NAME

jar − Manipulates Java Archive (JAR) files.

SYNOPSIS

Create JAR file

jar c[**efmMnv0**] [*entrypoint*] [*jarfile*] [*manifest*] [**−C** *dir*] *file* ... [−J*option* ...] [@*arg*−*file* ...]

Update JAR file

jar u[**efmMnv0**] [*entrypoint*] [*jarfile*] [*manifest*] [**−C** *dir*] *file* ... [−J*option* ...] [@*arg*−*file* ...]

Extract JAR file

jar x[**vf**] [*jarfile*] *file* ... [−J*option* ...] [@*arg*−*file* ...]

List Contents of JAR file

jar t[**vf**] [*jarfile*] *file* ... [−J*option* ...] [@*arg*−*file* ...]

Add Index to JAR file

jar i *jarfile* [−J*option* ...] [@*arg*−*file* ...]

DESCRIPTION

The **jar** command is a general-purpose archiving and compression tool, based on ZIP and the ZLIB compression format. However, the **jar** command was designed mainly to package Java applets or applications into a single archive. When the components of an applet or 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 allows individual entries in a file to be signed by the applet author so that their origin can be authenticated. A JAR file can be used as a class path entry, whether or not it is compressed.

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

OPERATION ARGUMENTS

When using the **jar** command, you have to select an operation to be performed by specifying one of the following operation arguments. You can mix them up with other one-letter options on the command line, but usually the operation argument is the first argument specified.

c Create a new JAR archive.

i Generate index information for a JAR archive.

t List the contents of a JAR archive.

u Update a JAR archive.

x Extract files from a JAR archive.

OPTIONS

Use the following options to customize how the JAR file is created, updated, extracted, or viewed:

e Sets the class specified by the *entrypoint* operand to be the entry point for a standalone Java application bundled into an executable JAR file. The use of this option creates or overrides the **Main-Class** attribute value in the manifest file. The **e** option can be used when creating (**c**) or updating (**u**) the JAR file.

For example, the following command creates the **Main.jar** archive with the **Main.class** file where the **Main-Clas**s attribute value in the manifest is set to **Main**:

jar cfe Main.jar Main Main.class

The Java Runtime Environment (JRE) can directly call this application by running the following command:

java –jar Main.jar

If the entry point class name is in a package, then it could use either the dot (.) or slash (/) as the delimiter. For example, if **Main.class** is in a package called **mydir**, then the entry point can be specified in one of the following ways:

jar –cfe Main.jar mydir/Main mydir/Main.class
jar –cfe Main.jar mydir.Main mydir/Main.class

Note

Specifying both **m** and **e** options together when a particular manifest also contains the **Main-Class** attribute results in an ambiguous **Main-Class** specification. The ambiguity leads to an error and the **jar** command creation or update operation is terminated.

f Sets the file specified by the *jarfile* operand to be the name of the JAR file that is created (**c**), updated (**u**), extracted (**x**) from, or viewed (**t**). Omitting the **f** option and the *jarfile* operand instructs the **jar** command to accept the JAR file name from **stdin** (for **x** and **t**) or send the JAR file to **stdout** (for **c** and **u**).

m Includes names and values of attributes from the file specified by the **manifest** operand in the manifest file of the **jar** command (located in the archive at **META-INF/MANIFEST.MF**). The **jar** command adds the attribute's name and value to the JAR file unless an entry already exists with the same name, in which case the **jar** command updates the value of the attribute. The **m** option can be used when creating (**c**) or updating (**u**) the JAR file.

You can add special-purpose name-value attribute pairs to the manifest that are not contained in the default manifest file. For example, you can add attributes that specify vendor information, release information, package sealing, or to make JAR-bundled applications executable. For examples of using the **m** option, see Packaging Programs at http://docs.oracle.com/javase/tutorial/deployment/jar/index.html

M Does not create a manifest file entry (for **c** and **u**), or delete a manifest file entry when one exists (for **u**). The **M** option can be used when creating (**c**) or updating (**u**) the JAR file.

n When creating (**c**) a JAR file, this option normalizes the archive so that the content is not affected by the packing and unpacking operations of the pack200(1) command. Without this normalization, the signature of a signed JAR can become invalid.

v Generates verbose output to standard output. See Examples.

0 (Zero) Creates (**c**) or updates (**u**) the JAR file without using ZIP compression.

-C *dir*

When creating (**c**) or updating (**u**) a JAR file, this option temporarily changes the directory while processing files specified by the *file* operands. Its operation is intended to be similar to the **-C** option of the UNIX **tar** utility.For example, the following command changes to the **classes** directory and adds the **Bar.class** file from that directory to **my.jar**:

jar uf my.jar −C classes Bar.class

The following command changes to the **classes** directory and adds to **my.jar** all files within the classes directory (without creating a **classes** directory in the JAR file), then changes back to the original directory before changing to the **bin** directory to add **Xyz.class** to **my.jar**.

jar uf my.jar −C classes . −C bin Xyz.class

If **classes** contained files **bar1** and **bar2**, then the JAR file will contain the following after running the previous command:

% jar tf my.jar
META−INF/
META−INF/MANIFEST.MF
bar1
bar2
Xyz.class

-J*option*

Sets the specified JVM option to be used when the JRE runs the JAR file. JVM options are described on the reference page for the java(1) command. For example, **-J-Xms48m** sets the startup memory to 48 MB.

