REFACTORING C++

Using LibTooling

@ArvidGerstmann

REFACTORING?

Code refactoring is the process of restructuring existing computer code — changing the factoring — without changing its external behavior.

— Wikipedia

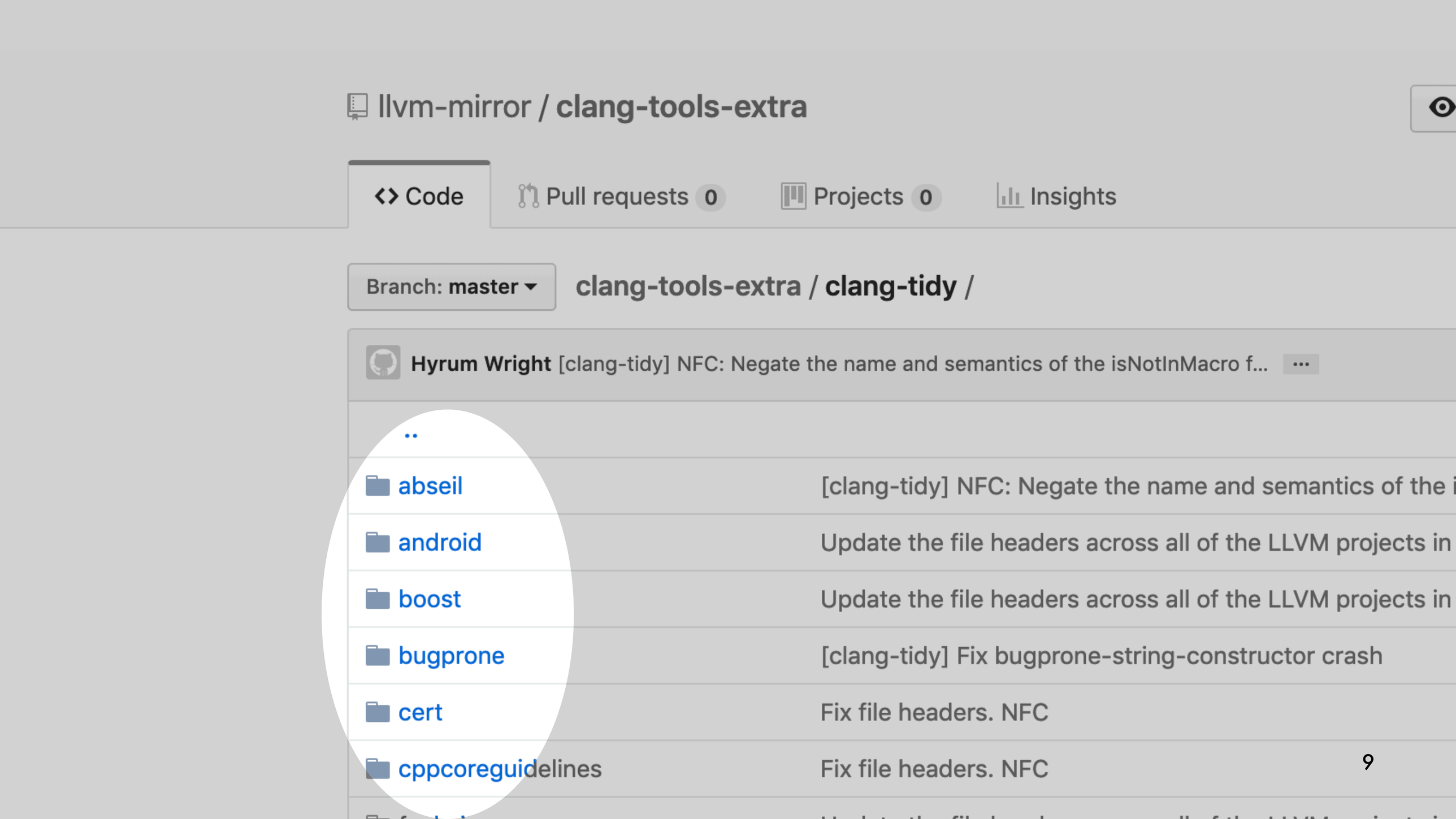
AUTOMATED
REFACTORING

WHY AUTOMATE?

LIVE @ HEAD

Semantic Versioning

NON-ATOMIC API MIGRATION


llvm-mirror / clang-tools-extra
Code
Pull requests 0
Projects 0
Insights
Branch: master
clang-tools-extra / clang-tidy /
Hyrum Wright [clang-tidy] NFC: Negate the name and semantics of the isNotInMacro f...
..
abseil
[clang-tidy] NFC: Negate the name and semantics of the i
android
Update the file headers across all of the LLVM projects in
boost
Update the file headers across all of the LLVM projects in
bugprone
[clang-tidy] Fix bugprone-string-constructor crash
cert
Fix file headers. NFC
cppcoreguidelines
Fix file headers. NFC

EXAMPLE

```
class Button {
    void SetPosition(int x, int y);
    void SetDimensions(int width, int height);
    void SetTitle(char const *title);
};

[[deprecated("Use CreateButtonEx")]]
Button *CreateButton(
    int x,
    int y,
    int width,
    int height,
    char const *title);

Button *CreateButtonEx();
```

```
void Dialog::Create()
{
    /* ... */

    m_Button = CreateButton(32, 32, 150, 40, "Ok");
}
```

```
void Dialog::Create()
{
    /* ... */

    m_Button = CreateButtonEx();
    m_Button->SetPosition(32, 32);
    m_Button->SetDimensions(150, 40);
    m_Button->SetTitle("Ok");
}
```

