Overview

EDIT

The CLI of coursier has a number of commands to deal with dependencies and artifacts:

- `complete` allows one to complete Maven coordinates,
- `resolve` lists the transitive dependencies of one or more dependencies,
- `fetch` fetches the artifacts of one or more dependencies,
- `launch` runs applications based on Maven / Ivy dependencies,
- `bootstrap` generates convenient launchers to run them,
- `install` installs applications based on Maven / Ivy dependencies,
- `java` and `java-home` install, run, and get the home directory of JVMs,
- `setup` checks if your system has a JVM and the standard Scala applications, and installs them if needed.

See Installation for how to install the CLI of coursier.

This page succinctly describes each of these commands. More details about each them are then given in the dedicated documentation page of each command (see links on the left).

Available commands

`complete`

`complete` allows one to complete Maven coordinates.

For example, let's display all published versions of the `HikariCP` library:

```
$ cs complete com.zaxxer:HikariCP:
1.1.3
1.1.4

<elided>

3.3.1
3.4.0
```

As another example, let's see which versions of the Scala compiler are available:

```
$ cs complete org.scala-lang:scala-compiler:
2.3.1
2.3.3
2.4.0-RC1
2.4.0-RC2
2.4.0
2.5.0-RC1
2.5.0-RC2
2.5.0
2.5.1

<elided>

2.12.0
2.12.1
2.12.2
2.12.3
2.12.4
2.12.5
2.12.6
2.12.7
2.12.8
2.12.9
2.12.10
2.12.11
2.12.12
2.13.0-M1
2.13.0-M2
2.13.0-M3
2.13.0-M3-f73b161
2.13.0-M4
2.13.0-M4-pre-20d3c21
2.13.0-M5
2.13.0-M5-5eef812
2.13.0-M5-6e0cba7
2.13.0-M5-1775dba
2.13.0-RC1
2.13.0-RC2
2.13.0-RC3
2.13.0
2.13.1
2.13.2
```

`resolve`

`resolve` lists the transitive dependencies of one or more other dependencies. Use like

```
$ cs resolve io.circe::circe-generic:0.12.3
com.chuusai:shapeless_2.13:2.3.3:default
io.circe:circe-core_2.13:0.12.3:default
io.circe:circe-generic_2.13:0.12.3:default
io.circe:circe-numbers_2.13:0.12.3:default
org.scala-lang:scala-library:2.13.0:default
org.typelevel:cats-core_2.13:2.0.0:default
org.typelevel:cats-kernel_2.13:2.0.0:default
org.typelevel:cats-macros_2.13:2.0.0:default
```

Note that this only relies on metadata files (POMs in particular), and doesn't download any JAR.

`resolve` has more options, to print trees, find which dependency brings another one, etc. See the dedicated page for more details.

`fetch`

`fetch` fetches the JARs of one or more dependencies.

```
$ cs fetch io.circe::circe-generic:0.12.3
/path/to/coursier/cache/https/repo1.maven.org/maven2/io/circe/circe-generic_2.13/0.12.3/c
/path/to/coursier/cache/https/repo1.maven.org/maven2/org/scala-lang/scala-library/2.13.0/
/path/to/coursier/cache/https/repo1.maven.org/maven2/io/circe/circe-core_2.13/0.12.3/circ
/path/to/coursier/cache/https/repo1.maven.org/maven2/com/chuusai/shapeless_2.13/2.3.3/sha
/path/to/coursier/cache/https/repo1.maven.org/maven2/io/circe/circe-numbers_2.13/0.12.3/c
/path/to/coursier/cache/https/repo1.maven.org/maven2/org/typelevel/cats-core_2.13/2.0.0/c
/path/to/coursier/cache/https/repo1.maven.org/maven2/org/typelevel/cats-macros_2.13/2.0.0
/path/to/coursier/cache/https/repo1.maven.org/maven2/org/typelevel/cats-kernel_2.13/2.0.0
```

`fetch` has more options, to join its output with the path separator, fetch source JARs, fetch javadoc, etc. See the dedicated page for more details.

`launch`

`launch` launches applications from one or more dependencies.

```
    -h, --help                  prints this usage text
    -v, --version               print version
  …
```

Arguments can be passed to the application after `--`.

`launch` has more options, to specify a main class, pass Java options, etc. See the dedicated page for more details.

`bootstrap`

`bootstrap` creates binary launchers from one or more dependencies.

```
$ cs bootstrap org.scalameta::scalafmt-cli:2.4.2 -o scalafmt
$ ./scalafmt --version
scalafmt 2.4.2
```

`bootstrap` can generate a variety of launchers, including native ones like GraalVM native images or Scala Native executables. See the dedicated page for more details.

`install`

The `install` command creates launchers for applications in the installation directory (`~/.local/share/coursier/bin` on Linux):

```
$ eval "$(cs install --env)" # add installation directory in PATH in the current session
$ cs install scalafmt
$ scalafmt --version
scalafmt 2.4.2
```

One can run `cs install --setup` to update profile files (Linux / macOS) or user environment variables (Windows), to add the installation directory to `PATH`.

See the dedicated page for more details.

`java`

The `java` command manages JVMs. For example, the following command will automatically download the latest AdoptOpenJDK 11 (in the coursier cache), unpack it (in the managed JVM directory), and run it with `-version`.

It uses the index of jabba to know where to download JVM archives, and assumes AdoptOpenJDK if only a version is passed.

The `java-home` command prints the Java home of a JVM, like

```
$ cs java-home --jvm 9
~/Library/Caches/Coursier/jvm/adopt@1.9.0-0/Contents/Home
```

One can update profile files (Linux / macOS) or user environment variables (Windows) for a specific JVM with `--setup`, like

```
$ cs java --jvm openjdk:1.14 --setup
```

(This updates `JAVA_HOME` and `PATH`.)

You don't have to update these environment variables if you prefer not to. For example, one can just get a Java home with

```
$ cs java-home --jvm graalvm:20
~/Library/Caches/Coursier/jvm/graalvm@20.0.0/Contents/Home
```

or set environment variables only for the current session with

```
$ eval "$(cs java --env --jvm 11)"
$ java -version
openjdk version "11.0.6" 2020-01-14
OpenJDK Runtime Environment AdoptOpenJDK (build 11.0.6+10)
OpenJDK 64-Bit Server VM AdoptOpenJDK (build 11.0.6+10, mixed mode)
```

These two commands only write things in the coursier cache, and the managed JVM directory, which is a cache too (both of these directories live under `~/.cache/coursier` on Linux, `~/Library/Caches/Coursier` on macOS).

The dedicated page gives more details about the `java` and `java-home` commands.

- the installation directory of the `install` command is in your `PATH`,
- the standard Scala applications are installed on your system.

`setup` is a "repackaging" of features of the `java` and `install` commands. One doesn't have to run `setup` for the other commands to work fine, it's just a convenience.

The features that `setup` wraps up are not tied to each other. You can rely on the `java` or `java-home` commands while managing applications with your preferred package manager. Or you can manage JVMs any other way while using the `install` command. `setup` only conveniently sets them up in one go, and it only sets up a JVM if it doesn't find one already installed.

See the dedicated page for more details about what the `setup` command does.

← SBT-PGP-COURSIER INSTALLATION →

Copyright © 2021 coursier contributors