Managing Your Story Workbench Files

1. What is source control?

A **source control** system is a collection of software tools for managing files that are shared among multiple **users**. The source control system provides for central management of files, distribution of files to users, snapshots of file versions, and assurance that users have not made inconsistent edits to the same file.

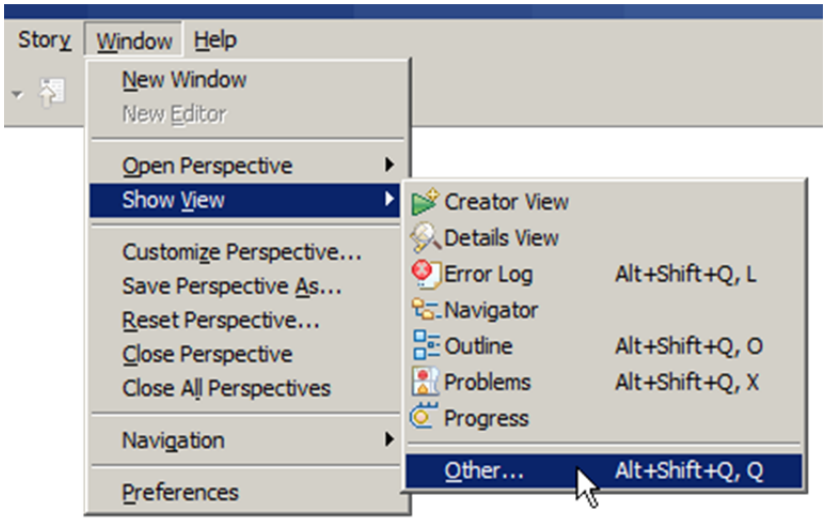
A conventional source control system can be split up into two parts: the **server** and the **client**. The server hosts the **repository**, which is the master set of files. Using a client program on their own computer, a user can connect to the repository (usually via the Internet) and **checkout** a set of files that he wants to edit. The local copy of the repository files, those actually present on the user's machine, is called the user's **working copy**. When the user is done editing a file, he **commits** the changes to that file to the repository, producing a new version of the master copy of the file. If one user commits changes to a file or folder, a second user can obtain those changes by **updating** their working copy. A file that is under version control is said to be **managed**.

2. What is Subversion?

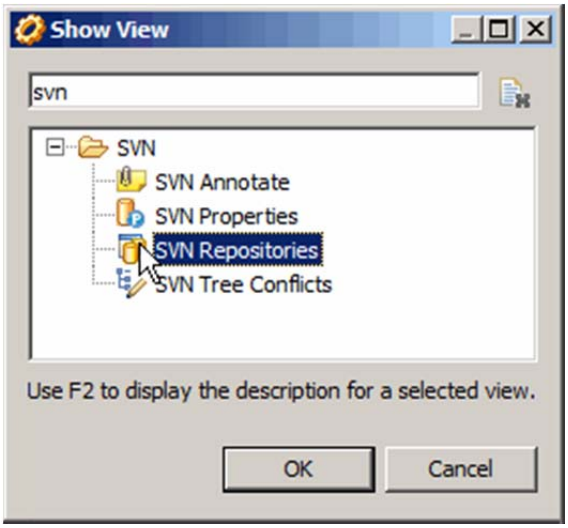
Subversion is a specific source control implementation. An important aspect of Subversion is that every repository has a global repository **revision** number. When a new, empty repository is created, its revision number is 0. Every individual commit made to the repository increments the global repository revision. A commit may be of varying complexity: it may make only a single change to a single file; it may make multiple changes to a single file; or multiple changes to multiple files. Users have the option of committing their changes in lots of small commits or one large commit. Regardless, every successful commit operation increments the repository reversion number by 1. The revision number of a file in the repository is the same as the repository revision in which that file was last changed.

