

Control de versiones con TFS

Parte 1

Estrategias de Branching

Spanish translation
by Juan María Laó Ramos

Visual Studio ALM Rangers

La información contenida en este documento representa la visión de Microsoft Corporation sobre los asuntos analizados a la fecha de publicación. Dado que Microsoft debe responder a las condiciones cambiantes del mercado, no debe interpretarse como un compromiso por parte de Microsoft, y Microsoft no puede garantizar la exactitud de la información presentada después de la fecha de publicación.

Este documento es sólo para fines informativos. MICROSOFT NO OFRECE NINGUNA GARANTÍA, EXPRESA, IMPLÍCITA O LEGAL, EN CUANTO A LA INFORMACIÓN CONTENIDA EN ESTE DOCUMENTO.

Microsoft publica este documento bajo los términos de la licencia Creative Commons Attribution 3.0 License. Todos los demás derechos están reservados.

© 2014 Microsoft Corporation.

Microsoft, Active Directory, Excel, Internet Explorer, SQL Server, Visual Studio, and Windows son marcas comerciales del grupo de compañías de Microsoft.

Todas las demás marcas son propiedad de sus respectivos dueños

The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, you should not interpret this to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

This document is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS DOCUMENT.

Microsoft grants you a license to this document under the terms of the Creative Commons Attribution 3.0 License. All other rights are reserved.

© 2014 Microsoft Corporation.

Microsoft, Active Directory, Excel, Internet Explorer, SQL Server, Visual Studio, and Windows are trademarks of the Microsoft group of companies.

All other trademarks are property of their respective owners.

Índice

Prólogo.....	4
Introducción	5
¿Qué hay de Nuevo?	6
Conceptos	9
Vocabulario	9
Conceptos de Branching	9
Tipos de Branch	12
Estrategias de Branching.....	14
Main Only	14
Development Isolation	15
Release Isolation.....	15
Development y Release Isolation	16
Servicing y Release Isolation	16
Servicing, Hotfix, y Release Isolation	18
Code Promotion	19
Feature Isolation.....	19
Estrategias alternativas.....	21
Adapta tu proceso de branching para cosas inesperadas.....	21
Feature Toggling.....	22
Integración continua.....	24
Tutoriales.....	26
De la nada a la complejidad o no.....	26
Adapta tu proceso de branching para los casos excepcionales.....	29
Casos reales.....	31
Entregar software en intervalos desde días a meses.....	31
FAQ.....	34
Hands-on Lab (HOL) – De la simplicidad a la complejidad, ¿o no?	37
Ejercicio 1: Configuración del entorno	37
Ejercicio 2: MAIN Only – Reglas simples.....	43
Ejercicio 3: Development Isolation... bienvenido al branching	49
Ejercicio 4: Feature Isolation... ¡un especial!	56
Ejercicio 5: Release Isolation... alarma de auditoría	64
Ejercicio 6: Servicing & Release Isolation	67
Conclusión	70

Prólogo

Desde la primera versión de la guía de Branching de TFS, han cambiado muchas cosas en el mundo del control de versiones. Las soluciones de control de versiones están por todas partes, y muchas de ellas incluyen integración con builds, seguimiento de proyectos, y otros servicios. Los sistemas de Control de Versiones Distribuidos ya no son un nicho, y han cambiado lo que para muchos desarrolladores significa “tener versionado el código”. Hay muchos más desarrolladores que usan control de versiones que antes – y eso es algo fantástico para miles de millones de usuarios finales de su software.

Más desarrolladores usando control de versiones también significa, ahora más que nunca, que la industria necesita guías sólidas, prácticas y fáciles de entender. Esta guía, y todas las que la precedieron, se esfuerzan en eso – ofrecer una guía sobre el control de versiones que todo equipo de desarrollo necesita para ser efectivo y a la vez accesible y flexible. En esta última edición, hemos racionalizado la orientación y simplificado conceptos con el objetivo de ayudar a equipos de todas las formas y tamaños para que consigan una estrategia que les permita conseguir la flexibilidad y agilidad que los equipos de desarrollo modernos necesitan.

También quiero decir que esta guía no podría ir por esta versión sin los lectores. Gracias a todos los que habéis contribuido con el feedback y las ideas que nos han ayudado a darle forma a esta guía durante los últimos años. Como en versiones anteriores, si ves algo en esta guía que te gustaría cambiar o mejorar, ¡háznoslo saber!

¡Feliz versionado!

Matthew Mittrik – Program Manager, Cloud Dev Services

Introducción

Esta guía tiene como objetivo ofrecer una orientación perspicaz y práctica con respecto a las estrategias de branching con Team Foundation Server. El branching de software es un tema muy amplio y hay que tener en cuenta que cada organización es diferente, y no hay 'una talla para todos' en las estrategias de branching.

Estas directrices sirven como punto de partida para la planificación y es posible que desviarse en algunos casos añada valor a tu entorno. En esta guía, el foco está puesto en escenarios reales y prácticos de branching que podrás aplicar inmediatamente. Evitamos hablar de los detalles de la mecánica de branching para centrarnos en las mejores prácticas y en ejemplos reales de patrones que funcionan para muchos casos que hemos visto. Esta guía es un conjunto de guías tal como se indica en el capítulo **¿Qué hay de nuevo?**, en la página [6](#).

A quién va dirigida

En esta guía, nos centramos en **Garry**, el jefe de equipo, **Doris**, la desarrolladora, y **Dave**, el administrador del TFS. Leed [ALM Rangers Personas and Customer Profiles](#)¹ para más información sobre estos y otros personajes.

Visual Studio ALM Rangers

Los Visual Studio ALM Rangers ofrecen una guía profesional, experiencia práctica y proporcionan soluciones a la comunidad ALM. Son un grupo especial compuesto por miembros del grupo de producto de Visual Studio, de Microsoft Services, de Microsoft Most Valuable Professionals (MVP) y de Visual Studio Community Leads. La información sobre sus miembros está disponible aquí [online](#)².

Colaboradores

Matthew Mitrik, [Michael Fourie](#), [Micheal Learned](#) y [Willy-Peter Schaub](#).

Un agradecimiento especial a los equipos e ALM Ranger que sentaron las bases con las versiones v1 y v2: Anil Chandr Lingam, Bijan Javidi, Bill Heys, Bob Jacobs, Brian Minisi, Clementino de Mendonca, Daniel Manson, Jahangeer Mohammed, James Pickell, Jansson Lennart, Jelle Druyts, Jens Suessmeyer, Krithika Sambamoorthy, Lennart Jansson, Mathias Olausson, Matt Velloso, Matthew Mitrik, Michael Fourie, Micheal Learned, Neno Loje, Oliver Hilgers, Sin Min Lee, Stefan Mieth, Taavi Koosaar, Tony Whitter, Willy-Peter Schaub, y la comunidad ALM.

Uso del código de ejemplo, erratas y soporte

Todo el código Fuente y la revisión de esta guía está disponible para su descarga a través del sitio [Version Control Guide](#)³ (anteriormente conocida como Branching and Merging Guide). Puedes contactar con el equipo usando el foro de CodePlex.

