ClangFormat

配置文档

The Clang Team

目录

1. 使用

1.1 介绍

clang-format `clang/tools/clang-format` , 可用于格式化 `C` / `C++` / `Java` / `JavaScript` / `Objective-C` / `Protobuf` / `C#`

```
$ clang-format -help
OVERVIEW: A tool to format C/C++/Java/JavaScript/JSON/Objective-C/Protobuf/C# code.

If no arguments are specified, it formats the code from standard input
and writes the result to the standard output.
If <file>s are given, it reformats the files. If -i is specified
together with <file>s, the files are edited in-place. Otherwise, the
result is written to the standard output.

USAGE: clang-format [options] [<file> ...]

OPTIONS:

Clang-format options:

  --Werror                   - If set, changes formatting warnings to errors
  --Wno-error=<value>        - If set don't error out on the specified warning type.
    =unknown                 -   If set, unknown format options are only warned about.
                                 This can be used to enable formatting, even if the
                                 configuration contains unknown (newer) options.
                                 Use with caution, as this might lead to dramatically
                                 differing format depending on an option being
                                 supported or not.
  --assume-filename=<string> - Override filename used to determine the language.
                               When reading from stdin, clang-format assumes this
                               filename to determine the language.
  --cursor=<uint>            - The position of the cursor when invoking
                               clang-format from an editor integration
  --dry-run                  - If set, do not actually make the formatting changes
  --dump-config              - Dump configuration options to stdout and exit.
                               Can be used with -style option.
  --fallback-style=<string>  - The name of the predefined style used as a
                               fallback in case clang-format is invoked with
                               -style=file, but can not find the .clang-format
                               file to use.
                               Use -fallback-style=none to skip formatting.
  --ferror-limit=<uint>      - Set the maximum number of clang-format errors to
                               emit before stopping (0 = no limit). Used only
                               with --dry-run or -n
  -i                         - Inplace edit <file>s, if specified.
  --length=<uint>            - Format a range of this length (in bytes).
                               Multiple ranges can be formatted by specifying
                               several -offset and -length pairs.
                               When only a single -offset is specified without
                               -length, clang-format will format up to the end
                               of the file.
                               Can only be used with one input file.
  --lines=<string>           - <start line>:<end line> - format a range of
                               lines (both 1-based).
                               Multiple ranges can be formatted by specifying
                               several -lines arguments.
                               Can't be used with -offset and -length.
                               Can only be used with one input file.
  -n                         - Alias for --dry-run
  --offset=<uint>            - Format a range starting at this byte offset.
                               Multiple ranges can be formatted by specifying
                               several -offset and -length pairs.
                               Can only be used with one input file.
  --output-replacements-xml  - Output replacements as XML.
  --sort-includes            - If set, overrides the include sorting behavior
                               determined by the SortIncludes style flag
  --style=<string>           - Coding style, currently supports:
                                 LLVM, Google, Chromium, Mozilla, WebKit.
                               Use -style=file to load style configuration from
                               .clang-format file located in one of the parent
                               directories of the source file (or current
                               directory for stdin).
                               Use -style="{key: value, ...}" to set specific
                               parameters, e.g.:
                                 -style="{BasedOnStyle: llvm, IndentWidth: 8}"
  --verbose                  - If set, shows the list of processed files

Generic Options:

  --help                     - Display available options (--help-hidden for more)
  --help-list                - Display list of available options (--help-list-hidden for more)
  --version                  - Display the version of this program
```

最后更新

1.2 配置样式

`clang-format` `-style=` `-style=file` 目录的 `.clang-format` `_clang-format`

当使用 `-style=file` `clang-format` `.clang-format` 当前目录开始。

文件 `.clang-format` `YAML`

```yaml
key1: value1
key2: value2
#
...
```

配置文件可以由几个部分组成，每个部分使用不同的语言：参数表示该配置部分针对的编程语言。有关支持的语言列表，请参阅下面语言选项的说明。第一部分可能没有设置语言，它将为所有语言设置默认样式选项。特定语言的配置节将覆盖在默认部分的设置的选项。

当 `clang-format` `-assume-filename=` 可以用来覆盖 `clang-format`

多语言配置文件的示例：

```yaml
---
#               LLVM        4
BasedOnStyle: LLVM
IndentWidth: 4
---
Language: Cpp
#         C++
DerivePointerAlignment: false
PointerAlignment: Left
---
Language: JavaScript
#   JS    100
ColumnLimit: 100
---
Language: Proto
#          .proto
DisableFormat: true
---
Language: CSharp
#   C#    100
ColumnLimit: 100
...
```

获得包含所有预定义样式配置选项的有效 `.clang-format`

```
clang-format -style=llvm -dump-config > .clang-format
```

当在 `-style=`

```
-style='{key1: value1, key2: value2, ...}'
```

最后更新

1.3 禁用格式

Clang-format `// clang-format off` `/* clang-format off */` `// clang-format on` `/* clang-format on */` ( )

```
int formatted_code;
// clang-format off
    void    unformatted_code  ;
// clang-format on
void formatted_code_again;
```

最后更新

2. 配置

2.1 BasedOnStyle

2.1.1 BasedOnStyle (`string`)

用于配置中未特别设置的所有选项的样式。用于配置中未特别设置的所有选项的样式。

此选项仅在 **clang-format** `-style='{...}'` `.clang-format`

可能的值：

- `LLVM` LLVM
- `Google` Google C++
- `Chromium` Chromium
- `Mozilla` Mozilla
- `WebKit` WebKit
- `Microsoft` Microsoft
- `GNU` GNU

`InheritParentConfig` `.clang-format` ( )
会退回到 `fallback` | `--`
`style={BasedOnStyle: InheritParentConfig, ColumnLimit: 20}`

最后更新

2.2 AccessModifierOffset

2.2.1 AccessModifierOffset (`Integer`)

Info

clang-format 3.3

访问修饰符的额外缩进或缩出，例如 `public:`。

最后更新

2.3 AlignAfterOpenBracket

2.3.1 AlignAfterOpenBracket (`BracketAlignmentStyle`)

Info

clang-format 3.8

如果为 `true`，则在左括号后水平对齐参数。

这适用于圆括号、尖括号和方括号。

可能的值:

Align **DontAlign** **AlwaysBreak**

`BAS_Align` :

```
someLongFunction(argument1,
                 argument2);
```

`BAS_DontAlign` : ContinuationIndentWidth，例如:

```
someLongFunction(argument1,
    argument2);
```

`BAS_AlwaysBreak` :

```
someLongFunction(
    argument1, argument2);
```

最后更新

2.4 AlignArrayOfStructures

2.4.1 AlignArrayOfStructures (`ArrayInitializerAlignmentStyle`)

Info

clang-format 13

如果不是 `None`，则在对结构数组使用初始化时，会将字段对齐到列中。

可能的值：

Left **Right** **None**

`AIAS_Left`：

```
struct test demo[] =
{
    {56, 23,    "hello"},
    {-1, 93463, "world"},
    {7,  5,     "!!"   }
};
```

`AIAS_Right`：

```
struct test demo[] =
{
    {56,    23, "hello"},
    {-1, 93463, "world"},
    { 7,     5,    "!!"}
};
```

`AIAS_None`：

最后更新

2.5 AlignConsecutiveAssignments

2.5.1 AlignConsecutiveAssignments (`AlignConsecutiveStyle`)

Info

clang-format 3.8

对齐连续赋值的样式。

`Consecutive`

```
int a            = 1;
int somelongname = 2;
double c         = 3;
```

可能的值：

None   Consecutive   AcrossEmptyLines   AcrossComments   AcrossEmptyLinesAndComments

`ACS_None` :

`ACS_Consecutive` :

```
int a            = 1;
int somelongname = 2;
double c         = 3;

int d = 3;
/* A comment. */
double e = 4;
```

`ACS_AcrossEmptyLines` :   `ACS_Consecutive` |

```
int a            = 1;
int somelongname = 2;
double c         = 3;

int d            = 3;
/* A comment. */
double e = 4;
```

`ACS_AcrossComments` :   `ACS_Consecutive` |

```
int a            = 1;
int somelongname = 2;
double c         = 3;

int d    = 3;
/* A comment. */
double e = 4;
```

`ACS_AcrossEmptyLinesAndComments` :   `ACS_Consecutive` |

```
int a            = 1;
int somelongname = 2;
double c         = 3;

int d            = 3;
/* A comment. */
double e         = 4;
```

最后更新

2.6 AlignConsecutiveBitFields

2.6.1 AlignConsecutiveBitFields (`AlignConsecutiveStyle`)

Info

clang-format 11

对齐连续位字段的样式。

将连续行的位域分隔符对齐。这将导致如下格式：

```
int aaaa : 1;
int b    : 12;
int ccc  : 8;
```

可能的值：

None **Consecutive** **AcrossEmptyLines** **AcrossComments** **AcrossEmptyLinesAndComments**

`ACS_None`：

`ACS_Consecutive`：

```
int aaaa : 1;
int b    : 12;
int ccc  : 8;

int d : 2;
/* A comment. */
int ee : 3;
```

`ACS_AcrossEmptyLines`： `ACS_Consecutive`

```
int aaaa : 1;
int b    : 12;
int ccc  : 8;

int d    : 2;
/* A comment. */
int ee : 3;
```

`ACS_AcrossComments`： `ACS_Consecutive`

```
int aaaa : 1;
int b    : 12;
int ccc  : 8;

int d  : 2;
/* A comment. */
int ee : 3;
```

`ACS_AcrossEmptyLinesAndComments`： `ACS_Consecutive`

```
int aaaa : 1;
int b    : 12;
int ccc  : 8;

int d    : 2;
/* A comment. */
int ee   : 3;
```

最后更新

2.7 AlignConsecutiveDeclarations

2.7.1 AlignConsecutiveDeclarations (`AlignConsecutiveStyle`)

Info

clang-format 3.8

对齐连续声明的样式。

将连续行的声明名称对齐。这将导致如下格式：

```
int         aaaa = 12;
float       b = 23;
std::string ccc;
```

可能的值：

None   Consecutive   AcrossEmptyLines   AcrossComments   AcrossEmptyLinesAndComments