OPERANDS

The following operands are recognized by the **jar** command.

file When creating (**c**) or updating (**u**) a JAR file, the *file* operand defines the path and name of the file or directory that should be added to the archive. When extracting (**x**) or listing the contents (**t**) of a JAR file, the *file* operand defines the path and name of the file to be extrated or listed. At least one valid file or directory must be specified. Separate multiple *file* operands with spaces. If the *entrypoint*, *jarfile*, or *manifest* operands are used, the *file* operands must be specified after them.

entrypoint

When creating (**c**) or updating (**u**) a JAR file, the *entrypoint* operand defines the name of the class that should be the entry point for a standalone Java application bundled into an executable JAR file. The *entrypoint* operand must be specified if the **e** option is present.

jarfile Defines the name of the file to be created (**c**), updated (**u**), extracted (**x**), or viewed (**t**). The *jarfile* operand must be specified if the **f** option is present. Omitting the **f** option and the *jarfile* operand instructs the **jar** command to accept the JAR file name from **stdin** (for **x** and **t**) or send the JAR file to **stdout** (for **c** and **u**).

When indexing (**i**) a JAR file, specify the *jarfile* operand without the **f** option.

manifest

When creating (**c**) or updating (**u**) a JAR file, the *manifest* operand defines the preexisting manifest files with names and values of attributes to be included in **MANIFEST.MF** in the JAR file. The *manifest* operand must be specified if the **f** option is present.

@arg-file

To shorten or simplify the **jar** command, you can specify arguments in a separate text file and pass it to the **jar** command with the at sign (@) as a prefix. When the **jar** command encounters an argument beginning with the at sign, it expands the contents of that file into the argument list.

An argument file can include options and arguments of the **jar** command (except the **-J** options, because they are passed to the launcher, which does not support argument files). The arguments within a file can be separated by spaces or newline characters. File names within an argument file are relative to the current directory from which you run the **jar** command, not relative to the location of the argument file. Wild cards, such as the asterisk (*), that might otherwise be expanded by the operating system shell, are not expanded.

The following example, shows how to create a **classes.list** file with names of files from the current directory output by the **find** command:

find . –name '*.class' –print > classes.list

You can then execute the **jar** command and pass the **classes.list** file to it using the *@arg-file* syntax:

jar cf my.jar @classes.list

An argument file can be specified with a path, but any file names inside the argument file that have relative paths are relative to the current working directory of the **jar** command, not to the path passed in, for example:

jar @dir/classes.list

NOTES

The **e**, **f**, and **m** options must appear in the same order on the command line as the *entrypoint*, *jarfile*, and *manifest* operands, for example:

jar cmef myManifestFile MyMainClass myFile.jar *.class

EXAMPLES

Example 1 Adding All Files From the Current Directory With Verbose Output

% ls

```
1.au        Animator.class    monkey.jpg
2.au        Wave.class        spacemusic.au
3.au        at_work.gif
% jar cvf bundle.jar *
added manifest
adding: 1.au(in = 2324) (out= 67)(deflated 97%)
adding: 2.au(in = 6970) (out= 90)(deflated 98%)
adding: 3.au(in = 11616) (out= 108)(deflated 99%)
adding: Animator.class(in = 2266) (out= 66)(deflated 97%)
adding: Wave.class(in = 3778) (out= 81)(deflated 97%)
adding: at_work.gif(in = 6621) (out= 89)(deflated 98%)
adding: monkey.jpg(in = 7667) (out= 91)(deflated 98%)
adding: spacemusic.au(in = 3079) (out= 73)(deflated 97%)
```

Example 2 Adding Files From Subdirectories

```
% ls −F
audio/ classes/ images/
% jar cvf bundle.jar audio classes images
added manifest
adding: audio/(in = 0) (out= 0)(stored 0%)
adding: audio/1.au(in = 2324) (out= 67)(deflated 97%)
adding: audio/2.au(in = 6970) (out= 90)(deflated 98%)
adding: audio/3.au(in = 11616) (out= 108)(deflated 99%)
adding: audio/spacemusic.au(in = 3079) (out= 73)(deflated 97%)
adding: classes/(in = 0) (out= 0)(stored 0%)
adding: classes/Animator.class(in = 2266) (out= 66)(deflated 97%)
adding: classes/Wave.class(in = 3778) (out= 81)(deflated 97%)
adding: images/(in = 0) (out= 0)(stored 0%)
adding: images/monkey.jpg(in = 7667) (out= 91)(deflated 98%)
adding: images/at_work.gif(in = 6621) (out= 89)(deflated 98%)
% ls −F
audio/ bundle.jar classes/ images/
```

Example 3 Listing the Contents of JAR

```
% jar tf bundle.jar


META−INF/
META−INF/MANIFEST.MF
audio/1.au
audio/2.au
audio/3.au
audio/spacemusic.au
classes/Animator.class
classes/Wave.class
images/monkey.jpg
images/at_work.gif
```

Example 4 Adding an Index

Use the **i** option when you split the interdependent classes for a stock trade application into three JAR files: **main.jar**, **buy.jar**, and **sell.jar**. If you specify the **Class-Path** attribute in the **main.jar** manifest, then you can use the **i** option to speed up the class loading time for your application:

Class−Path: buy.jar sell.jar
jar i main.jar

An **INDEX.LIST** file is inserted to the **META-INF** directory. This enables the application class loader to download the specified JAR files when it is searching for classes or resources.

The application class loader uses the information stored in this file for efficient class loading. To copy directories, first compress files in **dir1** to **stdout**, then pipeline and extract from **stdin** to **dir2** (omitting the **-f** option from both **jar** commands):

(cd dir1; jar c .) | (cd dir2; jar x)

SEE ALSO

- pack200(1).
- The JAR section of The Java Tutorials at http://docs.oracle.com/javase/tutorial/deployment/jar/index.html