Más ALM Rangers y otros Recursos

[Understanding the ALM Rangers](#)⁴

[Visual Studio ALM Ranger Solutions](#)⁵

[Branching Taxonomy, by MS Research](#)⁶

[The Effect of Branching Strategies](#)⁷

¹ <http://vsarguidance.codeplex.com/releases/view/88001>

² <http://aka.ms/vsarindex>

³ <http://aka.ms/treasure18>

⁴ <http://aka.ms/vsarunderstand>

⁵ <http://aka.ms/vsarsolutions>

⁶ <http://research.microsoft.com/apps/pubs/?id=209683>

⁷ <http://research.microsoft.com/apps/pubs/default.aspx?id=163833>

¿Qué hay de Nuevo?

Esta guía ofrece un estilo más compacto, lo que hace que sea más fácil consumir desde diferentes dispositivos sin tener que sacrificar ningún contenido. Los autores han actualizado el contenido y lo han alineado con las últimas tecnologías de Visual Studio. Esta última guía se incorpora feedback clave de los lectores de las anteriores versiones.

NOTA

Branching es barato... ¡**Merging** es caro!

Esta guía parte de no tener ninguna estrategia de branching ni merging. En lugar de elegir la estrategia más adecuada y evolucionar los procesos y habilidades de tu equipo, deberías adoptar y evolucionar una o más estrategias cómo y cuándo sea necesario.

Los cambios de estrategia

Los nombres de estrategia simples, básicos y avanzados que se usaban en las versiones previas de la guía han sido reemplazados por otros nombres más contextuales y significativos que se resumen en la siguiente tabla. Además se han añadido dos nuevas estrategias para abordar nuevas y mejores prácticas emergentes: **feature toggling** y **continuous delivery**.

Nuevo nombre {Viejo nombre entre llaves}	Visual	Página
Main Only {No Branch Plan}		14
Release Isolation {Release Branching – Basic (Single Branch)}		15
Development Isolation {part of Release Branching – Basic (two branch)}		15
Development and Release Isolation {Release Branching – Basic (two branch)}		16
Servicing and Release Isolation {Release Branching – Standard}		16

Estrategias de Branching – ¿Qué hay de Nuevo?

Nuevo nombre {Viejo nombre entre llaves}	Visual	Página
Servicing, Hotfix and Release Isolation {Release Branching – Advanced}		18
Feature Isolation {Feature Branching}		19

Tabla 1 – Estrategias de Branching de un vistazo

Tutoriales

Para complementar los hands-on lab y las estrategias de branching, se han incluido nuevos tutoriales para guiarnos en escenarios comunes como:

- Partir de no tener estrategia a adoptar una o más según sea necesario.
- Adaptar nuestra estrategia para casos especiales en nuestro proceso estándar, por ejemplo, si necesitamos hacer un branch a partir de un changeset concreto.

Los tutoriales intentan ser listas de control prácticas que nos guiarán por el proceso que más nos convenga, describiendo también sus ventajas e inconvenientes.

Guías adicionales

Hemos dividido esta guía en cuatro temas. Esto nos permitirá elegir el “mundo” en el que estemos interesados y minimizar el ruido y las distracciones.





Guía	Contexto
	<p>Estrategias de Branching → esta guía.</p> <p>Guía práctica sobre algunas estrategias de branching y su uso.</p> <p>Secciones principales:</p> <ul style="list-style-type: none"> • Conceptos de Branching • Estrategias de Branching • Tutoriales
	<p>Joyas de TFVC</p> <p>Guía práctica sobre cómo usar las características de Team Foundation Version Control (TFVC).</p> <p>Secciones principales:</p> <ul style="list-style-type: none"> • Workspaces • Merging • Nuevas características, por ejemplo: Code Lens • Tutoriales
	<p>Gestión de dependencias con NuGet</p> <p>Guía práctica para la gestión de dependencias, usando NuGet con Visual Studio.</p> <p>Secciones principales:</p> <ul style="list-style-type: none"> • Gestionando recursos compartidos • Gestión de dependencias • Tutoriales
	<p>Git para usuarios de TFVC</p> <p>Guía práctica sobre Git desde la perspectiva de TFS.</p> <p>Secciones principales:</p> <ul style="list-style-type: none"> • Guía • Tutoriales • Terminología/Mapa de conceptos

Tabla 2 – Guías de Control de Versiones

Conceptos

1. **Empezamos** con **ningún branch** o con la estrategia Main Only.
2. **Haz un Branch** solo cuando sea necesario y después de analizar y entender el valor que añade y el coste de mantenimiento.
3. Las estrategias de esta guía no son las únicas estrategias válidas. Otras estrategias pueden funcionar mejor en tu equipo y en tu contexto.
4. Si **no estás seguro** de necesitar una estrategia de branching, vuelve al paso 1.



Vocabulario

"Olvidamos palabras así como olvidamos nombres. Nuestro vocabulario necesita fertilizante o morirá" - Evelyn Waugh

Las terminologías pueden variar según la organización por lo que es necesario indicar qué terminología se usa en esta guía.

Término	Descripción
Development Branch	Cambios para la siguiente versión
Forward Integrate (FI)	Merges desde el branch padre a los branch hijos
Hotfix	Un cambio para corregir un bug bloqueante de un cliente o una interrupción de servicio
Main Branch	Este branch es la unión del branch de desarrollo y los branch de release. Este branch debería ser un snapshot estable del producto que se puede compartir con QA o equipos externos
Release Branch	Un branch para aislar el código de preparación para una release. Se hacen correcciones que impidan el lanzamiento. Tras el lanzamiento este branch debe configurarse como sólo lectura dependiendo de nuestro proceso
Release Vehicle	Cómo llega nuestro producto a nuestros clientes (por ejemplo, nueva release, hotfixes y/o service packs).
Reverse Integrate (RI)	Merges desde los branch hijos a los padres
Service Pack (SP)	Una colección de hotfixes y características basadas en una release previa

Tabla 3 – TFVC vocabulario de branching

Conceptos de Branching

"La distancia de tu branch a la principal es igual a tu nivel de locura" – anónimo



Figura 1 – (Duro) El branching ofusca la simplicidad y recomendamos la estrategia "main only" más simple, la estrategia bambú ☺

El branching permite el desarrollo en paralelo ofreciendo a cada actividad de desarrollo un snapshot de los archivos de código fuente necesarios, de las herramientas, de las dependencias externas y de los procesos

automáticos. Tener más branches aumenta la complejidad y el coste de los Merges. Sin embargo, hay varios escenarios en los que tendremos que considerar mantener varios branches para mantener cierta velocidad de desarrollo. Ya sea para aislar o estabilizar temporalmente un cambio importante, desarrollar una funcionalidad, o comenzar el desarrollo de una versión futura, deberíamos considerar lo siguiente:

- Haz un branch desde un padre que tenga los últimos cambios (normalmente MAIN u otro branch de DEV).
- Haz un branch de todo lo que necesites para desarrollar en paralelo, normalmente esto significa hacer un branch completo del padre.
- Haz un merge desde el branch padre (FI) frecuentemente. Siempre FI antes que RI. Haz FI tantas veces como tenga sentido. Semanalmente es un buen punto de partida.
- Asegúrate de que compilas y ejecutas los suficientes Tests de Verificación de Build (BVT, en inglés Build Verification Tests) para medir la estabilidad del branch.
- Para estabilizar cambios, quédate con tus cambios, haz un forward integrate (FI), compila y pasa todos los BVT) antes de hacer un merge (RI) con los cambios al padre.
- Un merge (RI) a MAIN es como una “feature release” en equipos de desarrollo. Una vez que el branch hace merge (RI), otros equipos usarán esa versión hasta el próximo merge (RI).
- Cuando se hace un branch de MAIN para release (versión actual), las correcciones que hacen que no se pueda liberar la versión se deben hacer en la branch RELEASE. El branch de nuevas características es sólo para el trabajo de la próxima versión. En este ejemplo cuando se hace el branch de MAIN para la próxima release, todos los branches de DEV pueden comenzar a trabajar para la próxima versión.

