Apache Ant is a Java-based build tool.

**Getting Apache Ant**

To get up and running with the binary distribution of Ant quickly, follow these steps:

1. Make sure you have a Java environment installed.

(For the current version of Ant (1.10), you will also need a JDK installed on your system, version 8 or later required. The more up-to-date the version of Java, the more Ant tasks you get.)

Note: If a JDK is not present, only the runtime (JRE), then many tasks will not work.

Note: Ant 1.9.\* works with JDK 1.5 and higher, Ant 1.8.\* works with JDK 1.4 and higher, Ant 1.7.\* works with JDK 1.3 and higher, Ant 1.6.\* works with JDK 1.2 and higher, Ant 1.2 to Ant 1.5.\* work with JDK 1.1 and higher.)

1. Download Ant.

The latest stable version of Ant is available from the Ant web page <https://ant.apache.org/>

The binary distribution of Ant is available as 3 different archives

.zip—Recommended compression format for Windows, can also be used on other platforms. Supported by many programs and some operating systems natively.

.tar.gz—Using the tar program to gather files together, and gzip to compress and uncompress.

.tar.bz2—Using the tar program to gather files together, and bzip2 to compress and uncompress.

Choose the format that is best supported for your platform.

1. Uncompress the downloaded file into a directory.
2. Set environmental variables: JAVA\_HOME to your Java environment, ANT\_HOME to the directory you uncompressed Ant to, and %ANT\_HOME%/bin (Windows) to your PATH.

The binary distribution of Ant consists of the following directory layout:

ant

+--- README, LICENSE, fetch.xml, other text files. //basic information

+--- bin // contains launcher scripts

+--- lib // contains Ant JARs plus necessary dependencies

+--- manual // Ant documentation (a must read ;-)

+--- etc // contains xsl goodies to:

// - create an enhanced report from xml output of various tasks.

// - migrate your build files and get rid of 'deprecated' warning

// - ... and more ;-)

Only the bin and lib directories are required to run Ant.

To install Ant, choose a directory and copy the distribution files there. This directory will be known as ANT\_HOME.

**Check Installation**

You can check the basic installation with opening a new shell and typing ant. You should get a message like this

Buildfile: build.xml does not exist!

Build failed

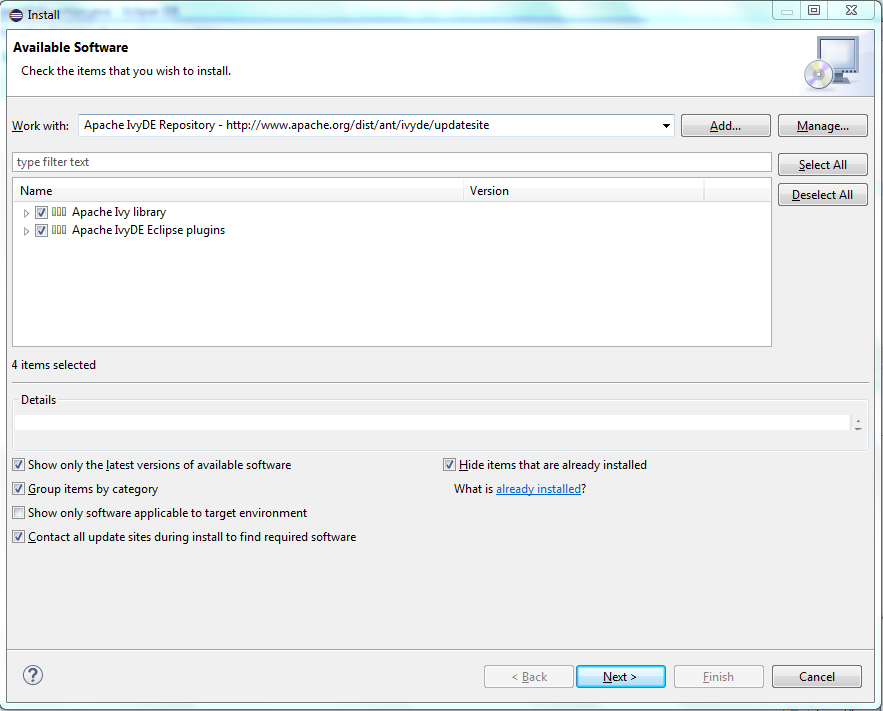
So Ant works. This message is there because you need to write a buildfile for your project. With a ant -version you should get an output like