`ACS_None`：

`ACS_Consecutive`：

```
int         aaaa = 12;
float       b = 23;
std::string ccc;

int a = 42;
/* A comment. */
bool c = false;
```

`ACS_AcrossEmptyLines`： `ACS_Consecutive`

```
int         aaaa = 12;
float       b = 23;
std::string ccc;

int         a = 42;
/* A comment. */
bool c = false;
```

`ACS_AcrossComments`： `ACS_Consecutive`

```
int         aaaa = 12;
float       b = 23;
std::string ccc;

int  a = 42;
/* A comment. */
bool c = false;
```

`ACS_AcrossEmptyLinesAndComments`： `ACS_Consecutive`

```
int         aaaa = 12;
float       b = 23;
std::string ccc;

int         a = 42;
/* A comment. */
bool        c = false;
```

最后更新

2.8 AlignConsecutiveMacros

2.8.1 AlignConsecutiveMacros (`AlignConsecutiveStyle`)

Info

clang-format 9

对齐连续宏定义的样式。

`Consecutive`

```
#define SHORT_NAME       42
#define LONGER_NAME      0x007f
#define EVEN_LONGER_NAME (2)
#define foo(x)           (x * x)
#define bar(y, z)        (y + z)
```

可能的值：

None　Consecutive　AcrossEmptyLines　AcrossComments　AcrossEmptyLinesAndComments

`ACS_None`：

`ACS_Consecutive`：

```
#define SHORT_NAME       42
#define LONGER_NAME      0x007f
#define EVEN_LONGER_NAME (2)

#define foo(x) (x * x)
/* some comment */
#define bar(y, z) (y + z)
```

`ACS_AcrossEmptyLines`： `ACS_Consecutive`

```
#define SHORT_NAME       42
#define LONGER_NAME      0x007f
#define EVEN_LONGER_NAME (2)

#define foo(x)           (x * x)
/* some comment */
#define bar(y, z) (y + z)
```

`ACS_AcrossComments`： `ACS_Consecutive`

```
#define SHORT_NAME       42
#define LONGER_NAME      0x007f
#define EVEN_LONGER_NAME (2)

#define foo(x)    (x * x)
/* some comment */
#define bar(y, z) (y + z)
```

`ACS_AcrossEmptyLinesAndComments`： `ACS_Consecutive`

```
#define SHORT_NAME       42
#define LONGER_NAME      0x007f
#define EVEN_LONGER_NAME (2)

#define foo(x)           (x * x)
/* some comment */
#define bar(y, z)        (y + z)
```

最后更新

2.9 AlignEscapedNewlines

2.9.1 AlignEscapedNewlines (`EscapedNewlineAlignmentStyle`)

Info

clang-format 5

在转义的换行符中对齐反斜杠的选项。

可能的值:

DontAlign **Left** **Right**

`ENAS_DontAlign` :

```
#define A \
int aaaa; \
int b; \
int dddddddddd;
```

`ENAS_Left` :

```
#define A \
int aaaa; \
int b;    \
int dddddddddd;
```

`ENAS_Right` :

```
#define A                                                                      \
int aaaa;                                                                      \
int b;                                                                         \
int dddddddddd;
```

最后更新

2.10 AlignOperands

2.10.1 AlignOperands (`OperandAlignmentStyle`)

Info

clang-format 12

如果为 `true`，则水平对齐二进制和三元表达式的操作数。

可能的值:

DontAlign **Align** **AlignAfterOperator**

`OAS_DontAlign`: ContinuationIndentWidth

`OAS_Align`:

具体地说，它将一个需要在多行中分割的单个表达式的操作数对齐，例如:

```
int aaa = bbbbbbbbbbbbbbb +
          ccccccccccccccc;
```

当 BreakBeforeBinaryOperators

```
int aaa = bbbbbbbbbbbbbbb
          + ccccccccccccccc;
```

`OAS_AlignAfterOperator`:

这类似于 `AOS_Align` `BreakBeforeBinaryOperators`

```
int aaa = bbbbbbbbbbbbbbb
        + ccccccccccccccc;
```

最后更新

2.11 AlignTrailingComments

2.11.1 AlignTrailingComments (`Boolean`)

Info

clang-format 3.7

如果为 `true`，则对齐尾随注释。

true **false**

```
int a;     // My comment a
int b = 2; // comment  b

int a; // My comment a
int b = 2; // comment about b
```

最后更新

2.12 AllowAllArgumentsOnNextLine

2.12.1 AllowAllArgumentsOnNextLine (`Boolean`)

Info

clang-format 9

如果函数调用或花括号的初始化列表不能放在一行中，则允许将所有参数放到下一行，即使 BinPackArguments `false`。

true **false**

```
callFunction(
    a, b, c, d);

callFunction(a,
             b,
             c,
             d);
```

最后更新

2.13 AllowAllConstructorInitializersOnNextLine

2.13.1 AllowAllConstructorInitializersOnNextLine (`Boolean`)

Info

clang-format 9

Warning

此选项已弃用。

请参阅 PackConstructorInitializers `NextLine`。

最后更新

2.14 AllowAllParametersOfDeclarationOnNextLine

2.14.1 AllowAllParametersOfDeclarationOnNextLine (`Boolean`)

Info

clang-format 3.3

如果函数声明不适合一行，则允许将函数声明的所有参数放到下一行，即使 BinPackParameters `false`。

true **false**

```
void myFunction(
    int a, int b, int c, int d, int e);

void myFunction(int a,
                int b,
                int c,
                int d,
                int e);
```

最后更新

2.15 AllowShortBlocksOnASingleLine

2.15.1 AllowShortBlocksOnASingleLine (`ShortBlockStyle`)

Info

clang-format 11

依赖于值，`while (true) {continue;`

可能的值：

Never **Empty** **Always**

`SBS_Never`：

```
while (true) {
}
while (true) {
continue;
}
```

`SBS_Empty`：

```
while (true) {}
while (true) {
continue;
}
```

`SBS_Always`：

```
while (true) {}
while (true) { continue; }
```

最后更新

2.16 AllowShortCaseLabelsOnASingleLine

2.16.1 AllowShortCaseLabelsOnASingleLine (`Boolean`)

Info

clang-format 3.6

如果为 `true`，短 `case`

true **false**

```
switch (a) {
case 1: x = 1; break;
case 2: return;
}

switch (a) {
case 1:
  x = 1;
  break;
case 2:
  return;
}
```

最后更新

2.17 AllowShortEnumsOnASingleLine

2.17.1 AllowShortEnumsOnASingleLine (`Boolean`)

Info

clang-format 12

允许在单行上使用短的枚举。

true **false**

```
enum { A, B } myEnum;

enum
{
    A,
    B
} myEnum;
```

最后更新

2.18 AllowShortFunctionsOnASingleLine

2.18.1 AllowShortFunctionsOnASingleLine (`ShortFunctionStyle`)

Info

clang-format 3.5

根据值， `int f() { return 0; }`

可能的值：

None InlineOnly Empty Inline All

`SFS_None`：

`SFS_InlineOnly`： `Inline` `Empty`：

```
class Foo {
    void f() { foo(); }
};
void f() {
    foo();
}
void f() {
}
```

`SFS_Empty`：

```
void f() {}
void f2() {
    bar2();
}
```

`SFS_Inline`： `Empty`。

```
class Foo {
    void f() { foo(); }
};
void f() {
    foo();
}
void f() {}
```

`SFS_All`：

```
class Foo {
    void f() { foo(); }
};
void f() { bar(); }
```

最后更新

2.19 AllowShortIfStatementsOnASingleLine

2.19.1 AllowShortIfStatementsOnASingleLine (`ShortIfStyle`)

Info

clang-format 9

如果为 `true`，则 `if (a) return;`

可能的值:

Never **WithoutElse** **OnlyFirstIf** **AllIfsAndElse**

`SIS_Never`： `if`

```
if (a)
    return;

if (b)
    return;
else
    return;

if (c)
    return;
else {
    return;
}
```

`SIS_WithoutElse`： `else`，只有当 `else` `if`

```
if (a) return;

if (b)
    return;
else
    return;

if (c)
    return;
else {
    return;
}
```

`SIS_OnlyFirstIf`： `if`，但不要放置 `else if` `else`

```
if (a) return;

if (b) return;
else if (b)
    return;
else
    return;

if (c) return;
else {
    return;
}
```

`SIS_AllIfsAndElse`： `else` `if`

```
if (a) return;

if (b) return;
else return;

if (c) return;
else {
    return;
}
```

最后更新

2.20 AllowShortLambdasOnASingleLine

2.20.1 AllowShortLambdasOnASingleLine (`ShortLambdaStyle`)

Info

clang-format 9

依赖于值, `auto lambda []() { return 0; }`

可能的值:

None **Empty** **Inline** **All**

`SLS_None` : `lambda`

`SLS_Empty` : `lambda`。

```
auto lambda = [](int a) {}
auto lambda2 = [](int a) {
    return a;
};
```

`SLS_Inline` : `lambda`

```
auto lambda = [](int a) {
    return a;
};
sort(a.begin(), a.end(), ()[] { return x < y; })
```

`SLS_All` : `lambda`

```
auto lambda = [](int a) {}
auto lambda2 = [](int a) { return a; };
```

最后更新

2.21 AllowShortLoopsOnASingleLine

2.21.1 AllowShortLoopsOnASingleLine (`Boolean`)

Info

clang-format 3.7

如果为 `true`   `while (true) continue;`

最后更新

2.22 AlwaysBreakAfterDefinitionReturnType

2.22.1 AlwaysBreakAfterDefinitionReturnType (`DefinitionReturnTypeBreakingStyle`)

Info

clang-format 3.7

Warning

该选项已弃用，为了向后兼容而保留。

函数定义返回要使用的类型中断样式。

可能的值：

None **All** **TopLevel**

`DRTBS_None`： PenaltyReturnTypeOnItsOwnLine

`DRTBS_All`：

`DRTBS_TopLevel`：

最后更新

2.23 AlwaysBreakAfterReturnType

2.23.1 AlwaysBreakAfterReturnType (`ReturnTypeBreakingStyle`)

Info

clang-format 3.8

函数声明返回要使用的类型中断样式。

可能的值:

None   All   TopLevel   AllDefinitions   TopLevelDefinitions

`RTBS_None`:   PenaltyReturnTypeOnItsOwnLine

```
class A {
    int f() { return 0; };
};
int f();
int f() { return 1; }
```

`RTBS_All`:

```
class A {
int
f() {
    return 0;
};
};
int
f();
int
f() {
    return 1;
}
```

`RTBS_TopLevel`:

```
class A {
    int f() { return 0; };
};
int
f();
int
f() {
    return 1;
}
```

`RTBS_AllDefinitions`:

```
class A {
int
f() {
    return 0;
};
};
int f();
int
f() {
    return 1;
}
```

`RTBS_TopLevelDefinitions`:

```
class A {
    int f() { return 0; };
};
int f();
int
f() {
    return 1;
}
```

最后更新

2.24 AlwaysBreakBeforeMultilineStrings

2.24.1 AlwaysBreakBeforeMultilineStrings (`Boolean`)

Info

clang-format 3.4

如果为 `true`，总是在多行字符串字面值之前中断。

这个标志是为了使文件中有多个多行字符串的情况看起来更一致。因此，只有在此时包装字符串导致从行开始的 ContinuationIndentWidth 进时，它才会生效。

true false

```
aaaa =
    "bbbb"
    "cccc";

aaaa = "bbbb"
       "cccc";
```

最后更新

2.25 AlwaysBreakTemplateDeclarations

2.25.1 AlwaysBreakTemplateDeclarations (`BreakTemplateDeclarationsStyle`)

Info

clang-format 7

要使用的模板声明中断样式。

可能的值：

No **MultiLine** **Yes**

`BTDS_No` : PenaltyBreakTemplateDeclaration

```
template <typename T> T foo() {
}
template <typename T> T foo(int aaaaaaaaaaaaaaaaaaaaa,
                            int bbbbbbbbbbbbbbbbbbbbb) {
}
```

`BTDS_MultiLine` :

```
template <typename T> T foo() {
}
template <typename T>
T foo(int aaaaaaaaaaaaaaaaaaaaa,
    int bbbbbbbbbbbbbbbbbbbbb) {
}
```

`BTDS_Yes` :

```
template <typename T>
T foo() {
}
template <typename T>
T foo(int aaaaaaaaaaaaaaaaaaaaa,
    int bbbbbbbbbbbbbbbbbbbbb) {
}
```

最后更新

2.26 AttributeMacros

2.26.1 AttributeMacros (`List of Strings`)

Info

clang-format 12

字符串的向量，应该被解释为属性　限定符而不是标识符。这对于语言扩展或静态分析器注释很有用。

例如：

```
x = (char *__capability)&y;
int function(void) __ununsed;
void only_writes_to_buffer(char *__output buffer);
```

在 `.clang-format`

```
AttributeMacros: ['__capability', '__output', '__ununsed']
```

最后更新

2.27 BinPackArguments

2.27.1 BinPackArguments (`Boolean`)

Info

clang-format 3.7

如果为 `false`，函数调用的参数要么都在同一行，要么各一行。

true **false**

```
void f() {
f(aaaaaaaaaaaaaaaaaaaa, aaaaaaaaaaaaaaaaaaaa,
    aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa);
}

void f() {
f(aaaaaaaaaaaaaaaaaaaa,
    aaaaaaaaaaaaaaaaaaaa,
    aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa);
}
```

最后更新

2.28 BinPackParameters

2.28.1 BinPackParameters (`Boolean`)

Info

clang-format 3.7

如果为 `false`，函数声明或函数定义的形参要么都在同一行，要么各有一行。

true **false**

```
void f(int aaaaaaaaaaaaaaaaaaaa, int aaaaaaaaaaaaaaaaaaaa,
    int aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa) {}

void f(int aaaaaaaaaaaaaaaaaaaa,
    int aaaaaaaaaaaaaaaaaaaa,
    int aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa) {}
```

最后更新

2.29 BitFieldColonSpacing

2.29.1 BitFieldColonSpacing (`BitFieldColonSpacingStyle`)

Info

clang-format 12

用于位字段的 `BitFieldColonSpacingStyle`。

可能的值：

Both **None** **Before** **After**

`BFCS_Both`：

```
unsigned bf : 2;
```

`BFCS_None`： `:` (AlignConsecutiveBitFields)

```
unsigned bf:2;
```

`BFCS_Before`： `:`

```
unsigned bf :2;
```

`BFCS_After`： `:` (AlignConsecutiveBitFields)

```
unsigned bf: 2;
```

最后更新

2.30 BraceWrapping

2.30.1 BraceWrapping (`BraceWrappingFlags`)

Info

clang-format 3.8

控制大括号换行情况。

如果 BreakBeforeBraces `BS_Custom`，使用这个来指定应该如何处理每个单独的大括号。否则，这将被忽略。

```
# Example of usage:
BreakBeforeBraces: Custom
BraceWrapping:
  AfterEnum: true
  AfterStruct: false
  SplitEmptyFunction: false
```

嵌套的配置标记：

- `bool AfterCaseLabel` : `case`

  true **false**

```
switch (foo) {
    case 1:
    {
        bar();
        break;
    }
    default:
    {
        plop();
    }
}

switch (foo) {
    case 1: {
        bar();
        break;
    }
    default: {
        plop();
    }
}
```

- `bool AfterClass` : `class`

  true **false**

```
class foo {};

class foo
{};
```

- `BraceWrappingAfterControlStatementStyle AfterControlStatement` : (`if` / `for` / `while` / `switch` /...)

可能的值:

Never **MultiLine** **Always**

`BWACS_Never`:

```
if (foo()) {
} else {
}
for (int i = 0; i < 10; ++i) {
}
```

`BWACS_MultiLine`:

```
if (foo && bar &&
    baz)
{
    quux();
}
while (foo || bar) {
}
```

`BWACS_Always`:

```
if (foo())
{
} else
{}
for (int i = 0; i < 10; ++i)
{}
```

- `bool AfterEnum` : `enum`

true **false**

```
enum X : int
{
    B
};

enum X : int { B };
```

- `bool AfterFunction` :

true **false**

```
void foo()
{
    bar();
    bar2();
}

void foo() {
    bar();
    bar2();
}
```

`bool AfterNamespace` : `namespace`

true **false**

```
namespace
{
    int foo();
    int bar();
}

namespace {
    int foo();
    int bar();
}
```

- `bool AfterObjCDeclaration` : `ObjC` (interfaces, implementations…). `@autoreleasepool` `@synchronized` AfterControlStatement
- `bool AfterStruct` : `struct`

true **false**

```
struct foo
{
    int x;
};

struct foo {
    int x;
};
```

- `bool AfterUnion` : `union`

true **false**

```
union foo
{
    int x;
}

union foo {
    int x;
}
```

- `bool AfterExternBlock` : `extern`

true **false**

```
extern "C"
{
int foo();
}

extern "C" {
int foo();
}
```

- `bool BeforeCatch` : `catch`

true **false**

```
try {
    foo();
}
catch () {
}

try {
    foo();
} catch () {
}
```

- `bool BeforeElse` : `else`

true **false**

```
if (foo()) {
}
else {
}

if (foo()) {
} else {
}
```

- `bool BeforeLambdaBody` : `lambda`

true **false**

```
connect(
    []()
    {
        foo();
        bar();
    });

connect([]() {
    foo();
    bar();
});
```

- `bool BeforeWhile` : `while`

true **false**

```
do {
    foo();
}
while (1);

do {
    foo();
} while (1);
```

- `bool IndentBraces` :
- `bool SplitEmptyFunction` : `false`，则空函数体可以放在一行上。该选项仅在函数的左括号已经被包装的情况下使用，即设置了 AfterFunction / (AllowShortFunctionsOnASingleLine)

true **false**

```
int f()
{
}

int f()
{}
```

- `bool SplitEmptyRecord` : `false`，空记录 例如类，结构体或联合 主体可以放在一行上。此选项仅在记录的开括号已经被包装时使用，即设置了 AfterClass ( )

true **flase**

```
class Foo
{
}

class Foo
{}
```

- `bool SplitEmptyNamespace` : `false`，则空的命名空间主体可以放在一行上。该选项仅在名称空间的左大括号已经被包装时使用，即设置了 `AfterNamespace`

true **false**

```
namespace Foo
{
}

namespace Foo
{}
```

最后更新

2.31 BreakAfterJavaFieldAnnotations

2.31.1 BreakAfterJavaFieldAnnotations (`Boolean`)

Info

clang-format 3.8

在 `Java`

true **false**

```
@Partial
@Mock
DataLoad loader;

@Partial @Mock DataLoad loader;
```

最后更新

2.32 BreakBeforeBinaryOperators

2.32.1 BreakBeforeBinaryOperators (`BinaryOperatorStyle`)

Info

clang-format 3.6

二元运算符换行的方式。

可能的值:

None **NonAssignment** **All**

`BOS_None` :

```
LooooooooooongType loooooooooooooooooooooooongVariable =
    someLooooooooooooooooongFunction();

bool value = aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa +
                    aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa ==
                aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa &&
            aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa >
                ccccccccccccccccccccccccccccccccccccccccc;
```

`BOS_NonAssignment` :

```
LooooooooooongType loooooooooooooooooooooooongVariable =
    someLooooooooooooooooongFunction();

bool value = aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
                    + aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
                == aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
            && aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
                    > ccccccccccccccccccccccccccccccccccccccccc;
```

`BOS_All` :

```
LooooooooooongType loooooooooooooooooooooooongVariable
    = someLooooooooooooooooongFunction();

bool value = aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
                    + aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
                == aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
            && aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
                    > ccccccccccccccccccccccccccccccccccccccccc;
```

最后更新

2.33 BreakBeforeBraces

2.33.1 BreakBeforeBraces (`BraceBreakingStyle`)

Info

clang-format 3.7

要使用的大括号中断样式。

可能的值：

Attach **Linux** **Mozilla** **Stroustrup** **Allman** **Whitesmiths** **GNU** **WebKit** **Custom**

