Hands on clang-format

Adding (n)one feature at a time

Kai Wolf
kai-wolf.me

March 31, 2016

A quick reality check - please raise hand

- We have a (living) coding style guide
- We use code generators
- We do (semi-)automatized refactoring (clang-tidy, Clang MapReduce, awk | grep | sed, ...)
- We use automatic code formatting tools
- We utilize pre/post-commit hooks for these kind of stuff

Why is it important?

- Inconsistent formatting distracts the reader
- Formatting code by hand is tedious
- Nobody reads the style guide anyways
- Automatic/large scale refactoring not feasible without automatic code formatting

Lots of auto formatting tools available

Artistic Style (AStyle)

Works. Focuses only on indentation and brace/bracket placement

Uncrustify

Many, many, many rules (around ~200). Might take a whole day to adjust to own style guide.

GNU indent

Works most of the time (see OpenSSL Blog - CodeReformat Finished). Basically has three presets available (Ansi, GNU, K&R). Indent needs to know about all types used within the code.

GC GreatCode

Intrusive formatting rules. May break your code.

How to use clang-format

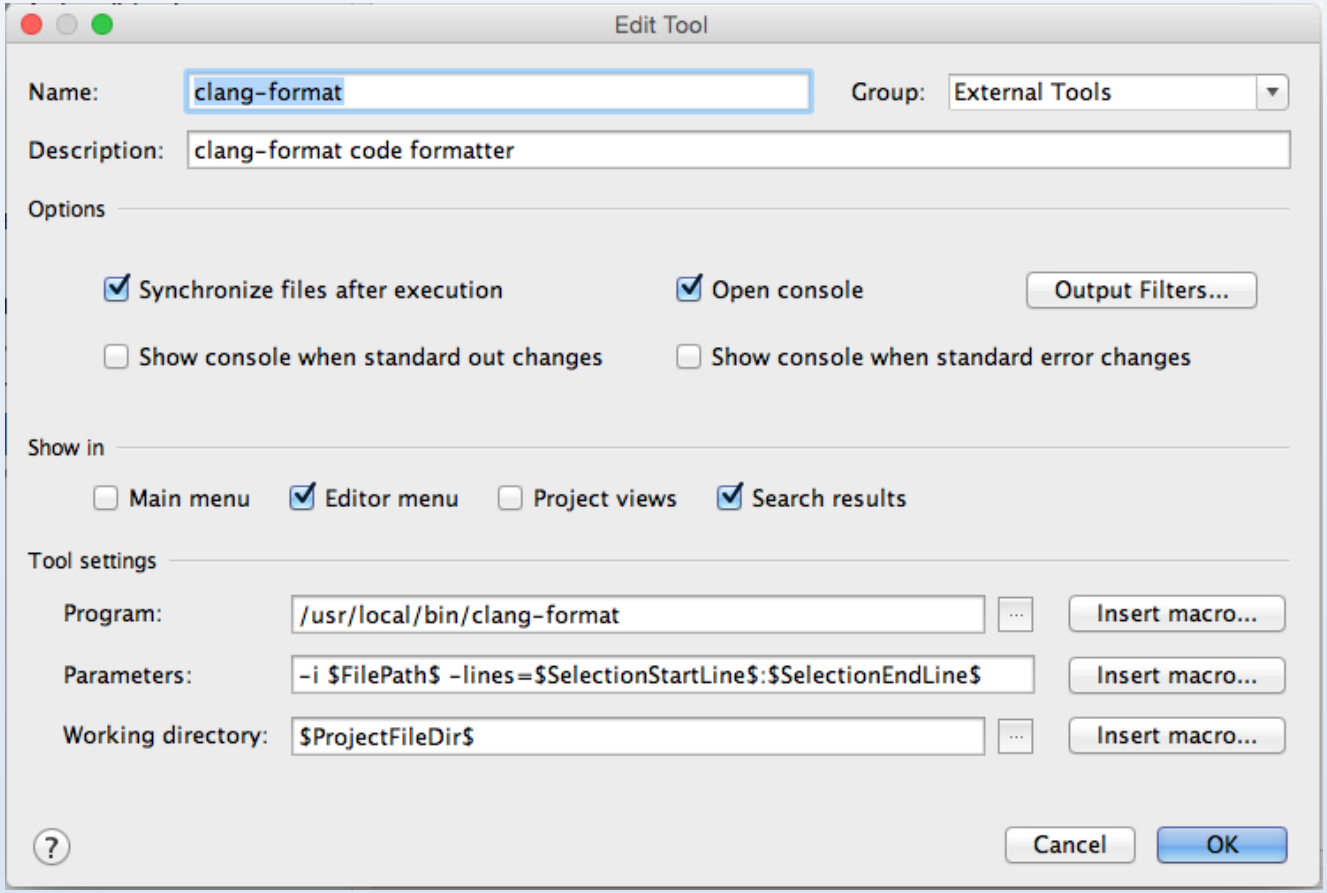
```
$ cat bbox.cpp

cv::Rect BoundingBox::getBoundingRect(const cv::Mat& Binary) const {

      assert(Binary.channels() == 1);
        using Contour = std::vector<cv::Point>;

          std::vector<Contour> contours;
auto hierarchy = std::vector<cv::Vec4i>();
              cv::findContours(Binary.clone(),contours,hierarchy,CV_RETR_TREE,
       CV_CHAIN_APPROX_SIMPLE);

          auto result = *std::max_element(contours.begin(), contours.end(),
   [](Contour a, Contour b) {
              return cv::contourArea(a) < cv::contourArea(b);});
        return cv::boundingRect(cv::Mat(result));}
```