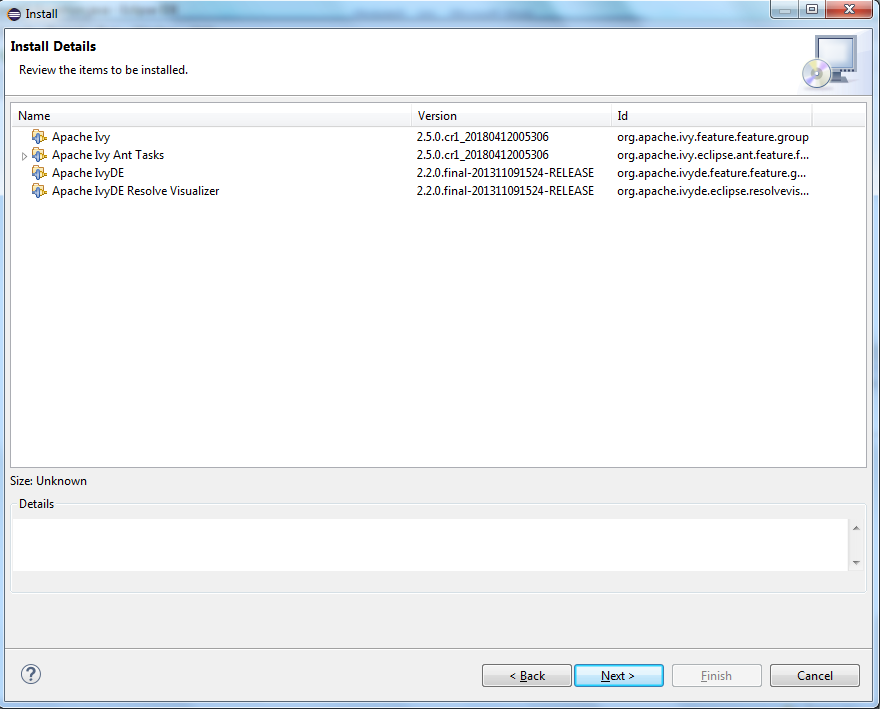
Apache Ant(TM) version 1.9.2 compiled on July 8 2013

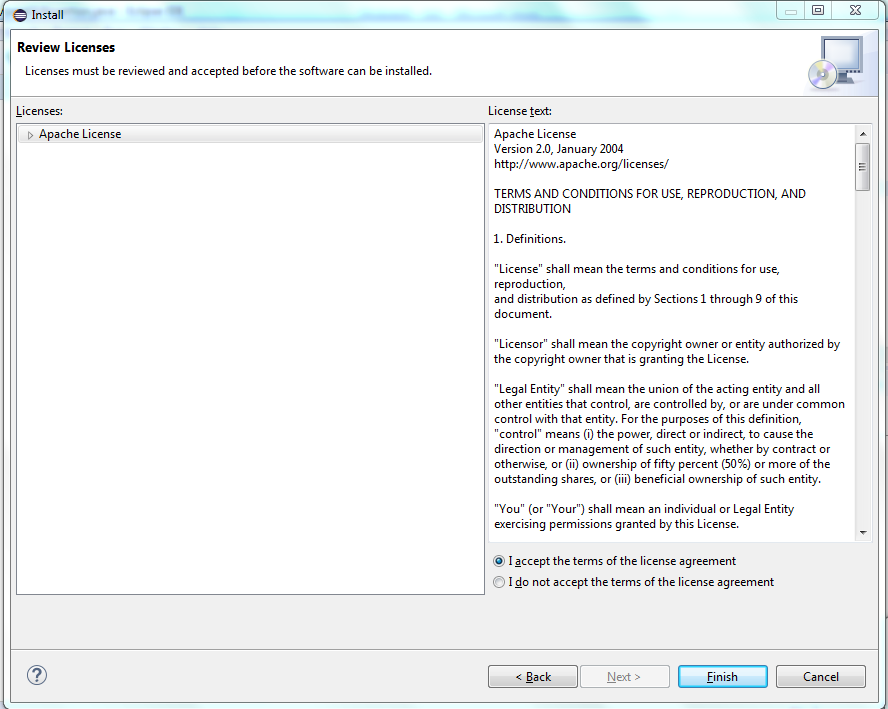
If this does not work, ensure your environment variables are set right. E.g., on Windows, they must resolve to:

* required: %ANT\_HOME%\bin\ant.bat
* optional: %JAVA\_HOME%\bin\java.exe
* required: %PATH%=...*maybe-other-entries*...;%ANT\_HOME%\bin;...*maybe-other-entries*...

ANT\_HOME is used by the launcher script for finding the libraries. JAVA\_HOME is used by the launcher for finding the JDK/JRE to use. (JDK is recommended as some tasks require the Java tools.) If not set, the launcher tries to find one via the %PATH% environment variable. PATH is set for user convenience. With that set you can just start ant instead of always typing the/complete/path/to/your/ant/installation/bin/ant.

****

****

****

**Bundled in IDEs**

All the main Java IDEs ship with Ant, products such as Eclipse, NetBeans and IntelliJ IDEA. If you install Ant this way you usually get the most recent release of Ant at the time the IDE was released. Some of the IDEs (Eclipse and NetBeans in particular) ship with extra tasks that only work if IDE-specific tools are on Ant's path. To use these on command-line versions of Ant, the relevant JARs need to be added to the command-line Ant as extra libraries/tasks. Note that if it is an IDE task or extension that is not behaving, the Ant team is unable to field bug reports. Try the IDE mailing lists first, who will cross-file bugs if appropriate.

IDEs can invariably be pointed at different Ant installations. This lets developers upgrade to a new release of Ant, and eliminate inconsistencies between command-line and IDE Ant.

**Bundled in Java Applications**

Many Java applications, most particularly application servers, ship with a version of Ant. These are primarily for internal use by the application, using the Java APIs to delegate tasks such as JSP page compilation to the Ant runtime. Such distributions are usually unsupported by everyone. Particularly troublesome are those products that not only ship with their own Ant release, they add their own version of ANT.BAT or ant.sh to the PATH. If Ant starts behaving weirdly after installing something, try the [diagnostics](https://ant.apache.org/manual/install.html#diagnostics) advice.