Evitar el branching

NOTA

Mientras menos branches tengamos mejor. En lugar de seleccionar y pre-crear una estrategia de branching, empecemos con ninguna y crearemos branches a medida que los vayamos necesitando. Leed **De nada a la complejidad o no**, página [34](#), para un tutorial práctico.

El listón para crear un branch debería estar muy alto para evitar los costes asociados. Tengamos en cuenta lo siguiente antes de crear un branch para un desarrollo temporal:

- **¿Puedes usar un shelve?** Un shelve permite a los desarrolladores guardar y compartir cambios en TFS, pero no se incluyen en el branch.
- **¿Hay otro work ítem en el que deberíamos de estar trabajando?**
- **¿Eres capaz de trabajar en el branch “próxima versión” que hemos mencionado más arriba?** Esto puede ser factible si los interesados aprueban el cambio para la próxima versión.
- **¿De verdad “necesitas” el branch ahora?** ¿Podemos crear el branch más tarde, cuando la necesitemos? En ese momento, quizás podemos crear el branch a partir de un changeset.

Asegúrate siempre de que puedes describir el valor que un branch aporta a tu organización; en otro caso quizás no necesitas el branch.

Branching con Labels

Las organizaciones que necesitan asegurar la estabilidad deberían considerar crear branches para capturar versiones específicas del código que se lanzó en una release. El uso de este tipo de estrategia de branching hace relativamente fácil para un desarrollador obtener una versión concreta del código y realizar un mantenimiento en esa versión. Crear un branch de una versión concreta de fuentes permite la creación de branches de mantenimiento a medida que hacen falta en lugar de tener efectos colaterales en cada release. Cuando hacemos branch por versión con TFVC, hay que conocer el valor que los labels ofrecen.

Los labels en TFS son una herramienta muy poderosa que permiten al equipo de desarrollo identificar rápidamente los archivos de una versión concreta. El uso de labels como parte de la estrategia de branching permite a los desarrolladores aislar el código de cambios que han podido ocurrir en el sistema de control de código. Por ejemplo, si una versión concreta necesita mantenimiento, podemos crear un branch en cualquier momento del futuro basándonos en un label. Esto llevará a una estructura de directorios en el control de versiones ya que sólo creamos directorios nuevos cuando hagan falta.

Cuando usemos labels, tengamos en mente las siguientes advertencias:

- Los labels son editables y se pueden borrar (hacen falta permisos). Estos cambios no se pueden auditar.
- Puede haber contención para un label si más de una persona quiere usar y modificar el label o los archivos que contiene.
- No podemos confiar en los label ya que se pueden borrar automáticamente cuando se aplica la política de retención.

De modo que, usemos labels en casos en los que necesitemos un snapshot de los fuentes y si es posible garanticemos, a través de permisos, que no cambiaremos el contenido de un label.

Creando Branches

Puede ser de ayuda nombrar las builds por branches para identificar fácilmente las builds y los output asociados, por ejemplo:

- **DEV**_NombreProducto_*infoversion*
- **REL**_NombreProducto_*infoversion*
- **FEA**_NombreProducto_NombreCaracteristica_*infoversion*

Automatizamos la actualización de la información de versión en el procedimiento de build. Por ejemplo, establecer la información del atributo del assembly de un assembly enlazado directamente al artefacto de una build específica.

Esta trazabilidad puede ser de mucha ayuda si tenemos varios branches y build agents. Para hacer más fácil la identificación de los binarios, añadamos el nombre completo de la build en el archivo de información de versión y/o en la información del assembly. Esto nos ayudará a identificar los binarios y sabremos a que branch pertenecen.

Permisos

TFS ofrece dos permisos relacionados con el branching y el merging. Uno permite a los equipos designar a ciertos individuos responsables de crear nuevos branches. El otro puede ser responsable de hacer Merges de código entre branches, mientras que los demás desarrolladores tendrán restringido trabajar en ciertos branches.

- Manage Branch
- Merge

Permiso de Manage Branch

El permiso Manage Branch permite las siguientes acciones:

- Convertir directorios en branches
- Convertir branches en directorios
- Actualizar los metadatos de un branch, como owner, descripción, etc.
- Crear branches hijos de un branch padre
- Cambiar las relaciones entre branches (por ejemplo: cambiar los padres de branches)

El permiso **Manage Branch** solo aplica a branches (ni a directorios o archivos "brancheados"). Denegar el permiso Manage Branch no evita que los usuarios hagan branches de directorios normales a los que les has denegado este permiso. TFS se salta el permiso Manage Branch para un directorio dado. Como un directorio normal, podemos organizar los branches en directorios. Esta puede ser una manera útil de organizar y aplicar permisos a grupos de branches.

Por ejemplo, un equipo puede denegar el permiso Manage Branch a los Contributors para todos los branches organizados bajo el path: \$/<TeamProject>/Main y \$/<TeamProject>/Release, pero permitir a los Contributors el permiso Manage Branch para los branches que estén bajo el path \$/<TeamProject>/Development. Con estos permisos, los miembros del grupo de TFS Contributors pueden crear branches hijos para desarrollo (a partir de branches existentes), pero no pueden crear branches de un branch de release existente o del branch Main.

Permiso Merge

NOTA

Esta guía de “Estrategias de Branching” no cubre el merging. Leer [MSDN](#)⁸ y la guía “Control de Versiones con TFS Parte 2 - Gemas de TFVC” para más información sobre el merging.

El permiso de **Merge** es necesario para realizar operaciones de merge en branches, directorios, y archivos en un path específico. Este permiso es necesario para el directorio de destino de una operación de merge. No hay ningún permiso que nos evite hacer un merging a un branch particular o de un directorio a otro directorio. El permiso de Merge no está limitado a los branches; podemos aplicar este permiso a directorios y branches en un path dado.

Tipos de Branch

“El árbol era manifiestamente viejo, debido al tamaño de su tallo. Medía aproximadamente seis pies de altura, las ramas nacían del tallo de manera regular y simétrica, y tenía toda la apariencia de un árbol en miniatura” - Robert Fortune

Podemos organizar los branches en tres categorías: MAIN, DEVELOPMENT y RELEASE. Dependiendo del tipo de branch, debemos tener presente las siguientes consideraciones:

- Compilarlas regularmente (continuamente) nos ofrece una cadencia regular y nos permite identificar rápidamente problemas de calidad.
- Los movimientos de cambios que se producen entre los branch padres e hijos deben ser entendidos y tenidos en cuenta por parte de todos los miembros del equipo.

NOTA

Si buscáis un pipeline de integración continua de builds y despliegue, os recomendamos leáis las guías [ALM Rangers DevOps](#)⁹ y [Building a Release Pipeline with Team Foundation Server](#)¹⁰.

La siguiente tabla muestra una lista de otras consideraciones a tener en cuenta de los tipos de branch que hemos mencionado antes.