REFACTORING C++

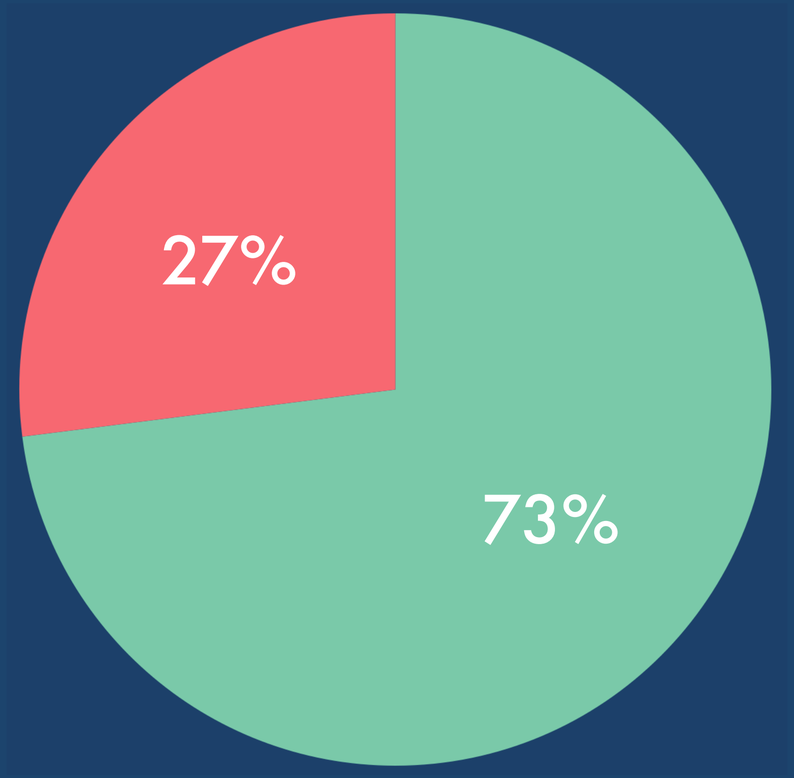
Using LibTooling

@ArvidGerstmann

LibTooling

REQUIREMENTS

Does your codebase compile with clang?



Source: https://twitter.com/ArvidGerstmann/status/1104778287677620231

BUILD SYSTEM

NOT A SILVER BULLET

Hyrum's Law

With a sufficient number of users of an API,
it does not matter what you promise in the contract:
all observable behaviors of your system
will be depended on by somebody.

IN SUMMARY

LLVM SETUP

STANDALONE

VS

CLANG-TIDY

```
using namespace clang::ast_matchers;

namespace clang {
namespace tidy {
namespace custom {

void MigrateCreateButtonCheck::registerMatchers(MatchFinder *Finder) {
  const auto CreateFunction = functionDecl(hasName("CreateButton"));
  const auto AssignmentExpr =
      binaryOperator(
          hasOperatorName("="),
          hasLHS(anyOf(declRefExpr().bind("lhs"), memberExpr().bind("lhs"))),
          hasRHS(callExpr(callee(CreateFunction)).bind("call")))
          .bind("op");

  Finder->addMatcher(AssignmentExpr, this);
}

void MigrateCreateButtonCheck::check(const MatchFinder::MatchResult &Result) {
  const ASTContext &Ctx = *Result.Context;
  const SourceManager &SM = Ctx.getSourceManager();
```

AST MATCHER
BASICS

Node Matchers

Node Matchers

- functionDecl
- callExpr
- compoundStmt
- binaryOperator
- ...

Node Matchers

- functionDecl
- callExpr
- compoundStmt
- binaryOperator
- ...

Narrowing Matchers

Node Matchers

- functionDecl
- callExpr
- compoundStmt
- binaryOperator
- ...

Narrowing Matchers

- allOf/anyOf
- equals
- isConst
- hasName
- ...

Node Matchers

- functionDecl
- callExpr
- compoundStmt
- binaryOperator
- ...

Narrowing Matchers

- allOf/anyOf
- equals
- isConst
- hasName
- ...

Traversal Matchers

Node Matchers

- functionDecl
- callExpr
- compoundStmt
- binaryOperator
- ...

Narrowing Matchers

- allOf/anyOf
- equals
- isConst
- hasName
- ...

