Don’t Fear Custom Clang Tools

Summary

- What is Clang Tooling
- Killer App 1: Large Scale Refactoring
- Killer App 2: Dead Code Elimination

What are Clang Tools

- Standalone tools for C++ developers
  - Automatic Formatting
  - Refactoring
  - Checking
  - Requirement: Code compiles with Clang

Just compiling with Clang found bugs in my code

It Compiles with Clang! What do I win

- Clang Tidy
  - “An excellent C++ Linter”

It Compiles with Clang! What do I win

```
struct Base {
   virtual void reimplementMe(int a) {}
};
struct Derived : public Base  {
   virtual void reimplementMe(int a) {}
};
```

clang-tidy -checks='modernize-use-override' demo.cpp

It Compiles with Clang! What do I win

```
struct Base {
   virtual void reimplementMe(int a) {}
};
struct Derived : public Base  {
   virtual void reimplementMe(int a) {}
};
```

```
demo.cpp:5:17: warning: prefer using 'override' or (rarely) 'final'
instead of 'virtual' [modernize-use-override]
   virtual void reimplementMe(int a) {}
   ~~~~~~~~     ^
                                override
```

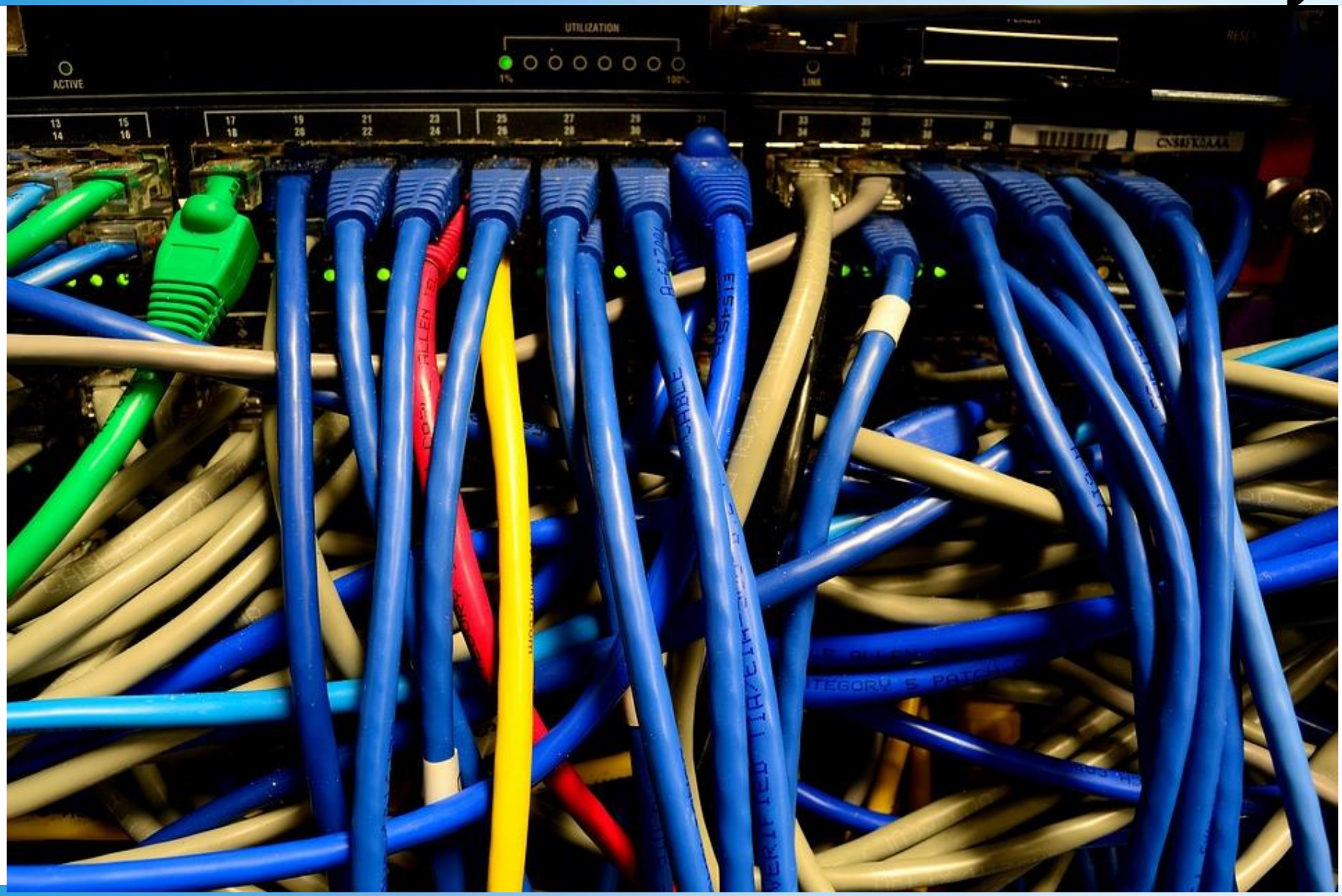
But Wait, There's More. -fix

```
struct Base {
   virtual void reimplementMe(int a) {}
};
struct Derived : public Base  {
   virtual void reimplementMe(int a) override {}
};
```

clang-tidy -checks='modernize-use-override' -fix demo.cpp -- -std=c+11

Maybe it just works and it’s awesome

Sometimes the real world is messy



Maybe you don’t care about overrides

Killer App: Large Scale Refactoring

Killer App: Large Scale Refactoring

```
void Car::add( Person *p )
{
  ...
}

void pack_for_vacation()
{
        car->add( driver );
        car->add( kid );
        car->add( dog );
        trailer->add( grandma );
}
```

Killer App: Large Scale Refactoring

```
void Car::push_back( std::unique_ptr<Person> p )
{
  ...
}

void pack_for_vacation()
{
        car->add( driver );
        car->add( kid );
        car->add( dog );
        trailer->add( grandma );
}
```

Killer App: Large Scale Refactoring

```
void Car::push_back( std::unique_ptr<Person> p )
{
  ...
}

void pack_for_vacation()
{
        car->push_back( std::unique_ptr<Person>(driver) );
        car->push_back( std::unique_ptr<Person>(kid) );
        car->add( dog );
        trailer->add( grandma );
}
```

Killer App: Dead Code Elimination

Dead Code Elimination

```
void Car::add( Person *p )
{
  if ( type() == RENAULT && color() == BLACK )
  {
    do_case_1();
  }
  else {
    do_case_2();
  }
  ...
}
```

Renaults: Not Supported Anymore

Dead Code Elimination

```
void Car::add( Person *p )
{
  if ( false && color() == BLACK )
  {
    do_case_1();
  }
  else {
    do_case_2();
  }
  ...
}
```

type() == RENAULT is always false
We do not support Renaults anymore

Dead Code Elimination

```
void Car::add( Person *p )
{
  if ( false )
  {
    do_case_1();
  }
  else {
    do_case_2();
  }
  ...
}
```

false && expression is always false

Dead Code Elimination

```
void Car::add( Person *p )
{
  {
    do_case_2();
  }
  ...
}
```

The if branch is trivially dead code

My Experience with Clang Tools

- Tool #1 — 4 days dev time
- Tool #2 — 4 hours dev time
- Tool #3 — 2 hours dev time