Bzlmod Inspection Tool (a.k.a ***modquery***)

Author: andreisolo@bazel.build (Andrei-Oliviu Sologon)
Status: Under review
Created: 2022-04-26
Updated: 2022-06-22

Please read Bazel Code of Conduct before commenting.

Hello bazel-discuss@googlegroups.com,

I am working on developing an inspection tool for the *Bzlmod* external module dependency system (Manage external dependencies with Bzlmod | Bazel) (somehow similar to `bazel query` Bazel Query How-To).

My experience with complex dependency trees is not very vast, therefore I would really appreciate some suggestions on what problems you have encountered and what features may be interesting or useful.

Feel free to suggest improvements on the **possible usages** as well as on the **actual syntax** and **output** of the commands.

mod

Description:

Queries the Bzlmod external dependency graph.

Usage:

```
bazel mod [<option> ...] <query_type> [<args> ...]
```

The command will display a dependency tree or parts of the dependency tree, structured to display different kinds of insights depending on the query type.

Calling the command with no argument will default to:
`bazel mod tree root`

`<query_type> [<args> ...]` can be one of the following:

- `tree`: Displays the full dependency tree. Use the `--from` option to specify which module(s) you want the tree to start from (defaults to `root` which displays the whole dependency tree).
- `deps <module(s)>`: Displays the direct dependencies of the target module(s).
- `path <module(s)_to>`: Displays the shortest path found in the dependency graph from (any of) the `--from` module(s) to (any of) `<module(s)_to>`.
- `all_paths <module(s)_to>`: Display the dependency graph starting from (any of) the `--from` module(s) and containing any existing paths to (any of) the `<module(s)_to>`.
- `explain <module(s)>`: Prints all the places where the module is (or was) requested as a direct dependency, along with the reason why the respective final version was selected. It will display a pruned version of the `all_paths root <module(s)>` command which only contains the direct deps of the root, the `<module(s)>` leaves, along with their dependants (can be modified with `--depth`).
- `show <repo(s)>`: Prints the rule that generated these repos (i.e. `http_archive()`).
- `show_extension <extension(s)>`: Prints information about the specified extension(s): the repositories generated by them, and the directives used (attributes set and `use_repo`) in each of the `--module_usages` modules specified (if present).

`<module>` arguments must be of type:

- `“<root>”`: The current (root) module you are inside of.
- `<name>@<version>`: A specific module version.
- `<name>@_`: Specifies the empty version of a module (for non-registry overridden) modules).

- `<name>` or `<name>@*`: Can be used as a placeholder for all the present versions of the module `<name>`. Used the later one if a `repo_name` with the same name exists, as the `repo_name` will take precedence.
- `<repo_name>:` The *repo_name* of one of the root project's direct dependencies, as it is defined in the *MODULE.bazel* file.

`<modules>` means:

- `<module>,<module>,...` : A list of comma separated modules, where each `<module>` has the form of one of the above.

`<extension>` can be one of the following:

- `<module>%extensionFileLabel%extensionName`: The module extension with the file label resolved relative to the specified module. The specified module should represent exactly one existing version.
- `extensionFileLabel%extensionName`: The module extension with the file label resolved relative to the root.

`<extensions>` means:

- `<extension>,<extension>,...` : A list of comma separated modules, where each `<extension>` has the form of one of the above.

`<repo>` can be one of the following:

- `<module>`
- `<module>:` Repo name of either a module or an extension-generated repo (relative to the `--module_mapping` module specified.

`<repos>` means:

- `<repo>,<repo>,...` : A list of comma separated modules, where each `<repo>` has the form of one of the above.

Query options:

- `--from <module(s)`: The module(s) starting from which the dependency graph query will be displayed. Check each query's description for the exact semantic. Defaults to *root*.