Main

- Los branch de DEVELOPMENT y RELEASE, están a un merge de distancia de MAIN.
- El branch de la “verdad”, que debe ser “buildable”, cumple con unos mínimos de calidad y es una fuente de confianza para los equipos de QA.
- Salvo en el caso de la estrategia “Main Only”, hay que evitar hacer cambios en MAIN.
- A medida que aumenta el número de branches de DEVELOPMENT, el coste y la necesidad de merge (FI) tras una MAIN exitosa aumentan.

Dev (Desarrollo)

- Todos los branches son áreas auto contenidas (aisladas) que permiten a cada actividad de desarrollo avanzar, sin depender unas de otras.
- Debemos basar un branch DEV en un padre con un estado conocido y bueno, normalmente MAIN.
- Haz merges (FI) frecuentemente para reducir la complejidad del tipo “big bang”.
- Podemos hacer merges (FI) opcionalmente cada vez que el branch padre sea construido y pase los BVTs, pero esto suele conllevar un gran overhead.

⁸ <http://msdn.microsoft.com>

⁹ <http://aka.ms/treasure54>

¹⁰ <http://aka.ms/treasure53>

Rel (Release)

- Los branches de Release deben **soportar** a nuestro **release vehicle**.
- Los release vehicles más comunes son major release, hotfix y service pack.
- Como con todos los branches en general, ¡menos es mejor!
- La relación padre/hijo entre MAIN→SP→y HOTFIX permiten el merge de cambios en release futuras (por ejemplo: Los hotfixes se mergean al branch SP en su camino a MAIN) reduciendo el riesgo de bugs de regresión en próximas releases.

Estrategias de Branching

"Guarda tu creatividad para tu producto... no para el plan de branches" – Anónimo

Los elementos de estrategias de branching que veremos en esta sección son normalmente aditivos, partiendo de la estrategia **Main Only** y adoptando otras estrategias para evolucionar y mezclarlo en un entorno más complejo siempre y cuando **sea necesario**.

META

La meta de esta sección es introducir las estrategias básicas que hemos encontrado y usado de manera efectiva en el campo de batalla. Por favor, pon los escenarios que tengas y no estén cubiertos por la guía en nuestro foro de [CodePlex](#)¹¹, para que podamos sopesar las alternativas para futuras actualizaciones de la guía.

Main Only

La estrategia **Main Only** puede estar basada en directorios o con el directorio principal promovido a un branch para soportar las características de control de versiones que ofrecen una visibilidad extra.

En la siguiente ilustración, el icono de branch de la izquierda indica que hemos promovido el directorio principal a un branch, esto es un paso opcional. Recomendamos esta opción sólo si queremos evolucionar a otras estrategias de branching en el futuro.

Leed [Branch Folders and Files](#)¹² y la FAQ para más información sobre directorios de branch. Fijaos que los ejemplos (V1.0, V1.1, y V2.0) muestran cómo hemos aplicado labels.



Figura 2 – Estrategia de branching Main Only

Escenarios de uso

Esta estrategia es el **recomendado** punto de partida para cualquier equipo que no ha necesitado el aislamiento de código usando branches y cuando no hay problemas con las etiquetas mutables para marcar los hitos, en el único branch.

NOTA

Los proyectos de los ALM Ranger comienzan su vida con esta estrategia, creados con [TFS Branch Tool](#)¹³, y evolucionan a otras estrategias de branching cuando es necesario, reduciendo la complejidad de los merges y sus costes.

Consideraciones

- Sin branching, necesitamos labels para marcar el desarrollo y los hitos de las releases. La mutabilidad y la falta de históricos de labels añaden un riesgo al control de cambios.
- Lee [Team Foundation Server Permissions](#)¹⁴ y considera proteger la administración de labels.

¹¹ <http://aka.ms/treasure18>

¹² <http://msdn.microsoft.com/en-us/library/ms181425.aspx>

¹³ <http://aka.ms/treasure35>

¹⁴ [http://msdn.microsoft.com/en-us/library/ms252587\(v=vs.100\).aspx](http://msdn.microsoft.com/en-us/library/ms252587(v=vs.100).aspx)

Development Isolation

La estrategia de Development Isolation añade uno o más branches de desarrollo desde Main, lo que permite el desarrollo concurrente para la próxima release, experimentos, o resolución de bugs en branches **aislados**.

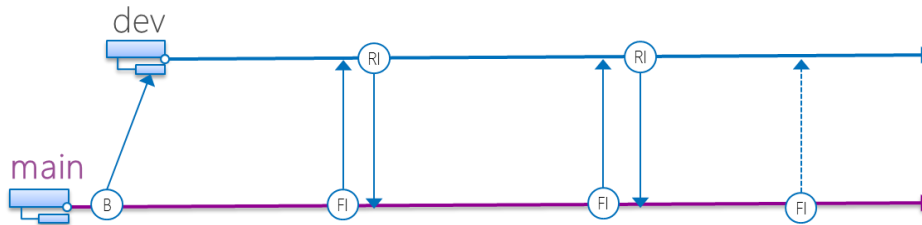


Figura 3 – Estrategia de branching Development isolation

Escenarios de uso

- Cuando necesites desarrollo **aislado** y **concurrente**, protegiendo el branch estable principal.
- Cuando necesites **sólo una major release**, soportada y entregada al cliente usando el branch principal.
- Cuando necesites un modelo de servicios para que tus clientes se actualicen a la **próxima versión**, por ejemplo: v1 → v2.

Consideraciones

- Cada branch de desarrollo debería ser un branch hijo completo del branch principal. Evita crear branches parciales.
- Podemos aislar el trabajo de desarrollo en branches por funcionalidad, organización o colaboración temporal.
- Los branches de desarrollo deben construir y ejecutar los Build Verification Tests (BVTs) al igual que el branch principal.
- Haz merges (FI) frecuentemente desde el branch principal al de desarrollo si se producen cambios directamente en Main.
- Haz merges (FI) y luego Reverse Integrate (RI) desde los branch de desarrollo al principal basándote en el objetivo del equipo (por ejemplo: aseguramiento de la calidad interna, fin de sprints, etc.).

Release Isolation

La estrategia **Release Isolation** incluye uno o más branches de release desde MAIN, permitiendo así la gestión concurrente de releases.

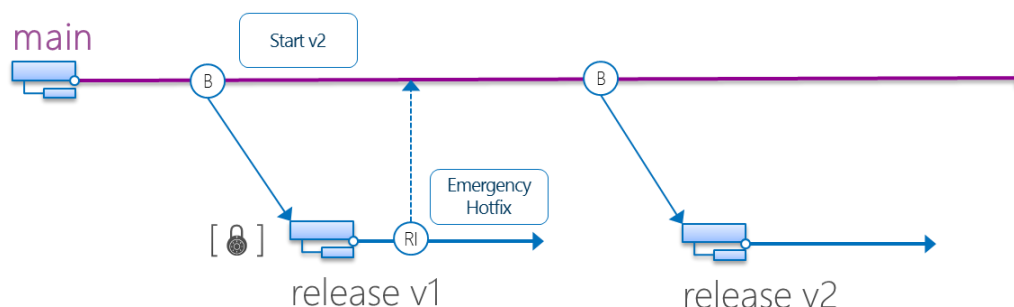


Figura 4 – Estrategia de branching Release isolation

NOTA

Habrás casos que no son blancos o negros, es por eso por lo que continuamos mostrando hotfixes de emergencia en los branches de release que, en teoría, deberían ser inmutables.

