Day-to-Day Usage of AccuRev 4.6

This document presents enough information for the individual user to work with AccuRev Version 4.6 on a day-to-day basis. We provide a brief overview and discuss a handful of commands. It's a short document, because AccuRev is an elegantly simple configuration management system.

> Note: words with a double-underscore are defined in the AccuRev Glossary, in the *AccuRev Concepts Manual*.

The AccuRev Usage Model

AccuRev's flexibility makes it easy to use for a variety of development scenarios. But like every software system, AccuRev has usage models that were foremost in the minds of its architects. This section describes the most common usage model.

AccuRev is a configuration management (CM) system, designed for use by a team of people (users) who are developing a set of files. This set of files might contain source code in any programming language, images, technical and marketing documents, audio/video tracks, etc. The files — and the directories in which the files reside — are said to be "version-controlled" or "under source control".

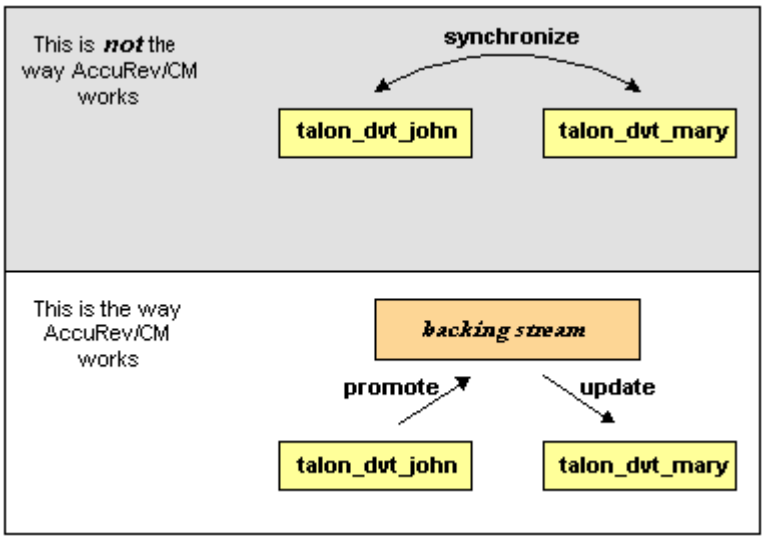
For maximum productivity, the team's users must be able to work independently of each other — sometimes for just a few hours or days, other times for many weeks. Accordingly, each user has his own private copy of all the version-controlled files. The private copies are stored on the user's own machine (or perhaps in the user's private area on a public machine), in a directory tree called a workspace. We can picture the independent workspaces for a three-user team as follows:

This set of users' workspaces uses the convention of having like names, suffixed with the individual usernames. AccuRev enforces this username-suffix convention. **widget_dvt** might mean "development work on the Widget product"; **john**, **mary**, and **derek** would be the users' operating system login names.

From AccuRev's perspective, development work in this set of workspaces is a continual back-and-forth between "getting in sync" and "getting out of sync":

- Initially, the workspaces are completely synchronized: they all have copies of the same set of version-controlled files.
- The workspaces lose synchronization as each user makes changes to some of the files.
- Periodically, users share their changes with each other. When **john** incorporates some or all of **mary**'s changes into his workspace, their two workspaces become more closely (perhaps completely) synchronized.

You might assume that the workspace synchronization process involves the direct transfer of data from one workspace to another. But this is not the way AccuRev organizes the work environment. Instead of transferring data directly between private areas (that is, between users' workspaces), AccuRev organizes the data transfer into two steps:



1. One user makes his changes public — available to all the other members of his team. This step is called promotion.
2. Whenever they wish, other team members incorporate the public changes into their own workspaces. This step is called updating.

The first step involves a public data area, called a stream. AccuRev has several kinds of streams; the kind that we're discussing here is called a backing stream. We'll see below how the data in this public stream "is in back of" or "provides a backstop for" all the private workspaces of the team members.

