Indent

Format C Code

Edition 1.3, for Indent Version 1.9
January, 1994

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 11, for a list of options, including both long and short names.

Here is another example:

```
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`.

[1] '`+`' is being superseded by '`--`' to maintain consistency with the POSIX standard.

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 called `.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.

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 2). 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.

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 neccessary 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 -bl -bli2 -c33 -cd33 -ncdb -nce -cli0
-cp1 -di2 -nfc1 -nfca -i2 -ip5 -lp -pcs -psl -cs
-nsc -nsob -nss -ts8 -d0 -ci0 -l78
```

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 -nbc -br -c33 -cd33 -ncdb -ce -ci4
-cli0 -cp33 -d0 -di1 -nfc1 -nfca -i4 -ip0 -l75 -lp
-npcs -npsl -nsc -nsob -nss -ts8 -cs
```

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 specifyfying '`--original`', using the long option name). This style is equivalent to the following settings:

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

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 '`-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 '`/*`' and 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 amongst 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 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.

Comments which are not boxed 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' or the '-l' option. '-l' specifies the right margin for all code, and '-lc' specifies the margin for only for comments. If '-l' is used alone, comments will be formatted according to the margin specified with that 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. 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.

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 `/* Claustrophobia */` can be transformed into:

```
/*
   Claustrophobia
 */
```

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:

```
/*
 * Claustrophobia
 */
```

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--;
  }
```

These options also affect structure and enumeration declarations. The '-br' option produces structure declarations like the following:

```
struct Sname {
    int i;
    char chp;
} Vname;
```

The default behaviour, also obtained by specifying '-bl', would yield the following format for the same declaration:

```
struct Sname
  {
     int i;
     char chp;
  }
Vname;
```

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.

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.

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 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.

If you are not using the '-di1' option to place variables being declared immediately after their type, you need to 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'.

