Indent

Format C Code

Edition 2.2.5, for Indent Version 2.2.5
16 January 2000

Carlo Wood
Joseph Arceneaux
Jim Kingdon

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 2, 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`':

```
indent slithy_toves.c -o slithy_toves.out

indent -st slithy_toves.c > slithy_toves.out

cat slithy_toves.c | indent -o slithy_toves.out
```

Most other options to `indent` control how programs are formatted. As of version 1.2, `indent` also recognizes a long name for each option name. Long options are prefixed by either '`--`' or '`+`'.[1] In most of this document, the traditional, short names are used for the sake of brevity. See Appendix A [Option Summary], page 15, for a list of options, including both long and short names.

Here is another example:

[1] '`+`' is being superseded by '`--`' to maintain consistency with the POSIX standard.

```
indent -br test/metabolism.c -l85
```

This will indent the program 'test/metabolism.c' using the '-br' and '-l85' options, write the output back to 'test/metabolism.c', and write the original contents of 'test/metabolism.c' to a backup file in the directory 'test'.

Equivalent invocations using long option names for this example would be:

```
indent --braces-on-if-line --line-length185 test/metabolism.c

indent +braces-on-if-line +line-length185 test/metabolism.c
```

If you find that you often use indent with the same options, you may put those options into a file named '.indent.pro'. indent will first look for '.indent.pro' in the current directory and use that if found. Otherwise, indent will search your home directory for '.indent.pro' and use that file if it is found. This behaviour is different from that of other versions of indent, which load both files if they both exist.

The format of '.indent.pro' is simply a list of options, just as they would appear on the command line, separated by white space (tabs, spaces, and newlines). Options in '.indent.pro' may be surrounded by C or C++ comments, in which case they are ignored.

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 3). 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 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 'momeraths.c' look like 'momeraths.c.~23~', where 23 is the version of this particular backup. When making a numbered backup of the file 'src/momeraths.c', the backup file will be named 'src/momeraths.c.~*V*~', where *V* is one greater than the highest version currently existing in the directory 'src'. The environment variable VERSION_WIDTH controls the number of digits, using left zero padding when necessary. For instance, setting this variable to "2" will lead to the backup file being named 'momeraths.c.~04~'.

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.

1.3 Common styles

There are several common styles of C code, including the GNU style, the Kernighan & Ritchie style, and the original Berkeley style. A style may be selected with a single *background* option, which specifies a set of values for all other options. However, explicitly specified options always override options implied by a background option.

As of version 1.2, the default style of GNU `indent` is the GNU style. Thus, it is no longer necessary to specify the option '`-gnu`' to obtain this format, although doing so will not cause an error. Option settings which correspond to the GNU style are:

```
-nbad -bap -nbc -bbo -bl -bli2 -bls -ncdb -nce -cp1 -cs -di2
-ndj -nfc1 -nfca -hnl -i2 -ip5 -lp -pcs -nprs -psl -nsc -nsob
```

The GNU coding style is that preferred by the GNU project. It is the style that the GNU Emacs C mode encourages and which is used in the C portions of GNU Emacs. (People interested in writing programs for Project GNU should get a copy of *The GNU Coding Standards*, which also covers semantic and portability issues such as memory usage, the size of integers, etc.)

The Kernighan & Ritchie style is used throughout their well-known book *The C Programming Language*. It is enabled with the '`-kr`' option. The Kernighan & Ritchie style corresponds to the following set of options:

```
-nbad -bap -bbo -nbc -br -brs -c33 -cd33 -ncdb -ce -ci4 -cli0
-cp33 -cs -d0 -di1 -nfc1 -nfca -hnl -i4 -ip0 -l75 -lp -npcs
-nprs -npsl -nsc -nsob -nss
```

Kernighan & Ritchie style does not put comments to the right of code in the same column at all times (nor does it use only one space to the right of the code), so for this style `indent` has arbitrarily chosen column 33.

