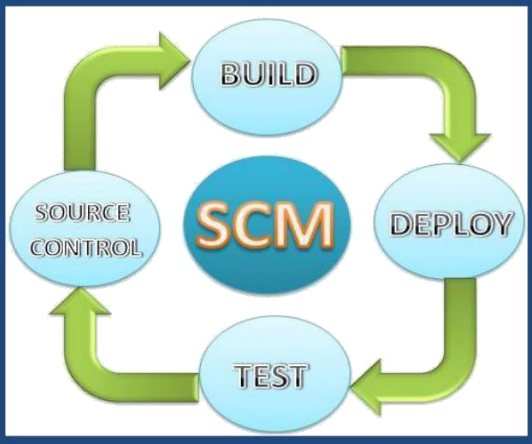

SCM/Build & Release Management

NAGABABU KANNEGANTI

Contents

Build Automation

Build automation is the act of scripting or automating a wide variety of tasks that software developers do in their day-to-day activities including things like:

compiling computer source code into binary code
packaging binary code
running tests
deployment to production systems

We have different types of scripting available in the market to make the software builds as an automated task.

Ex: Ant scripting, Maven, groovy etc…

Ant scripting:

Ant – originally an acronym for “Another Neat Tool”

Apache Ant is a software tool for automating software build processes

It is a powerful XML-based scripting tool which can automate your mundane tasks and allow you to concentrate on your business rules and code development.

Ant specification is available at http://ant.apache.org/

It allows the developer to automate the repeated process involved in the development of J2EE application.

Developers can easily write the script to automate the build process like compilation, archiving and deployment.

It is platform independent tool

Ant is particularly good at automating complicated repetitive tasks and thus is well suited for automating standardized build processes.

Ant accepts instructions in the form of XML documents thus is extensible and easy to maintain

Installation:

Download the ant from below URL and unzip it.

http://apache.techartifact.com/mirror//ant/binaries/apache-ant-1.8.4-bin.zip

After downloading add the Ant bin directory to your path (for Windows)

- set ANT_HOME = C:\ant\apache-ant-1.6.1
- set JAVA_HOME = C:\jdk1.4
- set PATH = %PATH%;%ANT_HOME%\bin

Test installation of Ant by running this command

C:\Documents and Settings\user**>ant -version**

Apache Ant version 1.6.5 compiled on June 2 2005

Running Ant

Ant by default always looks for a file named build.xml. If a build file other than build.xml needs to be used then ant should be run with options

ant –buildfile anotherbuild.xml

ant –file anotherbuild.xml

ant –f anotherbuild.xml

Build file contains exactly only one project.

A typical build file looks as below:

<?xml version="1.0" ?>

<project>

<target>

task 1 ….

task 2 ….

</target>

</project>

Every build files can contain one Project.

Every project can have multiple Targets –each one for a specific purpose

Targets can be dependent on one another

Every Target is made up of numerous Tasks bases on the Targets requirement

Project Tag – defines the properties of the build project

Each project defines one or more targets. A target is a set of tasks you want to be executed. When starting Ant, you can select which target(s) you want to have executed. When no target is given, the project's default is used.

This is the root element of the all Ant build files.

Attributes:

name – Project Name

default – The default target

basedir – The base directory for all file and path references. The default value is the current directory.

Target Tag – defines the each target build

A target can depend on other targets. You might have a target for compiling, for example, and a target for creating a distributable. You can only build a distributable when you have compiled first, so the distribute target depends on the compile target. Ant resolves these dependencies.

Attributes

name – target name

depends – a comma separated list of targets on which this target depends on

description – a short description about this target build

E.g.

<target name= “ProgA” depends= “LibB, LibC”>

<target name= “LibB” depends= “LibC”>

<target name= “LibC” >

When you set dependencies like this, before the target "ProgA" is built, Ant will check for targets, "LibB" and "LibC". If they have not already been built using the latest versions of their source codes, they will be built before "ProgA".

Tasks – tag that represents the actions that has to be done during the build process

A task is a piece of code that can be executed.A task can have multiple attributes (or arguments, if you prefer). The value of an attribute might contain references to a property. These references will be resolved before the task is executed.