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));
```

`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`'.

ASCII 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 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`.

1.10 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.

1.11 Bugs

The "-troff" option is strongly deprecated, and is not supported. A good thing for someone to do is to rewrite 'indent' to generate TeX source as a hardcopy output option, amoung other things.

1.12 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 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.

‘`-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.

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

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

‘`-cd`*n*’
‘`--declaration-comment-column`*n*’
: 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.

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

‘-dn’
‘--line-comments-indentationn’
Set indentation of comments not to the right of code to *n* spaces.
See Section 1.5 [Comments], page 5.

‘-din’
‘--declaration-indentationn’
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 2.

‘`-in`’
‘`--indent-leveln`’
: Set indentation level to *n* spaces.
See Section 1.8 [Indentation], page 8.

‘`-ipn`’
‘`--parameter-indentationn`’
: Indent parameter types in old-style function definitions by *n* spaces.
See Section 1.8 [Indentation], page 8.

‘`-kr`’
‘`--k-and-r-style`’
: Use Kernighan & Ritchie coding style.
See Section 1.3 [Common styles], page 2.

‘`-ln`’
‘`--line-lengthn`’
: Set maximum line length to *n*.
See Section 1.5 [Comments], page 5.

‘`-lcn`’
‘`--comment-line-lengthn`’
: 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 8.

‘`-lps`’
‘`--leave-preprocessor-space`’
: Leave space between ‘`#`’ and preprocessor directive. See Section 1.8 [Indentation], page 8.

‘`-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.

‘-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.

‘-nip’
‘--no-parameter-indentation’
Zero width indentation for parameters.
See Section 1.8 [Indentation], page 8,

‘-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.

‘`-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 1.10 [Miscellaneous options], page 10.

‘`-orig`’
‘`--original`’
: Use the original Berkeley coding style.
See Section 1.3 [Common styles], page 2.

‘`-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.

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

‘`-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.

‘-tsn’
‘--tab-sizen’
Set tab size to *n* spaces.
See Section 1.8 [Indentation], page 8.

‘-v’
‘--verbose’
Enable verbose mode.
See Section 1.10 [Miscellaneous options], page 10.

‘-version’
Output the version number of `indent`.
See Section 1.10 [Miscellaneous options], page 10.

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-on-if-line . . . . . -br
--case-indentation . . . . . -clin
--comment-delimiters-on-blank-lines . . . . . -cdb
--comment-indentation . . . . . -cn
--continuation-indentation . . . . . -cin
--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
--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-verbosity . . . . . -nv
--original . . . . . -orig
--parameter-indentation . . . . . -ip*n*
--procnames-start-lines . . . . . -psl
--space-after-cast . . . . . -cs
--space-after-procedure-calls . . . . . -pcs
--space-special-semicolon . . . . . -ss
--standard-output . . . . . -st
--start-left-side-of-comments . . . . . -sc
--swallow-optional-blank-lines . . . . . -sob
--tab-size . . . . . -ts*n*
--verbose . . . . . -v

Index

—

--blank-after-sizeof ... 8
--blank-lines-after-commas ... 8
--blank-lines-after-declarations ... 3
--blank-lines-after-procedures ... 4
--brace-indent*n* ... 7
--braces-after-if-line ... 6
--braces-on-if-line ... 6
--case-indentation*n* ... 7
--comment-delimiters-on-blank-lines ... 6
--comment-indentation*n* ... 6
--continuation-indentation*n* ... 8
--continue-at-parentheses ... 8
--cuddle-else ... 7
--declaration-comment-column*n* ... 6
--declaration-indentation*n* ... 8
--dont-break-procedure-type ... 8
--dont-cuddle-else ... 7
--dont-format-comments ... 5
--dont-format-first-column-comments ... 5
--dont-line-up-parentheses ... 8
--dont-space-special-semicolon ... 7
--dont-star-comments ... 6
--else-endif-column*n* ... 6
--format-all-comments ... 5
--format-first-column-comments ... 5
--gnu-style ... 2
--ignore-profile ... 2
--indent-level*n* ... 8
--k-and-r-style ... 3
--leave-optional-blank-lines ... 3
--leave-preprocessor-space ... 9
--line-comments-indentation*n* ... 6
--line-length*n* ... 5
--no-blank-lines-after-commas ... 8
--no-blank-lines-after-declarations ... 3
--no-blank-lines-after-procedures ... 4
--no-comment-delimiters-on-blank-lines ... 6
--no-parameter-indentation ... 9
--no-space-after-casts ... 8
--no-space-after-function-call-names ... 7
--no-verbosity ... 10
--original ... 3
--output-file ... 1
--parameter-indentation*n* ... 9
--procnames-start-lines ... 8
--remove-preprocessor-space ... 9
--space-after-cast ... 8
--space-after-procedure-calls ... 7
--space-special-semicolon ... 7
--standard-output ... 1
--star-left-side-of-comments ... 6
--swallow-optional-blank-lines ... 3
--tab-size*n* ... 9
--verbose ... 10
-bad ... 3
-bap ... 4
-bc ... 8
-bl ... 6
-bli*n* ... 7
-br ... 6
-bs ... 8
-cdb ... 6
-cd*n* ... 6
-ce ... 7
-ci*n* ... 8
-cli*n* ... 7
-c*n* ... 6
-cp*n* ... 6
-cs ... 8
-dce ... 7
-di*n* ... 8
-d*n* ... 6
-fc1 ... 5
-fca ... 5
-gnu ... 2
-i*n* ... 8
-ip*n* ... 9
-kr ... 3
-l*n* ... 5
-lp ... 8
-lps ... 9
-nbad ... 3
-nbap ... 4
-nbc ... 8
-ncdb ... 6
-ncs ... 8
-nfc1 ... 5
-nfca ... 5
-nip ... 9
-nlp ... 8
-nlps ... 9
-npcs ... 7
-npro ... 2
-npsl ... 8
-nsc ... 6
-nsob ... 3
-nss ... 7
-nv ... 10
-o ... 1
-orig ... 3
-pcs ... 7
-psl ... 8
-sc ... 6
-sob ... 3
-ss ... 7
-st ... 1
-ts*n* ... 9
-T ... 8

-v 10
-version 10

.
.indent.pro file 2

B
Beginning indent 1
Berkeley style 3
Blank lines 3

C
Comments 5

E
etags requires '-psl' 8

G
GNU style 2

I
Initialization file 2
Invoking indent 1

K
Kernighan & Ritchie style 3

L
Long options, use of 1

O
Original Berkeley style 3
Output File Specification 1

S
Standard Output 1
Starting indent 1

T
Typenames 8

U
Using Standard Input 1

Table of Contents

1 The `indent` Program 1

1.1 Invoking `indent` 1
1.2 Backup Files 2
1.3 Common styles 2
1.4 Blank lines 3
1.5 Comments 5
1.6 Statements 6
1.7 Declarations 8
1.8 Indentation 8
1.9 Disabling Formatting 9
1.10 Miscellaneous options 10
1.11 Bugs 10
1.12 Copyright 10

Appendix A Option Summary 11

Index 19