Documentation

☰ NAVIGATION

Bazel query how-to

Edit (https://github.com/bazelbuild/bazel/tree/master/site/docs/query-how-to.md)

This is a quick tutorial to get you started using Bazel's query language to trace dependencies in your code.

For a language details and `--output` flag details, please see the reference manual, Bazel query reference (query.html). You can get help for Bazel query by typing `bazel help query`.

To execute a query while ignoring errors such as missing targets, use the `--keep_going` flag.

Contents

        - What genproto rules does bar depend upon?
        - Find the definition of some JNI (C++) library that is transitively depended upon by a Java binary rule in the servlet tree.
            - ...Now find the definitions of all the Java binaries that depend on them
    - What file dependencies exist ...
        - What's the complete set of Java source files required to build QUX?
        - What is the complete set of Java source files required to build QUX's tests?
    - What differences in dependencies between X and Y exist ...
        - What targets does `//foo` depend on that `//foo:foolib` does not?
        - What C++ libraries do the `foo` tests depend on that the `//foo` production binary does *not* depend on?
    - Why does this dependency exist ...
        - Why does `bar` depend on `groups2` ?
        - Show me a path from `docker/updater:updater_systest` (a `py_test`) to some `cc_library` that it depends upon:
        - Why does library `//photos/frontend:lib` depend on two variants of the same library `//third_party/jpeglib` and `//third_party/jpeg` ?
    - What depends on ...
        - What rules under bar depend on Y?
        - What targets directly depend on T, in T's package
    - How do I break a dependency ...
        - What dependency paths do I have to break to make `bar` no longer depend on X?
    - Misc ...
        - How many sequential steps are there in the `ServletSmokeTests` build?

Finding the Dependencies of a Rule

To see the dependencies of `//src/main/java/com/example/base:base` , use the `deps` function in bazel query:

```
$ bazel query "deps(src/main/java/com/example/base:base)"
//resources:translation.xml
//src/main/java/com/example/base:AbstractPublishedUri.java
...
```

This is the set of all targets required to build `//src/main/java/com/example/base:base` .

Tracing the Dependency Chain between Two Packages

The library `//third_party/zlib:zlibonly` isn't in the BUILD file for `//src/main/java/com/example/base` , but it is an indirect dependency. How can we trace this dependency path? There are two useful functions here: `allpaths` and `somepath`

```
$ bazel query "somepath(src/main/java/com/example/base:base, third_party/zlib:zlibonly)"
//src/main/java/com/example/base:base
//translations/tools:translator
//translations/base:base
//third_party/py/MySQL:MySQL
//third_party/py/MySQL:_MySQL.so
//third_party/mysql:mysql
//third_party/zlib:zlibonly
$ bazel query "allpaths(src/main/java/com/example/common/base:base, third_party/...)"
  ...many errors detected in BUILD files...
//src/main/java/com/example/common/base:base
//third_party/java/jsr166x:jsr166x
//third_party/java/sun_servlet:sun_servlet
//src/main/java/com/example/common/flags:flags
//src/main/java/com/example/common/flags:base
//translations/tools:translator
//translations/tools:aggregator
//translations/base:base
//tools/pkg:pex
//tools/pkg:pex_phase_one
//tools/pkg:pex_lib
//third_party/python:python_lib
//translations/tools:messages
//third_party/py/xml:xml
//third_party/py/xml:utils/boolean.so
//third_party/py/xml:parsers/sgmlop.so
//third_party/py/xml:parsers/pyexpat.so
//third_party/py/MySQL:MySQL
//third_party/py/MySQL:_MySQL.so
//third_party/mysql:mysql
//third_party/openssl:openssl
//third_party/zlib:zlibonly
//third_party/zlib:zlibonly_v1_2_3
//third_party/python:headers
//third_party/openssl:crypto
```

Aside: implicit dependencies

The BUILD file for `src/main/java/com/example/common/base` never references `//translations/tools:aggregator`. So, where's the direct dependency?