Leed **Casos Reales** en la página [31](#), para ver casos reales que han usado esta guía de branching, pero desviándose (evolucionando) desde las reglas “blanco y negro” sin afectar a las entregas ni a la calidad. **No** veas esta guía como algo inmutable ni grabado en piedra. Al contrario, evoluciona continuamente para que encaje en tu entorno y se ajuste a tus necesidades.

Escenarios de uso

- Cuando necesitemos **asilamiento** y release **concurrentes**, manteniendo el branch principal estable.
- Cuando necesitamos **varias releases en paralelo** para soportar y entregar a nuestros clientes el branch de release adecuado.
- Cuando necesitamos un modelo de servicios para que nuestros clientes se actualicen a la próxima release por ejemplo: v1 → v2.
- Cuando **necesitemos** snapshots de nuestro código fuente en tiempo de releases.

Consideraciones

- Cada branch de release debe ser un hijo completo del branch principal.
- Los lanzamientos de producto se hacen desde branches de release.
- Bloquea (sólo lectura) el branch de release con permisos de acceso para evitar modificaciones en una release.
- Los cambios del branch de release se deben mergear (RI) a la principal. No deberíamos mergear (FI) desde el branch principal a los branches de release.
- Crea nuevas branches de release para las siguientes releases si queremos mantener ese nivel e aislamiento.
- Cualquier corrección entregada desde un branch de release puede incluir parches desde ese branch. Los parches pueden ser acumulativos o no-acumulativos independientemente del plan de branching.

Development y Release Isolation

La estrategia de **Development y Release Isolation** combina las estrategias de Development Isolation y Release Isolation, une tanto los escenarios de uso como las consideraciones.

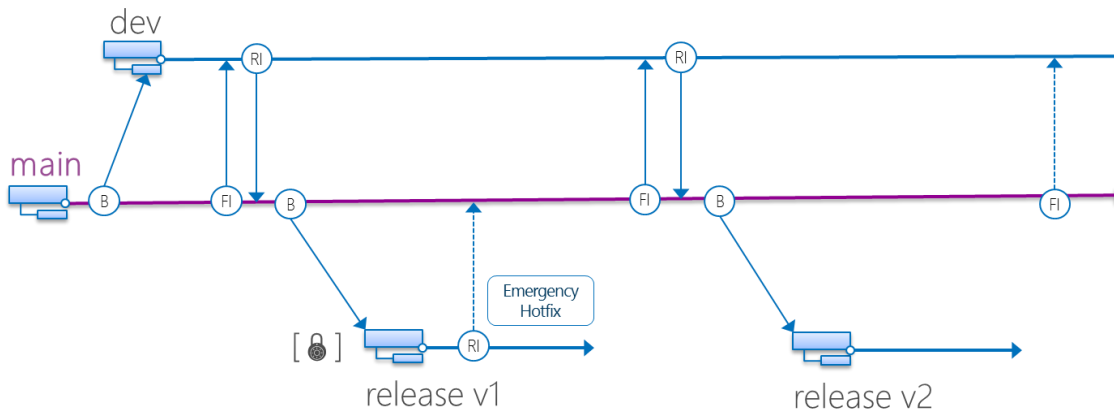


Figura 5 – Estrategia de branching Development y release isolation

Servicing y Release Isolation

AVISO

Ésta es una estrategia avanzada de aislamiento, añade cierta complejidad y costes de mantenimiento por temas de tracking, merging y gestión de servicios. Recomendamos que evolucionéis a esta estrategia cuando sea necesario y no empecéis un proyecto con esta estrategia o derivados de ella.

La estrategia **Servicing y Release Isolation** añade branches de servicio, que permiten la gestión concurrente de bugs y service packs. Esta estrategia es una evolución de Release isolation, y/o mejor dicho de la estrategia de Development y Release Isolation.

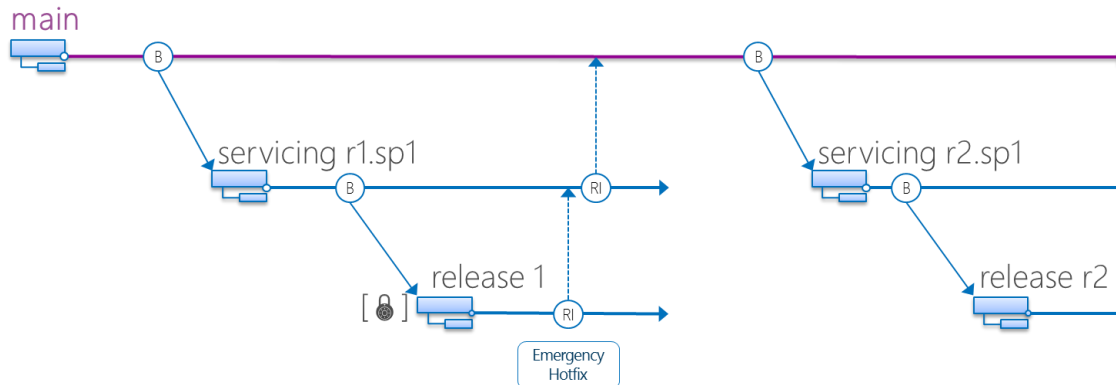


Figura 6 – Estrategia de branching Servicing y release isolation

NOTA

Las versiones anteriores de la guía de Branching and Merging llaman a esta estrategia **Standard Branch Plan**.

Escenarios de uso

- Cuando necesitemos releases **aisladas** y **concurrentes**, manteniendo en un estado estable el branch principal.
- Cuando tengamos **varias mayor releases**, permitiendo dar soporte y entregar a clientes el branch adecuado.
- Cuando necesitemos un modelo de servicios para que nuestros clientes se actualicen a la **siguiente mayor release**, por ejemplo: v1→v2
- Cuando necesitemos un modelo de servicios para que nuestros clientes se actualicen a **service packs adicionales**, por ejemplo: v1 SP1 → v1 SP2
- Cuando necesitemos **snapshots** del código en tiempo de release y servicio.

Consideraciones

- Los branches de servicio y release nacen del branch principal al mismo tiempo que se crea una relación main → servicing → release.
- Branch de servicio
 - Cada branch de servicio debe ser un hijo completo del branch principal.
 - Nuestra release de service pack nace del branch de servicio.
 - Los cambios del branch de servicio se mergean (RI) a la principal. No deberíamos mergear (FI) desde el branch principal a los branches de release.
 - Bloquea (sólo lectura) el branch de servicio usando los permisos de acceso para evitar modificaciones de una release.
 - Cualquier corrección del branch de servicio puede contener todas las correcciones anteriores de ese branch. Los parches pueden ser acumulativos o no acumulativos, independientemente del plan de branching.
- Branch de Release
 - Cada branch de release debe ser hijo de un branch de servicio.
 - Las releases de nuestro mayor product deben ser hijos del branch de release.
 - Los cambios del branch de release se mergean (RI) al branch de servicio. No debemos mergear (FI) del branch principal en los branches de release.

- Bloquea (sólo lectura) el branch de release usando los permisos de acceso para evitar modificaciones a una release.
- Cualquier corrección realizada en el branch release puede incluir todas las correcciones anteriores de ese branch. Los parches pueden ser acumulativos o no acumulativos independientemente del plan de branching.
- Crea un nuevo branch de servicio y de release para las siguientes mayor releases si necesitas ese nivel de aislamiento.

