s, -silent

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

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

-t, -trace

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

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

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



-tt, -ttrace

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

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

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

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

`latexindent.pl` will always load `defaultSettings.yaml` (rhymes with camel) and if it is called with the `-l` switch and it finds `localSettings.yaml` in the same directory as `myfile.tex` then 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`. In fact, you can specify *relative* path names to the current directory, but *not* absolute paths – for absolute paths, see Section 4 on page 10. Explicit demonstrations of how to use the `-l` switch are given throughout this documentation.

-d, -onlydefault

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

Only `defaultSettings.yaml`: you might like to read Section 5 before using this switch. By default, `latexindent.pl` will always search for `indentconfig.yaml` or `.indentconfig.yaml` in your home directory. If you would prefer it not to do so then (instead of deleting or renaming `indentconfig.yaml`/`.indentconfig.yaml`) you can simply call the script with the `-d` switch; note that this will also tell the script to ignore `localSettings.yaml` even if it has been called with the `-l` switch.

-c, -cruft=<directory>

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

If you wish to have backup files and `indent.log` written to a directory other than the current working directory, then you can send these ‘cruft’ files to another directory.

-g, -logfile

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

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



`-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 6 on page 40

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

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

and in which case, you can specify the order in which extensions are searched for; see Listing 11 on page 12 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`. `arara` ships with a rule, `indent.yaml`, but in case you do not have this rule, you can find it at [2].

You can use the rule in 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 }
\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.

TABLE 1: `arara` directive arguments and corresponding switches

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

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



4 User, local settings, indentconfig.yaml and .indentconfig.yaml

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` (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`² Listing 8 shows a sample `indentconfig.yaml` file.

LISTING 8: `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 9 for an example that uses four tabs for the default indent, adds the `tabbing` environment/command to the list of environments that contains alignment delimiters; you might also like to refer to the many YAML files detailed throughout the rest of this documentation.

LISTING 9: `mysettings.yaml` (example)

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

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

You can make sure that your settings are loaded by checking `indent.log` for details – if you have specified a path that `latexindent.pl` doesn’t recognize then you’ll get a warning, otherwise you’ll get confirmation that `latexindent.pl` has read your settings file³.

²If 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.

³Windows users may find that they have to end `.yaml` files with a blank line



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 `whateveryoulike.yaml` file.

If `latexindent.pl` can not read your `.yaml` file it will tell you so in `indent.log`.

4.1 localSettings.yaml

The `-l` switch tells `latexindent.pl` to look for `localSettings.yaml` in the *same directory* as `myfile.tex`. 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
```

Any settings file(s) specified using the `-l` switch will be read *after* `defaultSettings.yaml` and, assuming they exist, user settings from `indentconfig.yaml`.

Your settings file can contain any switches that you'd like to change; a sample is shown in Listing 10, and you'll find plenty of further examples throughout this manual.

LISTING 10: `localSettings.yaml` (example)

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

You can make sure that your settings file has been loaded by checking `indent.log` for details; if it can not be read then you receive a warning, otherwise you'll get confirmation that `latexindent.pl` has read your settings file.

4.2 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 4.1). You may specify relative paths to other YAML files using the `-l` switch, separating multiple files using commas.

A visual representation of this is given in Figure 1.

5 defaultSettings.yaml

`latexindent.pl` loads its settings from `defaultSettings.yaml`. The idea is to separate the behaviour of the script from the internal working – this is very similar to the way that we separate content from form when writing our documents in \LaTeX .

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.

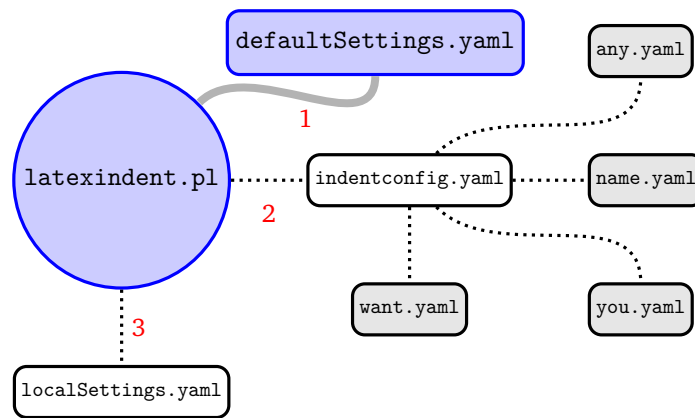


FIGURE 1: Schematic of the load order described in Section 4.2; solid lines represent mandatory files, dotted lines represent optional files. `indentconfig.yaml` can contain as many files as you like. The files will be loaded in order; if you specify settings for the same field in more than one file, the most recent takes priority.

`fileExtensionPreference:` *<fields>*

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

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

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

Calling `latexindent.pl myfile` with the (default) settings specified in Listing 11 means that the script will first look for `myfile.tex`, then `myfile.sty`, `myfile.cls`, and finally `myfile.bib` in order⁴.

LISTING 11:
`fileExtensionPreference`

```
38 fileExtensionPreference:
39   .tex: 1
40   .sty: 2
41   .cls: 3
42   .bib: 4
```

`backupExtension:` *<extension name>*

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

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

`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.

⁴Throughout this manual, listings with line numbers represent code taken directly from `defaultSettings.yaml`.



`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 12. If you load your own user settings (see Section 4 on page 10) 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 12: 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 13.

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 13:
verbatimEnvironments

verbatimEnvironments:
 verbatim: 1
 lstlisting: 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

LISTING 14:
verbatimCommands

verbatimCommands:
 verb: 1
 lstinlne: 1



routines (only relevant if the `-m` is active, see Section 6 on page 40).

```
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 15.

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 16 for example.

LISTING 15:
noIndentBlock

noIndentBlock:
 noindent: 1
 cmhtest: 1

LISTING 16: 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 17; each of the fields can take the values 0 or 1. See Listings 226 to 228 on page 52 for before and after results. Thanks to [12] for providing this feature.

LISTING 17:
removeTrailingWhitespace

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

LISTING 18:
fileContentsEnvironments

fileContentsEnvironments:
 filecontents: 1
 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 19, if you set, for example, `.tex` to 0, then regardless of the setting of the value of

LISTING 19:
lookForPreamble

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



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 20.

LISTING 20: Motivating `preambleCommandsBeforeEnvironments`

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

`defaultIndent`: $\langle horizontal\ space \rangle$

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` in Section 5.2 on page 21 for more details.

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

`lookForAlignDelims`: $\langle fields \rangle$

This contains a list of environments and/or commands that are operated upon in a special way by `latexindent.pl` (see Listing 21). In fact, the fields in `lookForAlignDelims` can actually take two different forms: the *basic* version is shown in Listing 21 and the *advanced* version in Listing 24; 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 Section 7), but in many cases it will produce results such as those in Listings 22 and 23.

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 15).

LISTING 22: `tabular1.tex`

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

LISTING 21:
`lookForAlignDelims`
(basic)

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

LISTING 23: `tabular1.tex` default
output

```
\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 24 is for you.



LISTING 24: tabular.yaml

```
lookForAlignDelims:
  tabular:
    delims: 1
    alignDoubleBackSlash: 0
    spacesBeforeDoubleBackSlash: 0
  tabularx:
    delims: 1
  longtable: 1
```

Note that you can use a mixture of the basic and advanced form: in Listing 24 `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 21 (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 24 saved in `tabular.yaml`, and the code from Listing 22 in `tabular1.tex` and you run

```
cmh:~$ latexindent.pl -l tabular.yaml tabular1.tex
```

then you should receive the before-and-after results shown in Listings 25 and 26; note that the ampersands have been aligned, but the `\\` have not (compare the alignment of `\\` in Listings 23 and 26).

LISTING 25: tabular1.tex

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

LISTING 26: tabular1.tex using Listing 24

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

Saving Listing 24 into `tabular1.yaml` as in Listing 28, and running the command

```
cmh:~$ latexindent.pl -l tabular1.yaml tabular1.tex
```

gives Listing 27; note the spacing before the `\\`.

LISTING 27: tabular1.tex using Listing 28

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

LISTING 28: tabular1.yaml

```
lookForAlignDelims:
  tabular:
    delims: 1
    alignDoubleBackSlash: 0
    spacesBeforeDoubleBackSlash: 3
  tabularx:
    delims: 1
  longtable: 1
```

As of Version 3.0, the alignment routine works on mandatory and optional arguments within commands, and also within ‘special’ code blocks (see `specialBeginEnd` on page 18); for example, assuming that you have a command called `\matrix` and that it is populated within `lookForAlignDelims` (which it is, by default), and that you run the command



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

then the before-and-after results shown in Listings 29 and 30 are achievable by default.

LISTING 29: matrix1.tex

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

LISTING 30: matrix1.tex default output

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

If you have blocks of code that you wish to align at the & character that are *not* wrapped in, for example, `\begin{tabular} ... \end{tabular}`, then you can use the mark up illustrated in Listing 31; the default output is shown in Listing 32. Note that the `%*` must be next to each other, but that there can be any number of spaces (possibly none) between the `*` and `\begin{tabular}`; note also that you may use any environment name that you have specified in `lookForAlignDelims`.

LISTING 31: align-block.tex

```
%*\begin{tabular}
\1\&\2\&\3\&\4\\
\5\&\6\&\7\&\8\\
\9%\end{tabular}
```

LISTING 32: align-block.tex default output

```
%*\begin{tabular}
\1\&\2\&\3\&\4\\
\5\&\6\&\7\&\8\\
%\end{tabular}
```

With reference to Table 2 on page 20 and the, yet undiscussed, fields of `noAdditionalIndent` and `indentRules` (see Section 5.2 on page 21), these comment-marked blocks are considered environments.