**Source Distribution**

If you prefer the source distribution, you can download the source for the latest Ant release from <https://ant.apache.org/srcdownload.cgi>.

If you prefer the leading-edge code, you can access the code as it is being developed via Git. The Ant website has details on [accessing Git](https://ant.apache.org/git.html). All bug fixes will go in against the HEAD of the source tree, and the first response to many bug reports will be "have you tried the latest version". Don't be afraid to download and build a prerelease distribution, as everything other than new features are usually stable.

See the section [Building Ant](https://ant.apache.org/manual/install.html#buildingant) on how to build Ant from the source code. You can also access the [Ant Git repository](https://gitbox.apache.org/repos/asf?p=ant.git;a=summary) on-line.

**Archive Download Area Layout**

Older versions of Ant are available in the archives at <https://archive.apache.org/dist/ant/>. The files are organized as follows.

|  |  |
| --- | --- |
| **Filename or Path** | **Description** |
| KEYS | PGP keyfile. It contains the PGP keys of Ant developers so you can 'trust' the distribution. |
| RELEASE-NOTES-{version}.html | Release notes of the given version in HTML format. When upgrading your Ant installation you should have a look at the *Changes that could break older environments* section. |
| ant-current-bin.zip | ZIP archive containing the compiled version of Ant in the last released version. It is recommended that you do not download the latest version this way, as the standard way of downloading described above will redirect you to a mirror closer to you, thus making the download faster for you and reducing the load on Apache servers. |
| ant-current-src.zip | ZIP archive containing the sources of Ant. If you have this you can compile Ant. If you do not have the *required* dependencies, the classes depending on them are just not built. Again, it is preferred to use the standard way of getting the source package described above to make your download quicker and to reduce the load on Apache servers. |
| ant-current-\*.asc | Security file for checking the correctness of the zip file. This one is the [PGP](https://en.wikipedia.org/wiki/Pretty_Good_Privacy) signature. |
| ant-current-\*.md5 | Security file for checking the correctness of the zip file. This one is the [MD5](https://en.wikipedia.org/wiki/Md5) checksum. |
| ant-current-\*.sha1 | Security file for checking the correctness of the zip file. This one is the [SHA1](https://en.wikipedia.org/wiki/SHA-1) checksum. |
| ant-current-\*.sha512 | Security file for checking the correctness of the zip file. This one is the [SHA512](https://en.wikipedia.org/wiki/SHA-2) checksum. |
| antlibs/ | This directory holds the Antlibs that are made of available by the Apache Ant project. Antlibs are bundles of Ant tasks that are not delivered as part of the Ant core but are available as optional downloads. |
| binaries/ | The binaries directory holds specific Ant releases bundled in both ZIP and tar.gz archive formats. The named releases are in contrast to the ant-current-bin.zip file in the parent directory, which is always guaranteed to be the most current release of Ant. |
| common/ | The common directory holds various files, such as the Apache License file that Ant is licensed under, that people may wish to examine without having to download the whole Ant distribution. |
| source/ | The source directory holds the source code for specific Ant releases bundled in both ZIP and tar.gz archive formats. The named releases are in contrast to the ant-current-src.zip file in the parent directory, which is always guaranteed to hold the source code for the most current release of Ant. |

**System Requirements**

Ant has been used successfully on many platforms, including Linux, commercial flavours of Unix such as Solaris and HP-UX, macOS, Windows NT descendants, OS/2 Warp, Novell Netware 6, OpenVMS. The platforms used most for development are, in no particular order, Linux, macOS, Microsoft Windows and Unix; these are therefore that platforms that tend to work best. *Since Ant 1.7*, Windows 9x is no longer supported.

For the current version of Ant (1.10), you will also need a JDK installed on your system, version 8 or later required. The more up-to-date the version of Java, the more Ant tasks you get.

**Note**: If a JDK is not present, only the runtime (JRE), then many tasks will not work.

**Note**: Ant 1.9.\* works with JDK 1.5 and higher, Ant 1.8.\* works with JDK 1.4 and higher, Ant 1.7.\* works with JDK 1.3 and higher, Ant 1.6.\* works with JDK 1.2 and higher, Ant 1.2 to Ant 1.5.\* work with JDK 1.1 and higher.

**Open Source Java Runtimes**

The Ant team strongly supports users running Ant on [OpenJDK](http://openjdk.java.net/) and other open source Java runtimes, and so strives to have a product that works well on those platforms.

**Installing Ant**

The binary distribution of Ant consists of the following directory layout:

ant

+--- README, LICENSE, fetch.xml, other text files. //basic information

+--- bin // contains launcher scripts

|

+--- lib // contains Ant JARs plus necessary dependencies

|

+--- manual // Ant documentation (a must read ;-)

|

+--- etc // contains xsl goodies to:

// - create an enhanced report from xml output of various tasks.

// - migrate your build files and get rid of 'deprecated' warning

// - ... and more ;-)

Only the bin and lib directories are required to run Ant.

To install Ant, choose a directory and copy the distribution files there. This directory will be known as ANT\_HOME.

