

# Branching with Eclipse and CVS

Paul Glezen  
pglezen@us.ibm.com

In any source control management (SCM) environment, branching is a very powerful technique for controlled isolation. Unfortunately it is often avoided because of its implementation complexities despite being a simple concept. This short article demonstrates how to branch using Eclipse's CVS plugin. The intended audience is software development professionals with an appreciation of the benefits of branching and merging in SCM.

The IBM redbook, *WebSphere Studio Application Developer Programming Guide*, contains a section, *Streams in CVS*. It addresses branching and merging using CVS with Eclipse. But many of the menu choices have changed since its publication. The Eclipse documentation is accurate, but does not provide an end-to-end scenario. The intent of this article is to fill that gap.

The scenario is that of two programmers, p1 and p2, working on separate branches of the same project. P1 will branch off main and modify some files while p2 continues to work on the main branch, also modifying files. P1 will then merge his branch back to main. In doing so, p1 will address the resulting merge conflicts.

## Setup

The sample will use a few text files inside a simple project. In order to carry out the sample, you should already have a working connection to a CVS repository configured in your Eclipse workspace.

1. Create a new simple project called **brtest**.
2. Add a file called `f1.txt`. Add the following contents.

```
This file will only be edited  
by programmer p1.  
  
This line will be changed later.  
  
The rest of this file will  
remain the same.
```

3. Add a file called `f2.txt` with the following contents.

```
This file will only be edited  
by programmer p2.  
  
This line will be changed later.  
  
The rest of this file will  
remain the same.
```

4. Add a file called f3trivial.txt with the following contents.

```
This file will be edited by
programmers p1 and p2.

This line will be changed
by p1 only.

This line will be left alone.

This line will be changed
by p2 only.

The rest of this file will
remain the same.
```

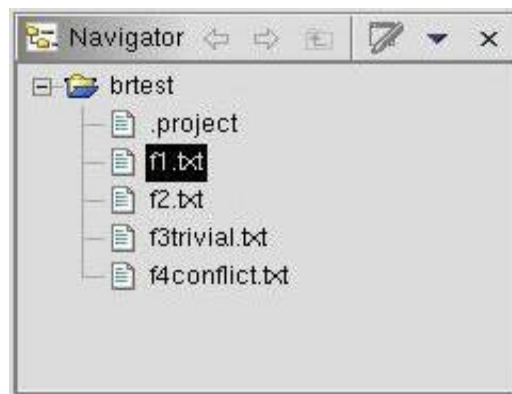
5. Add a file called f4conflict.txt with the following contents.

```
This file will be edited by
programmers p1 and p2.

This line will be changed by
by both p1 and p2.

The rest of this file will
remain the same.
```

Then your workspace should look something like this.



6. Add the project to CVS. Right-click on the project folder and select **Team --> Share Project**. Select CVS as the project type and click **Next**. Choose a suitable repository and click **Next**. This creates a CVS project called **brtest**. But we still need to add the files to source control and commit them.

7. In the **Synchronize – Outgoing Mode** view, right-click the **brtest** folder and select **Add to Version Control**. Once again right-click on the **brtest** folder and select **Commit ...**. Provide a comment such as “Initial Version.”

This takes care of setting up the environment for p1. P2 will work from the command line on the main branch. P2 just needs a working directory. From the command line she can get the project just saved by p1 with the command

```
cvs co brtest
```

## ***Branch Method***

When branching off a development stream, it is often advisable to **rebase** before merging the branch back to the main branch. Rebase means to first merge the contents of the main branch to the subbranch. If any conflicts are to be resolved, they are done so on the subbranch rather than the main branch. If the merge goes fine, then the subbranch is merged to the main branch. After having previously resolved all merge conflicts, this last merge would be a trivial one. The advantage of taking the extra rebase step is that it avoids having conflicts in the main branch.

For this simple example, p1 will simply merge his changes directly into the main branch. From the prospective of implementation details, the only difference is the merge target. Since this article is principally concerned with implementation details of the Eclipse branch support using CVS, the simplest branch method is sufficient.