Servicing, Hotfix, y Release Isolation

Podemos evolucionar la estrategia de servicing y release isolation añadiendo branches adicionales para hotfix, consiguiendo así el soporte necesario para tener varios release vehicles y escenarios de servicios. Como estamos remarcando el hecho de hacer branches bajo demanda y mantenerlo todo muy simple, lo decimos, pero no entraremos en muchos detalles de esta estrategia.

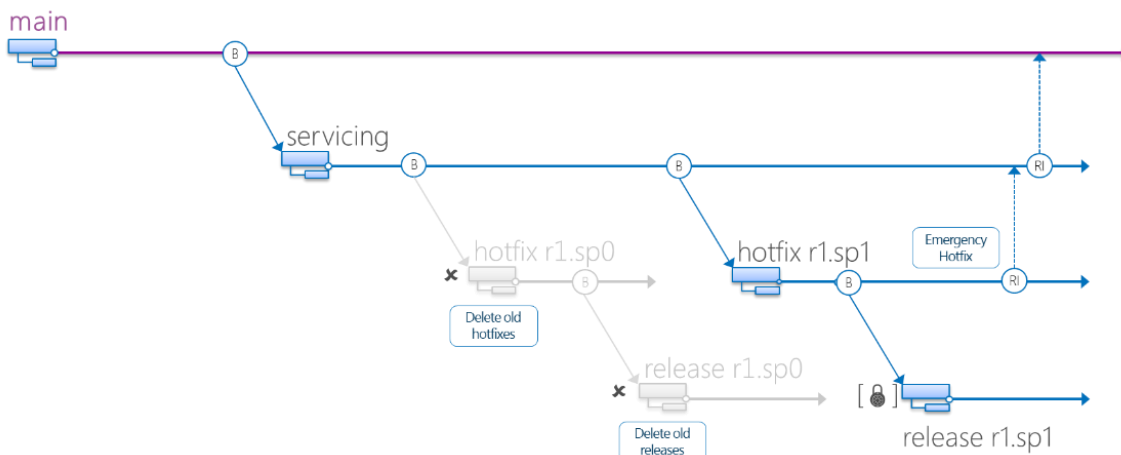


Figura 7 – Estrategia de branching Servicing, hotfix y release

NOTA

En versiones anteriores de la guía Branching and Merging se llama a esta estrategia **Advanced Branch Plan**.

Code Promotion

El plan **code promotion** promueve versiones del código en base a nuevos niveles a medida que se vuelve más estable. Otros sistemas de control de versiones implementan técnicas similares usando propiedades de archivo que muestran en qué nivel de promoción está. A medida que el código del branch principal se vuelve más estable, lo promovemos al branch de test (también conocido como branch de QA), donde se ejecutan todos los test de regresión y donde pueden corregirse bugs de manera adicional. Una vez que el código está listo para una release en el branch de testing, se mergea en el branch de producción, donde se realiza la certificación de calidad final y se testa. El desarrollo concurrente puede realizarse en el branch principal, que promueve el concepto de “código no congelado”.

NOTA

La estrategia Code Promotion se suele entender como una reliquia de la era del desarrollo en cascada. Debemos asociarlo con ciclos de test largos y con departamentos de desarrollo y testing separados. Normalmente, no recomendamos esta estrategia, ya que es muy antigua.

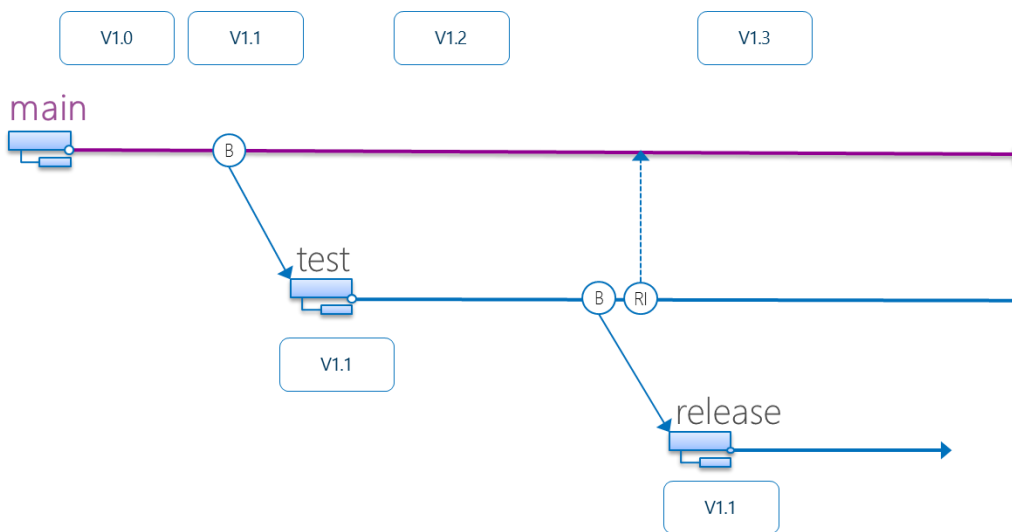


Figure 8 – Estrategia de branching Code promotion

Escenarios de uso

- Cuando sólo tenemos **una major release**, soportada y entregada a nuestros clientes desde el branch principal.
- Cuando tenemos **largos ciclos de testing**.
- Cuando necesitamos **evitar código congelado**, y queremos tener aislada nuestra major release.

Consideraciones

- El branch de test es un hijo completo del branch principal.
- El branch de Release es un hijo completo del branch de test.
- Las versiones de nuestro producto nacen del branch de release
- Los cambios del branch de test se mergean (RI) al principal. Este merge es de un sólo sentido.
- Restringe los permisos del branch de release para aislar el resto.

Feature Isolation

La estrategia **Feature Isolation** añade uno o más branches de funcionalidad desde el principal, permitiendo el desarrollo concurrente de funcionalidades definidas claramente para la próxima release.

La estrategia Feature Isolation es una evolución especial de la estrategia Development Isolation, esencialmente con dos o más branches de desarrollo (funcionalidad).

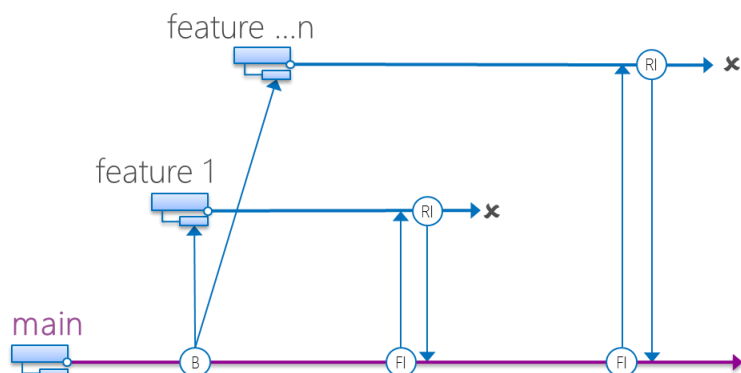


Figura 9 – Estrategia de branching Feature isolation

Considera el uso de directorios para organizar los branches de cada funcionalidad, por ejemplo: `$/BranchingScenarios/funcionalidades/funcionalidadPOWER`

Con la estrategia de Feature Isolation, podemos aislar cada funcionalidad en su propio branch dándonos una gran flexibilidad en términos de cuando movemos una release a producción. Esta estrategia también ayuda a evitar escenarios en los que un branch necesita ser publicado con las funcionalidades que estén terminadas en un branch mientras que las funcionalidades que no estén terminadas se quedan en sus branches. Mergea funcionalidades al branch principal a medida que se vuelven estables para una release, borra y crea nuevas branch de funcionalidad a medida que son necesarias.

