TFS Branching Guidance – Q&A

Original work: Bob Leithiser, Brian Blackman, Caroline Williams, Eugene Zakhareyev, Oliver Hilgers, Pramod Vasanth
Updated by: Mat Velloso, Neno Loje, Sin Min Lee, Bill Heys

Microsoft Corporation

VSTS Rangers
This content was created in a VSTS Ranger project. VSTS Rangers is a special group with members from the VSTS Product Team and Microsoft Services. Their mission is to provide out of band solutions for missing features or guidance.

Table of Contents

Question

What are the disadvantages of cherry picking changes?

Answer

Cherry picking is the process of selectively choosing which files/changesets to branch/merge. Although it gives you total control of when and how you want to execute a merge operation, it can become quite confusing along the time. Imagine, for example that you are merging a feature branch into your main source. You then decide that since you know which files have been changed from that branch you will select only those files to be merged back to main. Unfortunately another developer had to change more files than you are aware of and because you didn’t execute a full merge you left those files behind. Now your main source is “broken”, might not even build correctly and you will probably spend hours trying to discover what went wrong. This is one of the examples of what can go wrong with this practice.

Also, cherry picked changeset can fall in the middle of a future merge range resulting in remerge of that changeset. This can result in merge conflicts.

While merging all the content of a branch might sound like too much work in some cases, it actually ensures that the complete solution fits into the model, no files or changes are left behind and no one in your team needs to know which file was specifically changed in each situation.

Although, there are situations where cherry picking could be necessary, such as when we need to fast-track a critical fix from one branch into another. But as the complexity of a project increases, cherry picking is more likely to cause problems such as fixed bugs that appear again in new builds or broken builds due to incorrect file versions.

Question

Are work items a part of the branching/merging model?

Answer

No, work items exist at a logical level that is distinct from that of the source control repository. That means the branching of the source code or merging between two branches has no impact on work items:

- Work items will NOT be duplicated during branching or merging process
- The work item links will NOT be updated or extend with links referencing the new branch/merged items

All branching/merging operations require a new check-in and do not carry over work item history. Although work item history is not propagated during this operation, previous work item associations are unaffected. Note that since branch/merge operations require check-in, this provides an opportunity for work item association. Work items may be associated with the branch/merge changeset; work item association can be forced through check-in policy. For reverse/forward integration, you may create separate work item associated with a merge. For example, when integrating code from the development branch into the main branch after finishing integration testing, create the work item "Complete integration testing for release X." If it is still necessary to duplicate work items for the association with the branch/merge operation, use "Copy Work Item" function (see reference). You can copy work items within the same Team Project or into another Team Project. Note that after copying a work item, the links within copied work item still reference same artifacts as in the original work item.

Keep in mind that work items in TFS 2010 can be organized in a hierarchical manner, therefore you could possibly have a main work item called "Feature set XYZ" that groups other work items, each with a link to specific changesets. Work items have greatly improved in TFS 2010 and this is one of the examples on how you can take advantage of these improvements.

Additional Resources

- "How to: Copy a Work Item" at http://msdn.microsoft.com/en-us/library/ms181321(VS.100).aspx

Question

When does auto merging not work?

Answer

TFS 2010 offers two options for automatic merge: “Auto Merge” and “AutoMerge All”.

When merge operation is performed and there are conflicts, there is “Auto Merge” option available in Resolve Conflict dialog. For this option to work, the following criteria must be met:

1) File merging must be enabled for the file type (Team Project Collection Settings Settings » Source Control File Types). Note, that by default all source code file types commonly used by Visual Studio are enabled for file merging.