Change and Synchronization: The Four Basic Commands

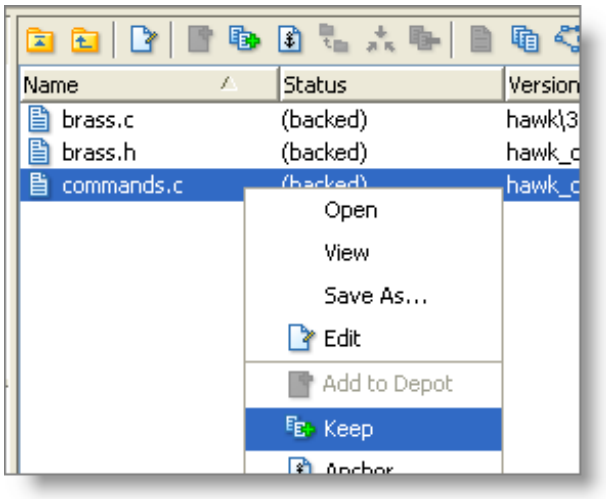
With the usage model described above, you'll be able to accomplish most of your AccuRev work with four simple commands: **Keep**, **Promote**, **Update**, and **Merge**. We describe these commands in the following sections. Each section has a subsection titled "The Fine Print", in which we present additional usage details, notes on the way AccuRev implements certain features, and other tidbits of interest. You might want to skip over these sections on your first reading of this material.

Keep: Preserving Changes in Your Private Workspace

An AccuRev workspace is just a normal directory tree, in which you make changes to version-controlled files. You can work with the files using text editors, build and test tools, IDEs, etc., just as if the files weren't version-controlled at all. For example, you might edit a source file and invoke the editor's "Save" command a dozen times over the course of an hour or two. These operations don't involve AccuRev at all — they simply have the operating system change the contents and the timestamp of the file in your workspace.

You don't need to perform a "check out" operation or otherwise get permission from AccuRev before editing a file in your workspace. (Some legacy CM systems do impose such a regimen; AccuRev can be configured to require checkouts, too.)

Every so often, you want AccuRev to preserve the current contents of the file as an official new version of the file. You accomplish this using AccuRev's **Keep** command. This figure shows how to invoke the **Keep** command from a file's context (right-click) menu in the AccuRev File Browser tool, which has a Windows Explorer-like interface. You can also invoke **Keep** with the  toolbar button.



You can continue modifying the file, then using **Keep** to preserve the latest changes, as often as you like. Other team members won't complain about "thrashing", because these new versions stay within your workspace; without affecting any other user's workspace.

AccuRev retains all the versions that you **Keep**. This makes it possible for you to roll back to any previous version you created.

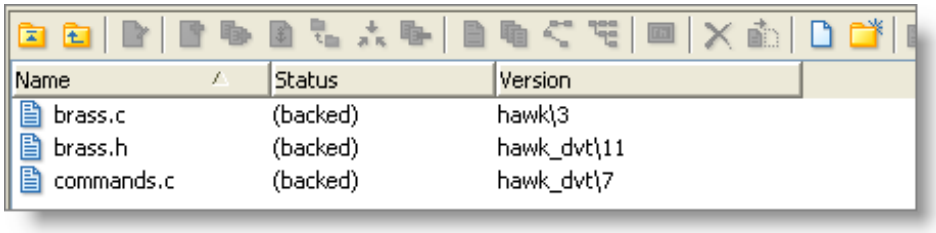
Several other operations are similar to **Keep**, in that they create a new version of a file in your workspace, without affecting any other user's workspace. The most important are:

- **Rename**/**Move**: You can rename a file or move it to a different directory (or both), using AccuRev commands. Other users will continue to see the file at its original pathname in their workspaces.
- **Defunct**: You can remove a file from your workspace with the AccuRev command **Defunct**. Other users will continue to see the file in their workspaces.

The Fine Print

We said above that AccuRev "retains all the versions that you **Keep**". But where? Each time you **Keep** a file, its current contents are copied to the AccuRev repository, located on the machine where the AccuRev Server runs. You don't need to care about the name and precise location of this copy. Each version you create has a version-ID, such as **widget_dvt_john/12** ("the **12**th version of this file created in workspace **widget_dvt_john**").

AccuRev keeps track of the status of each file in a workspace. After you **Keep** a file, the Status column in the AccuRev File Browser contains the indicator **(kept)**. It also contains the indicator **(member)**, meaning that the file belongs to the set of files you're actively working on. (See **Active and Inactive Files** below.) The Version column displays the version-ID.

