How to run include-what-you-use (IWYU) on the Swift project

include-what-you-use (IWYU) is a Clang-based tool that analyzes `#include`s in a file and makes suggestions to add or remove `#include`s based on usage in the code. This has two key benefits:

- Removing unused `#include` statements reduces work for the compiler.
- Adding `#include` statements for usage avoids a refactoring in a header file from breaking downstream implementation files due to accidental transitive usage.

Running IWYU is a bit tricky, so this how-to guide provides the steps for how to get it up and running on the Swift project for macOS. If you get IWYU working on a different platform and some steps need to be changed, please update this document with platform-specific steps.

Pre-requisites

- A built Swift project with exported compilation commands. By default, compilation commands are generated in the file `build/[BuildSystem]-[BuildVariant]/swift-[target]/compile_commands.json`. Check that this file is present before proceeding.
  - If this file is missing, try building with `CMAKE_EXPORT_COMPILATION_COMMANDS=ON`. If you use `build-script` to manage your builds, you can do this with

    ```
    swift/utils/build-script <other options> \
      --extra-cmake-options='-DCMAKE_EXPORT_COMPILATION_COMMANDS=ON'
    ```

- Install `jq`. It's not strictly necessary, but we will use it for some JSON munging.

Cloning and branch checkout

The directory structure we will be using is

```
swift-project/
  |--- build/
  |    |--- [BuildSystem]-[BuildVariant]/
  |    |    |--- swift-[target]/
  |    |    |     |--- compile_commands.json
  |    |    |     `--- ...
  |    |    |--- iwyu-[target]/
  |    |    `--- ...
  |    `--- ...
  |--- swift/
  |--- iwyu/
  |    |--- src/
  |    |--- logs/
```

```
|    `--- scripts/
`--- ...
```

As a running example, the description below uses `[BuildSystem] = Ninja`, `[BuildVariant] = ReleaseAssert` and `[target] = macosx-x86_64`.

Start with `swift-project` as the working directory.

1. Check out IWYU.

   ```
   mkdir -p iwyu/src
   git clone https://github.com/include-what-you-use/include-what-you-use.git
   iwyu/src
   ```

2. Find out the version of the `clang` built recently.

   ```
   build/Ninja-ReleaseAssert/llvm-macosx-x86_64/bin/clang --version
   ```

   This should say something like `clang version 10.0.0` or similar.

3. Based on the `clang` version, make sure you check out the correct branch.

   ```
   git -C iwyu/src checkout clang_10
   ```

Building IWYU

1. Configure IWYU with CMake.

   ```
   cmake -G Ninja \
     -DCMAKE_PREFIX_PATH=build/Ninja-ReleaseAssert/llvm-macosx-x86_64 \
     -DCMAKE_CXX_STANDARD=14 \
     -B build/Ninja-ReleaseAssert/iwyu-macosx-x86_64 \
     iwyu/src
   ```

2. Build IWYU

   ```
   cmake --build build/Ninja-ReleaseAssert/iwyu-macosx-x86_64
   ```

3. Create an extra symlink so IWYU can find necessary Clang headers:

   ```
   ln -sF build/Ninja-ReleaseAssert/llvm-macosx-x86_64/lib build/Ninja-
   ReleaseAssert/iwyu-macosx-x86_64/lib
   ```

4. Spot check IWYU for a basic C example.

   ```
   echo '#include <stdint.h>' > tmp.c
   ./bin/include-what-you-use tmp.c -E -o /dev/null \
     -I "$(xcrun --show-sdk-path)/usr/include"
   rm tmp.c
   ```

   You should see output like:

   ```
   tmp.c should add these lines:

   ```

```
tmp.c should remove these lines:
- #include <stdint.h>  // lines 1-1

The full include-list for tmp.c:
---
```

5. Spot check IWYU for a basic C++ example. Notice the extra C++-specific include path.

```
echo '#include <string>\n#include <cmath>' > tmp.cpp
./bin/include-what-you-use tmp.cpp -E -o /dev/null \
  -I "$(clang++ -print-resource-dir)/../../../include/c++/v1" \
  -I "$(xcrun --show-sdk-path)/usr/include"
rm tmp.cpp
```

You should see output like:

```
tmp.cpp should add these lines:

tmp.cpp should remove these lines:
- #include <cmath>  // lines 2-2
- #include <string>  // lines 1-1

The full include-list for tmp.cpp:
---
```