2) The changes made to source and target file should be of *edit* type (rename/delete changes cannot be auto merged. For some of these cases the “AutoMerge All” function can be used, see below)

3) Content changes should be made to different portions of content (different lines)

However, sometimes “auto merge” is used to refer to a different process; namely, when merging two branches, some of the files will be merged from source to target without ever needing user intervention (that is *merge* change will be pended automatically). That will happen to all items that were modified in source branch, and were not modified in target branch since the last merge. Otherwise, you will have to resolve conflict occurring during merge process. Note, that when baseless merge is performed, the conflict will occur for every item being merged (no matter if the item is represented by the file with identical name/content). This is due to the absence of relationship between the items; once conflicts are resolved and the merge changes are checked in, subsequent merges will behave in the same way as ordinary merge

TFS 2010 also brings the AutoMerge All functionality. AutoMerge All supports executing a batch operation on common tasks that could be considered common sense in a merging process. Situations such as a file that has been renamed (which would generate a conflict but can be solved using “AutoMerge All”) or identical changes happening on both source and destination can be automatically solved by using the AutoMerge All.

AutoMerge All supports automatic conflict management of the following types:

- Conflicts with content changes made only in the local workspace or target branch
- Conflicts with content changes made only in the server version or source branch
- Conflicts caused by renaming file in the server version or source branch
- Conflicts caused by identical change in the server and local workspace

You can either select all options above or just the only ones you want to execute for your specific needs.

Additional Resources

- Refer to Baseless Merge within Branching Guidance section
- "Understanding Merging" - http://msdn.microsoft.com/en-us/library/ms181427(VS.80).aspx

Question

Is it possible to merge all changes manually?

Answer

When two branches are merged, TFS will not create conflicts for the items that were modified (or added) only in the source branch and were not modified in the target branch since the last merge; changes of *merge* type will be automatically pended for those items.

You may wish to review every single change prior to merging; however, TFS will not create conflicts for such items nor can conflict creation be forced.

But since any of merge changes must be checked in, you may review all changes pending as a result of merge prior to performing check in.

Question

What is required to consolidate or transition branches?

Answer

Ideally, your branching strategy is the result of planning and reflects the needs and goals of your teams. However, in today's constantly shifting environment of mergers and acquisitions, you may find that you will need to integrate the work of two or more previously autonomous teams into a single consolidated environment. It is important to consider the benefits vs. cost of making this decision, in terms of several factors including: current configuration, team structure, release cycles, migration effort required and future needs.

Consolidating and transitioning of branches should be considered with the equivalent importance of the initial migration of existing source code into the Team Foundation Server environment. Questions that should be asked of the teams considering this action should include:

1. What are the predominant reasons for considering this action?
2. What are the benefits of this action? What are the benefits for remaining with the status quo?
3. What are the risks of consolidation?
4. What resources will be required to complete a consolidation?
5. What is the exit plan?
6. What effects will this have on future development?
7. How will any existing best practices be incorporated?
8. Do we need to keep history and associated work items with the source?

Once you have decided on the answers to these questions, your next steps will be determined by whether you wish to consolidate your branches or transition into a new environment. Transitioning to a new branch may be the easier of the two options. You can essentially lock off your existing branches and have everyone work in the new environment from the designated start point. This is most appropriate when applications that have been created and maintained by teams that were previously separate and do not have anything in common, other than they are now all products of a combined entity.

For consolidating branches, you will have to review your existing branches and determine what must be included. Once you have made those decisions, you will need to create a new mainline and proceed to migrate each designated branch. Along each step, you should verify that branch that is migrated does not break the new mainline. If at all possible, this should be done in a sandbox environment prior to attempting in a production environment.

It should be expected in either case that you may find some unexpected results when attempting consolidation or transitions, both at the time of and later.

Additional Resources

- "How to: Branch Folders and Files" at http://msdn.microsoft.com/en-us/library/ms181425(VS.100).aspx
- "How to: Merge Folders and Files" at http://msdn.microsoft.com/en-us/library/ms181428(VS.100).aspx
- Refer to Baseless Merge within Branching Guidance section

Question

Is there a preferred "Branch from version" value when creating a branch?

Answer

When creating a branch, there are five options available to choose the source from. They are: changeset, date, label, latest version and workspace version. Each of these options addresses a very particular need.