How to use clang-format

```
$ cat bbox.cpp | clang-format -style='{BasedOnStyle: Google, IndentWidth: 4'

cv::Rect BoundingBox::getBoundingRect(const cv::Mat& Binary) const {
    assert(Binary.channels() == 1);
    using Contour = std::vector<cv::Point>;

    std::vector<Contour> contours;
    auto hierarchy = std::vector<cv::Vec4i>();
    cv::findContours(Binary.clone(), contours, hierarchy, CV_RETR_TREE,
                     CV_CHAIN_APPROX_SIMPLE);

    auto result = *std::max_element(
        contours.begin(), contours.end(), [](Contour a, Contour b) {
            return cv::contourArea(a) < cv::contourArea(b);
        });
    return cv::boundingRect(cv::Mat(result));
}
```

How to use clang-format

Plugins available for Vi, Emacs, Visual Studio etc. Custom integration almost always possible. For example: CLion



How to use clang-format

There are many deliberate options available:

Release	Number of options
clang-format v3.5	53
clang-format v3.6	62
clang-format v3.7	65
clang-format v3.8	70
clang-format v3.9	75

Motivating example

Say for example, our style guide suggests that this:

```
void foo(int a, int b, double c, const char* str);
```

should be formatted like this:

```
void foo(int a,int b,double c,const char* str);
```

What does clang-format offer here?

SpaceAfterCStyleCast (`bool`)

If `true`, a space may be inserted after C style casts.

SpaceBeforeAssignmentOperators (`bool`)

If `false`, spaces will be removed before assignment operators.

SpaceBeforeParens (`SpaceBeforeParensOptions`)

Defines in which cases to put a space before opening parentheses.

SpaceInEmptyParentheses (`bool`)

If `true`, spaces may be inserted into `()`.

SpacesBeforeTrailingComments (`unsigned`)

The number of spaces before trailing line comments (`//` - comments).

SpacesInAngles (`bool`)

If `true`, spaces will be inserted after `<` and before `>` in template argument lists.

SpacesInCStyleCastParentheses (`bool`)

If `true`, spaces may be inserted into C style casts.

SpacesInContainerLiterals (`bool`)

If `true`, spaces are inserted inside container literals (e.g. ObjC and Javascript array and dict literals).

SpacesInParentheses (`bool`)

If `true`, spaces will be inserted after `(` and before `)`.

SpacesInSquareBrackets (`bool`)

If `true`, spaces will be inserted after `[` and before `]`.

Let's try to fix that

In order to start hacking we need the llvm repository with the following subprojects:

```
$ export LLVM_SVN=http://llvm.org/svn/llvm-project
$ svn co ${LLVM_SVN}/llvm/trunk llvm
$ cd llvm/tools
$ svn co ${LLVM_SVN}/cfe/trunk clang
$ cd ../projects
$ svn co ${LLVM_SVN}/test-suite/trunk test-suite
$ cd ../tools/clang/tools
$ svn co ${LLVM_SVN}/clang-tools-extra/trunk extra
```

Navigate around the code base

Formatting rules for clang-format are located in

```
llvm/tools/clang/lib/Format/
```

Unittests are located in

```
llvm/tools/clang/unittests/Format/
```

Frontend for clang-format

```
$ cd llvm/tools/clang # clang subproject
$ cat tools/clang-format/ClangFormat.cpp

int main(int argc, const char **argv) {
...
  for (unsigned i = 0; i < FileNames.size(); ++i)
    Error |= clang::format::format(FileNames[i]);
```

clang::format::format

```
static bool format(StringRef FileName) {
  ...
  if (fillRanges(Code.get(), Ranges))
    return true;
  ...
  FormatStyle FormatStyle =
    getStyle(Style, AssumedFileName, FallbackStyle)
  ...
  Replacements Replaces =
    sortIncludes(FormatStyle, Code, Ranges)
  ...
  Replacements FormatChanges =
    reformat(FormatStyle, ChangedCode, Ranges)
}
```

