Indent

Format C Code

Edition 0.02, for Indent Version 1.3
May 1992

Jim Kingdon
Joseph Arceneaux

1 The `indent` Program

The `indent` program changes the appearance of a C program by inserting or deleting whitespace. It can be used to make code easier to read. It can also convert from one style of writing C to another.

`indent` understands a substantial amount about the syntax of C, but it also attempts to cope with incomplete and misformed syntax.

In version 1.2 and more recent versions, the GNU style of indenting is the default.

1.1 Invoking `indent`

As of version 1.3, the format of the `indent` command is:

`indent` [*options*] [*input-files*]

`indent` [*options*] [*single-input-file*] [`-o` *output-file*]

This format is different from earlier versions and other versions of `indent`.

In the first form, one or more input files are specified. `indent` makes a backup copy of each file, and the original file is replaced with its indented version. See Section 1.2 [Backup files], page 3, for an explanation of how backups are made.

In the second form, only one input file is specified. In this case, or when the standard input is used, you may specify an output file after the '`-o`' option.

To cause `indent` to write to standard output, use the '`-st`' option. This is only allowed when there is only one input file, or when the standard input is used.

If no input files are named, the standard input is read for input. Also, if a filename named '`-`' is specified, then the standard input is read.

As an example, each of the following commands will input the program '`slithy_toves.c`' and write its indented text to '`slithy_toves.out`':

Command line switches are handled *after* processing '.indent.pro'. Options specified later override arguments specified earlier, with one exception: Explicitly specified options always override background options (see Section 1.3 [Common styles], page 4). You can prevent indent from reading an '.indent.pro' file by specifying the '-npro' option.

1.2 Backup Files

As of version 1.3, GNU indent makes GNU–style backup files, the same way GNU Emacs does. This means that either *simple* or *numbered* backup filenames may be made.

Simple backup file names are generated by appending a suffix to the original file name. The default for the this suffix is the one-character string '~' (tilde). Thus, the backup file for 'python.c' would be 'python.c~'.

Instead of the default, you may specify any string as a suffix by setting the environment variable SIMPLE_BACKUP_SUFFIX to your preferred suffix.

Numbered backup versions of a file 'momewraths' look like 'momewraths.c.~23~', where 23 is the version of this particular backup. When making a numbered backup of the file 'src/momewrath.c', the backup file will be named 'src/momewrath.c.~*V*~', where *V* is one greater than the highest version currently existing in the directory 'src'.

The type of backup file made is controlled by the value of the environment variable VERSION_CONTROL. If it is the string 'simple', then only simple backups will be made. If its value is the string 'numbered', then numbered backups will be made. If its value is 'numbered-existing', then numbered backups will be made if there *already exist* numbered backups for the file being indented; otherwise, a simple backup is made. If VERSION_CONTROL is not set, then indent assumes the behaviour of 'numbered-existing'.

Other versions of indent use the suffix '.BAK' in naming backup files. This behaviour can be emulated by setting SIMPLE_BACKUP_SUFFIX to '.BAK'.

Note also that other versions of indent make backups in the current directory, rather than in the directory of the source file as GNU indent now does.

```
-nbap -nbad -nbbb -bc -br -c33 -cd33 -cdb -ce -ci4
-cli0 -cp33 -d4 -di16 -fc1 -fca -i4 -ip4 -l75 -lp
-npcs -psl -sc -nsob -nss -ts8
```

1.4 Blank lines

Various programming styles use blank lines in different places. `indent` has a number of options to insert or delete blank lines in specific places.

The '`-bad`' option causes `indent` to force a blank line after every block of declarations. The '`-nbad`' option causes `indent` not to force such blank lines.

The '`-bap`' option forces a blank line after every procedure body. The '`-nbap`' option forces no such blank line.

The '`-bbb`' option forces a blank line before every block comment. A block comment is one which starts in column one when formatting of such comments is disabled, or one with `-` or `*` immediately following the `/*`. The '`-nbbb`' option does not force such blank lines.

The '`-sob`' option causes `indent` to swallow optional blank lines (that is, any optional blank lines present in the input will be removed from the output). If the '`-nsob`' is specified, any blank lines present in the input file will be copied to the output file.