3. How do I view the contents of a repository?

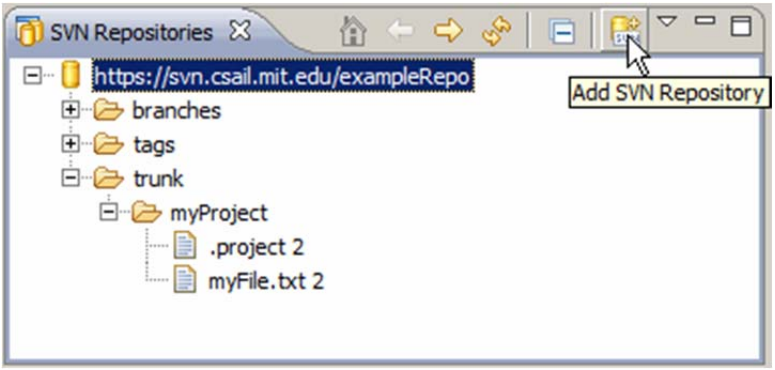
In the Story Workbench you can connect to repositories, and view their contents, using the **SVN Repositories** view. To open this view, go to Window → Show View → Other.



In the **Show View** dialog, navigate to the SVN group. (Alternatively, type "svn" in the Show View dialog filter text box to reveal all SVN-related views.) Select **SVN Repositories** to open the view.



The SVN Repositories view will list all the repositories to which you are connected. If you have not yet connected to any repositories, this view will be empty. You can connect to a new repository by selecting "Add SVN Repository" in the view toolbar and entering the URL provided by your project manager.

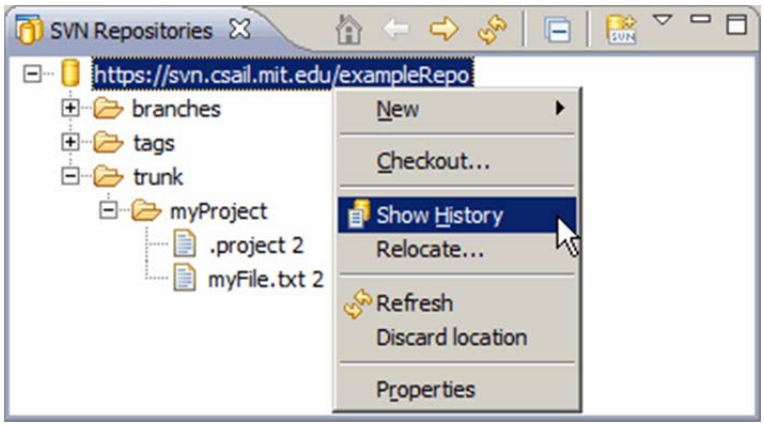


Once you have added a repository, you can explore its contents. When you expand a particular repository in the view, it will show you the folder structure of the repository. Files in the

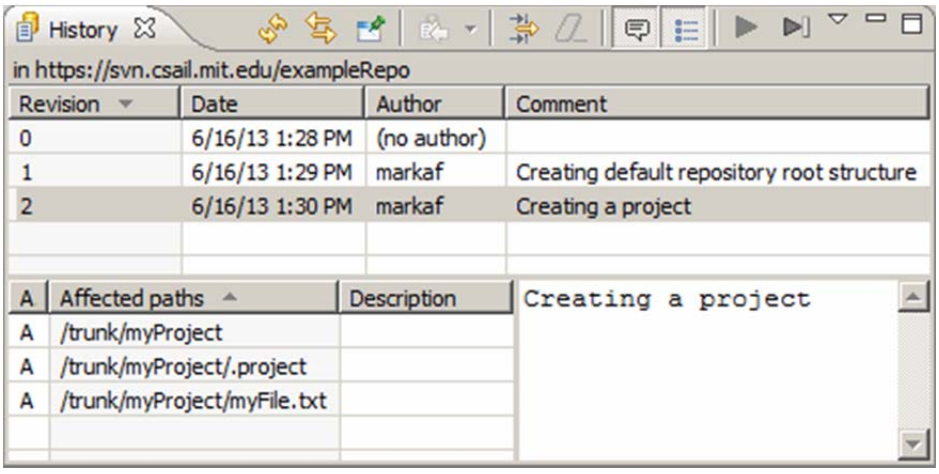
repository have a number to their right. This is the last revision of the repository in which that file was changed.