**Windows 95, Windows 98 & Windows ME Note**

*Note that current releases of Ant no longer support these systems. If you are using an older version of Ant, however, the script used to launch Ant will have problems if ANT\_HOME is a long filename (i.e. a filename which is not of the format known as "8.3"). This is due to limitations in the OS's handling of the "for" batch file statement. It is recommended, therefore, that Ant be installed in a* ***short****, 8.3 path, such as C:\Ant.*

On these systems you will also need to configure more environment space to cater for the environment variables used in the Ant launch script. To do this, you will need to add or update the following line in the config.sys file

shell=c:\command.com c:\ /p /e:32768

**Setup**

Before you can run Ant there is some additional setup you will need to do unless you are installing the [RPM Version from jpackage.org](https://ant.apache.org/manual/install.html#jpackage):

* Add the bin directory to your path.
* Set the ANT\_HOME environment variable to the directory where you installed Ant. On some operating systems, Ant's startup scripts can guess ANT\_HOME (Unix dialects and Windows NT descendants), but it is better to not rely on this behavior.
* Optionally, set the JAVA\_HOME environment variable (see the [Advanced](https://ant.apache.org/manual/install.html#advanced) section below). This should be set to the directory where your JDK is installed.

Operating System-specific instructions for doing this from the command line are in the [Windows](https://ant.apache.org/manual/install.html#windows), [Linux/Unix (bash)](https://ant.apache.org/manual/install.html#bash), and [Linux/Unix (csh)](https://ant.apache.org/manual/install.html#tcshcsh) sections. Note that using this method, the settings will only be valid for the command line session you run them in.

**Note**: Do not install Ant's ant.jar file into the lib/ext directory of the JDK/JRE. Ant is an application, whilst the extension directory is intended for JDK extensions. In particular there are security restrictions on the classes which may be loaded by an extension.

**Windows Note**

The ant.bat script makes use of three environment variables—ANT\_HOME, CLASSPATH and JAVA\_HOME. **Ensure** that ANT\_HOME and JAVA\_HOME variables are set, and that they do **not** have quotes (either ' or ") and they do **not** end with \ or with /. CLASSPATH should be unset or empty.

**Check Installation**

You can check the basic installation with opening a new shell and typing ant. You should get a message like this

Buildfile: build.xml does not exist!

Build failed

So Ant works. This message is there because you need to write a buildfile for your project. With a ant -version you should get an output like

Apache Ant(TM) version 1.9.2 compiled on July 8 2013

If this does not work, ensure your environment variables are set right. E.g., on Windows, they must resolve to:

* required: %ANT\_HOME%\bin\ant.bat
* optional: %JAVA\_HOME%\bin\java.exe
* required: %PATH%=...*maybe-other-entries*...;%ANT\_HOME%\bin;...*maybe-other-entries*...

ANT\_HOME is used by the launcher script for finding the libraries. JAVA\_HOME is used by the launcher for finding the JDK/JRE to use. (JDK is recommended as some tasks require the Java tools.) If not set, the launcher tries to find one via the %PATH% environment variable. PATH is set for user convenience. With that set you can just start ant instead of always typing the/complete/path/to/your/ant/installation/bin/ant.

**Optional Tasks**

Ant supports a number of optional tasks. An optional task is a task which typically requires an external library to function. The optional tasks are packaged together with the core Ant tasks.

The external libraries required by each of the optional tasks is detailed in the [Library Dependencies](https://ant.apache.org/manual/install.html#librarydependencies) section. These external libraries must be added to Ant's classpath, in any of the following ways:

* In *ANT\_HOME*/lib. This makes the JAR files available to all Ant users and builds.
* In ${user.home}/.ant/lib (*since Ant 1.6*). This allows different users to add new libraries to Ant. All JAR files added to this directory are available to command-line Ant.
* On the command line with a -lib parameter. This lets you add new JAR files on a case-by-case basis.
* In the CLASSPATH environment variable. Avoid this; it makes the JAR files visible to *all* Java applications, and causes no end of support calls. See [below](https://ant.apache.org/manual/install.html#classpath) for details.
* In some <classpath> accepted by the task itself. *Since Ant 1.7.0*, you can run the <junit> task without junit.jar in Ant's own classpath, so long as it is included (along with your program and tests) in the classpath passed when running the task.

Where possible, this option is generally to be preferred, as the Ant script itself can determine the best path to load the library from: via relative path from the basedir (if you keep the library under version control with your project), according to Ant properties, environment variables, Ivy downloads, whatever you like.

If you are using the binary distribution of Ant, or if you are working from source code, you can easily gather most of the dependencies and install them for use with your Ant tasks. In your ANT\_HOME directory you should see a file called fetch.xml. This is an Ant script that you can run to install almost all the dependencies that the optional Ant tasks need.

To do so, change to the ANT\_HOME directory and execute the command:

ant -f fetch.xml -Ddest=*[option]*

where option is one of the following, as described above:

* system—store in Ant's lib directory (*Recommended*)
* user—store in the user's home directory
* optional—store in Ant's source code lib/optional directory, used when building Ant source code

You may also need to set proxy settings. See the [Proxy Settings](https://ant.apache.org/manual/install.html#proxy) section for details.

Note that not all dependencies are gathered using fetch.xml. Tasks that depend on commercial software, in particular, will require you to have the commercial software installed in order to be used.

The Apache Ant Project also provides additional tasks and types that are available as separately downloaded Ant Libraries. You can see the the list of available Antlibs at the [Ant Libraries](https://ant.apache.org/antlibs/proper.html) page.

