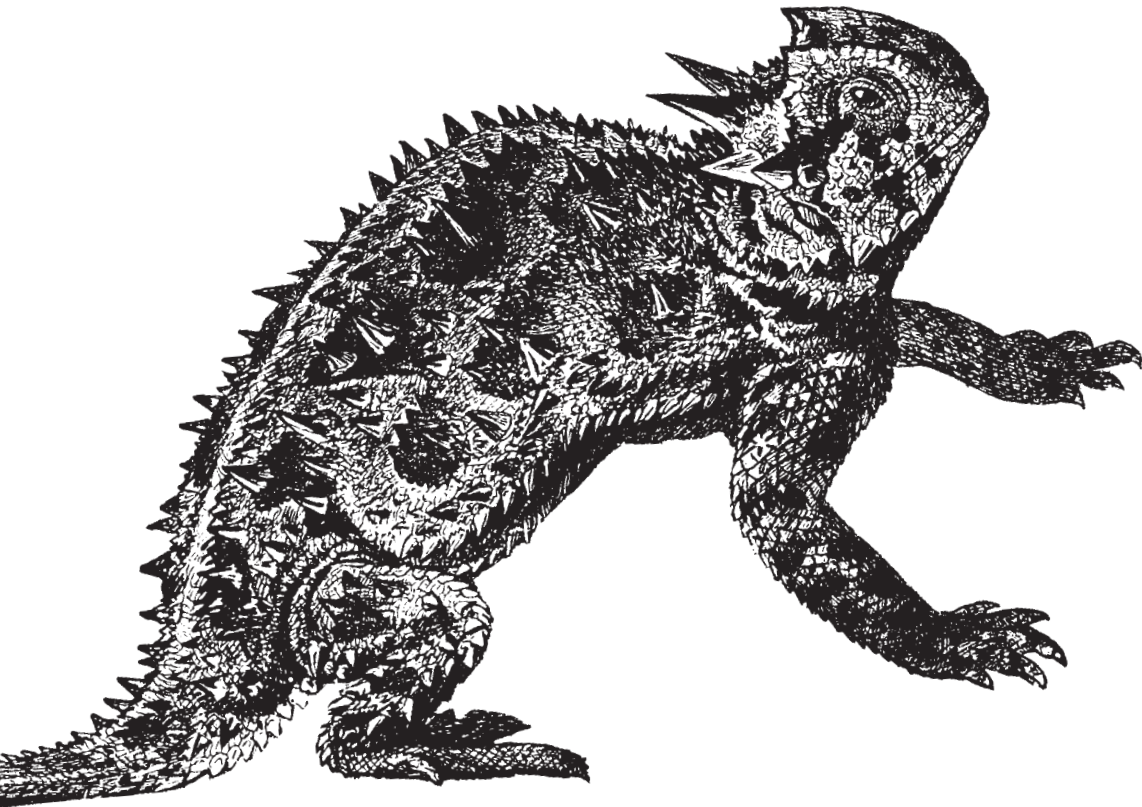


Complete Build Management for Java

2nd Edition
Covers Ant v1.6

Ant

The Definitive Guide



O'REILLY®

Steve Holzner

Getting Source Code from CVS Repositories

Up to this point, you've been working solo with Ant, but—as with any major build tool—Ant can be used in team environments. There's a lot of support built in for the Concurrent Version System (CVS) in Ant, and this chapter is all about making code sharing in teams with CVS happen.

Source Control and Ant

When you work in teams, you have to coordinate your efforts. That means discussing and planning, but even with the best of intentions, you can still end up with unintentional conflicts. You may have made some brilliant changes to the code, only to find them wiped out by mistake when another programmer uploads his own version of the same file.

Source control helps prevent these problems by controlling access to code and by maintaining a history of the changes made so things aren't destroyed unintentionally. Storing a history of your code is powerful; you can compare a new (buggy) file against an older one, and you can revert to a previous version in case things have gone bad.