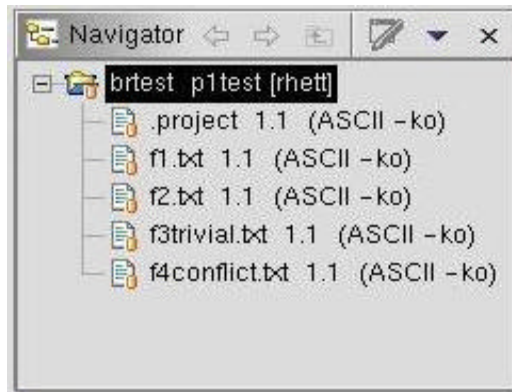
## ***Creating the Branch***

At this point, programmers p1 and p2 have the same versions of all files. It is now time for p1 to create an independent branch on which to work.

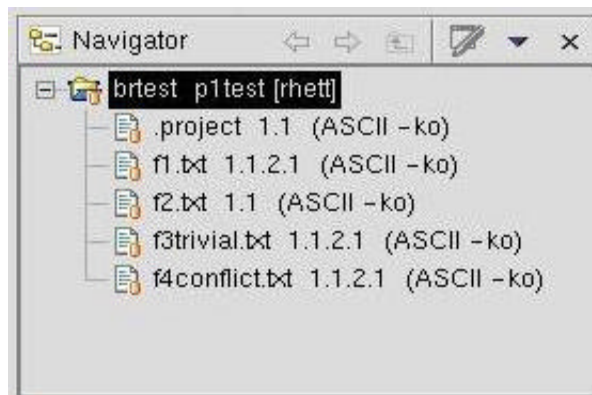
1. Right-click on the brtest project and select **Team --> Branch ...**. This brings up a **Create Branch** dialog box. Enter a branch name like “p1test”. Leave the check box checked for **Start working in the branch**. Notice that a version name is automatically filled in for you called “Root\_p1test”. You can name it anything you want (so long as it doesn't conflict with a prior tag). This keeps track of where the branch was created. It will be used later by the merge tools to determine changes since the branch. Click **OK**.

You should be able to see the result of your branch in two ways. By right-clicking the project and selecting **Properties** and then **CVS**, you should see “p1test” in the tag field. If you have CVS label decorations enabled, you'll see the branch tag in the navigator view.

To enable CVS label decorations, go to **Window --> Preferences** and navigate to **Workbench --> Label Decorations**. Check the CVS box.



2. As programmer p1 we will modify our first file. Open the editor for `f1.txt`. Change line 4 from “This line will be changed later” to “This line has been changed.” Save the change.
3. Open the editor for `f3trivial.txt`. Change lines 4 and 5 from “This line will be changed by p1 only” to “This line has been changed by p1.” Save the change.
4. Open the editor for `f4conflict.txt`. Change lines 4 and 5 from “This line will be changed by both p1 and p2” to “This line has been changed by p1.” Save the change.
5. Right-click the `brtest` project and select **Team --> Synchronize Outgoing Changes**. In the **Outgoing Mode** view, right-click the `brtest` project folder and select **commit**. Enter a comment such as “P1 changes”. Note that for the files that have changed, the revision numbers are four digits. This is a CVS convention for branched revisions.



6. Now it is p2's turn. She will make her changes in the main branch. If she hadn't already done so, she would check out the main branch. That the default with the CVS checkout command. As p2 in the main branch, edit `f2.txt`. Change “This line will be changed later” to “This line has been changed.” Save the change.
7. Edit `f3trivial.txt` in p2's workspace. Change lines 9 and 10 from “This line will be changed by p2 only” to “This line has been changed by p2.” Save the change.
8. Edit `f4conflict.txt` in p2's workspace. Change lines 4 and 5 from “This line will be changed by p1 and p2” to “This line was changed by p2.” Save the change.