For example, given the input

```
char *foo;
char *bar;
/* This separates blocks of declarations.  */
int baz;
```

`indent -bad` produces

```
int
foo ()
{
  puts ("Hi");
}

/* The procedure bar is even less interesting.  */
char *
bar ()
{
  puts ("Hello");
}
```

and `indent -nbap` produces

```
int
foo ()
{
  puts ("Hi");
}
/* The procedure bar is even less interesting.  */
char *
bar ()
{
  puts ("Hello");
}
```

No blank line will be added after the procedure `foo`.

1.5 Comments

Comments are no longer formatted by default as of version 1.2. This can be enabled with the '`-fca`' option. Doing so will cause newlines in the comment text to be ignored and the line will be filled up to the length of a line (which can be modified with '`-l`'). When formatting is enabled, blank lines indicate paragraph breaks.

The '`-fc1`' option enables the formatting of comments which begin in the first column. The '`-nfc1`' option disables the formatting of first column comments. When comment formatting is disabled, overall comment indentation may still be adjusted.

```
/* This is a comment which extends from one line
   onto the next line, thus causing us to consider
   how it should continue.  */
```

1.6 Statements

The '-br' or '-bl' option specifies how to format braces.

The '-br' option formats braces like this:

```
if (x > 0) {
  x--;
}
```

The '-bl' option formats them like this:

```
if (x > 0)
  {
    x--;
  }
```

If you use the '-bl' option, you may also want to specify the '-bli' option. This option specifies the number of spaces by which braces are indented. '-bli2', the default, gives the result shown above. '-bli0' results in the following:

```
if (x > 0)
{
  x--;
}
```

If you are using the '-br' option, you probably want to also use the '-ce' option. This causes the else in an if-then-else construct to cuddle up to the immediately preceding '}'. For example, with '-br -ce' you get the following:

```
int             foo;
char           *bar;
```

Using a small value (such as one or two) for the '-di' option can be used to cause the indentifiers to be placed in the first available position, for example

```
int foo;
char *bar;
```

The value given to the '-di' option will still affect variables which are put on separate lines from their types, for example '-di2' will lead to

```
int
  foo;
```

If the '-bc' option is specified, a newline is forced after each comma in a declaration. For example,

```
int a,
  b,
  c;
```

With the '-nbc' option this would look like

```
int a, b, c;
```

The '-psl' option causes the type of a procedure being defined to be placed on the line before the name of the procedure. This style is required for the etags program to work correctly, as well as some of the c-mode functions of Emacs.

`indent` assumes that tabs are placed at regular intervals of both input and output character streams. These intervals are by default 8 columns wide, but (as of version 1.2) may be changed by the '`-ts`' option. Tabs are treated as the equivalent number of spaces.

The indentation of type declarations in old-style function definitions is controlled by the '`-ip`' parameter. This is a numeric parameter specifying how many spaces to indent type declarations. For example, the default '`-ip5`' makes definitions look like this:

```
char *
create_world (x, y, scale)
     int x;
     int y;
     float scale;
{
  . . .
}
```

For compatibility with other versions of indent, the option '`-nip`' is provided, which is equivalent to '`-ip0`'.

1.9 Miscellaneous options

To find out what version of `indent` you have, use the command `indent -version`. This will report the version number of `indent`, without doing any of the normal processing.

The '`-v`' option can be used to turn on verbose mode. When in verbose mode, `indent` reports when it splits one line of input into two more more lines of output, and gives some size statistics at completion.

1.10 Copyright

The following copyright notice applies to the `indent` program. The copyright and copying permissions for this manual appear near the beginning of this document.

Appendix A Option Summary

Here is a list of all the options for `indent`, alphabetized by short option. It is followed by a cross key alphabetized by long option.

‘`-bad`’
‘`--blank-lines-after-declarations`’
: Force blank lines after the declarations.
See Section 1.4 [Blank lines], page 5.

‘`-bap`’
‘`--blank-lines-after-procedures`’
: Force blank lines after procedure bodies.
See Section 1.4 [Blank lines], page 5.

‘`-bbb`’
‘`--blank-lines-after-block-comments`’
: Force blank lines after block comments.
See Section 1.4 [Blank lines], page 5.

‘`-bc`’
‘`--blank-lines-after-commas`’
: Force newline after comma in declaration.
See Section 1.7 [Declarations], page 10.

‘`-bl`’
‘`--braces-after-if-line`’
: Put braces on line after `if`, etc.
See Section 1.6 [Statements], page 9.

