clang-format 的介紹和使用

阿新 • 來源：網路 • 發佈：2020-07-16

目錄

參考資訊

- 官方參考：https://clang.llvm.org/docs/ClangFormat.html
- 入門部落格參考：https://blog.csdn.net/softimite_zifeng/article/details/78357898

介紹

OVERVIEW: A tool to format C/C++/Java/JavaScript/Objective-C/Protobuf/C# code.

Clang-Format 可用於格式化（排版）多種不同語言的程式碼。

其自帶的排版格式主要有：LLVM, Google, Chromium, Mozilla, WebKit 等

若-style=google ，則表示應用 Google 的格式化風格

安裝

請看官網上的安裝方式，或者 Google 一下你的 OS 安裝方式吧。

這裡給出我常用 OS 安裝方式：

macOS10.14.6 brew install clang-format

ubuntu18.04 sudo apt install clang-format

命令格式

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

clang-format --help 建議至少瀏覽一遍幫助資訊。

基本使用

```
// 以 LLVM 程式碼風格格式化 main.cpp，結果輸出到 stdout
clang-format -style=LLVM main.cpp

// 以 LLVM 程式碼風格格式化 main.cpp，結果直接寫到 main.cpp
clang-format -style=LLVM -i main.cpp

// 當然也支援對指定行格式化，格式化 main.cpp 的第 1，2 行
clang-format -lines=1:2 main.cpp
```

使用.clang-format 來實現自定義格式化

匯出.clang-format 檔案

When the desired code formatting style is different from the available options, the style can be customized using the -style="{key:

value, ...}" option or by putting your style configuration in the .clang-format or _clang-format file in your project's directory and using clang-format -style=file.

An easy way to create the .clang-format file is:

```
clang-format -style=可選格式名 -dump-config > .clang-format
```

```
# 可選格式最好寫預設那那幾個寫最接近你想要的格式. 比如我想要接近 google C++ style 的。 我就寫-style=google
```

看了官網的介紹，我們知道我們可以使用.clang-format 檔案來自定義格式。在使用時不再是-style=llvm 等內建可選的那幾種的格式。

使用我們自定義.clang-formart 檔案來格式化。指定方式為-style=file

> 注意：一般取.clang-format 或_clang-format，因為自定義的排版格式檔案只有取這兩種名字之一，才能被 Clang-Format 識別。

現在對生成的.clang-format 檔案根據自己需求進行修改吧

使用.clang-format 檔案

直接將修改後的檔案放在和程式碼檔案相同的資料夾中，並且設定格式化選項-style=file，即可以使用自定義的排版格式。

VS Code 只要將該檔案放在和程式碼檔案相同的資料夾中即可，不需要額外的設定。

格式化檔案放在程式碼檔案的上一級資料夾中，也可以使用。(個人建議放在專案的根目錄下。我看不少開源專案就是這麼幹的，嘿嘿嘿)

注意，檔名必須為.clang-format 或_clang-format。

```
# 格式化的結果打到 stdout(終端上)
clang-format -style=file main.cc

# 直接修改到檔案
clang-format -style=file -i main.cc
```