9. Commit p2's changes using the CVS commit command. Below is the command line result.

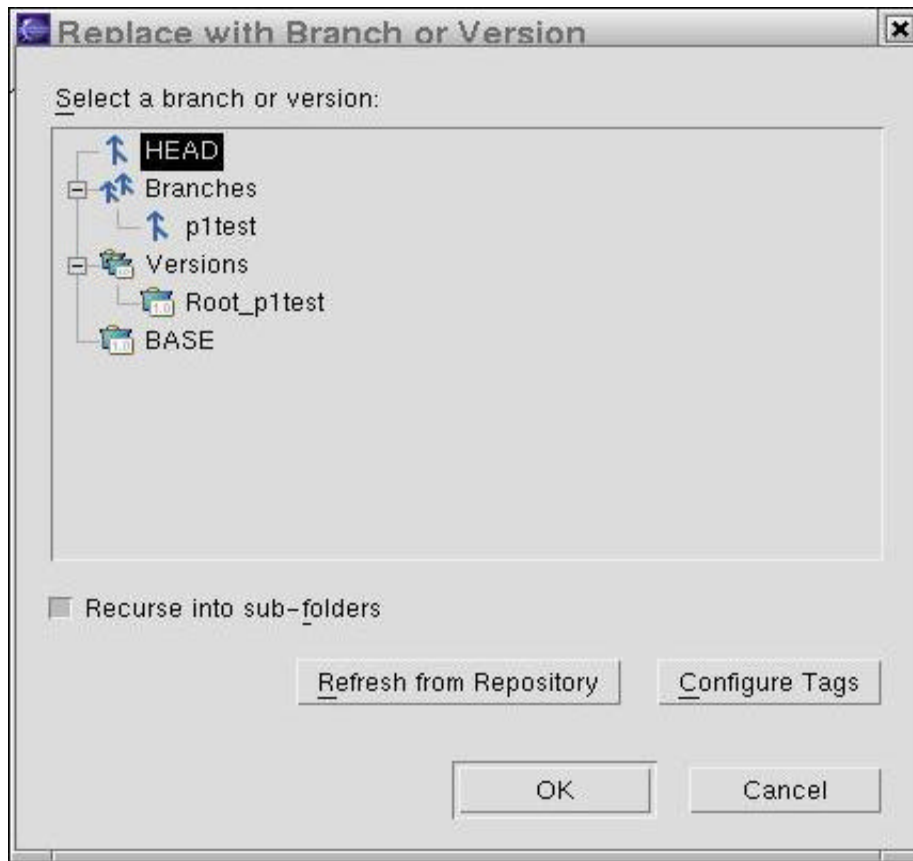
```
[~/working/brtest]$ cvs commit -m 'P2 changes' f2.txt f3trivial.txt
f4conflict.txt
Checking in f2.txt;
/home/cvs/brtest/f2.txt,v <-- f2.txt
new revision: 1.2; previous revision: 1.1
done
Checking in f3trivial.txt;
/home/cvs/brtest/f3trivial.txt,v <-- f3trivial.txt
new revision: 1.2; previous revision: 1.1
done
Checking in f4conflict.txt;
/home/cvs/brtest/f4conflict.txt,v <-- f4conflict.txt
new revision: 1.2; previous revision: 1.1
done
[~/working/brtest]$
```

Notice that p2's new revision numbers are all two digits. That's because her new revisions occur on the main branch.

## ***The Merge***

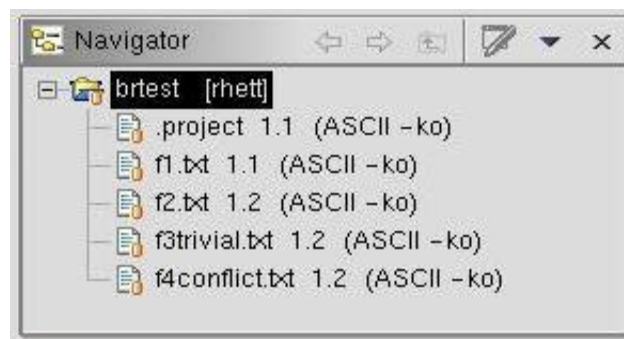
Now it's time for p1 to merge his changes into the main branch with p2's work.