Certain rules include implicit dependencies on additional libraries or tools. For example, to build a `genproto` rule, you need first to build the Protocol Compiler, so every `genproto` rule carries an implicit dependency on the protocol compiler. These dependencies are not mentioned in the build file, but added in by the build tool. The full set of implicit dependencies is currently undocumented; read the source code of `RuleClassProvider` (https://github.com/bazelbuild/bazel/tree/master/src/main/java/com/google/devtools/build/lib/packages/RuleClassProvider.java).

Reverse Dependencies

You might want to know the set of targets that depends on some target. e.g., if you're going to change some code, you might want to know what other code you're about to break. You can use `rdeps(u, x)` to find the reverse dependencies of the targets in `x` within the transitive closure of `u`.

Unfortunately, invoking, e.g., `rdeps(..., daffie/annotations2:constants-lib)` is not practical for a large tree, because it requires parsing every BUILD file and building a very large dependency graph (Bazel may run out of memory). If you would like to execute this query across a large repository, you may have to query subtrees and then combine the results.

Miscellaneous Uses

You can use `bazel query` to analyze many dependency relationships.

What exists ...

What packages exist beneath `foo`?

```
bazel query 'foo/...' --output package
```

What rules are defined in the `foo` package?

```
bazel query 'kind(rule, foo:all)' --output label_kind
```

What files are generated by rules in the `foo` package?

```
bazel query 'kind("generated file", //foo:*)'
```

What's the set of BUILD files needed to build `//foo`?

```
bazel query 'buildfiles(deps(//foo))' --output location | cut -f1 -d:
```

What are the individual tests that a `test_suite` expands to?

```
bazel query 'tests(//foo:smoke_tests)'
```

Which of those are C++ tests?

```
bazel query 'kind(cc_.*, tests(//foo:smoke_tests))'
```

Which of those are small? Medium? Large?

```
bazel query 'attr(size, small, tests(//foo:smoke_tests))'

bazel query 'attr(size, medium, tests(//foo:smoke_tests))'

bazel query 'attr(size, large, tests(//foo:smoke_tests))'
```

What are the tests beneath `foo` that match a pattern?

```
bazel query 'filter("pa?t", kind(".*_test rule", //foo/...))'
```

The pattern is a regex and is applied to the full name of the rule. It's similar to doing

```
bazel query 'kind(".*_test rule", //foo/...)' | grep -E 'pa?t'
```

What package contains file src/main/java/com/example/cache/LRUCache.java`?

```
bazel query 'buildfiles(src/main/java/com/example/cache/LRUCache.java)' --output=package
```

What is the build label for src/main/java/com/example/cache/LRUCache.java?

```
bazel query src/main/java/com/example/cache/LRUCache.java
```

What rule target(s) contain file `src/main/java/com/example/cache/LRUCache.java` as a source?

```
fullname=$(bazel query src/main/java/com/example/cache/LRUCache.java)
bazel query "attr('srcs', $fullname, ${fullname//:*/}:*)"
```

What package dependencies exist ...

What packages does `foo` depend on? (What do I need to check out to build `foo`)

```
bazel query 'buildfiles(deps(//foo:foo))' --output package
```

Note, `buildfiles` is required in order to correctly obtain all files referenced by `subinclude`; see the reference manual for details.

What packages does the `foo` tree depend on, excluding `foo/contrib`?

```
bazel query 'deps(foo/... except foo/contrib/...)' --output package
```

What rule dependencies exist ...

What genproto rules does bar depend upon?

```
bazel query 'kind(genproto, deps(bar/...))'
```

Find the definition of some JNI (C++) library that is transitively depended upon by a Java binary rule in the servlet tree.

```
bazel query 'some(kind(cc_.*library, deps(kind(java_binary, src/main/java/com/example/fr
ontend/...))))' --output location
```

...Now find the definitions of all the Java binaries that depend on them

```
bazel query 'let jbs = kind(java_binary, src/main/java/com/example/frontend/...) in
  let cls = kind(cc_.*library, deps($jbs)) in
    $jbs intersect allpaths($jbs, $cls)'
```

What file dependencies exist ...