- `--extra`: The queries will also display the reason why modules were resolved to their current version (if changed). Defaults to true only for the `explain` query.
- `--unused`: The queries will also take into account and display the *unused* modules, which are not present in the module resolution graph after selection (due to the *Minimal-Version Selection* or *override* rules). This can have different effects for each of the query types i.e. include new paths in the `all_paths` command, or extra dependants in the `explain` command.
- `--extension_filter <extension(s)>`: Only display the usages of these module extensions and the repositories generated by them if their respective flags are set. The result graph will not include paths which do not contain modules that use the specified extensions if any. Use the `all` keyword to describe all present extensions if you want to only include modules that use extensions in the graph.
- `--extension_info <mode>:` specify how much detail about extension usages to include in the query result. Can be one of:
    - `hidden` (default): Nothing is shown.
    - `usages`: Include the extension usages only.
    - `repos`: Include the extension usages and the imported repos.
    - `all:` Include the above, and all the other non-imported repos generated by the extension as a result of its usages.
- `--module_mapping <module>:` specify a module relative to which the specified target repos will be interpreted. Defaults to `<root>`.
- `--module_usages <module(s)>:` specify modules for which their extension usages will be displayed in the `show_extension` query.
- `--depth <depth>`: Maximum display depth of the dependency tree. A depth of 1 displays the direct dependencies, for example. For `tree`, `path` and `all_paths` it defaults to `Integer.MAX_VALUE`, while for `deps` and `explain` it defaults to `1` (only displays direct deps of the root besides the target leaves and their parents).
- `--cycles`: Points out dependency cycles inside the displayed tree, which are normally ignored by default.
- `--builtin`: Include the builtin modules in the dependency graph. Disabled by default because it is quite noisy.

- `--charset` *charset:* Chooses the character set to use for the tree. Valid values are "utf8" or "ascii". Default is "utf8".
- `--prefix` *prefix:* Sets how each line is displayed (only affects `text` output). The *prefix* value can be one of:
    - indent (default) — Shows each line indented as a tree.
    - depth — Show as a list, with the numeric depth printed before each entry.

    - none — Show as a flat list.
- `--output <mode>`: specify the output format. Can be one of:
    - `text` (default): A human readable output in stdout
    - `json`: Outputs the result in JSON format
    - `graph`: Outputs the result in the Graphviz *dot* graph format

Example usages (**deprecated** - head to Example 2.0):

1. Display the direct dependencies of a module (inside your project)

```
#     A 1.0 (root)
#    /     \
# B 1.0   C 1.0

$ bazel mod deps
A@1.0 (root)
 └──B@1.0
 └──C@2.0

#           A 1.0
#         /    |
#   B 1.0    B 2.0
#     |        |    \
#   D 3.0   D 3.5   E 1.0
# multiple_version_override: B[1.0,2.0]

# The displayed result is post-resolution
$ bazel mod deps B
A@1.0 (root)
 └──B@1.0:
    └──D-3.5
 └──B@2.0:
    └──D-3.5
    └──E-1.0
```

2. Display the whole dependency tree of a module (with extra resolution information and unused modules).

```
# before resolution (initial)
#           A 1.0
#         /    |
#   B 1.0    B 2.0
#     |        |    \
#   D 3.0    D 3.5   E 1.0
#              |
#           E 1.2
# multiple_version_override: B[1.0,2.0]

# The displayed result is post-resolution
$ bazel mod --extra --unused tree
A@1.0 (root)
├──B@1.0
│  └──D@3.5 (was 3.0, from B@2.0)
├──B@2.0
│  └──D@3.5 (was 3.0, from B@2.0)
│     └──E@1.2
└──E1.2
```

3. Display the dependency trees of some of your modules (i.e. B 1.0 and D 3.5)

```
# the dashed line -- means the dependency is not a direct one, but a transitive
# edge
$ bazel mod --extra --unused --from B@1.0,D@3.5 tree
A@1.0 (root)
└──B@1.0
│  └──D@3.0 (replaced by 3.5, from B@2.0)
└┄D@3.5 (was 3.0, from B@2.0)
   └──E@1.2
```