Ant has several source control tasks, shown in Table 6-1.

Table 6-1. Source control tasks

Task name	Description
clearcase	Tasks for ClearCase <code>cleartool</code> <code>checkin</code> , <code>checkout</code> , <code>uncheckout</code> , <code>update</code> , <code>lock</code> , <code>unlock</code> , <code>mk1btype</code> , <code>rmtype</code> , <code>mklabel</code> , <code>mkattr</code> , <code>mkdir</code> , <code>mkelem</code> , and <code>mkbl</code> commands
Continuus/Synergy tasks	Tasks for Continuus <code>ccmcheckin</code> , <code>ccmcheckout</code> , <code>ccmcheckintask</code> , <code>ccmreconfigure</code> , and <code>ccmcreateTask</code> commands
cvs	Specifies how to work with packages and modules retrieved from a CVS repository

Table 6-1. Source control tasks (continued)

Task name	Description
cvschangelog	Creates change reports from a CVS repository
cvspass	Adds entries to a <code>.cvspass</code> file
cvstagdiff	Creates an XML-formatted report of the changes between two tags or dates recorded in a CVS repository
Microsoft Visual Sourcesafe tasks	Tasks for Visual SourceSafe <code>vssget</code> , <code>vsslabel</code> , <code>vsshistory</code> , <code>vsscheckin</code> , <code>vsscheckout</code> , <code>vssadd</code> , <code>vsscp</code> , and <code>vsscreate</code> commands
perforce	Tasks for Perforce <code>p4sync</code> , <code>p4change</code> , <code>p4edit</code> , <code>p4submit</code> , <code>p4have</code> , <code>p4label</code> , <code>p4counter</code> , <code>p4reopen</code> , <code>p4revert</code> , and <code>p4add</code> commands
pvc	Retrieves and handles source code from a PVCS repository
sourceoffsite	Tasks for SourceOffSite <code>sosget</code> , <code>soslabel</code> , <code>soscheckin</code> , and <code>soscheckout</code> commands
starteam	Tasks for StarTeam <code>stcheckout</code> , <code>stcheckin</code> , <code>stlabel</code> , and <code>stlist</code> commands

Though Ant lets you work with various source control systems, most of its support revolves around CVS, which is used throughout this chapter. CVS is an open source project that started as a set of Unix shell scripts in 1986 and came into its own with dedicated software in 1989. Support for CVS is available on many operating systems: Unix, Linux, Windows, Mac, and others. For the full CVS story, look at <http://www.cvshome.org>.



To work with CVS using Ant, you need access to a CVS server. Most Linux and Unix installations come with a built-in CVS server. To test if you have a working CVS installation, type `cvs --help` at the prompt; you should see a list of help items. If you can't find a CVS server, you can download what you need from <http://www.cvshome.org>. Many CVS servers are available for Windows, such as CVSNT, available for free from <http://www.cvsnt.org>. To install CVSNT, download the executable file and run it.

The idea behind CVS, as with any repository software, is to manage and record changes to source code. What corresponds to a project for Ant is a *module* in CVS. Modules are represented by directories in CVS; the files you share are stored in the CVS *repository*. When you retrieve a file from the repository, you *check the file out*. After you've modified the file, you *commit* the file, checking it back in and sending those changes to the repository. If you want to refresh your own copy of a file, you *update* it from the repository.

Because each file must be independently tracked, CVS gives the individual files a version number automatically. Each time a file is committed, its version number is

incremented. When you commit files to the repository, they'll get a new version number as well.

Using the cvs task, you communicate with the CVS server using CVS commands, which appear in Table 6-2.



Read more about these commands in the CVS guide at https://www.cvshome.org/docs/manual/cvs-1.11.7/cvs_16.html.

