indent.pl

Chris Hughes *

April 21, 2013

FIX

Abstract

`indent.pl` is a `Perl` script that indents `.tex` files according to an indentation scheme that the user can modify to suit their taste. Environments, including those with alignment delimiters (such as `tabular`), commands, including those that can split braces and brackets across lines, are *usually* handled correctly by the script.

Contents

Listings

1 Before we begin

1.1 Thanks

I first created `indent.pl` to help me format chapter files in a big project. After I blogged about it on the TEX stack exchange [**cmhblog**] I received some positive feedback and follow-up feature requests. A big thank you to Harish Kumar who has really helped to drive the script forward and has put it through a number of challenging tests– I look forward to more challenges in the future Harish!

The `yaml`-based interface of `indent.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 `indent.pl`, but the release of `arara` has meant there is no need. Thank you to Paulo for all of your advice and encouragement.

1.2 License

*christopher dot michael dot hughes at gmail dot com

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 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 that come with it in the `samples` directory.

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.

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

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

LISTING 1: `filecontents` before

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

LISTING 2: `filecontents` after

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

LISTING 3: `tikzset` before

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

LISTING 4: `tikzset` after

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

LISTING 5: pstricks before

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

LISTING 6: pstricks after

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

3 How to use the script

There are two ways to use `indent.pl`: from the command line, and using `arara`. We will discuss how to change the settings and behaviour of the script in Section 4.

3.1 Executable files

`indent.pl` ships with a few standalone exectuble files:

- `indent.exe` for Windows users
- `indent` for Linux users

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

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

3.2 From the command line

`indent.pl` has a number of different switches/flags/options, which can be combined in any way that you like. `indent.pl` produces a `.log` file, `indent.log` every time it is run. 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.

`indent.pl`

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

`-h` `indent.pl -h`

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

`indent.pl myfile.tex`

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

`-w` `indent.pl -w myfile.tex`

This *will* overwrite `myfile.tex`, but it will make a copy of `myfile.tex` first. You can control the name of the extension (default is `.bak`), and how many different backups are made– more on this in Section 4; see `backupExtension` and `onlyOneBackUp`.

Note that if `indent.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` `indent.pl -o myfile.tex outputfile.tex`

This will indent `myfile.tex` and output it to `outputfile.tex`, overwriting it (`outputfile.tex`) if it already exists. Note that if `indent.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 `indent.pl myfile.tex > outputfile.tex`

`-s` `indent.pl -s myfile.tex`

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

`-t` `indent.pl -t myfile.tex`

Tracing mode: verbose output will be given to `indent.log`. This is useful if `indent.pl` has made a mistake and you're trying to find out where and why.

`-l` `indent.pl -l myfile.tex`

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

3.3 From `arara`

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

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

LISTING 7: `arara` sample usage

```
% arara: indent
% arara: indent: {overwrite: yes}
% arara: indent: {output: myfile.tex}
% arara: indent: {silent: yes}
% arara: indent: {trace: yes}
% arara: indent: {localSettings: yes}
\documentclass{article}
...
```

Hopefully the use of these rules is fairly self-explanatory, but for completeness Table 1 shows the relationship between `arara` orb-tags and the switches given in Section 3.2.

FIX

TABLE 1: `arara` orb tags and corresponding switches

`arara` orb tags	switch
`overwrite`	`-w`
`output`	`-o`
`silent`	`-s`
`trace`	`-t`
`localSettings`	`-l`

4 default, user, and local settings

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

4.1 `defaultSettings.yaml`

If you look in `defaultSettings.yaml` you'll find the switches that govern the behaviour of `indent.pl`. The code is commented, but here is a description of what each switch is designed to do. The default value is given in each case.

You can certainly feel free to edit `defaultSettings.yaml`, but this is not ideal as it may be overwritten when you update your distribution– all of your hard work tweaking the script would be undone! Don't worry, there's a solution– feel free to peek ahead to Section 4.2 if you like.

`defaultIndent "\t"`

This is the default indentation (`\t` means a tab) used in the absence of other details for the command or environment we are working with– see `indentRules` for more details (page 8).

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

`backupExtension .bak`

If you call `indent.pl` with the `-w` switch (to overwrite `myfile.tex`) then it will create a backup file before doing any indentation: `myfile.bak0`

By default, every time you call `indent.pl` after this with the `-w` switch it will create `myfile.bak1`, `myfile.bak2`, etc.

