



Linux Ubuntu 26.04 Manual Pages on command 'jar.1'

\$ man jar.1

JAR(1)

JDK Commands

JAR(1)

NAME

jar - create an archive for classes and resources, and manipulate or restore individual classes or resources from an archive

SYNOPSIS

jar [OPTION ...] [[--release VERSION] [-C dir] files] ...

DESCRIPTION

The jar command is a general-purpose archiving and compression tool, based on the ZIP and ZLIB compression formats. Initially, the jar command was designed to package Java applets (not supported since JDK 11) or applications; however, beginning with JDK 9, users can use the jar command to create modular JARs. For transportation and deployment, it's usually more convenient to package modules as modular JARs.

The syntax for the jar command resembles the syntax for the tar command. It has several main operation modes, defined by one of the mandatory operation arguments. Other arguments are either options that modify the behavior of the operation or are required to perform the operation.

When modules or the components of an application (files, images and sounds) are combined into a single archive, they can be downloaded by a Java agent (such as a browser) in a single HTTP transaction, rather than requiring a new connection for each piece. This dramatically improves download times. The jar command also compresses files, which further improves download time. The jar command also enables individual entries in a file to be signed so that their origin can be authenticated. A JAR file can be used as a class path entry, whether or not it's compressed.

An archive becomes a modular JAR when you include a module descriptor, `module-info.class`, in the root of the given directories or in the root of the `.jar` archive. The following operations described in Operation Modifiers Valid Only in Create and Update Modes are valid only when creating or updating a modular jar or updating an existing non-modular jar:

? `--module-version`

? `--hash-modules`

? `--module-path`

Note:

All mandatory or optional arguments for long options are also mandatory or optional for any corresponding short options.

MAIN OPERATION MODES

When using the `jar` command, you must specify the operation for it to perform. You specify the operation mode for the `jar` command by including the appropriate operation arguments described in this section. You can mix an operation argument with other one-letter options. Generally the operation argument is the first argument specified on the command line.

`-c` or `--create`

Creates the archive.

`-i` FILE or `--generate-index=FILE`

Generates index information for the specified JAR file. This option is deprecated and may be removed in a future release.

`-t` or `--list`

Lists the table of contents for the archive.

`-u` or `--update`

Updates an existing JAR file.

`-x` or `--extract`

Extracts the named (or all) files from the archive. If a file with the same name appears more than once in the archive, each copy will be extracted, with later copies overwriting (replacing) earlier copies unless `-k` is specified.

`-d` or `--describe-module`

Prints the module descriptor or automatic module name.

`--validate`

Validate the contents of the JAR file. See Integrity of a JAR File section below for more details.

OPERATION MODIFIERS VALID IN ANY MODE

You can use the following options to customize the actions of any operation mode included in the jar command.

-C DIR When used with the create operation mode, changes the specified directory and includes the files specified at the end of the command line.

jar [OPTION ...] [[--release VERSION] [-C dir] files]

When used with the extract operation mode, specifies the destination directory where the JAR file will be extracted. Unlike with the create operation mode, this option can be specified only once with the extract operation mode.

-f FILE or --file=FILE

Specifies the archive file name.

--release VERSION

Creates a multirelease JAR file. Places all files specified after the option into a versioned directory of the JAR file named META-INF/versions/VERSION/, where VERSION must be a positive integer whose value is 9 or greater.

At run time, where more than one version of a class exists in the JAR, the JDK will use the first one it finds, searching initially in the directory tree whose VERSION number matches the JDK's major version number. It will then look in directories with successively lower VERSION numbers, and finally look in the root of the JAR.

-v or --verbose

Sends or prints verbose output to standard output.

OPERATION MODIFIERS VALID ONLY IN CREATE AND UPDATE MODES

You can use the following options to customize the actions of the create and the update main operation modes:

-e CLASSNAME or --main-class=CLASSNAME

Specifies the application entry point for standalone applications bundled into a modular or executable modular JAR file.

-m FILE or --manifest=FILE

Includes the manifest information from the given manifest file.

-M or --no-manifest

Doesn't create a manifest file for the entries.

--module-version=VERSION

Specifies the module version, when creating or updating a modular JAR file, or updating

ing a non-modular JAR file.

`--hash-modules=PATTERN`

Computes and records the hashes of modules matched by the given pattern and that depend upon directly or indirectly on a modular JAR file being created or a non-modular JAR file being updated.

`-p` or `--module-path`

Specifies the location of module dependence for generating the hash.

`@file` Reads `jar` options and file names from a text file as if they were supplied on the command line

OPERATION MODIFIERS VALID ONLY IN CREATE, UPDATE, AND GENERATE-INDEX MODES

You can use the following options to customize the actions of the create (`-c` or `--create`) the update (`-u` or `--update`) and the generate-index (`-i` or `--generate-index=FILE`) main operation modes:

`-O` or `--no-compress`

Stores without using ZIP compression.

