ANT

ANT stands for Another Neat Tool. It is a Java-based build tool from Apache. Before going into the details of Apache Ant, let us first understand why we need a build tool in the first place.

Need for a Build Tool

On an average, a developer spends a substantial amount of time doing mundane tasks like build and deployment that include:

* Compiling the code
* Packaging the binaries
* Deploying the binaries to the test server
* Testing the changes
* Copying the code from one location to another

To automate and simplify the above tasks, Apache Ant is useful. It is an Operating System build and deployment tool that can be executed from the command line.

Installing Apache Ant

It is assumed that you have already downloaded and installed Java Development Kit (JDK) on your computer. If not, please follow the instructions[here](https://www.tutorialspoint.com/java/java_environment_setup.htm).

* Ensure that the JAVA\_HOME environment variable is set to the folder where your JDK is installed.
* Download the binaries from [http://ant.apache.org](http://ant.apache.org/)
* Unzip the zip file to a convenient location c:\folder. using Winzip, winRAR, 7-zip or similar tools.
* Create a new environment variable called **ANT\_HOME** that points to the Ant installation folder, in this case **c:\apache-ant-1.8.2-bin**folder.
* Append the path to the Apache Ant batch file to the PATH environment variable. In our case this would be the **c:\apache-ant-1.8.2-bin\bin**folder.

Verifying Apache Ant Installation

To verify the successful installation of Apache Ant on your computer, type ant on your command prompt.

You should see an output similar to:

C:\>ant -version

Apache Ant(TM) version 1.8.2 compiled on December 20 2010

If you do not see the above output, then please verify that you have followed the installation steps properly.

or this exercise, create a file called build.xml anywhere in your computer with the following contents in it:

<?xml version="1.0"?>

<project name="Hello World Project" default="info">

<target name="info">

<echo>Hello World - Welcome to Apache Ant!</echo>

</target>

</project>

Note that there should be no blank line(s) or whitespace(s) before the xml declaration. If you allow them, the following error message occurs while executing the ant build -

*The processing instruction target matching "[xX][mM][lL]" is not allowed.* All build files require the **project** element and at least one **target** element.

The XML element **project** has three attributes :

|  |  |
| --- | --- |
| **Attributes** | **Description** |
| name | The Name of the project. (Optional) |
| default | The default target for the build script. A project may contain any number of targets. This attribute specifies which target should be considered as the default. (Mandatory) |
| basedir | The base directory (or) the root folder for the project. (Optional) |

A target is a collection of tasks that you want to run as one unit. In our example, we have a simple target to provide an informational message to the user.

Targets can have dependencies on other targets. For example, a **deploy**target may have a dependency on the **package** target, the **package** target may have a dependency on the **compile** target and so forth. Dependencies are denoted using the **depends** attribute. For example:

<target name="deploy" depends="package">

....

</target>

<target name="package" depends="clean,compile">

....

</target>

<target name="clean" >

....

</target>

<target name="compile" >

....

</target>

The target element has the following attributes:

|  |  |
| --- | --- |
| **Attributes** | **Description** |
| name | The name of the target (Required) |
| depends | Comma separated list of all targets that this target depends on. (Optional) |
| description | A short description of the target. (optional) |
| if | Allows the execution of a target based on the trueness of a conditional attribute. (optional) |
| unless | Adds the target to the dependency list of the specified Extension Point. An Extension Point is similar to a target, but it does not have any tasks. (Optional) |

The **echo** task in the above example is a trivial task that prints a message. In our example, it prints the message *Hello World*.

To run the ant build file, open up command prompt and navigate to the folder where the build.xml resides, and type **ant info**. You could also type **ant**instead. Both will work, because **info** is the default target in the build file. You should see the following output:

C:\>ant

Buildfile: C:\build.xml

info: [echo] Hello World - Welcome to Apache Ant!

BUILD SUCCESSFUL

Total time: 0 seconds

C:\>

Ant uses the **property** element which allows you to specify properties. This allows the properties to be changed from one build to another or from one environment to another.