Traversal Matchers

- hasDescendant
- hasElse
- hasLHS/hasRHS
- callee
- ...

```
for (int i = 0; i < n; ++i) {
    /* ... */
}
```

```
forStmt()
```

```
forStmt(
  hasLoopInit(
    declStmt(hasSingleDecl(varDecl()))))
```

```
forStmt(
  hasLoopInit(
    declStmt(hasSingleDecl(varDecl(
      hasInitializer(
        integerLiteral(equals(0)))))))
```

```
forStmt(
  hasLoopInit(
    declStmt(hasSingleDecl(varDecl(
      hasInitializer(
        integerLiteral(equals(0)))))))
.bind("forloop");
```

```
forStmt(
  hasLoopInit(
    declStmt(hasSingleDecl(varDecl(
      hasInitializer(
        integerLiteral(equals(0)))))))))
```

```
using namespace clang::ast_matchers;

namespace clang {
namespace tidy {
namespace custom {

void MigrateCreateButtonCheck::registerMatchers(MatchFinder *Finder) {
  const auto CreateFunction = functionDecl(hasName("CreateButton"));
  const auto AssignmentExpr =
      binaryOperator(
          hasOperatorName("="),
          hasLHS(anyOf(declRefExpr().bind("lhs"), memberExpr().bind("lhs"))),
          hasRHS(callExpr(callee(CreateFunction)).bind("call")))
          .bind("op");

  Finder->addMatcher(AssignmentExpr, this);
}

void MigrateCreateButtonCheck::check(const MatchFinder::MatchResult &Result) {
  const ASTContext &Ctx = *Result.Context;
  const SourceManager &SM = Ctx.getSourceManager();
```

```
using namespace clang::ast_matchers;

namespace clang {
namespace tidy {
namespace custom {

void MigrateCreateButtonCheck::registerMatchers(MatchFinder *Finder) {
  const auto CreateFunction = functionDecl(hasName("CreateButton"));
  const auto AssignmentExpr =
      binaryOperator(
          hasOperatorName("="),
          hasLHS(anyOf(declRefExpr().bind("lhs"), memberExpr().bind("lhs"))),
          hasRHS(callExpr(callee(CreateFunction)).bind("call")))
          .bind("op");

  Finder->addMatcher(AssignmentExpr, this);
}

void MigrateCreateButtonCheck::check(const MatchFinder::MatchResult &Result) {
  const ASTContext &Ctx = *Result.Context;
  const SourceManager &SM = Ctx.getSourceManager();
```

```
using namespace clang::ast_matchers;

namespace clang {
namespace tidy {
namespace custom {

void MigrateCreateButtonCheck::registerMatchers(MatchFinder *Finder) {
  const auto CreateFunction = functionDecl(hasName("CreateButton"));
  const auto AssignmentExpr =
      binaryOperator(
          hasOperatorName("="),
          hasLHS(anyOf(declRefExpr().bind("lhs"), memberExpr().bind("lhs"))),
          hasRHS(callExpr(callee(CreateFunction)).bind("call")))
          .bind("op");

  Finder->addMatcher(AssignmentExpr, this);
}

void MigrateCreateButtonCheck::check(const MatchFinder::MatchResult &Result) {
  const ASTContext &Ctx = *Result.Context;
  const SourceManager &SM = Ctx.getSourceManager();
```

```
using namespace clang::ast_matchers;

namespace clang {
namespace tidy {
namespace custom {

void MigrateCreateButtonCheck::registerMatchers(MatchFinder *Finder) {
  const auto CreateFunction = functionDecl(hasName("CreateButton"));
  const auto AssignmentExpr =
      binaryOperator(
          hasOperatorName("="),
          hasLHS(anyOf(declRefExpr().bind("lhs"), memberExpr().bind("lhs"))),
          hasRHS(callExpr(callee(CreateFunction)).bind("call")))
          .bind("op");

  Finder->addMatcher(AssignmentExpr, this);
}

void MigrateCreateButtonCheck::check(const MatchFinder::MatchResult &Result) {
  const ASTContext &Ctx = *Result.Context;
  const SourceManager &SM = Ctx.getSourceManager();
```

```
using namespace clang::ast_matchers;

namespace clang {
namespace tidy {
namespace custom {

void MigrateCreateButtonCheck::registerMatchers(MatchFinder *Finder) {
  const auto CreateFunction = functionDecl(hasName("CreateButton"));
  const auto AssignmentExpr =
      binaryOperator(
          hasOperatorName("="),
          hasLHS(anyOf(declRefExpr().bind("lhs"), memberExpr().bind("lhs"))),
          hasRHS(callExpr(callee(CreateFunction)).bind("call")))
          .bind("op");

  Finder->addMatcher(AssignmentExpr, this);
}

void MigrateCreateButtonCheck::check(const MatchFinder::MatchResult &Result) {
  const ASTContext &Ctx = *Result.Context;
  const SourceManager &SM = Ctx.getSourceManager();
```

```
using namespace clang::ast_matchers;

namespace clang {
namespace tidy {
namespace custom {

void MigrateCreateButtonCheck::registerMatchers(MatchFinder *Finder) {
  const auto CreateFunction = functionDecl(hasName("CreateButton"));
  const auto AssignmentExpr =
      binaryOperator(
          hasOperatorName("="),
          hasLHS(anyOf(declRefExpr().bind("lhs"), memberExpr().bind("lhs"))),
          hasRHS(callExpr(callee(CreateFunction)).bind("call")))
          .bind("op");

  Finder->addMatcher(AssignmentExpr, this);
}

void MigrateCreateButtonCheck::check(const MatchFinder::MatchResult &Result) {
  const ASTContext &Ctx = *Result.Context;
  const SourceManager &SM = Ctx.getSourceManager();
```

```
using namespace clang::ast_matchers;

namespace clang {
namespace tidy {
namespace custom {

void MigrateCreateButtonCheck::registerMatchers(MatchFinder *Finder) {
  const auto CreateFunction = functionDecl(hasName("CreateButton"));
  const auto AssignmentExpr =
      binaryOperator(
          hasOperatorName("="),
          hasLHS(anyOf(declRefExpr().bind("lhs"), memberExpr().bind("lhs"))),
          hasRHS(callExpr(callee(CreateFunction)).bind("call")))
          .bind("op");

  Finder->addMatcher(AssignmentExpr, this);
}

void MigrateCreateButtonCheck::check(const MatchFinder::MatchResult &Result) {
  const ASTContext &Ctx = *Result.Context;
  const SourceManager &SM = Ctx.getSourceManager();
```

```
using namespace clang::ast_matchers;

namespace clang {
namespace tidy {
namespace custom {

void MigrateCreateButtonCheck::registerMatchers(MatchFinder *Finder) {
  const auto CreateFunction = functionDecl(hasName("CreateButton"));
  const auto AssignmentExpr =
      binaryOperator(
          hasOperatorName("="),
          hasLHS(anyOf(declRefExpr().bind("lhs"), memberExpr().bind("lhs"))),
