CLANG-TIDY & CLANG-FORMAT

St. Louis C++ Meetup

April 10, 2019

John Fultz

Reminder…what is clang?

- Compiler front end to LLVM
- A part of the LLVM distribution
- Available for all major desktop platforms
- Built into Xcode, integration with Visual Studio possible
- Embeddable into other applications via libTooling
- This talk is based on LLVM 8

The basic clang-format styles

- LLVM
- Google
- Chromium
- Mozilla
- WebKit
- Microsoft (coming in LLVM 9)

LLVM style

- Brace style: attached
- Pointer alignment: always right
- Allow short template decls, functions on a single line

Google style

- Brace style: attached
- Pointer alignment: derived, then left
- Allow short conditionals, loops, functions on a single line
- Single-line constructors
- Indented case statements
- 1 space access modifier indent

Chromium style

- Brace style: attached
- Pointer alignment: always left
- Allow short inline functions on a single line
- Parameters are all on one line or one per line

Mozilla style

- Brace style: Mozilla
- Pointer alignment: always left
- Allow inline functions on a single line
- Indented case statements
- Break after return type for function decls

WebKit style

- Brace style: WebKit
- Pointer alignment: always left
- Allow short template decls, functions on a single line
- Break constructor initializers *before* commas and colons
- Break *before* binary operators
- No column limits
- 4 space indents

Using clang-format

- Configurable via the `.clang-format` project file
  - Looks for the file relative to the target, so directory-specific formatting is possible
  - File is in YAML format. Settings are at https://clang.llvm.org/docs/ClangFormatStyleOptions.html
  - You can dump a starting point file using -dump-config. Dump an entire style by using -dump-config -style=<style>.
- Run with `-i` command-line option to format in place
- Most development environments support some level of integration
  - Often, the integration allows formatting just a selection, or the current line

Calling out notable integrations

- Microsoft Visual Studio 2017 ships with [an older version] of clang-format, and can auto-format on save
- Xcode, surprisingly, has no clang-format support. Options:
  - Old versions – https://github.com/travisjeffery/ClangFormat-Xcode
    - Can't run in Xcode versions >8 without violating Xcode's code signature
    - Probably doesn't work in latest 9 or 10
  - New versions – https://github.com/mapbox/XcodeClangFormat
    - Due to plugin limitations, can't read the .clang-format file
    - So, one global style for all your Xcode usage
  - "Behavior" – https://github.com/sean-parent/tools/
    - Adds custom keyboard command to format the selection
    - Only works on the selection, and doesn't format-on-save

Clang-tidy: What it is

- It has a full AST representing your C++ code
- Furthermore, the AST represents individual bits of C++ syntax, including comments and keywords
- It can understand what macros mean. But it also understands that something was accomplished via macro
- It has large parts of clang inside of it
  - It accepts many command-line arguments that clang does
  - It emits compiler errors and warnings just like clang
  - It has clang's static analyzer in it…and turned on by default

Clang-tidy: What it is not

- It's not a linker; it only understands your code in translation units
- It's not a compiler, so rewrite rules can be applied even if there are compiler warnings
- It's not omniscient; if code conditionally compiles, then it only rewrites code if it sets the condition
- It can't read minds and fix bugs.
  - It can do (nearly) perfect refactoring rewrites, but it also detects issues which it can't rewrite because the rewrites might destroy information
- It does not format your code
  - Often want to use in combination with clang-format
- Its rewrite rules have limitations

The perils of regexing your code

A sample code refactor we wanted to do:

`readability-else-after-return`

Before

```
if (cond1)
{
    do_work();
    return true;
}
else
{
    do_other_work();
}
return false;
```

After

```
if (cond1)
{
    do_work();
    return true;
}

do_other_work();
return false;
```

The reg-ex attempt

- Method grep:
  - search: `(return\W[^\{\}\n]+;\s*)\n(\s*)else `
  - replace: `\1\n\2`

The reg-ex attempt

- Method grep:
  - search: `(return\W[^\{\}\n]+;\s*)\n(\s*)else `
  - replace: `\1\n\2`

A resulting change;
Can you spot the problem?

```
4077 4077  bool HiddenByThisParent(CellHandle parentCell, CellHandle theCell)
4078 4078  {
4079 4079      if ( !BelongsToMe(parentCell, theCell) || !TopOfGroups(parentCell, false))
4080 4080          return false;
4081      -      else if ((parentCell->groupOpenExpr->exprEvaluated != nullptr) && (parentCell->groupOpenExpr->exprEvaluated->isList()))
     4081 +      if ((parentCell->groupOpenExpr->exprEvaluated != nullptr) && (parentCell->groupOpenExpr->exprEvaluated->isList()))
4082 4082      {
4083 4083          int32       cellOffset = theCell->data->index - parentCell->data->index + 1;
4084 4084          Expr        openList = parentCell->groupOpenExpr->exprEvaluated;
4085 4085          int32       length = openList->length();
4086 4086          int32       i;
4087 4087
4088 4088          for(i = 1; i <= length; ++i)
4089 4089              if (cellOffset == openList->partAsInteger(i))
4090 4090                  break;
4091 4091
4092 4092          if (i > length)
4093 4093              return true;
4094 4094      }
4095 4095      else if (theCell->isInlineCell && (GetInlineCellTopLevelParent(theCell) == parentCell))
4096 4096          return false;
4097      -      else if ((parentCell != theCell) && (CellGroupOpen(parentCell) != cellGroupOpen))
     4097 +      if ((parentCell != theCell) && (CellGroupOpen(parentCell) != cellGroupOpen))
4098 4098          return true;
4099 4099
4100 4100      return false;
4101 4101  }
```

What happens if `!(i > length)`?

```
4077 4077  bool HiddenByThisParent(CellHandle parentCell, CellHandle theCell)
4078 4078  {
4079 4079      if ( !BelongsToMe(parentCell, theCell) || !TopOfGroups(parentCell, false))
4080 4080          return false;
4081      -      else if ((parentCell->groupOpenExpr->exprEvaluated != nullptr) && (parentCell->groupOpenExpr->exprEvaluated->isList()))
     4081 +      if ((parentCell->groupOpenExpr->exprEvaluated != nullptr) && (parentCell->groupOpenExpr->exprEvaluated->isList()))
4082 4082      {
4083 4083          int32       cellOffset = theCell->data->index - parentCell->data->index + 1;
4084 4084          Expr        openList = parentCell->groupOpenExpr->exprEvaluated;
4085 4085          int32       length = openList->length();
4086 4086          int32       i;
4087 4087
4088 4088          for(i = 1; i <= length; ++i)
4089 4089              if (cellOffset == openList->partAsInteger(i))
4090 4090                  break;
4091 4091
4092 4092          if (i > length)
4093 4093              return true;
4094 4094      }
4095 4095      else if (theCell->isInlineCell && (GetInlineCellTopLevelParent(theCell) == parentCell))
4096 4096          return false;
4097      -      else if ((parentCell != theCell) && (CellGroupOpen(parentCell) != cellGroupOpen))
     4097 +      if ((parentCell != theCell) && (CellGroupOpen(parentCell) != cellGroupOpen))
4098 4098          return true;
4099 4099
4100 4100      return false;
4101 4101  }
```

else...

else...

It might no longer return `false`

```
4077 4077  bool HiddenByThisParent(CellHandle parentCell, CellHandle theCell)
4078 4078  {
4079 4079      if ( !BelongsToMe(parentCell, theCell) || !TopOfGroups(parentCell, false))
4080 4080          return false;
4081      -      else if ((parentCell->groupOpenExpr->exprEvaluated != nullptr) && (parentCell->groupOpenExpr->exprEvaluated->isList()))
     4081 +      if ((parentCell->groupOpenExpr->exprEvaluated != nullptr) && (parentCell->groupOpenExpr->exprEvaluated->isList()))
4082 4082      {
4083 4083          int32       cellOffset = theCell->data->index - parentCell->data->index + 1;
4084 4084          Expr        openList = parentCell->groupOpenExpr->exprEvaluated;
4085 4085          int32       length = openList->length();
4086 4086          int32       i;
4087 4087
4088 4088          for(i = 1; i <= length; ++i)
4089 4089              if (cellOffset == openList->partAsInteger(i))
4090 4090                  break;
4091 4091
4092 4092          if (i > length)
4093 4093              return true;
4094 4094      }
4095 4095      else if (theCell->isInlineCell && (GetInlineCellTopLevelParent(theCell) == parentCell))
4096 4096          return false;
4097      -      else if ((parentCell != theCell) && (CellGroupOpen(parentCell) != cellGroupOpen))
     4097 +      if ((parentCell != theCell) && (CellGroupOpen(parentCell) != cellGroupOpen))
4098 4098          return true;
4099 4099
4100 4100      return false;
4101 4101  }
```

else...

else...

The .clang-tidy file

```
---
# My working file modulo comments; note the format is YAML

Checks: >
  -clang-analyzer-*,                              # returns too many false positives for now
  modernize-redundant-void-arg,
  modernize-use-bool-literals,
  modernize-use-default-member-init,
  modernize-use-equals-default,
  modernize-use-equals-delete,
  modernize-use-nullptr,
  modernize-use-override,
  readability-delete-null-pointer,
  readability-else-after-return

WarningsAsErrors: ''
# run-clang-tidy.pl insists on putting ^build_dir in front of this.  So,
# for now, we just override it on the command-line.  Sigh.
HeaderFilterRegex: '.*'
AnalyzeTemporaryDtors: false
FormatStyle:     file
CheckOptions:
  - key:        modernize-use-default-member-init.UseAssignment
    value:        '1'
  - key:        modernize-use-override.IgnoreDestructors
    value:        '1'
...
```

Running clang-tidy

Normal clang-tidy invocation

```
clang-tidy -i <compiler flags>
      <list of .cpp files>
```

Better...use run-clang-tidy.py in LLVM's share/clang directory; requires a compile_commands.json directory

```
run-clang-tidy.py -p <compile_commands dir>
      -quiet -fix -header-filter ".*"
      <optional list of .cpp files>
# Omit –fix to not apply rewrites
# Add '-j n' to run n parallel instances
# Maybe redirect stderr to null: 2> /dev/null
```

Using clang-tidy – my workflow

- Use `run-clang-tidy.py` in combination with a cmake-generated `compile_commands.json` file
- Start with no checks and fix errors and warnings
- Add checks with rewrite rules one at a time so you can vet them
- When running a new rule, run it, `git add` the result, then run it again and use `git status` to see if changes are still happening
- If a rule requires several applications, rerun it only on the files that are changing to increase iteration speed
- Break up rewrite by individual rules into separate git commits or pull requests