By default, Ant provides the following pre-defined properties that can be used in the build file:

|  |  |
| --- | --- |
| **Properties** | **Description** |
| ant.file | The full location of the build file. |
| ant.version | The version of the Apache Ant installation. |
| basedir | The basedir of the build, as specified in the **basedir**attribute of the **project** element. |
| ant.java.version | The version of the JDK that is used by Ant. |
| ant.project.name | The name of the project, as specified in the **name**atrribute of the **project** element. |
| ant.project.default-target | The default target of the current project. |
| ant.project.invoked-targets | Comma separated list of the targets that were invoked in the current project. |
| ant.core.lib | The full location of the Ant jar file. |
| ant.home | The home directory of Ant installation. |
| ant.library.dir | The home directory for Ant library files - typically ANT\_HOME/lib folder. |

Ant also makes the system properties (Example: file.separator) available to the build file.

In addition to the above, the user can define additional properties using the**property** element. The following example shows how to define a property called **sitename**:

<?xml version="1.0"?>

<project name="Hello World Project" default="info">

<property name="sitename" value="www.tutorialspoint.com"/>

<target name="info">

<echo>Apache Ant version is ${ant.version} - You are at ${sitename} </echo>

</target>

</project>

Running Ant on the above build file produces the following output:

C:\>ant

Buildfile: C:\build.xml

info: [echo] Apache Ant version is Apache Ant(TM) version 1.8.2

compiled on December 20 2010 - You are at www.tutorialspoint.com

BUILD SUCCESSFUL

Total time: 0 seconds

C:\>

The following example shows a **build.xml** file and its associated**build.properties** file:

build.xml

<?xml version="1.0"?>

<project name="Hello World Project" default="info">

<property file="build.properties"/>

<target name="info">

<echo>Apache Ant version is ${ant.version} - You are at ${sitename} </echo>

</target>

</project>

build.properties

# The Site Name

sitename=www.tutorialspoint.com

buildversion=3.3.2

In the above example, **sitename** is a custom property which is mapped to the website name. You can declare any number of custom properties in this fashion. Another custom property listed in the above example is the**buildversion**, which, in this instance refers to the version of the build.

In addition to the above, Ant comes with a number of predefined build properties, which are listed in the previous section, but is represented below once again.

|  |  |
| --- | --- |
| **Properties** | **Description** |
| ant.file | The full location of the build file. |
| ant.version | The version of the Apache Ant installation. |
| basedir | The basedir of the build, as specified in the **basedir**attribute of the **project** element. |
| ant.java.version | The version of the JDK that is used by Ant. |
| ant.project.name | The name of the project, as specified in the **name**atrribute of the **project** element. |
| ant.project.default-target | The default target of the current project. |
| ant.project.invoked-targets | Comma separated list of the targets that were invoked in the current project. |
| ant.core.lib | The full location of the Ant jar file. |
| ant.home | The home directory of Ant installation. |
| ant.library.dir | The home directory for Ant library files - typically ANT\_HOME/lib folder. |

The example presented in this chapter uses the **ant.version** built-in property.

The following data types are provided by Apache Ant.

## Fileset

The fileset data types represents a collection of files. It is used as a filter to include or exclude files that match a particular pattern.

For example, refer the following code. Here, the src attribute points to the source folder of the project.

The fileset selects all .java files in the source folder except those contain the word 'Stub'. The case-sensitive filter is applied to the fileset which means a file with the name Samplestub.java will not be excluded from the fileset.

<fileset dir="${src}" casesensitive="yes">

<include name="\*\*/\*.java"/>

<exclude name="\*\*/\*Stub\*"/>

</fileset>

## Pattern set

A pattern set is a pattern that allows to filter files or folders easily based on certain patterns. Patterns can be created using the following meta characters:

* **?** - Matches one character only.
* **\*** - Matches zero or many characters.
* **\*\*** - Matches zero or many directories recursively.

The following example depicts the usage of a pattern set.

<patternset id="java.files.without.stubs">

<include name="src/\*\*/\*.java"/>

<exclude name="src/\*\*/\*Stub\*"/>

</patternset>

The patternset can then be reused with a fileset as follows:

<fileset dir="${src}" casesensitive="yes">

<patternset refid="java.files.without.stubs"/>

</fileset>

## File list

The filelist data type is similar to the file set except the following differences:

* filelist contains explicitly named lists of files and it does not support wild cards.
* filelist data type can be applied for existing or non-existing files.

Let us see the following example of the filelist data type. Here, the attribute**webapp.src.folder** points to the web application source folder of the project.

<filelist id="config.files" dir="${webapp.src.folder}">

<file name="applicationConfig.xml"/>

<file name="faces-config.xml"/>

<file name="web.xml"/>

<file name="portlet.xml"/>

</filelist>

## Filter set

Using a filterset data type along with the copy task, you can replace certain text in all files that matches the pattern with a replacement value.

A common example is to append the version number to the release notes file, as shown in the following code.

<copy todir="${output.dir}">

<fileset dir="${releasenotes.dir}" includes="\*\*/\*.txt"/>

<filterset>

<filter token="VERSION" value="${current.version}"/>

</filterset>

</copy>

In this Code:

* The attribute **output.dir** points to the output folder of the project.
* The attribute **releasenotes.dir** points to the release notes folder of the project.
* The attribute **current.version** points to the current version folder of the project.
* The copy task, as the name suggests, is used to copy files from one location to another.

## Path

The **path** data type is commonly used to represent a class-path. Entries in the path are separated using semicolons or colons. However, these characters are replaced at the run-time by the executing system's path separator character.

The classpath is set to the list of jar files and classes in the project, as shown in the example below.

<path id="build.classpath.jar">

<pathelement path="${env.J2EE\_HOME}/${j2ee.jar}"/>

<fileset dir="lib">

<include name="\*\*/\*.jar"/>

</fileset>

</path>

The next logical step after compiling your java source files, is to build the java archive, i.e., the JAR file. Creating JAR files with Ant is quite easy with the **jar**task. The commonly used attributes of the jar task are as follows:

|  |  |
| --- | --- |
| **Attributes** | **Description** |
| basedir | The base directory for the output JAR file. By default, this is set to the base directory of the project. |
| compress | Advises Ant to compress the file as it creates the JAR file. |
| keepcompression | While the **compress** attribute is applicable to the individual files, the **keepcompression** attribute does the same thing, but it applies to the entire archive. |
| destfile | The name of the output JAR file. |
| duplicate | Advises Ant on what to do when duplicate files are found. You could add, preserve, or fail the duplicate files. |
| excludes | Advises Ant to not include these comma separated list of files in the package. |
| excludesfile | Same as above, except the exclude files are specified using a pattern. |
| inlcudes | Inverse of excludes. |
| includesfile | Inverse of excludesfile. |
| update | Advises Ant to overwrite files in the already built JAR file. |

Continuing our **Hello World** Fax Application project, let us add a new target to produce the jar files. But before that, let us consider the jar task given below.

<jar destfile = "${web.dir}/lib/util.jar"

basedir = "${build.dir}/classes"

includes = "faxapp/util/\*\*"

excludes = "\*\*/Test.class" />

Here, the **web.dir** property points to the path of the web source files. In our case, this is where the util.jar will be placed.

The **build.dir** property in this example points to the build folder where the class files for the util.jar can be found.

In this example, we create a jar file called **util.jar** using the classes from the**faxapp.util.\*** package. However, we are excluding the classes that end with the name Test. The output jar file will be placed in the web application lib folder.

If we want to make the util.jar an executable jar file we need to add the**manifest** with the **Main-Class** meta attribute.

Therefore, the above example will be updated as:

<jar destfile = "${web.dir}/lib/util.jar"

basedir = "${build.dir}/classes"

includes = "faxapp/util/\*\*"

excludes = "\*\*/Test.class">

<manifest>

<attribute name = "Main-Class" value = "com.tutorialspoint.util.FaxUtil"/>

</manifest>

</jar>

To execute the jar task, wrap it inside a target, most commonly, the build or package target, and execute them.