4. Check how your project depends on some module E (the paths)

```
# (graph with original dependencies - before resolution):
#            A 1.0
#          /    |     \
#   B 1.0    B 2.5     F 1.0
#     |        |   \        \
#   C 1.0    D 3.5  E 1.0    B 2.3
#     |                       |
```

```
#   E 1.2                  D 3.0
# multiple_version_override: B[1.0,2.5]
# single_version_override: D[4.0]

# (gets resolved to):
#            A 1.0
#          /    |     \
#   B 1.0   B 2.5    F 1.0
#     |       |   \       \
#   C 1.0   D 4.0  E 1.2   B 2.5
#     |                      |   \
#   E 1.2                 D 4.0 E 1.2

$ bazel mod all_paths E@1.2
A@1.0 (root)
 ├──B@1.0
 │  └──C@1.0
 │     └──E@1.2
 ├──B@2.5
 │  └──E@1.2
 └──F@1.0
    └──B@2.5
       └──E@1.2


$ bazel mod path E@1.0
Module E@1.0 is not currently used. It was deprecated due to Minimal-Version
Selection.
To include its original usage use the --unused flag.


$ bazel mod --unused all_paths E
A@1.0 (root)
 └──B@1.0
 │  └──C@1.0
 │     └──E@1.2
 ├──B@2.5
 │  └──E@1.0 (unused)
 │  └──E@1.2
 └──F@1.0
    └──B@2.5 (*)
```

5. Check how each version of B in your project depends on each version of E (the paths)

```
# the dashed line -- means the dependency is not a direct one, but a transitive
# edge
$ bazel mod --unused --from B all_paths E
A@1.0 (root)
 ├──B@1.0
 |  └──C@1.0
 |     └──E@1.2
 └─B@2.5
 |  └──E@1.0 (unused)
 |  └──E@1.2
 └──B@2.5 (*)
```

6. Check how your project depends on some module D (paths, graph with cycles)

```
# A dep graph with cycles
# A 1.0
#   |
# B 1.0
#   |   \
# C 1.0  D 1.0
#   |
# B 1.0
$ bazel mod path D

A@1.0 (root)
 └──B@1.0
    └──D@1.0

$ bazel mod --cycles all_paths D

A@1.0 (root)
 └──B@1.0
    └──D@1.0
    └──C@1.0
       └──B@1.0 (cycle)
```

7. See why a module is part of your dependency graph

```
# (graph with original dependencies - before resolution):
#            A 1.0
#         /    |  \
```

```
#   B 1.0   B 2.5  E 2.5
#     |       |
#   C 1.0   D 3.5
#     |       |
#   D 3.0   E 3.5
#     |
#   E 3.0
# single_version_override: D[4.0]
# multiple_version_override: B[1.0,2.5]

# (gets resolved to):
#            A 1.0
#          /    |   \
#   B 1.0   B 2.5  E 4.0
#     |       |
#   C 1.0   D 4.0
#     |       |
#   D 4.0   E 4.0
#     |
#   E 4.0

$ bazel mod explain E
A@1.0 (root)
 ├──B@1.0
 |  └─D@4.0 (was 3.0, SVO)
 |     └──E@4.0
 ├──B@2.5
 |  └──D@4.0 (was 3.0, SVO)
 |     └──E@4.0
 └──E@4.0 (was 2.5, from D@4.0)
```

8. Inspect the repository generated by one of your modules.

```
$ bazel mod show E@1.2

http_archive(
  name="E.1.2",
  url="https://bazel.com/bazel-central-registry/modules/E.zip”
)

# Rule http_archive defined at (most recent call last):
#   $output_base/external/bazel_tools/tools/build_defs/repo/http.bzl:355:31 in
<toplevel>
```

9. See what module extensions are used within your project

```
# (resolved graph):
#            A 1.0
#         /    |   \
#   B 1.0   B 2.5  E 4.0
#     |       |
#   C 1.0   D 4.0
#     |       |
#     |       |
#   D 4.0   E 4.0
#     |
#   E 4.0
#
# (extension usages):
# C 1.0 (maven) -> repoF
# D 4.0 (maven) -> repoG
#              \-> repoH
# E 4.0 (gradle) -> repoJ

$ bazel mod tree --extension_show usages
A@1.0 (root)
├──B@1.0
│  └──C@1.0
│     ├─$ maven (@maven.3.8/:src%maven)
│     └──D@4.0
│        ├─$ gradle (@gradle.7.4/:src%gradle)
│        └──E@4.0
│           └─$ maven (@maven.3.8/:src%maven)
├──B@2.5
│  └──D@4.0 (*)
│     └──E@4.0 (*)
└──E@4.0 (*)
```

