ANT:

This deals with development

Source is not a machine understandable language. We need some conversion process. And that is called compilation which means binary language

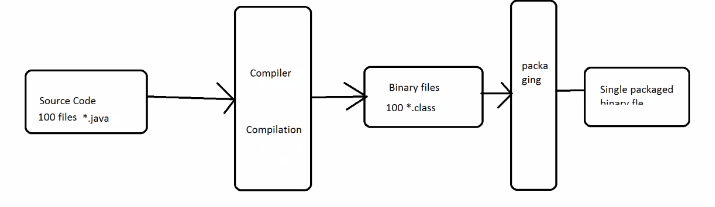
The outcome of compilation will be binary files.

If there are 100 code files, then the outcome binary files will also be 100 files

Generally, in java source code files will be with .java extension and outcome files are .class files

We can make all the files combined with one binary file called package

The entire process is called build process



The outcome file might be for standalone with .exe or .msi etc extension or might be for web application like war file

If java is a source platform, we need below pre-requisites for build process

* We should have java compiler installed
* We should have java packaging tool installed
* If we want run the package we need java runtime installed

All these three available in JDK

if we want to store the .class file in a particular location, use the below command from the same directory

javac -sourcepath src -d build\classes src\HelloWorld.java

below is the command to package the class files. Here input is class files and output is jar file

jar cfm build\jar\HelloWorld.jar -C build\classes .

in java packaging, there are two files

1. Executable files
2. Non-executable files

An executable file run with the help of java -jar helloworld.jar

To create an executable file, we need a manifest file which is metadata file contains class name of java file. Use below command to create a manifest file

Echo main-class: HelloWorld>myManifest

Now, use the below command to

Use the below command to create a jar file after manifest

jar cfm build\jar\HelloWorld.jar myManifest -C build\classes .

use the below command to check if jar is working fine

java -jar build\jar\HelloWorld.jar

all these called build process

this is called the manual process which is time consuming

maintenance is also difficult

there might be quality issues in manual process

so, we better use automation tools

Automation tools:

Java platform – ant, maven, gradle

Dotnet – msbuild, nant, VS

c/c++ - make, rake ….

Apache Ant:

This tool is from apache organisation

This is open source and platform independence

This is written in java and designed for java

To run ant, JDK is mandatory

For installing ant, copy the path and paste it on adapter settings

Then to check, open command prompt, type ant -version

Ant is a scripting language

It’s based-on xml, the scripts we need write on XML

Build.xml is the default input ant xml file

If we want to change it, use the below cmd

Ant -f abc.xml

By running the above command, it will look for the abc.xml file, if we just run ant, then it will look for build.xml file

Build.xml structure:

It starts with <project> and ends with </project> or <project/> tags

Project is like a root element. The project starts from here.

<project name=”devopstest” default=”targetA” basedir=”C:\DevOps>

Default is what we indicate to run first. It won’t run the targets in order. It follows the order which we give. And basedir is where we want to run our code

Target is a block of code separated from test of instructions

Target is something like a functions in a program

Target also has attributes

<target name=”targetA” depends=”targetB” description=”compiling”>

Description and basedir is optional, if we don’t write basedir, it will run it on current directory

Target is another subset of tag which is written inside the project

Target contains task. We can have many tasks

Task:

Task is a actual a actual thing happened in project

<echo> hello world </echo> it just prints hello world

<mkdir dir=”build”> -> it creates build directory

<delete dir=”build”/> 🡪 it deleted build directory

These tasks should be performed inside task

Build.xml should be in below format

<project>

<target>

<task> </task>

<task> </task>

</target>

<target>

<task> </task>

</target>

<target>

<task> </task>

</target>

</project>

Sample xml file:

<project name=”test” default=”A”>

<target name=”A”>

<echo> hello world </echo>

</target>

</project>

Save the file as build.xml

Now just run ant in cmd prompt.