Table 6-2. CVS commands

CVS command	Does this
add	Specifies you want to add a new file or directory to the CVS repository
admin	Lets you work with administrative commands
annotate	Specifies the revision where each line was modified
authserver	Specifies authentication mode for the server
chacl	Specifies the access list for a directory
checkout	Checks out source code from the repository
chown	Changes the owner of a directory in the repository
commit	Checks source code files into the repository
diff	Displays the differences between source code revisions
edit	Specifies you want to edit a file
editors	Specifies you want to watch who has been editing a particular file
export	Exports source code from a CVS repository (similar to checkout)
history	Displays the CVS repository access history
import	Imports source code into a CVS repository
info	Displays information about the CVS repository and its supported protocols
init	Creates a CVS repository and initializes it
log	Displays information on file history
login	Asks for a password, if needed
logout	Logs out of a server
ls	Lists the files in the CVS repository
lsacl	Lists the CVS directories access control list
passwd	Sets a user's CVS password
rannotate	Displays the revision in which source lines were modified
rdiff	Creates patch files by comparing two files and outputting a file that be used to update one into the other
release	Specifies that a module will not be used anymore
remove	Removes an item from a CVS repository
rlog	Displays CVS history information for a module

Table 6-2. CVS commands (continued)

CVS command	Does this
rtag	Adds a tag to a CVS module
server	Lets you set server modes for access
status	Displays checked-out file information
tag	Adds a tag to checked-out files
unedit	Undoes an edit command that's been executed
update	Updates local copies of a file with those in the CVS repository
version	Shows the CVS version
watch	Watches a file or files
watchers	Displays which users are watching a file or files

The first step in working with CVS is to log into the CVS server, typically done with the `cvspass` task.

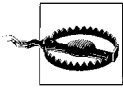
Logging In

You use the `cvspass` task to log into a CVS server to get access to the code stored in the CVS repository. This task adds entries to a `.cvspass` file, which has the same affect as a CVS login command. When a `.cvspass` file has been created, subsequent logins will get the needed data from this file, and you won't have to supply a password again.

The values you assign to the attribute named `cvsroot` use the same format of strings that appear in a CVS `.cvspass` file, which specifies the protocol type, username, server, and repository location. For example, using the `pserver` protocol with a user named Steven, a server named STEVE, and a repository location of `/home/steven/repository`, `cvspass` would look like:

```
<?xml version="1.0"?>
<project default="main" basedir=".">
  <property name="cvs.dir" value="project" />
  <target name="main" >
    <cvspass cvsroot=":pserver:steven@STEVE:/home/steven/repository"
      password="opensesame" />
    .
    .
    .
  </target>
</project>
```

The CVS-related tasks can read the CVS root value from the `cvsroot` attribute, if they support that attribute, or from the `CVSROOT` environment variable.



In Windows, when your username includes a space or spaces, you might run into problems with the `cvsrc` attribute. In that case, assign a value to the `CVSR` environment variable instead (e.g., `C:\ant\ch06>set CVSR=pserver:Steven Holzner@STEVE:/home/steven/repository`) and then use `cvspass` or other CVS-related tasks in your build file normally.

The attributes of the `cvspass` task appear in Table 6-3.

Table 6-3. Attributes for the `cvspass` task

Attribute	Description	Required	Default
<code>cvsrc</code>	Specifies the CVS repository you want to add an entry for	Yes	
<code>passfile</code>	Specifies the password file you want to add the entry to	No	<code>~/cvspass</code>
<code>password</code>	Specifies the password you want to be added to the password file	Yes	

Working with the Server

The `cvsrc` task lets you interact with the CVS server after you've logged in. The attributes of this task appear in Table 6-4; to use this task, the `cvsrc` command must work on the command line (i.e., the `cvsrc` binary must be in your path).