clang::format::reformat

```
...
for (const tooling::Range &Range : Ranges) {
  SourceLocation Start =
      StartOfFile.getLocWithOffset(
          Range.getOffset());
  SourceLocation End =
      Start.getLocWithOffset(Range.getLength());
  CharRanges.push_back(
      CharSourceRange::getCharRange(Start, End));
}
return reformat(Style, SourceMgr, ID, CharRanges,
                IncompleteFormat);
```

clang::format::reformat

```
FormatStyle Expanded = expandPresets(Style);
if (Expanded.DisableFormat)
  return tooling::Replacements();
Formatter formatter(Expanded, SourceMgr, ID,
                    Ranges);
return formatter.format(IncompleteFormat);
```

clang::tok::TokenKind

```
void foo(int a, int b);
```

```
UnwrappedLineParser Parser();
  -> readTokens();
  -> parseFile();
    -> parseLevel();
    -> parseStructuralElement();
<- AnnotatedLines
```

0. kw_void
1. identifier
2. l_param
3. kw_int
4. identifier
5. comma
6. kw_int
7. identifier
8. r_param
9. semi

clang::format

For each annotated line (`TokenAnnotator.cpp`):

```
void TokenAnnotator::
  calculateFormattingInformation(
      AnnotatedLine &Line);
```

Fortunately not much to do here for us:

```
bool TokenAnnotator::spaceRequiredBetween(
  const AnnotatedLine &Line,
  const FormatToken &Left,
  const FormatToken &Right);
```

Our patch (excerpt)

Format.h

```
/// \b If \c true, spaces will be inserted between
/// function parameters.
bool SpacesBetweenFunctionParameters;
```

TokenAnnotator.cpp

```
...
if (Right.is(TT_OverloadedOperatorLParen))
  return Style.SpaceBeforeParens ==
         FormatStyle::SBPO_Always;
if (Left.is(tok::comma))
  // return true;
  return Style.SpacesBetweenFunctionParameters;
if (Right.is(tok::comma))
  return false;
```

Preparing the patch

Tests are looking good.

```
TEST_F(FormatTest, SpacesBetweenFunctionParameters) {
  FormatStyle Spaces = getLLVMStyle();
  Spaces.SpacesBetweenFunctionParameters = false;

  verifyFormat("void foo(int a);", Spaces);
  verifyFormat("void foo(int a,double b);", Spaces);
  verifyFormat("void foo(int a,double b,char c);", Spaces);

  Spaces.SpacesBetweenFunctionParameters = true;
  verifyFormat("void foo(int a, double b);", Spaces);
  verifyFormat("void foo(int a, double b, char c);", Spaces);
}
```

By the way: If you manage an open source project, make sure to explain how to contribute!

- How to get going? How to submit your patch?
- Who decides what patches are accepted?
- Who should review your patch?
- What should be included in the patch? (doc, unittests, full file context diff etc.)

Oops..

Differential > D16765

[Clang-Format] Add option for spacing between function parameters

Needs Review Public

Author NewProggie
Reviewers poiru
klimek
djasper
Subscribers klimek, cfe-commits
Projects *None*

Press ? to show keyboard shortcuts.

Edit Revision
Update Diff
Edit Dependencies
Download Raw Diff
Automatically Subscribed
Award Token
Flag For Later

SUMMARY

This patch adds an option to remove the blank after comma between several function parameters. The default value is still to keep the blank.

Patch by: Kai Wolf

Diff Detail

Diff 46518

djasper added a comment. Via Web · Feb 1 2016, 2:43 AM

Please read http://clang.llvm.org/docs/ClangFormatStyleOptions.html#adding-additional-style-options

http://reviews.llvm.org/D16765

That paragraph must be new...

Adding additional style options

Each additional style option adds costs to the clang-format project. Some of these costs affect the clang-format development itself, as we need to make sure that any given combination of options work and that new features don't break any of the existing options in any way. There are also costs for end users as options become less discoverable and people have to think about and make a decision on options they don't really care about.

The goal of the clang-format project is more on the side of supporting a limited set of styles really well as opposed to supporting every single style used by a codebase somewhere in the wild. Of course, we do want to support all major projects and thus have established the following bar for adding style options. Each new style option must ..