Tasks have a common structure:

```
<nameattribute1="value1" attribute2="value2" ... />
```


Example build file:

```
<?xml version="1.0"?>

<project name="Sample Buildfile" default="compile"

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

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

<property name = "build.classes" value="${build.dir}/classes"/>

<property name = "build.lib" value="${build.dir}/lib"/>

<target name="prepare">

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

<mkdirdir ="${build.classes}"/>

<mkdirdir ="${build.lib}"/>

</target>

<target name="clean" description="Removes all generated

files"> <delete dir="${build.dir}"/>

</target>

<target name="compile" depends="prepare" description="Compile all source

code"> <javacsrcdir="${src.dir}" destdir="${build.classes}"/>

</target>

<target name="jar" depends="compile" description="Prepares jar and placed in dist">
```

```
<jar jarfile="${build.lib}/sample.jar" basedir="${build.classes}"
excludes="**/*Test.class"/> </target>
<target name = "all" depends = "clean,jar" description"Clean,compiles, then bulds the JAR
file"/> </project>
```

Other tasks:

Echo task: Echoes a message to the current loggers and listeners which means System.out unless overridden. A level can be specified, which controls at what logging level the message is filtered at.

```
<echo message="Hello, world"/>
```

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

```
<mkdirdir="build"/>
```

Copy task: 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

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">
<filesetdir="src_dir"/>
</copy>
```

Move task: 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

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">
<filesetdir="src/dir"/>
</move>
```

Javac task: Compiles a Java source tree. The source and destination directory will be recursively scanned for Java source files to compile. Only Java files that have no corresponding .class file or where the class file is older than the .java file will be compiled

```
<javacsrcdir="src"
destdir="build/classes"
  />
```

Sample build script for HelloWorld program

Please follow the below link to complete the exercise

http://ant.apache.org/manual/tutorial-HelloWorldWithAnt.html

How do we interpret the manual build steps into Ant build script

java-only	Ant

`md build\classes`
`javac`
`  -sourcepathsrc`
`  -d build\classes`
`src\oata\HelloWorld.java`
`echo Main-Class:`
`oata.HelloWorld>mf`
`md build\jar`
`jar cfm`
`  build\jar\HelloWorld.jar`
`  mf`
`  -C build\classes`
`  .`
 	
`java -jar`
`build\jar\HelloWorld.jar`
	`<mkdirdir="build/classes"/>`
	`<javac`
	`srcdir="src"`
	`destdir="build/classes"/>`
	`<!-- automatically detected -->`
	`<!-- obsolete; done via manifest tag -->`
	`<mkdirdir="build/jar"/>`
	`<jar`
	`destfile="build/jar/HelloWorld.jar"`
	
	`basedir="build/classes">`
	`<manifest>`
	`<attribute name="Main-Class"`
	`value="oata.HelloWorld"/>`
	`</manifest>`
	`</jar>`
	`<java jar="build/jar/HelloWorld.jar"`
	`fork="true"/>`

Properties:

These are like variables in programming languages.

Properties are immutable: whoever sets a property first freezes it for the rest of the build; they are most definitely not variables.

Ex:

```
<property name="source_dir" value="src"/>
```

Or

```
<property name="source_dir">src</property>
```

Properties can be placed in separate file and this file can be imported into the build script.

Ex:

```
<property file="build.properties"/>
```

There few built in properties available with ant, please find them below

```
basedir             the absolute path of the project's basedir (as set
with the basedir attribute of <project>).
ant.file            the absolute path of the buildfile.
ant.version         the version of Ant
ant.project.name    the name of the project that is currently executing;
it is set in the name attribute of
<project>. ant.project.default-target
the name of the currently executing
project's default target; it is set via the
default attribute of <project>.
ant.project.invoked-targets
a comma separated list of the targets that
have been specified on the command line (the
IDE, an<ant> task ...) when invoking the
current project.
ant.java.version    the JVM version Ant detected; currently it can hold
the values "1.2", "1.3",
                    "1.4",  "1.5" and "1.6".
ant.core.lib        the absolute path of the ant.jar file.
ant.home            home directory of Ant
```

read the properties inside script using ${property_name} string.

Fileset Type

Fileset is a subtask used within the other tasks like copy,move ..etc

Attributes

dir – base directory for the file set

This type is used to specify a list of files to be processed. This can be done using the nested tags given below.

<include> tag

Attributes:

name – name or the pattern of the file/files to be selected.

<includesfile> tag

Attributes:

name – name of a text file, whose each line represents a pattern of files to be included.

<exclude>tag

This is similar to include tag, but used to remove the files that have been included using the

Include and includesfile tags.

<excludesfile> tag

This is similar to includesfile tag, but used to remove the files that have been included using the include and includesfile tag.

Ex:

```
<filesetbasedir="bin">
        <include name="**.java**"/>
        <include name="test.java"/>
</fileset>
```

Patterns:

* – Any one or many characters

? – Any one character

** – Any directory or its sub

directory Ex:.

build/*.java – all java files in build directory

build/** – all the files in build directory and all its sub directories

**/mydir/* - all the files in the directory named “mydir” in any location in the directory tree

Archive tasks:

Jar :Jars a set of .class files.