10. See where and how an extension was used .

```
$ bazel mod show_extension C@1.0%//:src/main.bzl%maven --from E@4.0

# @C.1.0//:maven.bzl


Fetched repositories:
   -   repoF (imported by C@1.0)
   -   repoJ (imported by E@4.0)
```

```
    -   repoM
    -   repoN
    -   repoO
    -   repoP


## E@4.0

# At @E.4.0//MODULE.bazel:33:10

maven = use_extension("@C.1.0//:src", "maven")
maven.dep(coord="repoJ:1.0")
maven.pom(pom_xml="//:pom.xml")
use_repo(
    maven,
    repoJ
)
```

11. See where a particular extension is used (along all the other repos generated by the extension which the requested repos depend on)

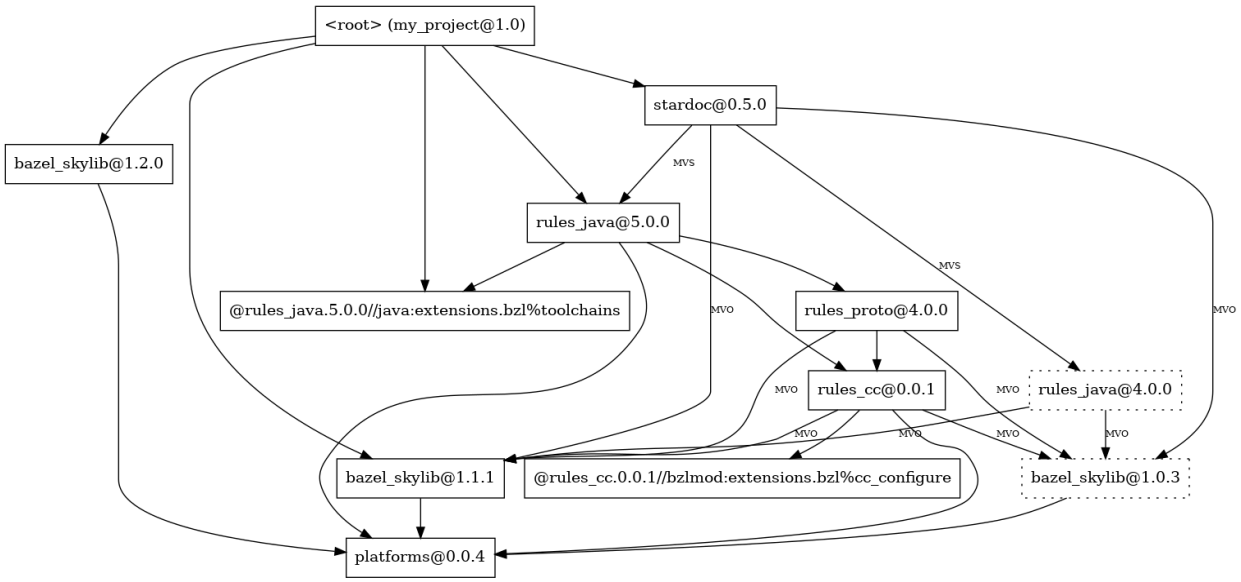
```
$ bazel mod tree --extension_show usages
A@1.0 (root)
 ├──B@1.0
 |  └──C@1.0
 |     ├─$ maven (@maven.3.8/:src/main.bzl%maven)
 |     └──D@4.0
 |        ├─$ gradle (@gradle.7.4/:src/main.bzl%gradle)
 |        └──E@4.0
 |           └─$ maven (@maven.3.8/:src/main.bzl%maven)
 ├──B@2.5
 |  └──D@4.0 (*)
 |     └──E@4.0 (*)
 └──E@4.0 (*)


$ bazel mod tree --extension_filter C@1.0%@maven.3.8/:src/main.bzl%maven
--extension_show all

A@1.0 (root)
 └──B@1.0
    └──C@1.0
       ├──$maven (@maven.3.8/:src/main.bzl%maven)
       |  ├──repoF
       |  ├─-repoM
       |  ├─-repoN
       |  ├─-repoO
```

```
      |  └─repoP
      └──D@4.0
         └──E@4.0
            └──$maven (@maven.3.8/:src/main.bzl%maven) (*)
               └──repoJ
```