<target name="build-jar">

<jar destfile="${web.dir}/lib/util.jar"

basedir="${build.dir}/classes"

includes="faxapp/util/\*\*"

excludes="\*\*/Test.class">

<manifest>

<attribute name="Main-Class" value="com.tutorialspoint.util.FaxUtil"/>

</manifest>

</jar>

</target>

Running Ant on this file creates the util.jar file for us.

The following outcome is the result of running the Ant file:

C:\>ant build-jar

Buildfile: C:\build.xml

BUILD SUCCESSFUL

Total time: 1.3 seconds

The util.jar file is now placed in the output folder.

Creating WAR files with Ant is extremely simple, and very similar to the creating JAR files task. After all, WAR file, like JAR file is just another ZIP file.

The WAR task is an extension to the JAR task, but it has some nice additions to manipulate what goes into the WEB-INF/classes folder, and generating the web.xml file. The WAR task is useful to specify a particular layout of the WAR file.

Since the WAR task is an extension of the JAR task, all attributes of the JAR task apply to the WAR task.

|  |  |
| --- | --- |
| **Attributes** | **Description** |
| webxml | Path to the web.xml file |
| lib | A grouping to specify what goes into the WEB-INF\lib folder. |
| classes | A grouping to specify what goes into the WEB-INF\classes folder. |
| metainf | Specifies the instructions for generating the MANIFEST.MF file. |

Continuing our **Hello World** Fax Application project, let us add a new target to produce the jar files. But before that let us consider the war task. Consider the following example:

<war destfile = "fax.war" webxml = "${web.dir}/web.xml">

<fileset dir = "${web.dir}/WebContent">

<include name = "\*\*/\*.\*"/>

</fileset>

<lib dir = "thirdpartyjars">

<exclude name = "portlet.jar"/>

</lib>

<classes dir = "${build.dir}/web"/>

</war>

As per the previous examples, the **web.dir** variable refers to the source web folder, i.e, the folder that contains the JSP, css, javascript files etc.

The **build.dir** variable refers to the output folder - This is where the classes for the WAR package can be found. Typically, the classes will be bundled into the WEB-INF/classes folder of the WAR file.

In this example, we are creating a war file called fax.war. The WEB.XML file is obtained from the web source folder. All files from the 'WebContent' folder under web are copied into the WAR file.

The WEB-INF/lib folder is populated with the jar files from the thirdpartyjars folder. However, we are excluding the portlet.jar as this is already present in the application server's lib folder. Finally, we are copying all classes from the build directory's web folder and putting into the WEB-INF/classes folder.

Wrap the war task inside an Ant target (usually package) and run it. This will create the WAR file in the specified location.

It is entirely possible to nest the classes, lib, metainf and webinf directors so that they live in scattered folders anywhere in the project structure. But best practices suggest that your Web project should have the Web Content structure that is similar to the structure of the WAR file. The Fax Application project has its structure outlined using this basic principle.

To execute the war task, wrap it inside a target, most commonly, the build or package target, and run them.

<target name="build-war">

<war destfile="fax.war" webxml="${web.dir}/web.xml">

<fileset dir="${web.dir}/WebContent">

<include name="\*\*/\*.\*"/>

</fileset>

<lib dir="thirdpartyjars">

<exclude name="portlet.jar"/>

</lib>

<classes dir="${build.dir}/web"/>

</war>

</target>

Running Ant on this file will create the **fax.war** file for us.

We have learnt the different aspects of Ant using the **Hello World** Fax web application in bits and pieces.