You can also find tasks and types provided by third-party projects at the [External Tools and Tasks](https://ant.apache.org/external.html) page.

IDEs have different ways of adding external JAR files and third-party tasks to Ant. Usually it is done by some configuration dialog. Sometimes JAR files added to a project are automatically added to Ant's classpath.

**The CLASSPATH Environment Variable**

The CLASSPATH environment variable is a source of many Ant support queries. As the round trip time for diagnosis on the Ant user mailing list can be slow, and because filing bug reports complaining about 'ant.bat' not working will be rejected by the developers as WORKSFORME "this is a configuration problem, not a bug", you can save yourself a lot of time and frustration by following some simple steps.

1. Do not ever set CLASSPATH. Ant does not need it, it only causes confusion and breaks things.
2. If you ignore the previous rule, do not ever, ever, put quotes in the CLASSPATH, even if there is a space in a directory. This will break Ant, and it is not needed.
3. If you ignore the first rule, do not ever, ever, have a trailing backslash in a CLASSPATH, as it breaks Ant's ability to quote the string. Again, this is not needed for the correct operation of the CLASSPATH environment variable, even if a DOS directory is to be added to the path.
4. You can stop Ant using the CLASSPATH environment variable by setting the -noclasspath option on the command line. This is an easy way to test for classpath-related problems.

The usual symptom of CLASSPATH problems is that Ant will not run with some error about not being able to find org.apache.tools.ant.launch.Launcher, or, if you have got the quotes/backslashes wrong, some very weird Java startup error. To see if this is the case, run ant -noclasspath or unset the CLASSPATH environment variable.

You can also make your Ant script reject this environment variable just by placing the following at the top of the script (or in an init target):

<property environment="env."/>

<property name="env.CLASSPATH" value=""/>

<fail message="Unset $CLASSPATH / %CLASSPATH% before running Ant!">

<condition>

<not>

<equals arg1="${env.CLASSPATH}" arg2=""/>

</not>

</condition>

</fail>

**Proxy Configuration**

Many Ant built-in and third-party tasks use network connections to retrieve files from HTTP servers. If you are behind a firewall with a proxy server, then Ant needs to be configured with the proxy. Here are the different ways to do this.

* **With Java 5 or above**

When you run Ant on Java 5 or above, you could try to use the automatic proxy setup mechanism with -autoproxy.

* **With explicit JVM properties.**

These are documented in [Java's Networking Properties](https://docs.oracle.com/javase/8/docs/technotes/guides/net/properties.html), and control the proxy behaviour of the entire JVM. To set them in Ant, declare them in the ANT\_OPTS environment variable. This is the best option for a non-mobile system. For a laptop, you have to change these settings as you roam. To set ANT\_OPTS:

For csh/tcsh:

setenv ANT\_OPTS "-Dhttp.proxyHost=proxy -Dhttp.proxyPort=8080"

For bash:

export ANT\_OPTS="-Dhttp.proxyHost=proxy -Dhttp.proxyPort=8080"

For Windows, set the environment variable in the appropriate dialog box and open a new console or, by hand

set ANT\_OPTS = -Dhttp.proxyHost=proxy -Dhttp.proxyPort=8080

* **In the build file itself**

If you are writing a build file that is always to be used behind the firewall, the [setproxy](https://ant.apache.org/manual/Tasks/setproxy.html) task lets you configure the proxy (which it does by setting the JVM properties). If you do this, we strongly recommend using Ant properties to define the proxy host, port, etc, so that individuals can override the defaults.

The Ant team acknowledges that this is unsatisfactory. Until the JVM automatic proxy setup works properly everywhere, explicit JVM options via ANT\_ARGS are probably the best solution. Setting properties on Ant's command line do not work, because those are *Ant properties* being set, not JVM options. This means the following does not set up the command line:

ant -Dhttp.proxyHost=proxy -Dhttp.proxyPort=81

All it does is set up two Ant properties.

One other trouble spot with proxies is with authenticating proxies. Ant cannot go beyond what the JVM does here, and as it is very hard to remotely diagnose, test and fix proxy-related problems, users who work behind a secure proxy will have to spend much time configuring the JVM properties until they are happy.

**Windows and OS/2**

Assume Ant is installed in c:\ant\. The following sets up the environment:

set ANT\_HOME=c:\ant

set JAVA\_HOME=c:\jdk1.7.0\_51

set PATH=%PATH%;%ANT\_HOME%\bin

**Linux/Unix (bash)**

Assume Ant is installed in /usr/local/ant. The following sets up the environment:

export ANT\_HOME=/usr/local/ant

export JAVA\_HOME=/usr/local/jdk1.7.0\_51

export PATH=${PATH}:${ANT\_HOME}/bin

**Linux/Unix (csh)**

setenv ANT\_HOME /usr/local/ant

setenv JAVA\_HOME /usr/local/jdk/jdk1.7.0\_51

set path=( $path $ANT\_HOME/bin )

Having a symbolic link set up to point to the JVM/JDK version makes updates more seamless.

**RPM version from jpackage.org**

The [JPackage project](http://www.jpackage.org) distributes an RPM version of Ant. With this version, it is not necessary to set JAVA\_HOME or ANT\_HOME environment variables and the RPM installer will correctly place the ant executable on your path.