`BS_Attach`:

```
namespace N {
enum E {
  E1,
  E2,
};

class C {
public:
  C();
};

bool baz(int i) {
  try {
    do {
      switch (i) {
      case 1: {
        foobar();
        break;
      }
      default: {
        break;
      }
      }
    } while (--i);
    return true;
  } catch (...) {
    handleError();
    return false;
  }
}

void foo(bool b) {
  if (b) {
    baz(2);
  } else {
    baz(5);
  }
}

void bar() { foo(true); }
} // namespace N
```

`BS_Linux`: `Attach`

```
namespace N
{
enum E {
  E1,
  E2,
};

class C
{
public:
  C();
};

bool baz(int i)
{
  try {
    do {
      switch (i) {
      case 1: {
        foobar();
        break;
      }
      default: {
        break;
      }
      }
    } while (--i);
    return true;
  } catch (...) {
    handleError();
    return false;
  }
}

void foo(bool b)
{
  if (b) {
    baz(2);
  } else {
    baz(5);
  }
}

void bar() { foo(true); }
} // namespace N
```

`BS_Mozilla`: `Attach`

```
namespace N {
enum E
{
```

最后更新

2.34 BreakBeforeConceptDeclarations

2.34.1 BreakBeforeConceptDeclarations (`Boolean`)

Info

clang-format 13

如果为 `true`， `concept`

true **false**

```
template<typename T>
concept ...

template<typename T> concept ...
```

最后更新

2.35 BreakBeforeTernaryOperators

2.35.1 BreakBeforeTernaryOperators (`Boolean`)

Info

clang-format 3.7

如果为 `true`，将在换行符之后放置三元操作符。

true **false**

```
veryVeryVeryVeryVeryVeryVeryVeryVeryVeryVeryLongDescription
    ? firstValue
    : SecondValueVeryVeryVeryVeryLong;

veryVeryVeryVeryVeryVeryVeryVeryVeryVeryVeryLongDescription ?
    firstValue :
    SecondValueVeryVeryVeryVeryLong;
```

最后更新

2.36 BreakConstructorInitializers

2.36.1 BreakConstructorInitializers (`BreakConstructorInitializersStyle`)

Info

clang-format 5

要使用的构造函数初始化式样式。

可能的值：

BeforeColon **BeforeComma** **AfterColon**

`BCIS_BeforeColon`：

```
Constructor()
    : initializer1(),
    initializer2()
```

`BCIS_BeforeComma`：

```
Constructor()
    : initializer1()
    , initializer2()
```

`BCIS_AfterColon`：

```
Constructor() :
    initializer1(),
    initializer2()
```

最后更新

2.37 BreakInheritanceList

2.37.1 BreakInheritanceList (`BreakInheritanceListStyle`)

Info

clang-format 7

要使用的继承列表样式。

可能的值:

BeforeColon **BeforeComma** **AfterColon**

`BILS_BeforeColon` :

```
class Foo
    : Base1,
      Base2
{};
```

`BILS_BeforeComma` :

```
class Foo
    : Base1
    , Base2
{};
```

`BILS_AfterColon` :

```
class Foo :
    Base1,
    Base2
{};
```

最后更新

2.38 BreakStringLiterals

2.38.1 BreakStringLiterals (`Boolean`)

Info

clang-format 3.9

允许在格式化时中断字符串文字。

true **false**

```
const char* x = "veryVeryVeryVeryVe"
                "ryVeryVeryVeryVery"
                "VeryLongString";

const char* x =
    "veryVeryVeryVeryVeryVeryVeryVeryVeryVeryVeryVeryLongString";
```

最后更新

2.39 ColumnLimit

2.39.1 ColumnLimit (`Unsigned`)

Info

clang-format 3.7

列的限制。

列限制为 `0` `clang-format`

最后更新

2.40 CommentPragmas

2.40.1 CommentPragmas (`String`)

Info

clang-format 3.7

一个正则表达式，它描述具有特殊含义的注释，不应该将注释分成行或以其他方式更改注释。

```
// CommentPragmas: '^ FOOBAR pragma:'
// Will leave the following line unaffected
#include <vector> // FOOBAR pragma: keep
```

最后更新

2.41 CompactNamespaces

2.41.1 CompactNamespaces (`Boolean`)

Info

clang-format 5

如果为 `true`，连续的名称空间声明将在同一行上。如果为 `false`，则每个名称空间都声明在一个新的行中。

true **false**

```
namespace Foo { namespace Bar {
}}

namespace Foo {
namespace Bar {
}
}
```

如果它不能放在一行中，溢出的名称空间将被包装:

```
namespace Foo { namespace Bar {
namespace Extra {
}}}
```

最后更新

2.42 ConstructorInitializerAllOnOneLineOrOnePerLine

2.42.1 ConstructorInitializerAllOnOneLineOrOnePerLine (`Boolean`)

Info

clang-format 3.7

Warning

此选项已弃用。

请参阅 PackConstructorInitializers `CurrentLine`。

最后更新

2.43 ConstructorInitializerIndentWidth

2.43.1 ConstructorInitializerIndentWidth (`Unsigned`)

Info

clang-format 3.7

用于缩进构造函数初始化列表和继承列表的字符数。

最后更新

2.44 ContinuationIndentWidth

2.44.1 ContinuationIndentWidth (`Unsigned`)

Info

clang-format 3.7

连续行缩进宽度。

```
ContinuationIndentWidth: 2
```

```
int i =         //  VeryVeryVeryVeryVeryLongComment
  longFunction( // Again a long comment
    arg);
```

最后更新

2.45 Cpp11BracedListStyle

2.45.1 Cpp11BracedListStyle (`Boolean`)

Info

clang-format 3.4

如果为 `true`，将带大括号的列表格式化为最适合 `C++11`

重要区别：

- 括号内没有空格。
- 在右括号之前不能换行。
- 使用延续缩进，而不是使用块缩进。

基本上，带大括号的 `C++11` (|)
`clang-format` `{}`

true **false**

```
vector<int> x{1, 2, 3, 4};
vector<T> x{{}, {}, {}, {}};
f(MyMap[{composite, key}]);
new int[3]{1, 2, 3};

vector<int> x{ 1, 2, 3, 4 };
vector<T> x{ {}, {}, {}, {} };
f(MyMap[{ composite, key }]);
new int[3]{ 1, 2, 3 };
```

最后更新

2.46 DeriveLineEnding

2.46.1 DeriveLineEnding (`Boolean`)

Info

clang-format 11

分析格式化文件中最常用的行结束符 `\r\n` 或 `\n`)   UseCRLF

最后更新

2.47 DerivePointerAlignment

2.47.1 DerivePointerAlignment (`Boolean`)

Info

clang-format 3.7

如果为 `true`，分析格式化文件，以确定最常见的 `&` `*`
PointerAlignment

最后更新

2.48 DisableFormat

2.48.1 DisableFormat (`Boolean`)

Info

clang-format 3.7

完全禁用格式。

最后更新

2.49 EmptyLineAfterAccessModifier

2.49.1 EmptyLineAfterAccessModifier (`EmptyLineAfterAccessModifierStyle`)

o

clang-format 14

定义何时在访问修饰符后放置空行。 `EmptyLineBeforeAccessModifier`

可能的值：

Never **Leave** **Always**

`ELAAMS_Never`：

```
struct foo {
private:
  int i;
protected:
  int j;
  /* comment */
public:
  foo() {}
private:
protected:
};
```

`ELAAMS_Leave`： MaxEmptyLinesToKeep。

`ELAAMS_Always`： MaxEmptyLinesToKeep。

```
struct foo {
private:

  int i;
protected:

  int j;
  /* comment */
public:

  foo() {}
private:

protected:

};
```

最后更新

2.50 EmptyLineBeforeAccessModifier

2.50.1 EmptyLineBeforeAccessModifier (`EmptyLineBeforeAccessModifierStyle`)

Info

clang-format 13

定义在何种情况下在访问修饰符之前放置空行。

可能的值:

Never **Leave** **LogicalBlock** **Always**

`ELBAMS_Never` :

```
struct foo {
private:
  int i;
protected:
  int j;
  /* comment */
public:
  foo() {}
private:
protected:
};
```

`ELBAMS_Leave` :

`ELBAMS_LogicalBlock` :

```
struct foo {
private:
  int i;

protected:
  int j;
  /* comment */
public:
  foo() {}

private:
protected:
};
```

`ELBAMS_Always` :

```
struct foo {
private:
  int i;

protected:
  int j;
  /* comment */

public:
  foo() {}

private:

protected:
};
```

最后更新

2.51 ExperimentalAutoDetectBinPacking

2.51.1 ExperimentalAutoDetectBinPacking (`Boolean`)

Info

clang-format 3.7

如果为 `true`，`clang-format`

每个调用都可以被包含，每行或不确定。如果它是不确定的，例如完全在一行上，但需要做出决定，`clang-format`
况，并相应地采取行动。

Note

这是一个实验性的标志，它可能会消失或被重命名。不要在配置文件中使用它，等等。使用时自负风险。

最后更新

2.52 FixNamespaceComments

2.52.1 FixNamespaceComments (`Boolean`)

Info

clang-format 5

如果为 `true` , `clang-format` ShortNamespaceLines

true **false**

```
namespace a {
    foo();
    bar();
} // namespace a

namespace a {
    foo();
    bar();
}
```

最后更新

2.53 ForEachMacros

2.53.1 ForEachMacros (`List of Strings`)

Info

clang-format 3.7

宏的矢量，应该解释为 `foreach`

这些应该是以下形式的宏：

```
FOREACH(<variable-declaration>, ...)
  <loop-body>
```

在 `.clang-format`

```
ForEachMacros: ['RANGES_FOR', 'FOREACH']
```

例如 `BOOST_FOREACH`

最后更新

2.54 IfMacros

2.54.1 IfMacros (`List of Strings`)

Info

clang-format 14

应解释为条件而不是函数调用的宏向量。

这些期望是以下形式的宏：

```
IF(...)
  <conditional-body>
else IF(...)
  <conditional-body>
```