Repositories usually have three folders in their root: “branches”, “tags”, and “trunk”. This is the standard SVN repository layout. The “branches” and “tags” folders have special purposes and are not normally used by regular users. Any folders you will be interested in checking out will usually be somewhere inside the “trunk” folder. (Note that one restriction of Subversion is that you may only checkout folders from the repository, not individual files. Your project manager will tell you what folders you should checkout from your assigned repository.)

You can view the history of changes to a file, folder, or even the whole repository, by clicking on that object in the SVN Repositories view and selecting **Show History**.

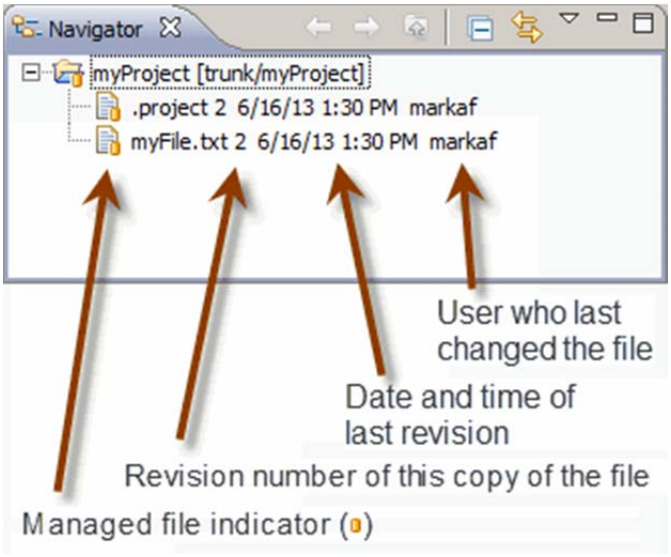


This will show the History view that lists all the commits that affected that file or folder in the whole history of the repository. The History view will show you relevant information such as the revision in which a commit occurred, the date and time of the commit, who made the commit, and their commit comment. For each commit, it will also list the files and folders affected by the commit.



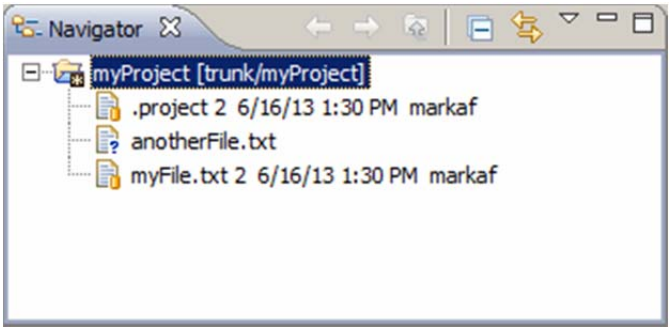
4. How do I view the contents of my working copy?

When you check out a folder into the Story Workbench, it becomes a project shown in the **Navigator** view. If the Navigator view is not already open, you can open it as you would any other view: the Window → Show View menu. If it is already open, but not visible (i.e., it is minimized), find the minimized icon and click on it.

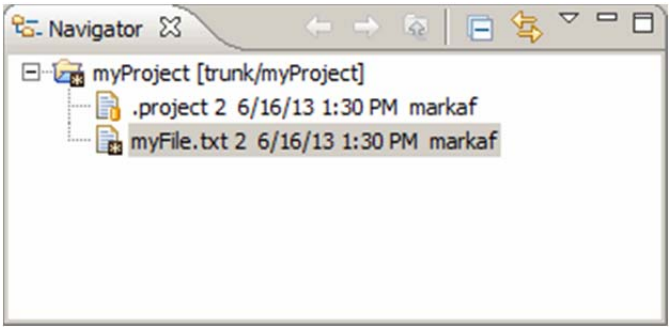


The navigator view will show some information about the managed files and folders. It shows, of course, the filename. It will also indicate the last revision of the file you have received from the repository, the date of that revision, and the user who made the last change.