Now it is time to put everything together to create a full and complete build.xml file. Consider **build.properties** and **build.xml** files as follows:

## build.properties

deploy.path = c:\tomcat6\webapps

## build.xml

<?xml version = "1.0"?>

<project name = "fax" basedir = "." default = "usage">

<property file = "build.properties"/>

<property name = "src.dir" value = "src"/>

<property name = "web.dir" value = "war"/>

<property name = "javadoc.dir" value = "doc"/>

<property name = "build.dir" value = "${web.dir}/WEB-INF/classes"/>

<property name = "name" value = "fax"/>

<path id = "master-classpath">

<fileset dir = "${web.dir}/WEB-INF/lib">

<include name = "\*.jar"/>

</fileset>

<pathelement path = "${build.dir}"/>

</path>

<target name = "javadoc">

<javadoc packagenames = "faxapp.\*" sourcepath = "${src.dir}"

destdir = "doc" version = "true" windowtitle = "Fax Application">

<doctitle><![CDATA[<h1> = Fax Application = </h1>]]>

</doctitle>

<bottom><![CDATA[Copyright © 2011. All Rights Reserved.]]>

</bottom>

<group title = "util packages" packages = "faxapp.util.\*"/>

<group title = "web packages" packages = "faxapp.web.\*"/>

<group title = "data packages" packages = "faxapp.entity.\*:faxapp.dao.\*"/>

</javadoc>

</target>

<target name = "usage">

<echo message = ""/>

<echo message = "${name} build file"/>

<echo message = "-----------------------------------"/>

<echo message = ""/>

<echo message = "Available targets are:"/>

<echo message = ""/>

<echo message = "deploy --> Deploy application as directory"/>

<echo message = "deploywar --> Deploy application as a WAR file"/>

<echo message = ""/>

</target>

<target name = "build" description = "Compile main source tree java files">

<mkdir dir = "${build.dir}"/>

<javac destdir = "${build.dir}" source = "1.5" target = "1.5" debug = "true"

deprecation = "false" optimize = "false" failonerror = "true">

<src path = "${src.dir}"/>

<classpath refid = "master-classpath"/>

</javac>

</target>

<target name = "deploy" depends = "build" description = "Deploy application">

<copy todir = "${deploy.path}/${name}" preservelastmodified = "true">

<fileset dir = "${web.dir}">

<include name = "\*\*/\*.\*"/>

</fileset>

</copy>

</target>

<target name = "deploywar" depends = "build" description = "Deploy application as a WAR file">

<war destfile = "${name}.war" webxml = "${web.dir}/WEB-INF/web.xml">

<fileset dir = "${web.dir}">

<include name = "\*\*/\*.\*"/>

</fileset>

</war>

<copy todir = "${deploy.path}" preservelastmodified = "true">

<fileset dir = ".">

<include name = "\*.war"/>

</fileset>

</copy>

</target>

<target name = "clean" description = "Clean output directories">

<delete>

<fileset dir = "${build.dir}">

<include name = "\*\*/\*.class"/>

</fileset>

</delete>

</target>

</project>

In this example:

* We first declare the path to the webapps folder in Tomcat in the build properties file as the **deploy.path** variable.
* We also declare the source folder for the java files in the **src.dir**variable.
* Then we declare the source folder for the web files in the **web.dir**variable. **javadoc.dir** is the folder for storing the java documentation, and **build.dir** is the path for storing the build output files.
* Then we declare the name of the web application, which is **fax** in our case.
* We also define the master class path which contains the JAR files present in the WEB-INF/lib folder of the project.
* We also include the class files present in the **build.dir** in the master class path.
* The Javadoc target produces the javadoc required for the project and the usage target is used to print the common targets that are present in the build file.

The above example shows two deployment targets : **deploy** and **deploywar.**

The deploy target copies the files from the web directory to the deploy directory preserving the last modified date time stamp. This is useful when deploying to a server that supports hot deployment.

The clean target clears all the previously built files.

The deploywar target builds the war file and then copies the war file to the deploy directory of the application server.

## AntCall

### Description

Call another target within the same buildfile optionally specifying some properties (params in this context). **This task must not be used outside of a target.**

By default, all of the properties of the current project will be available in the new project. Alternatively, you can set the *inheritAll*attribute to false and only "user" properties (i.e., those passed on the command-line) will be passed to the new project. In either case, the set of properties passed to the new project will override the properties that are set in the new project

### Examples

<target name="default">

<antcall target="doSomethingElse">

<param name="param1" value="value"/>

</antcall>

</target>

<target name="doSomethingElse">

<echo message="param1=${param1}"/>

</target>

Will run the target 'doSomethingElse' and echo 'param1=value'.

