Artistic Style 1.22

A Free, Fast and Small Automatic Formatter for C, C++, C#, and Java Source Code

Contents

General Information

Line Endings

Line endings in the formatted file will be the same as the input file. If there are mixed line endings the most frequent occurrence will be used.

File Type
Artistic Style will determine the file type from the file extension. The extension ".java" indicates a Java file, and ".cs" indicates a C# file. Everything else is a C or C++ file. If you are using a non-standard file extension for Java or C#, use one of the `--mode=` options described in "Formatting Options".

Wildcards and Recursion
Artistic Style can process directories recursively and wildcards are processed internally. If a shell is used it should pass the wildcards to Artistic Style instead of resolving them first. For Linux use double quotes around paths whose filename contains wildcards. For Windows use double quotes around paths whose filename contains spaces. The "Other Options" section contains information on recursive processing.

Quick Start
If you have never used Artistic Style there are a couple of ways to start. One is to run it with no options at all. This will format the file with 4 spaces per indent and will leave the brackets unchanged. Another is to use one of the predefined styles described in the "Predefined Style Options". Select one with a bracket formatting style you like. Once you are familiar with the options you can customize the format to your personal preference.

Usage

Artistic style is a console program that receives information from the command line. The format of the command line is:

```
astyle [options] SourceFile1 SourceFile2 SourceFile3 [ . . . ]
```

Or to save the file with a different name:

```
astyle [options] < OriginalSourceFile > BeautifiedSourceFile
```

The < and > characters are used to redirect the files into standard input and out of standard output - don't forget them!

The block parens [ ] indicate that more than one option or more than one filename can be entered. They are NOT actually included in the command. For the options format see the following Options section.

With the first option, the newly indented file **retains the original file name**, while a copy of the original file is created with an `.orig` appended to the original file name. (This can be set to a different string by the option `--suffix=`, or suppressed altogether by the options `-n` or `--suffix=none`). Thus, after indenting *`SourceFile1.cpp`* as above, the indented result will be named *`SourceFile1.cpp`*, while the original pre-indented file will be renamed to *`SourceFile1.cpp.orig`*.

With the second option, the file is saved with a different name. Therefore a copy is **not** created.

`Wildcards` (such as "*.cpp"), can be used if the project is compiled to include them. See the Installation Information for instructions.

Options File

A default options file may be used to set your favorite source style.

- The command line options have precedence. If there is a conflict between a command line option and an option in the default options file, the command line option will be used.
- Artistic Style looks for this file in the following locations (in order):
  1. the file indicated by the **--options=** command line option;
  2. the **file** and **directory** indicated by the environment variable **ARTISTIC_STYLE_OPTIONS** (if it exists);
  3. the file named **.astylerc** in the directory pointed to by the **HOME** environment variable (e.g. "**$HOME/.astylerc**" on Linux);
  4. the file named **astylerc** in the directory pointed to by the **USERPROFILE** environment variable (e.g. "**%USERPROFILE%\astylerc**" on Windows).
- This option file lookup can be disabled by specifying `--options=none` on the command line.
- Options may be set apart by new-lines, tabs or spaces.
- Long options **in the options file** may be written **without** the preceding '**--**'.
- Lines within the options file that begin with '**#**' are considered **line-comments**.
- Example of a default options file:

```
# brackets should be attached to pre-bracket lines
brackets=attach
# set 6 spaces per indent
indent=spaces=6
# indent switch blocks
indent-switches
# suffix of original files should be .pre
suffix=.pre
```

Options

Not specifying any option will result in 4 spaces per indent, no change in bracket placement, and no formatting changes.

Options may be written in two different ways:

- **Long options:**
  These options start with '`--`', and **must be written one at a time.**
  (Example: '`--brackets=attach --indent=spaces=4`')
- **Short Options:**
  These options start with a single '**-**', and **may be concatenated together**.
  (Example: '`-bps4`' is the same as writing '`-b -p -s4`'.)

Predefined Style Options

Predefined Style options define the style by setting several other options. If other options are also used, the placement of the predefined style option in the command line is important. If the predefined style option is placed first, the other options may override the predefined style. If placed last, the predefined style will override the other options.

For example the style --style=ansi sets the option --brackets=break. If the command line specifies "--style=ansi --brackets=attach", the brackets will be attached and the style will not be ansi style. If the order on the command line is reversed to "--brackets=attach --style=ansi ", the brackets will be broken (ansi style) and the attach option will be ignored.

For the options set by each style check the parseOption function in astyle_main.cpp.