Table 6-4. The `cvsrc` attributes

Attribute	Description	Required	Default
<code>append</code>	Specifies whether you want to append output when redirecting text to a file.	No	<code>false</code>
<code>command</code>	Specifies the CVS command you want to execute.	No	<code>checkout</code>
<code>compression</code>	The same as <code>compressionlevel="3"</code> .	No	<code>false</code>
<code>compressionlevel</code>	Specifies the compression level you want to use, via a number between 1 and 9. Any other value sets <code>compression="false"</code> .	No	<code>false</code>
<code>cvsrc</code>	Specifies the <code>CVSR</code> variable.	No	
<code>cvsrc</code>	Specifies the <code>CVS_RSH</code> variable.	No	
<code>date</code>	Specifies that you want to use the most recent revision, as long as it is no later than the given date.	No	
<code>dest</code>	Specifies the directory where you want checked-out files to be placed.	No	The project's basedir.
<code>error</code>	Specifies the file where you want error messages stored.	No	Sends errors to the Ant Log as <code>MSG_WARN</code> .
<code>failonerror</code>	Stops the build if the task encounters an error.	No	<code>false</code>
<code>noexec</code>	Specifies that CVS actions should report only, without changing any files.	No	<code>false</code>

Table 6-4. The cvs attributes (continued)

Attribute	Description	Required	Default
output	Specifies the file to which standard output should be directed.	No	Sends output to the Ant Log as MSG_INFO.
package	Specifies the module you want to check out.	No	
passfile	Specifies a password file you want to have the task read passwords from.	No	~/ .cvspass.
port	Specifies the port used by the task to communicate with the CVS server.	No	2401
quiet	Suppresses messages. This is the same as using -q on the command line.	No	false
reallyquiet	Suppresses all messages. This is the same as using -Q on the command line. Since Ant 1.6.	No	false
tag	Specifies the module to check out by tag name.	No	

This task is designed to pass commands on to CVS verbatim. For example, here's how you'd pass a CVS diff command to the CVS server:

```
<cvs command="diff -u -N" output="diff.txt"/>
```

You can nest `commandline` elements and use the `value` attribute of `argument` elements to pass arguments to the CVS server; you can pass the diff command this way:

```
<cvs output="patch">
  <commandline>
    <argument value="diff"/>
    <argument value="-u"/>
    <argument value="-N"/>
  </commandline>
</cvs>
```

or this way, using the `argument` element's `line` attribute:

```
<cvs output="patch">
  <commandline>
    <argument line="-q diff -u -N"/>
  </commandline>
</cvs>
```

Checking Out Modules

To check out a module from the CVS server, you can use the `cvs` task without specifying a CVS command; the default for the `command` attribute is `checkout`. In Example 6-1, a module named `GreetingApp` is checked out and stored in a directory named `project`.



In this and the following CVS-related build files, you can omit the `cvspass` task if you've stored your password in the `.cvspass` file (which is what `cvspass` does). If you omit `cvspass`, set the `cvsroot` attribute in the `cvs` task, or set the `CVSR00T` environment variable.

Example 6-1. Checking out a CVS module (ch06/checkout/build.xml)

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

<project default="checkout" basedir=".">

  <property name="cvs.dir" value="project" />

  <target name="checkout" >
    <cvspass cvsroot=":pserver:steven@STEVE:/home/steven/repository"
      password="opensesame" />
    <cvs package="GreetingApp" dest="${cvs.dir}" />
  </target>

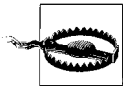
</project>
```

Here's what this build file looks like in action:

```
%ant
Buildfile: build.xml

checkout:
[cvs] Using cvs passfile: /home/.cvspass
[cvs] cvs server: Updating GreetingApp
[cvs] U GreetingApp/.classpath
[cvs] U GreetingApp/.project
[cvs] cvs server: Updating GreetingApp/org
[cvs] cvs server: Updating GreetingApp/org/antbook
[cvs] cvs server: Updating GreetingApp/org/antbook/ch06
[cvs] U GreetingApp/org/antbook/ch06/GreetingClass.java