Importantly, every file and folder inside of a managed project will have an indicator icon that shows what the status of that file is relative to the repository. A normal file will show with an orange cylinder overlay. If you create a new file, however, that file is shown with a question mark symbol before you commit the file to the repository. This indicates that the file is new and needs to be either checked into the repository or explicitly ignored.



If you modify a managed file and save it, this will be shown with a black star instead of the normal orange cylinder. This black star will be shown on the parent directory, the parent's parent directory, and so forth, all the way to the project root. This way, you can easily tell at a glance if your project has any uncommitted changes.



5. How does Subversion know when a file has changed?

Inside of every managed folder is a hidden subfolder named ".svn". The Story Workbench hides these folders from view, and you won't even normally see this folder in the operating system viewers (unless you have turned on "view hidden files"). Inside this hidden subfolder Subversion keeps all the information that it needs to manage the files in that particular directory; in particular, it keeps a pristine, unchanged copy of every file in that folder. This is how it knows, for example, when you have changed a file, or added or deleted a file.

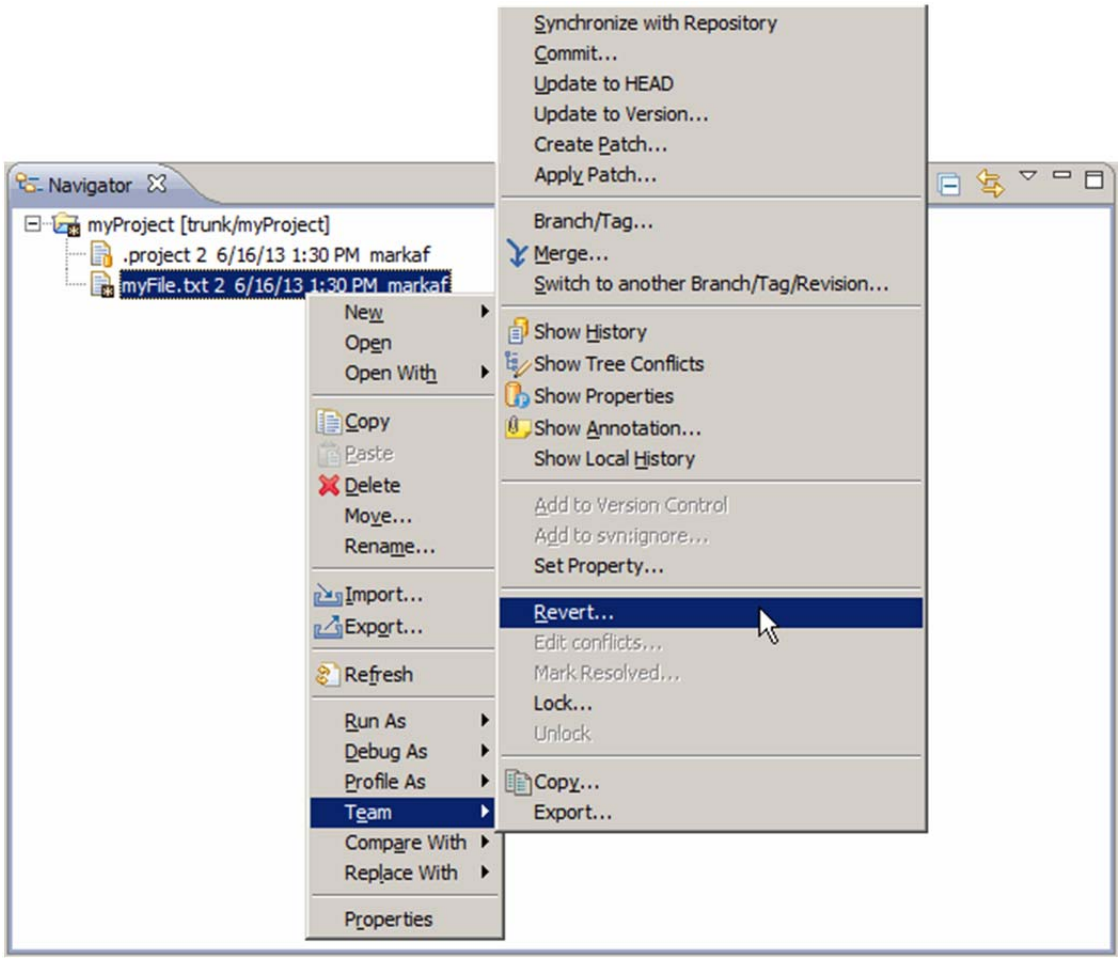
Note that this pristine copy of the file is a copy of the file as it looked when you last updated your files from the repository. If someone has made additional changes to the repository version of the file (the master version), these changes will not be reflected in your copy until you update your working copy from the repository.