在 `.clang-format`

```
IfMacros: ['IF']
```

例如： `KJ_IF_MAYBE`

最后更新

2.55 IncludeBlocks

2.55.1 IncludeBlocks (`IncludeBlocksStyle`)

Info

clang-format 7

根据这个值，可以将多个 `#include`

可能的值：

Preserve **Merge** **Regroup**

`IBS_Preserve`： `#include`

```
#include "b.h"               into      #include "b.h"

#include <lib/main.h>                  #include "a.h"
#include "a.h"                         #include <lib/main.h>
```

`IBS_Merge`： `#include`

```
#include "b.h"               into      #include "a.h"
                                       #include "b.h"
#include <lib/main.h>                  #include <lib/main.h>
#include "a.h"
```

`IBS_Regroup`： `#include` IncludeCategories。

```
#include "b.h"               into      #include "a.h"
                                       #include "b.h"
#include <lib/main.h>
#include "a.h"                         #include <lib/main.h>
```

最后更新

2.56 IncludeCategories

2.56.1 IncludeCategories (`List of IncludeCategories`)

o

clang-format 7

正则表达式表示用于排序 `#include` `#include`

支持 POSIX

这些正则表达式按顺序匹配包含的文件名 包括 `<>` 或 `""`) `#include`
个类别内按字母顺序排序。

如果所有正则表达式都不匹配，则 `INT_MAX` `0` 。因此它通常保存在 `#include` (https://llvm.org/docs/CodingStandards.html#include-style)

当 `IncludeBlocks = IBS_Regroup` `SortPriority` `#include` `Priority` `#include` `#include` `SortPriority` ，则将其设置为 `Priority`

每个正则表达式都可以用大小写敏感字段标记为区分大小写，但默认情况下它不是。

要在 `.clang-format`

```yaml
IncludeCategories:
  - Regex:           '^"(llvm|llvm-c|clang|clang-c)/'
    Priority:        2
    SortPriority:    2
    CaseSensitive:   true
  - Regex:           '^(<|"(gtest|gmock|isl|json)/)'
    Priority:        3
  - Regex:           '<[[:alnum:].]+>'
    Priority:        4
  - Regex:           '.*'
    Priority:        1
    SortPriority:    0
```

最后更新

2.57 IncludeIsMainRegex

2.57.1 IncludeIsMainRegex (`String`)

Info

clang-format 7

指定在 `file-to-main-include`

当猜测 `#include` `main` `include` (`0`) :

- "" “ ”
- "$" “ ”

例如，如果配置为 `(_test)?$`，则头文件 `a.h` `a.cc` `a_test.cc` `main`。

最后更新

2.58 IncludeIsMainSourceRegex

2.58.1 IncludeIsMainSourceRegex (`String`)

Info

clang-format 7

为在 `file-to-main-include` `main`

默认情况下，只有当文件以 `.c` ，`.cc` ，`.cpp` ，`.c++` ，`.cxx` ，`.m` `.mm` `clang-format` 才会将文件视为 `main`。对于这些文件，会猜测 `main` include(`0` ，参见上面 。这个配置选项允许附加后缀和扩展名的文件被视为`main`。

例如，如果这个选项被配置为 `(Impl\.hpp)$` ，那么一个文件 `ClassImpl.hpp` `main` (`Class.c`、`Class.cc`、`Class.cpp`) `main` `include file` (IncludeIsMainRegex) `ClassImpl.hpp` 何其他包含文件之前。

最后更新

2.59 IndentAccessModifiers

2.59.1 IndentAccessModifiers (`Boolean`)

Info

clang-format 13

指定访问修饰符是否应该有自己的缩进级别。

当为 `false` ( ) AccessModifierOffset。记录成员缩进比记录低一级。当为 `true` 将获得自己的缩进级别。因此，记录成员总是比记录缩进 `2` AccessModifierOffset

true **fasle**

```
class C {
    class D {
        void bar();
        protected:
        D();
    };
    public:
        C();
};
void foo() {
    return 1;
}
```

```
class C {
    class D {
        void bar();
    protected:
        D();
    };
public:
    C();
};
void foo() {
    return 1;
}
```

最后更新

2.60 IndentCaseBlocks

2.60.1 IndentCaseBlocks (`Boolean`)

Info

clang-format 11

缩进 `case` `case`

当为 `false` `case` `case` `case` if `true`

true **false**

```
switch (fool) {
case 1:
  {
    bar();
  }
  break;
default:
  {
    plop();
  }
}

switch (fool) {
case 1: {
  bar();
} break;
default: {
  plop();
}
}
```

最后更新

2.61 IndentCaseLabels

2.61.1 IndentCaseLabels (`Boolean`)

Info

clang-format 3.3

缩进 `case` `switch`

当为 `false` `switch` `switch` `case` (`case`
代码 除非启用了 IndentCaseBlocks)

true **false**

```
switch (fool) {
  case 1:
    bar();
    break;
  default:
    plop();
}

switch (fool) {
case 1:
    bar();
    break;
default:
    plop();
}
```

最后更新

2.62 IndentExternBlock

2.62.1 IndentExternBlock (`IndentExternBlockStyle`)

Info

clang-format 12

`IndentExternBlockStyle` `extern`

可能的值：

AfterExternBlock **NoIndent** **Indent**

`IEBS_AfterExternBlock` : `AfterExternBlock`

```
IndentExternBlock: AfterExternBlock
BraceWrapping.AfterExternBlock: true
```

```
extern "C"
{
    void foo();
}
```

```
IndentExternBlock: AfterExternBlock
BraceWrapping.AfterExternBlock: false
```

```
extern "C" {
void foo();
}
```

`IEBS_NoIndent` :

```
extern "C" {
void foo();
}
```

`IEBS_Indent` :

```
extern "C" {
    void foo();
}
```

最后更新

2.63 IndentGotoLabels

2.63.1 IndentGotoLabels (`Boolean`)

Info

clang-format 10

缩进 `goto`

当为 `false` `goto`

true **false**

```
int f() {
    if (foo()) {
    label1:
        bar();
}
label2:
    return 1;
}

int f() {
    if (foo()) {
label1:
        bar();
}
label2:
    return 1;
}
```

最后更新

2.64 IndentPPDirectives

2.64.1 IndentPPDirectives (`PPDirectiveIndentStyle`)

Info

clang-format 6

要使用的预处理器指令缩进样式。

可能的值：

None **AfterHash** **BeforeHash**

`PPDIS_None`：

```
#if FOO
#if BAR
#include <foo>
#endif
#endif
```

`PPDIS_AfterHash`： `#` 后的指令。

```
#if FOO
#  if BAR
#    include <foo>
#  endif
#endif
```

`PPDIS_BeforeHash`： `#` 前缩进指令。

```
#if FOO
  #if BAR
    #include <foo>
  #endif
#endif
```

最后更新

2.65 IndentRequires

2.65.1 IndentRequires (`Boolean`)

Info

clang-format 13

在模板中缩进 `requires`

true **false**

```
template <typename It>
    requires Iterator<It>
void sort(It begin, It end) {
//....
}

template <typename It>
requires Iterator<It>
void sort(It begin, It end) {
//....
}
```

最后更新

2.66 IndentWidth

IndentWidth (`Unsigned`)

Info

clang-format 3.7

用于缩进的列数。

```
IndentWidth: 3
```

```
void f() {
   someFunction();
   if (true, false) {
      f();
   }
}
```

最后更新

2.67 IndentWrappedFunctionNames

2.67.1 IndentWrappedFunctionNames (`Boolean`)

Info

clang-format 3.7

如果函数定义或声明包装在类型之后，则缩进。

true **false**

```
LooooooooooooooooooooooooooooooooooooooooongReturnType
    LooooooooooooooooooooooooooooooooongFunctionDeclaration();

LooooooooooooooooooooooooooooooooooooooooongReturnType
LooooooooooooooooooooooooooooooooongFunctionDeclaration();
```

最后更新

2.68 InsertTrailingCommas

2.68.1 InsertTrailingCommas (`TrailingCommaStyle`)

Info

clang-format 12

如果设置为 `TCS_Wrapped`，则会在跨多行换行的容器文本 数组和对象 中插入末尾的逗号。它目前仅`JavaScript`
`TCS_None`。`InsertTrailingCommas` BinPackArguments

```
const someArray = [
aaaaaaaaaaaaaaaaaaaaaaaaaaa,
aaaaaaaaaaaaaaaaaaaaaaaaaaa,
aaaaaaaaaaaaaaaaaaaaaaaaaaa,
//                        ^ inserted
]
```

可能的值:

None **Wrapped**

`TCS_None`:

`TCS_Wrapped`:
( )

最后更新

2.69 JavaImportGroups

2.69.1 JavaImportGroups (`List of Strings`)

Info

clang-format 8

一个由 `Java`

一个组的前缀可以是另一个组的子集 总是匹配最长的前缀。在组中，导入按字典顺序排序。静态导入被单独分组，并遵循相同的分组规则。默认情况下，静态导入被放置在非静态导入之前，但是这个行为会被另一个选项 SortJavaStaticImport

在 `.clang-format` `yaml` | `Java` |

```yaml
JavaImportGroups: ['com.example', 'com', 'org']
```

```java
import static com.example.function1;

import static com.test.function2;

import static org.example.function3;

import com.example.ClassA;
import com.example.Test;
import com.example.a.ClassB;

import com.test.ClassC;

import org.example.ClassD;
```

最后更新

2.70 JavaScriptQuotes

2.70.1 JavaScriptQuotes (JavaScriptQuoteStyle)

Info

clang-format 3.9

`JavaScript` `JavaScriptQuoteStyle`。

可能的值：

Leave **Single** **Double**

`JSQS_Leave`：

```
string1 = "foo";
string2 = 'bar';
```

`JSQS_Single`：

```
string1 = 'foo';
string2 = 'bar';
```

`JSQS_Double`：

```
string1 = "foo";
string2 = "bar";
```

最后更新

2.71 JavaScriptWrapImports

2.71.1 JavaScriptWrapImports (`boolean`)

Info

clang-format 3.9

是否包装 `JavaScript` /

true **false**

```
import {
    VeryLongImportsAreAnnoying,
    VeryLongImportsAreAnnoying,
    VeryLongImportsAreAnnoying,
} from 'some/module.js'

import {VeryLongImportsAreAnnoying, VeryLongImportsAreAnnoying, VeryLongImportsAreAnnoying,} from "some/module.js"
```