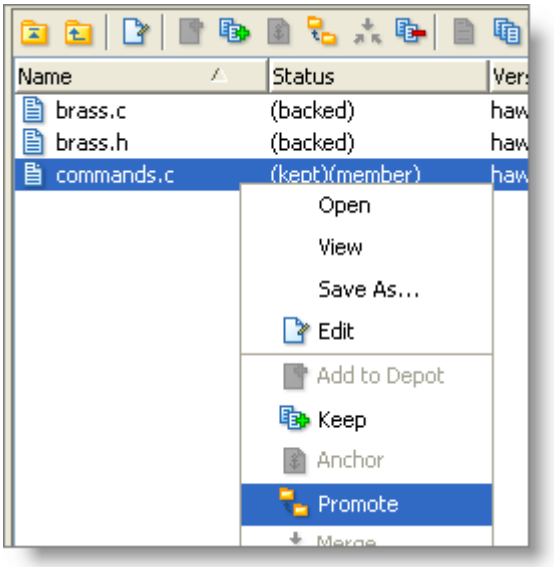


A change to the data within a file, recorded by **Keep**, is termed a content change; the change made by **Rename**/**Move** or **Defunct** is termed a namespace change. (Many CM systems don't handle namespace changes at all, or have very limited capabilities in this area.) As noted above, AccuRev saves a new copy of the file in the repository whenever you make a content change. But it doesn't need to copy the file when you make a namespace change; rather, the AccuRev Server just records the change in its database.

To perform version control on directories, AccuRev only needs to keep track of namespace changes — renaming, moving, or deleting a directory. Unlike some legacy CM systems, AccuRev doesn't need to record a new directory version when you make a content change — for example, adding a new file to the directory.

Promote: Making Your Private Changes Public

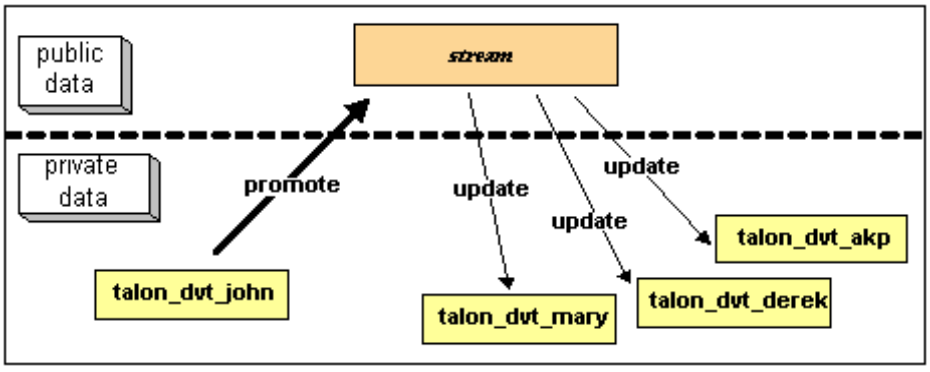
At some point, after you've used **Keep** to create one or more new, private versions of a file in your workspace, you typically want to share the changes you've made with the other team members. To make your (most recent) new version "available to the public", you promote it. This figure shows how to invoke the **Promote** command from a file's context (right-click) menu in the File Browser. You can also invoke **Promote** with the toolbar button.



Promoting your new version of a file does not automatically "push" it into the workspaces of the other team members. When a user decides that he's ready to incorporate versions of files that other team members have **Promote**d, he "pulls" them into his workspace with the **Update** command (details below).

Streams

The **Promote** command sends data to — and the **Update** command gets data from — a sophisticated AccuRev data structure called a stream. The stream acts as a "central data exchange" for the set of workspaces used by a development team. A stream also has a bit of "traffic cop" built in,



preventing team members' efforts from colliding and providing other mechanisms to control the flow of data.

A stream is not, as you might initially suppose, a set of copies of promoted files. Rather, it's more like a list of version-IDs.

- the 4th version created in workspace **talon_dvt_akp** of file **command.c**
- the 7th version created in workspace **talon_dvt_mary** of file **talon.c**
- ... etc.

In CM-speak, a stream is a configuration of a collection of version-controlled files. The term "stream" is apt, because it implies the ongoing change of a development project. Each time a user promotes a version of file **brass.c**, the stream configuration changes for that file — for example, from "the 5th version created in workspace **talon_dvt_derek**" to "the 7th version created in workspace **talon_dvt_mary**".

