TFS Branching Guidance - Main

Bijan Javidi, James Pickell, Tina Erwee, Willy-Peter Schaub

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

Introduction

Welcome to the Team Foundation Server Branching guide! Since the original release of this guidance in 2007 we've received many encouraging messages confirming the usefulness of this guide in planning, executing and maintaining branches, in allowing TFS users to devise, execute and maintain their branch plans. Using feedback from users, we, a team of Microsoft VSTS Most Valuable Professionals (MVPs) and Microsoft Services consultants have refreshed the guidance presented in the original document produced as part of a VSTS Ranger project. While some of the original document has been retained, you will notice 3 new themes through the guide.

This guide targets the Microsoft "200-300 level" users of TFS. The target group is considered as intermediate to advanced users of TFS and has in-depth understanding of the product features in a real-world environment. Parts of this guide may be useful to the TFS novices and experts but users at these skill levels are not the focus of this content.

Before executing on your branch plan, pay attention to this cautionary message - every branch you create does have a cost so make sure you get some value from it. The mechanics of branching in TFS are simplified to a single right click→branch command. However, the total cost of branching is paid by reduced code velocity to main, merge conflicts and additional testing can be expensive. Throughout this guidance we ask the user to confirm that a branch is really needed and always ask the question "how does this branch support my development project?" Readers can casually think of this as the branch ROI. This is not a firm metric that we can collect; rather a general question that should be asked before creating a branch. By justifying each branch in terms of required level of isolation for development or release activity, we hope to avoid situations where this guidance was assumed to be an "off the shelf" solution and an unnecessarily complex branch plan was created.

Once you begin thinking of branches in terms of cost and benefit we hope most teams will find a productive middle ground that gives them the right amount of branching to achieve their business goals.

Branching and merging of software is a very large topic. It is an area where there is a lot of maturity in the software industry. This document focuses on applied and practical examples of branching that you can use right now. We purposely avoid any academic discussion on the topic of branching and focus on patterns that work for the majority of business cases we have seen.

Our hope is that this guide covers the majority of our customer branching needs. In addition to this guide you will find separate branching scenarios, Q&A, tutorials and updated branch diagrams that will grow over time to include additional edge cases. We encourage ongoing contributions from our user community to submit ideas for new Scenarios that we will add to these packages. These new packages that will become available on Codeplex will be refreshed based on user feedback.

Vocabulary

This guide uses common vocabulary and general categories to group work and where check-ins are made. Language may vary in your organization so some translation may be required.

DEVELOPMENT branches – changes for next version work.

MAIN branch – This branch is the junction branch between the development and release branches. This branch should represent a stable snapshot of the product that can be shared with QA or external teams.

SERVICE PACK (SP) – A collection of Hotfixes and features targeting a previous product release.

HOTFIX – A change to fix a specific customer blocking bug or service disruption.

RELEASE branch – A branch where ship stopping bug fixes are made before major product release. After product release this branch usually becomes read-only.

FORWARD INTEGRATE (FI) – Merges from parent to child branches.

REVERSE INTEGRATE – Merges from child to parent branches.

RELEASE VEHICLE – How your product gets to your customer (e.g. major release, Hotfixes and/or service packs).

Branch Plan – quick start

"Save your creativity for your product… not the branch plan." – anonymous

Below are three branch plans that represent Basic, Standard and Advanced software development projects. The elements of these plans are additive so starting with the Basic plan will allow you to transition to the Standard plan if your product becomes more complex.

The most common reason for adding complexity to a branch plan are additional release vehicles (i.e. Service Packs and Hotfixes). Additional release vehicles will require a branch plan that supports concurrent development for each of these. Minimizing the number of release vehicles is the key to keeping your branch plan simple.

The goal of the Basic, Standard and Advanced branch plans presented below is to cover most customer branching scenarios. Please post your scenarios not covered below to the community section associated with this document at http://codeplex.com/TFSBranchingGuideII. Our hope is to leverage the Codeplex community to cover alternative (but still valid) branch cases.

Basic Branch Plan

Below is a basic plan that enables concurrent development for your next release, a stable main branch for testing and a release branch for any ship blocking bug fixes.

Multiple development areas are supported by creating additional development branches from main. These are peers to each other and children of main.

Additional releases are supported by creating additional release branches for each product release. Each release branch is a child of main and a peer to each other (e.g. release2.0 branch is peer to release3.0 and both are children of main).

Once the release branch is created main and the development branches can start taking changes approved for the next product release.

The Basic branch plan will work well for your organization if you meet some of the following criteria:

1. You have a single major release that is shipped (i.e. a single release vehicle) to customers.
2. Your servicing model is to have customers upgrade to the next major release.
3. Any fixes shipped from the release branch will include all previous fixes from that branch.

Key elements of this plan include

1. DEVELOPMENT (dev) branches for next version work.
    a. Focus on wide, flat branches to enable steady code flow to MAIN and then back to peer DEVELOPMENT branches
    b. Work in DEVELOPMENT branches can be segregated by feature, organization, or temporary collaboration.
    c. Each DEVELOPMENT branch should be a full branch of MAIN.
    d. DEVELOPMENT branches should build and run Build Verification Tests (BVT's) the same way as MAIN.
    e. Forward Integrate (FI) with each successful build of MAIN
    f. Reverse Integrate (RI) based on some objective team criteria (e.g. internal quality gates, end of sprint, etc.).

2. RELEASE branch where you ship your major release from.
    a. RELEASE is a child branch of MAIN.
    b. Your major product releases from the RELEASE branch and then RELEASE branch access permissions are set to read only.
    c. Changes from the RELEASE branch RI to main. This merge is one way. Once the release branch is created MAIN may be taking changes for next version work not approved for the release branch
    d. Duplicate RELEASE branch plan for subsequent major releases.
3. Changes should be checked into one branch only
    a. All branches have a merge path back to MAIN.
    b. No need for baseless merges.

Standard Branch Plan

As you add additional release vehicles you may need to create additional branches in the production/release area to enable concurrent development. The Standard branch plan below introduces a new release branch to support an additional release vehicle. Most organizations will call this a servicing branch to enable development of Hotfixes and Service packs.

This plan will work well for your organization if you meet some of following criteria:

1. You have multiple ship vehicles (e.g. major release and additional service packs for that release).
2. You want to enable concurrent development of service pack and next version products.
3. You have any compliance requirements that require you to have an accurate snapshot of your sources at release time.

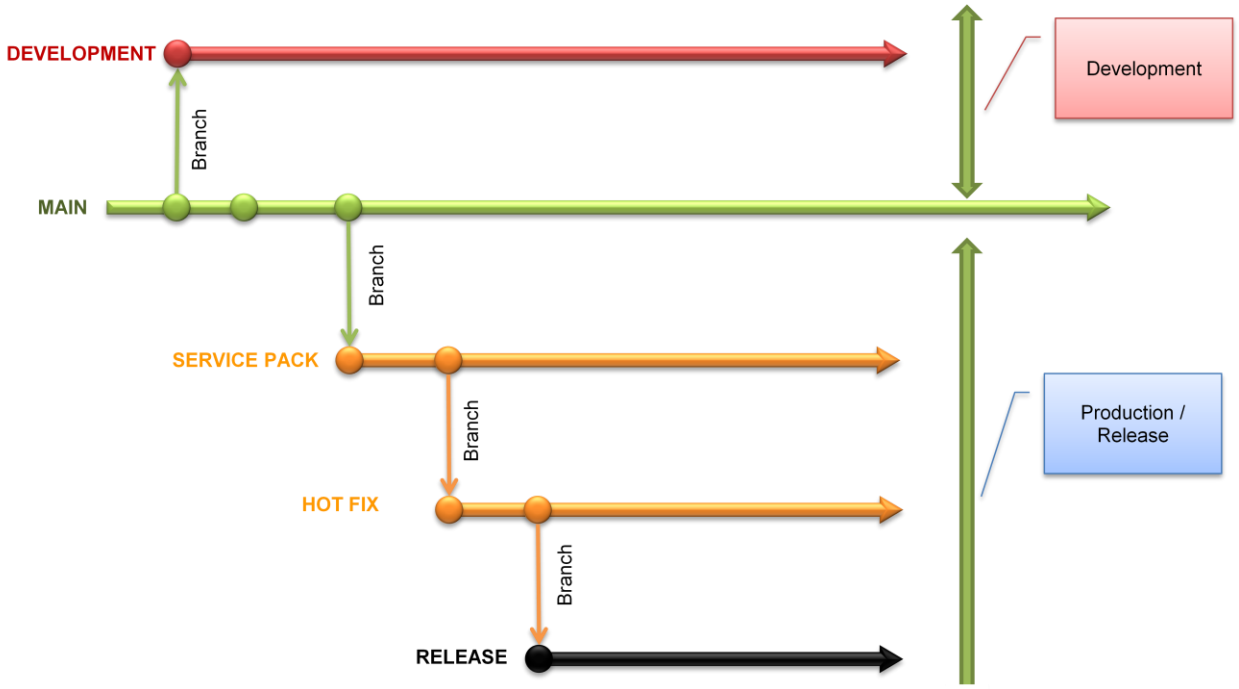
All of the guidance any key points from the Basic plan applies to the Standard plan. The Standard plan has these additional items to consider.

RELEASE branches for release safekeeping and Service Pack work

1. RELEASE tree (i.e. SP and RELEASE) are branched from MAIN at the same time to create MAIN→SP→RELEASE parent/child relationship.
2. Product releases from the RELEASE branch and then that branch is changed to read only.
3. Servicing changes are checked into the Service Pack (SP) branch.
4. Changes SP branches merge one-way to MAIN (SP→MAIN).
5. Ship stopping bug fixes checked into the release branch should merge back to MAIN through the SP branch (RELEASE→SP→MAIN).
6. Duplicate RELEASE tree plan for subsequent major releases.

Advanced Branch Plan

The Advanced plan is for products have must support many release vehicles and servicing scenarios. The plan allows for concurrent development of a major release, service packs, Hotfixes and next version work.



All of the guidance any key points from the Basic and Standard plans applies to the Advanced plan. The Advanced plan has these additional items to consider.

1. RELEASE branches for release safekeeping, HOTFIX and Service Pack work
    a. RELEASE tree (i.e. SP, HOTFIX, and RELEASE) are branched from MAIN at the same time to create MAIN→SP→HOTFIX→RELEASE parent/child relationship.
    b. Product releases from the RELEASE branch and then that branch is changed to read only.
    c. Check-in based on which release the change applies to (e.g. Hotfixes are checked into the HOTFIX branch).
    d. Changes in HOTFIX and SP branches merge one-way through intermediate branches to MAIN (HOTFIX→SP→MAIN).
    e. Duplicate RELEASE branch plan for subsequent major releases.

The plans above covers the majority of software development activities. Using these plans as a base will enable your teams to spend more time on developing high quality code and assures that each branch has a specific role. The additional content below may help you determine where to deviate from the plan above.

MAIN Branch

The MAIN branch is junction between development and RELEASE branches. Changes in the MAIN branch will FI into every DEVELOPMENT branch, so it is critical that builds from MAIN remain high quality. At minimum this means MAIN must remain buildable and pass all build verification tests

Attributes of MAIN

- Main should build daily to give the team a daily cadence to work toward.
- Every branch should have a natural merge (RI) path back to MAIN (i.e. no baseless merges).
- Breaks in MAIN need to be fixed immediately.
- No direct check-ins to MAIN branch; only build and BVT fixes.
- Successful MAIN build indicates child DEVELOPMENT branches should FI from MAIN.
- QA teams should be able to pick up any MAIN build for testing.

Code flow, the movement of changes between child and parent branches, is a concept all team members must consider. As the number of DEVELOPMENT branches increases the need to FI each successful MAIN build increases. Any DEVELOPMENT branch that is more than a couple days out of sync with MAIN is effectively working in the past.

About versioning…

One strategy to keep track, at a glance, of how far out of sync a DEVELOPMENT branch is from MAIN is to increment the build number in MAIN only. When DEVELOPMENT branches FI from MAIN they get the latest version resource from MAIN. If MAIN builds daily then the build number would increment each day.

> Example:
> MAIN build 1.0.**27**.0
> Dev_team1 build 1.0.**25**.0
>
> The Dev_team1 branch has not FI'd MAIN in 2 days if MAIN builds daily.

Teams that adopt this strategy will frequently append a date time stamp to the build number to prevent overwriting builds on the release share and provide clearer reporting since the dev builds may keep the same build number for a few days.

All I need is MAIN…

Some teams are small enough that only one branch, MAIN, is needed. This is great but almost always short-lived. Eventually individuals or teams will need some sort of isolation to work on next version work, complicated bug fixes or breaking changes.

If you do not create a specific place for development work, developers will create their own. The danger here is that the ad-hoc branch created by the developer may not have a natural branch path back to MAIN.

Since these situations are inevitable it is best to have the development branch plan ready when needed.

If ever you need advice on when to branch, think of this:

Create a new branch when the check-in policy is so restrictive you can't do work.

This means when the check in policy of your main branch is to take version 1.0 changes only and you have a version 1.1 changes then that would be a good signal to create a child branch of main for your 1.1 work.

DEVELOPMENT Branches

"Your branch distance from main is equal to your level of insanity" - anonymous

Branching enables parallel development by providing each development activity for the current release its own self-contained snapshot of needed sources, tools, external dependencies and process automation. Such a self-contained snapshot effectively enables each development activity to proceed at its own pace, without taking any dependency on another. It follows that these snapshots are allowed to diverge their respective sources along the particular development activity they are involved with – fixing a bug, implementing a feature or stabilizing a breaking change.