‘`-bli`*n*’
‘`--brace-indent`*n*’
: Indent braces *n* spaces.
See Section 1.6 [Statements], page 9.

‘`-br`’
‘`--braces-on-if-line`’
: Put braces on line with `if`, etc.
See Section 1.6 [Statements], page 9.

‘`-c`*n*’
‘`--comment-indentation`*n*’
: Put comments to the right of code in column *n*.
See Section 1.5 [Comments], page 7.

'-di*n*'

'--declaration-indentation*n*'

Put variables in column *n*.
See Section 1.7 [Declarations], page 10.

'-fc1'

'--format-first-column-comments'

Format comments in the first column.
See Section 1.5 [Comments], page 7.

'-fca'

'--format-all-comments'

Do not disable all formatting of comments.
See Section 1.5 [Comments], page 7

'-gnu'

'--gnu-style'

Use GNU coding style. This is the default.
See Section 1.3 [Common styles], page 4.

'-i*n*'

'--indent-level*n*'

Set indentation level to *n* spaces.
See Section 1.8 [Indentation], page 12.

'-ip*n*'

'--parameter-indentation*n*'

Indent parameter types in old-style function definitions by *n* spaces.
See Section 1.8 [Indentation], page 12.

'-kr'

'--k-and-r-style'

Use Kernighan & Ritchie coding style.
See Section 1.3 [Common styles], page 4.

'-l*n*'

'--line-length*n*'

Set maximum line length to *n*.
See Section 1.5 [Comments], page 7.

'-lp'

'--continue-at-parentheses'

Line up continued lines at parentheses.
See Section 1.8 [Indentation], page 12.

'`-nip`'
'`--no-parameter-indentation`'

Zero width indentation for parameters.
See Section 1.8 [Indentation], page 12

'`-nlp`'
'`--dont-line-up-parentheses`'

Do not line up parentheses.
See Section 1.6 [Statements], page 9.

'`-npcs`'
'`--no-space-after-function-call-names`'

Do not put space after the function in function calls.
See Section 1.6 [Statements], page 9.

'`-npsl`'
'`--dont-break-procedure-type`'

Put the type of a procedure on the same line as its name.
See Section 1.7 [Declarations], page 10.

'`-nsc`'
'`--dont-star-comments`'

Do not put the '`*`' character at the left of comments.
See Section 1.5 [Comments], page 7.

'`-nsob`'
'`--leave-optional-blank-lines`'

Do not swallow optional blank lines.
See Section 1.4 [Blank lines], page 5.

'`-nss`'

'`--dont-space-special-semicolon`'

Do not force a space before the semicolon after certain statements. Disables '`-ss`'.
See Section 1.6 [Statements], page 9.

'`-nv`'
'`--no-verbosity`'

Disable verbose mode. See Section 1.9 [Miscellaneous options], page 13.

'`-orig`'
'`--original`'

Use the original Berkeley coding style.
See Section 1.3 [Common styles], page 4.

'-version'

Output the version number of `indent`.
See Section 1.9 [Miscellaneous options], page 13.

Options' Cross Key

Here is a list of options alphabetized by long option, to help you find the corresponding short option.

--blank-lines-after-block-comments	-bbb
--blank-lines-after-commas	-bc
--blank-lines-after-declarations	-bad
--blank-lines-after-procedures	-bap
--braces-after-if-line	-bl
--brace-indent	-bli*n*
--braces-on-if-line	-br
--case-indentation	-cli*n*
--comment-delimiters-on-blank-lines	-cdb
--comment-indentation	-c*n*
--continuation-indentation	-ci*n*
--continue-at-parentheses	-lp
--cuddle-else	-ce
--declaration-comment-column	-cd*n*
--declaration-indentation	-di*n*
--dont-break-procedure-type	-npsl
--dont-cuddle-else	-nce
--dont-format-comments	-nfca
--dont-format-first-column-comments	-nfc1
--dont-line-up-parentheses	-nlp
--dont-space-special-semicolon	-nss
--dont-star-comments	-nsc
--else-endif-column	-cp*n*
--format-all-comments	-fca
--format-first-column-comments	-fc1
--gnu-style	-gnu
--ignore-profile	-npro
--indent-level	-i*n*
--k-and-r-style	-kr

Index

(Index is nonexistent)

Table of Contents