Example 2.0



MODULE.bazel

```
module(
  name = "my_project",
  version = "1.0",
)

bazel_dep(name = "bazel_skylib", version = "1.1.1", repo_name = "skylib1")
bazel_dep(name = "bazel_skylib", version = "1.2.0", repo_name = "skylib2")
multiple_version_override(module_name = "bazel_skylib", versions =
["1.1.1", "1.2.0"])

bazel_dep(name = "stardoc", version = "0.5.0")
bazel_dep(name = "rules_java", version = "5.0.0")


toolchains = use_extension("@rules_java//java:extensions.bzl",
```

```
"toolchains")
use_repo(toolchains, my_jdk="remotejdk17_linux")
```

In the following example the builtin modules and their dependency edges are not shown, because they are quite noisy. They can be displayed with the --builtin option. Also, currently Bzlmod is an experimental feature and the --epxerimental_enable_bzlmod flag must be set for the `mod` command to work.

Usages:

1. See what dependencies one of your modules has.

```
$bazel mod deps stardoc@0.5.0

<root> (my_project@1.0)
└────stardoc@0.5.0
    ├────bazel_skylib@1.1.1
    └────rules_java@5.0.0
```

2. See the whole dependency tree of your project.

```
$bazel mod tree

<root> (my_project@1.0)
├────bazel_skylib@1.1.1
│    └────platforms@0.0.4
├────bazel_skylib@1.2.0
│    └────platforms@0.0.4 (*)
├────rules_java@5.0.0
│    ├────platforms@0.0.4 (*)
│    ├────rules_cc@0.0.1
│    │    ├────bazel_skylib@1.1.1 (*)
│    │    └────platforms@0.0.4 (*)
│    └────rules_proto@4.0.0
│         ├────bazel_skylib@1.1.1 (*)
│         └────rules_cc@0.0.1 (*)
└────stardoc@0.5.0
    ├────bazel_skylib@1.1.1 (*)
    └────rules_java@5.0.0 (*)
```

3. See the whole dependency tree of one of your modules.

```
$bazel mod tree --from stardoc@0.5.0

<root> (my_project@1.0)
└────stardoc@0.5.0
    ├────bazel_skylib@1.1.1
    │   └────platforms@0.0.4
    └────rules_java@5.0.0
        ├────platforms@0.0.4 (*)
        ├────rules_cc@0.0.1
        │   ├────bazel_skylib@1.1.1 (*)
        │   └────platforms@0.0.4 (*)
        └────rules_proto@4.0.0
            ├────bazel_skylib@1.1.1 (*)
            └────rules_cc@0.0.1 (*)
```

4. See the dependency tree starting from more modules (including unused).

```
$bazel mod tree --unused --from rules_java@*

<root> (my_project@1.0)
├────rules_java@5.0.0
│   ├────platforms@0.0.4
│   ├────rules_cc@0.0.1
│   │   ├────bazel_skylib@1.0.3 (*) (unused)
│   │   ├────bazel_skylib@1.1.1 (*)
│   │   └────platforms@0.0.4 (*)
│   └────rules_proto@4.0.0
│       ├────bazel_skylib@1.0.3 (*) (unused)
│       ├────bazel_skylib@1.1.1 (*)
│       └────rules_cc@0.0.1 (*)
└──rules_java@4.0.0 (unused)
    ├────bazel_skylib@1.0.3 (unused)
    │   └────platforms@0.0.4 (*)
    └────bazel_skylib@1.1.1
        └────platforms@0.0.4 (*)
```

5. See why your project depends on some modules. Use the --extra flag to include information about the module version resolution.

```
$bazel mod explain bazel_skylib@1.1.1 --extra --unused

<root> (my_project@1.0)
```

```
├───bazel_skylib@1.1.1
├───rules_java@5.0.0
│   ├───rules_cc@0.0.1
│   │   └───bazel_skylib@1.1.1 (*) (was 1.0.3, cause multiple_version_override)
│   └───rules_proto@4.0.0
│       ├───bazel_skylib@1.1.1 (*) (was 1.0.3, cause multiple_version_override)
│       └───rules_cc@0.0.1 (*)
└───stardoc@0.5.0
    ├───bazel_skylib@1.1.1 (*) (was 1.0.3, cause multiple_version_override)
    ├───rules_cc@0.0.1
    │   └───bazel_skylib@1.1.1 (*) (was 1.0.3, cause multiple_version_override)
    └───rules_proto@4.0.0
        ├───bazel_skylib@1.1.1 (*) (was 1.0.3, cause multiple_version_override)
        └───rules_cc@0.0.1 (*)
```