It is important to call out that TFS uses labels to refer not to a point in time but rather to any collection of files that have been grouped together by the user, so a branch that is created by label is often specific to the needs of an individual developer or task. The same thing applies when creating a branch from a workspace version. In TFS, workspaces are mapped from the developer's machines back to the server, so a branch would reflect the local files. Labels in TFS are changeable. Deletions are also not part of the label.

When creating a branch that will address the needs of the larger team, using date, latest version or changeset would be more preferable. Each one serves a different function. For example, if you were in a hotfix situation, creating a branch from the most relevant stable version would be most applicable. If you were fixing a bug or in maintenance mode and you knew where the break occurred, you would branch by changeset. If you are moving into the next iteration of your release, branching by date might be the best fit.

Additional Resources

"How to: Branch Folders and Files" at http://msdn.microsoft.com/en-us/library/ms181425(VS.100).aspx

Question

How do I manage permissions across branches for my team?

Answer

Managing access and permissions to source control should be carefully evaluated. Taking the time to evaluate the levels of access that are needed across roles and defining a roles and responsibilities matrix can provide you with a consistent and security focused solution. Consider using secure groups, based on Active Directory, since TFS can refresh automatically. Remember to include those people from outside your immediate teams who might also need access, such as IT Ops or Support. You may also need to take into consideration any corporate security or governance policies that apply.

In terms of source control, permissions in TFS 2010 can be granted separately from permissions to team projects, work items, portals and reporting services. A user may have distinct permissions on each branch to which they have access. Permissions may be inherited or granted explicitly. For each branch that you wish to set permissions to, take the following steps as referenced in the MSDN article "How to: Control Access to Team Foundation Version Control".

Branch permissions will have the same permissions as the main branch it was created from. It is important to set the permissions to a more restrictive level immediately after the branch is created.

It is good practice to set permissions to the level of access actually needed rather than using a more generalized model. Ideally, branch permissions should be restricted to your team members that actually require access to source code. The permissions should be specific to the branch rather than inherited. Implementing this practice will make your branches more secure against changes that were not meant for your branch. Otherwise it can easily happen that a developers checks in code to the wrong branch by accident. One scenario would be to grant all external developers access to a development branch but not to your main branch. The specified permissions on your main branch control who is allowed to merge the changesets from the development branch to the main branch.

In TFS 2010 there are two new permissions: The **Manage branch** permission is required to be able to convert folders to branches and vice versa. Additionally it allows editing of branch properties as well re-parenting the branch. Users with the **Merge** permission for a given branch can merge changes into this branch.

Additional Resources

- "How to: Control Access to Team Foundation Version Control" at http://msdn.microsoft.com/en-us/library/ms253158(VS.100).aspx

Question

How should bugs be handled over branches?

Answer

Branching of source code has no impact on work items (e.g. bugs). It means that the bugs are **not** cloned during the branching process. So, the handling of the bugs depends on the overall process:

Case 1:
Responsibility and ownership for the bugs should also move to the team being responsible for the new branch (either within the same team project or in another team project).

The bugs can be copied into a new or within the same team project and additionally assigned to different areas, which might correspond to branches. The fields in the newly created work item are pre-populated with the value of the corresponding fields in the source work item. The work item links will also be copied. Be aware of the following limitations:

- A work item belongs to the team project in which it was created and cannot be moved (only duplicated) to another team project
- Field information can only be transferred, if the target work item type has the same fields defined
- Permissions on both team projects are required

When copying into another team project, the original bug should be closed and linked with the copied bug.

Case 2:
Only source code will be branched (within the same team project or into another team project). Responsibility and ownership for the bugs will stay in the original team.

The bug fix can be implemented in the original source code or in the branch source code. When fixing the bug in the branch source code, the bug needs to be updated as resolved within the merge process.

Case 3:
Only source code will be branched into another team project. The responsibility for testing the bug fix is located in both projects (e.g. if the bug fix must also be retested in the new branch)

In this case the bugs can be duplicated into the new team project by copying the relevant bugs. So, both teams can track the quality verification of the bug fixes without dependencies.