The style of the original Berkeley `indent` may be obtained by specifying '`-orig`' (or by specifying '`--original`', using the long option name). This style is equivalent to the following settings:

```
-nbad -nbap -bbo -bc -br -brs -c33 -cd33 -cdb -ce -ci4 -cli0
-cp33 -di16 -fc1 -fca -hnl -i4 -ip4 -l75 -lp -npcs -nprs -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 boxed comment, See Section 1.5 [Comments], page 5. 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

```
char *foo;
char *bar;

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

and `indent -nbad` produces

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

The '-bap' option forces a blank line after every procedure body.

For example, given the input

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

`indent -bap` 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

`indent` formats both C and C++ comments. C comments are begun with '`/*`', terminated with '`*/`' and may contain newline characters. C++ comments begin with the delimiter '`//`' and end at the newline.

`indent` handles comments differently depending upon their context. `indent` attempts to distinguish between comments which follow statements, comments which follow declarations, comments following preprocessor directives, and comments which are not preceded by code of any sort, i.e., they begin the text of the line (although not neccessarily in column 1).

`indent` further distinguishes between comments found outside of procedures and aggregates, and those found within them. In particular, comments beginning a line found within a procedure will be indented to the column at which code is currently indented. The exception to this a comment beginning in the leftmost column; such a comment is output at that column.

`indent` attempts to leave *boxed comments* unmodified. The general idea of such a comment is that it is enclosed in a rectangle or "box" of stars or dashes to visually set it apart. More precisely, boxed comments are defined as those in which the initial '`/*`' is followed immediately by the character '`*`', '`=`', '`_`', or '`-`', or those in which the beginning comment delimiter ('`/*`') is on a line by itself, and the following line begins with a '`*`' in the same column as the star of the opening delimiter.

Examples of boxed comments are:

```
/**********************
 * Comment in a box!! *
 **********************/

       /*
        * A different kind of scent,
        * for a different kind of comment.
        */
```

`indent` attempts to leave boxed comments exactly as they are found in the source file. Thus the indentation of the comment is unchanged, and its length is not checked in any way. The only alteration made is that an embedded tab character may be converted into the appropriate number of spaces.

If the '-bbb' option is specified, all such boxed comments will be preceded by a blank line, unless such a comment is preceded by code.

Comments which are not boxed comments may be formatted, which means that the line is broken to fit within a right margin and left-filled with whitespace. Single newlines are equivalent to a space, but blank lines (two or more newlines in a row) are taken to mean a paragraph break. Formatting of comments which begin after the first column is enabled with the '-fca' option. To format those beginning in column one, specify '-fc1'. Such formatting is disabled by default.

The right margin for formatting defaults to 78, but may be changed with the '-lc' option. If the margin specified does not allow the comment to be printed, the margin will be automatically extended for the duration of that comment. The margin is not respected if the comment is not being formatted.

If the comment begins a line (i.e., there is no program text to its left), it will be indented to the column it was found in unless the comment is within a block of code. In that case, such a comment will be aligned with the indented code of that block (unless the comment began in the first column). This alignment may be affected by the '-d' option, which specifies an amount by which such comments are moved to the *left*, or unindented. For example, '-d2' places comments two spaces to the left of code. By default, comments are aligned with code, unless they begin in the first column, in which case they are left there by default — to get them aligned with the code, specify '-fc1'.

Comments to the right of code will appear by default in column 33. This may be changed with one of three options. '-c' will specify the column for comments following code, '-cd' specifies the column for comments following declarations, and '-cp' specifies the column for comments following preprocessor directives `#else` and `#endif`.

If the code to the left of the comment exceeds the beginning column, the comment column will be extended to the next tabstop column past the end of the code, or in the case of preprocessor directives, to one space past the end of the directive. This extension lasts only for the output of that particular comment.

The '-cdb' option places the comment delimiters on blank lines. Thus, a single line comment like `/* Loving hug */` can be transformed into:

```
/*
   Loving hug
 */
```

Stars can be placed at the beginning of multi-line comments with the '-sc' option. Thus, the single-line comment above can be transformed (with '-cdb -sc') into:

```
/*
 * Loving hug
 */
```

1.6 Statements

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

The '-br' option formats statement 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:

```
if (x > 0) {
  x--;
} else {
  fprintf (stderr, "...something wrong?\n");
}
```

With '-br -nce' that code would appear as

```
if (x > 0) {
  x--;
}
else {
  fprintf (stderr, "...something wrong?\n");
}
```

The '-cli' option specifies the number of spaces that case labels should be indented to the right of the containing switch statement.

