# Introduction to Ant

James Brucker

## What is Ant?

- Ant is a *build tool* to build software according to a set of rules.
- Idea is similar to Make, but "targets" are more complex.
  - Actions are "tasks" -- there many predefined tasks
  - "javac" - task to compile a Java source tree
- Ant targets and tasks defined in **`build.xml`** file
- Project Home: **`http://ant.apache.org/`**
- Open source, Apache License

# Example of Using Ant

This example is for a project with an Ant `build.xml` file.
First we ask for a list of targets:

```
cmd>  ant -p
Buildfile:   /home/jim/workspace/demo-ci/build.xml
Main targets:

 clean         Delete build products and build directory
 compile       Compile source code
 deps          Install JUnit jars. Needed for Travis CI
 init          Create output directories if they don't exist
 test          Run unit tests
Default target: test
```

Note:  the description of each target is written by the programmer in build.xml.  Some build files may not have descriptoins.

# Example of Using Ant (2)

There is "compile" target, so let's try it:

```
cmd>  ant compile

init:
    [mkdir] Created dir: demo-ci/bin
    [mkdir] Created dir: demo-ci/bin/test

compile:
    [javac] Compiling 2 source files to demo-ci/bin

BUILD SUCCESSFUL
```

Ant performs the "init" target

Ant performs "compile"

# Example of Using Ant (3)

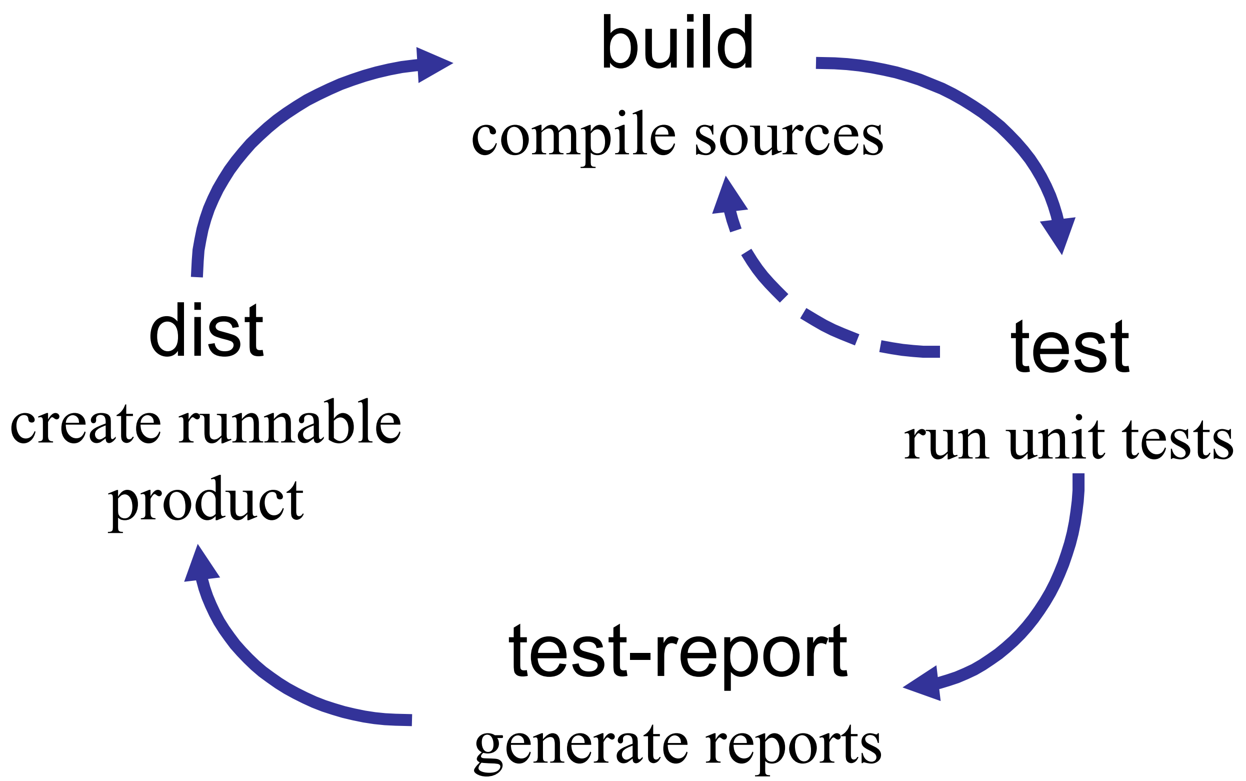
Since "compile" succeeded, we can try running tests.

```
cmd>  ant test

init:
compile:

test:
    [junit] Running demo-ci/bin/test/ArrayMathTest
    [junit] Tests run: 8, Failures: 2, Errors: 0,
            Skipped: 0, Time elapsed: 0.026 sec
    [junit] Test ArrayMathTest FAILED

BUILD SUCCESSFUL
```

"test" requires the "init" and "compile" targets. These targets are up-to-date, so Ant does nothing

# Typical Development Cycle

The typical work cycle for building a project with Ant is:



## Ant: a modern "make"

- Makefile designed for humans to read and edit. But hard for computer progams to process.
- Make targets are fairly low-level. Originally for C code:

```
# build object files from C source file
%.o:  %.c
   ${CC} -c ${CFLAGS} $<
```

- Ant provides lots of "tasks" that eliminate need to write low-level rules -- just specify parameters for the task.
  - Example: `<javac srcdir="src" destdir="bin" />`
  - Will conditionally compile all files in `src` *and* subdirs.
- Ant uses XML for rules: easier for software to read and write.

# Installing Ant

Windows:

- Download from http://ant.apache.org/
- Unzip to a convenient directory -- avoid path with spaces!
  I use: C:\lib\ant
- Add `antdir\bin` to the PATH. I use:

```
ANT_HOME=D:\lib\ant
PATH=%PATH%;%ANT_HOME%\bin
```

Ubuntu Linux:

"`apt-get install ant`" will install the GNU Java and lots of other packages. ***Don't do it!***

Download Ant from ant.apache.org, unzip (/opt or /usr/local/bin). This way you can use your ownJRE.

# Test the Installation

```
cmd> ant -help

ant [options] [target [target2 [target3] ...]]
Options:
  -help, -h                 print this message
  -version                  print the version and exit
  -quiet, -q                be extra quiet
  -verbose, -v              be extra verbose
  -logfile <file>           use given file for log
  -buildfile <file>         use given buildfile
  -f <file> | -file <file>         ''
  -D<property>=<value>      use value for given property
  -keep-going, -k           execute all targets that do not
                            depend on failed target(s)
  -propertyfile <name>      load all properties from file
```

If you get a "command not found" message, then ant/bin isn't on your PATH.
If java is not found, then the JDK "bin" directory isn't on your PATH.

## Learn Ant

Work through the "Hello World with Ant" tutorial

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

# Sample Application

```
SampleApp/
build.xml   Ant build file
src/        source code
    sample/
        domain/
            City.java
            ...
test/      test code
    sample/
        domain/
           CityTest.java
build/      build products
    classes/  java classes
dist/
    sampleapp.jar
lib/
  *.jar    libraries the project uses
```

# A Simple Ant Build file

- The default build file name is: **`build.xml`**

```
<project name="SampleApp" basedir=".">
    <description>
        Sample Application built with Ant
    </description>
  <!-- classpath for required jar files -->
  <path id="classpath">
    <fileset dir="lib">
      <include name="**/*.jar"/>
    </fileset>
    <pathelement location="build/classes"/>
  </path>
```

## A Simple Ant Build file (2)

- The actual work is defined in "`targets`":

```
<project name="SampleApp" basedir=".">

  <target name="init">
  instructions for "init" job
  </target>
  <target name="build" depends="init">
  instructions for "build" job
  </target>
  <target name="test" depends="build">
  instructions for "test" job
  </target>
```

# Define a "build" task

- This task tells Ant how to compile our program

```
<!-- Compile the java code -->
<target name="build" depends="init"
       description="compile the source" >
  <javac destdir="build/classes" >
    <src path="src"/>
    <classpath refid="classpath"/>
  </javac>
  <!-- compile JUnit tests -->
  <javac debug="true" destdir="build/test">
      <src path="test"/>
      <classpath refid="classpath"/>
   </javac>
</target>
```