We can write n no of tasks inside target

If we have multiple tasks inside a project, then we need to mention which tasks needs to be run as default in project, otherwise it won’t run any task

If there are three tasks, A, B & C and the default one is C. if we want to run B first, then we can mention depends:B after the name of target C. so, first it runs the dependencies

Tasks:

File system tasks: echo, mkdir, delete, copy, move

Java related tasks: javac, jar, java

Properties

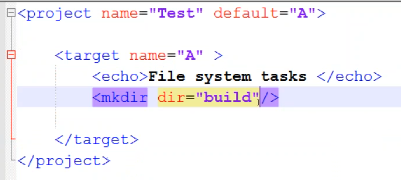
Advanced tasks

Web app building: war, path variable

External tasks

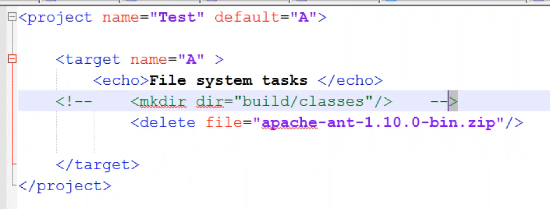
Options

Example for mkdir task



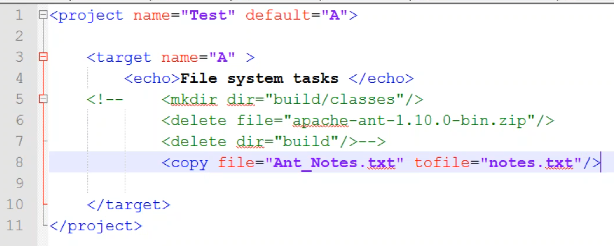
We can create not only a single task, but also chain of tasks

We can comment the line in xml file, which means ignoring the line as below screenshot

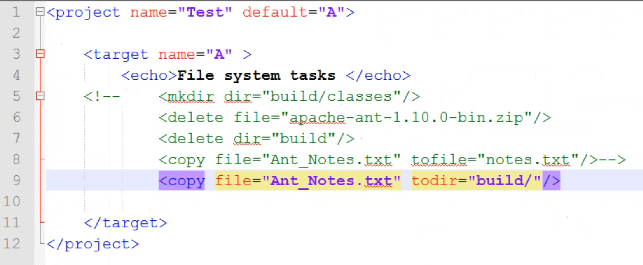


As above screenshot, it won’t perform any mkdir operation, as it is commented

We can add commenting for multiple lines as below and also refer the below image to copy from file to file

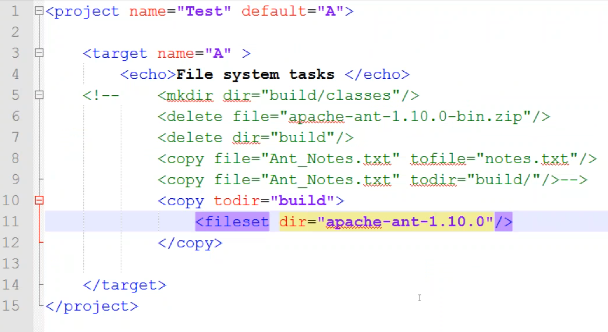


So, whatever the data in ant\_notes will be copied to notes file



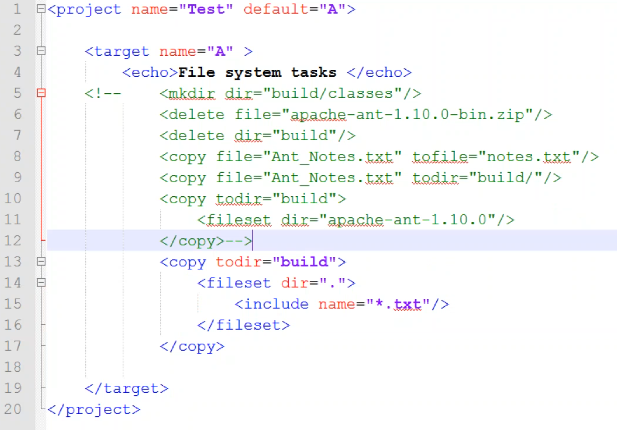
Copying the file to another directory as above

Copy one directory to another



With the below script, we can copy the .txt files from current directory to build directory. Here ‘.’ Means current directory

Move syntax is also just like the same



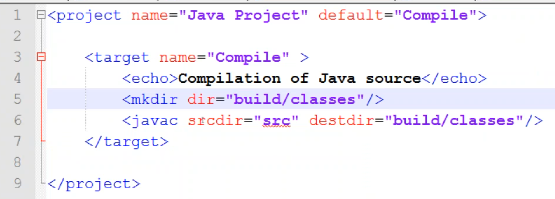
Move is also used to rename, just as below

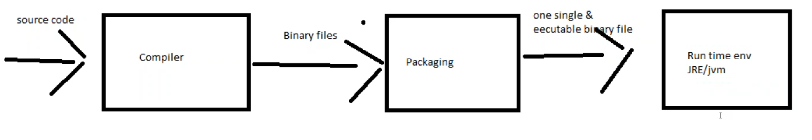


Java tasks:

Srcdir is an attribute which take the source code as an output and destdir is an output folder

Java files inside the srcdir will be compiled and copied the output to destdir



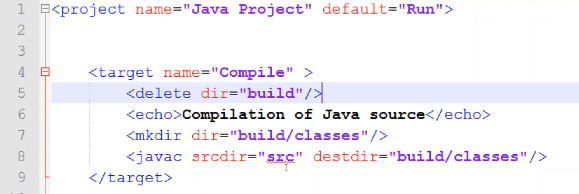


Basedir is the location of class files and destfile is the file name of jar files

Jar attribute contains the jar file

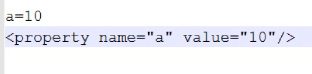
To execute the program, we need to create manifest file also while creating the package





By deleting the build directory, it will start a fresh build

Properties are the variables which are written inside the project tag. Just like below

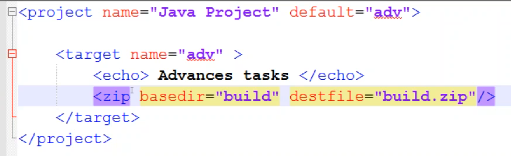


This property value we can fetch across the project

Advanced tasks:

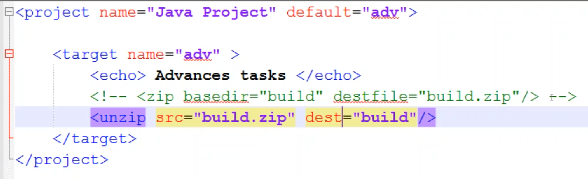
Zip, unzip, exec, antcall, antfile

Below is the example of zip



Basedir is the source build file and destfile is the destination filename

If we delete the build file and want it back, we can unzip the build.zip file, so, we can get it back



Using exec, we can perform external process execution as below

/c means command





Like this, we can execute some external commands within the ant script

Antcall:

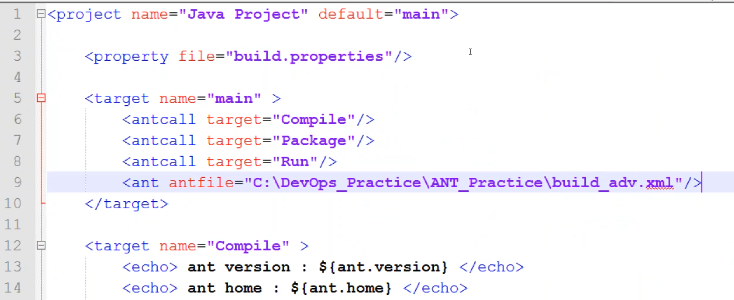
It is used for the sequence of running tasks without depends in task



Here, the default is main and in main, we have the tasks in order

Antfile:

With this, we can execute some other xml file from one xml file



As per above, once the antcall done, it will execute buils\_adv.xml as we mentioned in antfile

Generally, java applications are categorised as standalone/web application/ web apps

If it is a standalone, then the output will be .jar file

If it is a web application/web apps, the output will be .war file

Standalone app is a collection of dynamic content (\*.class)

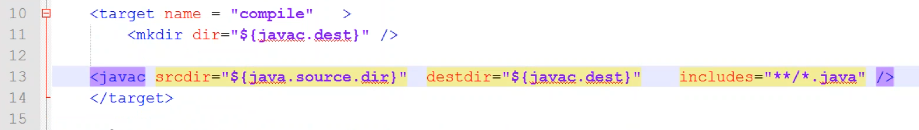
Web application is a collection of static & dynamic content (\*.class/\*.jar & html/css/jps/audio/video)

html/css/jps/audio/video are static content. We no need to compile them. By default, they are in binary format

in web application, we compile only dynamic files but when comes to packaging, we do it on dynamic + static files

every task in ant have n no of attributes

in javac, if we add includes, it will include the files which we mention. For example, in our directory there might be .txt etc. and if we try to compile the files in directory, it might take .txt also. So better mention .java in includes attribute as below



If we write a program which depends on n no of variables like class or jar files. So, if we try to execute our program with some external packages, it might throw an error as unidentified package because we didn’t import them

We have to create some path variable to import them. This path variable we have to reference to all external libraries and we need to use the path to compile

For that we need to create a path variable, path variable can be used anywhere in build.xml

We can give any name as path id. And in fileset dir., we need to give the location

Also we need to give the path in compilation also, as classpathref as below



For web applications, we have a file called web.xml which is very important, and we need to add path to that

Below is the example to create a war file



External tasks: