TFS

Software Development Branching Model

File Name	Version Number	Author
Software Development Branching Model	1.0	Gaurav Lakhani

SOFTWARE DEVELOPMENT BRANCHING MODEL

INTRODUCTION

Branching Model needs to be defined to streamline and standardize the process of development, test, build and release. As the product grows, branching model may become complex to support development and releases.

TFS Terminology (Mainline, Development, Releases)

SVN Terminology (Trunk, Branches, Tags)

What is Mainline \Development \Releases and what is their use?

Meaning of Branches – Branches are copies of the Mainline in the Source control repository (TFS\SVN).

Mainline

This is the mainline for development. Changes from other branches are integrated\merged here.

This should always be kept stable at all times so that it’s always ready to release.

Development

Development contains branches created from main. It’s used for parallel development of features/bug fixes/enhancements and hence isolates active development, without impacting main.

Releases

Releases contain branches you have already shipped to production but now need to maintain for customers. This provides isolation from active development occurring in your development branch. It also contains a current release branch which is branched from **Main.** This may be locked down prior to release. This is for supporting releases in production.

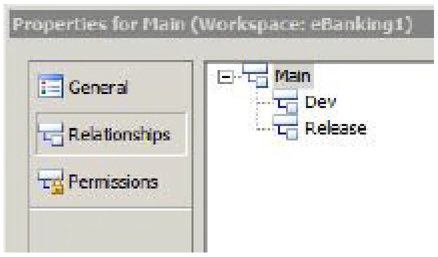
Labels are used to create a snapshot of a project at a particular stage. So that at a later date you can refer back and use it to view, build or even rollback to the previous state. Team Foundation Server does not retain a history of changes made to the label.

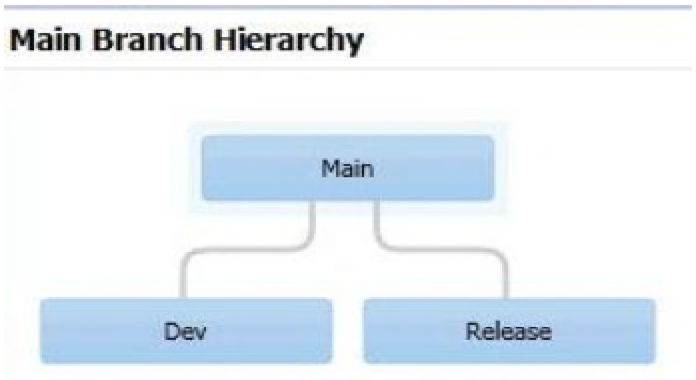
Applying a label provides many of the same benefits as creating a branch, but typically with lower cost and complexity.

“Do not branch unless it becomes necessary for your development team. Branching introduces additional source tree maintenance and merging tasks.” Hence more merge cost.

Now in TFS, when we create a project, 3 folders are created: MAIN, DEV and RELEASE.

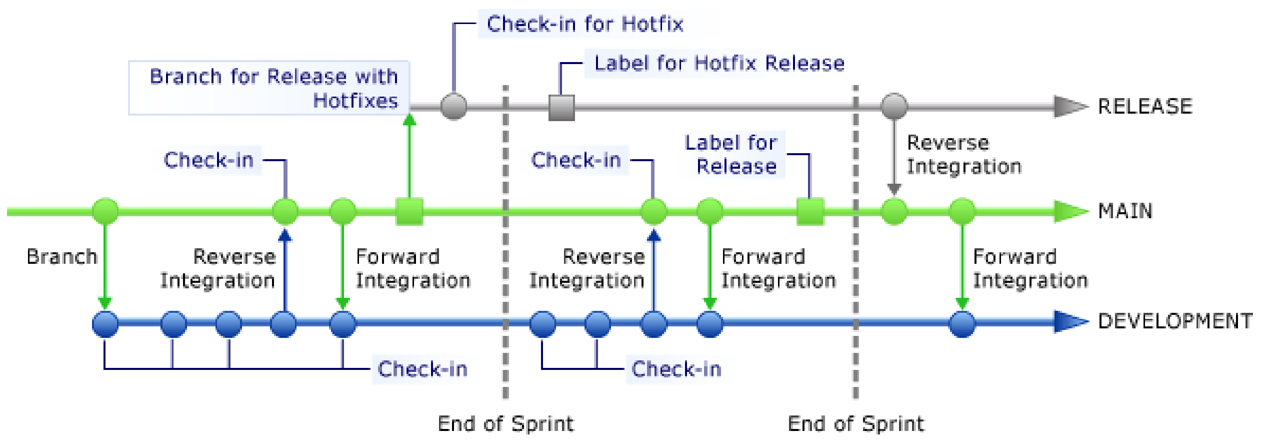
In actual, they all are branches but the way they are used is different.





General considerations for all branches (Main, Dev and Release):

1. Build daily to check stability.
2. Implement continuous integration builds to immediately identify quality issues.
3. Code flow, the movement of changes between child and parent branches, is a concept all team members must consider and understand i.e. FI and RI (discussed in this document)



Branching Model for TFS – Figure 1

1. If your project or team is small, then simply work in MAIN. As the project will grow, branching model will also grow. So it's important to understand.
2. Now, in a bigger model as shown above, when coding for a new functionality or sprint or milestone will complete in DEV, code needs to build successfully, pass integration tests if any. Then it needs to be merged back to MAIN. This is known as **REVERSE INTEGRATION.**
3. If you merged from MAIN to DEV then it's known as **FORWARD INTEGRATION.**
4. You can assign merging to a team member who can create branch and do FI and RI so that he can become expert of that.
5. FI is important when MAIN also integrates changes from other branches.
6. Its recommended that MAIN should have gated check-in build type. This means that the DEVELOPMENT branch must pass all requirements for the MAIN branch before you can commit a reverse integration. The DEVELOPMENT branch should run a continuous build type because your team must know as soon as possible when a new check-in affects the DEVELOPMENT branch.
7. **FREQUENCY of INTEGRATION** - Reverse integration and Forward integration should occur at least when you complete a user story.
8. Although each team might define completeness differently, completion of a user story generally means that you complete both the functionality and the corresponding unit tests. You can reverse integrate to the MAIN branch only after unit tests have verified the stability of the DEVELOPMENT branch.
9. If you have more than one work (DEVELOPMENT) branch, forward integration to all work branches should occur as soon as any branch integrates into the MAIN branch. Since the MAIN branch is kept stable. So, forward integration is safe. Conflicts or failures at the work branches might occur because you cannot guarantee that the work branches are stable.

10. It is important that you resolve all conflicts as soon as possible. By using a gated check-in for the MAIN branch, you help make the reverse integration much easier because quality gates help avoid conflicts or errors in the MAIN branch.
11. you can reverse integrate to the MAIN branch only when you complete all the in-progress work. It is recommended that you group user stories by similar size because you do not want a large user story to block the integration of many small ones. You can split the two sets of user stories into two branches.
12. Ideally, your team should be able to release code at the end of any sprint. By using TFS, you can label a branch to take a snapshot of the code at a specific point in time.
13. Because you must implement updates on releases, creating a branch for a release helps your team continue to work independently on the next sprint without creating conflicts with future releases.
14. Figure shows a branch that contains code for an update and that is reverse integrated into the MAIN branch after a release at the end of the second sprint.
15. When you create a branch for a release, you should create that branch from the MAIN branch, which is the most stable. If you branch for release from a work branch, it can cause integration challenges because the stability of work branches is not guaranteed.