BUILD SUCCESSFUL
Total time: 2 seconds
```



Before using the build files for this chapter in the downloadable code, make sure you replace the `cvsroot` attribute value or the `CVSR00T` environment variable with an appropriate value for your CVS server.

After running this build file, the project directory will hold the checked-out module, including a CVS `.project` file and a CVS directory, which holds logging and tracking information. You're free to work with the code that's been downloaded, and when you want to commit the project back to the CVS server, specify the same directory you downloaded the project to.

Updating Shared Code

When you want to update your local copy of a module from the CVS repository, you can use the `update` command. You can see how that works in Example 6-2; as before, you can omit the `cvspass` task if your password is in the `.cvspass` file though it causes no harm to leave it in.

Example 6-2. Updating a CVS module `ch06/update/build.xml`

```
<?xml version="1.0"?>
<project default="main" basedir=".>
    <property name="cvs.dir" value="project" />
    <target name="main" depends="login, update">
        <echo>
            Updating...
        </echo>
    </target>
    <target name="login">
        <cvspass cvsroot=":pserver:steven@STEVE:/home/steven/repository"
            password="opensesame" />
    </target>
    <target name="update" depends="login">
        <cvs dest="{cvs.dir}" command="update"/>
    </target>
</project>
```

Here's what you see when running this build file:

```
%ant
Buildfile: build.xml

login:
    [cvs] Using cvs passfile: /home/.cvspass

update:
    [cvs] Using cvs passfile: /home/.cvspass
    [cvs] cvs server: Updating GreetingApp
    [cvs] cvs server: Updating GreetingApp/org
    [cvs] cvs server: Updating GreetingApp/org/antbook
    [cvs] cvs server: Updating GreetingApp/org/antbook/ch06

main:
    [echo]
```

```
[echo]          Updating....
[echo]
```

```
BUILD SUCCESSFUL
Total time: 3 seconds
```

This updates your local copy of a module with what's currently in the CVS repository.

Committing Source Code

After you've made changes to the code in a checked-out module, you can send the revised module back to the CVS repository by setting the `command` attribute to `commit`, as shown in Example 6-3. In this example, the build file commits a new version of a checked-out module, adding the comment "New Version."

Example 6-3. Committing a CVS module `ch06/commit/build.xml`

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

<project default="main" basedir=".">

  <property name="cvs.dir" value="project" />

  <target name="main" depends="login, commit">
    <echo>
      Committing....
    </echo>
  </target>

  <target name="login">
    <cvspass cvsroot=":pserver:steven@STEVE:/home/steven/repository"
      password="opensesame" />
  </target>

  <target name="commit" depends="login">
    <cvs dest="{cvs.dir}/GreetingApp" command="commit -m 'New Version'"/>
  </target>

</project>
```

Here's what this build file gives you when you run it and the CVS server commits the new code:

```
%ant
Buildfile: build.xml

login:

commit:
[cvs] Using cvs passfile: /home/.cvspass
[cvs] cvs commit: Examining .
[cvs] cvs commit: Examining org
```

```
[cvs] cvs commit: Examining org/antbook
[cvs] cvs commit: Examining org/antbook/ch06
[cvs] Checking in org/antbook/ch06/GreetingClass.java;
[cvs] /home/steven/repository/GreetingApp/org
/antbook/ch06/GreetingClass.java,v
<-- GreetingClass.java
[cvs] new revision: 1.5; previous revision: 1.4
[cvs] done
```

```
main:
[echo]
[echo]          Committing...
[echo]
```

```
BUILD SUCCESSFUL
Total time: 1 second
```

Comparing Files

You can compare local files to those in the CVS repository with the CVS `diff` command. For example, say that the module you've been working with, `GreetingApp`, contains `GreetingClass.java`, which holds these contents (presumably committed earlier by you or another developer):

```
package org.antbook.ch06;

public class GreetingClass
{
    public static void main(String[] args)
    {
        System.out.println("No problems here.");
    }
}
```

Then suppose you change the displayed message from “No problems here.” to “No problems at all.” in the local version of the file:

```
package org.antbook.ch06;

public class GreetingClass
{
    public static void main(String[] args)
    {
        System.out.println("No problems at all.");
    }
}
```

The CVS `diff` command finds the difference between your local copy and the server's version. You can see a build file using this command in Example 6-4; in this case, the differences are written to a file named `patch.txt`.