```
indentAfterItems: {fields}
```

The environment names specified in `indentAfterItems` tell `latexindent.pl` to look for `\item` commands; if these switches are set to 1 then indentation will be performed so as indent the code after each item. A demonstration is given in Listings 34 and 35

LISTING 34: items1.tex

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

LISTING 33: indentAfterItems

```
171 indentAfterItems:
172   itemize: 1
173   enumerate: 1
174   list: 1
```

LISTING 35: items1.tex default output

```
\begin{itemize}
  \item some text here
  \item some more text here
  \item some more text here
  \item another item
  \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 4 on page 10 for

LISTING 36: itemNames

```
180 itemNames:
181   item: 1
182   myitem: 1
```

details of how to configure user settings, and Listing 9 on page 10 in particular .)

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 37 shows the default settings of specialBeginEnd.

LISTING 37: specialBeginEnd

```
186 specialBeginEnd:
187   displayMath:
188     begin: '\\\\['
189     end: '\\\\]'
190     lookForThis: 1
191   inlineMath:
192     begin: '(?!\\$)(?!\\\\)\\$(?!\\$)'
193     end: '(?!\\\\)\\$(?!\\$)'
194     lookForThis: 1
195   displayMathTeX:
196     begin: '\\$\\$'
197     end: '\\$\\$'
198     lookForThis: 1
```

The field displayMath represents `\[...]`, inlineMath represents `$...$` and displayMathTeX represents `$$...$$`. You can, of course, rename these in your own YAML files (see Section 4.1 on page 11); 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 38 and 39.

LISTING 38: special1.tex before

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

LISTING 39: special1.tex default output

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

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

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.⁵

The default settings do *not* place indentation after a heading, but you can easily

⁵There is a slight difference in interface for this field when comparing Version 2.2 to Version 3.0; see appendix C on page 62 for details.

LISTING 40: indentAfterHeadings

```
indentAfterHeadings:
  part:
    indentAfterThisHeading: 0
    level: 1
  chapter:
    indentAfterThisHeading: 0
    level: 2
  section:
    indentAfterThisHeading: 0
    level: 3
```



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

You can add any of your own custom heading commands to this field, specifying the level as appropriate. You can also specify your own indentation in `indentRules` (see Section 5.2 on page 21); 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 the code in Listing 41 saved into `headings1.yaml`, and that you have the text from Listing 42 saved into `headings1.tex`.

LISTING 41: `headings1.yaml`

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

LISTING 42: `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 43.

LISTING 43: `headings1.tex` using
Listing 41

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

LISTING 44: `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 41 so that the paragraph level is 1; after running

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

you should receive the code given in Listing 44; notice that the paragraph and subsection are at the same indentation level.

5.1 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_\\	<code>\begin{myenv}</code> body of myenv <code>\end{myenv}</code>
optionalArguments	<i>inherits</i> name from parent (e.g environment name)	<code>[</code> opt arg text <code>]</code>
mandatoryArguments	<i>inherits</i> name from parent (e.g environment name)	<code>{</code> mand arg text <code>}</code>
commands	+a-zA-Z@*0-9_:	<code>\mycommand{arguments}</code>
keyEqualsValuesBracesBrackets	a-zA-Z@*0-9_/. \h\{\}: \#-	my key/.style={arguments}
namedGroupingBracesBrackets	a-zA-Z@*><	in{arguments}
UnNamedGroupingBracesBrackets	<i>No name!</i>	{ or [or , or & or) or (or \$ followed by {arguments}
ifElseFi	@a-zA-Z but must begin with either <code>\if</code> of <code>\@if</code>	<code>\ifnum...</code> ... <code>\else</code> ... <code>\fi</code>
items	User specified, see Listings 33 and 36 on page 17	<code>\begin{enumerate}</code> <code>\item ...</code> <code>\end{enumerate}</code>
specialBeginEnd	User specified, see Listing 37 on page 18	<code>\[</code> ... <code>\]</code>
afterHeading	User specified, see Listing 40 on page 18	<code>\chapter{title}</code> ... <code>\section{title}</code>



filecontents	User specified, see Listing 18 on page 14...	<pre>\begin{filecontents} ... \end{filecontents}</pre>
--------------	--	--

We will refer to these code blocks in what follows.

5.2 noAdditionalIndent and indentRules

latexindent.pl operates on files by looking for code blocks, as detailed in Section 5.1 on page 19; for each type of code block in Table 2 on page 20 (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 20; for reference, there follows a list of the code blocks covered.

5.2.1	Environments and their arguments	21
5.2.2	Environments with items	28
5.2.3	Commands with arguments	29
5.2.4	ifelsefi code blocks	31
5.2.5	specialBeginEnd code blocks	32
5.2.6	afterHeading code blocks	33
5.2.7	The remaining code blocks	35
5.2.8	Summary	37

5.2.1 Environments and their arguments

There are a few different YAML switches governing the indentation of environments; let's start with the code shown in Listing 53.

LISTING 53: 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 54 and 55.



LISTING 54:
myenv-noAdd1.yaml

```
noAdditionalIndent:
  myenv: 1
```

LISTING 55:
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 56; note in particular that the environment `myenv` has not received any *additional* indentation, but that the outer environment *has* still received indentation.

LISTING 56: `myenv.tex` output (using either Listing 54 or Listing 55)

```
\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 57 and 58, 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 59.

LISTING 57:
myenv-noAdd3.yaml

```
noAdditionalIndent:
  myenv: 0
```

LISTING 58:
myenv-noAdd4.yaml

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

LISTING 59: `myenv.tex` output (using either Listing 57 or Listing 58)

```
\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 60.



LISTING 60: 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 61; 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 54), then *all* parts of the environment (body, optional and mandatory arguments) are assumed to want no additional indent.

LISTING 61: myenv-args.tex using Listing 54

```

\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 62 and 63.

LISTING 62: myenv-noAdd5.yaml

```

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

```

LISTING 63: 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 64 and 65. Note that in Listing 64 the text for the *optional* argument has not received any additional indentation, and that in Listing 65 the *mandatory* argument has not received any additional indentation; in both cases, the *body* has not received any additional indentation.



LISTING 64: myenv-args.tex using Listing 62

```

\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 65: myenv-args.tex using Listing 63

```

\begin{outer}
  \begin{myenv}[%
    \optional_argument_text
    \optional_argument_text]%
    \mandatory_argument_text
    \mandatory_argument_text}
  \body_of_environment
  \body_of_environment
  \body_of_environment
\end{myenv}
\end{outer}

```

```
indentRules: {fields}
```

We may also specify indentation rules for environment code blocks using the `indentRules` field; see, for example, Listings 66 and 67.

LISTING 66:
myenv-rules1.yaml

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

LISTING 67:
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 68; note in particular that the environment `myenv` has received one tab (from the outer environment) plus three spaces from Listing 66 or 67.

LISTING 68: myenv.tex output (using either Listing 66 or Listing 67)

```

\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.

Returning to the example in Listing 60 that contains optional and mandatory arguments. Upon using Listing 66 as in

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

we obtain the output in Listing 69; note that the body, optional argument and mandatory argument have *all* received the same customised indentation.



LISTING 69: 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}

```

You can specify different indentation rules for the different features using, for example, Listings 70 and 71

LISTING 70: myenv-rules3.yaml

```

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

```

LISTING 71: 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 72 and 73.

LISTING 72: myenv-args.tex using Listing 70

```

\begin{outer}
  \begin{myenv}[%
    \optional_argument_text
    \optional_argument_text]%
    \{ \mandatory_argument_text
    \mandatory_argument_text}
    \body_of_environment
    \body_of_environment
    \body_of_environment
  \end{myenv}
\end{outer}

```

LISTING 73: myenv-args.tex using Listing 71

```

\begin{outer}
  \begin{myenv}[%
    \optional_argument_text
    \optional_argument_text]%
    \{ \mandatory_argument_text
    \mandatory_argument_text}
    \body_of_environment
    \body_of_environment
    \body_of_environment
  \end{myenv}
\end{outer}

```

Note that in Listing 72, 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 73, 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 List-

263
264

LISTING 74:

```
noAdditionalIndentGlobal
```

```
noAdditionalIndentGlobal:
  environments: 0

```



ing 74). Let's say that you change the value of environments to 1 in Listing 74, and that you run

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

The respective output from these two commands are in Listings 75 and 76; in Listing 75 notice that *both* environments receive no additional indentation but that the arguments of myenv still *do* receive indentation. In Listing 76 notice that the *outer* environment does not receive additional indentation, but because of the settings from myenv-rules1.yaml (in Listing 66 on page 24), the myenv environment still *does* receive indentation.