<antcall ... >

<reference refid="path1" torefid="path2"/>

</antcall>

will copy the parent's definition of path1 into the new project using the id path2.

## Exec

### Description

Executes a system command. When the *os* attribute is specified, then the command is only executed when Apache Ant is run on one of the specified operating systems

#### Windows Users

The <exec> task delegates to Runtime.exec which in turn apparently calls [::CreateProcess](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dllproc/base/createprocess.asp). It is the latter Win32 function that defines the exact semantics of the call. In particular, if you do not put a file extension on the executable, only ".EXE" files are looked for, not ".COM", ".CMD" or other file types listed in the environment variable PATHEXT. That is only used by the shell.

Note that .bat files cannot in general by executed directly. One normally needs to execute the command shell executable cmd using the /cswitch.

<target name="help">

<exec executable="cmd">

<arg value="/c"/>

<arg value="ant.bat"/>

<arg value="-p"/>

</exec>

</target>

### Examples

<exec dir="${src}" executable="cmd.exe" os="Windows 2000" output="dir.txt">

<arg line="/c dir"/>

</exec>

## Copy

### Description

Copies a file or resource collection to a new file or directory. By default, files are only copied if the source file is newer than the destination file, or when the destination file does not exist. However, you can explicitly overwrite files with the overwrite attribute

### Examples

**Copy a single file**

<copy file="myfile.txt" tofile="mycopy.txt"/>

**Copy a single file to a directory**

<copy file="myfile.txt" todir="../some/other/dir"/>

**Copy a directory to another directory**

<copy todir="../new/dir">

<fileset dir="src\_dir"/>

</copy>

**Copy a set of files to a directory**

<copy todir="../dest/dir">

<fileset dir="src\_dir">

<exclude name="\*\*/\*.java"/>

</fileset>

</copy>

<copy todir="../dest/dir">

<fileset dir="src\_dir" excludes="\*\*/\*.java"/>

</copy>

**Copy a set of files to a directory, appending .bak to the file name on the fly**

<copy todir="../backup/dir">

<fileset dir="src\_dir"/>

<globmapper from="\*" to="\*.bak"/>

</copy>

**Copy a set of files to a directory, replacing @TITLE@ with Foo Bar in all files.**

<copy todir="../backup/dir">

<fileset dir="src\_dir"/>

<filterset>

<filter token="TITLE" value="Foo Bar"/>

</filterset>

</copy>

**Collect all items from the current CLASSPATH setting into a destination directory, flattening the directory structure.**

<copy todir="dest" flatten="true">

<path>

<pathelement path="${java.class.path}"/>

</path>

</copy>

**Copies some resources to a given directory.**

<copy todir="dest" flatten="true">

<resources>

<file file="src\_dir/file1.txt"/>

<url url="http://ant.apache.org/index.html"/>

</resources>

</copy>

If the example above didn't use the flatten attribute, the <file> resource would have returned its full path as source and target name and would not have been copied at all. In general it is a good practice to use an explicit mapper together with resources that use an absolute path as their names.

**Copies the two newest resources into a destination directory.**

<copy todir="dest" flatten="true">

<first count="2">

<sort>

<date xmlns="antlib:org.apache.tools.ant.types.resources.comparators"/>

<resources>

<file file="src\_dir/file1.txt"/>

<file file="src\_dir/file2.txt"/>

<file file="src\_dir/file3.txt"/>

<url url="http://ant.apache.org/index.html"/>

</resources>

</sort>

</first>

</copy>

## Copydir

### *Deprecated*

*This task has been deprecated. Use the Copy task instead.*

### Description

Copies a directory tree from the source to the destination.

### Examples

<copydir src="${src}/resources"

dest="${dist}"

/>

copies the directory ${src}/resources to ${dist}.