Additional Resources

- “How to: Copy a Work Item” at http://msdn.microsoft.com/en-us/library/ms181321(VS.100).aspx
- “How to: Create or Delete Relationships Between Work Items” at http://msdn.microsoft.com/en-us/library/dd286694(VS.100).aspx

Question

What are labels and when should they be used?

Answer

In Team Foundation Version Control, a label is a marker that you can attach selectively to a set of otherwise unrelated file and folder versions in the source control server to facilitate their collective retrieval to a workspace for either development or build purposes. Typical scenarios to use labels can be to represent a snapshot of source code that was successfully built on a given day (Team Build does this automatically for each build) or store a version of some changes made to the code base.

Depending on the permissions granted to specific users, labels can be modified – files can be changed, added, removed from the label. While powerful in its own respect, labels need to be used with caution given that:

- Team Foundation Server does not retain history of changes made to the label.
- Given certain permissions, labels might be deleted or otherwise invalidated by changes, and there is no way of auditing those changes.
- There can be contention for a label name as label names must be unique throughout a specified scope
- Items deleted will not be available in a label.

Labels are recommended in cases where a snapshot of sources is needed for any future references/use. In scenarios where it can be guaranteed via permissions that the label will not be changed, we can use labels as markers for branching i.e. if you do not have an immediate requirement to branch but expect a need for branch in future, you can Label the source now and then branch from the label when required.

For any other development activity that needs to use a snapshot of sources over a longer duration of time, combined with the need for history and auditing of changes, and especially where several users might need to access the snapshot and possibly change the files and their contents, it is recommend to use branching. Branching by definition allows files and folders to diverge from their original state (while also preserving the original state), include history, and allow a rich set of file- and folder-level permissions to control changes.

Question

Can source code be branched across team projects?

Answer

Source code can be shared between Team Projects in various ways within the same Team Project Collection, including branching source from one Team Project into another. One of the common scenarios where a source code is branched between Team Projects is when we have a separate Team Project, which houses codebase of common functionalities and is referenced by multiple projects. A separate team project for common assemblies also provides a convenient place to manage artifacts associated with that shared source such as design documents and bugs. The downside of branching across team projects is that new sources can be brought to the branch only by merging it from the parent project.

Additional Resources

- Refer to Branching and Team Projects in the Branching Guidance

Question

When creating a new Team Project, when should one use "Create a new source control branch option"?