**Note**: *Since Ant 1.7.0*, if the ANT\_HOME environment variable is set, the JPackage distribution will be ignored.

Optional JARs for the JPackage version are handled in two ways. The easiest, and best way is to get these external libraries from JPackage if JPackage has them available. (Note: for each such library, you will have to get both the external package itself (e.g. oro-2.0.8-2jpp.noarch.rpm) and the small library that links Ant and the external package (e.g. ant-apache-oro-1.6.2-3jpp.noarch.rpm).

However, JPackage does not package proprietary software, and since some of the optional packages depend on proprietary JARs, they must be handled as follows. This may violate the spirit of JPackage, but it is necessary if you need these proprietary packages. For example, suppose you want to install support for NetRexx, which JPackage does not support:

1. Decide where you want to deploy the extra JARs. One option is in $ANT\_HOME/lib, which, for JPackage is usually /usr/share/ant/lib. Another, less messy option is to create an .ant/lib subdirectory of your home directory and place your non-JPackage Ant JARs there, thereby avoiding mixing JPackage libraries with non-JPackage stuff in the same folder. More information on where Ant finds its libraries is available [here](https://ant.apache.org/manual/running.html#libs)
2. Download a non-JPackage binary distribution from the regular [Apache Ant site](https://ant.apache.org/bindownload.cgi)
3. Unzip or untar the distribution into a temporary directory
4. Copy the linking JAR, in this case ant-jai.jar, into the library directory you chose in step 1 above.
5. Copy the proprietary JAR itself into the same directory.

Finally, if for some reason you are running on a system with both the JPackage and Apache versions of Ant available, if you should want to run the Apache version (which will have to be specified with an absolute file name, not found on the path), you should use Ant's --noconfig command-line switch to avoid JPackage's classpath mechanism.

**Advanced**

There are many different ways to run Ant. What you need is at least the following:

* The classpath for Ant must contain ant.jar and any JARs/classes needed for your chosen JAXP-compliant XML parser.
* When you need JDK functionality (such as for the [javac](https://ant.apache.org/manual/Tasks/javac.html) task or the [rmic](https://ant.apache.org/manual/Tasks/rmic.html) task), then tools.jar must be added. The scripts supplied with Ant, in the bin directory, will add the required JDK classes automatically, if the JAVA\_HOME environment variable is set.
* When you are executing platform-specific applications, such as the [exec](https://ant.apache.org/manual/Tasks/exec.html) task or the [cvs](https://ant.apache.org/manual/Tasks/cvs.html) task, the property ant.home must be set to the directory containing where you installed Ant. Again this is set by the Ant scripts to the value of the ANT\_HOME environment variable.

The supplied Ant shell scripts all support an ANT\_OPTS environment variable which can be used to supply extra options to Ant. Some of the scripts also read in an extra script stored in the users home directory, which can be used to set such options. Look at the source for your platform's invocation script for details.

**Building Ant**

To build Ant from source, you can either install the Ant source distribution or clone the Ant repository from Git. See [Source Distribution](https://ant.apache.org/manual/install.html#sourceEdition) for details.

Once you have installed the source, change into the installation directory.

Set the JAVA\_HOME environment variable to the directory where the JDK is installed. See [Installing Ant](https://ant.apache.org/manual/install.html#installing) for examples on how to do this for your operating system.

**Note**: The bootstrap process of Ant requires a greedy compiler like OpenJDK or Oracle's javac. It does not work with gcj or kjc.

Make sure you have downloaded any auxiliary JARs required to build tasks you are interested in. These should be added to the lib/optional directory of the source tree. See [Library Dependencies](https://ant.apache.org/manual/install.html#librarydependencies) for a list of JAR requirements for various features. Note that this will make the auxiliary JAR available for the building of Ant only. For running Ant you will still need to make the JARs available as described under [Installing Ant](https://ant.apache.org/manual/install.html#installing).

You can also get most of the auxiliary JAR files (i.e. the JAR files that various optional Ant tasks depend on) by running Ant on the fetch.xml build file. See [Optional Tasks](https://ant.apache.org/manual/install.html#optionalTasks) for instructions on how to do this.

*Since Ant 1.7.0*, Ant has a hard dependency on JUnit. The fetch.xml build script will download JUnit automatically, but if you don't use this you must install it manually into lib/optional (download it from [JUnit.org](https://junit.org/)) if you are using a source distribution of Ant.

Your are now ready to build Ant:

build -Ddist.dir=<*directory-to-contain-Ant-distribution*> dist    (*Windows*)

sh build.sh -Ddist.dir=<*directory-to-contain-Ant-distribution*> dist    (*Unix*)

This will create a binary distribution of Ant in the directory you specified.

The above action does the following:

* If necessary it will bootstrap the Ant code. Bootstrapping involves the manual compilation of enough Ant code to be able to run Ant. The bootstrapped Ant is used for the remainder of the build steps.
* Invokes the bootstrapped Ant with the parameters passed to the build script. In this case, these parameters define an Ant property value and specify the dist target in Ant's own build.xml file.
* Create the ant.jar and ant-launcher.jar JAR files
* Create optional JARs for which the build had the relevant libraries. If a particular library is missing from lib/optional, then the matching ant-library JAR file will not be created. For example, ant-junit.jar is only built if there is a junit.jar in the lib/optional directory.