The default gives code like:

```
switch (i)
  {
  case 0:
    break;
  case 1:
    {
      ++i;
    }
  default:
    break;
  }
```

Using the '-cli2' that would become:

```
switch (i)
  {
    case 0:
      break;
    case 1:
      {
        ++i;
      }
    default:
      break;
  }
```

The indentation of the braces below a case statement can be controlled with the '-cbi*n*' option. For example, using '-cli2 -cbi0' results in:

```
switch (i)
  {
    case 0:
      break;
    case 1:
    {
      ++i;
    }
    default:
      break;
  }
```

If a semicolon is on the same line as a `for` or `while` statement, the '-ss' option will cause a space to be placed before the semicolon. This emphasizes the semicolon, making it clear that the body of the `for` or `while` statement is an empty statement. '-nss' disables this feature.

The '-pcs' option causes a space to be placed between the name of the procedure being called and the '(' (for example, `puts ("Hi");`. The '-npcs' option would give `puts("Hi");`).

If the '-cs' option is specified, `indent` puts a space after a cast operator.

The '-bs' option ensures that there is a space between the keyword `sizeof` and its argument. In some versions, this is known as the 'Bill_Shannon' option.

The '-prs' option causes all parentheses to be seperated with a space from the what is between them. For example, using '-prs' results in code like:

```
  while ( ( e_code - s_code ) < ( dec_ind - 1 ) )
    {
      set_buf_break ( bb_dec_ind );
      *e_code++ = ' ';
    }
```

1.7 Declarations

By default `indent` will line up identifiers, in the column specified by the '-di' option. For example, '-di16' makes things look like:

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

Using a small value (such as one or two) for the '-di' option can be used to cause the identifiers 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.

You must use the '-T' option to tell `indent` the name of all the typenames in your program that are defined by `typedef`. '-T' can be specified more than once, and all names specified are used. For example, if your program contains

```
typedef unsigned long CODE_ADDR;
typedef enum {red, blue, green} COLOR;
```

you would use the options '-T CODE_ADDR -T COLOR'.

The '-brs' or '-bls' option specifies how to format braces in struct declarations. The '-brs' option formats braces like this:

```
struct foo {
  int x;
};
```

The '-bls' option formats them like this:

```
struct foo
{
  int x;
};
```

1.8 Indentation

One issue in the formatting of code is how far each line should be indented from the left margin. When the beginning of a statement such as `if` or `for` is encountered, the indentation level is increased by the value specified by the '-i' option. For example, use '-i8' to specify an eight character indentation for each level. When a statement is broken across two lines, the second line is indented by a number of additional spaces specified by the '-ci' option. '-ci' defaults to 0. However, if the '-lp' option is specified, and a line has a left parenthesis which is not closed on that line, then continuation lines will be lined up to start at the character position just after the left parenthesis. This processing also applies to '[' and applies to '{' when it occurs in initialization lists. For example, a piece of continued code might look like this with '-nlp -ci3' in effect:

```
p1 = first_procedure (second_procedure (p2, p3),
   third_procedure (p4, p5));
```

With '-lp' in effect the code looks somewhat clearer:

```
p1 = first_procedure (second_procedure (p2, p3),
                      third_procedure (p4, p5));
```

When a statement is broken in between two or more paren pairs (...), each extra pair causes the indentation level extra indentation:

```
if ((((i < 2 &&
        k > 0) || p == 0) &&
    q == 1) ||
  n = 0)
```

The option '-ip*N*' can be used to set the extra offset per paren. For instance, '-ip0' would format the above as:

```
if ((((i < 2 &&
  k > 0) || p == 0) &&
  q == 1) ||
  n = 0)
```

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'.