          hasRHS(callExpr(callee(CreateFunction)).bind("call")))
          .bind("op");

  Finder->addMatcher(AssignmentExpr, this);
}

void MigrateCreateButtonCheck::check(const MatchFinder::MatchResult &Result) {
  const ASTContext &Ctx = *Result.Context;
  const SourceManager &SM = Ctx.getSourceManager();
```

```
using namespace clang::ast_matchers;

namespace clang {
namespace tidy {
namespace custom {

void MigrateCreateButtonCheck::registerMatchers(MatchFinder *Finder) {
  const auto CreateFunction = functionDecl(hasName("CreateButton"));
  const auto AssignmentExpr =
      binaryOperator(
          hasOperatorName("="),
          hasLHS(anyOf(declRefExpr().bind("lhs"), memberExpr().bind("lhs"))),
          hasRHS(callExpr(callee(CreateFunction)).bind("call")))
          .bind("op");

  Finder->addMatcher(AssignmentExpr, this);
}

void MigrateCreateButtonCheck::check(const MatchFinder::MatchResult &Result) {
  const ASTContext &Ctx = *Result.Context;
  const SourceManager &SM = Ctx.getSourceManager();
```

```
using namespace clang::ast_matchers;

namespace clang {
namespace tidy {
namespace custom {

void MigrateCreateButtonCheck::registerMatchers(MatchFinder *Finder) {
  const auto CreateFunction = functionDecl(hasName("CreateButton"));
  const auto AssignmentExpr =
      binaryOperator(
          hasOperatorName("="),
          hasLHS(anyOf(declRefExpr().bind("lhs"), memberExpr().bind("lhs"))),
          hasRHS(callExpr(callee(CreateFunction)).bind("call")))
          .bind("op");

  Finder->addMatcher(AssignmentExpr, this);
}

void MigrateCreateButtonCheck::check(const MatchFinder::MatchResult &Result) {
  const ASTContext &Ctx = *Result.Context;
  const SourceManager &SM = Ctx.getSourceManager();
```

```
using namespace clang::ast_matchers;

namespace clang {
namespace tidy {
namespace custom {

void MigrateCreateButtonCheck::registerMatchers(MatchFinder *Finder) {
  const auto CreateFunction = functionDecl(hasName("CreateButton"));
  const auto AssignmentExpr =
      binaryOperator(
          hasOperatorName("="),
          hasLHS(anyOf(declRefExpr().bind("lhs"), memberExpr().bind("lhs"))),
          hasRHS(callExpr(callee(CreateFunction)).bind("call")))
          .bind("op");

  Finder->addMatcher(AssignmentExpr, this);
}

void MigrateCreateButtonCheck::check(const MatchFinder::MatchResult &Result) {
  const ASTContext &Ctx = *Result.Context;
  const SourceManager &SM = Ctx.getSourceManager();
```

```
using namespace clang::ast_matchers;

namespace clang {
namespace tidy {
namespace custom {

void MigrateCreateButtonCheck::registerMatchers(MatchFinder *Finder) {
  const auto CreateFunction = functionDecl(hasName("CreateButton"));
  const auto AssignmentExpr =
      binaryOperator(
          hasOperatorName("="),
          hasLHS(anyOf(declRefExpr().bind("lhs"), memberExpr().bind("lhs"))),
          hasRHS(callExpr(callee(CreateFunction)).bind("call")))
          .bind("op");

  Finder->addMatcher(AssignmentExpr, this);
}

void MigrateCreateButtonCheck::check(const MatchFinder::MatchResult &Result) {
  const ASTContext &Ctx = *Result.Context;
  const SourceManager &SM = Ctx.getSourceManager();
```

```
using namespace clang::ast_matchers;

namespace clang {
namespace tidy {
namespace custom {

void MigrateCreateButtonCheck::registerMatchers(MatchFinder *Finder) {
  const auto CreateFunction = functionDecl(hasName("CreateButton"));
  const auto AssignmentExpr =
      binaryOperator(
          hasOperatorName("="),
          hasLHS(anyOf(declRefExpr().bind("lhs"), memberExpr().bind("lhs"))),
          hasRHS(callExpr(callee(CreateFunction)).bind("call")))
          .bind("op");

  Finder->addMatcher(AssignmentExpr, this);
}

void MigrateCreateButtonCheck::check(const MatchFinder::MatchResult &Result) {
  const ASTContext &Ctx = *Result.Context;
  const SourceManager &SM = Ctx.getSourceManager();
```

```
using namespace clang::ast_matchers;

namespace clang {
namespace tidy {
namespace custom {

void MigrateCreateButtonCheck::registerMatchers(MatchFinder *Finder) {
  const auto CreateFunction = functionDecl(hasName("CreateButton"));
  const auto AssignmentExpr =
      binaryOperator(
          hasOperatorName("="),
          hasLHS(anyOf(declRefExpr().bind("lhs"), memberExpr().bind("lhs"))),
          hasRHS(callExpr(callee(CreateFunction)).bind("call")))
          .bind("op");

  Finder->addMatcher(AssignmentExpr, this);
}

void MigrateCreateButtonCheck::check(const MatchFinder::MatchResult &Result) {
  const ASTContext &Ctx = *Result.Context;
  const SourceManager &SM = Ctx.getSourceManager();
```

```
using namespace clang::ast_matchers;

namespace clang {
namespace tidy {
namespace custom {

void MigrateCreateButtonCheck::registerMatchers(MatchFinder *Finder) {
  const auto CreateFunction = functionDecl(hasName("CreateButton"));
  const auto AssignmentExpr =
      binaryOperator(
          hasOperatorName("="),
          hasLHS(anyOf(declRefExpr().bind("lhs"), memberExpr().bind("lhs"))),
          hasRHS(callExpr(callee(CreateFunction)).bind("call")))
          .bind("op");

  Finder->addMatcher(AssignmentExpr, this);
}

void MigrateCreateButtonCheck::check(const MatchFinder::MatchResult &Result) {
  const ASTContext &Ctx = *Result.Context;
  const SourceManager &SM = Ctx.getSourceManager();
```

```
using namespace clang::ast_matchers;

namespace clang {
namespace tidy {
namespace custom {

void MigrateCreateButtonCheck::registerMatchers(MatchFinder *Finder) {
  const auto CreateFunction = functionDecl(hasName("CreateButton"));
  const auto AssignmentExpr =
      binaryOperator(
          hasOperatorName("="),
          hasLHS(anyOf(declRefExpr().bind("lhs"), memberExpr().bind("lhs"))),
          hasRHS(callExpr(callee(CreateFunction)).bind("call")))
          .bind("op");

  Finder->addMatcher(AssignmentExpr, this);
}

void MigrateCreateButtonCheck::check(const MatchFinder::MatchResult &Result) {
  const ASTContext &Ctx = *Result.Context;
  const SourceManager &SM = Ctx.getSourceManager();
```