Escenarios de uso

- Cuando necesitemos **asilamiento** y desarrollo **concurrente** de **funcionalidades bien** definidas, protegiendo el branch principal.
- Cuando tengamos **equipos separados** desarrollando varias funcionalidades para un mismo proyecto.
- Cuando estemos creando un **producto grande** con **muchas características** que necesitan aislarse.
- Cuando desarrollamos funcionalidades en paralelo que no deben estar en el mismo ciclo de release.
- Cuando necesitemos la habilidad de quitar funcionalidades de una release. Eliminar una funcionalidad puede ser costoso y cuando sea necesario hay que hacerlo con cuidado.

Consideraciones

- Cada branch de funcionalidad debe ser un hijo completo del branch principal.
- Mantén corta la vida del desarrollo de una funcionalidad, y mergea (RI) con la principal frecuentemente.
- Los branches de funcionalidad deben compilar y ejecutar los Build Verification tests (BVTs) del mismo modo que se hace en el branch principal.
- Mergea (FI) frecuentemente desde el branch principal a los branch de funcionalidad cuando haya cambios directamente en el branch principal.
- Mergea (RI) desde el branch de funcionalidad al branch principal en base a criterios de equipo objetivos, por ejemplo: Definition of Done (Dod).

Estrategias alternativas

Adapta tu proceso de branching para cosas inesperadas.

Incluso con el mejor plan de proyecto y las mejores intenciones, los stakeholders pueden pedir a tus equipos de desarrollo que cambien el rumbo. Puede ser un cambio temporal en las prioridades, no llegar a una fecha de entrega o cambiar la dirección, debes ser capaz de reaccionar ante eso cuando ocurra y, lo más importante, saber convivir con las consecuencias que pueda tener a largo plazo.

Recuerda, todo lo que necesitamos para garantizar que podemos repetir una compilación es un changeset porque es inmutable. Los labels y branches son sólo conveniencias. Úsalos en tu favor para gestionar esas cosas inesperadas en tus procesos y para simplificar el soporte y la entrega de tu software.

Tanto si tienes sólo un branch principal o una estrategia de branching avanzada con 'main, servicing, hotfix, feature', el proceso es el mismo:

1. Analiza el nuevo requisito y que no te entre el pánico.

Mientras el equipo se está volviendo loco tratando de lidiar con lo inesperado, mantén la mente fijada en tu objetivo y recuerda cuál es tu estrategia. Asegúrate de que la persona que controla tus branches está involucrada en este análisis. La persona que controla los branches de tu sistema de control de código sabrá cuáles son los mejores procedimientos a seguir, cuál es el estado exacto del sistema y de los release vehicles. Sabrá aportar un gran valor a la solución.

2. Haz branches sólo si es necesario.

Una vez que entiendas los requisitos, puede ocurrir que no sea necesario crear otro branch, sin embargo, si necesitas otro branch, es importante entender las opciones disponibles. Visual Studio nos permite crear branches a partir de **Changeset**, **Fecha**, **Label**, **Latest Version** o una **Workspace Version**.

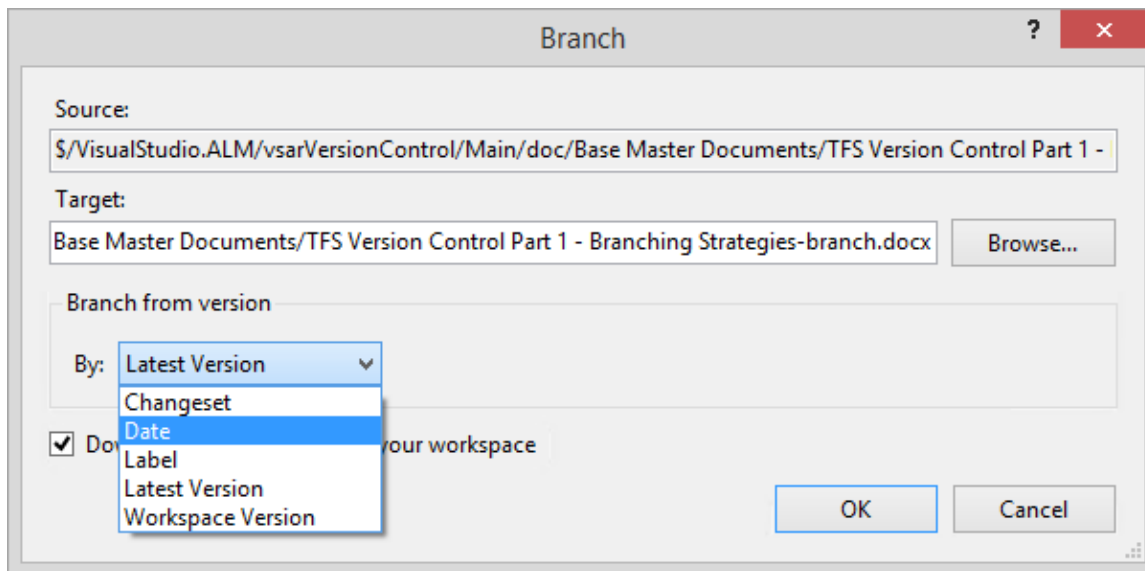


Figura 10 – Branch de una versión a partir de una Fecha

Recomendamos que tengas un entendimiento profundo de los requisitos para crear un branch a partir de un Changeset.

3. Borra la rama

Si necesitas crear un branch, la meta normalmente será incluir el código en la operativa normal de branches para evitar consecuencias a largo plazo. Intenta ser comprensivo pero firme mientras trabajas por llegar a la meta. La temporalidad y la gran cantidad de branches causan confusión y un gran overhead en el equipo.

Lee el tutorial de la página [29](#), en el que tratamos con un cambio de requisitos que requiere branching para soportar un parche 'out-of-band' y el subsiguiente colapso del branch.

Feature Toggling

NOTA

Por favor, leed el artículo [Using Feature Toggles as an Alternative to Development or Feature Branches](#) ¹⁵, de [Bill Heys](#) ¹⁶ para más información sobre feature toggling.

La técnica **Feature Toggling** nos permite activar y desactivar funcionalidades dependiendo de si están disponibles en tiempo de ejecución. Pueden ser funcionalidades visibles en la UI o una parte de código que aporta nueva funcionalidad. Esta técnica tiene varios nombres que incluyen flags, toggles y switches y varias estrategias de implementación pensadas tanto para el tiempo de compilación como para el de ejecución. En esta guía veremos las técnicas que se suelen usar para el tiempo de ejecución, que incluyen el activar o desactivar funcionalidades en tiempo de ejecución. La mayoría de las veces la implementación involucra un archivo de configuración donde se realizan las activaciones y desactivaciones. Podemos combinarlo con el mapeo de elementos en la UI o en clases, pudiendo controlar así la visibilidad de varias funcionalidades. Algunas implementaciones siguen una aproximación guiada por datos que involucran mantener el estado de las activaciones en una base de datos.

