Announcements

- Project 0 is out. Project submission instruction is online. due on Sept 17 @ 11 pm.
- TA office hours posted
- Lecture notes (with answers) uploaded

CMSC 341

CVS / Ant

CVS

CVS – why do you need it?

- Concurrent version control
- Benefits:
  - Avoids disaster caused by deletion; recover is easy
  - Allows team work
  - Keeps a record of the changes made over time
  - Supports multiple software releases
  - Is a time machine
  - Is location-independent

What is CVS?

- Concurrent Versioning System (CVS) is a place to store all the various revisions of the stuff you write while developing an application.
  - Open source
  - Easy to install and use
  - Simple command line client
  - Wide integration in a lot of development tools
  - Project 0 and project 0 only in this course
- Resources:
  - Pragmatic Version Control using CVS (on our schedule page.)

CVS Terminology

- **Repository** – the place where resources (files) are stored
- **Checkout** – copy resources from the repository and create a working copy
- **Checkin/Commit** – place resources from your working copy into the repository
- **Add** – place a resource under version control
- **Remove** – delete a resource from version control
- **Update** – pull down changes from the repository into your working copy

CVS commands

- cvs add <file or dir name>
- cvs update .
- cvs checkout .
- cvs remove <file or dir name>
- cvs commit –m “say something here.”
- cvs log
- cvs diff –r 1.1 r 1.2 <file or dir name>
- cvs update –j 1.3 –j 1.2 <file name>
- Resolve conflict…..

What should NOT be stored?

- Generated files
  - .o, doc

Ant

What is Ant?

- Ant is a Java based tool for automating the build process
- Platform independent commands (works on Windows, Mac & Unix)
- XML based format
- Easily extendable using Java classes
- Ant is an open source (free) Apache project

- ***Ant files used in this course require the package directory structure.***

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 to which the ANT file applies
- Projects tags typically contain 3 attributes
  - name – a logical name for the project
  - default – the default target to execute
  - basedir – the base directory relative to which all operations are performed
- Additionally, a description for the project can be specified from within the project tag

Project tag

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

  <description>
    A sample build file for this project
    Recall that “.” (dot) refers to the current directory
  </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 and easily changeable
- Projects can have a set of properties
- Property tags consist of a name/value pair
  - Use the property names throughout the build file
  - The value is substituted for the name when the build file is "executed"

Build File with Properties

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

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

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

</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 built-in tasks, most of which are things which you would typically do as part of a build process
  - mkdir - create a directory
  - javac - compile java source code
  - java - execute a Java .class file
  - javadoc - run the javadoc tool over some files
  - And many, many others…
    - For a full list see: http://ant.apache.org/manual/tasksoverview.html

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
- 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
- Targets contain a list of tasks to be executed

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">
        <!-- list of tasks to be executed -->
  </target>

  <!-- Compile the java code in src dir into build dir -->
  <target name="compile" depends="init" description="compile java sources">
        <!-- list of tasks to be executed -->
  </target>

  <!-- Generate javadocs for current project into docs dir -->
  <target name="doc" depends="init" description="generate documentation">
        <!-- list of tasks to be executed -->
  </target>

<!-- Execute main in the specified class under ${build.dir} -->
  <target name=”run" depends=“compile” description=”run the application">
        <!-- list of tasks to be executed -->
  </target>

  <!-- Delete the build & doc directories and Emacs backup (*~) files -->
  <target name="clean" description="tidy up the workspace">
        <!-- list of tasks to be executed -->
  </target>
</project>
```

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>
```

Run Target & Tasks

- Our run target will execute main in the fully specified class
  - Typically dependent on the compile task

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

  ...

  <!-- Execute main in the fully qualified name under ${build.dir} -->
  <target name=”run" depends=”compile" description=“run the application">
        <java directory=“${build.dir}” classname=“${main.class}” fork=“yes”>
                <arg line=“${args}” />
        </java>
  </target>

  ...

</project>
```

Running Ant – Command Line

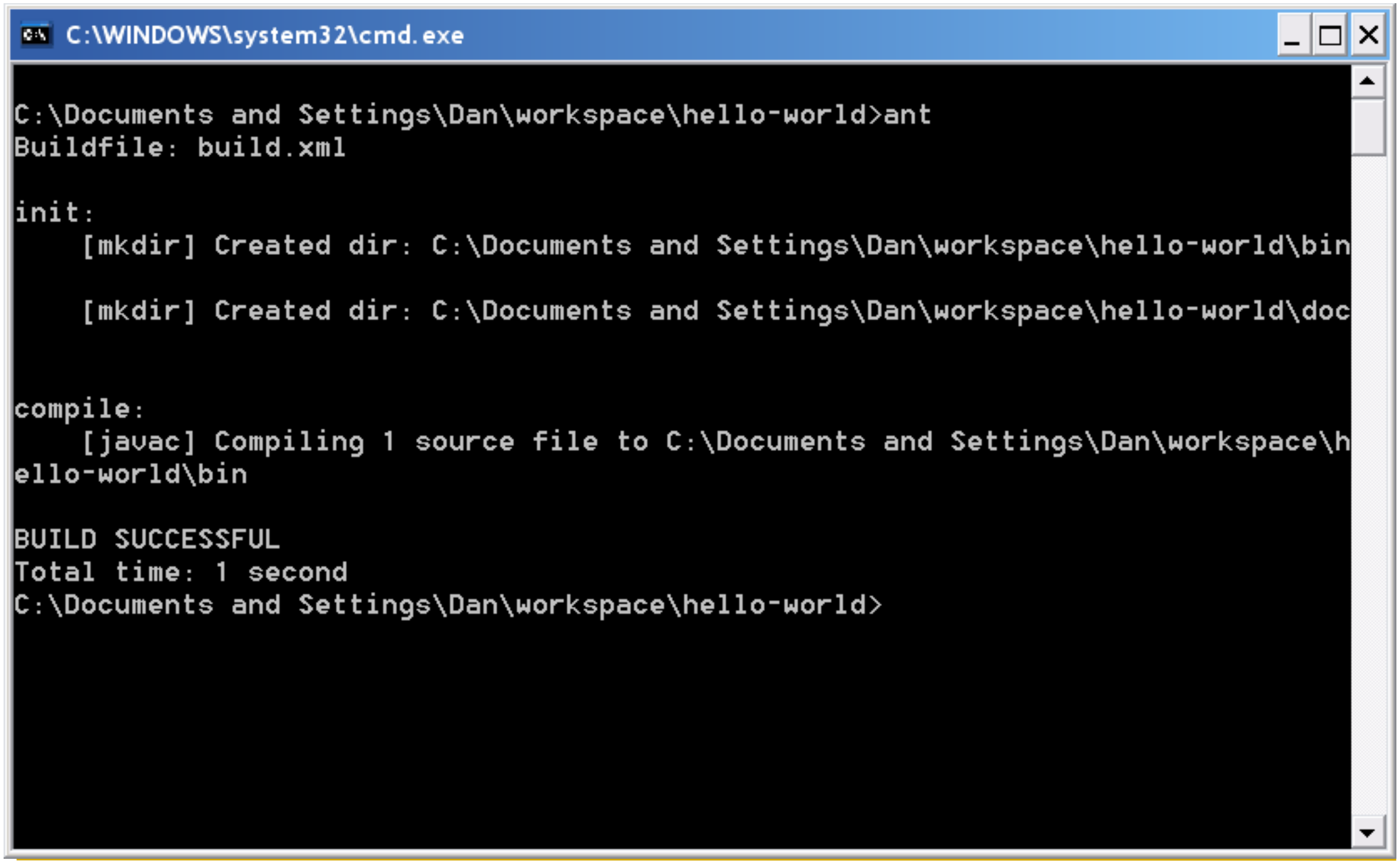
- Move into the directory which contains the build.xml file
- Type *ant* followed by the name of a target

```
unix> ant run
unix> ant compile
```

- Type *ant* at the unix prompt to run the project's default target -- see screen shot on next page

```
unix> ant
```

Ant screen snapshot



```
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\h
ello-world\bin

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