AST MATCHER
TIPS & TRICKS

```
$ clang -fsyntax-only -Xclang -ast-dump dialog.cpp $CXXFLAGS
...
`-CXXMethodDecl 0x7fe9ac0720d8 parent 0x7fe9ac071d20 prev 0x7fe9ac071f58 <dialog.cpp:3:1,
line:6:1> line:3:14 Create 'void ()'
  `-CompoundStmt 0x7fe9ac072490 <line:4:1, line:6:1>
    `-BinaryOperator 0x7fe9ac072468 <line:5:5, col:50> 'Button *' lvalue '='
      |-MemberExpr 0x7fe9ac0721d8 <col:5> 'Button *' lvalue ->m_Button 0x7fe9ac072030
      | `-CXXThisExpr 0x7fe9ac0721c0 <col:5> 'Dialog *' this
      `-CallExpr 0x7fe9ac072400 <col:16, col:50> 'Button *'
        |-ImplicitCastExpr 0x7fe9ac0723e8 <col:16> 'Button *(*)(int, int, int, int, const char
*)' <FunctionToPointerDecay>
        | `-DeclRefExpr 0x7fe9ac072398 <col:16> 'Button *(int, int, int, int, const char *)'
lvalue Function 0x7fe9ac071af0 'CreateButton' 'Button *(int, int, int, int, const char *)'
        |-IntegerLiteral 0x7fe9ac072268 <col:29> 'int' 32
        |-IntegerLiteral 0x7fe9ac072288 <col:33> 'int' 32
        |-IntegerLiteral 0x7fe9ac0722a8 <col:37> 'int' 150
        |-IntegerLiteral 0x7fe9ac0722c8 <col:42> 'int' 40
        `-ImplicitCastExpr 0x7fe9ac072450 <col:46> 'const char *' <ArrayToPointerDecay>
          `-StringLiteral 0x7fe9ac072368 <col:46> 'const char [3]' lvalue "Ok"
```

```
$ clang -fsyntax-only -Xclang -ast-dump dialog.cpp $CXXFLAGS
...
`-CXXMethodDecl 0x7fe9ac0720d8 parent 0x7fe9ac071d20 prev 0x7fe9ac071f58 <dialog.cpp:3:1,
line:6:1> line:3:14 Create 'void ()'
  `-CompoundStmt 0x7fe9ac072490 <line:4:1, line:6:1>
    `-BinaryOperator 0x7fe9ac072468 <line:5:5, col:50> 'Button *' lvalue '='
      |-MemberExpr 0x7fe9ac0721d8 <col:5> 'Button *' lvalue ->m_Button 0x7fe9ac072030
      | `-CXXThisExpr 0x7fe9ac0721c0 <col:5> 'Dialog *' this
      `-CallExpr 0x7fe9ac072400 <col:16, col:50> 'Button *'
        |-ImplicitCastExpr 0x7fe9ac0723e8 <col:16> 'Button *(*)(int, int, int, int, const char
*)' <FunctionToPointerDecay>
        | `-DeclRefExpr 0x7fe9ac072398 <col:16> 'Button *(int, int, int, int, const char *)'
lvalue Function 0x7fe9ac071af0 'CreateButton' 'Button *(int, int, int, int, const char *)'
        |-IntegerLiteral 0x7fe9ac072268 <col:29> 'int' 32
        |-IntegerLiteral 0x7fe9ac072288 <col:33> 'int' 32
        |-IntegerLiteral 0x7fe9ac0722a8 <col:37> 'int' 150
        |-IntegerLiteral 0x7fe9ac0722c8 <col:42> 'int' 40
        `-ImplicitCastExpr 0x7fe9ac072450 <col:46> 'const char *' <ArrayToPointerDecay>
          `-StringLiteral 0x7fe9ac072368 <col:46> 'const char [3]' lvalue "Ok"
```

```
$ clang -fsyntax-only -Xclang -ast-dump dialog.cpp $CXXFLAGS
...
`-CXXMethodDecl 0x7fe9ac0720d8 parent 0x7fe9ac071d20 prev 0x7fe9ac071f58 <dialog.cpp:3:1,
line:6:1> line:3:14 Create 'void ()'
  `-CompoundStmt 0x7fe9ac072490 <line:4:1, line:6:1>
    `-BinaryOperator 0x7fe9ac072468 <line:5:5, col:50> 'Button *' lvalue '='
      |-MemberExpr 0x7fe9ac0721d8 <col:5> 'Button *' lvalue ->m_Button 0x7fe9ac072030
      | `-CXXThisExpr 0x7fe9ac0721c0 <col:5> 'Dialog *' this
      `-CallExpr 0x7fe9ac072400 <col:16, col:50> 'Button *'
        |-ImplicitCastExpr 0x7fe9ac0723e8 <col:16> 'Button *(*)(int, int, int, int, const char
*)' <FunctionToPointerDecay>
        | `-DeclRefExpr 0x7fe9ac072398 <col:16> 'Button *(int, int, int, int, const char *)'
lvalue Function 0x7fe9ac071af0 'CreateButton' 'Button *(int, int, int, int, const char *)'
        |-IntegerLiteral 0x7fe9ac072268 <col:29> 'int' 32
        |-IntegerLiteral 0x7fe9ac072288 <col:33> 'int' 32
        |-IntegerLiteral 0x7fe9ac0722a8 <col:37> 'int' 150
        |-IntegerLiteral 0x7fe9ac0722c8 <col:42> 'int' 40
        `-ImplicitCastExpr 0x7fe9ac072450 <col:46> 'const char *' <ArrayToPointerDecay>
          `-StringLiteral 0x7fe9ac072368 <col:46> 'const char [3]' lvalue "Ok"
```

```
$ clang -fsyntax-only -Xclang -ast-dump dialog.cpp $CXXFLAGS
...
`-CXXMethodDecl 0x7fe9ac0720d8 parent 0x7fe9ac071d20 prev 0x7fe9ac071f58 <dialog.cpp:3:1,
line:6:1> line:3:14 Create 'void ()'
  `-CompoundStmt 0x7fe9ac072490 <line:4:1, line:6:1>
    `-BinaryOperator 0x7fe9ac072468 <line:5:5, col:50> 'Button *' lvalue '='
      |-MemberExpr 0x7fe9ac0721d8 <col:5> 'Button *' lvalue ->m_Button 0x7fe9ac072030
      | `-CXXThisExpr 0x7fe9ac0721c0 <col:5> 'Dialog *' this
      `-CallExpr 0x7fe9ac072400 <col:16, col:50> 'Button *'
        |-ImplicitCastExpr 0x7fe9ac0723e8 <col:16> 'Button *(*)(int, int, int, int, const char
*)' <FunctionToPointerDecay>
        | `-DeclRefExpr 0x7fe9ac072398 <col:16> 'Button *(int, int, int, int, const char *)'
lvalue Function 0x7fe9ac071af0 'CreateButton' 'Button *(int, int, int, int, const char *)'
        |-IntegerLiteral 0x7fe9ac072268 <col:29> 'int' 32
        |-IntegerLiteral 0x7fe9ac072288 <col:33> 'int' 32
        |-IntegerLiteral 0x7fe9ac0722a8 <col:37> 'int' 150
        |-IntegerLiteral 0x7fe9ac0722c8 <col:42> 'int' 40
        `-ImplicitCastExpr 0x7fe9ac072450 <col:46> 'const char *' <ArrayToPointerDecay>
          `-StringLiteral 0x7fe9ac072368 <col:46> 'const char [3]' lvalue "Ok"
```

```
$ clang -fsyntax-only -Xclang -ast-dump dialog.cpp $CXXFLAGS
...
`-CXXMethodDecl 0x7fe9ac0720d8 parent 0x7fe9ac071d20 prev 0x7fe9ac071f58 <dialog.cpp:3:1,
