Testing custom queries

In this article

You can set up tests for your CodeQL queries to ensure that they continue to return the expected results with new releases of the CodeQL CLI.

GitHub CodeQL is licensed on a per-user basis upon installation. You can use CodeQL only for certain tasks under the license restrictions. For more information, see "About the CodeQL CLI." If you have a GitHub Advanced Security license, you can use CodeQL for automated analysis, continuous integration, and continuous delivery. For more information, see "About GitHub Advanced Security."

About testing custom queries

CodeQL provides a simple test framework for automated regression testing of queries. Test your queries to ensure that they behave as expected.

During a query test, CodeQL compares the results the user expects the query to produce with those actually produced. If the expected and actual results differ, the query test fails. To fix the test, you should iterate on the query and the expected results until the actual results and the expected results exactly match. This topic shows you how to create test files and execute tests on them using the `test run` subcommand.

Setting up a test CodeQL pack for custom queries

All CodeQL tests must be stored in a special "test" CodeQL pack. That is, a directory for test files with a `qlpack.yml` file that defines:

```
name: <name-of-test-pack>
version: 0.0.0
dependencies:
  <codeql-libraries-and-queries-to-test>: "*"
extractor: <language-of-code-to-test>
```

The `dependencies` value specifies the CodeQL packs containing queries to test. Typically, these packs will be resolved from source, and so it is not necessary to specify a fixed version of the pack. The `extractor` defines which language the CLI will use to create test databases from the code files stored in this CodeQL pack. For more information, see "Customizing analysis with CodeQL packs."

You may find it useful to look at the way query tests are organized in the CodeQL repository. Each language has a `src` directory, `ql/<language>/ql/src`, that contains libraries and queries for analyzing codebases. Alongside the `src` directory, there is a `test` directory with tests for these libraries and queries.

Each `test` directory is configured as a test CodeQL pack with two subdirectories:

- `query-tests` a series of subdirectories with tests for queries stored in the `src` directory. Each subdirectory contains test code and a QL reference file that specifies the query to test.
- `library-tests` a series of subdirectories with tests for QL library files. Each subdirectory contains test code and queries that were written as unit tests for a library.

After creating the `qlpack.yml` file, you need to make sure that all of the dependencies are downloaded and available to the CLI. Do this by running the following command in the same directory as the `qlpack.yml` file:

```
codeql pack install
```

This will generate a `codeql-pack.lock.yml` file that specifies all of the transitive dependencies required to run queries in this pack. This file should be checked in to source control.

Setting up the test files for a query

For each query you want to test, you should create a sub-directory in the test CodeQL pack. Then add the following files to the subdirectory before you run the test command:

- A query reference file (`.qlref` file) defining the location of the query to test. The location is defined relative to the root of the CodeQL pack that contains the query. Usually, this is a CodeQL pack specified in the `dependencies` block of the test pack. For more information, see "Query reference files."

  You do not need to add a query reference file if the query you want to test is stored in the test directory, but it is generally good practice to store queries separately from tests. The only exception is unit tests for QL libraries, which tend to be stored in test packs, separate from queries that generate alerts or paths.

- The example code you want to run your query against. This should consist of one or more files containing examples of the code the query is designed to identify.

You can also define the results you expect to see when you run the query against the example code, by creating a file with the extension `.expected`. Alternatively, you can leave the test command to create the `.expected` file for you.

For an example showing how to create and test a query, see the example below.

> **Note:** Your `.ql`, `.qlref`, and `.expected` files must have consistent names:
>
> - If you want to directly specify the `.ql` file itself in the test command, it must have the same base name as the corresponding `.expected` file. For example, if the query is `MyJavaQuery.ql`, the expected results file must be `MyJavaQuery.expected`.
> - If you want to specify a `.qlref` file in the command, it must have the same base name as the corresponding `.expected` file, but the query itself may have a different name.
> - The names of the example code files don't have to be consistent with the other test files. All example code files found next to the `.qlref` (or `.ql`) file and in any subdirectories will be used to create a test database. Therefore, for simplicity, we recommend you don't save test files in directories that are ancestors of each other.

Running `codeql test run`

CodeQL query tests are executed by running the following command:

```
codeql test run <test|dir>
```

The `<test|dir>` argument can be one or more of the following:

- Path to a `.ql` file.
- Path to a `.qlref` file that references a `.ql` file.
- Path to a directory that will be searched recursively for `.ql` and `.qlref` files.

You can also specify:

- `--threads:` optionally, the number of threads to use when running queries. The default option is `1`. You can specify more threads to speed up query execution. Specifying `0` matches the number of threads to the number of logical processors.

For full details of all the options you can use when testing queries, see "test run."

Example

The following example shows you how to set up a test for a query that searches Java code for `if` statements that have empty `then` blocks. It includes steps to add the custom query and corresponding test files to separate CodeQL packs outside your checkout of the CodeQL repository. This ensures when you update the CodeQL libraries, or check out a different branch, you won't overwrite your custom queries and tests.