6. See how some modules depend on other modules of your project (full paths between them).

```
$bazel mod --from stardoc all_paths bazel_skylib@1.1.1 --extra --unused

<root> (my_project@1.0)
└───stardoc@0.5.0
    ├───bazel_skylib@1.1.1 (was 1.0.3, cause multiple_version_override)
    └───rules_java@5.0.0 (was 4.0.0, cause <root>, bazel_tools@_)
        ├───rules_cc@0.0.1
        │   └───bazel_skylib@1.1.1 (*) (was 1.0.3, cause multiple_version_override)
        └───rules_proto@4.0.0
            ├───bazel_skylib@1.1.1 (*) (was 1.0.3, cause multiple_version_override)
            └───rules_cc@0.0.1 (*)
```

7. See the underlying rule of one of your modules' repo. (`repo_name`s can also be used to specify modules in any query type. They will be relative to the `--module_mapping <module>` specified, which defaults to `<root>`.)

```
$bazel mod show skylib1,skylib2

# skylib1:
# <builtin>
http_archive(
  name = "bazel_skylib.1.1.1",
  urls =
["https://bcr.bazel.build/test-mirror/github.com/bazelbuild/bazel-skylib/releases/download/
1.1.1/bazel-skylib-1.1.1.tar.gz",
```

```
"https://github.com/bazelbuild/bazel-skylib/releases/download/1.1.1/bazel-skylib-1.1.1.tar.gz"],
  integrity = "sha256-xpZuyCjaGYxdmtuqlMBeOhx/Ib0BKgspuo3bzLLJOw0=",
  strip_prefix = "",
  remote_patches = {},
  remote_patch_strip = 0,
)
# Rule http_archive defined at (most recent call last):
#
/home/user/.cache/bazel/bazel_user/6e893e0f5a92cc4cf5909a6e4b2770f9/external/bazel_tools/tools/build_defs/repo/http.bzl:355:31 in <toplevel>

## skylib2:
# <builtin>
http_archive(
  name = "bazel_skylib.1.2.0",
  urls =
["https://bcr.bazel.build/test-mirror/github.com/bazelbuild/bazel-skylib/releases/download/1.2.0/bazel-skylib-1.2.0.tar.gz",
"https://github.com/bazelbuild/bazel-skylib/releases/download/1.2.0/bazel-skylib-1.2.0.tar.gz"],
  integrity = "sha256-r4eVmv5Jfcjf1MbLZuEnnLmMzIQoRhnr/sJ9nAmpA94=",
  strip_prefix = "",
  remote_patches = {},
  remote_patch_strip = 0,
)
# Rule http_archive defined at (most recent call last):
#
/home/user/.cache/bazel/bazel_user/6e893e0f5a92cc4cf5909a6e4b2770f9/external/bazel_tools/tools/build_defs/repo/http.bzl:355:31 in <toplevel>
```

8. See what module extensions are being used throughout the project.

```
$bazel mod tree --extension_info usages --extension_filter all

<root> (my_project@1.0)
├────$@rules_java.5.0.0//java:extensions.bzl%toolchains
├────rules_java@5.0.0 #
│    ├────$@rules_java.5.0.0//java:extensions.bzl%toolchains
│    ├────rules_cc@0.0.1 #
│    │    └────$@rules_cc.0.0.1//bzlmod:extensions.bzl%cc_configure
│    └────rules_proto@4.0.0
│         └────rules_cc@0.0.1 (*)
└────stardoc@0.5.0
     └────rules_java@5.0.0 (*)
```