line:6:1> line:3:14 Create 'void ()'
  `-CompoundStmt 0x7fe9ac072490 <line:4:1, line:6:1>
    `-BinaryOperator 0x7fe9ac072468 <line:5:5, col:50> 'Button *' lvalue '='
      |-MemberExpr 0x7fe9ac0721d8 <col:5> 'Button *' lvalue ->m_Button 0x7fe9ac072030
      | `-CXXThisExpr 0x7fe9ac0721c0 <col:5> 'Dialog *' this
      `-CallExpr 0x7fe9ac072400 <col:16, col:50> 'Button *'
        |-ImplicitCastExpr 0x7fe9ac0723e8 <col:16> 'Button *(*)(int, int, int, int, const char
*)' <FunctionToPointerDecay>
        | `-DeclRefExpr 0x7fe9ac072398 <col:16> 'Button *(int, int, int, int, const char *)'
lvalue Function 0x7fe9ac071af0 'CreateButton' 'Button *(int, int, int, int, const char *)'
        |-IntegerLiteral 0x7fe9ac072268 <col:29> 'int' 32
        |-IntegerLiteral 0x7fe9ac072288 <col:33> 'int' 32
        |-IntegerLiteral 0x7fe9ac0722a8 <col:37> 'int' 150
        |-IntegerLiteral 0x7fe9ac0722c8 <col:42> 'int' 40
        `-ImplicitCastExpr 0x7fe9ac072450 <col:46> 'const char *' <ArrayToPointerDecay>
          `-StringLiteral 0x7fe9ac072368 <col:46> 'const char [3]' lvalue "Ok"
```

```
$ clang -fsyntax-only -Xclang -ast-dump dialog.cpp $CXXFLAGS
...
`-CXXMethodDecl 0x7fe9ac0720d8 parent 0x7fe9ac071d20 prev 0x7fe9ac071f58 <dialog.cpp:3:1,
line:6:1> line:3:14 Create 'void ()'
  `-CompoundStmt 0x7fe9ac072490 <line:4:1, line:6:1>
    `-BinaryOperator 0x7fe9ac072468 <line:5:5, col:50> 'Button *' lvalue '='
      |-MemberExpr 0x7fe9ac0721d8 <col:5> 'Button *' lvalue ->m_Button 0x7fe9ac072030
      | `-CXXThisExpr 0x7fe9ac0721c0 <col:5> 'Dialog *' this
      `-CallExpr 0x7fe9ac072400 <col:16, col:50> 'Button *'
        |-ImplicitCastExpr 0x7fe9ac0723e8 <col:16> 'Button *(*)(int, int, int, int, const char
*)' <FunctionToPointerDecay>
        | `-DeclRefExpr 0x7fe9ac072398 <col:16> 'Button *(int, int, int, int, const char *)'
lvalue Function 0x7fe9ac071af0 'CreateButton' 'Button *(int, int, int, int, const char *)'
        |-IntegerLiteral 0x7fe9ac072268 <col:29> 'int' 32
        |-IntegerLiteral 0x7fe9ac072288 <col:33> 'int' 32
        |-IntegerLiteral 0x7fe9ac0722a8 <col:37> 'int' 150
        |-IntegerLiteral 0x7fe9ac0722c8 <col:42> 'int' 40
        `-ImplicitCastExpr 0x7fe9ac072450 <col:46> 'const char *' <ArrayToPointerDecay>
          `-StringLiteral 0x7fe9ac072368 <col:46> 'const char [3]' lvalue "Ok"
```

```
}

void MigrateCreatebuttonCheck::check(const MatchFinder::MatchResult &Result) {
  const ASTContext &Ctx = *Result.Context;
  const SourceManager &SM = Ctx.getSourceManager();
  const LangOptions &Lang = Ctx.getLangOpts();

  const auto *LhsRef = Result.Nodes.getNodeAs<DeclRefExpr>("lhs");
  const auto *LhsMember = Result.Nodes.getNodeAs<MemberExpr>("lhs");
  const auto *Op = Result.Nodes.getNodeAs<BinaryOperator>("op");
  const auto *Call = Result.Nodes.getNodeAs<CallExpr>("call");

  assert((LhsRef != nullptr || LhsMember != nullptr));
  assert(Op != nullptr);
  assert(Call != nullptr);
  const auto *Lhs = LhsRef ? static_cast<const Expr *>(LhsRef)
                           : static_cast<const Expr *>(LhsMember);

  auto GetArg = [&](unsigned i) {
    return Lexer::getSourceText(
        CharSourceRange::getTokenRange(Call->getArg(i)->getSourceRange()), SM,
```

```
}

void MigrateCreatebuttonCheck::check(const MatchFinder::MatchResult &Result) {
  const ASTContext &Ctx = *Result.Context;
  const SourceManager &SM = Ctx.getSourceManager();
  const LangOptions &Lang = Ctx.getLangOpts();
  if (!Lang.CPlusPlus)
    return;

  const auto *LhsRef = Result.Nodes.getNodeAs<DeclRefExpr>("lhs");
  const auto *LhsMember = Result.Nodes.getNodeAs<MemberExpr>("lhs");
  const auto *Op = Result.Nodes.getNodeAs<BinaryOperator>("op");
  const auto *Call = Result.Nodes.getNodeAs<CallExpr>("call");

  assert((LhsRef != nullptr || LhsMember != nullptr));
  assert(Op != nullptr);
  assert(Call != nullptr);
  const auto *Lhs = LhsRef ? static_cast<const Expr *>(LhsRef)
                           : static_cast<const Expr *>(LhsMember);

  auto GetArg = [&](unsigned i) {
```

```
}

void MigrateCreatebuttonCheck::check(const MatchFinder::MatchResult &Result) {
  const ASTContext &Ctx = *Result.Context;
  const SourceManager &SM = Ctx.getSourceManager();
  const LangOptions &Lang = Ctx.getLangOpts();
  if (!Lang.CPlusPlus)
    return;

  const auto *LhsRef = Result.Nodes.getNodeAs<DeclRefExpr>("lhs");
  const auto *LhsMember = Result.Nodes.getNodeAs<MemberExpr>("lhs");
  const auto *Op = Result.Nodes.getNodeAs<BinaryOperator>("op");
  const auto *Call = Result.Nodes.getNodeAs<CallExpr>("call");

  assert((LhsRef != nullptr || LhsMember != nullptr));
  assert(Op != nullptr);
  assert(Call != nullptr);
  const auto *Lhs = LhsRef ? static_cast<const Expr *>(LhsRef)
                           : static_cast<const Expr *>(LhsMember);

  auto GetArg = [&](unsigned i) {
```

```
}

void MigrateCreatebuttonCheck::check(const MatchFinder::MatchResult &Result) {
  const ASTContext &Ctx = *Result.Context;
  const SourceManager &SM = Ctx.getSourceManager();
  const LangOptions &Lang = Ctx.getLangOpts();
  if (!Lang.CPlusPlus)
    return;

  const auto *LhsRef = Result.Nodes.getNodeAs<DeclRefExpr>("lhs");