.clang-format 組態檔案的各個選項的含義

```
---

# 語言: None, Cpp, Java, JavaScript, ObjC, Proto, TableGen, TextProto
Language:   Cpp

# BasedOnStyle: LLVM

# 訪問說明符(public、private 等)的偏移
AccessModifierOffset:   -4

# 開括號(開圓括號、開尖括號、開方括號)後的對齊: Align, DontAlign, AlwaysBreak(總是在開括號後換行)
AlignAfterOpenBracket:  Align

# 連續賦值時，對齊所有等號
AlignConsecutiveAssignments:    true

# 連續宣告時，對齊所有宣告的變數名
AlignConsecutiveDeclarations:   true

# 左對齊逃脫換行(使用反斜槓換行)的反斜槓
AlignEscapedNewlinesLeft:   true

# 水平對齊二元和三元表示式的運算元
AlignOperands:  true

# 對齊連續的尾隨的註釋
AlignTrailingComments:  true

# 允許函式宣告的所有引數在放在下一行
AllowAllParametersOfDeclarationOnNextLine:  true

# 允許短的塊放在同一行
AllowShortBlocksOnASingleLine:  false

# 允許短的 case 標籤放在同一行
AllowShortCaseLabelsOnASingleLine:  false
```

```
# 允許短的函式放在同一行: None, InlineOnly(定義在類中), Empty(空函式), Inline(定義在類中，空函式), All
AllowShortFunctionsOnASingleLine:   Empty

# 允許短的 if 語句保持在同一行
AllowShortIfStatementsOnASingleLine:    false

# 允許短的迴圈保持在同一行
AllowShortLoopsOnASingleLine:   false

# 總是在定義返回型別後換行(deprecated)
AlwaysBreakAfterDefinitionReturnType:   None

# 總是在返回型別後換行: None, All, TopLevel(頂級函式，不包括在類中的函式),
#   AllDefinitions(所有的定義，不包括宣告), TopLevelDefinitions(所有的頂級函式的定義)
AlwaysBreakAfterReturnType: None

# 總是在多行 string 字面量前換行
AlwaysBreakBeforeMultilineStrings:  false

# 總是在 template 宣告後換行
AlwaysBreakTemplateDeclarations:    false

# false 表示函式實參要麼都在同一行，要麼都各自一行
BinPackArguments:   true

# false 表示所有形參要麼都在同一行，要麼都各自一行
BinPackParameters:  true

# 大括號換行，只有當 BreakBeforeBraces 設定為 Custom 時才有效
BraceWrapping:

  # class 定義後面
  AfterClass:   false

  # 控制語句後面
  AfterControlStatement:    false
```

```
  # enum 定義後面
  AfterEnum:    false

  # 函式定義後面
  AfterFunction:    false

  # 名稱空間定義後面
  AfterNamespace:   false

  # ObjC 定義後面
  AfterObjCDeclaration: false

  # struct 定義後面
  AfterStruct:  false

  # union 定義後面
  AfterUnion:   false

  # catch 之前
  BeforeCatch:  true

  # else 之前
  BeforeElse:   true

  # 縮排大括號
  IndentBraces: false

# 在二元運運算元前換行: None(在運算子後換行), NonAssignment(在非賦值的運算子前換行), All(在運算子前換行)
BreakBeforeBinaryOperators: NonAssignment

# 在大括號前換行: Attach(始終將大括號附加到周圍的上下文), Linux(除函式、名稱空間和類定義，與 Attach 類似),
#   Mozilla(除列舉、函式、記錄定義，與 Attach 類似), Stroustrup(除函式定義、catch、else，與 Attach 類似),
#   Allman(總是在大括號前換行), GNU(總是在大括號前換行，並對於控制語句的大括號增加額外的縮排), WebKit(在函式前換行), Custom
#   註：這裡認為語句塊也屬於函式
BreakBeforeBraces:  Custom
```

```
# 在三元運運算元前換行
BreakBeforeTernaryOperators:    true

# 在建構函式的初始化列表的逗號前換行
BreakConstructorInitializersBeforeComma:    false

# 每行字元的限制，0 表示沒有限制
ColumnLimit:    200

# 描述具有特殊意義的註釋的正則表示式，它不應該被分割為多行或以其它方式改變
CommentPragmas: '^ IWYU pragma:'

# 建構函式的初始化列表要麼都在同一行，要麼都各自一行
ConstructorInitializerAllOnOneLineOrOnePerLine: false

# 建構函式的初始化列表的縮排寬度
ConstructorInitializerIndentWidth:  4

# 延續的行的縮排寬度
ContinuationIndentWidth:    4