--style=ansi

ANSI style formatting/indenting. Brackets are broken, indentation is 4 spaces. Namespaces, classes, and switches are NOT indented.

```
namespace foospace
{
int Foo()
{
    if (isBar)
    {
        bar();
        return 1;
    }
    else
        return 0;
}
}
```

--style=gnu

GNU style formatting/indenting. Brackets are broken, blocks are indented, and indentation is 2 spaces. Namespaces, classes, and switches are NOT indented.

```
namespace foospace
  {
    int Foo()
      {
        if (isBar)
          {
            bar();
            return 1;
          }
        else
          return 0;
      }
  }
```

`--style=kr`

Kernighan & Ritchie style formatting/indenting. Brackets are attached, indentation is 4 spaces. Namespaces, classes, and switches are NOT indented.

```
namespace foospace {
int Foo() {
    if (isBar) {
        bar();
        return 1;
    } else
        return 0;
}
}
```

`--style=linux`

Linux style formatting/indenting. All brackets are linux style, indentation is 8 spaces. Namespaces, classes, and switches are NOT indented.

```
namespace foospace
{
int Foo()
{
        if (isBar) {
                bar();
                return 1;
        } else
                return 0;
}
}
```

`--style=java`

Java style formatting/indenting. Brackets are attached, indentation is 4 spaces. Switches are NOT indented.

```
class foospace {
    int Foo() {
        if (isBar) {
            bar();
            return 1;
        } else
            return 0;
    }
}
```

Tab and Bracket Options

`default indent option`

If no indentation option is set, the default option of **4 spaces** will be used (e.g. `-s4 --indent=spaces=4`).

--indent=spaces / --indent=spaces=<<span class="option">#

/ -s#

Indent using # spaces per indent (e.g. -s6 --indent=spaces=6). # must be between **1** and **20**. Not specifying # will result in a default of **4 spaces** per indent.

--indent=tab / --indent=tab=# / -t#

Indent using tab characters. Treat each tab as # spaces (e.g. -t6 / --indent=tab=6). # must be between **1** and **20**. If no # is set, treats tabs as **4 spaces**.

--force-indent=tab=# / -T#

Indent using tab characters. Treat each tab as # spaces (e.g. -T6 / --force-indent=tab=6). Uses tabs as indents where --indent=tab prefers to use spaces, such as inside multi-line statements. # must be between **1** and **20**. If no # is set, treats tabs as **4 spaces**.

default brackets option

If no brackets option is set, the brackets **will not be changed.**

--brackets=break / -b

Break brackets from their pre-block statements (e.g. ANSI C / C++ style).

```
void Foo(bool isFoo)
{
    if (isFoo)
    {
        bar();
    }
    else
    {
        anotherBar();
    }
}
```

--brackets=attach / -a

Attach brackets to their pre-block statements (e.g. Java / K&R style).

```
void Foo(bool isFoo) {
    if (isFoo) {
        bar();
    } else {
        anotherBar();
    }
}
```

--brackets=linux / -l

Break brackets from class and function declarations, but attach brackets to pre-block command statements.

```
void Foo(bool isFoo)
{
    if (isFoo) {
        bar();
    } else {
        anotherBar;
    }
}
```

--brackets=break-closing / -y

When used with either `--brackets=attach` or `--brackets=linux`, breaks closing headers (e.g. '`else`', '`catch`', ...) from their immediately preceding closing brackets.

```
void Foo(bool isFoo) {
    if (isFoo) {
        bar();
    } else {
        anotherBar();
    }
}
```

becomes (with a broken 'else'):

```
void Foo(bool isFoo) {
    if (isFoo) {
        bar();
    }
    else {
        anotherBar();
    }
}
```

Indentation Options

--indent-classes / -C

Indent '`class`' and '`struct`' blocks so that the blocks '`public:`', '`protected:`' and '`private:`' are indented. The entire block is indented. This option has no effect on Java and C# files.

```
class Foo
{
public:
    Foo();
    virtual ~Foo();
};
```

becomes:

```
class Foo
{
    public:
        Foo();
        virtual ~Foo();
};
```

--indent-switches / -S

Indent '`switch`' blocks so that the '`case X:`' statements are indented in the `switch` block. The entire case block is indented.

```
switch (foo)
{
case 1:
    a += 1;
    break;

case 2:
{
    a += 2;
    break;
}
}
```

becomes:

```
switch (foo)
{
    case 1:
        a += 1;
        break;

    case 2:
    {
        a += 2;
        break;
    }
}
```

--indent-cases / -K

Indent '`case X:`' blocks from the '`case X:`' headers. Case statements not enclosed in blocks are NOT indented.

```
switch (foo)
{
    case 1:
        a += 1;
        break;

    case 2:
    {
        a += 2;
        break;
    }
}
```

becomes:

```
switch (foo)
{
    case 1:
        a += 1;
        break;
```

```
        case 2:
            {
                a += 2;
                break;
            }
    }
```