Example 6-4. Finding differences in a CVS module *ch06/diff/build.xml*

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

<project default="main" basedir=". ">

  <property name="cvs.dir" value="project" />

  <target name="main" >
    <cvspass cvsroot=":pserver:steven@STEVE:/home/steven/repository"
      password="opensesame" />
    <cvs command="diff" dest="${cvs.dir}/GreetingApp" output="patch.txt"/>
  </target>

</project>
```

Here's what the build process looks like at work:

```
%ant
Buildfile: build.xml

main:
 [cvs] Using cvs passfile: /home/.cvspass
 [cvs] cvs server: Diffing .
 [cvs] cvs server: Diffing org
 [cvs] cvs server: Diffing org/antbook
 [cvs] cvs server: Diffing org/antbook/ch06
```

```
BUILD SUCCESSFUL
Total time: 1 second
```

In *patch.txt*, the `diff` command caught the difference between the local copy of the file and the version in the CVS repository:

```
Index: org/antbook/ch06/GreetingClass.java
=====
RCS file: /home/steven/repository/GreetingApp/org/antbook/ch06/GreetingClass.java,v
retrieving revision 1.6
diff -r1.6 GreetingClass.java
20c20
<     System.out.println("No problems at all.");
---
>     System.out.println("No problems here.");
```



If you want to create a patch file that you can, with the `patch` utility, update code files with, use the CVS `rdiff` command, not `diff`.

That's how the `cvs` task works; you pass the CVS command, along with any command-line options, in the `command` attribute or a `commandline` element. You can extrapolate from the CVS examples given here to other CVS commands easily.

Getting Version Data

The `cvsversion` task retrieves the version of the CVS client and server. For example, this `cvsversion` element stores the server's version number in the property `cvsserverVersion`, and the client's version in `cvscClientVersion`:

```
<cvsversion cvsroot=":pserver:steven@STEVE:/home/steven/repository"
  password="opensesame"
  cvsserverproperty="cvsserverVersion"
  cvscClientproperty="cvscClientVersion"
/>
```

The attributes for this task appear in Table 6-5.

Table 6-5. The `cvsversion` task's attributes

Attribute	Description	Required	Default
<code>cvscClientproperty</code>	Specifies the name of the property in which you want the version of the <code>cvscClient</code> to be placed	No	
<code>cvsroot</code>	Specifies the <code>CVSROOT</code> variable you want to use	No	
<code>cvsrsh</code>	Specifies the <code>CVS_RSH</code> variable you want to use	No	
<code>cvsserverproperty</code>	Specifies the name of a property where you want the CVS server version to be placed	No	
<code>dest</code>	Specifies the directory which holds, or will hold, a checked-out project	No	Project's basedir
<code>failonerror</code>	Makes the build fail if this task encounters an error	No	false
<code>package</code>	Specifies the module you want to check out	No	
<code>passfile</code>	Specifies the password file you want the task to read passwords from	No	~/ .cvspass
<code>port</code>	Specifies the port used to communicate with the CVS server	No	2401

Creating Change Logs

This task creates an XML-formatted report file of the change logs in a CVS repository. If you want to track what's been happening with a module, this is the way to do it. For example, take a look at the build file in Example 6-5, which creates a change log, `changelog.xml`, for the `GreetingApp` module:

Example 6-5. Getting a CVS change log (`ch06/changelog/build.xml`)

```
<?xml version="1.0"?>
<project default="main" basedir=".">
  <property name="cvs.dir" value="project" />
```

Example 6-5. Getting a CVS change log (*ch06/changelog/build.xml*) (continued)