`onlyOneBackUp 0`

If you don't want a backup for every time that you call `indent.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`.

`indentPreamble 0`

The preamble of a document can sometimes contain some trickier code for `indent.pl` to work with. By default, `indent.pl` won't try to operate on the preamble, but if you'd like it to try then change `indentPreamble` to `1`.

`alwaysLookforSplitBraces 1`

This switch tells `indent.pl` to look for commands that can split *braces* across lines, such as `parbox`, `tikzset`, etc. In older versions of `indent.pl` you had to specify each one in `checkunmatched`– this clearly became tedious, hence the introduction of `alwaysLookforSplitBraces`.

As long as you leave this switch on (set to 1) you don't need to specify which commands can split braces across lines– you can ignore the fields `checkunmatched` *and* `checkunmatchedELSE` *described later.*

`alwaysLookforSplitBrackets 1`

This switch tells `indent.pl` to look for commands that can split *brackets* across lines, such as `psSolid`, `pgfplotstabletypeset`, etc. In older versions of `indent.pl` you had to specify each one in `checkunmatchedbracket`– this clearly became tedious, hence the introduction of `alwaysLookforSplitBraces`.

As long as you leave this switch on (set to 1) you don't need to specify which commands can split brackets across lines– you can ignore `checkunmatchedbracket` *described later.*

`lookForAlignDelims` This is the first example of a field in `defaultSettings.yaml` that has more than one line; listing 8 shows more details.

Listing 8: `lookForAlignDelims`

```
lookForAlignDelims:
   tabular: 1
   align: 1
   align*: 1
   alignat: 1
   alignat*: 1
   cases: 1
   dcases: 1
   aligned: 1
   pmatrix: 1
   listabla: 1
```

The environments specified in this field will be operated on in a special way by `indent.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 9 and 10.

Listing 9: `tabular` before

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

Listing 10: `tabular` after

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

If you find that `indent.pl` does not perform satisfactorily on such environments then you can either remove them from `lookForAlignDelims` altogether, or set the relevant key to `0`, for example `tabular: 0`, or if you just want to ignore *specific* instances of the environment, you could wrap them in something from `noIndentBlock` (see listing 12).

You can populate `lookForAlignDelims` with any other environments that you have that contain `&`, in any order that you wish. If you change your mind, just turn them off by setting them to `0` instead.

`verbatimEnvironments` A field that contains a list of environments that you would like left completely alone– no indentation will be done to environments that you have specified in this field– see listing 11.

Listing 11: `verbatimEnvironments`

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

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

`noIndentBlock` If you have a block of code that you don't want `indent.pl` to touch (even if it is *not* a verbatim-like environment) then you can wrap it in an environment from `noIndentBlock`; you can use any name you like for this, provided you populate it as demonstrate in listing 12.

Listing 12: `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 13 for example.

Listing 13: `noIndentBlock` demonstration

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

`noAdditionalIndent` If you would prefer some of your environments or commands not to receive any additional indent, then populate `noAdditionalIndent`; see listing 14. Note that these environments will still receive the *current* level of indentation unless they belong to `verbatimEnvironments`, or `noIndentBlock`.

Listing 14: `noAdditionalIndent`

```
noAdditionalIndent:
    document: 1
    pccexample: 1
    pccdefinition: 1
    problem: 1
    exercises: 1
```

```
 7      pccsolution: 1
 8      foreach: 0
 9      widepage: 1
10      comment: 1
11      \[: 1
12      \]: 1
13      frame: 0
```

Note in particular from listing 14 that if you wish content within `\[` and `\]` to receive no additional content then you have to specify *both* as `1` (the default is `0`). If you do not specify both as the same value you may get some interesting results!

`indentRules` If you would prefer to specify individual rules for certain environments or commands, just populate `indentRules`; see listing 15

LISTING 15: `indentRules`

```
 1  indentRules:
 2     myenvironment: "\t\t"
 3     anotherenvironment: "\t\t\t\t"
 4     \[: "\t"
```

Note that in contrast to `noAdditionalIndent` you do *not* need to specify both `\[` and `\]` in this field.