# "build" depends on "init" task

- Most projects have an "init" task to create output directories.

```
<!-- initialize build environment -->
<target name="init" description="create dirs">
  <mkdir dir="build"/>  (this is not required)
  <mkdir dir="build/classes"/>
  <mkdir dir="build/test"/>
   <!-- copy property files, .fxml files,
        etc. to the build path -->
   <copy includeemptydirs="false"
         todir="build/classes">
       <fileset dir="src"
          excludes="**/*.launch, **/*.java" />
   </copy>
</target>
```

Ant wildcards

# Test Your Build File

```
cmd>  ant -p

cmd>  ant build
```

```
Buildfile: build.xml
init:
    [mkdir] Created dir: build/classes
     [copy] Copying 2 files to ...

build:
    [javac] Compiling 6 source files to build/classes

BUILD SUCCESSFUL
```

## Use properties instead of strings

- We have used "build/classes", "src", many times in the build file.
- Difficult to maintain and possible typing errors.
- Better to use named constants (properties) for directories:

```
<property name="src.dir" location="src"/>
<property name="build.dir"
          location="build/classes"/>

<target name="build" depends="init" ...>
   <javac srcdir="${src.dir}"
          destdir="${build.dir}"
          includeatruntime="false"/>
```

# Create a "test" task

```
<target name="test" depends="build">
    <junit fork="true" printsummary="on"
           haltonfailure="false">
       <classpath>
          <path refid="classpath"/>
          <pathelement
                location="${test.build.dir}"/>
       </classpath>
       <!-- Where are the JUnit test classes? -->
       <batchtest todir="${test.reports.dir}">
          <fileset dir="${test.build.dir}"
                   includes="**/*Test.class"/>
       </batchtest>
    </junit>
</target>
```

## Tools

- List of Ant tools:

  **`http://ant.apache.org/external.html`**

- Eclipse can "export" an Ant build file, but it contains a lot of Eclipse-specific references that make the build file not very portable.

- Ivy (`http://ant.apache.org/ivy`) is a dependency manager for Ant.
  - Install dependencies (libraries), similar to Maven.
  - But Ivy is lighter weight (more specific targets).

## Resources

- Ant Home: **`http://ant.apache.org`**
- *"Hello World with Ant"* - easy to follow tutorial!

  https://ant.apache.org/manual/tutorial-HelloWorldWithAnt.html
- *Apache Ant Manual*. Installed with ant in the **`ant/docs`** directory. It describes all Ant tasks.
- *Ant: The Definitive Guide*. O'Reilly. Terse, but lots of info.

## More About Tasks

- The following slides describe how to use common Ant task.
- You can skip them.
- Same material is in Ant docs and on Internet.

# Common Ant Tasks

Ant has a large set of built-in tasks, such as:

| | |
|---|---|
| **`<echo ...>`** | output a message |
| **`<mkdir ...>`** | create a directory (if it doesn't exist) |
| **`<copy ...>`** | copy a file, directory, or tree |
| **`<javac ...>`** | compile Java files |
| **`<jar ...>`** | create a jar file |
| **`<junit ...>`** | run JUnit tests |

## <property name="src" value="...">

- Defines a property for use in the build script
- To access value of a property use: ${propertyname}.
- Properties a) avoid duplication, b) clarify the build file, c) make it more portable

Example:

```
<property name="src.dir" value="src"/>
<javac ...>
  <src path="${src.dir}"/>
</javac>
```

## Using External Properties

- Ant can read all properties from a plain-text properties file.

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

- Can also use system environment vars as properties!
- Prefix environment variables with a "env."

```
<property environment="env"/>
<echo message=
       "CLASSPATH is ${env.CLASSPATH}"/>
<echo message=
       "JAVA_HOME is ${env.JAVA_HOME}"/>
```

# <copy file="*pattern*" tofile="..."/>

- Copies a file or set of files to another location.
- Does not overwrite existing files if they are newer than the source file (unless you specify that you want it to overwrite).