Prepare a query and test files

1. Develop the query. For example, the following simple query finds empty `then` blocks in Java code:

   ```
   import java

   from IfStmt ifstmt
   where ifstmt.getThen() instanceof EmptyStmt
   select ifstmt, "This if statement has an empty then."
   ```

2. Save the query to a file named `EmptyThen.ql` in a directory with your other custom queries. For example, `custom-queries/java/queries/EmptyThen.ql`.

3. If you haven't already added your custom queries to a CodeQL pack, create a CodeQL pack now. For example, if your custom Java queries are stored in `custom-queries/java/queries`, add a `qlpack.yml` file with the following contents to `custom-queries/java/queries`:

   ```
   name: my-custom-queries
   dependencies:
     codeql/java-queries: "*"
   ```

   For more information about CodeQL packs, see "Customizing analysis with CodeQL packs."

4. Create a CodeQL pack for your Java tests by adding a `qlpack.yml` file with the following contents to `custom-queries/java/tests`, updating the `dependencies` to match the name of your CodeQL pack of custom queries:

The following `qlpack.yml` file states that `my-github-user/my-query-tests` depends on `my-github-user/my-custom-queries` at a version greater than or equal to 1.2.3 and less than 2.0.0. It also declares that the CLI should use the Java `extractor` when creating test databases. The `tests: .` line declares that all `.ql` files in the pack should be run as tests when `codeql test run` is run with the `--strict-test-discovery` option. Typically, test packs do not contain a `version` property. This prevents you from accidentally publishing them.

```
name: my-github-user/my-query-tests
dependencies:
  my-github-user/my-custom-queries: ^1.2.3
extractor: java-kotlin
tests: .
```

5. Run `codeql pack install` in the root of the test directory. This generates a `codeql-pack.lock.yml` file that specifies all of the transitive dependencies required to run queries in this pack.
6. Within the Java test pack, create a directory to contain the test files associated with `EmptyThen.ql`. For example, `custom-queries/java/tests/EmptyThen`.
7. In the new directory, create `EmptyThen.qlref` to define the location of `EmptyThen.ql`. The path to the query must be specified relative to the root of the CodeQL pack that contains the query. In this case, the query is in the top level directory of the CodeQL pack named `my-custom-queries`, which is declared as a dependency for `my-query-tests`. Therefore, `EmptyThen.qlref` should simply contain `EmptyThen.ql`.
8. Create a code snippet to test. The following Java code contains an empty `if` statement on the third line. Save it in `custom-queries/java/tests/EmptyThen/Test.java`.

```
class Test {
  public void problem(String arg) {
    if (arg.isEmpty())
      ;
    {
      System.out.println("Empty argument");
    }
  }

  public void good(String arg) {
    if (arg.isEmpty()) {
      System.out.println("Empty argument");
    }
  }
}
```

Execute the test

To execute the test, move into the `custom-queries` directory and run `codeql test run java/tests/EmptyThen`.

When the test runs, it:

1. Finds one test in the `EmptyThen` directory.
2. Extracts a CodeQL database from the `.java` files stored in the `EmptyThen` directory.
3. Compiles the query referenced by the `EmptyThen.qlref` file.

If this step fails, it's because the CLI can't find your custom CodeQL pack. Re-run the command and specify the location of your custom CodeQL pack, for example:

```
codeql test run --search-path=java java/tests/EmptyThen
```

For information about saving the search path as part of your configuration, see "Specifying command options in a CodeQL configuration file."

4. Executes the test by running the query and generating an `EmptyThen.actual` results file.
5. Checks for an `EmptyThen.expected` file to compare with the `.actual` results file.
6. Reports the results of the test — in this case, a failure: `0 tests passed; 1 tests failed:`. The test failed because we haven't yet added a file with the expected results of the query.

View the query test output

CodeQL generates the following files in the `EmptyThen` directory:

- `EmptyThen.actual`, a file that contains the actual results generated by the query.
- `EmptyThen.testproj`, a test database that you can load into VS Code and use to debug failing tests. When tests complete successfully, this database is deleted in a housekeeping step. You can override this step by running `test run` with the `--keep-databases` option.

In this case, the failure was expected and is easy to fix. If you open the `EmptyThen.actual` file, you can see the results of the test:

```
| Test.java:3:5:3:22 | stmt | This if statement has an empty then. |
```

This file contains a table, with a column for the location of the result, along with separate columns for each part of the `select` clause the query outputs. Since the results are what we expected, we can update the file extension to define this as the expected result for this test (`EmptyThen.expected`).

If you rerun the test now, the output will be similar but it will finish by reporting: `All 1 tests passed.`.

If the results of the query change, for example, if you revise the `select` statement for the query, the test will fail. For failed results, the CLI output includes a unified diff of the `EmptyThen.expected` and `EmptyThen.actual` files. This information may be sufficient to debug trivial test failures.

For failures that are harder to debug, you can import `EmptyThen.testproj` into CodeQL for VS Code, execute `EmptyThen.ql`, and view the results in the `Test.java` example code. For more information, see "Analyzing your projects" in the CodeQL for VS Code help.

Further reading

- "CodeQL queries"
- "Testing CodeQL queries in Visual Studio Code."