

An Introduction to Ant



Overview

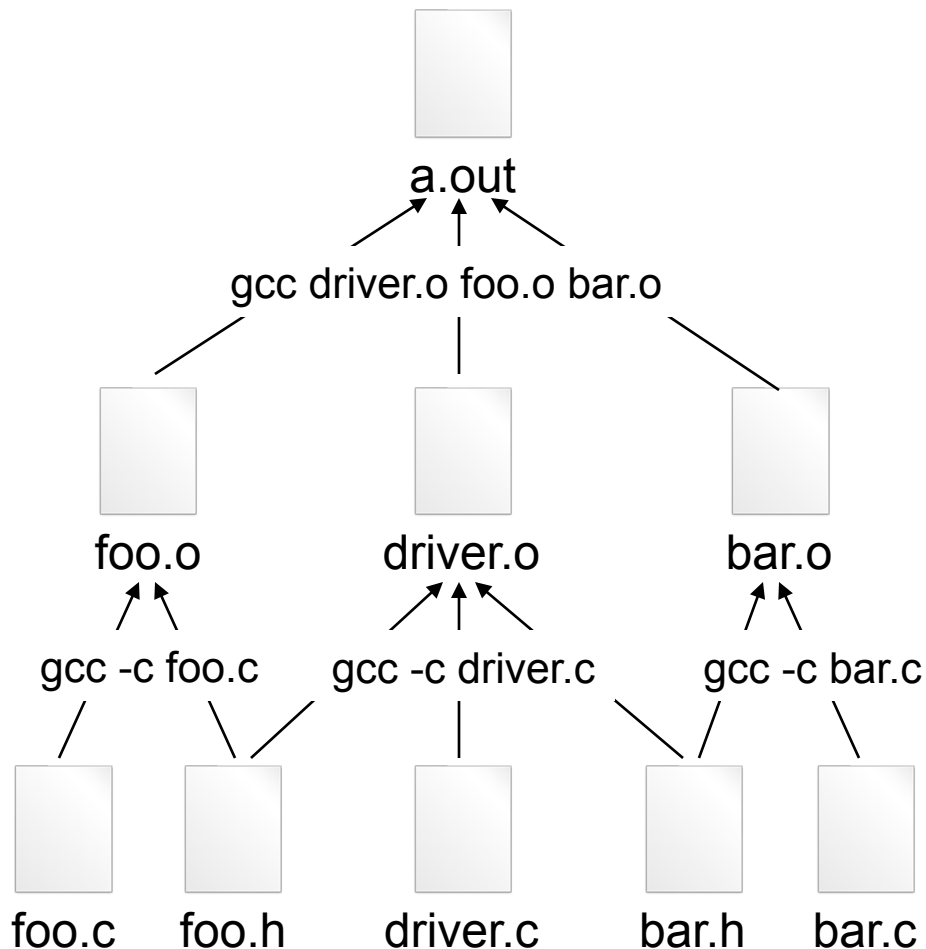
- What is Ant?
- Installing Ant
- Anatomy of a build file
 - Projects
 - Properties
 - Targets
 - Tasks
- Example build file
- Running a build file

What is Ant?

- Ant is a Java based tool for automating the build process
- Similar to make but implemented using Java
 - Platform independent commands (works on Windows, Mac & Unix)
- XML based format
 - Avoids the dreaded tab issue in make files
- Easily extendable using Java classes
- Ant is an open source (free) Apache project

Automating the Build (C & make)

- The goal is to automate the build process



```
a.out: driver.o foo.o bar.o
gcc driver.o foo.o bar.o
driver.o: driver.c foo.h bar.h
gcc -c driver.c
foo.o: foo.c foo.h
gcc -c foo.c
bar.o:
gcc -c bar.c
```

```
linux3[1]% make
gcc -c driver.c
gcc -c foo.c
gcc -c bar.c
gcc driver.o foo.o bar.o
linux3[2]%
```