<copydir src="${src}/resources"

dest="${dist}"

includes="\*\*/\*.java"

excludes="\*\*/Test.java"

/>

copies the directory ${src}/resources to ${dist} recursively. All java files are copied, except for files with the name Test.java.

<copydir src="${src}/resources"

dest="${dist}"

includes="\*\*/\*.java"

excludes="mypackage/test/\*\*"/>

copies the directory ${src}/resources to ${dist} recursively. All java files are copied, except for the files under the mypackage/testdirectory.

## Copyfile

### Description

Copies a file from the source to the destination. The file is only copied if the source file is newer than the destination file, or when the destination file does not exist.

### Parameters

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Description** | **Required** |
| src | the filename of the file to copy. | Yes |
| dest | the filename of the file where to copy to. | Yes |
| filtering | indicates whether token filtering should take place during the copy | No |
| forceoverwrite | overwrite existing files even if the destination files are newer (default is false). | No |

### Examples

<copyfile src="test.java" dest="subdir/test.java"/>

<copyfile src="${src}/index.html" dest="${dist}/help/index.html"/>

## Delete

### Description

Deletes a single file, a specified directory and all its files and subdirectories, or a set of files specified by one or more [resource collection](https://ant.apache.org/manual/Types/resources.html#collection)s.

### Examples

<delete file="/lib/ant.jar"/>

deletes the file /lib/ant.jar.

<delete dir="lib"/>

deletes the lib directory, including all files and subdirectories of lib.

<delete>

<fileset dir="." includes="\*\*/\*.bak"/>

</delete>

deletes all files with the extension .bak from the current directory and any subdirectories.

<delete includeEmptyDirs="true">

<fileset dir="build"/>

</delete>

deletes all files and subdirectories of build, including build itself.

<delete includeemptydirs="true">

<fileset dir="build" includes="\*\*/\*"/>

</delete>

deletes all files and subdirectories of build, without build itself.

<delete includeemptydirs="true">

<fileset dir="src" includes="\*\*/.svn/" defaultexcludes="false"/>

</delete>

deletes the subversion metadata directories under src. Because .svn is on of the [default excludes](https://ant.apache.org/manual/dirtasks.html#defaultexcludes) you have to use the defaultexcludesflag, otherwise Ant wont delete these directories and the files in it.

## Mkdir

### Description

Creates a directory. Also non-existent parent directories are created, when necessary. Does nothing if the directory already exist.

### Parameters

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Description** | **Required** |
| dir | the directory to create. | Yes |

### Examples

<mkdir dir="${dist}"/>

creates a directory ${dist}.

<mkdir dir="${dist}/lib"/>

creates a directory ${dist}/lib.

## Move

### Description

Moves a file to a new file or directory, or collections of files to a new directory. By default, the destination file is overwritten if it already exists. When overwrite is turned off, then files are only moved if the source file is newer than the destination file, or when the destination file does not exist.

### Examples

**Move a single file (rename a file)**

<move file="file.orig" tofile="file.moved"/>

**Move a single file to a directory**

<move file="file.orig" todir="dir/to/move/to"/>

**Move a directory to a new directory**

<move todir="new/dir/to/move/to">

<fileset dir="src/dir"/>

</move>

*or, since Ant 1.6.3:*

<move file="src/dir" tofile="new/dir/to/move/to"/>

**Move a set of files to a new directory**

<move todir="some/new/dir">

<fileset dir="my/src/dir">

<include name="\*\*/\*.jar"/>

<exclude name="\*\*/ant.jar"/>

</fileset>

</move>

**Move a list of files to a new directory**

<move todir="some/new/dir">

<filelist dir="my/src/dir">

<file name="file1.txt"/>

<file name="file2.txt"/>

</filelist>

</move>

**Append ".bak" to the names of all files in a directory.**

<move todir="my/src/dir" includeemptydirs="false">

<fileset dir="my/src/dir">

<exclude name="\*\*/\*.bak"/>

</fileset>

<mapper type="glob" from="\*" to="\*.bak"/>

</move>