Promotion and Parallel Development

Sometimes, AccuRev doesn't allow you to promote a file to the development team's stream, because another team member has already promoted the same file (after modifying it and performing a **Keep** on it). AccuRev is preventing you from overwriting your colleague's change to the team's shared stream. This situation is called an overlap: two users working at the same time on the same goal, to create the stream's next version of a particular file.

Before you can promote your changes to the stream, you must first perform a merge on the file that has an overlap (details below).

Active and Inactive Files

As you work with a file using the commands described above, AccuRev considers the file to alternate between being *active* in your workspace and *inactive*:

- The file is initially *inactive*.
- When you create a new version in your workspace, using **Keep**, **Rename**/**Move**, or **Defunct**, the file becomes *active*.
- When you make your private version public, using the **Promote** command, the file becomes *inactive* again.

Later, you might restart this cycle, making the file active again by creating another new version of it. Alternatively, an update of your workspace might overwrite your inactive file with a newer version that another team member promoted.

AccuRev keeps track of the set of active files in your workspace. Officially, this set is called the default group. You might find it easier to think of it as the workspace's "active group" or its "active set".

The Fine Print

The **Promote** command doesn't copy the promoted version to the AccuRev repository. It doesn't need to. Promotion just gives an additional name to a version that *already exists* in the repository — having been placed there by a previous **Keep** command (or **Rename**/**Move** or **Defunct**). For example, promoting "the 7th version created in workspace **talon_dvt_mary**" might give that version the additional name "the 3rd version promoted to stream **talon_dvt**".

Just to emphasize the previous point: a stream does not reside in the file system, but in the database managed by the AccuRev Server. Promoting a version to a stream does not create a copy of a file; it just creates an additional file-reference in the Server database.

It might seem strange at first that deleting a file with the **Defunct** command makes the file *active*. The File Browser continues to list the file — with a **(defunct)** status — even though the file has been removed from your workspace's disk storage. This design feature enables AccuRev to implement the file-deletion operation using the same private-change/public-change scheme as all other changes.

We've discussed *the* stream behind a set of workspaces. But a typical development project has many streams, organized into a hierarchy. Promoting a version to a higher-level stream from a lower-level stream makes that version "even more public" — for example, available to users outside your local development team.

Update: Incorporating Others' Changes into Your Workspace

As users work independently of each other, the contents of their workspaces increasingly diverge. Typically, some of the differences between workspaces are inconsistencies. For example, changes that John makes in a report-library routine might cause errors in the report program that Mary's writing. To minimize the time and effort required to resolve inconsistencies during the "integration" phase of a project, it makes sense to have users synchronize their workspaces on a regular basis.

With AccuRev, synchronization does not mean incorporating data into your workspace directly from one or more other workspaces. Instead, synchronization involves copying data into the workspace from the stream to which all team members **Promote** their changes. This operation is performed by the **Update** command. This figure shows the **Update** toolbar button. You can also invoke this command as **File > Update** from the main menu.

> Note: the stream's role as a provider of data — through **Update**s — to a set of workspaces motivates the term backing stream. Think of restocking a store's shelves with merchandise retrieved from "the back room".

So an update operation on your workspace copies versions of certain files from the backing stream to the workspace, overwriting/replacing the files currently in the workspace. But which

files? **Update** changes a file if (1) there is a newer version in the backing stream, and (2) the file is *not* currently active in your workspace.

Update won't overwrite an active file, even if there's a new version of it in the backing stream. No matter how good someone else's code is, you don't want his changes to wipe out the changes that you've been making! This situation is another instance of an overlap, which we described in the **Promote** section above. (You can encounter an overlap both (1) if you're trying to make your private changes public (promotion), or (2) if you're trying to bring already-public changes into your private workspace (updating).) In all such situations, AccuRev resolves the overlap situation with a merge operation (details below).

Update handles namespace changes as well as content changes. Thus, if your colleague renamed a file and promoted the change, an update will cause the file to be renamed in your workspace. And if your colleague removed a file (**Defunct** command), an update will cause the file to disappear from your workspace.

The Fine Print

Here's how AccuRev prevents an update from "clobbering" your changes. The first thing **Update** does is to analyze your workspace, determining whether each version-controlled file is "active" or "inactive". Initially, all the files in a workspace are inactive — each one is a copy of some version in the repository. (For each version-controlled file, AccuRev keeps track of *which* particular version.)