# 去除 C++11 的列表初始化的大括號{後和}前的空格
Cpp11BracedListStyle:   false

# 繼承最常用的指標和引用的對齊方式
DerivePointerAlignment: false

# 關閉格式化
DisableFormat:  false

# 自動檢測函式的呼叫和定義是否被格式為每行一個引數(Experimental)
ExperimentalAutoDetectBinPacking:   false

# 需要被解讀為 foreach 迴圈而不是函式呼叫的巨集
ForEachMacros:  [ foreach, Q_FOREACH, BOOST_FOREACH ]

# 對#include 進行排序，匹配了某正則表示式的#include 擁有對應的優先順序，匹配不到的則
預設優先順序為 INT_MAX(優先順序越小排序越靠前)，
#    可以定義負數優先順序從而保證某些#include 永遠在最前面
IncludeCategories:
```

```
  - Regex:  '^"(llvm|llvm-c|clang|clang-c)/'
    Priority:   2
  - Regex:  '^(<|"(gtest|isl|json)/)'
    Priority:   3
  - Regex:  '.*'
    Priority:   1

# 縮排 case 標籤
IndentCaseLabels:   false

# 縮排寬度
IndentWidth:    4

# 函式返回型別換行時，縮排函式宣告或函式定義的函式名
IndentWrappedFunctionNames: false

# 保留在塊開始處的空行
KeepEmptyLinesAtTheStartOfBlocks:   true

# 開始一個塊的巨集的正則表示式
MacroBlockBegin:    ''

# 結束一個塊的巨集的正則表示式
MacroBlockEnd:  ''

# 連續空行的最大數量
MaxEmptyLinesToKeep:    1

# 名稱空間的縮排: None, Inner(縮排巢狀的名稱空間中的內容), All
NamespaceIndentation:   Inner

# 使用 ObjC 塊時縮排寬度
ObjCBlockIndentWidth:   4

# 在 ObjC 的@property 後新增一個空格
ObjCSpaceAfterProperty: false

# 在 ObjC 的 protocol 列表前新增一個空格
ObjCSpaceBeforeProtocolList:    true
```

```
# 在 call( 後對函式呼叫換行的 penalty
PenaltyBreakBeforeFirstCallParameter:   19

# 在一個註釋中引入換行的 penalty
PenaltyBreakComment:    300

# 第一次在<<前換行的 penalty
PenaltyBreakFirstLessLess:  120

# 在一個字串字面量中引入換行的 penalty
PenaltyBreakString: 1000

# 對於每個在行字元數限制之外的字元的 penalty
PenaltyExcessCharacter: 1000000

# 將函式的返回型別放到它自己的行的 penalty
PenaltyReturnTypeOnItsOwnLine:  60

# 指標和引用的對齊: Left, Right, Middle
PointerAlignment:   Left

# 允許重新排版註釋
ReflowComments: true

# 允許排序#include
SortIncludes:   true

# 在 C 風格型別轉換後新增空格
SpaceAfterCStyleCast:   false

# 在賦值運運算元之前新增空格
SpaceBeforeAssignmentOperators: true

# 開圓括號之前新增一個空格: Never, ControlStatements, Always
SpaceBeforeParens:  ControlStatements

# 在空的圓括號中新增空格
SpaceInEmptyParentheses:    false
```

```
# 在尾隨的評論前新增的空格數(只適用於//)
SpacesBeforeTrailingComments:   2

# 在尖括號的<後和>前新增空格
SpacesInAngles: true

# 在容器(ObjC 和 JavaScript 的陣列和字典等)字面量中新增空格
SpacesInContainerLiterals:  true

# 在 C 風格型別轉換的括號中新增空格
SpacesInCStyleCastParentheses:  true

# 在圓括號的(後和)前新增空格
SpacesInParentheses:    true

# 在方括號的[後和]前新增空格，lamda 表示式和未指明大小的陣列的宣告不受影響
SpacesInSquareBrackets: true

# 標準: Cpp03, Cpp11, Auto
Standard:   Cpp11

# tab 寬度
TabWidth:   4

# 使用 tab 字元: Never, ForIndentation, ForContinuationAndIndentation, Always
UseTab: Never
```

整合到 vim 中(官方給出了各種 IDE 或編輯器整合方式)

詳細看官方網站

常用命令

```
find . -regex '.*\.\(cpp\|hpp\|cu\|c\|h\)' -exec clang-format -style=file
-i {} \;
```

clang-format-diff.py 指令碼

clang-format 還提供一個 clang-format-diff.py 指令碼，用來格式化 patch，code review 提交程式碼前，可以先跑一下這個指令碼，看有沒有問題。

```
// 格式化最新的 commit，並直接在原檔案上修改
git diff -U0 HEAD^ | clang-format-diff.py -i -p1
```

官網寫了 git svn Mercurial/hg 的使用，詳細請到官網看吧