On most occasions you will not need to explicitly bootstrap Ant since the build scripts do that for you. However, if the build file you are using makes use of features not yet compiled into the bootstrapped Ant, you will need to manually bootstrap. Run bootstrap.bat (Windows) or bootstrap.sh (UNIX) to build a new bootstrap version of Ant.

If you wish to install the build into the current ANT\_HOME directory, you can use:

build install    (*Windows*)

sh build.sh install    (*Unix*)

You can avoid the lengthy Javadoc step, if desired, with:

build install-lite    (*Windows*)

sh build.sh install-lite    (*Unix*)

This will only install the bin and lib directories.

Both the *install* and *install-lite* targets will overwrite the current Ant version in ANT\_HOME.

Ant's build script will try to set executable flags for its shell scripts on Unix(-like) systems. There are various reasons why the [chmod](https://ant.apache.org/manual/Tasks/chmod.html) task might fail (like when you are running the build script as a different user than the one who installed Ant initially). In this case you can set the Ant property chmod.fail to false when starting the build like in

sh build.sh install -Dchmod.fail=false

and any error to change permission will not result in a build failure.

**Library Dependencies**

The following libraries are needed in Ant's classpath if you are using the indicated feature. Note that only one of the regexp libraries is needed for use with the mappers (and Java includes a regexp implementation which Ant will find automatically). You will also need to install the particular Ant optional JAR containing the task definitions to make these tasks available. Please refer to the [Installing Ant / Optional Tasks](https://ant.apache.org/manual/install.html#optionalTasks) section above.

|  |  |  |
| --- | --- | --- |
| **JAR Name** | **Needed For** | **Available At** |
| jakarta-regexp-1.4.jar | [regexp](https://ant.apache.org/manual/Types/mapper.html#regexp-mapper) type with mappers (if you do not wish to use java.util.regex) | <https://attic.apache.org/projects/jakarta-regexp.html> |
| jakarta-oro-2.0.8.jar | [regexp](https://ant.apache.org/manual/Types/mapper.html#regexp-mapper) type with mappers (if you do not wish to use java.util.regex) or [ftp](https://ant.apache.org/manual/Tasks/ftp.html) task with [commons-net](https://ant.apache.org/manual/install.html#commons-net) 1.4.1 | <https://attic.apache.org/projects/jakarta-oro.html> |
| junit.jar | [junit](https://ant.apache.org/manual/Tasks/junit.html) task (may be in classpath passed to task rather than Ant's classpath) | <https://junit.org/> |
| xalan.jar | [junitreport](https://ant.apache.org/manual/Tasks/junitreport.html) task | <https://xml.apache.org/xalan-j/> |
| antlr.jar | [antlr](https://ant.apache.org/manual/Tasks/antlr.html) task | <https://www.antlr.org/> |
| bsf.jar | [script](https://ant.apache.org/manual/Tasks/script.html) task **Note**: Ant 1.6 and later require Apache BSF 2.3.0 or later. **Note**: BSF 2.4.0 is needed to use a 1.5R4 or later versions of Rhino JavaScript. **Note**: BSF 2.4.0 uses Commons Logging so it needs the commons-logging.jar. | <https://commons.apache.org/bsf/> |
| Groovy JARs | Groovy Ant tasks with bindings or Groovy with [script](https://ant.apache.org/manual/Tasks/script.html) and [scriptdef](https://ant.apache.org/manual/Tasks/scriptdef.html) tasks | <http://groovy-lang.org/> Use either groovy-ant for Groovy Ant tasks with bindings or groovy-bsf for Groovy with script and scriptdef tasks (or groovy-all) |
| netrexx.jar | [netrexxc](https://ant.apache.org/manual/Tasks/netrexxc.html) task, Rexx with [script](https://ant.apache.org/manual/Tasks/script.html) task | <https://www.ibm.com/software/awdtools/netrexx/library.html> |
| rhino.jar (included in Java 7 runtime, replaced by Nashorn in Java 8 and later) | JavaScript with [script](https://ant.apache.org/manual/Tasks/script.html) task **Note**: Apache BSF 2.4.0 works only with Rhino 1.5R4 and later versions. | <https://www.mozilla.org/rhino/> |
| jython.jar | Python with [script](https://ant.apache.org/manual/Tasks/script.html) task | <http://www.jython.org/> |
| jacl.jar and tcljava.jar | TCL with [script](https://ant.apache.org/manual/Tasks/script.html) task | [http://tcljava.sourceforge.net/](http://tcljava.sourceforge.net) |
| jtcl.jar | TCL with [script](https://ant.apache.org/manual/Tasks/script.html) task | <https://jtcl-project.github.io/jtcl/> |
| BeanShell JAR(s) | BeanShell with [script](https://ant.apache.org/manual/Tasks/script.html) task. **Note**: Ant requires BeanShell version 1.3 or later | <http://www.beanshell.org/> |
| jruby.jar | Ruby with [script](https://ant.apache.org/manual/Tasks/script.html) task | <http://jruby.org/> |
| judo.jar | Judo language with [script](https://ant.apache.org/manual/Tasks/script.html) task | <http://www.judoscript.org/> |
| commons-logging.jar | CommonsLoggingListener | <https://commons.apache.org/logging/> |
| log4j.jar | Log4jListener | <https://logging.apache.org/log4j/> |
| commons-net.jar | [ftp](https://ant.apache.org/manual/Tasks/ftp.html), [rexec](https://ant.apache.org/manual/Tasks/rexec.html) and [telnet](https://ant.apache.org/manual/Tasks/telnet.html) tasks A minimum version of commons-net of 1.4.0 is needed to compile Ant, earlier versions did not support the full range of configuration options. jakarta-oro 2.0.8 is required together with commons-net 1.4.x at run time. **Note**: do not use commons-net 3.2 because of [performance issues](https://issues.apache.org/jira/browse/NET-493) | <https://commons.apache.org/net/> |
| bcel.jar | [classfileset](https://ant.apache.org/manual/Types/classfileset.html) data type, JavaClassHelper used by the ClassConstants filter reader and optionally used by [ejbjar](https://ant.apache.org/manual/Tasks/ejb.html#ejbjar) task for dependency determination | <https://commons.apache.org/bcel/> |
| javax.mail-api.jar | [mail](https://ant.apache.org/manual/Tasks/mail.html) task with MIME encoding, and *deprecated* [mimemail](https://ant.apache.org/manual/Tasks/mimemail.html) task | <https://javaee.github.io/javamail/> |
| activation.jar (included in Java 6 to Java 10 but the java.activation module is deprecated and marked for removal in Java 9 and needs to be enabled explicitly on Java 10) | [mail](https://ant.apache.org/manual/Tasks/mail.html) task with MIME encoding, and *deprecated* [mimemail](https://ant.apache.org/manual/Tasks/mimemail.html) task | [https://www.oracle.com/technetwork/java/javase/jaf-135115.html](https://github.com/javaee/activation) |
| jdepend.jar | [jdepend](https://ant.apache.org/manual/Tasks/jdepend.html) task | <https://github.com/clarkware/jdepend> |
| resolver.jar **1.1 or later** | [xmlcatalog](https://ant.apache.org/manual/Types/xmlcatalog.html) datatype *only if support for external catalog files is desired* | <https://xerces.apache.org/xml-commons/components/resolver/> |
| jsch.jar **0.1.54 or later** | [sshexec](https://ant.apache.org/manual/Tasks/sshexec.html) and [scp](https://ant.apache.org/manual/Tasks/scp.html) tasks | <http://www.jcraft.com/jsch/> |
| JAI—Java Advanced Imaging | [image](https://ant.apache.org/manual/Tasks/image.html) task | <https://download.java.net/media/jai/builds/release/1_1_3/INSTALL.html> |
| XZ—XZ for Java **1.6 or later** | [xz](https://ant.apache.org/manual/Tasks/pack.html) and [unxz](https://ant.apache.org/manual/Tasks/unpack.html) tasks, [xzresource](https://ant.apache.org/manual/Types/resources.html#xzresource), xz compression in [tar](https://ant.apache.org/manual/Tasks/tar.html)/[untar](https://ant.apache.org/manual/Tasks/unzip.html) tasks | <https://www.tukaani.org/xz/java.html> |
| JUnit 5 Platform jars:   * junit-platform-commons.jar * junit-platform-engine.jar * junit-platform-launcher.jar | [junitlauncher](https://ant.apache.org/manual/Tasks/junitlauncher.html) task. Additional libraries maybe needed depending on the selected test engines, details of which are available in that task's documentation | <https://junit.org/junit5/> |