Answer

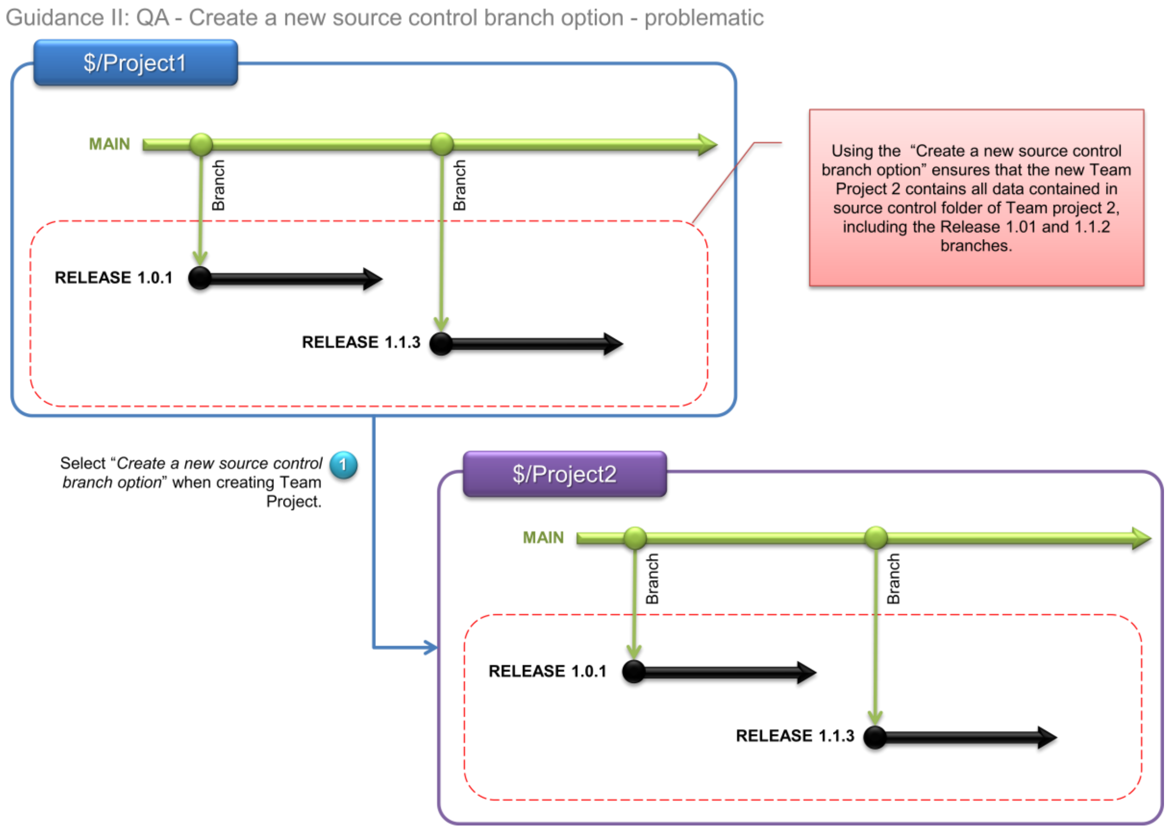
When creating a new Team Project, "Specify Source Control Settings" tab has the following options: "Create an empty source control folder" or "Create a new source control branch".

With the "Create a new source control branch" option selected, new Team Project source control folder will contain all data contained in source control folder of another existing Team Project. Essentially, by enabling this option you declare that "Team Project X will contain all of the sources in Team Project Y".

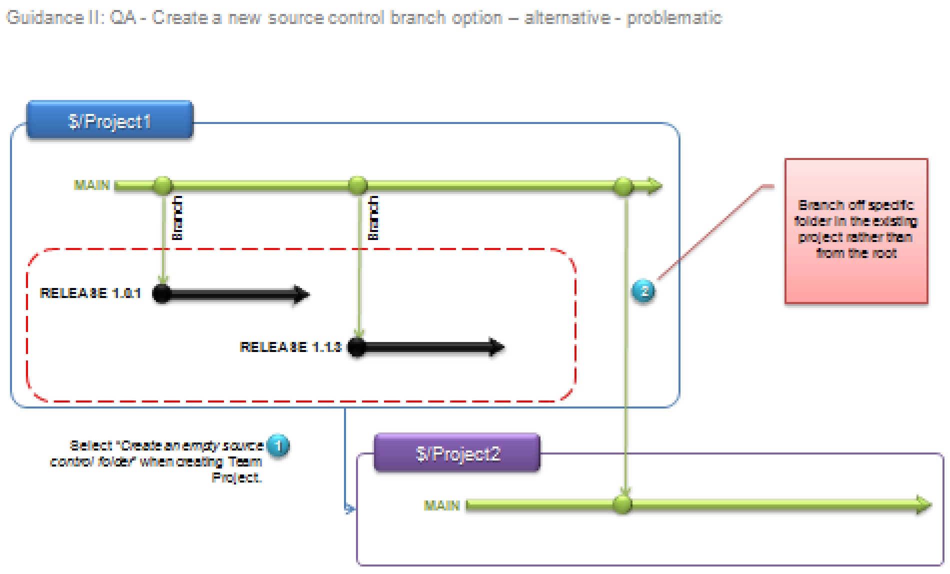
Thus the answer to the question whether to use this option depends on the internal structure and contents of the existing Team Project. If the branch is created from existing Team Project that contains code related only to single release, and new Team Project being created will be next release based solely on the previous code base, using "Create a new source control branch" may be a valid option.