Before creating a DEVELOPMENT branch, make sure you can do the following:

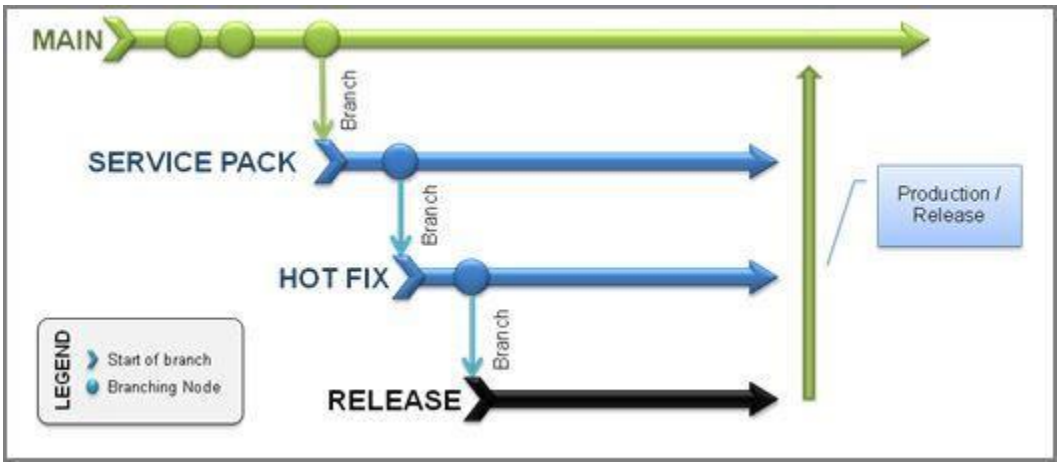
- Select a parent branch – if your changes are focused on your next product release then the parent branch should be MAIN.
- Branch from a recent known good state of the parent – start your branch from the latest successful build of the parent branch. There should be a label associated with the parent build that you can use as a starting point for your child branch. On day one of your DEVELOPMENT branch it should be in the same state as its parent (i.e. build and pass BVT's successfully).
- FI frequently – your goal should be to be no more than 1-2 days out of sync with main. Ideally you should FI every time the parent branch builds and passes BVTs.
- RI from child to parent based on quality. Minimum RI requirements are
    - Be in sync with parent branch
    - Build successfully
    - Pass BVT's

RELEASE Branches

"There will be hotfixes!" – anonymous

Your release branch plan should be built around your software release vehicles. A release vehicle is how your software is delivered to your customer. The most common release vehicles are the major release, Hotfix and service pack. In a software plus services scenario the names may be different however and the release may be more frequent.

The Advanced branch plan assumes that your software will require concurrent Hotfix, service pack and next version (MAIN). Almost all software servicing have a need for these classes of release so we setup the release branch plan to support these from the very beginning. Note, that all 3 branches are created at the same time to ensure the correct parent child relationship.



A successful release branch strategy enables the following 3 scenarios.

1. Developers only need to check in once based on which release vehicle the change is for (i.e. Hotfixes go into the product Hotfix branch).
2. No need for baseless merges. Create a natural merge path back to main by creating a hierarchal branch structure based on your release vehicles.
3. Reduce risk of regressions. By creating a parent/child branch relationship between main->SP-> and Hotfix branches changes are naturally merged into future release (i.e. Hotfixes merge into the SP branch on their way to main) reducing risk of bug regressions in future releases.

After the release branches are created changes from main should not FI into the release branches. Changes should merge – one way – from release to main. Also, changes should always merge through intermediate branches (i.e. release->HF->Service Pack->main) to ensure that bug fixes remain consistent in subsequent releases.

Note, once you make your final change to your major release (i.e. the release branch in the diagram above) this branch should be set to read only. This branch is retained for compliance purposes only.

Since there is no change history in TFS for labels the only sure way to know the state of your sources is to branch and ship your product from that branch.

Integrations

"…when ambition meets faith". – anonymous

The main cost of a branch is in the time and potential for mistakes to be made during integrations. There is one solution to resolve this. The general guidance is below.

- Keep branching to a minimum.
- FI frequently ideally do not let your branch get more than 1-2 days out of sync with the parent.
- RI frequently based on build and automated test results.

Build Quality

"It works on my machine." – anonymous

A successful build only indicates that sources are syntactically correct and generated no build time (i.e. compile, link, packaging) errors. Before committing more resources, usually people, hardware and time, to testing this build it's important to determine if this build is "testable." This is the role of the build verification test (BVT).

BVT's should meet the following 3 requirements.

1. Fully automated
2. 100% reproducible
3. Failure of a BVT blocks more than 30% of your test organization.

The only 2 outcomes from a BVT failure should be a code change to fix the break or a test change to fix a broken test. If a BVT is frequently generating failures that are ignored then the BVT should be removed.

Sample BVT's for an operating system would include, setup, joining a domain and writing a file to disk. Remember, these are very basic tests to indicate if the build is testable. More extended testing can be done outside of the build system.

Conclusion

"No fate but what we make." – anonymous

We expect the branch plan presented here to work for most software development projects. If you have an unusual situation consider reviewing the scenarios companion package of this document to see if others have provided a solution to your situation. However, don't try to make your branch plan more complicated than what we have presented here unless you can justify the additional branches or unusual patterns.