**Troubleshooting**

**Diagnostics**

Ant has a built in diagnostics feature. If you run ant -diagnostics, Ant will look at its internal state and print it out. This code will check and print the following things.

* Where Ant is running from. Sometimes you can be surprised.
* The version of ant.jar and of the ant-\*.jar containing the optional tasks—and whether they match
* Which JAR files are in ANT\_HOME/lib
* Which optional tasks are available. If a task is not listed as being available, either it is not present, or libraries that it depends on are absent.
* XML Parser information
* JVM system properties
* The status of the temp directory. If this is not writable, or its timestamp is horribly wrong (possible if it is on a network drive), a lot of tasks may fail with obscure error messages.
* The current time zone as Java sees it. If this is not what it should be for your location, then dependency logic may get confused.

Running ant -diagnostics is a good way to check that Ant is installed. It is also a first step towards self-diagnosis of any problem. Any configuration problem reported to the user mailing list will probably result ins someone asking you to run the command and show the results, so save time by using it yourself.

For diagnostics from within IDE, use the [diagnostics](https://ant.apache.org/manual/Tasks/diagnostics.html) task to run the same tests as an Ant option. This can be added to a diagnostics target in a build file to see what tasks are available under the IDE, what the XML parser and classpath is, etc.

**User Mailing List**

If you cannot get Ant installed or working, the Ant user mailing list is the best place to start with any problem. Please do your homework first, make sure that it is not a [CLASSPATH](https://ant.apache.org/manual/install.html#classpath) problem, and run a [diagnostics check](https://ant.apache.org/manual/install.html#diagnostics) to see what Ant thinks of its own state. Why the user list, and not the developer list? Because there are more users than developers, so more people who can help you.

Please only file a bug report against Ant for a configuration/startup problem if there really is a fixable bug in Ant related to configuration, such as it not working on a particular platform, with a certain JVM version, etc, or if you are advised to do it by the user mailing list.