The basedir attribute is the reference directory from where to jar.

Ex:

```
<jar destfile="lib/app.jar" basedir="${build}/classes"/>
```

Jar with manifest file

```
<jar destfile="build/main/checksites.jar" basedir=”build/classes”>
<manifest>
<attribute name="Main-Class"
        value="com.acme.checksites.Main"/>
</manifest>
</jar>
```

Zip :Creates a zipfile.

The basedir attribute is the reference directory from where to zip.

```
<zip destfile="${dist}/manual.zip"

basedir="htdocs/manual"

 />
```

Unzip: Unzips a zip-, war-, or jar file.

PatternSetsare used to select files to extract from the archive. If no patternset is used, all files are extracted.

```
<unzip src="${tomcat_src}/tools-src.zip" dest="${tools.home}"/>
```

War

An extension of the Jar task with special treatment for files that should end up in the WEB-INF/lib, WEB-INF/classes or WEB-INF directories of the Web Application Archive.

Ex: <war destfile="dist/AntExample.war" webxml="WebContent/WEB-INF/web.xml">

<filesetdir="WebContent"/>

<lib dir="WebContent/WEB-INF/lib"/>

<classes dir="build/classes"/>

</war>

AntCall :

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**.**

Ex:

```
target name="default">
<antcall target="doSomethingElse">
<param name="param1" value="value"/>
</antcall>
</target>

<target name="doSomethingElse">
<echo message="param1=${param1}"/>
</target>
```

Ant inputs:

These are command line inputs passed to ant at run time.

Ex: ant –f build.xml –Dinput=”value”

${input} can be accessible from the build script and it contains the value as

“value” We can pass multiple inputs from command line

Ex: ant –f build.xml –Dinput=”value” –Denv=”QA” –Djdk=”1.7”

Path &Classpath variables:

Path Data type , is using to set the path.

Ex: <path>

<pathelement location=“${libdir}/servlet.jar”/>

<pathelement path=“${builddir}”/>

</path>

<classpath>

<pathelement path="“${builddir}”/>

<filesetdir=“${libdir}” includes=“**/*.jar”/>

</classpath>

Advanced tasks:

Exec task:

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.

```
Ex: <exec dir="${src}" executable="cmd.exe" os="Windows 2000"
output="dir.txt"> <arg line="/c dir"/>
</exec>
```

If :

Perform some tasks based on whether a given condition holds true or not.This task is heavily based on the Condition framework that can be found in Ant 1.4 and later, therefore it cannot be used inconjunction with versions of Ant prior to 1.4. Due to numeruos bugs in Ant 1.4(.1) that affect this task, we recommend to use Ant 1.5 or later.

Download ant-contrib.jar to run the below task and define it in your build.xml

Ex:

```
<if>
<equals arg1="${foo}" arg2="bar"
/> <then>
<echo message="The value of property foo is bar"
/> </then>
<else>
<echo message="The value of property foo is not bar"
/> </else>
</if>
```

Ssh :Runs a command on a remote machine running SSH daemon

```
Ex: <sshexec host="somehost"
            username="dude"
```

```
        password="yo"
        command="touch somefile"/>
```

scp :Copies a file or FileSet to or from a (remote) machine running an SSH daemon. FileSet only works for copying files from the local machine to a remote machine.

```
Ex: <scp file="myfile.txt" todir="user@somehost:/home/chuck" password="password"/>
```

External libraries:

Ant by default provides a list of tasks, apart from them if we need to use any external tasks then we need to download the respective library and copy it to lib dir of ant installation.

Ant Options:

ant -help

Get help about the usage of Ant.

Ant

Search for a file with the extension ".build". If one is found, use it to build its default target.

ant –buildfile:project1.build

Build the default target of the build file "project1.build".

ant clean

Build the target named "clean".

ant -debug

Displays messages that Ant and task developers have flagged as debugging messages.

ant – logger filename

Redirects logging output to the specified logger file.

ant – logfile filename

Redirects logging output to the specified file.

ant –version

Provides the version of ant.

Sample Project demo

Write a build script to check out the code from svn and build , make a war and deploy the same on tomcat server

Review yourself:

1) What is ant?

2) How do I get started to use ant? Can you give me a "Hello World" ant script?

3) How to delete files from a directory if it exists?

5) How does ant read properties? How to set my property system?

6) How to modify properties in ant?

7) How to use ant-contrib tasks?

9) Why do I get an exception when I use location="D:\\Code\\include" as attribute of includepath?

10) How can I use ant to run a Java application?

11) How to use ant to run command line command? How to get perl script running result?

12) How to exclude multi directories in copy or delete task?

13) How to do conditional statement in ant?