Installing Ant

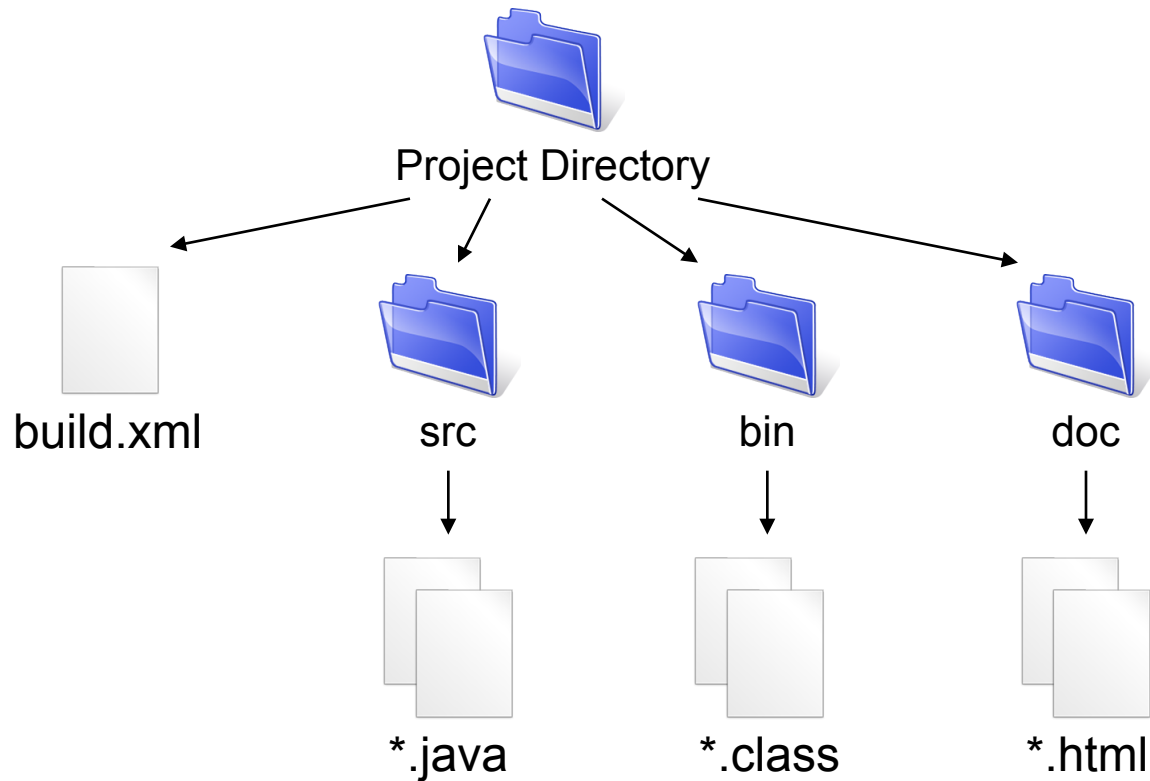
- Ant can be downloaded from...
 - <http://ant.apache.org/>
- Ant comes bundled as a zip file or a tarball
- Simply unwrap the file to some directory where you want to store the executables
 - I typically unwrap the zip file into C:\Program Files, and rename to C:\Program Files\ant\
 - This directory is known as ANT_HOME

Ant Setup

- Set the ANT_HOME environment variable to where you installed Ant
- Add the ANT_HOME/bin directory to your path
- Set the JAVA_HOME environment variable to the location where you installed Java
- Setting environment variables
 - Windows: right click My Computer → Properties → Advanced → Environment Variables
 - UNIX: shell specific settings

Project Organization

- The following example assumes that your workspace will be organized like so...



Anatomy of a Build File

- Ant's build files are written in XML
 - Convention is to call file build.xml
- Each build file contains
 - A project
 - At least 1 target
- Targets are composed of some number of tasks
- Build files may also contain properties
 - Like macros in a make file
- Comments are within `<!-- -->` blocks

Projects

- The [project tag](#) is used to define the project you wish to work with
- Projects tags typically contain 3 attributes
 - name – a logical name for the project
 - default – the default target to execute
 - basedir – the base directory for which all operations are done relative to
- Additionally, a description for the project can be specified from within the project tag

Build File

```
<project name="Sample Project" default="compile" basedir=". ">  
  
  <description>  
    A sample build file for this project  
  </description>  
  
</project>
```

Properties

- Build files may contain constants (known as properties) to assign a value to a variable which can then be used throughout the project
 - Makes maintaining large build files more manageable
- Projects can have a set of properties
- Property tags consist of a name/value pair
 - Analogous to macros from make

Build File with Properties

```
<project name="Sample Project" default="compile" basedir=". ">
```

```
<description>
```

```
  A sample build file for this project
```

```
</description>
```

```
<!-- global properties for this build file -->
```

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

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

```
<property name="doc.dir" location="doc"/>
```

```
</project>
```

Targets

- The [target tag](#) has the following required attribute
 - name – the logical name for a target
- Targets may also have optional attributes such as
 - depends – a list of other target names for which this task is dependant upon, the specified task(s) get executed first
 - description – a description of what a target does
- Like make files, targets in Ant can depend on some number of other targets
 - For example, we might have a target to create a jarfile, which first depends upon another target to compile the code
- A build file may additionally specify a default target

Build File with Targets

```
<project name="Sample Project" default="compile" basedir=". ">  
  
...  
  
<!-- set up some directories used by this project -->  
<target name="init" description="setup project directories">  
</target>  
  
<!-- Compile the java code in src dir into build dir -->  
<target name="compile" depends="init" description="compile java sources">  
</target>  
  
<!-- Generate javadocs for current project into docs dir -->  
<target name="doc" depends="init" description="generate documentation">  
</target>  
  
<!-- Delete the build & doc directories and Emacs backup (*~) files -->  
<target name="clean" description="tidy up the workspace">  
</target>  
  
</project>
```

Tasks

- A task represents an action that needs execution
- Tasks have a variable number of attributes which are task dependant
- There are a number of build-in tasks, most of which are things which you would typically do as part of a build process
 - Create a directory
 - Compile java source code
 - Run the javadoc tool over some files
 - Create a jar file from a set of files
 - Remove files/directories
 - And many, many others...
 - For a full list see: <http://ant.apache.org/manual/coretasklist.html>

Initialization Target & Tasks

- Our initialization target creates the build and documentation directories
 - The [mkdir task](#) creates a directory

```
<project name="Sample Project" default="compile" basedir=". ">
...
<!-- set up some directories used by this project -->
<target name="init" description="setup project directories">
  <mkdir dir="${build.dir}"/>
  <mkdir dir="${doc.dir}"/>
</target>
...
</project>
```


Compilation Target & Tasks

- Our compilation target will compile all java files in the source directory
 - The [javac task](#) compiles sources into classes
 - Note the dependence on the init task

```
<project name="Sample Project" default="compile" basedir=".">
...
<!-- Compile the java code in ${src.dir} into ${build.dir} -->
<target name="compile" depends="init" description="compile java sources">
  <javac srcdir="${source.dir}" destdir="${build.dir}"/>
</target>
...
</project>
```

Javadoc Target & Tasks

- Our documentation target will create the HTML documentation
 - The [javadoc task](#) generates HTML documentation for all sources

```
<project name="Sample Project" default="compile" basedir=".">
...
<!-- Generate javadocs for current project into ${doc.dir} -->
<target name="doc" depends="init" description="generate documentation">
  <javadoc sourcepath="${source.dir}" destdir="${doc.dir}"/>
</target>
...
</project>
```

Cleanup Target & Tasks

- We can also use ant to tidy up our workspace
 - The [delete task](#) removes files/directories from the file system

```
<project name="Sample Project" default="compile" basedir=". ">
...
<!-- Delete the build & doc directories and Emacs backup (*~) files -->
<target name="clean" description="tidy up the workspace">
  <delete dir="${build.dir}"/>
  <delete dir="${doc.dir}"/>
  <delete>
    <fileset defaultexcludes="no" dir="${source.dir}" includes="**/*~"/>
  </delete>
</target>
...
</project>
```

Completed Build File (1 of 2)

```
<project name="Sample Project" default="compile" basedir=". ">

  <description>
    A sample build file for this project
  </description>

  <!-- global properties for this build file -->
  <property name="source.dir" location="src"/>
  <property name="build.dir" location="bin"/>
  <property name="doc.dir" location="doc"/>

  <!-- set up some directories used by this project -->
  <target name="init" description="setup project directories">
    <mkdir dir="${build.dir}"/>
    <mkdir dir="${doc.dir}"/>
  </target>

  <!-- Compile the java code in ${src.dir} into ${build.dir} -->
  <target name="compile" depends="init" description="compile java sources">
    <javac srcdir="${source.dir}" destdir="${build.dir}"/>
  </target>
```

Completed Build File (2 of 2)

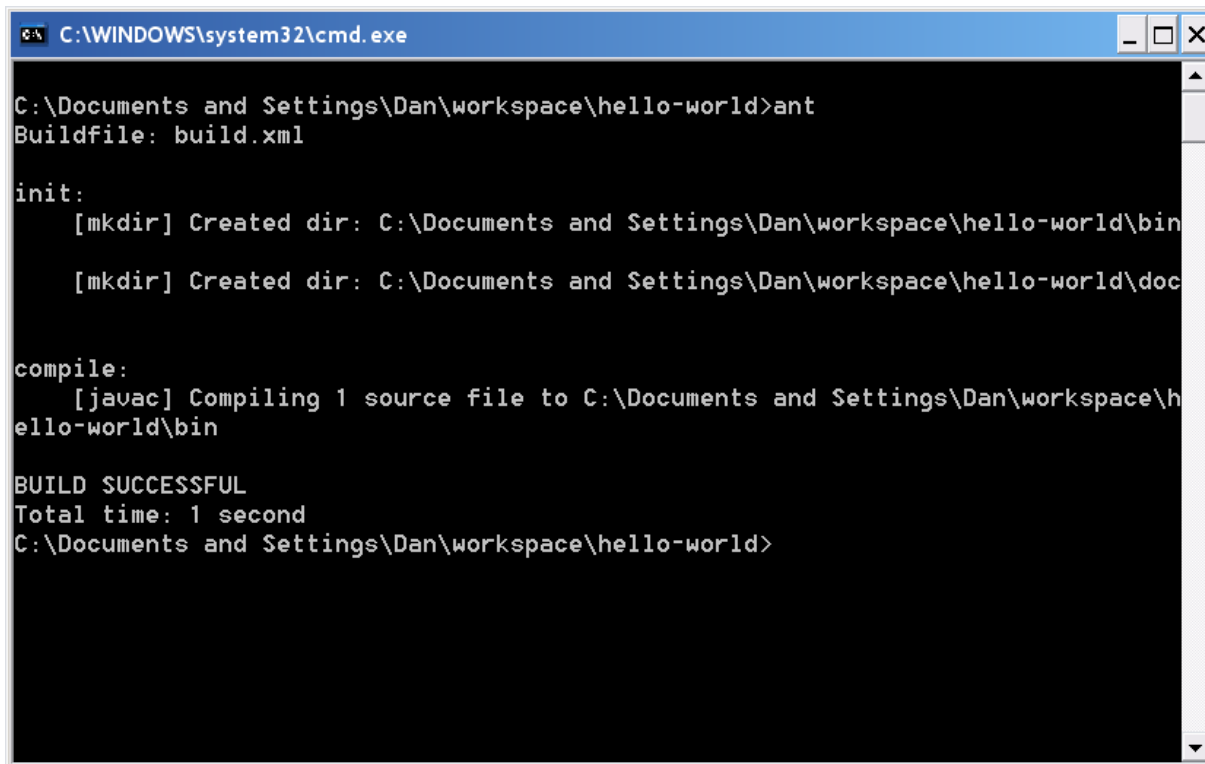
```
<!-- Generate javadocs for current project into ${doc.dir} -->
<target name="doc" depends="init" description="generate documentation">
  <javadoc sourcepath="${source.dir}" destdir="${doc.dir}"/>
</target>

<!-- Delete the build & doc directories and Emacs backup (*~) files -->
<target name="clean" description="tidy up the workspace">
  <delete dir="${build.dir}"/>
  <delete dir="${doc.dir}"/>
  <delete>
    <fileset defaultexcludes="no" dir="${source.dir}" includes="**/*~"/>
  </delete>
</target>

</project>
```

Running Ant – Command Line

- Simply cd into the directory with the build.xml file and type ant to run the project default target
- Or, type ant followed by the name of a target



```
C:\WINDOWS\system32\cmd.exe

C:\Documents and Settings\Dan\workspace\hello-world>ant
Buildfile: build.xml

init:
  [mkdir] Created dir: C:\Documents and Settings\Dan\workspace\hello-world\bin
  [mkdir] Created dir: C:\Documents and Settings\Dan\workspace\hello-world\doc

compile:
  [javac] Compiling 1 source file to C:\Documents and Settings\Dan\workspace\hello-world\bin

BUILD SUCCESSFUL
Total time: 1 second
C:\Documents and Settings\Dan\workspace\hello-world>
```

Running Ant – Eclipse

- Eclipse comes with out of the box support for Ant
 - No need to separately download and configure Ant
- Eclipse provides an Ant view
 - Window → Show View → Ant
- Simply drag and drop a build file into the Ant view, then double click the target to run