  const auto *LhsMember = Result.Nodes.getNodeAs<MemberExpr>("lhs");
  const auto *Op = Result.Nodes.getNodeAs<BinaryOperator>("op");
  const auto *Call = Result.Nodes.getNodeAs<CallExpr>("call");

  assert((LhsRef != nullptr || LhsMember != nullptr));
  assert(Op != nullptr);
  assert(Call != nullptr);
  const auto *Lhs = LhsRef ? static_cast<const Expr *>(LhsRef)
                           : static_cast<const Expr *>(LhsMember);

  auto GetArg = [&](unsigned i) {
```

```
}

void MigrateCreatebuttonCheck::check(const MatchFinder::MatchResult &Result) {
  const ASTContext &Ctx = *Result.Context;
  const SourceManager &SM = Ctx.getSourceManager();
  const LangOptions &Lang = Ctx.getLangOpts();
  if (!Lang.CPlusPlus)
    return;

  const auto *LhsRef = Result.Nodes.getNodeAs<DeclRefExpr>("lhs");
  const auto *LhsMember = Result.Nodes.getNodeAs<MemberExpr>("lhs");
  const auto *Op = Result.Nodes.getNodeAs<BinaryOperator>("op");
  const auto *Call = Result.Nodes.getNodeAs<CallExpr>("call");

  assert((LhsRef != nullptr || LhsMember != nullptr));
  assert(Op != nullptr);
  assert(Call != nullptr);
  const auto *Lhs = LhsRef ? static_cast<const Expr *>(LhsRef)
                           : static_cast<const Expr *>(LhsMember);

  auto GetArg = [&](unsigned i) {
```

```
}

void MigrateCreatebuttonCheck::check(const MatchFinder::MatchResult &Result) {
  const ASTContext &Ctx = *Result.Context;
  const SourceManager &SM = Ctx.getSourceManager();
  const LangOptions &Lang = Ctx.getLangOpts();
  if (!Lang.CPlusPlus)
    return;

  const auto *LhsRef = Result.Nodes.getNodeAs<DeclRefExpr>("lhs");
  const auto *LhsMember = Result.Nodes.getNodeAs<MemberExpr>("lhs");
  const auto *Op = Result.Nodes.getNodeAs<BinaryOperator>("op");
  const auto *Call = Result.Nodes.getNodeAs<CallExpr>("call");

  assert((LhsRef != nullptr || LhsMember != nullptr));
  assert(Op != nullptr);
  assert(Call != nullptr);
  const auto *Lhs = LhsRef ? static_cast<const Expr *>(LhsRef)
                           : static_cast<const Expr *>(LhsMember);

  auto GetArg = [&](unsigned i) {
```

```
}

void MigrateCreatebuttonCheck::check(const MatchFinder::MatchResult &Result) {
  const ASTContext &Ctx = *Result.Context;
  const SourceManager &SM = Ctx.getSourceManager();
  const LangOptions &Lang = Ctx.getLangOpts();
  if (!Lang.CPlusPlus)
    return;

  const auto *LhsRef = Result.Nodes.getNodeAs<DeclRefExpr>("lhs");
  const auto *LhsMember = Result.Nodes.getNodeAs<MemberExpr>("lhs");
  const auto *Op = Result.Nodes.getNodeAs<BinaryOperator>("op");
  const auto *Call = Result.Nodes.getNodeAs<CallExpr>("call");

  assert((LhsRef != nullptr || LhsMember != nullptr));
  assert(Op != nullptr);
  assert(Call != nullptr);
  const auto *Lhs = LhsRef ? static_cast<const Expr *>(LhsRef)
                           : static_cast<const Expr *>(LhsMember);

  auto GetArg = [&](unsigned i) {
```

```
assert(Op != nullptr);
assert(Call != nullptr);
const auto *Lhs = LhsRef ? static_cast<const Expr *>(LhsRef)
                         : static_cast<const Expr *>(LhsMember);

auto GetArg = [&](unsigned i) {
  return Lexer::getSourceText(
      CharSourceRange::getTokenRange(Call->getArg(i)->getSourceRange()), SM,
      Lang);
};

const unsigned IndentSpaces = SM.getExpansionColumnNumber(Lhs->getBeginLoc()) - 1;

SmallString<32> IndentStr;
IndentStr.resize(IndentSpaces);
std::fill_n(IndentStr.begin(), IndentSpaces, ' ');

const StringRef LhsStr = Lexer::getSourceText(
    CharSourceRange::getTokenRange(Lhs->getBeginLoc(), Lhs->getEndLoc()), SM,
    Lang);
```

```
assert(Op != nullptr);
assert(Call != nullptr);
const auto *Lhs = LhsRef ? static_cast<const Expr *>(LhsRef)
                         : static_cast<const Expr *>(LhsMember);

auto GetArg = [&](unsigned i) {
  return Lexer::getSourceText(
      CharSourceRange::getTokenRange(Call->getArg(i)->getSourceRange()), SM,
      Lang);
};

const unsigned IndentSpaces = SM.getExpansionColumnNumber(Lhs->getBeginLoc()) - 1;

SmallString<32> IndentStr;
IndentStr.resize(IndentSpaces);
std::fill_n(IndentStr.begin(), IndentSpaces, ' ');

const StringRef LhsStr = Lexer::getSourceText(
    CharSourceRange::getTokenRange(Lhs->getBeginLoc(), Lhs->getEndLoc()), SM,
    Lang);
```

```
  const StringRef LhsStr = Lexer::getSourceText(
      CharSourceRange::getTokenRange(Lhs->getBeginLoc(), Lhs->getEndLoc()), SM,
      Lang);

  const auto SetPositionStr =
      ("\n" + IndentStr + LhsStr + "->SetPosition(" + GetArg(0) + ", " + GetArg(1) +
");").str();
  const auto SetDimensionsStr =
      ("\n" + IndentStr + LhsStr + "->SetDimensions(" + GetArg(2) + ", " + GetArg(3)
+ ");").str();
  const auto SetTitleStr =
      ("\n" + IndentStr + LhsStr + "->SetTitle(" + GetArg(4) + ");").str();

  const auto TokAfterCall = Lexer::findNextToken(Call->getEndLoc(), SM, Lang);
  assert(bool(TokAfterCall));

  auto Diagnostic =
      diag(Call->getBeginLoc(), "Found old CreateButton call. Fix available.");
```

```
  const StringRef LhsStr = Lexer::getSourceText(
      CharSourceRange::getTokenRange(Lhs->getBeginLoc(), Lhs->getEndLoc()), SM,
      Lang);

  const auto SetPositionStr =