LISTING 75: myenv-args.tex using Listing 74

```
\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 76: myenv-args.tex using Listings 66 and 74

```
\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 77 and 78

LISTING 77:

opt-args-no-add-glob.yaml

```
noAdditionalIndentGlobal:
  optionalArguments: 1
```

LISTING 78:

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 79 and 80. Notice that in Listing 79 the *optional* argument has not received any additional indentation, and in Listing 80 the *mandatory* argument has not received any additional indentation.

LISTING 79: myenv-args.tex using Listing 77

```
\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 80: myenv-args.tex using Listing 78

```
\begin{outer}
  \begin{myenv}[%
    \optional_argument_text
    \optional_argument_text]%
    \{\mandatory_argument_text
    \mandatory_argument_text}
    \body_of_environment
    \body_of_environment
    \body_of_environment
  \end{myenv}
\end{outer}
```



```
indentRulesGlobal: {fields}
```

The final check that `latexindent.pl` will make is to look for `indentRulesGlobal` as detailed in Listing 81; if you change the `environments` field to anything involving horizontal space, say " ", and then run the following commands

279
280

LISTING 81:

`indentRulesGlobal`

```
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 82 and 83. Note that in Listing 82, both the environment blocks have received a single-space indentation, whereas in Listing 83 the outer environment has received single-space indentation (specified by `indentRulesGlobal`), but `myenv` has received " ", as specified by the particular `indentRules` for `myenv` Listing 66 on page 24.

LISTING 82: `myenv-args.tex` using Listing 81

```
\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 83: `myenv-args.tex` using Listings 66 and 81

```
\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
  body_of_environment
\end{myenv}
\end{outer}
```

You can specify `indentRulesGlobal` for both optional and mandatory arguments, as detailed in Listings 84 and 85

LISTING 84:

`opt-args-indent-rules-glob.yaml`

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

LISTING 85:

`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 86 and 87. Note that the *optional* argument in Listing 86 has received two tabs worth of indentation, while the *mandatory* argument has done so in Listing 87.



LISTING 86: myenv-args.tex using Listing 84

```

\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 87: myenv-args.tex using Listing 85

```

\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}

```

5.2.2 Environments with items

With reference to Listings 33 and 36 on page 17, some commands may contain item commands; for the purposes of this discussion, we will use the code from Listing 34 on page 17.

Assuming that you've populated itemNames with the name of your item, you can put the item name into noAdditionalIndent as in Listing 88, 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 89

LISTING 88: item-noAdd1.yaml

```

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

```

LISTING 89: 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 90 and 91; note that in Listing 90 that the text after each item has not received any additional indentation, and in Listing 91, the text after each item has received a single space of indentation, specified by Listing 89.

LISTING 90: items1.tex using Listing 88

```

\begin{itemize}
  \item_some_text_here
  \some_more_text_here
  \some_more_text_here
  \item_another_item
  \some_more_text_here
\end{itemize}

```

LISTING 91: items1.tex using Listing 89

```

\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 92 and 93. Note that there is a need to 'reset/remove' the item field from indentRules in both cases (see the hierarchy description given on page 21) as the item command is a member of indentRules by default.



LISTING 92:
items-noAdditionalGlobal.yaml

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

LISTING 93:
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 90 and 91 are obtained; note, however, that *all* such item commands without their own individual noAdditionalIndent or indentRules settings would behave as in these listings.

5.2.3 Commands with arguments

Let's begin with the simple example in Listing 94; when latexindent.pl operates on this file, the default output is shown in Listing 95.⁶

LISTING 94: mycommand.tex

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

LISTING 95: 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 54 and 55 on page 22) we may specify noAdditionalIndent either in 'scalar' form, or in 'field' form, as shown in Listings 96 and 97

LISTING 96:
mycommand-noAdd1.yaml

```
noAdditionalIndent:
  mycommand: 1
```

LISTING 97:
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 98 and 99

⁶The command code blocks have quite a few subtleties, described in Section 5.3 on page 37.



LISTING 98: mycommand.tex using
Listing 96

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

LISTING 99: mycommand.tex using
Listing 97

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

Note that in Listing 98 that the ‘body’, optional argument *and* mandatory argument have *all* received no additional indentation, while in Listing 99, 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 62 and 63 on page 23; explicit examples are given in Listings 100 and 101.

LISTING 100:
mycommand-noAdd3.yaml

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

LISTING 101:
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 102 and 103.

LISTING 102: mycommand.tex using
Listing 100

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

LISTING 103: mycommand.tex using
Listing 101

```
\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 102 and 103 has received no additional indent, even though body is explicitly set to 0 in both Listings 100 and 101. This is because, by default, noAdditionalIndentGlobal for commands is set to 1 by default; this can be easily fixed as in Listings 104 and 105.

LISTING 104:
mycommand-noAdd5.yaml

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

LISTING 105:
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 106 and 107.

LISTING 106: mycommand.tex using Listing 104

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

LISTING 107: mycommand.tex using Listing 105

```
\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 70 and 71 on page 25 and Listings 81, 84 and 85 on page 27.

5.2.4 ifelsefi code blocks

Let's use the simple example shown in Listing 108; when `latexindent.pl` operates on this file, the output as in Listing 109; note that the body of each of the `\if` statements have been indented, and that the `\else` statement has been accounted for correctly.

LISTING 108: ifelsefi1.tex

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

LISTING 109: 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 110 and 111.

LISTING 110:
ifnum-noAdd.yaml

```
noAdditionalIndent:
  ifnum: 1
```

LISTING 111:
ifnum-indent-rules.yaml

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

After running the following commands,

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

we receive the respective output given in Listings 112 and 113; note that in Listing 112, the `ifnum` code block has *not* received any additional indentation, while in Listing 113, the `ifnum` code block has received one tab and two spaces of indentation.



LISTING 112: ifelsefi1.tex using Listing 110

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

LISTING 113: ifelsefi1.tex using Listing 111

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

We may specify `noAdditionalIndentGlobal` and `indentRulesGlobal` as in Listings 114 and 115.

LISTING 114:
ifelsefi-noAdd-glob.yaml

```
noAdditionalIndentGlobal:
  ifElseFi: 1
```

LISTING 115:
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 116 and 117; notice that in Listing 116 neither of the `ifelsefi` code blocks have received indentation, while in Listing 117 both code blocks have received a single space of indentation.

LISTING 116: ifelsefi1.tex using Listing 114

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

LISTING 117: ifelsefi1.tex using Listing 115

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

5.2.5 specialBeginEnd code blocks

Let's use the example from Listing 38 on page 18 which has default output shown in Listing 39 on page 18.

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 118 and 119.

LISTING 118:
displayMath-noAdd.yaml

```
noAdditionalIndent:
  displayMath: 1
```

LISTING 119:
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 120 and 121; note that in Listing 120, the `displayMath` code block has *not* received any additional indentation, while in Listing 121, the `displayMath` code block has received three tabs worth of indentation.



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)
$
```

LISTING 121: special1.tex using Listing 119

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

We may specify `noAdditionalIndentGlobal` and `indentRulesGlobal` as in Listings 122 and 123.

LISTING 122:
special-noAdd-glob.yaml

```
noAdditionalIndentGlobal:
  specialBeginEnd: 1
```

LISTING 123:
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 124 and 125; notice that in Listing 124 neither of the special code blocks have received indentation, while in Listing 125 both code blocks have received a single space of indentation.

LISTING 124: special1.tex using Listing 122

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

LISTING 125: special1.tex using Listing 123

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

5.2.6 afterHeading code blocks

Let's use the example Listing 126 for demonstration throughout this Section. As discussed on page 19, by default `latexindent.pl` will not add indentation after headings.

LISTING 126: headings2.tex

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

On using the YAML file in Listing 128 by running the command

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



we obtain the output in Listing 127. Note that the argument of paragraph has received (default) indentation, and that the body after the heading statement has received (default) indentation.

LISTING 127: headings2.tex using Listing 128

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

LISTING 128: headings3.yaml

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

If we specify noAdditionalIndent as in Listing 130 and run the command

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

then we receive the output in Listing 129. 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 129: headings2.tex using Listing 130

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

LISTING 130: headings4.yaml

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

Similarly, if we specify indentRules as in Listing 132 and run analogous commands to those above, we receive the output in Listing 131; note that the *body*, *mandatory argument* and content *after the heading* of paragraph have *all* received three tabs worth of indentation.

LISTING 131: headings2.tex using Listing 132

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

LISTING 132: headings5.yaml

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

We may, instead, specify noAdditionalIndent in ‘field’ form, as in Listing 134 which gives the output in Listing 133.

LISTING 133: headings2.tex using Listing 134

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

LISTING 134: headings6.yaml

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

Analogously, we may specify indentRules as in Listing 136 which gives the output in Listing 135; note that mandatory argument text has only received a single space of indentation, while the body after the heading has received three tabs worth of indentation.