Running IWYU

1. Create a directory, say `iwyu/scripts`, and copy the following script there.

```
#!/usr/bin/env bash

# iwyu_run.sh
set -eu

SWIFT_PROJECT_DIR="$HOME/swift-project"
SWIFT_BUILD_DIR="$SWIFT_PROJECT_DIR/build/Ninja-ReleaseAssert/swift-macosx-
x86_64"

pushd "$SWIFT_BUILD_DIR"

if [ -f original_compile_commands.json ]; then
    mv original_compile_commands.json compile_commands.json
fi

# HACK: The additional include path needs to be added before other include
# paths, it doesn't seem to work if we add it at the end.
# It is ok to rely on the presence of `-D__STDC_LIMIT_MACROS` flag, since
# it is added by the LLVM CMake configuration for all compilation commands.
( EXTRA_CXX_INCLUDE_DIR="$(clang++ -print-resource-
dir)/../../../include/c++/v1";
  cat compile_commands.json \
```

```
 | jq '[.[] | select(.file | test("\\.mm" | "\\.m") | not) | {directory:
.directory, command: (.command + " -Wno-everything -ferror-limit=1"), file:
.file}]' \
 | sed -e "s|-D__STDC_LIMIT_MACROS |-D__STDC_LIMIT_MACROS -I
$EXTRA_CXX_INCLUDE_DIR |" \
) > filtered_compile_commands.json

mv compile_commands.json original_compile_commands.json
mv filtered_compile_commands.json compile_commands.json

mkdir -p "$SWIFT_PROJECT_DIR/iwyu/logs"

( PATH="$SWIFT_PROJECT_DIR/iwyu/build/bin:$PATH"; \
  "$SWIFT_PROJECT_DIR/iwyu/include-what-you-use/iwyu_tool.py" -p
"$SWIFT_BUILD_DIR"
) | tee "$SWIFT_PROJECT_DIR/iwyu/logs/suggestions.log"

popd
```

We filter out Objective-C files because IWYU does not support Objective-C. If that step is missed, you might hit errors like:

```
iwyu.cc:2097: Assertion failed: TODO(csilvers): for objc and clang lang
extensions
```

2. Update the `SWIFT_PROJECT_DIR` and `SWIFT_BUILD_DIR` variables based on your project and build directories.

3. Run the script.

```
chmod +x iwyu/scripts/iwyu_run.sh
iwyu/scripts/iwyu_run.sh
```

This will generate a log file under `iwyu/logs/suggestions.log`. Note that IWYU might take several hours to run, depending on your system.

NOTE: The IWYU README suggests several different ways of running IWYU on a CMake project, including using the `CMAKE_CXX_INCLUDE_WHAT_YOU_USE` and `CMAKE_C_INCLUDE_WHAT_YOU_USE` variables. At the time of writing, those did not reliably work on macOS; suggestions were generated only for specific subprojects (e.g. the stdlib) and not others (e.g. the compiler). Using CMake variables also requires reconfiguring and rebuilding, which makes debugging much more time-consuming.

Debugging

While the above steps should work, in case you run into issues, you might find the following steps for debugging helpful.

Try different include path ordering

If you see errors with `<cmath>`, or similar system headers, one thing that might be happening is that the include paths are in the wrong order. Try moving the include paths for the corresponding header before/after all other include paths.

Iterate on files one at a time

Instead of trying to make changes to the CMake configuration and recompiling the whole project, first try working on individual compilation commands as emitted in `compile_commands.json` and see if IWYU works as expected.

For each command, try replacing the compiler with the `include-what-you-use` binary or `iwyu_stub.py` (below) to see if the behavior is as expected. You may need to manually add some include paths as in `iwyu_run.sh` above. Make sure you update paths in the script before it works.

```
#!/usr/bin/env python3

# iwyu_stub.py

import os
import re
import subprocess
import sys

clang_path = "/usr/bin/clang"
clangxx_path = "/usr/bin/clang++"
project_dir = "/Users/username/swift-project/"
iwyu_bin_path = project_dir + "iwyu/build/bin/include-what-you-use"
log_dir = project_dir + "iwyu/logs/"

log_file = open(log_dir + "passthrough.log", "a+")

argv = sys.argv

def call_with_args(executable_path, args=argv):
    new_argv = args[:]
    new_argv[0] = executable_path
    log_file.write("# about to run:\n{}\n#---\n".format(' '.join(new_argv)))
    sys.exit(subprocess.call(new_argv))

# HACK: Relies on the compilation commands generated by CMake being
# of the form:
#
#   /path/to/compiler <other options> -c MyFile.ext
#
def try_using_iwyu(argv):
    return (argv[-2] == "-c") and ("/swift/" in argv[-1])

# Flag for quickly switching between IWYU and Clang for iteration.
# Useful for checking behavior for different include path combinations.
if argv[1] == "--forward-to-clangxx":
    call_with_args(clangxx_path, args=([argv[0]] + argv[2:]))

# Check that we are getting a compilation command.
if try_using_iwyu(argv):
  _, ext = os.path.splitext(argv[-1])
  if ext == ".m":
      call_with_args(clang_path)
```

```
  elif ext == ".mm":
      call_with_args(clangxx_path)
  elif ext in [".cxx", ".cc", ".cpp", ".c"]:
      call_with_args(iwyu_bin_path)
  log_file.write(
      "# Got a strange file extension.\n{}\n#---\n".format(' '.join(argv)))
  call_with_args(iwyu_bin_path)
else:
  # got something else, just forward to clang/clang++
  log_file.write(
      "# Not going to try using iwyu.\n{}\n#---\n".format(' '.join(argv)))
  _, ext = os.path.splitext(argv[-1])
  if ext == ".m" or ext == ".c":
      call_with_args(clang_path)
  else:
      call_with_args(clangxx_path)
```