最后更新

2.72 KeepEmptyLinesAtTheStartOfBlocks

2.72.1 KeepEmptyLinesAtTheStartOfBlocks (`Boolean`)

Info

clang-format 3.7

如果为 `true`，则保留块开头的空行。

true **false**

```
if (foo) {

    bar();
}

if (foo) {
    bar();
}
```

最后更新

2.73 LambdaBodyIndentation

2.73.1 LambdaBodyIndentation (`LambdaBodyIndentationKind`)

Info

clang-format 13

`lambda` `lambda` `OuterScope` `lambda`
`lambda` `OuterScope` KJ
`OuterScope`。KJ

Signature **OuterScope**

`LBI_Signature`: `lambda` `lambda`

```
someMethod(
    [](SomeReallyLongLambdaSignatureArgument foo) {
      return;
    });
```

`LBI_OuterScope`: `lambda` `lambda`

```
someMethod(
    [](SomeReallyLongLambdaSignatureArgument foo) {
  return;
});
```

最后更新

2.74 Language

2.74.1 Language (`LanguageKind`)

Info

clang-format 3.5

语言，这种格式风格是针对的。

可能的值：

None Cpp CSharp Java JavaScript ObjC Proto TableGen TextProto

`LK_None` :

`LK_Cpp` : `C` , `C++` 。

`LK_CSharp` : `C#` 。

`LK_Java` : `Java` 。

`LK_JavaScript` : `JavaScript` 。

`LK_ObjC` : `Objective-C` , `Objective-C++` 。

`LK_Proto` : (https://developers.google.com/protocol-buffers)

`LK_TableGen` : `TableGen`

`LK_TextProto` : (https://developers.google.com/protocol-buffers)

最后更新

2.75 MacroBlockBegin

2.75.1 MacroBlockBegin (`String`)

Info

clang-format 3.7

匹配开始块的宏的正则表达式。

```
MacroBlockBegin: "^NS_MAP_BEGIN|\
NS_TABLE_HEAD$"
MacroBlockEnd: "^\
NS_MAP_END|\
NS_TABLE_.*_END$"
```

With **Without**

```
NS_MAP_BEGIN
  foo();
NS_MAP_END

NS_TABLE_HEAD
  bar();
NS_TABLE_FOO_END

NS_MAP_BEGIN
foo();
NS_MAP_END

NS_TABLE_HEAD
bar();
NS_TABLE_FOO_END
```

最后更新

2.76 MacroBlockEnd

2.76.1 MacroBlockEnd (`String`)

Info

clang-format 3.7

匹配以块结束的宏的正则表达式。

最后更新

2.77 MaxEmptyLinesToKeep

2.77.1 MaxEmptyLinesToKeep (`Unsigned`)

Info

clang-format 3.7

要保留的最大连续空行数。

```
MaxEmptyLinesToKeep: 1         vs.     MaxEmptyLinesToKeep: 0
```

```
int f() {                              int f() {
  int = 1;                                 int i = 1;
                                           i = foo();
  i = foo();                               return i;
                                       }
  return i;
}
```

最后更新

2.78 NamespaceIndentation

2.78.1 NamespaceIndentation (`NamespaceIndentationKind`)

Info

clang-format 3.7

名称空间使用的缩进。

可能的值：

None **Inner** **All**

`NI_None`：

```
namespace out {
int i;
namespace in {
int i;
}
}
```

`NI_Inner`： ( )

```
namespace out {
int i;
namespace in {
    int i;
}
}
```

`NI_All`：

```
namespace out {
    int i;
    namespace in {
        int i;
    }
}
```

最后更新

2.79 NamespaceMacros

2.79.1 NamespaceMacros (`List of Strings`)

Info

clang-format 9

用于打开命名空间块的宏的向量。

这些应该是以下形式的宏：

```
NAMESPACE(<namespace-name>, ...) {
  <namespace-content>
}
```

例如 `TESTSUITE`

最后更新

2.80 ObjCBinPackProtocolList

2.80.1 ObjCBinPackProtocolList (`BinPackStyle`)

Info

clang-format 7

控制包装 `Objective-C` ColumnLimit。

如果是 `Auto` ( ) `BinPackParameters` `Objective-C` 超过 ColumnLimit。

如果是 `Always`，那么当 `Objective-C` ColumnLimit

如果是 `Never`，当超过 ColumnLimit `Objective-C`

- Always (`Auto`, `BinPackParameters=true`)

```
@interface ccccccccccccc () <
    ccccccccccccc, ccccccccccccc,
    ccccccccccccc, ccccccccccccc> {
}
```

- Never (`Auto`, `BinPackParameters=false`)

```
@interface ddddddddddddd () <
    ddddddddddddd,
    ddddddddddddd,
    ddddddddddddd,
    ddddddddddddd> {
}
```

可能的值：

- `BPS_Auto`：
- `BPS_Always`：
- `BPS_Never`：

最后更新

2.81 ObjCBlockIndentWidth

2.81.1 ObjCBlockIndentWidth (`Unsigned`)

Info

clang-format 3.7

用于 `ObjC`

```
ObjCBlockIndentWidth: 4
```

```
[operation setCompletionBlock:^{
    [self onOperationDone];
}];
```

最后更新

2.82 ObjCBreakBeforeNestedBlockParam

2.82.1 ObjCBreakBeforeNestedBlockParam (`Boolean`)

Info

clang-format 12

当函数调用中有嵌套的块形参时，将形参列表分成几行。

true **false**

```
(void)_aMethod
{
    [self.test1 t:self
                w:self
        callback:^(typeof(self) self, NSNumber *u, NSNumber *v) {
            u = c;
        }]
}
```

```
(void)_aMethod
{
    [self.test1 t:self w:self callback:^(typeof(self) self, NSNumber
    *u, NSNumber *v) {
        u = c;
    }]
}
```

最后更新

2.83 ObjCSpaceAfterProperty

2.83.1 ObjCSpaceAfterProperty (`Boolean`)

Info

clang-format 3.7

在 `Objective-C` `@property` `@property (readonly)` `@property(readonly)`。

最后更新

2.84 ObjCSpaceBeforeProtocolList

2.84.1 ObjCSpaceBeforeProtocolList (`Boolean`)

Info

clang-format 3.7

在 `Objective-C` | `Foo <protocol>` `Foo<protocol>`。

最后更新

2.85 PPIndentWidth

2.85.1 PPIndentWidth (`Integer`)

Info

clang-format 14

用于缩进预处理器语句的列数。 当设置为 （默认）时，IndentWidth

```
PPIndentWidth: 1
```

```
#ifdef __linux__
# define FOO
#else
# define BAR
#endif
```

最后更新

2.86 PackConstructorInitializers

2.86.1 PackConstructorInitializers (`PackConstructorInitializersStyle`)

Info

clang-format 14

要使用的包构造函数初始值设定项样式。

可能的值：

Never **BinPack** **CurrentLine** **NextLine**

`PCIS_Never` :

```
Constructor()
    : a(),
      b()
```

`PCIS_BinPack` : Bin-pack

```
Constructor()
    : aaaaaaaaaaaaaaaaaaaa(), bbbbbbbbbbbbbbbbbbbb(),
      cccccccccccccccccccc()
```

`PCIS_CurrentLine` :

```
Constructor() : a(), b()

Constructor()
    : aaaaaaaaaaaaaaaaaaaa(),
      bbbbbbbbbbbbbbbbbbbb(),
      ddddddddddddd()
```

`PCIS_NextLine` : `PCIS_CurrentLine`

```
Constructor() : a(), b()

Constructor()
    : aaaaaaaaaaaaaaaaaaaa(), bbbbbbbbbbbbbbbbbbbb(), ddddddddddddd()

Constructor()
    : aaaaaaaaaaaaaaaaaaaa(),
      bbbbbbbbbbbbbbbbbbbb(),
      cccccccccccccccccccc()
```

最后更新

2.87 PenaltyBreakAssignment

2.87.1 PenaltyBreakAssignment (`Unsigned`)

Info

clang-format 5

突破赋值运算符的补偿。

最后更新

2.88 PenaltyBreakBeforeFirstCallParameter

2.88.1 PenaltyBreakBeforeFirstCallParameter (`Unsigned`)

Info

clang-format 3.7

`call(`

最后更新

2.89 PenaltyBreakComment

2.89.1 PenaltyBreakComment (`Unsigned`)

Info

clang-format 3.7

注释中引入的每个换行符的补偿。

最后更新

2.90 PenaltyBreakFirstLessLess

2.90.1 PenaltyBreakFirstLessLess (`Unsigned`)

Info

clang-format 3.7

在第一个 `<<`

最后更新

2.91 PenaltyBreakString

2.91.1 PenaltyBreakString (`Unsigned`)

Info

clang-format 3.7

在字符串文字中引入的每个换行符的补偿。

最后更新

2.92 PenaltyBreakTemplateDeclaration

2.92.1 PenaltyBreakTemplateDeclaration (`Unsigned`)

Info

clang-format 7

模板声明后中断的补偿。

最后更新

2.93 PenaltyExcessCharacter

2.93.1 PenaltyExcessCharacter (`Unsigned`)

Info

clang-format 3.7

列限制之外的每个字符的补偿。

最后更新

2.94 PenaltyIndentedWhitespace

2.94.1 PenaltyIndentedWhitespace (`Unsigned`)

Info

clang-format 12

空格缩进的每个字符的补偿（相对于前导非空格列计算）。

最后更新

2.95 PenaltyReturnTypeOnItsOwnLine

2.95.1 PenaltyReturnTypeOnItsOwnLine (Unsigned)

Info

clang-format 3.7

将函数的返回类型放在它自己的行上的补偿。

最后更新

2.96 PointerAlignment

2.96.1 PointerAlignment (`PointerAlignmentStyle`)

Info

clang-format 3.7

指针和引用对齐样式。

可能的值：

Left **Right** **Middle**

`PAS_Left` :

```
int* a;
```

`PAS_Right` :

```
int *a;
```

`PAS_Middle` :

```
int * a;
```

最后更新

2.97 QualifierAlignment

2.97.1 QualifierAlignment (`QualifierAlignmentStyle`)

Info

clang-format 14

排列说明符和限定符的不同方式（例如 `const` / `volatile` ）。

Waring

将 `QualifierAlignment` `Leave` `clang-format`
定。因此，应格外小心地审查使用此选项所做的代码更改。

可能的值：

Leave **Left** **Right** **Custom**

`QAS_Leave` : /

```
int const a;
const int *a;
```

`QAS_Left` : /

```
const int a;
const int *a;
```

`QAS_Right` : /

```
int const a;
int const *a;
```

`QAS_Custom` : / QualifierOrder

```
QualifierOrder: ['inline', 'static' , 'type', 'const']
```

```
int const a;
int const *a;
```