However, if existing Team Project contains several releases (even if they are internal ones) the better option would be to "Create an empty source control folder" in a new Team Project, and then branch off specific folder in the existing Team Project rather than from the root. That would allow bringing over only desired folders, and generally provide for far better flexibility in creating desired folder structure in the a new Team Project. Below diagrams illustrate the problem:

Branching off Team Project


Guidance II: QA - Create a new source control branch option - problematic
$/Project1
MAIN
Branch
Branch
RELEASE 1.0.1
RELEASE 1.1.3
Using the "Create a new source control branch option" ensures that the new Team Project 2 contains all data contained in source control folder of Team project 2, including the Release 1.01 and 1.1.2 branches.
Select "*Create a new source control branch option*" when creating Team Project.
1
$/Project2
MAIN
Branch
Branch
RELEASE 1.0.1
RELEASE 1.1.3

Branching off specific folder


Guidance II: QA - Create a new source control branch option – alternative - problematic
$/Project1
MAIN
Branch
Branch
RELEASE 1.0.1
RELEASE 1.1.3
2
Branch off specific folder in the existing project rather than from the root
Select "*Create an empty source control folder*" when creating Team Project.
1
$/Project2
MAIN

Question

What is baseless merge and how does it compare to regular merge?

Answer

A baseless merge allows merging two folders that do not already have merge relationship. Normally a merge relationship is created when a new branch is created. The parent branch and child branch have a merge relationship. Two different child branches, although they share the same parent (sibling branches) do not have a merge relationship directly with each other.

Once baseless merge is performed (currently only possible using the command line, it becomes possible to perform ordinary merges between the branches (as if the two branches had a normal merge relationship between them).

Though baseless merge is helpful when merge between two logically related folders is absolutely necessary in the absence of main-branch relationship, it shouldn't be seen as a normal process but rather as an exception. After all, if the branching model is effectively planned and implemented, a baseless merge shouldn't be necessary.

As an administrator of Team Foundation Server 2010, you should understand that baseless merges are likely to be an indication that the branching structure implemented or the process around it is not serving the purpose of simplifying and standardizing the branching/merging model of your source code repository. Moreover, frequent baseless merges also mean that too much manual intervention is currently necessary in order to deliver branching/merging to your source code. This manual intervention can eventually lead to human error and consequently inconsistent source code state. A similar risk as the one mentioned earlier in this section about cherry picking.

Additional Resources

- "How To: Perform a Baseless Merge" at http://www.codeplex.com/VSTSGuidance/Wiki/View.aspx?title=How%20To%3A%20Perform%20a%20Baseless%20Merge%20in%20Team%20Foundation%20Server&referringTitle=Whats%20New
- Refer to Baseless Merge within Branching Guidance section

Question

Can branches be deleted?

Answer

Deleting branch is not different from deleting any source control folder. One difference to be aware of when deleting branch is the effect it may have on future merge operations.

TFS is possible either to merge (FI) a parent branch to a direct child branch or to merge (RI) a child branch its immediate parent (unless you perform *baseless* merge or *reparent branch* after a baseless merge); thus if intermediate branch is deleted, certain merge paths will be invalidated.

Consider the following example:

1. Create branch A to B
2. Create branch B to C

Merge from A to C is possible (from A to B, from B to C) as is merge from C to A (C to B, B to A). However, if B is deleted, those merges become impossible.

Therefore, if you were to delete branch folder, consider if it has any child branches created off it. If such branches exist, consider making the branch invisible through the use of source control permissions; that would allow future merges while hiding the folder from the end user.

Additional Resources

- Refer to Baseless Merge within Branching Guidance section
- "Team Foundation Server Permissions" at http://msdn.microsoft.com/ru-ru/library/ms252587(VS.100).aspx

Question

When merging between two branches with "All changes up to a specific version" option selected, what version type is preferred to specify the version ("Latest Version" [default], "Changeset", "Date", "Label" or "Workspace" version)?