Copy a single file.

```
<copy file="${src.dir}/myfile.txt"
      tofile="${target.dir}/mycopy.txt"/>
```

## <copy todir="..."> copy sets of files

- Copy files from one directory to another, omit any java source files.

```
<copy todir="${dest.dir}" >
    <fileset dir="src">
       <exclude name="**/*.java"/>
    </fileset>
</copy>
```

- Copy all files from the directory “../backup/” to “src_dir”. Replace occurrences of "TITLE" in the files with "Foo".

```
<copy todir="../backup">
    <fileset dir="src_dir"/>
    <filterset>
        <filter token="TITLE" value="Foo"/>
    </filterset>
</copy>
```

## <delete>

- Deletes files, directories, or sets of files.
- Delete a single file.

```
<delete file="/lib/junk.jar"/>
```

- Delete all *.bak files from this directory and sub-directories.

```
<delete>
    <fileset dir="." includes="**/*.bak"/>
</delete>
```

- Delete the build directory and everything in it.

```
<delete includeEmptyDirs="true">
      <fileset dir="build"/>
</delete>
```

# <echo>

Display a message on terminal.

- Display a one-line message:

```
<echo message="Hello Ants" />
```

```
[echo] Hello Ants
```

- Display many lines of text:

```
<echo>
Usage:  ant target
clean   - delete compiler output files
build   - compile source code
dist    - create a distribution
</echo>
```

## <mkdir dir="..."/>

- Create a directory.

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

- Creates a subdirectory named "jars" in the location specified by the "`dist.dir`" property.

```
<mkdir dir="${dist.dir}/jars"/>
```

## <javac>

- Compiles Java source code.
- Attempts to analyze source such that up to date `.class` file are not recompiled.

Example: Compile all java source files under ${src.dir} and put the .class files in the ${build.classes} directory. Include debugging information in the .class files.

```
<javac srcdir="${src}"
    destdir="${build.dir}"
    classpath="mylib.jar"
    debug="true"/>
```

## <javac ...> (2)

- You can specify additional source directories and further restrict which files are compiled using `include` and `exclude`.

```
<javac destdir="${build}"
  classpath="xyz.jar" debug="on">
  <src path="${src}"/>
  <src path="${src2}"/>
  <include name="package/p1/**"/>
  <include name="package/p2/**"/>
  <exclude name="package/p1/test/**"/>
</javac>
```

## <jar ...>

- Creates a JAR file from a set of files or updates an existing JAR.
- Will automatically supply a manifest file for the JAR or use one you specify.

Example: make a jar file including all files in build/classes

```
<jar jarfile="${dist}/lib/myapp.jar"
     basedir="${build}/classes"/>
```

## <jar ...>

- Create a JAR file from all the files in ${build}/classes and ${src}/resources. (two sets of files)
- Any files named `*Test.class` in the build directory are not included in the JAR.

```
<jar jarfile="${dist}/lib/myapp.jar">

<fileset dir="${build}/classes"

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

<fileset dir="${src}/resources"/>

</jar>
```

## <javadoc>

- Creates Javadocs from Java source code files.

Example: Build Javadoc only for the packages beginning with "`org.ske...`" in the `${src}\` directory.

```
<javadoc packagenames="org.ske.*"
         sourcepath="${src}"
         destdir="${doc}/api"/>
```

This command will search all subdirectories of ${src} for *.java files.

# <java>

- Invoke a Java program from within an Ant build file.
- Can fork a separate process so that a System.exit() does not kill the Ant build.

```
<java classname="test.Main">
  <arg value="some-arg-to-main"/>
  <classpath>
    <pathelement location="test.jar"/>
    <pathelement
        path="${java.class.path}"/>
  </classpath>
</java>
```

## <java>

Invoke a class named test.Main in a separate Java VM.  The Java VM is invoked using the options:

-Xrunhprof:cpu=samples,file=log.txt,depth=3

to request profiling.

```
<java classname="test.Main" fork="yes">
  <sysproperty key="DEBUG" value="true"/>
  <arg value="-h"/>
  <jvmarg value=
   "-Xrunhprof:cpu=samples,file=log.txt,depth=3"/>
</java>
```

# More Ant Tasks

- The Apache Ant distribution includes more than 50 **core** tasks and many **optional** tasks.
- Examples: zip, gzip, war (create a war file),
- Many tasks correspond to standard Linux commands, like mkdir, copy, move.
- You can write your own Ant tasks using <taskdef />.
- See Ant manual (ant/docs directory) for how to use each task.