latexindent.pl

Version 3.0

Chris Hughes

January 31, 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 modifiy line breaks, and add comment symbols. All user options are customisable via the switches in the YAML interface.

Contents

Listings

1 defaultSettings.yaml

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

and contributors! (See **??** on page ??.) For all communication, please visit [1].

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.

`fileExtensionPreference`: ⟨*fields*⟩

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

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

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

Listing 1: `fileExtensionPreference`

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

Calling `latexindent.pl myfile` with the (default) settings specified in Listing 1 means that the script will first look for `myfile.tex`, then `myfile.sty`, `myfile.cls`, and finally `myfile.bib` in order[1].

`backupExtension`: ⟨*extension name*⟩

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

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

`onlyOneBackUp`: ⟨*integer*⟩

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

`maxNumberOfBackUps`: ⟨*integer*⟩

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

`cycleThroughBackUps`: ⟨*integer*⟩

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

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

```
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 2. If you load your own user settings (see **??** on page ??) 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 2: `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 3.

LISTING 3: `verbatimEnvironments`

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

LISTING 4: `verbatimCommands`

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

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

`verbatimCommands`: ⟨*fields*⟩

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

`noIndentBlock`: ⟨*fields*⟩

If you have a block of code that you don't want `latexindent.pl` to touch (even if it is *not* a verbatim-like environment) then you can wrap it in an environment from `noIndentBlock`; you can use any name you like for this, provided you populate it as demonstrate in Listing 5.

LISTING 5: `noIndentBlock`

```
noIndentBlock:
    noindent: 1
    cmhtest: 1
```

Of course, you don't want to have to specify these as null environments in your code, so you use them with a comment symbol, `%`, followed by as many spaces (possibly none) as you like; see Listing 6 for example.

LISTING 6: 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 7; each of the fields can take the values 0 or 1. See **??????** on page ??, on page ?? and on page ?? for before and after results. Thanks to [2] for providing this feature.

LISTING 7: 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 8. The behaviour of latexindent.pl on these environments is determined by their location (preamble or not), and the value indentPreamble, discussed next.

LISTING 8: 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 9, if you set, for example, .tex to 0, then regardless of the setting of the value of indentPreamble, preamble will not be assumed when operating upon .tex files.

LISTING 9: lookForPreamble

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

preambleCommandsBeforeEnvironments: **0|1**

Assuming that latexindent.pl is asked to operate upon the preamble of a document, when this switch is set to 0 then environment code blocks will be sought first, and then command code blocks. When this switch is set to 1, commands will be sought first. The example that first motivated this switch contained the code given in Listing 10.

LISTING 10: Motivating `preambleCommandsBeforeEnvironments`

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

`defaultIndent`: ⟨*horizontal space*⟩

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

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

`lookForAlignDelims`: ⟨*fields*⟩

This contains a list of environments and/or commands that are operated upon in a special way by `latexindent.pl` (see Listing 11). In fact, the fields in `lookForAlignDelims` can actually take two different forms: the *basic* version is shown in Listing 11 and the *advanced* version in Listing 14; we will discuss each in turn.

LISTING 11: `lookForAlignDelims` (basic)

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

The environments specified in this field will be operated on in a special way by `latexindent.pl`. In particular, it will try and align each column by its alignment tabs. It does have some limitations (discussed further in **??**), but in many cases it will produce results such as those in Listings 12 and 13.

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 5).

LISTING 12: `tabular1.tex`

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

LISTING 13: `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 14 is for you.

LISTING 14: `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 14 `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 11 (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 14 saved in `tabular.yaml`, and the code from Listing 12 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 15 and 16; note that the ampersands have been aligned, but the `\\` have not (compare the alignment of `\\` in Listings 13 and 16).

LISTING 15: `tabular1.tex`

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

LISTING 16: `tabular1.tex` using Listing 14

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

Saving Listing 14 into `tabular1.yaml` as in Listing 18, and running the command

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

gives Listing 17; note the spacing before the `\\`.

LISTING 17: `tabular1.tex` using Listing 18

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

LISTING 18: `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); 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

FIX

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

then the before-and-after results shown in Listings 19 and 20 are achievable by default.

LISTING 19: matrix1.tex

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

LISTING 20: 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 21; the default output is shown in Listing 22. 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 21: align-block.tex

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

LISTING 22: matrix1.tex default output

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

With reference to Table 1 on page 9 and the, yet undiscussed, fields of noAdditionalIndent and indentRules (see **??** on page ??), 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 24 and 25

LISTING 23: indentAfterItems

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

LISTING 24: 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 25: items1.tex default output

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

itemNames: ⟨*fields*⟩

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

LISTING 26: itemNames

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

specialBeginEnd: ⟨*fields*⟩

The fields specified in specialBeginEnd are, in their default state, focused on math mode begin and end statements, but there is no requirement for this to be the case; Listing 27 shows the default settings of `specialBeginEnd`.

LISTING 27: `specialBeginEnd`

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

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

A demonstration of the before-and-after results are shown in Listings 28 and 29.

LISTING 28: `special1.tex` before

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

LISTING 29: `special1.tex` after

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

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

`indentAfterHeadings`: ⟨*fields*⟩

This field enables the user to specify indentation rules that take effect after heading commands such as `\part`, `\chapter`, `\section`, `\subsection*`, or indeed any user-specified command written in this field.[2]

LISTING 30: `indentAfterHeadings`

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

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

[2]There is a slight difference in interface for this field when comparing Version 2.2 to Version 3.0; see **??** on page ?? for details.

with `level: 3` because you do not want the indentation to go too deep.

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

FIX

For example, assuming that you have read **??** on page ??, say that you have the code in Listing 31 saved into `headings1.yaml`, and that you have the text from Listing 32 saved into `headings1.tex`.

LISTING 31: `headings1.yaml`

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

LISTING 32: `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 33.

LISTING 33: `headings1.tex` using Listing 31

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

LISTING 34: `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 31 so that the `paragraph level` is 1; after running

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

you should now receive the code given in Listing 34; notice that the `paragraph` and `subsection` are at the same indentation level.

1.1 The code blocks known `latexindent.pl`

As of Version 3.0, `latexindent.pl` processes documents using code blocks; each of these are shown in Table 1.

TABLE 1: Code blocks known to `latexindent.pl`

Code block	characters allowed in name	example

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

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

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

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

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

items	User specified, see Listings 23 and 26 on page 7	`\begin{enumerate}`
		`  \item ...`
		`\end{enumerate}`

specialBeginEnd	User specified, see Listing 27 on page 8	`\[`
		`  ...`
		`\]`

afterHeading	User specified, see Listing 30 on page 8	`\chapter{title}`
		`  ...`
		`\section{title}`

filecontents	User specified, see Listing 8 on page 4	`\begin{filecontents}` `...` `\end{filecontents}`

We will refer to these code blocks in what follows.