LISTING 135: headings2.tex using Listing 136

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

LISTING 136: headings7.yaml

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

Finally, let's consider `noAdditionalIndentGlobal` and `indentRulesGlobal` shown in Listings 138 and 140 respectively, with respective output in Listings 137 and 139. Note that in Listing 138 the *mandatory argument* of `paragraph` has received a (default) tab's worth of indentation, while the body after the heading has received *no additional indentation*. Similarly, in Listing 139, the *argument* has received both a (default) tab plus two spaces of indentation (from the global rule specified in Listing 140), and the remaining body after `paragraph` has received just two spaces of indentation.

LISTING 137: headings2.tex using Listing 138

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

LISTING 138: headings8.yaml

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

LISTING 139: headings2.tex using Listing 140

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

LISTING 140: headings9.yaml

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

5.2.7 The remaining code blocks

Referencing the different types of code blocks in Table 2 on page 20, we have a few code blocks yet to cover; these are very similar to the `commands` code block type covered comprehensively in Section 5.2.3 on page 29, but a small discussion defining these remaining code blocks is necessary.

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

- it must immediately follow either `{` OR `[` OR `,` with comments and blank lines allowed;
- then it has a name made up of the characters detailed in Table 2 on page 20;
- then an `=` symbol;
- then at least one set of curly braces or square brackets (comments and line breaks allowed throughout).

An example is shown in Listing 141, with the default output given in Listing 142.

LISTING 141: pgfkeys1.tex

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

LISTING 142: pgfkeys1.tex default output

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

In Listing 142, note that the maximum indentation is three tabs, and these come from:



- the `\pgfkeys` command's mandatory argument;
- the `start coordinate/.initial` key's mandatory argument;
- the `start coordinate/.initial` key's body, which is defined as any lines following the name of the key that include its arguments. This is the part controlled by the `body` field for `noAdditionalIndent` and friends from page 21.

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

- it must immediately follow either *horizontal space* OR *one or more line breaks* OR `{` OR `[` OR `$`;
- the name may contain the characters detailed in Table 2 on page 20;
- then at least one set of curly braces or square brackets (comments and line breaks allowed throughout).

A simple example is given in Listing 143, with default output in Listing 144.

LISTING 143: child1.tex

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

LISTING 144: child1.tex default output

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

In particular, `latexindent.pl` considers `child`, `parent` and `node` all to be `namedGroupingBracesBrackets`⁷. Referencing Listing 144, note that the maximum indentation is two tabs, and these come from:

- the `child`'s mandatory argument;
- the `child`'s body, which is defined as any lines following the name of the `namedGroupingBracesBrackets` that include its arguments. This is the part controlled by the `body` field for `noAdditionalIndent` and friends from page 21.

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

- it must immediately follow either `{` OR `[` OR `,` OR `&` OR `)` OR `(` OR `$`;
- then at least one set of curly braces or square brackets (comments and line breaks allowed throughout).

An example is shown in Listing 145 with default output give in Listing 146.

LISTING 145: psforeach1.tex

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

LISTING 146: psforeach1.tex default output

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

Referencing Listing 146, there are *three* sets of unnamed braces. Note also that the maximum value of indentation is three tabs, and these come from:

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

⁷ You may like to verify this by using the `-tt` option and checking `indent.log`!



- the *first* un-named braces *body*, which we define as any lines following the first opening { or [that defined the code block. This is the part controlled by the *body* field for `noAdditionalIndent` and friends from page 21.

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

`filecontents` code blocks behave just as `environments`, except that neither arguments nor items are sought.

5.2.8 Summary

Having considered all of the different types of code blocks, the functions of the fields given in Listings 147 and 148 should now make sense.

LISTING 147: `noAdditionalIndentGlobal`

```
263 noAdditionalIndentGlobal:
264   environments: 0
265   commands: 1
266   optionalArguments: 0
267   mandatoryArguments: 0
268   ifElseFi: 0
269   items: 0
270   keyEqualsValuesBracesBrackets: 0
271   namedGroupingBracesBrackets: 0
272   UnNamedGroupingBracesBrackets: 0
273   specialBeginEnd: 0
274   afterHeading: 0
275   filecontents: 0
```

LISTING 148: `indentRulesGlobal`

```
279 indentRulesGlobal:
280   environments: 0
281   commands: 0
282   optionalArguments: 0
283   mandatoryArguments: 0
284   ifElseFi: 0
285   items: 0
286   keyEqualsValuesBracesBrackets: 0
287   namedGroupingBracesBrackets: 0
288   UnNamedGroupingBracesBrackets: 0
289   specialBeginEnd: 0
290   afterHeading: 0
291   filecontents: 0
```

5.3 Commands and the strings between their arguments

The command code blocks will always look for optional (square bracketed) and mandatory (curly braced) arguments which can contain comments, line breaks and ‘beamer’ commands `<.*?>` between them. There are switches that can allow them to contain other strings, which we discuss next.

```
commandCodeBlocks: {fields}
```

The `commandCodeBlocks` field contains a few switches detailed in Listing 149.

LISTING 149: `commandCodeBlocks`

```
294 commandCodeBlocks:
295   roundParenthesesAllowed: 1
296   stringsAllowedBetweenArguments:
297     node: 1
298     at: 1
299     to: 1
300     decoration: 1
301     ++: 1
302     --: 1
```

```
roundParenthesesAllowed: 0|1
```

The need for this field was mostly motivated by commands found in code used to generate images in `PSTricks` and `tikz`; for example, let’s consider the code given in Listing 150.



LISTING 150: pstricks1.tex

```
\defFunction[algebraic]{torus}(u,v)
{(2+cos(u))*cos(v+\Pi)}
{(2+cos(u))*sin(v+\Pi)}
{sin(u)}
```

LISTING 151: pstricks1 default output

```
\defFunction[algebraic]{torus}(u,v)
{(2+cos(u))*cos(v+\Pi)}
{(2+cos(u))*sin(v+\Pi)}
{sin(u)}
```

Notice that the `\defFunction` command has an optional argument, followed by a mandatory argument, followed by a round-parenthesis argument, (u, v) .

By default, because `roundParenthesesAllowed` is set to 1 in Listing 149, then `latexindent.pl` will allow round parenthesis between optional and mandatory arguments. In the case of the code in Listing 150, `latexindent.pl` finds *all* the arguments of `defFunction`, both before and after (u, v) .

The default output from running `latexindent.pl` on Listing 150 actually leaves it unchanged (see Listing 151); note in particular, this is because of `noAdditionalIndentGlobal` as discussed on page 30.

Upon using the YAML settings in Listing 153, and running the command

```
cmh:~$ latexindent.pl pstricks1.tex -l noRoundParentheses.yaml
```

we obtain the output given in Listing 152.

LISTING 152: pstricks1.tex using Listing 153

```
\defFunction[algebraic]{torus}(u,v)
{(2+cos(u))*cos(v+\Pi)}
  *{(2+cos(u))*sin(v+\Pi)}
  *{sin(u)}
```

LISTING 153:

noRoundParentheses.yaml

```
commandCodeBlocks:
  roundParenthesesAllowed: 0
```

Notice the difference between Listing 151 and Listing 152; in particular, in Listing 152, because round parentheses are *not* allowed, `latexindent.pl` finds that the `\defFunction` command finishes at the first opening round parenthesis. As such, the remaining braced, mandatory, arguments are found to be `UnNamedGroupingBracesBrackets` (see Table 2 on page 20) which, by default, assume indentation for their body, and hence the tabbed indentation in Listing 152.

Let's explore this using the YAML given in Listing 155 and run the command

```
cmh:~$ latexindent.pl pstricks1.tex -l defFunction.yaml
```

then the output is as in Listing 154.

LISTING 154: pstricks1.tex using Listing 155

```
\defFunction[algebraic]{torus}(u,v)
└{(2+cos(u))*cos(v+\Pi)}
└{(2+cos(u))*sin(v+\Pi)}
└{sin(u)}
```

LISTING 155: defFunction.yaml

```
indentRules:
  defFunction:
    body: " "
```

Notice in Listing 154 that the *body* of the `defFunction` command i.e., the subsequent lines containing arguments after the command name, have received the single space of indentation specified by Listing 155.

```
stringsAllowedBetweenArguments: {fields}
```

`tikz` users may well specify code such as that given in Listing 156; processing this code using `latexindent.pl` gives the default output in Listing 157.



LISTING 156: tikz-node1.tex

```
\draw[thin]
(c)_to[in=110,out=-90]
++(0,-0.5cm)
node[below,align=left,scale=0.5]
```

LISTING 157: tikz-node1 default output

```
\draw[thin]
(c)_to[in=110,out=-90]
++(0,-0.5cm)
node[below,align=left,scale=0.5]
```

With reference to Listing 149 on page 37, we see that the strings

to, node, ++

are all allowed to appear between arguments, as they are each set to 1; importantly, you are encouraged to add further names to this field as necessary. This means that when `latexindent.pl` processes Listing 156, it consumes:

- the optional argument `[thin]`
- the round-bracketed argument `(c)` because `roundParenthesesAllowed` is 1 by default
- the string `to` (specified in `stringsAllowedBetweenArguments`)
- the optional argument `[in=110,out=-90]`
- the string `++` (specified in `stringsAllowedBetweenArguments`)
- the round-bracketed argument `(0,-0.5cm)` because `roundParenthesesAllowed` is 1 by default
- the string `node` (specified in `stringsAllowedBetweenArguments`)
- the optional argument `[below,align=left,scale=0.5]`

We can explore this further, for example using Listing 159 and running the command

```
cmh:~$ latexindent.pl tikz-node1.tex -l draw.yaml
```

we receive the output given in Listing 158.