9. See where a particular extension is used and the repos it generated as part of the dependency tree.

```
$bazel mod tree --extension_info all --extension_filter
@rules_java//java:extensions.bzl%toolchains
# or
$bazel mod tree --extension_info all --extension_filter
rules_java@5.0.0%//java:extensions.bzl%toolchains

<root> (my_project@1.0)
├───$@rules_java.5.0.0//java:extensions.bzl%toolchains
│    ├───remotejdk17_linux
│    ├──remotejdk11_linux
│    ├──remotejdk11_linux_aarch64
│    ├──remotejdk11_linux_ppc64le
│    ├──remotejdk11_linux_s390x
│    ├──remotejdk11_macos
│    ├──remotejdk11_macos_aarch64
│    ├──remotejdk11_win
│    ├──remotejdk15_linux
│    ├──remotejdk15_macos
│    ├──remotejdk15_macos_aarch64
│    ├──remotejdk15_win
│    ├──remotejdk16_linux
│    ├──remotejdk16_macos
│    ├──remotejdk16_macos_aarch64
│    ├───remotejdk16_win
│    ├──remotejdk17_macos
│    ├──remotejdk17_macos_aarch64
│    └──remotejdk17_win
├───rules_java@5.0.0 #
│    └───$@rules_java.5.0.0//java:extensions.bzl%toolchains (*)
│        ├───local_jdk
│        ├───remote_java_tools
│        ├───remote_java_tools_darwin
│        ├───remote_java_tools_linux
│        ├───remote_java_tools_windows
│        ├───remotejdk11_linux_aarch64_toolchain_config_repo
│        ├───remotejdk11_linux_ppc64le_toolchain_config_repo
│        ├───remotejdk11_linux_s390x_toolchain_config_repo
│        ├───remotejdk11_linux_toolchain_config_repo
│        ├───remotejdk11_macos_aarch64_toolchain_config_repo
│        ├───remotejdk11_macos_toolchain_config_repo
│        ├───remotejdk11_win_toolchain_config_repo
```

```
│       ├───remotejdk15_linux_toolchain_config_repo
│       ├───remotejdk15_macos_aarch64_toolchain_config_repo
│       ├───remotejdk15_macos_toolchain_config_repo
│       ├───remotejdk15_win_toolchain_config_repo
│       ├───remotejdk16_linux_toolchain_config_repo
│       ├───remotejdk16_macos_aarch64_toolchain_config_repo
│       ├───remotejdk16_macos_toolchain_config_repo
│       ├───remotejdk16_win_toolchain_config_repo
│       ├───remotejdk17_linux_toolchain_config_repo
│       ├───remotejdk17_macos_aarch64_toolchain_config_repo
│       ├───remotejdk17_macos_toolchain_config_repo
│       └───remotejdk17_win_toolchain_config_repo
└───stardoc@0.5.0
    └───rules_java@5.0.0 (*)
```