6. How do I undo (revert) changes I have made?

Keeping a pristine copy of the file has a particular advantage: it allows you to **revert** any changes you've made to your copy of the file. Say, for example, that you are working on a file and it somehow becomes horribly broken. You can't open it to fix it, but you don't want to delete it because, if you commit the deletion, this will delete the copy in the repository. What you want to do is go back to the last working version of the file, the one you had when you last updated from the repository.

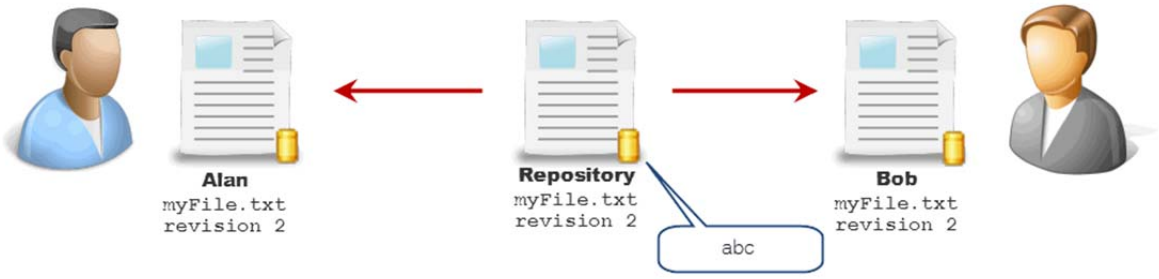
To do this, context-click on the file that you want to revert (it should be marked with a black star) and select Team → Revert. This will replace your changed version with the pristine copy of the file.

Be Careful! Once you revert a file, you have no way of recovering the uncommitted changes to the file that were present before the revert. If you are uncertain whether you should revert, and want to be extra-careful, you can save a copy of your changed file to somewhere else on your hard drive (drag-and-drop it from the Navigator view).

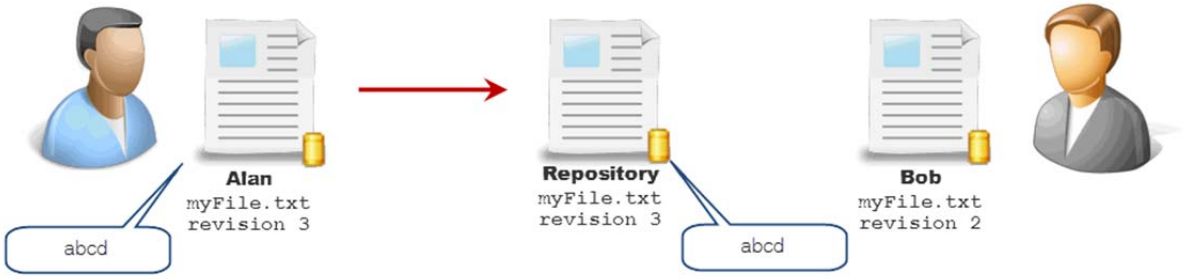


7. What is a conflict?

A conflict occurs when you have made changes to an outdated copy of a managed file. For example, suppose Alan and Bob have both checked out revision 2 of a file.