ANSI C allows white space to be placed on preprocessor command lines between the character '#' and the command name. By default, indent removes this space, but specifying the '-lps' option directs indent to leave this space unmodified.

1.9 Breaking long lines

With the option '-l*n*', or '--line-length*n*', it is possible to specify the maximum length of a line of C code, not including possible comments that follow it.

When lines become longer then the specified line length, GNU `indent` tries to break the line at a logical place. This is new as of version 2.1 however and not very intelligent or flexible yet.

Currently there are two options that allows one to interfere with the algorithm that determines where to break a line.

The '`-bbo`' option causes GNU `indent` to prefer to break long lines before the boolean operators `&&` and `||`. The '`-nbbo`' option causes GNU `indent` not have that preference. For example, the default option '`-bbo`' (together with '`--line-length60`' and '`--ignore-newlines`') makes code look like this:

```
if (mask
    && ((mask[0] == '\0')
        || (mask[1] == '\0'
            && ((mask[0] == '0') || (mask[0] == '*')))))
```

Using the option '`-nbbo`' will make it look like this:

```
if (mask &&
    ((mask[0] == '\0') ||
     (mask[1] == '\0' &&
      ((mask[0] == '0') || (mask[0] == '*')))))
```

The default '`-hnl`', however, honours newlines in the input file by giving them the highest possible priority to break lines at. For example, when the input file looks like this:

```
if (mask
    && ((mask[0] == '\0')
    || (mask[1] == '\0' && ((mask[0] == '0') || (mask[0] == '*')))))
```

then using the option '`-hnl`', or '`--honour-newlines`', together with the previously mentioned '`-nbbo`' and '`--line-length60`', will cause the output not to be what is given in the last example but instead will prefer to break at the positions where the code was broken in the input file:

```
if (mask
    && ((mask[0] == '\0')
        || (mask[1] == '\0' &&
            ((mask[0] == '0') || (mask[0] == '*')))))
```

The idea behind this option is that lines which are too long, but are already broken up, will not be touched by GNU `indent`. Really messy code should be run through `indent` at least once using the '`--ignore-newlines`' option though.

1.10 Disabling Formatting

Formatting of C code may be disabled for portions of a program by embedding special *control comments* in the program. To turn off formatting for a section of a program, place the disabling control comment `/* *INDENT-OFF* */` on a line by itself just before that section. Program text scanned after this control comment is output precisely as input with no modifications until the corresponding enabling comment is scanned on a line by itself. The disabling control comment is `/* *INDENT-ON* */`, and any text following the comment on the line is also output unformatted. Formatting begins again with the input line following the enabling control comment.

More precisely, `indent` does not attempt to verify the closing delimiter (`*/`) for these C comments, and any whitespace on the line is totally transparent.

These control comments also function in their C++ formats, namely `// *INDENT-OFF*` and `// *INDENT-ON*`.

It should be noted that the internal state of `indent` remains unchanged over the course of the unformatted section. Thus, for example, turning off formatting in the middle of a function and continuing it after the end of the function may lead to bizarre results. It is therefore wise to be somewhat modular in selecting code to be left unformatted.

As a historical note, some earlier versions of `indent` produced error messages beginning with `*INDENT**`. These versions of `indent` were written to ignore any input text lines which began with such error messages. I have removed this incestuous feature from GNU `indent`.

1.11 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.

The '`-pmt`' option causes `indent` to preserve the access and modification times on the output files. Using this option has the advantage that running indent on all source and header files in a project won't cause `make` to rebuild all targets. This option is only available on Operating Systems that have the POSIX `utime(2)` function.

1.12 Bugs

When `indent` is run twice on a file, with the same profile, it should *never* change that file the second time. With the current design of `indent`, this can not be guaranteed, however, and it has not been extensively tested.

`indent` does not understand C. In some cases this leads to the inability to join lines. The result is that running a file through `indent` is *irreversible*, even if the used input file was the result of running `indent` with a given profile ('`.indent.pro`').

While an attempt was made to get `indent` working for C++, is will not do a good job on any C++ source except the very simple.