What's the complete set of Java source files required to build QUX?

Source files:

```
bazel query 'kind("source file", deps(src/main/java/com/example/qux/...))' | grep java$
```

Generated files:

```
bazel query 'kind("generated file", deps(src/main/java/com/example/qux/...))' | grep jav
a$
```

What is the complete set of Java source files required to build QUX's tests?

Source files:

```
bazel query 'kind("source file", deps(kind(".*_test rule", javatests/com/example/qux/...
)))' | grep java$
```

Generated files:

```
bazel query 'kind("generated file", deps(kind(".*_test rule", javatests/com/example/qux/
...)))' | grep java$
```

What differences in dependencies between X and Y exist ...

What targets does `//foo` depend on that `//foo:foolib` does not?

```
bazel query 'deps(//foo) except deps(//foo:foolib)'
```

What C++ libraries do the `foo` tests depend on that the `//foo` production binary does *not* depend on?

```
bazel query 'kind("cc_library", deps(kind(".*test rule", foo/...)) except deps(//foo))'
```

Why does this dependency exist ...

Why does `bar` depend on `groups2`?

```
bazel query 'somepath(bar/...,groups2/...:*)'
```

Once you have the results of this query, you will often find that a single target stands out as being an unexpected or egregious and undesirable dependency of `bar`. The query can then be further refined to:

Show me a path from `docker/updater:updater_systest` (a `py_test`) to some `cc_library` that it depends upon:

```
bazel query 'let cc = kind(cc_library, deps(docker/updater:updater_systest)) in
  somepath(docker/updater:updater_systest, $cc)'
```

Why does library `//photos/frontend:lib` depend on two variants of the same library `//third_party/jpeglib` and `//third_party/jpeg`?

This query boils down to: "show me the subgraph of `//photos/frontend:lib` that depends on both libraries". When shown in topological order, the last element of the result is the most likely culprit.

```
% bazel query 'allpaths(//photos/frontend:lib, //third_party/jpeglib)
                intersect
               allpaths(//photos/frontend:lib, //third_party/jpeg)'
//photos/frontend:lib
//photos/frontend:lib_impl
//photos/frontend:lib_dispatcher
//photos/frontend:icons
//photos/frontend/modules/gadgets:gadget_icon
//photos/thumbnailer:thumbnail_lib
//third_party/jpeg/img:renderer
```

What depends on ...

What rules under bar depend on Y?

```
bazel query 'bar/... intersect allpaths(bar/..., Y)'
```

Note: `X intersect allpaths(X, Y)` is the general idiom for the query "which X depend on Y?" If expression X is non-trivial, it may be convenient to bind a name to it using `let` to avoid duplication.

What targets directly depend on T, in T's package?

```
bazel query 'let t = T in rdeps(siblings($t), $t, 1)'
```

How do I break a dependency ...

What dependency paths do I have to break to make `bar` no longer depend on X?

To output the graph to a `png` file:

```
bazel query 'allpaths(bar/...,X)' --output graph | dot -Tpng > /tmp/dep.png
```

Misc ...

How many sequential steps are there in the `ServletSmokeTests` build?

Unfortunately, the query language can't currently give you the longest path from x to y, but it can find the (or rather *a*) most distant node from the starting point, or show you the *lengths* of the longest path from x to every y that it depends on. Use `maxrank`:

```
% bazel query 'deps(//src/test/java/com/example/servlet:ServletSmokeTests)' --output max
rank | tail -1
85 //third_party/zlib:zutil.c
```

The result indicates that there exist paths of length 85 that must occur in order in this build.

Support Policy (https://www.bazel.build/support.html)

Stay Connected

Twitter (https://twitter.com/bazelbuild)
Blog (https://blog.bazel.build)
GitHub (https://github.com/bazelbuild/bazel)
Discussion group (https://groups.google.com/forum/#!forum/bazel-discuss)