A file is deemed to be active in your workspace if you've created a new version of it, using the **Keep**, **Rename**/**Move**, or **Defunct** command. (A couple of additional commands "activate" a file; one of them is discussed below.) When **Update** copies versions from the repository into your workspace, it skips over all such active files.

> Note: **Update** can tell if you've modified a file but have not yet stored the changes in the repository as a new **Keep** version. It uses timestamps and checksums to determine this. The presence of such files prevents the update from proceeding if updating would "clobber" one or more of them with the backing-stream version. You can use the **Anchor** command to activate such files, enabling **Update** to do its work.

Merge: When Changes Would Collide ...

The preceding sections, on the **Promote** and **Update** commands, both discuss the situation in which two users concurrently work on the same file. Their changes to the file are said to overlap. Both **Promote** and **Update** decline to process a file with overlap status, because doing so would cause one user's changes to overwrite the other's changes.

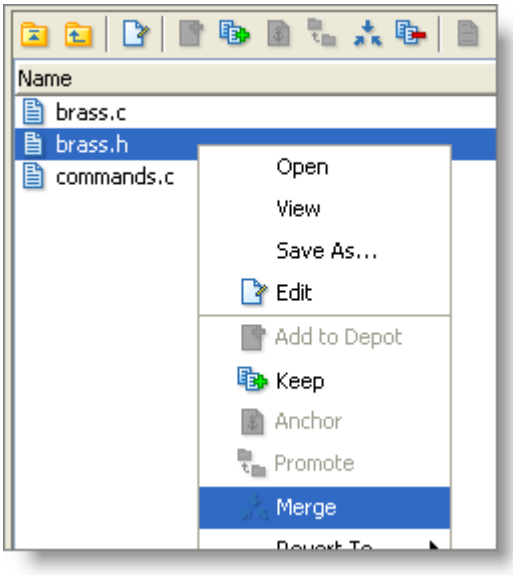
For example:

- Team members John and Mary both **Keep** one or more new private versions of **brass.c** in their respective workspaces.
- Mary **Promote**s her latest new version of **brass.h** to the backing stream.
- At this point, AccuRev will decline to do either of the following:

- **Promote** John's version of **brass.h** to the backing stream.
- Overwrite John's copy of **brass.h** during an update. (The **Update** command skips over this file, but continues its work on other files.)

To enable either a promotion or an update of **brass.h**, John must incorporate, or merge, the version in the backing stream — which contains Mary's changes — into his own copy of the file. The **Merge** command is essentially a fancy text editor, which combines the contents of two versions of a text file. The resulting "merged version" replaces the file in John's workspace.