`indent` does not look at the given '`--line-length`' option when writing comments to the output file. This results often in comments being put far to the right. In order to prohibit `indent` from joining a broken line that has a comment at the end, make sure that the comments start on the first line of the break.

`indent` does not count lines and comments (see the '`-v`' option) when `indent` is turned off with `/* *INDENT-OFF* */`.

Comments of the form `/*UPPERCASE*/` are not treated as comment but as an identifier, causing them to be joined with the next line. This renders comments of this type useless, unless they are embedded in the code to begin with.

1.13 Copyright

The following copyright notice applies to the `indent` program. The copyright and copying permissions for this manual appear near the beginning of '`indent.texinfo`' and '`indent.info`', and near the end of '`indent.1`'.

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 3.

‘`-bap`’
‘`--blank-lines-after-procedures`’
: Force blank lines after procedure bodies.
See Section 1.4 [Blank lines], page 3.

‘`-bbb`’
‘`--blank-lines-after-block-comments`’
: Force blank lines after block comments.
See Section 1.4 [Blank lines], page 3.

‘`-bbo`’
‘`--break-before-boolean-operator`’
: Prefer to break long lines before boolean operators.
See Section 1.9 [Breaking long lines], page 10.

‘`-bc`’
‘`--blank-lines-after-commas`’
: Force newline after comma in declaration.
See Section 1.7 [Declarations], page 8.

‘`-bl`’
‘`--braces-after-if-line`’
: Put braces on line after `if`, etc.
See Section 1.6 [Statements], page 6.

‘`-bli`*n*’
‘`--brace-indent`*n*’
: Indent braces *n* spaces.
See Section 1.6 [Statements], page 6.

‘`-bls`’
‘`--braces-after-struct-decl-line`’
: Put braces on the line after `struct` declaration lines.
See Section 1.7 [Declarations], page 8.

‘`-br`’
‘`--braces-on-if-line`’
: Put braces on line with `if`, etc.
See Section 1.6 [Statements], page 6.

‘`-brs`’
‘`--braces-on-struct-decl-line`’
: Put braces on `struct` declaration line.
See Section 1.7 [Declarations], page 8.

‘`-bs`’
‘`--Bill-Shannon`’
‘`--blank-before-sizeof`’
: Put a space between `sizeof` and its argument.
See Section 1.6 [Statements], page 6.

‘`-cn`’
‘`--comment-indentationn`’
: Put comments to the right of code in column *n*.
See Section 1.5 [Comments], page 5.

‘`-cbin`’
‘`--case-brace-indentationn`’
: Indent braces after a case label N spaces.
See Section 1.6 [Statements], page 6.

‘`-cdn`’
‘`--declaration-comment-columnn`’
: Put comments to the right of the declarations in column *n*.
See Section 1.5 [Comments], page 5.

‘`-cdb`’
‘`--comment-delimiters-on-blank-lines`’
: Put comment delimiters on blank lines.
See Section 1.5 [Comments], page 5.

‘`-ce`’
‘`--cuddle-else`’
: Cuddle else and preceeding ‘`}`’.
See Section 1.5 [Comments], page 5.

‘`-cin`’
‘`--continuation-indentationn`’
: Continuation indent of *n* spaces.
See Section 1.6 [Statements], page 6.

‘`-clin`’
‘`--case-indentationn`’
: Case label indent of *n* spaces.
See Section 1.6 [Statements], page 6.

‘`-cpn`’
‘`--else-endif-columnn`’
: Put comments to the right of `#else` and `#endif` statements in column *n*.
See Section 1.5 [Comments], page 5.

‘`-cs`’
‘`--space-after-cast`’
: Put a space after a cast operator.
See Section 1.6 [Statements], page 6.

'-d*n*'
'--line-comments-indentation*n*'
: Set indentation of comments not to the right of code to *n* spaces.
See Section 1.5 [Comments], page 5.

'-di*n*'
'--declaration-indentation*n*'
: Put variables in column *n*.
See Section 1.7 [Declarations], page 8.

'-fc1'
'--format-first-column-comments'
: Format comments in the first column.
See Section 1.5 [Comments], page 5.