`--date=TIMESTAMP`

The timestamp in ISO-8601 extended offset date-time with optional time-zone format, to use for the timestamp of the entries, e.g. "2022-02-12T12:30:00-05:00".

OPERATION MODIFIERS VALID ONLY IN EXTRACT MODE

`--dir DIR`

Directory into which the JAR file will be extracted.

`-k` or `--keep-old-files`

Do not overwrite existing files. If a Jar file entry with the same name exists in the target directory, the existing file will not be overwritten. As a result, if a file appears more than once in an archive, later copies will not overwrite earlier copies. Also note that some file system can be case insensitive.

OTHER OPTIONS

The following options are recognized by the `jar` command and not used with operation modes:

`-h` or `--help[:compat]`

Displays the command-line help for the `jar` command or optionally the compatibility help.

`--help-extra`

Displays help on extra options.

--version

Prints the program version.

INTEGRITY OF A JAR FILE

As a JAR file is based on ZIP format, it is possible to create a JAR file using tools other than the jar command. The --validate option may be used to perform the following integrity checks against a JAR file:

? That there are no duplicate Zip entry file names

? Verify that the Zip entry file name:

? is not an absolute path

? the file name is not '.' or '..'

? does not contain a backslash, '\'

? does not contain a drive letter

? path element does not include '.' or '..'

? The API exported by a multi-release jar archive is consistent across all different release versions.

The jar tool exits with a status of 0 if there were no integrity issues encountered and >0 if an error/warning occurred.

When an integrity issue is reported, it will often require that the JAR file is re-created by the original source of the JAR file.

EXAMPLES OF JAR COMMAND SYNTAX

? Create an archive, classes.jar, that contains two class files, Foo.class and Bar.class.

```
jar --create --file classes.jar Foo.class Bar.class
```

? Create an archive, classes.jar, that contains two class files, Foo.class and Bar.class setting the last modified date and time to 2021 Jan 6 12:36:00.

```
jar --create --date="2021-01-06T14:36:00+02:00" --file=classes.jar Foo.class  
Bar.class
```

? Create an archive, classes.jar, by using an existing manifest, mymanifest, that contains all of the files in the directory foo/.

```
jar --create --file classes.jar --manifest mymanifest -C foo/
```

? Create a modular JAR archive,foo.jar, where the module descriptor is located in classes/module-info.class.

```
jar --create --file foo.jar --main-class com.foo.Main --module-version 1.0 -C  
foo/classes resources
```

? Update an existing non-modular JAR, foo.jar, to a modular JAR file.

```
jar --update --file foo.jar --main-class com.foo.Main --module-version 1.0 -C
foo/module-info.class
```

? Create a versioned or multi-release JAR, foo.jar, that places the files in the classes di?

rectory at the root of the JAR, and the files in the classes-10 directory in the
META-INF/versions/10 directory of the JAR.

In this example, the classes/com/foo directory contains two classes, com.foo.Hello (the
entry point class) and com.foo.NameProvider, both compiled for JDK 8. The
classes-10/com/foo directory contains a different version of the com.foo.NameProvider
class, this one containing JDK 10 specific code and compiled for JDK 10.

Given this setup, create a multirelease JAR file foo.jar by running the following command
from the directory containing the directories classes and classes-10 .

```
jar --create --file foo.jar --main-class com.foo.Hello -C classes . --release 10 -C
classes-10 .
```

The JAR file foo.jar now contains:

```
% jar -tf foo.jar
META-INF/
META-INF/MANIFEST.MF
com/
com/foo/
com/foo/Hello.class
com/foo/NameProvider.class
META-INF/versions/10/com/
META-INF/versions/10/com/foo/
META-INF/versions/10/com/foo/NameProvider.class
```

As well as other information, the file META-INF/MANIFEST.MF, will contain the following
lines to indicate that this is a multirelease JAR file with an entry point of
com.foo.Hello.

```
...
Main-Class: com.foo.Hello
Multi-Release: true
```

Assuming that the com.foo.Hello class calls a method on the com.foo.NameProvider class,
running the program using JDK 10 will ensure that the com.foo.NameProvider class is the

one in META-INF/versions/10/com/foo/. Running the program using JDK 8 will ensure that the com.foo.NameProvider class is the one at the root of the JAR, in com/foo.

? Create an archive, my.jar, by reading options and lists of class files from the file classes.list.

Note:

To shorten or simplify the jar command, you can provide an arg file that lists the files to include in the JAR file and pass it to the jar command with the at sign (@) as a pre-fix.

```
jar --create --file my.jar @classes.list
```

If one or more entries in the arg file cannot be found then the jar command fails without creating the JAR file.

? Extract the JAR file foo.jar to /tmp/bar/ directory:

```
jar -xf foo.jar -C /tmp/bar/
```

Alternatively, you can also do:

```
jar --extract --file foo.jar --dir /tmp/bar/
```