```
<target name="main" >
  <cvspass cvsroot=":pserver:steven@STEVE:/home/steven/repository"
    password="opensesame" />
  <cvschangelog dir="${cvs.dir}/GreetingApp" destfile="changelog.xml" />
</target>

</project>
```

Here's the resulting change log, *changelog.xml*:

```
<?xml version="1.0" encoding="UTF-8"?>
<changelog>
  <entry>
    <date>2005-02-24</date>
    <time>16:18</time>
    <author><![CDATA[steven]]></author>
    <file>
      <name>org/antbook/ch06/GreetingClass.java</name>
      <revision>1.1</revision>
    </file>
    <msg><![CDATA[The Greeting App]]></msg>
  </entry>
  <entry>
    <date>2005-06-22</date>
    <time>16:25</time>
    <author><![CDATA[steven]]></author>
    <file>
      <name>org/antbook/ch06/GreetingClass.java</name>
      <revision>1.3</revision>
      <prevrevision>1.2</prevrevision>
    </file>
    <msg><![CDATA[*** empty log message ***]]></msg>
  </entry>
  <entry>
    <date>2005-02-25</date>
    <time>16:24</time>
    <author><![CDATA[steven]]></author>
    <file>
      <name>.classpath</name>
      <revision>1.1</revision>
    </file>
    <file>
      <name>.project</name>
      <revision>1.1</revision>
    </file>
    <msg><![CDATA[The Greeting App]]></msg>
  </entry>
  <entry>
    <date>2005-02-25</date>
    <time>16:34</time>
    <author><![CDATA[steven]]></author>
    <file>
      <name>org/antbook/ch06/GreetingClass.java</name>
```

```

        <revision>1.2</revision>
        <prevrevision>1.1</prevrevision>
    </file>
    <msg><![CDATA[*** empty log message ***]]></msg>
</entry>
<entry>
    <date>2005-06-22</date>
    <time>16:27</time>
    <author><![CDATA[steven]]></author>
    <file>
        <name>org/antbook/ch06/GreetingClass.java</name>
        <revision>1.4</revision>
        <prevrevision>1.3</prevrevision>
    </file>
    <msg><![CDATA[OK]]></msg>
</entry>
<entry>
    <date>2005-06-22</date>
    <time>16:29</time>
    <author><![CDATA[steven]]></author>
    <file>
        <name>org/antbook/ch06/GreetingClass.java</name>
        <revision>1.5</revision>
        <prevrevision>1.4</prevrevision>
    </file>
    <msg><![CDATA[New Version]]></msg>
</entry>
</changelog>

```

The attributes for this task appear in Table 6-6.

Table 6-6. The *cvschangelog* task's attributes

Attribute	Description	Required	Default
<code>cvsroot</code>	Specifies the CVSROOT variable you want to use	No	
<code>cvsrsh</code>	Specifies the CVS_RSH variable you want to use	No	
<code>daysinpast</code>	Specifies for how many days in the past you want change log information	No	
<code>destfile</code>	Specifies the file in which you want the change log report written	Yes	
<code>dir</code>	Specifies the directory from which to run the CVS log command	No	<code>\${basedir}</code>
<code>end</code>	Specifies the latest date for which you want to include change logs	No	
<code>failonerror</code>	Specifies that you want the task to fail if it encounters an error	No	<code>false</code>
<code>package</code>	Specifies the module you want to check out	No	
<code>passfile</code>	Specifies the password file you want the task to read passwords from	No	<code>~/cvspass</code>
<code>port</code>	Specifies the port the task should use to communicate with the CVS server	No	<code>2401</code>
<code>start</code>	Specifies the earliest date for which you want to include change logs	No	

Table 6-6. The *cvshangelog* task's attributes (continued)

Attribute	Description	Required	Default
tag	Lets you access change logs by tag	No	
usersfile	Specifies a property file holding name/value pairs connecting user IDs and names, allowing the task to report names instead of IDs	No	

The nested `user` element allows you to specify a mapping between a user ID (as it appears to the CVS server) and a name to include in the formatted report. The attributes of the `user` element appear in Table 6-7.

Table 6-7. The *user* element's attributes

Attribute	Description	Required
displayname	Specifies the name you want used in the CVS change log report	Yes
userid	Specifies the user ID of the person as far as the CVS server is concerned	Yes

Finding Changes Between Versions

The `cvstagdiff` task generates an XML-formatted report file of the changes between two tags or dates recorded in a CVS repository. Here's an example that creates a report, *datediff.xml*, for all the changes that have been made in the `GreetingApp` module in January 2005:

```
<cvstagdiff
  destfile="datediff.xml"
  package="GreetingApp"
  startDate="2005-01-01"
  endDate="2005-31-01"
/>
```

You can see the attributes of this task in Table 6-8.

Table 6-8. The *cvstagdiff* task's attributes

Attribute	Description	Required	Default
compression	Specifies the compression you want to use. Set to <code>true</code> , <code>false</code> , or a number (1–9) for compression level.	No	No compression
cvsroot	Specifies the <code>CVSROOT</code> variable you want to use.	No	
cvsrsh	Specifies the <code>CVS_RSH</code> variable you want to use.	No	
destfile	Specifies the file where the report should be stored.	Yes	
enddate	Sets the latest date for differences to still be included in the report.	One of <code>endtag</code> or <code>enddate</code>	

Table 6-8. The `cvstagdiff` task's attributes (continued)

Attribute	Description	Required	Default
<code>endtag</code>	Sets the latest tag for differences to still be included in the report.	One of <code>endtag</code> or <code>enddate</code>	
<code>failonerror</code>	Makes the build fail if this task encounters an error.	No	<code>false</code>
<code>package</code>	Specifies the module you want to analyze. Since Ant 1.6, multiple modules can be separated by spaces.	Yes	
<code>passfile</code>	Specifies the password file you want the task to read passwords from.	No	<code>~/ .cvspass</code>
<code>port</code>	Specifies the port used to communicate with the CVS server.	No	<code>2401</code>
<code>quiet</code>	Specifies that you want to suppress displayed messages.	No	<code>false</code>
<code>startdate</code>	Sets the earliest date for differences to still be included in the report.	One of <code>starttag</code> or <code>startdate</code>	
<code>starttag</code>	Sets the earliest tag for differences to still be included in the report.	One of <code>starttag</code> or <code>startdate</code>	



Here's something useful to know: Ant comes with an XSLT stylesheet, `${ant.home}/etc/tagdiff.xsl`, that you can use to generate a HTML report based on this task's XML output. Here's an example:

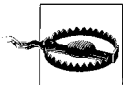
```
<style in="datediff.xml"
  out="datediff.html"
  style="{ant.home}/etc/tagdiff.xsl">
  <param name="title" expression="Date Differences"/>
  <param name="module" expression="GreetingApp"/>
</style>
```

Creating Patches

This task applies a patch file to local source code, updating the local code. You can create a patch file with the CVS `rdiff` command, which lets you compare two files. Here's an example, which applies `patch.txt` to the module in the current directory:

```
<patch patchfile="patch.txt"/>
```

The attributes for this task appear in Table 6-9.



To use this task, the patch utility must be in your path.

Table 6-9. The patch task's attributes

Attribute	Description	Required	Default
backups	Specifies you want to keep backups of unpatched files.	No	
destfile	Specifies the file you want to send the output to. Since Ant 1.6.	No	
dir	Specifies the directory where you want to run the patch command.	No	The project's basedir
ignorewhitespace	Specifies that you want to ignore whitespace differences.	No	
originalfile	Specifies the file you want to patch.	No	
patchfile	Specifies the file that contains the patch.	Yes	
quiet	Specifies you want to suppress messages unless an error occurs.	No	
reverse	Specifies you want to create the patch with old and new files in reverse order (swapped).	No	