--indent-blocks / -G

Add extra indentation to entire blocks.

```
if (isFoo)
{
    bar();
}
else
    anotherBar();
```

becomes:

```
if (isFoo)
    {
        bar();
    }
else
    anotherBar();
```

--indent-brackets / -B

Add extra indentation to brackets. This option has no effect if `--indent-blocks` is used.

```
if (isFoo)
{
    bar();
}
else
    anotherBar();
```

becomes:

```
if (isFoo)
    {
    bar();
    }
else
    anotherBar();
```

--indent-namespaces / -N

Add extra indentation to namespace blocks.

```
namespace foospace
{
class Foo
{
    public:
```

```
        Foo();
        virtual ~Foo();
};
}
```

becomes:

```
namespace foospace
{
    class Foo
    {
        public:
            Foo();
            virtual ~Foo();
    };
}
```

--indent-labels / -L

Add extra indentation to labels so they appear 1 indent less than the current indentation, rather than being flushed to the left (the default).

```
int foo() {
    while (isFoo) {
        ...
        if (isFoo)
            goto error;
    }

error:
        ...
}
```

becomes (with indented 'error'):

```
int foo() {
    while (isFoo) {
        ...
        if (isFoo)
            goto error;
    }

    error:
        ...
}
```

--indent-preprocessor / -w

Indent multi-line preprocessor definitions ending with a backslash. Should be used with `--convert-tabs` for proper results. Does a pretty good job, but can not perform miracles in obfuscated preprocessor definitions.

```
#define Is_Bar(arg,a,b) \
(Is_Foo((arg), (a)) \
|| Is_Foo((arg), (b)))
```

becomes:

```
#define Is_Bar(arg,a,b) \
    (Is_Foo((arg), (a)) \
     || Is_Foo((arg), (b)))
```

--max-instatement-indent=# / -M#

Indent a maximum of # spaces in a continuous statement, relative to the previous line (e.g. `--max-instatement-indent=40`). # must be less than **80**. If no # is set, the default value of **40** will be used.

```
fooArray[] = { red,
               green,
               blue };

fooFunction(barArg1,
            barArg2,
            barArg3);
```

becomes (with larger value):

```
fooArray[] = { red,
                  green,
                  blue };

fooFunction(barArg1,
               barArg2,
               barArg3);
```

--min-conditional-indent=# / -m#

Set the minimal indent that is added when a header is built of multiple-lines. This indent makes helps to easily separate the header from the command statements that follow. The value for # must be less than **40**. The default setting for this option is **twice the current indent** (e.g. `--min-conditional-indent=8`).

```
// default setting makes this non-bracketed code clear
if (a < b
        || c > d)
    foo++;

// but creates an exaggerated indent in this bracketed code
if (a < b
        || c > d)
{
    foo++;
}
```

becomes (when setting `--min-conditional-indent=0`):

```
// setting makes this non-bracketed code less clear
if (a < b
    || c > d)
    foo++;

// but makes this bracketed code clearer
if (a < b
```

```
    || c > d)
{
    foo++;
}
```

Formatting Options

`--break-blocks / -f`

Pad empty lines around header blocks (e.g. '`if`', '`while`'...). Be sure to read the Supplemental Documentation before using this option.

```
isFoo = true;
if (isFoo) {
    bar();
} else {
    anotherBar();
}
isBar = false;
```

becomes:

```
isFoo = true;

if (isFoo) {
    bar();
} else {
    anotherBar();
}

isBar = false;
```

`--break-blocks=all / -F`

Pad empty lines around header blocks (e.g. '`if`', '`while`'...). Treat closing header blocks (e.g. '`else`', '`catch`') as stand-alone blocks. Be sure to read the Supplemental Documentation before using this option.

```
isFoo = true;
if (isFoo) {
    bar();
} else {
    anotherBar();
}
isBar = false;
```

becomes:

```
isFoo = true;

if (isFoo) {
    bar();

} else {
    anotherBar();
}

isBar = false;
```

--break-elseifs / -e

Break '`else if`' header combinations into separate lines. This option has no effect if `one-line=keep-statements` is used. This option cannot be undone.

```
if (isFoo) {
    bar();
}
else if (isFoo1()) {
    bar1();
}
else if (isFoo2()) }
    bar2;
}
```

becomes:

```
if (isFoo) {
    bar();
}
else
    if (isFoo1()) {
        bar1();
    }
    else
        if (isFoo2()) {
            bar2();
        }
```