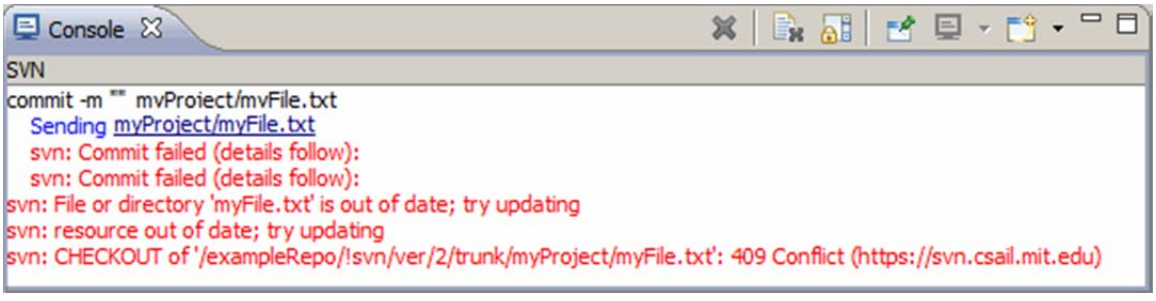


Suppose that Alan makes a change to his file and commits that change.



Now Bob has an outdated version of the file. If Bob now makes a change to his version of the file and tries to commit it, he will get a conflict error.

In the Workbench, he will see the following error:

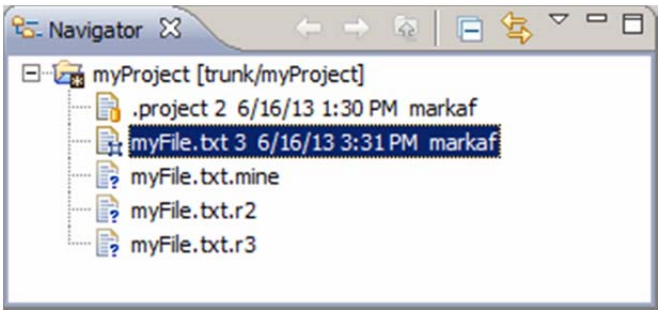


8. How do I resolve a simple conflict?

If Bob knows, for example, that he wants his changes to overwrite the changes in the repository, he there is a simple way to fix the conflict. **First**, he should copy his edited file somewhere else (drag and drop it to the desktop, for example). **Second**, he should revert the changed file in the workbench. This brings him back to a pristine, unchanged copy. **Third**, he should update his file to the latest revision, bring him into alignment with the repository. **Finally**, he can drag and drop the changed copy of the file back into his project. This will overwrite the old changes and he will now be able to commit his changes.

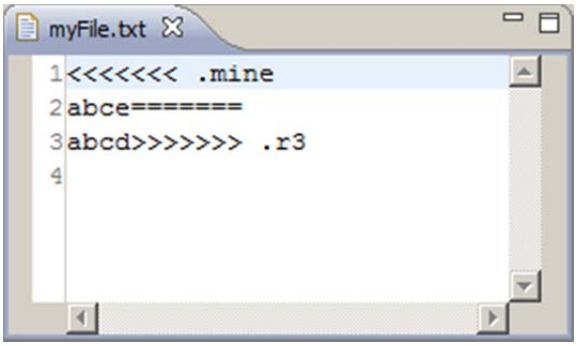
9. I have four strange copies of my file, how did this happen?

In the last example, we saw Bob have a conflict between his edits and edits committed by another user (in fact, if Bob is using multiple computers, he may be conflicting with himself!). The Workbench, via the Console view, has recommended he update his file. If he does update his working copy, without reverting his changes, he will get the following situation:



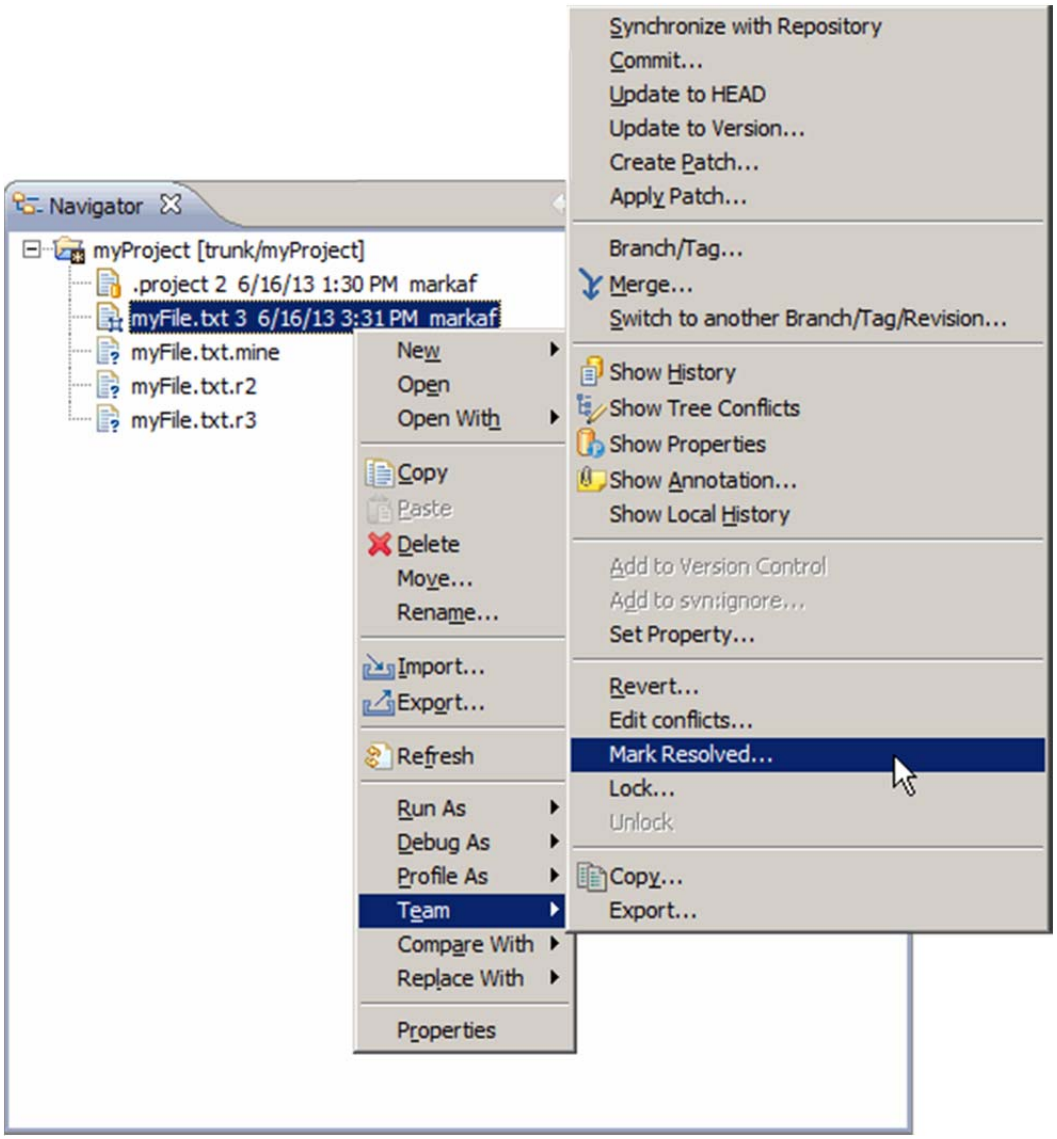
Each file here represents different states of the file.

- `myFile.txt.mine`: This is changed version of the file that Bob produced from revision 2. This is the version of the file he tried to commit.
- `myFile.txt.r2`: This is the pristine copy of the file in Bob's working copy. In this example, the "r2" stands for "Revision 2"
- `myFile.txt.r3`: This is the copy in the repository that Bob is trying to merge his changes into. In this example, this is revision 3.
- `myFile.txt`: This is **not** the original file. This is the original file with "difference" information inserted into it. <u>These insertions will prevent a story file from opening in the Story Workbench editor.</u> For this example, if we look at this file we see the following:

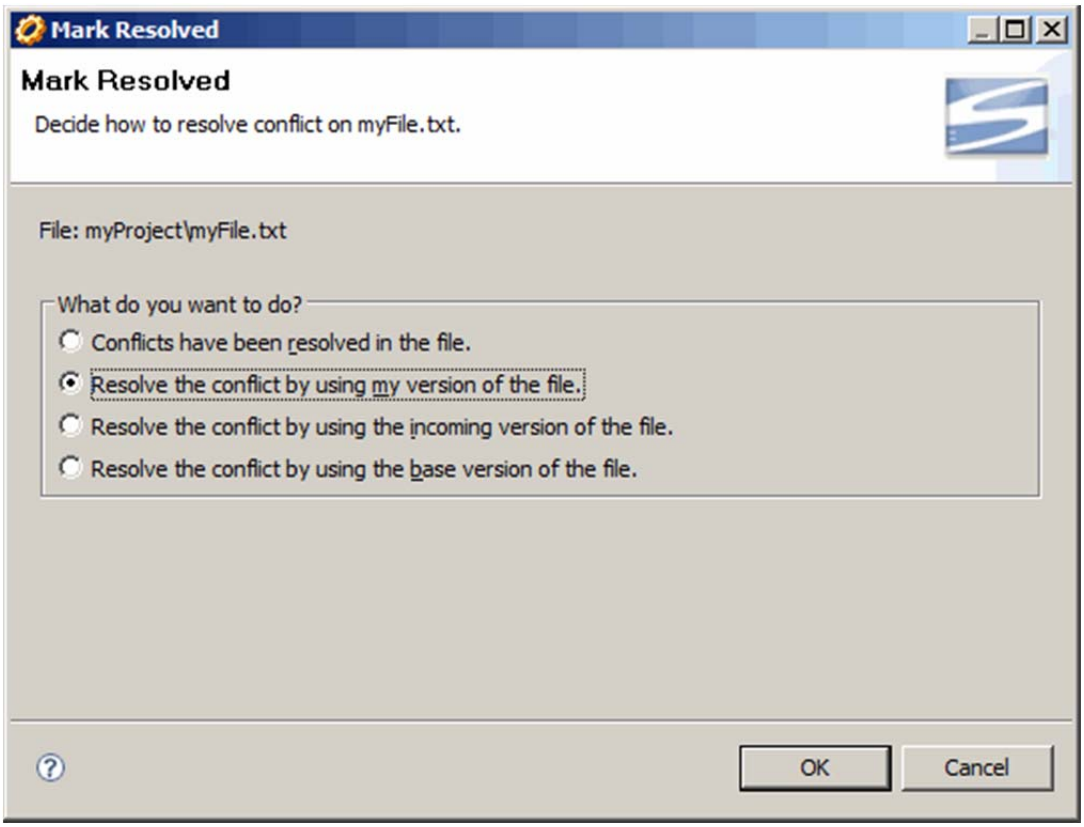


To fix this problem, you must resolve the conflict by telling the workbench which version of the file you want to use. If Bob knows, for example, that he wants his changes to overwrite the changes in the repository, he can tell the workbench to resolve the conflict with his version of the file.

First, he context-clicks one of the four files that indicate the conflict, goes to Team, and selected **Mark Resolved**.



In the Mark Resolved dialog, he then selects “Resolve the conflict by using my version of the file.”



This will resolve the conflict and allow Bob to commit his version of the file, overwriting the version in the repository.

10. How do I resolve a complex conflict?

Suppose, however, that Bob is not sure if he should overwrite the copy in the repository. Perhaps his project manager may have made some changes to his file that he was unaware of. Bob foolishly did not update his working copy before beginning his edits, and now he has an outdated file. Or perhaps he has been lazy about committing his work, and so he has had file changes waiting to be committed long enough for someone else to come along and change the file out from under him.

In the case where Bob needs to integrate the repository changes into his version before committing, he will have to manually compare the two versions of the file. The easiest is to follow the steps 1 through 3 listed in Question 8 “How do I resolve a simple conflict?” This puts the repository version of the file in Bob’s project. He can then copy his backup copy of the file (made in step 1) back into the project, being careful to rename it so that it does not overwrite the updated version. He can then visually compare the two files and manually copy over the changes from one file to the other. At this point Bob feels great remorse for not updating his files before starting his work, and vows never to fall so short again.