If you put an environment in both `noAdditionalIndent` and in `indentRules`↪ then `indent.pl` will resolve the conflict by ignoring `indentRules` and prioritizing `noAdditionalIndent`. You will get a warning message in `indent.log`; note that you will only get one warning message per command or environment. Further discussion is given in Section 4.1.1.

`indentAfterHeadings` This field enables the user to specify indentation rules that take effect after heading commands such as `\part`, `\chapter`, `\section`, `\subsection*` etc. This field is slightly different from all of the fields that we have considered previously, because each element is itself a field which has two elements: `indent` and `level`.

LISTING 16: `indentAfterHeadings`

```
 1  indentAfterHeadings:
 2     part:
 3        indent: 0
 4        level: 1
 5     chapter:
 6        indent: 0
 7        level: 2
 8     section:
 9        indent: 0
10        level: 3
11     ...
```

The default settings do *not* place indentation after a heading– you can easily switch them on by changing `indent: 0` to `indent: 1`. The `level` field tells `indent.pl` the hierarchy of the heading structure in your document. You might, for example,

like to have both `section` and `subsection` set with `level: 3` because you do not want the indentation to go too deep.

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

The following fields are marked in red, as they are not necessary unless you wish to micro manage your indentation scheme.

`checkunmatched` Assuming you keep `alwaysLookforSplitBraces` set to `1` (which is the default) then you don't need to worry about `checkunmatched`.

Should you wish to deactivate `alwaysLookforSplitBraces` by setting it to `0`, then you can populate `checkunmatched` with commands that can split braces across lines– see listing 17.

Listing 17: `checkunmatched`

```
1  checkunmatched:
2      parbox: 1
3      vbox: 1
```

`checkunmatchedELSE` Similarly, assuming you keep `alwaysLookforSplitBraces` set to `1` (which is the default) then you don't need to worry about `checkunmatchedELSE`.

As in `checkunmatched`, should you wish to deactivate `alwaysLookforSplitBraces` by setting it to `0`, then you can populate `checkunmatchedELSE` with commands that can split braces across lines *and* have an 'else' statement– see listing 18.

Listing 18: `checkunmatchedELSE`

```
1  checkunmatchedELSE:
2      pgfkeysifdefined: 1
3      DTLforeach: 1
4      ifthenelse: 1
```

`checkunmatchedbracket` Assuming you keep `alwaysLookforSplitBrackets` set to `1` (which is the default) then you don't need to worry about `checkunmatchedbracket`.

Should you wish to deactivate `alwaysLookforSplitBrackets` by setting it to `0`, then you can populate `checkunmatchedbracket` with commands that can split *brackets* across lines– see listing 19.

Listing 19: `checkunmatchedbracket`

```
1  checkunmatchedbracket:
2      psSolid: 1
3      pgfplotstablecreatecol: 1
4      pgfplotstablesave: 1
5      pgfplotstabletypeset: 1
6      mycommand: 1
```

4.1.1 Hierachy of fields

After reading the previous section, it should sound reasonable that `noAdditionalIndent`↪ , `indentRules`, and `verbatim` all serve mutually exclusive tasks. Naturally, you may well wonder what happens if you choose to ask `indent.pl` to prioritize one above the other.

For example, let's say that you put the fields in listing 20 into one of your settings files.

Listing 20: Conflicting ideas

```
1  indentRules:
2     myenvironment: "\t\t"
3  noAdditionalIndent:
4     myenvironment: 1
```

Clearly these fields conflict: first of all you are telling `indent.pl` that `myenvironment`↪ should receive two tabs of indentation, and then afterwards you are telling it not to put any indentation in the environment. `indent.pl` will always make the decision to prioritize `noAdditionalIndent` above `indentRules` regardless of the order that you load them in your settings file. The first time it encounters `myenvironment` it will put a warning in `indent.log` and delete the offending key from `indentRules` so that any future conflicts won't have to be addressed.

Let's consider another conflicting example in listing 21

Listing 21: More conflicting ideas

```
1  lookForAlignDelims:
2     myenvironment: 1
3  verbatimEnvironments:
4     myenvironment: 1
```

This is quite a significant conflict– we are first of all telling `indent.pl` to look for alignment delimiters in `myenvironment` and then telling it that actually we would like `myenvironment` to be considered as a `verbatim`-like environment. Regardless of the order that we state listing 21 the `verbatim` instruction will always win. As in listing 20 you will only receive a warning in `indent.log` the first time `indent.pl` encounters `myenvironment` as the offending key is deleted from `lookForAlignDelims`.