'-fca'
'--format-all-comments'
: Do not disable all formatting of comments.
See Section 1.5 [Comments], page 5.

'-gnu'
'--gnu-style'
: Use GNU coding style. This is the default.
See Section 1.3 [Common styles], page 3.

'-hnl'
'--honour-newlines'
: Prefer to break long lines at the position of newlines in the input.
See Section 1.9 [Breaking long lines], page 10.

'-i*n*'
'--indent-level*n*'
: Set indentation level to *n* spaces.
See Section 1.8 [Indentation], page 9.

'-ip*n*'
'--parameter-indentation*n*'
: Indent parameter types in old-style function definitions by *n* spaces.
See Section 1.8 [Indentation], page 9.

'-kr'
'--k-and-r-style'
: Use Kernighan & Ritchie coding style.
See Section 1.3 [Common styles], page 3.

'-l*n*'
'--line-length*n*'
: Set maximum line length for non-comment lines to *n*.
See Section 1.9 [Breaking long lines], page 10.

'-lc*n*'
'--comment-line-length*n*'
: Set maximum line length for comment formatting to *n*.
See Section 1.5 [Comments], page 5.

'-lp'
'--continue-at-parentheses'
: Line up continued lines at parentheses.
See Section 1.8 [Indentation], page 9.

'-lps'
'--leave-preprocessor-space'
: Leave space between '#' and preprocessor directive.
See Section 1.8 [Indentation], page 9.

'-nbad'
'--no-blank-lines-after-declarations'
: Do not force blank lines after declarations.
See Section 1.4 [Blank lines], page 3.

'-nbap'
'--no-blank-lines-after-procedures'
: Do not force blank lines after procedure bodies.
See Section 1.4 [Blank lines], page 3.

'-nbbo'
'--break-after-boolean-operator'
: Do not prefer to break long lines before boolean operators.
See Section 1.9 [Breaking long lines], page 10.

'-nbc'
'--no-blank-lines-after-commas'
: Do not force newlines after commas in declarations.
See Section 1.7 [Declarations], page 8.

'-ncdb'
'--no-comment-delimiters-on-blank-lines'
: Do not put comment delimiters on blank lines.
See Section 1.5 [Comments], page 5.

'-nce'
'--dont-cuddle-else'
: Do not cuddle } and else.
See Section 1.6 [Statements], page 6.

'-ncs'
'--no-space-after-casts'
: Do not put a space after cast operators.
See Section 1.6 [Statements], page 6.

'-nfc1'
'--dont-format-first-column-comments'
Do not format comments in the first column as normal.
See Section 1.5 [Comments], page 5.

'-nfca'
'--dont-format-comments'
Do not format any comments.
See Section 1.5 [Comments], page 5.

'-nhnl'
'--ignore-newlines'
Do not prefer to break long lines at the position of newlines in the input.
See Section 1.9 [Breaking long lines], page 10.

'-nip'
'--no-parameter-indentation'
Zero width indentation for parameters.
See Section 1.8 [Indentation], page 9.

'-nlp'
'--dont-line-up-parentheses'
Do not line up parentheses.
See Section 1.6 [Statements], page 6.

'-npcs'
'--no-space-after-function-call-names'
Do not put space after the function in function calls.
See Section 1.6 [Statements], page 6.

'-nprs'
'--no-space-after-parentheses'
Do not put a space after every '(' and before every ')'.
See Section 1.6 [Statements], page 6.

'-npsl'
'--dont-break-procedure-type'
Put the type of a procedure on the same line as its name.
See Section 1.7 [Declarations], page 8.

'-nsc'
'--dont-star-comments'
Do not put the '*' character at the left of comments.
See Section 1.5 [Comments], page 5.

'-nsob'
'--leave-optional-blank-lines'
Do not swallow optional blank lines.
See Section 1.4 [Blank lines], page 3.

'-nss'
'--dont-space-special-semicolon'
Do not force a space before the semicolon after certain statements. Disables '-ss'.
See Section 1.6 [Statements], page 6.