LISTING 158: tikz-node1.tex using Listing 159

```
\draw[thin]
  (c)_to[in=110,out=-90]
  ++(0,-0.5cm)
  node[below,align=left,scale=0.5]
```

LISTING 159: draw.yaml

```
indentRules:
  draw:
    body: " "
```

Notice that each line after the `\draw` command (its ‘body’) in Listing 158 has been given the appropriate two-spaces worth of indentation specified in Listing 159.

Let’s compare this with the output from using the YAML settings in Listing 161, and running the command

```
cmh:~$ latexindent.pl tikz-node1.tex -l no-to.yaml
```

given in Listing 160.

LISTING 160: tikz-node1.tex using Listing 161

```
\draw[thin]
(c)_to[in=110,out=-90]
++(0,-0.5cm)
node[below,align=left,scale=0.5]
```

LISTING 161: no-to.yaml

```
commandCodeBlocks:
  stringsAllowedBetweenArguments:
    to: 0
```

In this case, `latexindent.pl` sees that:



- the `\draw` command finishes after the (c) as (stringsAllowedBetweenArguments has to set to 0)
- it finds a namedGroupingBracesBrackets called to (see Table 2 on page 20) with argument `[in=110,out=-90]`
- it finds another namedGroupingBracesBrackets but this time called node with argument `[below,align=left,scale=0.5]`

6 The -m (modifylinebreaks) switch

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

`modifylinebreaks: <fields>`

As of Version 3.0, `latexindent.pl` has the `-m` switch, which permits `latexindent.pl` to modify line breaks, according to the specifications in the `modifyLineBreaks` field. *The settings in this field will only be considered if the `-m` switch has been used.* A snippet of the default settings of this field is shown in Listing 162.

LISTING 162: `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.

`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 163 shows a sample file with blank lines; upon running

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

the output is shown in Listing 164; note that the multiple blank lines have been condensed into one blank line, and note also that we have used the `-m` switch!



LISTING 163: mlb1.tex

```
before_blank_line
```

```
after_blank_line
```

```
after_blank_line
```

```
textWrapOptions: {fields}
```

LISTING 164: mlb1.tex out output

```
before_blank_line
```

```
after_blank_line
```

```
after_blank_line
```

When the `-m` switch is active `latexindent.pl` has the ability to wrap text using the options specified in the `textWrapOptions` field, see Listing 165. The value of `columns` specifies the column at which the text should be wrapped. By default, the value of `columns` is 0, so `latexindent.pl` will *not* wrap text; if you change it to a value of 2 or more, then text will be wrapped after the character in the specified column.

For example, consider the file give in Listing 166.

LISTING 165: textWrapOptions

```
textWrapOptions:
  columns: 0
```

LISTING 166: textwrap1.tex

```
Here_is_a_line_of_text_that_will_be_wrapped_by_latexindent.pl.Each_line_is_quite_long.
```

```
Here_is_a_line_of_text_that_will_be_wrapped_by_latexindent.pl.Each_line_is_quite_long.
```

Using the file `textwrap1.yaml` in Listing 168, and running the command

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

we obtain the output in Listing 167.

LISTING 167: textwrap1-mod1.tex

```
Here_is_a_line_of
text_that_will_be
wrapped_by
latexindent.pl.
Each_line_is_quite
long.
```

```
Here_is_a_line_of
text_that_will_be
wrapped_by
latexindent.pl.
Each_line_is_quite
long.
```

LISTING 168: textwrap1.yaml

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

The text wrapping routine is performed *after* verbatim environments have been stored, so verbatim environments and verbatim commands are exempt from the routine. For example, using the file in Listing 169,



LISTING 169: textwrap2.tex

Here is a line of text that will be wrapped by latexindent.pl. Each line is quite long.

```
\begin{verbatim}
      a long line in a verbatim environment, which will not be broken by latexindent.pl
\end{verbatim}
```

Here is a verb command: `\verb!this will not be text wrapped!`

and running the following command and continuing to use textwrap1.yaml from Listing 168,

```
cmh:~$ latexindent.pl -m textwrap2.tex -o textwrap2-mod1.tex -l textwrap1.yaml
```

then the output is as in Listing 170.

LISTING 170: textwrap2-mod1.tex

Here is a line of
text that will be
wrapped by
latexindent.pl.
Each line is quite
long.

```
\begin{verbatim}
      a long line in a verbatim environment, which will not be broken by latexindent.pl
\end{verbatim}
```

Here is a verb
command:
`\verb!this will not be text wrapped!`

Furthermore, the text wrapping routine is performed after the trailing comments have been stored, and they are also exempt from text wrapping. For example, using the file in Listing 171

LISTING 171: textwrap3.tex

Here is a line of text that will be wrapped by latexindent.pl. Each line is quite long.

Here is a line% text wrapping does not apply to comments by latexindent.pl

and running the following command and continuing to use textwrap1.yaml from Listing 168,

```
cmh:~$ latexindent.pl -m textwrap3.tex -o textwrap3-mod1.tex -l textwrap1.yaml
```

then the output is as in Listing 172.

LISTING 172: textwrap3-mod1.tex

Here is a line of
text that will be
wrapped by
latexindent.pl.
Each line is quite
long.

Here is a line
% text wrapping does not apply to comments by latexindent.pl



The text wrapping routine of `latexindent.pl` is performed by the `Text::Wrap` module, which provides a `separator` feature to separate lines with characters other than a new line (see [10]). By default, the separator is empty (see Listing 173), but you can change it as you see fit.

LISTING 173: `textWrapOptions`

```
textWrapOptions:
  columns: 0
  separator: ""
```

For example starting with the file in Listing 174

LISTING 174: `textwrap4.tex`

```
Here is a line of text.
```

and using `textwrap2.yaml` from Listing 176 with the following command

```
cmh:~$ latexindent.pl -m textwrap4.tex -o textwrap4-mod2.tex -l textwrap2.yaml
```

then we obtain the output in Listing 175.

LISTING 175: `textwrap4-mod2.tex`

```
Here||is_a||line||of||text||.
```

LISTING 176: `textwrap2.yaml`

```
modifyLineBreaks:
  textWrapOptions:
    columns: 5
    separator: "||"
```

Summary of text wrapping It is important to note the following:

- Verbatim environments (Listing 13 on page 13) and verbatim commands (Listing 14 on page 13) will *not* be affected by the text wrapping routine (see Listing 170 on page 42);
- comments will *not* be affected by the text wrapping routine (see Listing 172 on page 42);
- indentation is performed *after* the text wrapping routine; as such, indented code will likely exceed any maximum value set in the `columns` field.

```
removeParagraphLineBreaks: <fields>
```

When the `-m` switch is active `latexindent.pl` has the ability to remove line breaks from within paragraphs; the behaviour is controlled by the `removeParagraphLineBreaks` field, detailed in Listing 177.

LISTING 177: `removeParagraphLineBreaks`

```
378 removeParagraphLineBreaks:
379   all: 0
380   environments:
381     quotation: 0
382     ifElseFi: 0
383     optionalArguments: 0
384     mandatoryArguments: 0
385     items: 0
386     specialBeginEnd: 0
387     masterDocument: 0
```

This routine can be turned on *globally* for *every* code block type known to `latexindent.pl` (see Table 2 on page 20) by using the `all` switch; by default, this switch is *off*. Assuming that the `all` switch is off, then the routine can be controlled on a per-code-block-type basis, and within that, on a per-name basis. We will consider examples of each of these in turn.



Let's start with the .tex file in Listing 178, together with the YAML settings in Listing 179.

LISTING 178: shortlines.tex

```
\begin{myenv}
The_lines
in_this
environment
are_very
short
and_contain
many_linebreaks.
```

```
Another
paragraph.
\end{myenv}
```

LISTING 179: remove-para1.yaml

-m

```
modifyLineBreaks:
  removeParagraphLineBreaks:
    all: 1
```

Upon running the command

```
cmh:~$ latexindent.pl -m shortlines.tex -o shortlines1.tex -l remove-para1.yaml
```

then we obtain the output given in Listing 180.

LISTING 180: shortlines1.tex

```
\begin{myenv}
  The_lines_in_this_environment_are_very_short_and_contain_many_linebreaks.

  Another_paragraph.
\end{myenv}
```

Keen readers may notice that some trailing white space must be present in the file in Listing 178 which has crept in to the output in Listing 180. This can be fixed using the YAML file in Listing 226 on page 52 and running, for example,

```
cmh:~$ latexindent.pl -m shortlines.tex -o shortlines1-tws.tex -l
  remove-para1.yaml,removeTWS-before.yaml
```

in which case the output is as in Listing 181; notice that the double spaces present in Listing 180 have been addressed.

LISTING 181: shortlines1-tws.tex

```
\begin{myenv}
  The_lines_in_this_environment_are_very_short_and_contain_many_linebreaks.

  Another_paragraph.
\end{myenv}
```

Keeping with the settings in Listing 179, we note that the all switch applies to *all* code block types. So, for example, let's consider the files in Listings 182 and 183



LISTING 182: shortlines-mand.tex

```
\mycommand{
The_lines
in_this
command
are_very
short
and_contain
many_linebreaks.

Another
paragraph.
}
```

LISTING 183: shortlines-opt.tex

```
\mycommand[
The_lines
in_this
command
are_very
short
and_contain
many_linebreaks.

Another
paragraph.
]
```

Upon running the commands

```
cmh:~$ latexindent.pl -m shortlines-mand.tex -o shortlines-mand1.tex -l remove-para1.yaml
cmh:~$ latexindent.pl -m shortlines-opt.tex -o shortlines-opt1.tex -l remove-para1.yaml
```

then we obtain the respective output given in Listings 184 and 185.

LISTING 184: shortlines-mand1.tex

```
\mycommand{
  ¶The_lines_in_this_command_are_very_short_and_contain_many_linebreaks.

  ¶Another_paragraph.
}
```

LISTING 185: shortlines-opt1.tex

```
\mycommand[
  ¶The_lines_in_this_command_are_very_short_and_contain_many_linebreaks.

  ¶Another_paragraph.
]
```

Assuming that we turn *off* the all switch (by setting it to 0), then we can control the behaviour of `removeParagraphLineBreaks` either on a per-code-block-type basis, or on a per-name basis.

For example, let's use the code in Listing 186, and consider the settings in Listings 187 and 188; note that in Listing 187 we specify that *every* environment should receive treatment from the routine, while in Listing 188 we specify that *only* the one environment should receive the treatment.



LISTING 186: shortlines-envs.tex

```

\begin{one}
The_lines
in_this
environment
are_very
short
and_contain
many_linebreaks.

Another
paragraph.
\end{one}

\begin{two}
The_lines
in_this
environment
are_very
short
and_contain
many_linebreaks.

Another
paragraph.
\end{two}

```

Upon running the commands

```

cmh:~$ latexindent.pl -m shortlines-envs.tex -o shortlines-envs2.tex -l remove-para2.yaml
cmh:~$ latexindent.pl -m shortlines-envs.tex -o shortlines-envs3.tex -l remove-para3.yaml

```

then we obtain the respective output given in Listings 189 and 190.

LISTING 189: shortlines-envs2.tex

```

\begin{one}
  ¶The_lines_in_this_environment_are_very_short_and_contain_many_linebreaks.

  ¶Another_paragraph.
\end{one}

\begin{two}
  ¶The_lines_in_this_environment_are_very_short_and_contain_many_linebreaks.

  ¶Another_paragraph.
\end{two}

```

LISTING 187: remove-para2.yaml

-m

```

modifyLineBreaks:
  removeParagraphLineBreaks:
    environments: 1

```

LISTING 188: remove-para3.yaml

-m

```

modifyLineBreaks:
  removeParagraphLineBreaks:
    environments:
      one: 1

```



LISTING 190: shortlines-envs3.tex

```

\begin{one}
  \The_lines_in_this_environment_are_very_short_and_contain_many_linebreaks.

  \Another_paragraph.
\end{one}

\begin{two}
  \The_lines
  \in_this
  \environment
  \are_very
  \short
  \and_contain
  \many_linebreaks.

  \Another
  \paragraph.
\end{two}

```

The remaining code-block types can be customised in analogous ways, although note that commands, `keyEqualsValuesBracesBrackets`, `namedGroupingBracesBrackets`, `UnNamedGroupingBracesBrackets` are controlled by the `optionalArguments` and the `mandatoryArguments`.

The only special case is the `masterDocument` field; this is designed for ‘chapter’-type files that may contain paragraphs that are not within any other code-blocks. For example, consider the file in Listing 191, with the YAML settings in Listing 192.

LISTING 191: shortlines-md.tex

```

The_lines
in_this
document
are_very
short
and_contain
many_linebreaks.

Another
paragraph.

\begin{myenv}
The_lines
in_this
document
are_very
short
and_contain
many_linebreaks.
\end{myenv}

```

LISTING 192: remove-para4.yaml

-m

```

modifyLineBreaks:
  removeParagraphLineBreaks:
    masterDocument: 1

```

Upon running the following command

```
cmh:~$ latexindent.pl -m shortlines-md.tex -o shortlines-md4.tex -l remove-para4.yaml
```

then we obtain the output in Listing 193.



LISTING 193: shortlines-md4.tex

```
The_lines_in_this_document_are_very_short_and_contain_many_linebreaks.
```

```
Another_paragraph.
```

```
\begin{myenv}
  %The_lines
  %in_this
  %document
  %are_very
  %short
  %and_contain
  %many_linebreaks.
\end{myenv}
```

6.1 Poly-switches

Every other field in the `modifyLineBreaks` field uses poly-switches, and can take one of four integer values⁸:

- 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.

6.2 modifyLineBreaks for environments

We start by viewing a snippet of `defaultSettings.yaml` in Listing 194; note that it contains *global* settings (immediately after the `environments` field) and that *per-name* settings are also allowed – in the case of Listing 194, settings for `equation*` have been specified. Note that all poly-switches are *off* by default.

LISTING 194: environments

```
388 environments:
389   BeginStartsOnOwnLine: 0
390   BodyStartsOnOwnLine: 0
391   EndStartsOnOwnLine: 0
392   EndFinishesWithLineBreak: 0
393   equation*:
394     BeginStartsOnOwnLine: 0
395     BodyStartsOnOwnLine: 0
396     EndStartsOnOwnLine: 0
397     EndFinishesWithLineBreak: 0
```

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

Let's begin with the simple example given in Listing 195; note that we have annotated key parts of the file using ♠, ♥, ♦ and ♣, these will be related to fields specified in Listing 194.

⁸You might like to associate one of the four circles in the logo with one of the four given values



LISTING 195: env-mlb1.tex

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

Let's explore BeginStartsOnOwnLine and BodyStartsOnOwnLine in Listings 196 and 197, and in particular, let's allow each of them in turn to take a value of 1.

LISTING 196: env-mlb1.yaml

-m

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

LISTING 197: env-mlb2.yaml

-m

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

After running the following commands,

```
cmh:~$ latexindent.pl -m env-mlb.tex -l env-mlb1.yaml
cmh:~$ latexindent.pl -m env-mlb.tex -l env-mlb2.yaml
```

the output is as in Listings 198 and 199 respectively.

LISTING 198: env-mlb.tex using Listing 196

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

LISTING 199: env-mlb.tex using Listing 197

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

There are a couple of points to note:

- in Listing 198 a line break has been added at the point denoted by ♠ in Listing 195; no other line breaks have been changed;
- in Listing 199 a line break has been added at the point denoted by ♥ in Listing 195; furthermore, note that the *body* of myenv has received the appropriate (default) indentation.

Let's now change each of the 1 values in Listings 196 and 197 so that they are 2 and save them into env-mlb3.yaml and env-mlb4.yaml respectively (see Listings 200 and 201).

LISTING 200: env-mlb3.yaml

-m

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

LISTING 201: env-mlb4.yaml

-m

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

Upon running commands analogous to the above, we obtain Listings 202 and 203.

LISTING 202: env-mlb.tex using Listing 200

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

LISTING 203: env-mlb.tex using Listing 201

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

Note that line breaks have been added as in Listings 198 and 199, 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 204 and 205, and in particular, let's allow each of them in turn to take a value of 1.

LISTING 204: env-mlb5.yaml

-m

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

LISTING 205: env-mlb6.yaml

-m

```
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 206 and 207.

LISTING 206: env-mlb.tex using Listing 204

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

LISTING 207: env-mlb.tex using Listing 205

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

There are a couple of points to note:

- in Listing 206 a line break has been added at the point denoted by ♦ in Listing 195 on page 49; no other line breaks have been changed and the `\end{myenv}` statement has *not* received indentation (as intended);
- in Listing 207 a line break has been added at the point denoted by ♣ in Listing 195 on page 49.

Let's now change each of the 1 values in Listings 204 and 205 so that they are 2 and save them into env-mlb7.yaml and env-mlb8.yaml respectively (see Listings 208 and 209).

LISTING 208: env-mlb7.yaml

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

LISTING 209: env-mlb8.yaml

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

Upon running commands analogous to the above, we obtain Listings 210 and 211.

LISTING 210: env-mlb.tex using Listing 208

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

LISTING 211: env-mlb.tex using Listing 209

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

Note that line breaks have been added as in Listings 206 and 207, 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 212 using any of the YAML files presented so far in this section, it will be left unchanged.

LISTING 212: env-mlb2.tex

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

LISTING 213: env-mlb3.tex

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

In contrast, the output from processing the file in Listing 213 will vary depending on the poly-switches used; in Listing 214 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 215 you'll see that the comment has been accounted for correctly because `BodyStartsOnOwnLine` has been set to 2, and the comment symbol has *not* been moved to its own line. You're encouraged to experiment with Listing 213 and by setting the other poly-switches considered so far to 2 in turn.



LISTING 214: env-mlb3.tex using
Listing 197 on page 49

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

LISTING 215: env-mlb3.tex using
Listing 201 on page 49

```
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 194 on page 48, an example is shown for the equation* environment.

6.2.2 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 216, noting in particular the positions of the line break highlighters, ♠, ♥, ♦ and ♣, together with the associated YAML files in Listings 217 to 220.

LISTING 216: env-mlb4.tex

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

LISTING 217: env-mlb9.yaml

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

LISTING 218: env-mlb10.yaml

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

LISTING 219: env-mlb11.yaml

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

LISTING 220: 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 respective output in Listings 221 to 224.

LISTING 221: env-mlb4.tex using
Listing 217

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

LISTING 222: env-mlb4.tex using
Listing 218

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



LISTING 223: env-mlb4.tex using Listing 219

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

LISTING 224: env-mlb4.tex using Listing 220

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

Notice that in

- Listing 221 the line break denoted by ♠ in Listing 216 has been removed;
- Listing 222 the line break denoted by ♥ in Listing 216 has been removed;
- Listing 223 the line break denoted by ♦ in Listing 216 has been removed;
- Listing 224 the line break denoted by ♣ in Listing 216 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 217 to 220 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 195 on page 49.

About trailing horizontal space Recall that on page 14 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 225, which highlights trailing spaces.

LISTING 225: env-mlb5.tex

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

LISTING 226:

```
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 227 and 228; note that the trailing horizontal white space has been preserved (by default) in Listing 227, while in Listing 228, it has been removed using the switch specified in Listing 226.

LISTING 227: env-mlb5.tex using Listings 221 to 224

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

LISTING 228: env-mlb5.tex using Listings 221 to 224 and Listing 226

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

Blank lines Now let's consider the file in Listing 229, which contains blank lines.



LISTING 229: env-mlb6.tex

```
before words♠

\begin{myenv}♡

body of myenv◇

\end{myenv}♣

after words
```

LISTING 230:
UnpreserveBlankLines.yaml

-m

```
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 231 and 232. In Listing 231 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 232, we have allowed the poly-switches to remove blank lines because, in Listing 230, we have set `preserveBlankLines` to 0.

LISTING 231: env-mlb6.tex
using Listings 221 to 224

```
before_words

\begin{myenv}

  body_of_myenv

\end{myenv}

after_words
```

LISTING 232: env-mlb6.tex using Listings 221 to 224 and Listing 230

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

6.3 Poly-switches for other code blocks

Rather than repeat the examples shown for the environment code blocks (in Section 6.2 on page 48), we choose to detail the poly-switches for all other code blocks in Table 3; note that each and every one of these poly-switches is *off by default*, i.e., set to 0. Note also that, by design, line breaks involving `verbatim`, `filecontents` and ‘comment-marked’ code blocks (Listing 31 on page 17) can *not* be modified using `latexindent.pl`.

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

Code block	Sample	Poly-switch mapping
environment	<pre>before words♠ \begin{myenv}♡ body of myenv◇ \end{myenv}♣ after words</pre>	<ul style="list-style-type: none"> ♠ <code>BeginStartsOnOwnLine</code> ♡ <code>BodyStartsOnOwnLine</code> ◇ <code>EndStartsOnOwnLine</code> ♣ <code>EndFinishesWithLineBreak</code>



ifelsefi	before words♠ \if...♥ body of if statement★ \else□ body of else statement◇ \fi♣ after words	♠ IfStartsOnOwnLine ♥ BodyStartsOnOwnLine ★ ElseStartsOnOwnLine □ ElseFinishesWithLineBreak ◇ FiStartsOnOwnLine ♣ FiFinishesWithLineBreak
optionalArguments	...♠ [♥ body of opt arg◇]♣ ...	♠ LSqBStartsOnOwnLine ⁹ ♥ OptArgBodyStartsOnOwnLine ◇ RSqBStartsOnOwnLine ♣ RSqBFinishesWithLineBreak
mandatoryArguments	...♠ {♥ body of mand arg◇ }♣ ...	♠ LCuBStartsOnOwnLine ¹⁰ ♥ MandArgBodyStartsOnOwnLine ◇ RCuBStartsOnOwnLine ♣ RCuBFinishesWithLineBreak
commands	before words♠ \mycommand♥ {arguments}	♠ CommandStartsOnOwnLine ♥ CommandNameFinishesWithLineBreak
namedGroupingBraces Brackets	before words♠ myname♥ {braces/brackets}	♠ NameStartsOnOwnLine ♥ NameFinishesWithLineBreak
keyEqualsValuesBraces Brackets	before words♠ key=♥ {braces/brackets}	♠ KeyStartsOnOwnLine • EqualsStartsOnOwnLine ♥ EqualsFinishesWithLineBreak
items	before words♠ \item♥ ...	♠ ItemStartsOnOwnLine ♥ ItemFinishesWithLineBreak

⁹LSqB stands for Left Square Bracket

¹⁰LCuB stands for Left Curly Brace

	before words♠	♠ SpecialBeginStartsOnOwnLine
	\[♥	♥ SpecialBodyStartsOnOwnLine
specialBeginEnd	body of special◇	◇ SpecialEndStartsOnOwnLine
	\]♣	♣ SpecialEndFinishesWithLineBreak
	after words	

6.4 Partnering BodyStartsOnOwnLine with argument-based poly-switches

Some poly-switches need to be partnered together; in particular, when line breaks involving the *first* argument of a code block need to be accounted for using both BodyStartsOnOwnLine (or its equivalent, see Table 3 on page 53) and LCUbStartsOnOwnLine for mandatory arguments, and LSqbStartsOnOwnLine for optional arguments.

Let’s begin with the code in Listing 242 and the YAML settings in Listing 244; with reference to Table 3 on page 53, the key CommandNameFinishesWithLineBreak is an alias for BodyStartsOnOwnLine.

LISTING 242: mycommand1.tex

```
\mycommand
{
mand_arg_text
mand_arg_text}
{
mand_arg_text
mand_arg_text}
```

Upon running the command

```
cmh:~$ latexindent.pl -m -l=mycom-mlb1.yaml mycommand1.tex
```

we obtain Listing 243; note that the *second* mandatory argument beginning brace { has had its leading line break removed, but that the *first* brace has not.

LISTING 243: mycommand1.tex
using Listing 244

```
\mycommand
{
  mand_arg_text
  mand_arg_text}{
  mand_arg_text
  mand_arg_text}
```

LISTING 244: mycom-mlb1.yaml -m

```
modifyLineBreaks:
  commands:
    CommandNameFinishesWithLineBreak: 0
  mandatoryArguments:
    LCUbStartsOnOwnLine: -1
```

Now let’s change the YAML file so that it is as in Listing 246; upon running the analogous command to that given above, we obtain Listing 245; both beginning braces { have had their leading line breaks removed.

LISTING 245: mycommand1.tex
using Listing 246

```
\mycommand{
  mand_arg_text
  mand_arg_text}{
  mand_arg_text
  mand_arg_text}
```

LISTING 246: mycom-mlb2.yaml -m

```
modifyLineBreaks:
  commands:
    CommandNameFinishesWithLineBreak: -1
  mandatoryArguments:
    LCUbStartsOnOwnLine: -1
```

Now let’s change the YAML file so that it is as in Listing 248; upon running the analogous command to that given above, we obtain Listing 247.



LISTING 247: mycommand1.tex
using Listing 248

```
\mycommand
{
  \mand_arg_text
  \mand_arg_text}
{
  \mand_arg_text
  \mand_arg_text}
```

LISTING 248: mycom-mlb3.yaml

-m

```
modifyLineBreaks:
  commands:
    CommandNameFinishesWithLineBreak: -1
  mandatoryArguments:
    LCuBStartsOnOwnLine: 1
```

6.5 Conflicting poly-switches: sequential code blocks

It is very easy to have conflicting poly-switches; if we use the example from Listing 242 on page 55, and consider the YAML settings given in Listing 250. The output from running

```
cmh:~$ latexindent.pl -m -l=mycom-mlb4.yaml mycommand1.tex
```

is given in Listing 250.

LISTING 249: mycommand1.tex
using Listing 250

```
\mycommand
{
  \mand_arg_text
  \mand_arg_text}{
  \mand_arg_text
  \mand_arg_text}
```

LISTING 250: mycom-mlb4.yaml

-m

```
modifyLineBreaks:
  mandatoryArguments:
    LCuBStartsOnOwnLine: -1
    RCuBFinishesWithLineBreak: 1
```

Studying Listing 250, we see that the two poly-switches are at opposition with one another:

- on the one hand, LCuBStartsOnOwnLine should *not* start on its own line (as poly-switch is set to -1);
- on the other hand, RCuBFinishesWithLineBreak *should* finish with a line break.

So, which should win the conflict? As demonstrated in Listing 249, it is clear that LCuBStartsOnOwnLine won this conflict, and the reason is that *the second argument was processed after the first* – in general, the most recently-processed code block and associated poly-switch takes priority.

We can explore this further by considering the YAML settings in Listing 252; upon running the command

```
cmh:~$ latexindent.pl -m -l=mycom-mlb5.yaml mycommand1.tex
```

we obtain the output given in Listing 251.

