

The jdeps Command

- [Name](#)
- [Synopsis](#)
- [Description](#)
- [Possible Options](#)
- [Module Dependence Analysis Options](#)
- [Options to Filter Dependences](#)
- [Options to Filter Classes to be Analyzed](#)
- [Example of Analyzing Dependencies](#)
- [Example Using the --inverse Option](#)

Name

jdeps - launch the Java class dependency analyzer

Synopsis

jdeps [*options*] *path* ...

options

Command-line options. For detailed descriptions of the options that can be used, see

- [Possible Options](#)
- [Module Dependence Analysis Options](#)
- [Options to Filter Dependences](#)
- [Options to Filter Classes to be Analyzed](#)

path

A pathname to the `.class` file, directory, or JAR file to analyze.

Description

The `jdeps` command shows the package-level or class-level dependencies of Java class files. The input class can be a path name to a `.class` file, a directory, a JAR file, or it can be a fully qualified class name to analyze all class files. The options determine the output. By default, the `jdeps` command writes the dependencies to the system output. The command can generate the dependencies in DOT language (see the `-dotoutput` option).

Possible Options

`-?` or `-h` or `--help`

Prints the help message.

`-dotoutput dir` or `--dot-output dir`

Specifies the destination directory for DOT file output. If this option is specified, then the `jdeps` command generates one `.dot` file for each analyzed archive named `archive-file-name.dot` that lists the dependencies, and also a summary file named `summary.dot` that lists the dependencies among the archive files.

`-s` or `-summary`

Prints a dependency summary only.

`-v` or `-verbose`

Prints all class-level dependencies. This is equivalent to

`-verbose:class -filter:none`

`-verbose:package`

Prints package-level dependencies excluding, by default, dependencies within the same package.

`-verbose:class`

Prints class-level dependencies excluding, by default, dependencies within the same archive.

`-apionly` or `--api-only`

Restricts the analysis to APIs, for example, dependences from the signature of `public` and `protected` members of public classes including field type, method parameter types, returned type, and checked exception types.

`-jdkinternals` or `--jdk-internals`

Finds class-level dependences in the JDK internal APIs. By default, this option analyzes all classes specified in the `--classpath` option and input files unless you specified the `-include` option. You can't use this option with the `-p`, `-e`, and `-s` options.

Warning: The JDK internal APIs are inaccessible.

`-cp` *path*, `-classpath` *path*, or `--class-path` *path*

Specifies where to find class files.

`--module-path` *module-path*

Specifies the module path.

`--upgrade-module-path` *module-path*

Specifies the upgrade module path.

`--system` *java-home*

Specifies an alternate system module path.

`--add-modules` *module-name* [, *module-name*...]

Adds modules to the root set for analysis.

`--multi-release` *version*

Specifies the version when processing multi-release JAR files. *version* should be an integer ≥ 9 or `base`.

`-q` or `-quiet`

Doesn't show missing dependencies from `-generate-module-info` output.

`-version` or `--version`

Prints version information.

Module Dependence Analysis Options

`-m` *module-name* or `--module` *module-name*

Specifies the root module for analysis.

`--generate-module-info` *dir*

Generates `module-info.java` under the specified directory. The specified JAR files will be analyzed. This option cannot be used with `--dot-output` or `--class-path` options. Use the `--generate-open-module` option for open modules.

`--generate-open-module` *dir*

Generates `module-info.java` for the specified JAR files under the specified directory as open modules. This option cannot be used with the `--dot-output` or `--class-path` options.

`--check` *module-name* [, *module-name*...]

Analyzes the dependence of the specified modules. It prints the module descriptor, the resulting module dependences after analysis and the graph after transition reduction. It also

identifies any unused qualified exports.

`--list-deps`

Lists the module dependencies and also the package names of JDK internal APIs (if referenced). This option transitively analyzes libraries on class path and module path if referenced. Use `--no-recursive` option for non-transitive dependency analysis.

`--list-reduced-deps`

Same as `--list-deps` without listing the implied reads edges from the module graph. If module M1 reads M2, and M2 requires transitive on M3, then M1 reading M3 is implied and is not shown in the graph.

`--print-module-deps`

Same as `--list-reduced-deps` with printing a comma-separated list of module dependencies. The output can be used by `jlink --add-modules` to create a custom image that contains those modules and their transitive dependencies.