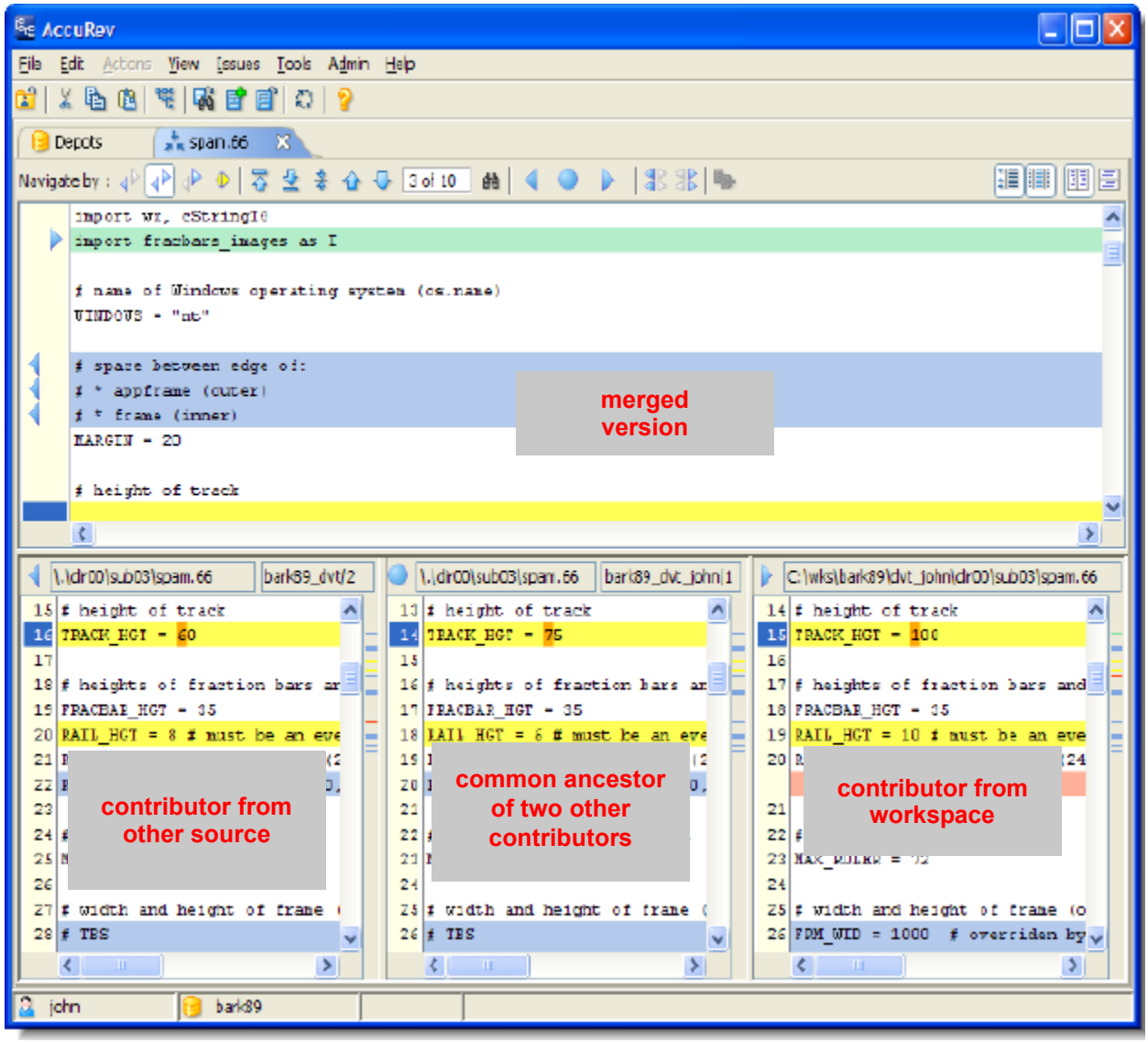
This figure shows how to invoke the **Merge** command from a file's context (right-click) menu in the File Browser. You can also invoke **Merge** with the  toolbar button.



Often, a merge operation is unambiguous, and so can be performed automatically. For example, suppose Mary's changes to file **brass.h** all occur in lines 1–50, and all of John's changes occur in lines 125–140. In this case, merging the two versions involves replacing some or all of John's first 50 lines with Mary's. Now, the edited version of **brass.h** in John's workspace contains both users' changes.

> Note: we don't claim that the two sets of changes are semantically consistent with each other. That's what the build-and-test cycle is for!

If both John and Mary have made changes to the same part of the file — say, lines 85–87 — then John must decide how to resolve this conflict. The graphical **Merge** tool makes this easy:



After performing a merge, AccuRev automatically **Keep**s the merged version to preserve the results of the merge operation. You can then **Promote** the merged version to the backing stream. After that, other team members can use **Update** — or perhaps **Merge** — to bring all the changes into their workspaces.

The Fine Print

The graphical Merge tool performs a “3-way merge”, which uses the common ancestor of the two versions being merged. This algorithm helps to automate the merge operation, often completely eliminating the need for human intervention. AccuRev performs merge operations on text files only, not on binary files.

AccuRev keeps track of all merge operations. This greatly simplifies subsequent merge operations on files that have been merged previously: you don’t need to resolve the same conflicts over and over again.

The most common overlap situation involves AccuRev's preventing you from promoting a file, because someone else "got there first" in creating a version in the backing stream. AccuRev can also detect deep overlaps, in which another user "got there first" in creating a version in the *parent* of the backing stream, or in other higher-level streams.

Learning More about AccuRev

Armed with the four commands **Keep**, **Promote**, **Update**, and **Merge**, you'll be able to work effectively in team parallel development environment. To make full use of AccuRev's configuration management capabilities, you'll need to dig a bit deeper. But no matter what your SCM challenges are, we think you'll find that AccuRev meets them with an architecture and user interface that are intuitive and easy to learn.