LISTING 251: mycommand1.tex
using Listing 252

```
\mycommand
{
  \mand_arg_text
  \mand_arg_text}
{
  \mand_arg_text
  \mand_arg_text}
```

LISTING 252: mycom-mlb5.yaml

-m

```
modifyLineBreaks:
  mandatoryArguments:
    LCuBStartsOnOwnLine: 1
    RCuBFinishesWithLineBreak: -1
```

As previously, the most-recently-processed code block takes priority – as before, the second (i.e., *last*) argument. Exploring this further, we consider the YAML settings in Listing 254, which give associated output in Listing 253.



LISTING 253: mycommand1.tex using Listing 254

```
\mycommand
{
  \mand_arg_text
  \mand_arg_text}%
{
  \mand_arg_text
  \mand_arg_text}
```

LISTING 254: mycom-mlb6.yaml

```
modifyLineBreaks:
  mandatoryArguments:
    LCuBStartsOnOwnLine: 2
    RCuBFinishesWithLineBreak: -1
```

Note that a % *has* been added to the trailing first `}`; this is because:

- while processing the *first* argument, the trailing line break has been removed (`RCuBFinishesWithLineBreak` set to `-1`);
- while processing the *second* argument, `latexindent.pl` finds that it does *not* begin on its own line, and so because `LCuBStartsOnOwnLine` is set to `2`, it adds a comment, followed by a line break.

6.6 Conflicting poly-switches: nested code blocks

Now let's consider an example when nested code blocks have conflicting poly-switches; we'll use the code in Listing 255, noting that it contains nested environments.

LISTING 255: nested-env.tex

```
\begin{one}
one_text
\begin{two}
two_text
\end{two}
\end{one}
```

Let's use the YAML settings given in Listing 257, which upon running the command

```
cmh:~$ latexindent.pl -m -l=nested-env-mlb1.yaml nested-env.tex
```

gives the output in Listing 256.

LISTING 256: nested-env.tex using Listing 256

```
\begin{one}
  \one_text
  \begin{two}
    \two_text\end{two}\end{one}
```

LISTING 257: nested-env-mlb1.yaml

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

In Listing 256, let's first of all note that both environments have received the appropriate (default) indentation; secondly, note that the poly-switch `EndStartsOnOwnLine` appears to have won the conflict, as `\end{one}` has had its leading line break removed.

To understand it, let's talk about the three basic phases of `latexindent.pl`:

1. Phase 1: packing, in which code blocks are replaced with unique ids, working from the *inside to the outside*, and then sequentially – for example, in Listing 255, the two environment is found *before* the one environment; if the `-m` switch is active, then during this phase:
 - line breaks at the beginning of the body can be added (if `BodyStartsOnOwnLine` is `1` or `2`) or removed (if `BodyStartsOnOwnLine` is `-1`);
 - line breaks at the end of the body can be added (if `EndStartsOnOwnLine` is `1` or `2`) or removed (if `EndStartsOnOwnLine` is `-1`);



- line breaks after the end statement can be added (if `EndFinishesWithLineBreak` is 1 or 2).
2. Phase 2: indentation, in which white space is added to the begin, body, and end statements;
 3. Phase 3: unpacking, in which unique ids are replaced by their *indented* code blocks; if the `-m` switch is active, then during this phase,
 - line breaks before begin statements can be added or removed (depending upon `BeginStartsOnOwnLine`);
 - line breaks after *end* statements can be removed but *NOT* added (see `EndFinishesWithLineBreak`).

With reference to Listing 256, this means that during Phase 1:

- the `two` environment is found first, and the line break ahead of the `\end{two}` statement is removed because `EndStartsOnOwnLine` is set to `-1`. Importantly, because, *at this stage*, `\end{two}` *does* finish with a line break, `EndFinishesWithLineBreak` causes no action.
- next, the `one` environment is found; the line break ahead of `\end{one}` is removed because `EndStartsOnOwnLine` is set to `-1`.

The indentation is done in Phase 2, and then in Phase 3, *there is no option to add a line break after the end statements*. We can justify this by remembering that during Phase 3, the `one` environment will be found and processed first, followed by the `two` environment. If the `two` environment were to add a line break after the `\end{two}` statement, then `latexindent.pl` would have no way of knowing how much indentation to add to the subsequent text (in this case, `\end{one}`).

We can explore this further using the poly-switches in Listing 259; upon running the command

```
cmh:~$ latexindent.pl -m -l=nested-env-mlb2.yaml nested-env.tex
```

we obtain the output given in Listing 258.

LISTING 258: `nested-env.tex` using Listing 259

```
\begin{one}
  \one_text
  \begin{two}
    \two_text
  \end{two}\end{one}
```

LISTING 259: `nested-env-mlb2.yaml`

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

During Phase 1:

- the `two` environment is found first, and the line break ahead of the `\end{two}` statement is not changed because `EndStartsOnOwnLine` is set to 1. Importantly, because, *at this stage*, `\end{two}` *does* finish with a line break, `EndFinishesWithLineBreak` causes no action.
- next, the `one` environment is found; the line break ahead of `\end{one}` is already present, and no action is needed.

The indentation is done in Phase 2, and then in Phase 3, the `one` environment is found and processed first, followed by the `two` environment. *At this stage*, the `two` environment finds `EndFinishesWithLineBreak` is `-1`, so it removes the trailing line break; remember, at this point, `latexindent.pl` has completely finished with the `one` environment.

7 Conclusions and known limitations

There are a number of known limitations of the script, and almost certainly quite a few that are *unknown*!

For example, `latexindent.pl` will not indent the following code correctly, because of the unmatched `[`. I'm hopeful to be able to resolve this issue in a future version.



```
\parbox{
\@ifnextchar [{\@assignmentwithcutoff}{\@assignmentnocutoff}
}
```

The main other limitation is to do with the alignment routine of environments/commands that contain delimiters which are specified in `lookForAlignDelims`.

The routine works well for ‘standard’ blocks of code that have the same number of & per line, but it will not do anything for lines that do not – such examples include `tabular` environments that use `\multicolumn` or perhaps spread cell contents across multiple lines. For each alignment block (`tabular`, `align`, etc) `latexindent.pl` first of all makes a record of the maximum number of &; if each row does not have that number of & then it will not try to format that row. Details will be given in `indent.log` assuming that trace mode is active.

You can run `latexindent` on `.sty`, `.cls` and any file types that you specify in `fileExtensionPreference` (see Listing 11 on page 12); if you find a case in which the script struggles, please feel free to report it at [6], and in the meantime, consider using a `noIndentBlock` (see page 14).

I hope that this script is useful to some; if you find an example where the script does not behave as you think it should, the best way to contact me is to report an issue on [6]; otherwise, feel free to find me on the <http://tex.stackexchange.com/users/6621/cmhughes>.

8 References

8.1 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] `Text::Wrap` Perl module. URL: <http://perldoc.perl.org/Text/Wrap.html> (visited on 05/01/2017).
- [11] Video demonstration of `latexindent.pl` on youtube. URL: <https://www.youtube.com/watch?v=wo38aaH2F4E&spfreload=10> (visited on 02/21/2017).

8.2 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).
- [12] 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 260 (`perl helloworld.pl`) then you will be able to run `latexindent.pl`, otherwise you may need to install the missing modules.



LISTING 260: helloworld.pl

```
#!/usr/bin/perl

use strict;
use warnings;
use utf8;
use PerlIO::encoding;
use Unicode::GCString;
use open ':std', ':encoding(UTF-8)';
use Text::Wrap;
use FindBin;
use YAML::Tiny;
use File::Copy;
use File::Basename;
use File::HomeDir;
use Getopt::Long;
use Data::Dumper;

print "hello␣world";
exit;
```

Installing the modules given in Listing 260 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 Updating the path variable

`latexindent.pl` has a few scripts (available at [6]) that can update the path variables¹¹. If you're on a Linux or Mac machine, then you'll want `CMakeLists.txt` from [6].

¹¹Thanks to [7] for this feature!



B.1 Add to path for Linux

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

1. download `latexindent.pl` and its associated modules, `defaultSettings.yaml`, to your chosen directory from [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;

4. 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
```

5. run

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

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

To remove the files, run

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

B.2 Add to path for Windows

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

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

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

3. right click on `add-to-path.bat` and *Run as administrator*;
4. log out, and log back in;
5. open a command prompt and run

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

to check that the appropriate directory has been added to your `%path%`.

To remove the directory from your `%path%`, run `remove-from-path.bat` as administrator.



C 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 any of the following, for example,

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

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

The fields given in Listing 261 are *obsolete* from Version 3.0 onwards.

LISTING 261: 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 262 and 263

LISTING 262:
`indentAfterThisHeading` in Version 2.2

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

LISTING 263:
`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 264; as of Version 3.0, you would write YAML as in Listing 265 or, if you're using `-m` switch, Listing 266.

LISTING 264: `noAdditionalIndent` in Version 2.2

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

LISTING 265: `noAdditionalIndent` for `displayMath` in Version 3.0

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

LISTING 266: `noAdditionalIndent` for `displayMath` in Version 3.0

```
noAdditionalIndent:
  displayMath: 1
```



End