Answer

Depending on the option selected, different changesets will be merged as described below:

- When "Latest Version" option is selected, all unmerged changesets from source branch checked in prior to the moment of merge will be merged to the target branch. However, since there may be active development ongoing on the source branch, the exact changes being merged may not be well-defined.
- When "Changeset" option is selected, all changesets from the source branch checked in up to the specified changeset will be merged to the target branch (specifying changeset is equivalent to specifying date, where date is the date of the changeset check-in).
- When "Date" option is selected, all changesets from the source branch checked in up to the specified date will be merged to the target branch.
- When "Label" option is selected, all changesets labeled in the source branch will be merged to the target branch. Because labels in TFS do not include "delete" changes, delete changes will never be merged over to the target branch.
- When "Workspace" option is selected, all source branch changes up to the versions in the specified workspace will be merged to the target branch. In a manner similar to "Label" option, changes of type "delete" will not be merged.

Thus, as the result of limitations of "Label" and "Workspace" options, those are not recommended for performing merges unless no delete changes are ever performed. "Latest Version" option may be used when one is reasonably sure what the latest version contents are. The temporary "freeze" on the source branch may be achieved by applying the lock prior to the merge. Using "Changeset" or "Date" options provide well-defined checkpoint of what merge contents are and should be used where possible.

If you are unsure about which changesets you did not merge yet, then you might want to select the "Selected changesets" option, which will give you a list of the pending changesets that have never been merged.

Additional Resources

- "Deceptive Allure Of Merging With Labels" at http://blogs.msdn.com/tfsvcs/archive/2007/03/22/the-deceptive-allure-of-merging-with-labels.aspx

Question

What is the difference between folders and branches?

Answer

Starting with TFS 2010 there is a distinction between branches and folders in version control. Assuming the necessary permissions a user can convert folders to branches (and vice versa).

This new concept of having a branch as a first-class object enables some branch-specific features as well as the ability to store properties like the owner and a comment to each branch, making branch management easier for teams with a large number of branches.

After converting a folder to a branch you can use many of the new features related to branches like visualizing the branch hierarchy or visually tracking changesets in order to see what branches contain the changeset, and when changesets were merged into specific branches.

When you upgrade from TFS 2008 to TFS 2010 you should convert branch folders to branches. Note – not all folders in TFS 2008 are branches. Some folders are containers, and will continue as folders in TFS 2010.

On the security side, there are two new permissions: The **Manage branch** permission is required to be able to convert folders to branches and vice versa. Additionally **Manage branch** permission allows editing of branch properties, creating new branches from a branch, and re-parenting a branch. Users with the **Merge** permission for a given branch can merge changes into this branch.
To summarize, when you want to create a new child branch from a parent branch you need **Manage branch** permissions on the **source** branch (the parent branch). When you want to merge changes **into** a branch, you need **Merge** permission on the **target** branch

Additional Resources

- "How to: Branch Folders and Files" at http://msdn.microsoft.com/en-us/library/ms181425(VS.100).aspx
- "How to: View the Branch Hierarchy of a Team Project" at http://msdn.microsoft.com/en-us/library/dd465202(VS.100).aspx
- "How to: View Where and When Changesets Have Been Merged" at http://msdn.microsoft.com/en-us/library/dd405662(VS.100).aspx
- "Team Foundation Server Permissions" at http://msdn.microsoft.com/en-us/library/ms252587(VS.100).aspx

Question

What is the “Reparent Branch” function and when should it be used?

Answer

Reparent Branch is a feature that can be used to establish a parent-child relationship between baseless merged branches as well as to alter existing branches' parent-child relationship in a branch hierarchy.

However, to "reverse" an existing parent-child relationship, the child branch will need to be disconnected from the parent branch via "No parent" option, and then re-parent the former parent branch to the former child branch.