      ("\n" + IndentStr + LhsStr + "->SetPosition(" + GetArg(0) + ", " + GetArg(1) +
");").str();
  const auto SetDimensionsStr =
      ("\n" + IndentStr + LhsStr + "->SetDimensions(" + GetArg(2) + ", " + GetArg(3)
+ ");").str();
  const auto SetTitleStr =
      ("\n" + IndentStr + LhsStr + "->SetTitle(" + GetArg(4) + ");").str();

  const auto TokAfterCall = Lexer::findNextToken(Call->getEndLoc(), SM, Lang);
  assert(bool(TokAfterCall));

  auto Diagnostic =
      diag(Call->getBeginLoc(), "Found old CreateButton call. Fix available.");
```

```
  const StringRef LhsStr = Lexer::getSourceText(
      CharSourceRange::getTokenRange(Lhs->getBeginLoc(), Lhs->getEndLoc()), SM,
      Lang);

  const auto SetPositionStr =
      ("\n" + IndentStr + LhsStr + "->SetPosition(" + GetArg(0) + ", " + GetArg(1) +
");").str();
  const auto SetDimensionsStr =
      ("\n" + IndentStr + LhsStr + "->SetDimensions(" + GetArg(2) + ", " + GetArg(3)
+ ");").str();
  const auto SetTitleStr =
      ("\n" + IndentStr + LhsStr + "->SetTitle(" + GetArg(4) + ");").str();

  const auto TokAfterCall = Lexer::findNextToken(Call->getEndLoc(), SM, Lang);
  assert(bool(TokAfterCall));

  auto Diagnostic =
      diag(Call->getBeginLoc(), "Found old CreateButton call. Fix available.");
```

```
  const auto TokAfterCall = Lexer::findNextToken(Call->getEndLoc(), SM, Lang);
  assert(bool(TokAfterCall));

  auto Diagnostic =
      diag(Call->getBeginLoc(), "Found old CreateButton call. Fix available.");

  Diagnostic
      << FixItHint::CreateInsertion(TokAfterCall->getEndLoc(), SetPositionStr)

      << FixItHint::CreateInsertion(TokAfterCall->getEndLoc(), SetDimensionsStr)

      << FixItHint::CreateInsertion(TokAfterCall->getEndLoc(), SetTitleStr)

      << FixItHint::CreateReplacement(
             CharSourceRange::getTokenRange(Call->getBeginLoc(),
                                            Call->getEndLoc()),
             "CreateButtonEx()");
}
```

```
const auto TokAfterCall = Lexer::findNextToken(Call->getEndLoc(), SM, Lang);
assert(bool(TokAfterCall));

auto Diagnostic =
    diag(Call->getBeginLoc(), "Found old CreateButton call. Fix available.");

Diagnostic
    << FixItHint::CreateInsertion(TokAfterCall->getEndLoc(), SetPositionStr)

    << FixItHint::CreateInsertion(TokAfterCall->getEndLoc(), SetDimensionsStr)

    << FixItHint::CreateInsertion(TokAfterCall->getEndLoc(), SetTitleStr)

    << FixItHint::CreateReplacement(
           CharSourceRange::getTokenRange(Call->getBeginLoc(),
                                          Call->getEndLoc()),
           "CreateButtonEx()");
}
```

```
const auto TokAfterCall = Lexer::findNextToken(Call->getEndLoc(), SM, Lang);
assert(bool(TokAfterCall));

auto Diagnostic =
    diag(Call->getBeginLoc(), "Found old CreateButton call. Fix available.");

Diagnostic
    << FixItHint::CreateInsertion(TokAfterCall->getEndLoc(), SetPositionStr)

    << FixItHint::CreateInsertion(TokAfterCall->getEndLoc(), SetDimensionsStr)

    << FixItHint::CreateInsertion(TokAfterCall->getEndLoc(), SetTitleStr)

    << FixItHint::CreateReplacement(
           CharSourceRange::getTokenRange(Call->getBeginLoc(),
                                          Call->getEndLoc()),
           "CreateButtonEx()");
}
```

```
$ ./clang-tidy -p . button.cpp dialog.cpp test.cpp -checks="-*,custom-*" -fix
```

```
$ ./clang-tidy -p . button.cpp dialog.cpp test.cpp -checks="-*,custom-*" -fix

2 warnings generated.
2 warnings generated.
/Users/leandros/llvm/tools/clang/tools/extra/migrate-v1/test/dialog.cpp:6:16:
warning: Found old CreateButton call. Fix available. [custom-migrate-create-
button]
    m_Button = CreateButton(32, 32, 150, 40, "Ok");
               ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
/Users/leandros/llvm/tools/clang/tools/extra/migrate-v1/test/dialog.cpp:6:16:
note: FIX-IT applied suggested code changes
/Users/leandros/llvm/tools/clang/tools/extra/migrate-v1/test/dialog.cpp:6:52:
note: FIX-IT applied suggested code changes
    m_Button = CreateButton(32, 32, 150, 40, "Ok");
                                                   ^
clang-tidy applied 2 of 2 suggested fixes.
Suppressed 1 warnings (1 with check filters).
```

Thank You!

 @ArvidGerstmann

 /arvidgerstmann

 arvid.io

EAST CONST

BONUS

LLVM Setup

1. Clone LLVM
   ```
   $ git clone https://git.llvm.org/git/llvm.git/ llvm
   ```
2. Clone clang into '$LLVM/tools/'
   ```
   $ cd llvm/tools
   $ git clone https://git.llvm.org/git/clang.git/
   ```
3. Clone clang-extra-tools into '$LLVM/tools/clang/tools/extra'
   ```
   $ cd clang/tools
   $ git clone https://git.llvm.org/git/clang-tools-extra.git/ extra
   ```
4. Add your project project
   ```
   $ mkdir yourproject
   $ touch yourproject/CMakeLists.txt
   $ echo "add_subdirectory(yourproject)" >> extra/CMakeLists.txt
   ```
5. Generate the project using CMake
   ```
   $ cmake -G"Ninja"
   ```

Links

- https://web.archive.org/web/20190310193023/https://eli.thegreenplace.net/2014/05/21/compilation-databases-for-clang-based-tools

- https://web.archive.org/web/20190310192856/https://sarcasm.github.io/notes/dev/compilation-database.html