To summarize, `indent.pl` will prioritize the various fields in the following order:

1. `verbatimEnvironments`
2. `noAdditionalIndent`
3. `indentRules`

4.2 `indentconfig.yaml` (for user settings)

Editing `defaultSettings.yaml` is not ideal as it may be overwritten when updating your distribution– a better way to customize the settings to your liking is to set up your own settings file, `mysettings.yaml` (or any name you like, provided it ends with `.yaml`). The only thing you have to do is tell `indent.pl` where to find it.

`indent.pl` will always check your home directory for `indentconfig.yaml`, which is a plain text file you can create that contains the *absolute* paths for any settings files that you wish `indent.pl` to load. Note that Mac and Linux users home directory is

~/username while Windows (Vista onwards) is C:\Users\username. Listing 22 shows a sample indentconfig.yaml file.

Listing 22: indentconfig.yaml

```
# Paths to user settings for indent.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 your 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 feel free to start changing the switches and adding more environments to it as you see fit– have a look at listing 23 for an example that uses four tabs for the default indent, and adds the tabbing environment to the list of environments that contains alignment delimiters.

Listing 23: 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 indent.pl doesn't recognize then you'll get a warning, otherwise you'll get confirmation that indent.pl has read your settings file.

FIX

When editing .yaml files it is *extremely* important to remember how sensitive they are to spaces. I highly recommend copying and pasting from defaultSettings.↪yaml when you create your first whatevernameyoulike.yaml file.

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

4.3 localSettings.yaml

You may remember on page 4 we discussed the -l switch that tells indent.pl to look for localSettings.yaml in the *same directory* as myfile.tex. This settings file will be

read *after* `defaultSettings.yaml` and, assuming they exist, user settings.

In contrast to the *user* settings which can be named anything you like (provided that they are detailed in `indentconfig.yaml`), the *local* settings file must be called `localSettings.yaml`. It can contain any switches that you'd like to change– a sample is shown in **??**.

LISTING 24: `localSettings.yaml` (example)

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

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

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

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

4.4 Settings load order

`indent.pl` loads the settings files in the following order:

1. `defaultSettings.yaml` (always loaded, can not be renamed)
2. `anyUserSettings.yaml` (and any other arbitrarily-named files specified in `indentconfig.yaml`)
3. `localSettings.yaml` (if found in same directory as `myfile.tex` and called with `-l` switch; can not be renamed)

5 Known limitations

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

The main limitation is to do with the alignment routine of environments that contain delimiters– in other words, environments that are entered in `lookForAlignDelims`. Indeed, this is the only part of the script that can *potentially* remove lines from `myfile.tex`. Note that `indent.log` will always finish with a comparison of line counts before and after.

The routine works well for 'standard' blocks of code that have the same number of `&` per line, but it will not do anything for blocks that do not– such examples include `tabular` environments that use `\multicolumn` or perhaps spread cell contents across multiple lines.

Nested environments that contain alignment delimiters will not be formatted using the alignment-delimeter routine. For each alignment block (`tabular`, `align`, etc) `indent.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.

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, feel free to e-mail me or else come and find me on the http://tex.stackexchange.com site; I'm often around and in the chat room.

A Required Perl modules

If you intend to use indent.pl and *not* one of the supplied standalone exectutable files, then you will need a few standard modules– if you can run the minimum code in **??** (perl helloworld.pl) then you will be able to run indent.pl, otherwise you may need to install the missing modules.

Listing 25: helloworld.pl

```
1  #!/usr/bin/perl
2
3  use strict;
4  use warnings;
5  use FindBin;
6  use YAML::Tiny;
7  use File::Copy;
8  use File::Basename;
9  use Getopt::Std;
10 use File::HomeDir;
11
12 print "hello␣world";
13 exit;
```

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

Installing the modules given in **??** will vary depending on your operating system and Perl distribution. For example, Ubuntu users might visit the software center, and Strawberry Perl users on Windows might use CPAN client. All of the modules are readily available on CPAN [**cpan**].

B The arara rule

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

Listing 26: The arara rule

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

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

LISTING 27: The `arara` rule (modified)

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

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

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