'-nv'
'--no-verbosity'
Disable verbose mode.
See Section 1.11 [Miscellaneous options], page 12.

'-orig'
'--original'
Use the original Berkeley coding style.
See Section 1.3 [Common styles], page 3.

'-npro'
'--ignore-profile'
Do not read '.indent.pro' files.
See Section 1.1 [Invoking indent], page 1.

'-pcs'
'--space-after-procedure-calls'
Insert a space between the name of the procedure being called and the '('.
See Section 1.6 [Statements], page 6.

'-pi*n*'
'--paren-indentation*n*'
Specify the extra indentation per open parentheses '(' when a statement is broken. See Section 1.6 [Statements], page 6.

'-pmt'
'--preserve-mtime'
Preserve access and modification times on output files. See Section 1.11 [Miscellaneous options], page 12.

'-prs'
'--space-after-parentheses'
Put a space after every '(' and before every ')'.
See Section 1.6 [Statements], page 6.

'-psl'
'--procnames-start-lines'
Put the type of a procedure on the line before its name.
See Section 1.7 [Declarations], page 8.

'-sbi*n*'
'--struct-brace-indentation*n*'
Indent braces of a struct, union or enum N spaces.
See Section 1.6 [Statements], page 6.

'-sc'
'--start-left-side-of-comments'
Put the '*' character at the left of comments.
See Section 1.5 [Comments], page 5.

'-sob'
'--swallow-optional-blank-lines'
Swallow optional blank lines.
See Section 1.4 [Blank lines], page 3.

'-ss'
'--space-special-semicolon'
On one-line `for` and `while` statments, force a blank before the semicolon.
See Section 1.6 [Statements], page 6.

'-st'
'--standard-output'
Write to standard output.
See Section 1.1 [Invoking indent], page 1.

'-T' Tell `indent` the name of typenames.
See Section 1.7 [Declarations], page 8.

'-ts*n*'
'--tab-size*n*'
Set tab size to *n* spaces.
See Section 1.8 [Indentation], page 9.

'-v'
'--verbose'
Enable verbose mode.
See Section 1.11 [Miscellaneous options], page 12.

'-version'
Output the version number of `indent`.
See Section 1.11 [Miscellaneous options], page 12.

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 . . . . . . . . . . . . . . . . . . . . . . . . . . . -blin
--braces-after-struct-decl-line . . . . . . . . . . . . . . . . . . -bls
--break-after-boolean-operator . . . . . . . . . . . . . . . . . . -nbbo
```

--break-before-boolean-operator	-bbo
--braces-on-if-line	-br
--braces-on-struct-decl-line	-brs
--case-indentation	-cli*n*
--case-brace-indentation	-cbi*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
--honour-newlines	-hnl
--ignore-newlines	-nhnl
--ignore-profile	-npro
--indent-level	-i*n*
--k-and-r-style	-kr
--leave-optional-blank-lines	-nsob
--line-comments-indentation	-d*n*
--leave-preprocessor-space	-lps
--line-length	-l*n*
--no-blank-lines-after-commas	-nbc
--no-blank-lines-after-declarations	-nbad
--no-blank-lines-after-procedures	-nbap
--no-comment-delimiters-on-blank-lines	-ncdb
--no-space-after-casts	-ncs
--no-parameter-indentation	-nip
--no-space-after-function-call-names	-npcs
--no-space-after-parentheses	-nprs
--no-verbosity	-nv
--original	-orig
--parameter-indentation	-ip*n*
--paren-indentation	-pi*n*
--preserve-mtime	-pmt
--procnames-start-lines	-psl
--space-after-cast	-cs

`--space-after-parentheses` . . . . `-prs`
`--space-after-procedure-calls` . . . . `-pcs`
`--space-special-semicolon` . . . . `-ss`
`--standard-output` . . . . `-st`
`--start-left-side-of-comments` . . . . `-sc`
`--struct-brace-indentation` . . . . `-sbi`*n*
`--swallow-optional-blank-lines` . . . . `-sob`
`--tab-size` . . . . `-ts`*n*
`--verbose` . . . . `-v`

Index

Table of Contents