`--ignore-missing-deps`

Ignore missing dependencies.

Options to Filter Dependencies

`-p pkg_name`, `-package pkg_name`, or `--package pkg_name`

Finds dependencies matching the specified package name. You can specify this option multiple times for different packages. The `-p` and `-e` options are mutually exclusive.

`-e regex`, `-regex regex`, or `--regex regex`

Finds dependencies matching the specified pattern. The `-p` and `-e` options are mutually exclusive.

`--require module-name`

Finds dependencies matching the given module name (may be given multiple times). The `--package`, `--regex`, and `--require` options are mutually exclusive.

`-f regex` or `-filter regex`

Filters dependencies matching the given pattern. If give multiple times, the last one will be selected.

`-filter:package`

Filters dependencies within the same package. This is the default.

`-filter:archive`

Filters dependencies within the same archive.

`-filter:module`

Filters dependencies within the same module.

`-filter:none`

No `-filter:package` and `-filter:archive` filtering. Filtering specified via the `-filter` option still applies.

`--missing-deps`

Finds missing dependencies. This option cannot be used with `-p`, `-e` and `-s` options.

Options to Filter Classes to be Analyzed

`-include regex`

Restricts analysis to the classes matching pattern. This option filters the list of classes to be analyzed. It can be used together with `-p` and `-e`, which apply the pattern to the dependencies.

`-P` or `-profile`

Shows the profile containing a package.

`-R` or `--recursive`

Recursively traverses all run-time dependences. The `-R` option implies `-filter:none`. If `-p`, `-e`, or `-f` options are specified, only the matching dependences are analyzed.

`--no-recursive`

Do not recursively traverse dependences.

`-I` or `--inverse`

Analyzes the dependences per other given options and then finds all artifacts that directly and indirectly depend on the matching nodes. This is equivalent to the inverse of the compile-time view analysis and the print dependency summary. This option must be used with the `--require`, `--package`, or `--regex` options.

`--compile-time`

Analyzes the compile-time view of transitive dependencies, such as the compile-time view of the `-R` option. Analyzes the dependences per other specified options. If a dependency is found from a directory, a JAR file or a module, all classes in that containing archive are analyzed.

Example of Analyzing Dependencies

The following example demonstrates analyzing the dependencies of the `Notepad.jar` file.

Oracle Solaris, Linux, and OS X:

```
$ jdeps demo/jfc/Notepad/Notepad.jar
```

```
Notepad.jar -> java.base
```

```
Notepad.jar -> java.desktop
```

```
Notepad.jar -> java.logging
```

```
<unnamed> (Notepad.jar)
```

```
-> java.awt
```

```
-> java.awt.event
```

```
-> java.beans
```

```
-> java.io
```

```
-> java.lang
```

```
-> java.net
```

```
-> java.util
```

```
-> java.util.logging
```

```
-> javax.swing
```

```
-> javax.swing.border
```

```
-> javax.swing.event
```

```
-> javax.swing.text
```

```
-> javax.swing.tree
```

```
-> javax.swing.undo
```

Windows:

```
C:\Java\jdk1.9.0>jdeps demo\jfc\Notepad\Notepad.jar
```

```
Notepad.jar -> java.base
```

```
Notepad.jar -> java.desktop
```

```
Notepad.jar -> java.logging
```

```
<unnamed> (Notepad.jar)
```

```
-> java.awt
```

```
-> java.awt.event
```

```
-> java.beans
```

```
-> java.io
```

```
-> java.lang
```

```
-> java.net
```

```
-> java.util
```

```
-> java.util.logging
```

```
-> javax.swing
-> javax.swing.border
-> javax.swing.event
-> javax.swing.text
-> javax.swing.tree
-> javax.swing.undo
```

Example Using the --inverse Option

```
$ jdeps --inverse --require java.xml.bind
Inverse transitive dependences on [java.xml.bind]
java.xml.bind <- java.se.ee
java.xml.bind <- jdk.xml.ws
java.xml.bind <- java.xml.ws <- java.se.ee
java.xml.bind <- java.xml.ws <- jdk.xml.ws
java.xml.bind <- jdk.xml.bind <- jdk.xml.ws
```

Copyright © 1993, 2020, Oracle and/or its affiliates, 500 Oracle Parkway, Redwood Shores, CA 94065 USA.
All rights reserved. Use is subject to [license terms](#) and the [documentation redistribution policy](#).