最后更新

2.98 QualifierOrder

2.98.1 QualifierOrder (`List of Strings`)

Info

clang-format 14

限定符出现的顺序。顺序是一个数组，可以包含以下任何一项：

- const
- inline
- static
- constexpr
- volatile
- restrict
- type

Attention

它必须包含 `type`。 `type` `type`

```
QualifierOrder: ['inline', 'static', 'type', 'const', 'volatile' ]
```

最后更新

2.99 RawStringFormats

2.99.1 RawStringFormats (`List of RawStringFormats`)

Info

clang-format 6

定义在原始字符串中检测支持的语言代码块的提示。

带有匹配的分隔符或包含匹配的函数名的原始字符串将根据 `.clang-format`
果在 `clang-format` BasedOnStyle BasedOnStyle，则格式化基于
`llvm`

如果指定了规范分隔符，同一语言中出现的其他分隔符将尽可能更新为规范。

每种语言最多应该有一个规范，每个分隔符和包围函数不应该出现在多个规范中。

要在 `.clang-format`

```
RawStringFormats:
  - Language: TextProto
      Delimiters:
        - 'pb'
        - 'proto'
      EnclosingFunctions:
        - 'PARSE_TEXT_PROTO'
      BasedOnStyle: google
  - Language: Cpp
      Delimiters:
        - 'cc'
        - 'cpp'
      BasedOnStyle: llvm
      CanonicalDelimiter: 'cc'
```

最后更新

2.100 ReferenceAlignment

2.100.1 ReferenceAlignment (`ReferenceAlignmentStyle`)

Info

clang-format 14

引用对齐样式（覆盖引用的 PointerAlignment ）。

可能的值：

Pointer Left Right Middle

`RAS_Pointer` : PointerAlignment

`RAS_Left` :

```
int& a;
```

`RAS_Right` :

```
int &a;
```

`RAS_Middle` :

```
int & a;
```

最后更新

2.101 ReflowComments

2.101.1 ReflowComments (`Boolean`)

Info

clang-format 4

如果为 `true`，`clang-format`

false **true**

```
// veryVeryVeryVeryVeryVeryVeryVeryVeryVeryVeryLongComment with plenty of information
/* second veryVeryVeryVeryVeryVeryVeryVeryVeryVeryVeryLongComment with plenty of information */

// veryVeryVeryVeryVeryVeryVeryVeryVeryVeryVeryLongComment with plenty of
// information
/* second veryVeryVeryVeryVeryVeryVeryVeryVeryVeryVeryLongComment with plenty of
* information */
```

最后更新

2.102 ShortNamespaceLines

2.102.1 ShortNamespaceLines (`Unsigned`)

Info

clang-format 14

短名称空间所跨的未换行的最大行数。默认为 `1`。

这通过计算未换行的行 即不包含打开或关闭名称空间大括号 来确定短名称空间的最大长度，并使 FixNamespaceComments 结束注释。

```
ShortNamespaceLines: 1     vs.     ShortNamespaceLines: 0
```

```
namespace a {                      namespace a {
  int foo;                           int foo;
}                                  } // namespace a
```

```
ShortNamespaceLines: 1     vs.     ShortNamespaceLines: 0
```

```
namespace b {                      namespace b {
  int foo;                           int foo;
  int bar;                           int bar;
} // namespace b                   } // namespace b
```

最后更新

2.103 SortIncludes

2.103.1 SortIncludes (SortIncludesOptions)

Info

clang-format 4

控制 `clang-format` `#include`。如果没有，则包含永远不会排序。如果不区分大小写，则包含以 `ascii` 或不区分大小写的方式排序。如果区分大小写，则包含按字母或区分大小写的方式排序。

可能的值：

Never **CaseSensitive** **CaseInsensitive**

`SI_Never`：

```
#include "B/A.h"
#include "A/B.h"
#include "a/b.h"
#include "A/b.h"
#include "B/a.h"
```

`SI_CaseSensitive`：`include`

```
#include "A/B.h"
#include "A/b.h"
#include "B/A.h"
#include "B/a.h"
#include "a/b.h"
```

`SI_CaseInsensitive`：`include`

```
#include "A/B.h"
#include "A/b.h"
#include "a/b.h"
#include "B/A.h"
#include "B/a.h"
```

最后更新

2.104 SortJavaStaticImport

2.104.1 SortJavaStaticImport (`SortJavaStaticImportOptions`)

Info

clang-format 12

在对 `Java` JavaStaticImportAfterImport
后。

可能的值:

Before **After**

`SJSIO_Before`:

```
import static org.example.function1;

import org.example.ClassA;
```

`SJSIO_After`:

```
import org.example.ClassA;

import static org.example.function1;
```

最后更新

2.105 SortUsingDeclarations

2.105.1 SortUsingDeclarations (`Boolean`)

Info

clang-format 5

如果为 `true`，`clang-format`

使用声明的顺序定义如下 使用`::` ;
称。按字典顺序对名称列表进行排序，其中个人名称的排序顺序是所有非名称空间名称出现在所有名称空间名称之前，并且在这些组中，名称不区分大小写。

true **false**

```
using std::cin;
using std::cout;

using std::cout;
using std::cin;
```

最后更新

2.106 SpaceAfterCStyleCast

2.106.1 SpaceAfterCStyleCast (`Boolean`)

Info

clang-format 3.5

如果为 `true`，则在 `C`

true	false

```
(int) i;

(int)i;
```

最后更新

2.107 SpaceAfterLogicalNot

2.107.1 SpaceAfterLogicalNot (`Boolean`)

Info

clang-format 9

如果为 `true`，则在逻辑否操作符 `!`)

true **false**

```
! someExpression();

!someExpression();
```

最后更新

2.108 SpaceAfterTemplateKeyword

2.108.1 SpaceAfterTemplateKeyword (`Boolean`)

Info

clang-format 4

如果为 `true`，则在 `template`

true	false

```
template <int> void foo();

template<int> void foo();
```

最后更新

2.109 SpaceAroundPointerQualifiers

2.109.1 SpaceAroundPointerQualifiers (`SpaceAroundPointerQualifiersStyle`)

Info

clang-format 12

定义在何种情况下在指针限定符之前或之后放置空格

可能的值：

Default **Before** **After** **Both**

`SAPQ_Default`: PointerAlignment。

```
PointerAlignment: Left                 PointerAlignment: Right
```

```
void* const* x = NULL;         vs.     void *const *x = NULL;
```

`SAPQ_Before`:

```
PointerAlignment: Left                 PointerAlignment: Right
```

```
void* const* x = NULL;         vs.     void * const *x = NULL;
```

`SAPQ_After`:

```
PointerAlignment: Left                 PointerAlignment: Right
```

```
void* const * x = NULL;         vs.     void *const *x = NULL;
```

`SAPQ_Both`:

```
PointerAlignment: Left                 PointerAlignment: Right
```

```
void* const * x = NULL;         vs.     void * const *x = NULL;
```

最后更新

2.110 SpaceBeforeAssignmentOperators

2.110.1 SpaceBeforeAssignmentOperators (`Boolean`)

Info

clang-format 3.7

如果为 `false`，则在赋值操作符之前删除空格。

true **false**

```
int a = 5;
a += 42;

int a= 5;
a+= 42;
```

最后更新

2.111 SpaceBeforeCaseColon

2.111.1 SpaceBeforeCaseColon (`Boolean`)

Info

clang-format 12

如果为 `false`，则在 `case`

true **false**

```
switch (x) {
case 1 : break;
}

switch (x) {
case 1: break;
}
```

最后更新

2.112 SpaceBeforeCpp11BracedList

2.112.1 SpaceBeforeCpp11BracedList (`Boolean`)

Info

clang-format 7

如果为 `true`，则在用于初始化对象的 `C++11` ( )

true **false**

```
Foo foo { bar };
Foo {};
vector<int> { 1, 2, 3 };
new int[3] { 1, 2, 3 };

Foo foo{ bar };
Foo{};
vector<int>{ 1, 2, 3 };
new int[3]{ 1, 2, 3 };
```

最后更新

2.113 SpaceBeforeCtorInitializerColon

2.113.1 SpaceBeforeCtorInitializerColon (`Boolean`)

Info

clang-format 7

如果为 `false`，在构造函数初始化器冒号之前的空格将被删除。

true **false**

```
Foo::Foo() : a(a) {}

Foo::Foo(): a(a) {}
```

最后更新

2.114 SpaceBeforeInheritanceColon

2.114.1 SpaceBeforeInheritanceColon (`Boolean`)

Info

clang-format 7

如果为 `false`，在继承冒号之前的空格将被删除。

true **false**

```
class Foo : Bar {}

class Foo: Bar {}
```

最后更新

2.115 SpaceBeforeParens

2.115.1 SpaceBeforeParens (`SpaceBeforeParensOptions`)

Info

clang-format 3.5

定义在何种情况下在开括号前放空格。

可能的值:

Never **ControlStatements** **ControlStatementsExceptForEachMacros** **NonEmptyParentheses** **Always**

`SBPO_Never` :

```
void f() {
    if(true) {
        f();
    }
}
```

`SBPO_ControlStatements` : (`for` / `if` / `while` ...)

```
void f() {
    if (true) {
        f();
    }
}
```