10. See info about an extension: generated repos and how it is used in different modules.

```
$bazel mod show_extension rules_java@5.0.0%//java:extensions.bzl%toolchains

## @rules_java.5.0.0//java:extensions.bzl%toolchains:

Fetched repositories:
  - local_jdk (imported by bazel_tools@_, rules_java@5.0.0)
  - remote_java_tools (imported by bazel_tools@_, rules_java@5.0.0)
  - remote_java_tools_darwin (imported by bazel_tools@_, rules_java@5.0.0)
  - remote_java_tools_linux (imported by bazel_tools@_, rules_java@5.0.0)
  - remote_java_tools_windows (imported by bazel_tools@_, rules_java@5.0.0)
  - remotejdk11_linux_aarch64_toolchain_config_repo (imported by rules_java@5.0.0)
  - remotejdk11_linux_ppc64le_toolchain_config_repo (imported by rules_java@5.0.0)
  - remotejdk11_linux_s390x_toolchain_config_repo (imported by rules_java@5.0.0)
  - remotejdk11_linux_toolchain_config_repo (imported by rules_java@5.0.0)
  - remotejdk11_macos_aarch64_toolchain_config_repo (imported by rules_java@5.0.0)
  - remotejdk11_macos_toolchain_config_repo (imported by rules_java@5.0.0)
  - remotejdk11_win_toolchain_config_repo (imported by rules_java@5.0.0)
  - remotejdk15_linux_toolchain_config_repo (imported by rules_java@5.0.0)
  - remotejdk15_macos_aarch64_toolchain_config_repo (imported by rules_java@5.0.0)
  - remotejdk15_macos_toolchain_config_repo (imported by rules_java@5.0.0)
  - remotejdk15_win_toolchain_config_repo (imported by rules_java@5.0.0)
  - remotejdk16_linux_toolchain_config_repo (imported by rules_java@5.0.0)
  - remotejdk16_macos_aarch64_toolchain_config_repo (imported by rules_java@5.0.0)
  - remotejdk16_macos_toolchain_config_repo (imported by rules_java@5.0.0)
  - remotejdk16_win_toolchain_config_repo (imported by rules_java@5.0.0)
  - remotejdk17_linux (imported by <root>)
  - remotejdk17_linux_toolchain_config_repo (imported by rules_java@5.0.0)
```

```
  - remotejdk17_macos_aarch64_toolchain_config_repo (imported by rules_java@5.0.0)
  - remotejdk17_macos_toolchain_config_repo (imported by rules_java@5.0.0)
  - remotejdk17_win_toolchain_config_repo (imported by rules_java@5.0.0)
  - remotejdk11_linux
  - remotejdk11_linux_aarch64
  - remotejdk11_linux_ppc64le
  - remotejdk11_linux_s390x
  - remotejdk11_macos
  - remotejdk11_macos_aarch64
  - remotejdk11_win
  - remotejdk15_linux
  - remotejdk15_macos
  - remotejdk15_macos_aarch64
  - remotejdk15_win
  - remotejdk16_linux
  - remotejdk16_macos
  - remotejdk16_macos_aarch64
  - remotejdk16_win
  - remotejdk17_macos
  - remotejdk17_macos_aarch64
  - remotejdk17_win

## Usage in <root> from <root>/MODULE.bazel
With the specified tags:...
maven.dep(key=value)
maven.pom(key=value)
use_repo(
  remotejdk17_linux="my_jdk",
)

## Usage in bazel_tools@_ from bazel_tools@_/MODULE.bazel:23
use_repo(
  "local_jdk",
  "remote_java_tools",
  "remote_java_tools_linux",
  "remote_java_tools_windows",
  "remote_java_tools_darwin",
)

## Usage in rules_java@5.0.0 from rules_java@5.0.0/MODULE.bazel:30
use_repo(
  "remote_java_tools",
  "remote_java_tools_linux",
  "remote_java_tools_windows",
  "remote_java_tools_darwin",
  "local_jdk",
  "remotejdk11_linux_toolchain_config_repo",
```

```
  "remotejdk11_macos_toolchain_config_repo",
  "remotejdk11_macos_aarch64_toolchain_config_repo",
  "remotejdk11_win_toolchain_config_repo",
  "remotejdk15_linux_toolchain_config_repo",
  "remotejdk15_macos_toolchain_config_repo",
  "remotejdk15_macos_aarch64_toolchain_config_repo",
  "remotejdk15_win_toolchain_config_repo",
  "remotejdk16_linux_toolchain_config_repo",
  "remotejdk16_macos_toolchain_config_repo",
  "remotejdk16_macos_aarch64_toolchain_config_repo",
  "remotejdk16_win_toolchain_config_repo",
  "remotejdk17_linux_toolchain_config_repo",
  "remotejdk17_macos_toolchain_config_repo",
  "remotejdk17_macos_aarch64_toolchain_config_repo",
  "remotejdk17_win_toolchain_config_repo",
  "remotejdk11_linux_aarch64_toolchain_config_repo",
  "remotejdk11_linux_ppc64le_toolchain_config_repo",
  "remotejdk11_linux_s390x_toolchain_config_repo",
)
```

11. See the underlying rule of an extension generated repo.

```
$bazel mod --experimental_enable_bzlmod show --module_mapping rules_java local_jdk

## local_jdk:
# <builtin>
_local_java_repository_rule(
  name = "rules_java.5.0.0.toolchains.local_jdk",
  build_file = "@rules_java.5.0.0//toolchains:jdk.BUILD",
  java_home = "",
  version = "",
)

# Rule _local_java_repository_rule defined at (most recent call last):
#
/home/user/.cache/bazel/bazel_user/6e893e0f5a92cc4cf5909a6e4b2770f9/external/rules_java.5.0
.0/toolchains/local_java_repository.bzl:248:46 in <toplevel>
```