10. The first step of a merge (when using Eclipse) is to replace the resources in the workspace with those of the target branch. In this case, the target of the merge is the main branch. To switch the project to the main branch right click on the **brtest** project in the Navigator view and select **Replace with --> Branch or version ...**. You should see a branch selection similar to the one below.



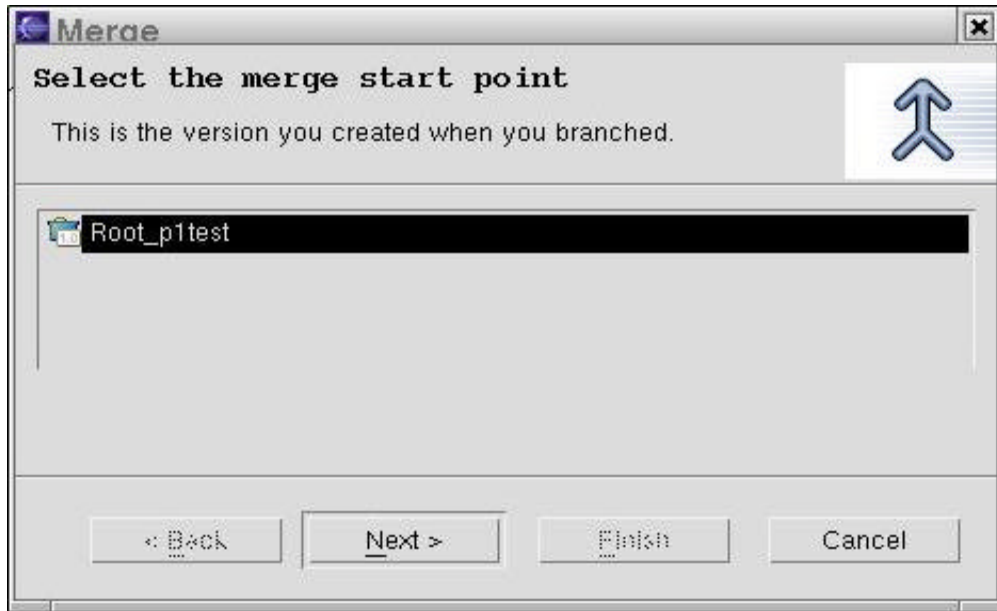
11. The p1test is the branch that p1 was working on. To switch back to the main branch, select **HEAD**. This is the CVS name for the main branch. Make sure “Recurse into sub-folders” is checked so that all the files in the project are replaced with the main branch contents. Then click **OK**.

The resource view should look like the one below. Notice the version numbers correspond to what p2 committed. All the revision numbers are two digits. The branch name is gone which implies the HEAD branch (or main).



12. Right-click on the **brtest** project and select **Team --> Merge ...**. This should display a panel allowing you to choose a “start point”. For the case of this example, the only start point

available is the one p1 created when the branch was created. Select it and click **Next**.

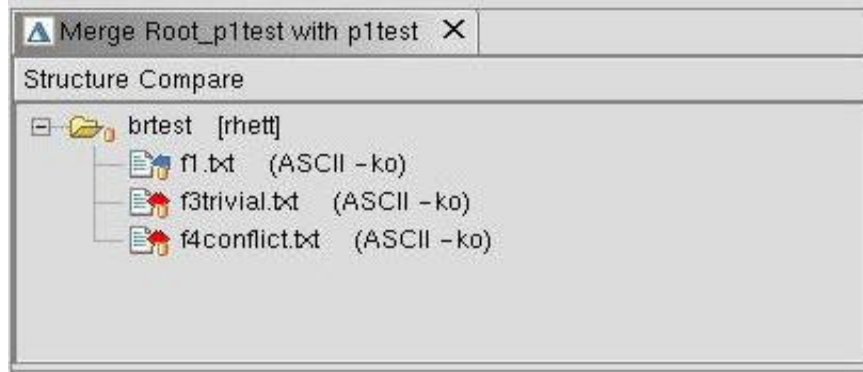


13. Next we determine from which branch to pull the changes. Expand the **Branches** node and select **p1test**. Click **Finish**.