- be used in a project of significant size (have dozens of contributors)
- have a publicly accessible style guide
- have a person willing to contribute and maintain patches

Yes, indeed

By the time I've created the patch, clang-format 3.7 was the current release

63.249.66.211/comparison.html

ChangeDetection.com

Automatic website change logs and notification | Back | Home | Webmasters | Directory | Compare | Help

Changes Show: show full html text extracts full text skip to 1st change
Options: show added and deleted only added only deleted

Clang 3.9~~3.7~~ documentation

CLANG-FORMAT STYLE OPTIONS

« ClangFormat :: Contents :: "Clang" CFE Internals Manual »

Clang-Format Style Options

Adding additional style options

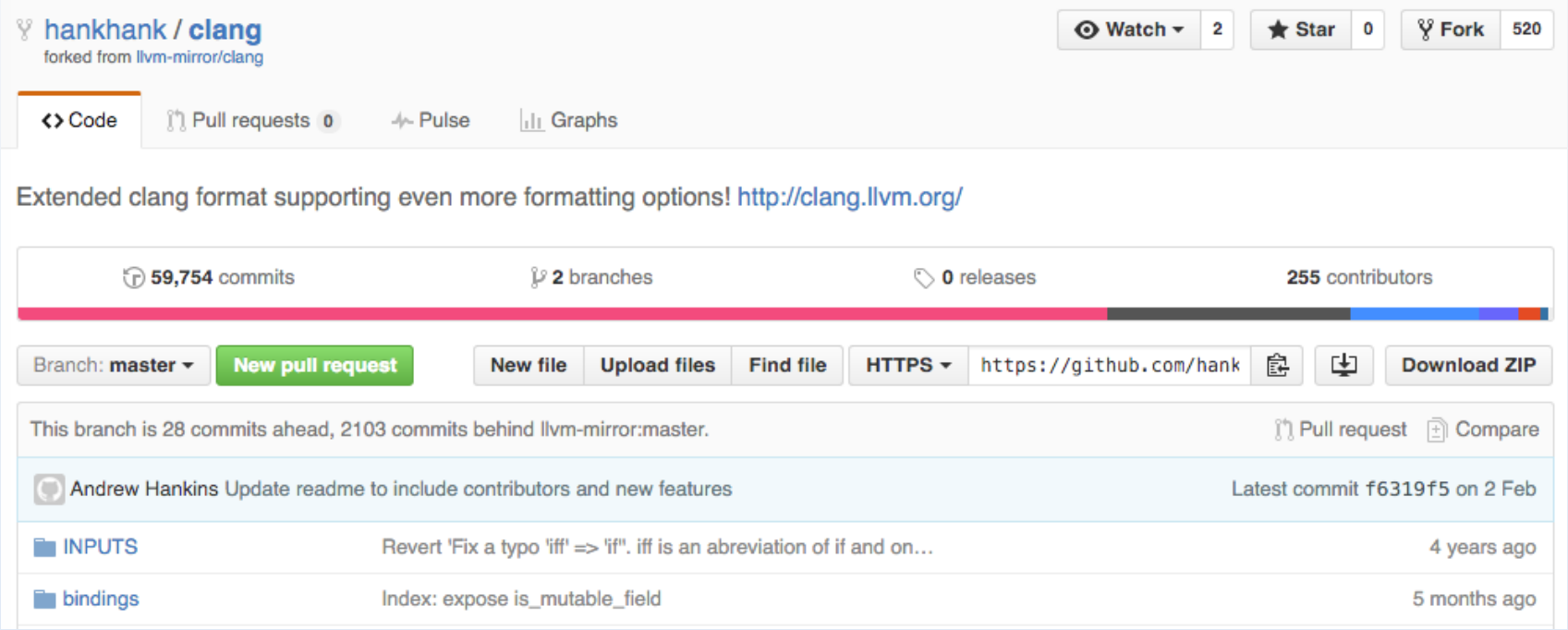
Each additional style option adds costs to the clang-format project. Some of these costs affect the clang-format development itself, as we need to make sure that any given combination of options work and that new features don't break any of the existing options in any way. There are also costs for end users as options become less discoverable and people have to think about and make a decision on options they don't really care about.

The goal of the clang-format project is more on the side of supporting a limited set of styles really well as opposed to supporting every single style used by a codebase somewhere in the wild. Of course, we do want to support all major projects and thus have established the following bar for adding style options. Each new style option must ..

- be used in a project of significant size (have dozens of contributors)
- have a publicly accessible style guide
- have a person willing to contribute and maintain patches

Examples

Yet there's hope..



https://github.com/hankhank/clang

I don’t always accidentally...

Add option for spacing between function parameters #1

Merged **hankhank** merged 1 commit into `hankhank:master` from `unknown repository` on 2 Feb

Conversation 1 | Commits 1 | Files changed 5

NewProggie commented on 1 Feb

This patch adds an option to remove the blank after comma between several function parameters. The default value is still to keep the blank.

Turns out the clang guys won't allow to get their "official" clang-format options polluted. Therefore I'm committing my contributions to this repo instead.

`Add option for spacing between function parameters` ··· `030f4f3`

hankhank commented on 2 Feb — Owner

Thanks for the pull request!
Nice feature - surprising it was missing

hankhank merged commit **7c2b04e** into `hankhank:master` on 2 Feb


THANKS
ANY QUESTIONS?
imgflip.com