--pad=oper / -p

Insert space padding around operators. Operators inside block parens [] are **not** padded. Note that there is no option to unpad. Once padded, they stay padded.

```
if (foo==2)
    a=bar((b-c)*a,*d--);
```

becomes:

```
if (foo == 2)
     a = bar((b - c) * a, * d--);
```

--pad=paren / -P

Insert space padding around parenthesis on both the **outside** and the **inside**.

```
if (isFoo(a, b))
    bar(a, b);
```

becomes:

```
if ( isFoo ( a, b ) )
    bar ( a, b );
```

--pad=paren-out / -d

Insert space padding around parenthesis on the **outside** only. This can be used with `unpad=paren` below to remove unwanted spaces.

```
if (isFoo(a, b))
    bar(a, b);
```

becomes:

```
if (isFoo (a, b) )
    bar (a, b);
```

--pad=paren-in / -D

Insert space padding around parenthesis on the **inside** only. This can be used with `unpad=paren` below to remove unwanted spaces.

```
if (isFoo(a, b))
    bar(a, b);
```

becomes:

```
if ( isFoo( a, b ) )
    bar( a, b );
```

--unpad=paren / -U

Remove extra space padding around parenthesis on the inside and outside. Can be used in combination with the paren padding options `pad=paren-out` and `pad=paren-in` above. Only padding that has not been requested by other options will be removed.

For example, if a source has parens padded on both the inside and outside, and you want inside only. You need to use `unpad=paren` to remove the outside padding, and `pad=paren-in` to retain the inside padding. Using only `pad=paren-in` would not remove the outside padding.

```
if ( isFoo( a, b ) )
    bar ( a, b );
```

becomes (with no padding option requested):

```
if (isFoo(a, b))
    bar(a, b);
```

--one-line=keep-statements / -o

Don't break complex statements and multiple statements residing on a single line.

```
if (isFoo)
{
    isFoo = false; cout << isFoo << endl;
}
```

remains as is.

```
if (isFoo) DoBar();
```

remains as is.

--one-line=keep-blocks / -O

Don't break one-line blocks.

```
if (isFoo)
{ isFoo = false; cout << isFoo << endl; }
```

remains as is.

--convert-tabs / -c

Converts tabs into single spaces.

--fill-empty-lines / -E

Fill empty lines with the white space of the previous line.

--mode=c

Indent a C or C++ file. The option is usually set from the file extension for each file. You can override the setting with this entry. It will be used for all files regardless of the file extension. It allows the formatter to identify language specific syntax such as C++ classes, templates, and keywords.

--mode=java

Indent a Java file. The option is usually set from the file extension for each file. You can override the setting with this entry. It will be used for all files regardless of the file extension. It allows the formatter to identify language specific syntax such as Java class's keywords.

--mode=cs

Indent a C sharp file. The option is usually set from the file extension for each file. You can override the setting with this entry. It will be used for

all files regardless of the file extension. It allows the formatter to identify language specific syntax such as C sharp classes and keywords.

Other Options

`--suffix=####`
Append the suffix #### instead of '.orig' to original filename (e.g. `--suffix=.bak`). If this is to be a file extension, the dot '.' must be included. Otherwise the suffix will be appended to the current file extension.

`--suffix=none / -n`
Do not retain a backup of the original file. The original file is purged after it is formatted.

`--options=####`
Specify an options file #### to read and use.

`--options=none`
Disable the default options file. Only the command-line parameters will be used.

`--recursive / -r / -R`
For each directory in the command line, process all subdirectories recursively. When using the recursive option the file name statement should contain a wildcard. Linux users should place the filepath and name in double quotes so the shell will not resolve the wildcards (e.g. "$HOME/src/*.cpp"). Windows users should place the filepath and name in double quotes if the path or name contains spaces.

`--exclude=####`
Specify a file or sub directory #### to be excluded from processing.

Excludes are matched from the end of the filepath. An exclude option of "templates" will exclude ALL directories named "templates". An exclude option of "cpp/templates" will exclude ALL "cpp/templates" directories. You may proceed backwards in the directory tree to exclude only the required directories.

Specific files may be excluded in the same manner. An exclude option of "default.cpp" will exclude ALL files named "default.cpp". An exclude option of "python/default.cpp" will exclude ALL files named "default.cpp" contained in a "python" subdirectory. You may proceed backwards in the directory tree to exclude only the required files.

Wildcards are NOT allowed. There may be more than one exclude statement. The filepath and name may be placed in double quotes (e.g. --exclude="foo bar.cpp").

`--errors-to-stdout / -X`
Print errors to standard-output rather than to standard-error.

This option should be helpful for systems/shells that do not have this option, such as in Windows95.

`--preserve-date / -Z`
Preserve the original file's date and time modified. The date and time modified will not be changed in the formatted file. This option is not effective if redirection is used.

`--verbose / -v`
Verbose mode. Display optional information, such as release number and statistical data.

`--quiet / -q`
Quiet mode. Suppress all output except error messages.

`--version / -V`
Print version number and quit. The short option must be by itself, it cannot be concatenated with other options.

`--help / -h / -?`
Print a help message and quit. The short option must be by itself, it cannot be concatenated with other options.

ENJOY !!!