14. This is where the fun begins. You should be presented with a Structure Compare view with

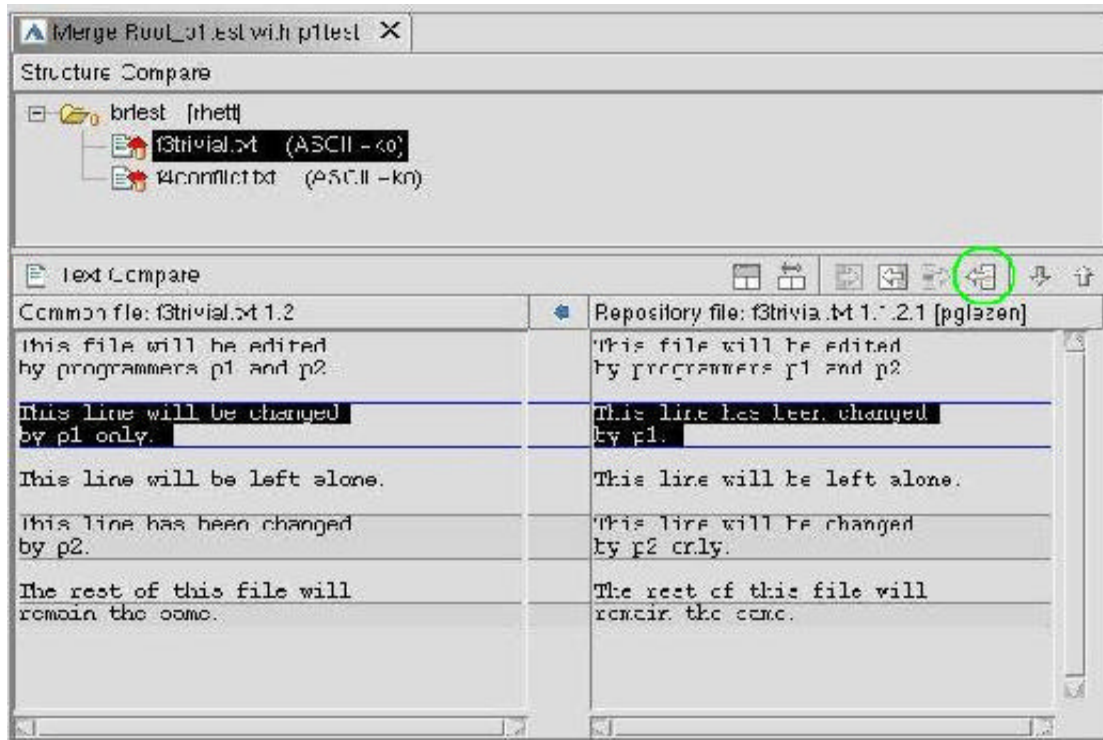
the following contents. `f1.txt` was not changed by p2, so it will come in without a problem. But `f3trivial.txt` and `f4conflict.txt` were each modified by both p1 and p2. This will require special attention during the merge.