`SBPO_ControlStatementsExceptForEachMacros` : `SBPO_ControlStatements` ForEach ForEach 为函数调用而不是控制语句的项目中非常有用。

```
void f() {
    Q_FOREACH(...) {
        f();
    }
}
```

`SBPO_NonEmptyParentheses` : `()`

```
void() {
    if (true) {
        f();
        g (x, y, z);
    }
}
```

`SBPO_Always` : ( ) ( )

```
void f () {
    if (true) {
        f ();
    }
}
```

最后更新

2.116 SpaceBeforeParensOptions

2.116.1 SpaceBeforeParensOptions (`SpaceBeforeParensCustom`)

Info

clang-format 14

控制括号前的单个空格。

如果 SpaceBeforeParens `Custom`，则使用它来指定应如何处理括号大小写前的每个单独的空格。否则，这将被忽略。

用法示例：

```
SpaceBeforeParens: Custom
SpaceBeforeParensOptions:
  AfterControlStatements: true
  AfterFunctionDefinitionName: true
Nested configuration flags:
```

- `bool AfterControlStatements`： `true`，则在控制语句关键字（ `for` / `if` / `while` ...

```
true:                               false:
if (...) {}                  vs.    if(...) {}
```

- `bool AfterForeachMacros`： `true`，则在 `foreach`

```
true:                               false:
FOREACH (...)                vs.    FOREACH(...)
  <loop-body>                         <loop-body>
```

- `bool AfterFunctionDeclarationName`： `true`，则在函数声明名称和左括号之间放置一个空格。

```
true:                               false:
void f ();                   vs.    void f();
```

- `bool AfterFunctionDefinitionName`： `true`，则在函数定义名称和左括号之间放置一个空格。

```
true:                               false:
void f () {}                 vs.    void f() {}
```

- `bool AfterIfMacros`： `true`，则在 `if`

```
true:                               false:
IF (...)                     vs.    IF(...)
  <conditional-body>                  <conditional-body>
```

- `bool BeforeNonEmptyParentheses`： `true`，则仅当括号不为空时才在括号前放置一个空格。

```
true:                               false:
void f (int a);              vs.    void f();
f (a);                              f();
```

最后更新

2.117 SpaceBeforeRangeBasedForLoopColon

2.117.1 SpaceBeforeRangeBasedForLoopColon (`Boolean`)

Info

clang-format 7

如果为 `false`，在基于范围的 `for loop`

true **false**

```
for (auto v : values) {}

for(auto v: values) {}
```

最后更新

2.118 SpaceBeforeSquareBrackets

2.118.1 SpaceBeforeSquareBrackets (`Boolean`)

Info

clang-format 11

如果为 `true`，空格将在 `[` `[`

true **false**

```
int a [5];
int a [5][5];

int a[5];
int a[5][5];
```

最后更新

2.119 SpaceInEmptyBlock

2.119.1 SpaceInEmptyBlock (`Boolean`)

Info

clang-format 11

如果为 `true`，将在 `{}`

true **false**

```
void f() { }
while (true) { }

void f() {}
while (true) {}
```

最后更新

2.120 SpaceInEmptyParentheses

2.120.1 SpaceInEmptyParentheses (`Boolean`)

Info

clang-format 3.7

如果为 `true`，可以在 `()`

true **false**

```
void f( ) {
    int x[] = {foo( ), bar( )};
    if (true) {
        f( );
    }
}

void f() {
    int x[] = {foo(), bar()};
    if (true) {
        f();
    }
}
```

最后更新

2.121 SpacesBeforeTrailingComments

2.121.1 SpacesBeforeTrailingComments (`Unsigned`)

Info

clang-format 3.7

尾行注释 `//` -   )

这不会影响末尾的块注释 `/*` -   )

```
SpacesBeforeTrailingComments: 3
```

```
void f() {
    if (true) {   // foo1
        f();      // bar
    }             // foo
}
```

最后更新

2.122 SpacesInAngles

2.122.1 SpacesInAngles (`SpacesInAnglesStyle`)

Info

clang-format 14

用于模板参数列表的 `SpacesInAnglesStyle`。

可能的值:

Never **Always** **Leave**

`SIAS_Never`: `<` `>`

```
static_cast<int>(arg);
std::function<void(int)> fct;
```

`SIAS_Always`: `<` `>`

```
static_cast< int >(arg);
std::function< void(int) > fct;
```

`SIAS_Leave`: `<` `Cpp03`

最后更新

2.123 SpacesInCStyleCastParentheses

2.123.1 SpacesInCStyleCastParentheses (`Boolean`)

Info

clang-format 3.7

如果为 `true`，则可以在 `C`

true **false**

```
x = ( int32 )y

x = (int32)y
```

最后更新

2.124 SpacesInConditionalStatement

2.124.1 SpacesInConditionalStatement (`Boolean`)

Info

clang-format 11

如果为 `true`，则在 `if` / `for` / `switch` / `while`

true	false

```
if ( a )  { ... }
while ( i < 5 )  { ... }

if (a) { ... }
while (i < 5) { ... }
```

最后更新

2.125 SpacesInContainerLiterals

2.125.1 SpacesInContainerLiterals (`Boolean`)

Info

clang-format 3.7

如果为 `true`，则在容器字面量　例如`ObjC`　`Javascript`　　`dict`　　)

true　**false**

```
var arr = [ 1, 2, 3 ];
f({a : 1, b : 2, c : 3});

var arr = [1, 2, 3];
f({a: 1, b: 2, c: 3});
```

最后更新

2.126 SpacesInLineCommentPrefix

2.126.1 SpacesInLineCommentPrefix (`SpacesInLineComment`)

Info

clang-format 14

一行注释的开头允许有多少个空格。要禁用最大值，请将其设置为 `-1`，但最大值优先于最小值。

```
Minimum = 1 Maximum = -1 //

//

Minimum = 0 Maximum = 0 //
```

请注意，在行注释部分，后续行的相对缩进被保留，这意味着如下：

```
before:                                    after:
Minimum: 1
//if (b) {                                 // if (b) {
//  return true;                           //   return true;
//}                                        // }

Maximum: 0
/// List:                                  ///List:
///  - Foo                                 /// - Foo
///    - Bar                               ///   - Bar
```

嵌套的配置标记：

- `unsigned Minimum`
- `unsigned Maximum`

最后更新

2.127 SpacesInParentheses

2.127.1 SpacesInParentheses (`Boolean`)

Info

clang-format 3.7

如果为 `true`，则在前后插入空格。

true **false**

```
t f( Deleted & ) & = delete;

t f(Deleted &) & = delete;
```

最后更新

2.128 SpacesInSquareBrackets

2.128.1 SpacesInSquareBrackets (`Boolean`)

Info

clang-format 3.7

如果为 `true`，将在 `[` `]`

true **false**

```
int a[ 5 ];
std::unique_ptr<int[]> foo() {} // Won't be affected

int a[5];
```

最后更新

2.129 Standard

2.129.1 Standard (`LanguageStandard`)

Info

clang-format 3.7

解析和格式化与这个标准兼容的 `C++`

c++03 **latest**

```
vector<set<int> > x;

vector<set<int>> x;
```

可能的值:

Cpp03 **Cpp11** **Cpp14** **Cpp17** **Cpp20** **Latest** **Auto**

`LS_Cpp03` : `C++03`。 `Cpp03` `c++03`

`LS_Cpp11` : `C++11`。

`LS_Cpp14` : `C++14`。

`LS_Cpp17` : `C++17`。

`LS_Cpp20` : `C++20`。

`LS_Latest` : `Cpp11`

`LS_Auto` :

最后更新

2.130 StatementAttributeLikeMacros

2.130.1 StatementAttributeLikeMacros (`List of Strings`)

Info

clang-format 12

在语句前面被忽略的宏，就像它们是一个属性一样。这样它们就不会被解析为标识符，例如 `Qts emit`。

```
AlignConsecutiveDeclarations: true
StatementAttributeLikeMacros: []
```

```
unsigned char data = 'x';
emit          signal(data); //
```

```
AlignConsecutiveDeclarations: true
StatementAttributeLikeMacros: [emit]
```

```
unsigned char data = 'x';
emit signal(data); //
```

最后更新

2.131 StatementMacros

2.131.1 StatementMacros (`List of Strings`)

Info

clang-format 8

宏的一个向量，应该被解释为完整的语句。

典型的宏是表达式，需要添加分号 有时情况并非如此，这允许`clang-format`

例如 `Q_UNUSED`

最后更新

2.132 TabWidth

2.132.1 TabWidth (`Unsigned`)

Info

clang-format 3.7

用于制表位的列数。

最后更新

2.133 TypenameMacros

2.133.1 TypenameMacros (`List of Strings`)

Info

clang-format 3.7

宏的一个向量，它应该被解释为类型声明而不是函数调用。

这些应该是以下形式的宏：

```
STACK_OF(...)
```

在 `.clang-format`

```
TypenameMacros: ['STACK_OF', 'LIST']
```

例如 `OpenSSL STACK_OF` , `BSD LIST_ENTRY`

最后更新

2.134 UseCRLF

2.134.1 UseCRLF (`Boolean`)

> **Info**
>
> clang-format 11

用 `\r\n` `\n` (DeriveLineEnding)(../DeriveLineEnding) `true`，也用作回退。

最后更新

2.135 UseTab

2.135.1 UseTab (`UseTabStyle`)

Info

clang-format 3.7

在结果文件中使用制表符的方法。

可能的值:

Never **ForIndentation** **ForContinuationAndIndentation** **AlignWithSpaces** **Always**

`UT_Never` :

`UT_ForIndentation` :

`UT_ForContinuationAndIndentation` : (|)

`UT_AlignWithSpaces` :

`UT_Always` :

最后更新

2.136 WhitespaceSensitiveMacros

2.136.1 WhitespaceSensitiveMacros (`List of Strings`)

Info

clang-format 12

一个宏向量，它是空格敏感的，不应该被触及。

这些应该是以下形式的宏：

```
STRINGIZE(...)
```

在 `.clang-format`

```
WhitespaceSensitiveMacros: ['STRINGIZE', 'PP_STRINGIZE']
```

例如 `BOOST_PP_STRINGIZE`。

最后更新