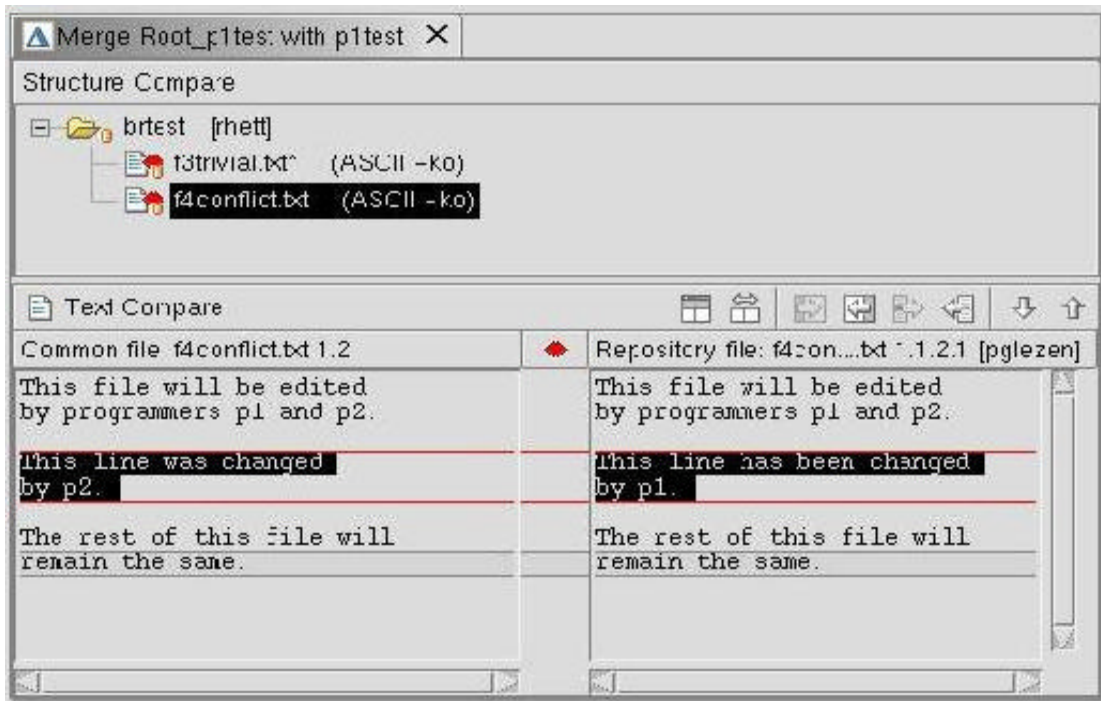
15. Right-click on the `brtest` folder and select **Update from repository**. This action only operates on non-conflicting resources in the structure compare (those with no red marks). In our example, it pulls in the changes to `f1.txt`. This should leave your structured compare with only the two conflicts.
16. Double-click `f3trivial.txt` in the Structure Compare view. This displays the Text Compare view. The left-hand side is the file in our workspace (main branch). The right-hand side is coming from the branch p1 had made. The first difference is the change p1 made that we want to merge into main. Clicking the button circled below will copy the highlighted change from right to left. If this button is initially disabled. Try clicking the down arrow and then the up arrow.

The other two differences will not be merged. That's because the second difference was due to a change in the main branch that we want to keep. The third difference is an unintentional space at the end of the last line.

After the first difference is resolved by copying from right to left, right-click in the left-hand editor (representing the local copy) and select **save**. An asterisk will appear next to the filename in the Structure Compare view to indicate this change.



17. Next double-click the f4conflict.txt to display its differences between the main branch and the brtest branch. This represents the most extreme conflict. It can't be resolved by simply choosing which line to copy. Edit the left-hand panel of the Text Compare view and change the line "by p2" to "by p1 and p2." Right-click on any line and select **Save**.



18. Since we have manually merged these files, we no longer need to retain them in the Structure Compare view. Choose them both and select **Remove From View** from their context menu.

19. At this point, the merged copy only exists in our workspace. We still need to save it to the CVS repository. Select the project in the Navigator view and select **Team --> Synchronize Outgoing Changes**. These changes should not present any conflicts. Right-click on the brtest folder in the Structure Compare and select **Commit**. A suitable comment would be "Merged brtest branch to main."

20. A successful merge is often an occasion for a release. In CVS, this amounts to tagging all the files with a given tag. In this case, we'll tag it with "p12merge". Select the brtest project in the Navigator view and select **Team --> Tag as version ...** from its context menu. Enter the name "p12merge" for the tag name.

As a review of what we did, we can select **Team --> Show in resource history** from f4conflict.txt context menu and see this.

Revision	Tags	Date	Author	Comment
*1.3	p12merge	3/23/0...	pglezen	Merged brtest branch to main.
1.2		3/16/0...	wing	P2 changes
1.1.2.1		3/16/0...	pglezen	P1 changes.
1.1	p1test, Root_...	3/15/0...	pglezen	Initial Version

We can see where revision 1.1 was tagged by the branch operation (the Root\_p1test branch). We can see the changes made by p1 and p2 (Paul and Wing). We can see the merge as well as the tag applied to the merge in the last step.