Escenarios de uso

- Cuando necesitemos liberar funcionalidades **frecuente** y **continuamente**.
- Cuando necesitemos **habilitar** y **deshabilitar** funcionalidades al vuelo.
- Cuando necesitemos habilitar/deshabilitar funcionalidades de manera selectiva a un conjunto de usuarios específicos.

Consideraciones

- Usa "**Canaries**" para liberar y evaluar características a una audiencia determinada.
- Asegúrate de que no liberas funcionalidades de manera inadvertida a una audiencia que no es o habilitar funcionalidades que no están listas.

Ventajas

Hay muchas ventajas por las que implementar feature toggling. Una muy simple es poder deshacer la activación de una funcionalidad de manera muy simple y al vuelo, sin tener que recompilar y redespargar el sistema.

Con las ventajas de la entrega continua, la liberación de software ocurre muy frecuentemente, con muchas funcionalidades parcialmente completas que pueden durar en completarse varios sprints. Suele ocurrir que esto necesita estrategias de branching y merging complejas para coordinar las diferentes actividades de desarrollo de las funcionalidades. Esto puede llegar a reducir la velocidad y aumentar los conflictos en los merge. Cuanto más distintos sean los branches entre sí, peor y más complejo será el merging de los branches. Con feature toggling, normalmente, podemos usar menos branches ya que podemos tener un código base que contenga todas las funcionalidades completas, y las que aún estén en desarrollo. Cualquiera puede integrar en Main continuamente y normalmente no preocuparse por estabilizar el código base.

Otra ventaja es la habilidad llamada "A/B", "canary", y otros escenarios en los que queráis testar funcionalidades sólo con algunos grupos de clientes o usuarios. Muchas de las redes sociales más populares utilizan estas técnicas para probar nuevas funcionalidades a un grupo reducido de usuarios. Podemos abrir la funcionalidad

¹⁵ <http://aka.ms/vsarfeaturetoggling>

¹⁶ http://blogs.msdn.com/b/willy-peter_schaub/archive/2010/07/23/introducing-the-visual-studio-alm-rangers-bill-heys.aspx

gradualmente a más y más usuarios a medida que vayamos teniendo éxito en el periodo de publicación. También podemos deshacer la activación de esas funcionalidades rápidamente si las cosas no van bien.

Desventajas

Como siempre, con cualquier elección tecnológica, siempre hay pros y contras. Una desventaja del feature toggling es que debemos asegurarnos de no olvidarnos de envolver la funcionalidad de manera adecuada. Feature toggling también requiere un plan de upfront y algún overhead de implementación de dicha estrategia. La liberación inadvertida es otro riesgo potencial. Puede haber situaciones en las que el código vivo contiene funcionalidades que están aún en desarrollo. Los usuarios más astutos podrían ver la funcionalidad mirando el código fuente. Esto puede dar a la competencia o a un usuario la visibilidad necesaria para ver las nuevas funcionalidades de tu producto. Esto puede hacerte implementar nombres crípticos para algunas funcionalidades si esto es un riesgo para tu producto.

También tenemos que asegurar que hay escenarios para probar la funcionalidad tanto cuando esté activada como desactivada, y por supuesto, tiene algún coste.

Técnicas y prácticas

Podemos aplicar algunas técnicas para asegurarnos de optimizar nuestra solución con feature toggling. Uno de los principios fundamentales del feature toggling, así como una buena práctica general en el desarrollo de software, es que los desarrolladores escriben el código de la manera más modular posible. Por ejemplo, podemos tener un método o función que implementa varias funcionalidades simultáneamente a través de sentencias if/else anidadas. Esto no se lleva muy bien con el aislamiento de funcionalidades. Asegúrate de compartimentar las funcionalidades todo lo que puedas.

Adopta alguna convención de nombres y de consistencia para asegurarte de que las técnicas de feature toggling son consistentes en todo tu código fuente. Además, si la seguridad es clave, asegúrate de nombrar a tus funcionalidades de manera adecuada para la ofuscación. Es muy importante tener en mente los escenarios en los que los usuarios pueden ver las funcionalidades en el código fuente. Los usuarios más listos pueden averiguar el significado de las funcionalidades ocultas siguiendo convenciones de nombrado pobres. Si es importante para ti, debes evaluar los riesgos y establecer una estrategia.

Seguramente querrás asegurarte de que haces pruebas tanto cuando las funcionalidades están activas como desactivas. La idea esencial es entender cómo se comporta nuestra aplicación en cada uno de los estados de la funcionalidad. Obviamente dependiendo del escenario exacto, puede que no haya diferencias en algunos casos.

En algunos casos queremos marcar la funcionalidad como “deprecada” en nuestro código, mientras que en otros, podremos querer tener la habilidad de activarla y desactivarla después de que esté en producción por un tiempo. Esta es una decisión de gustos, pero en general limpiar las funcionalidades que se implementaron hace tiempo y ya no se usan, ayuda a mantener tu código limpio.

Herramientas

Hay varias herramientas open source disponibles para implementar feature toggling. Algunas compañías emplean su propia implementación. Dependiendo del tipo de aplicación en la que estés trabajando, debes encontrar cual es la que mejor te conviene. Por ejemplo, en una aplicación ASP.NET, seguramente quieres envolver los elementos de UI, y hacer que se muestren a partir de un flag en el archivo de configuración.

En Microsoft, el equipo responsable de implementar Visual Studio Online (también llamado Team Foundation Service) usa feature toggling. La implementación que usan en este escenario es una solución personalizada que combina una aproximación guiada por datos y un servicio REST para activar y desactivar funcionalidades. Esto permite al equipo mantener el estado de las funcionalidades en una base de datos. Con esta solución, el equipo ha conseguido tener todas las ventajas que hemos visto en la guía.

Esta guía no recomienda una tecnología o una herramienta antes que otra en este momento. Aquí tenéis un punto donde empezar a mirar: [Feature Toggle libraries for .NET](http://david.gardiner.net.au/2012/07/feature-toggle-libraries-for-net.html) ¹⁷

Integración continua

La **integración continua** se basa en la automatización de la construcción, del testing y del despliegue. El cambio es continuo, frecuente y algunas operaciones de merge son más complejas que otras y requirieren una intervención manual. Es recomendable evitar el branching y confiar en otras estrategias, como el **feature toggling** (página 22), cuando estemos considerando la integración continua.

Escenarios de uso

- Cuando sólo tengamos **una major release**, que entregamos y soportamos a nuestros clientes desde el branch principal.
- Cuando tengamos **ciclos cortos** de desarrollo de funcionalidades.
- Cuando necesitemos publicar releases **frecuente** y **continuamente**.

Consideraciones

- Cuando usamos los labels como “marcas”, debemos recordar que son modificables y que no hay histórico de modificaciones de ellas.
- Leer [Team Foundation Server Permissions](https://docs.microsoft.com/en-us/azure/devops/server/permissions/) ¹⁸ y considera proteger la administración de labels.

Si buscáis un pipeline para integración continua, os recomendamos que leáis las guías [ALM Rangers DevOps](https://docs.microsoft.com/en-us/azure/devops/pipelines/) ¹⁹ y [Building a Release Pipeline with Team Foundation Server](https://docs.microsoft.com/en-us/azure/devops/pipelines/build/build-a-release-pipeline-with-team-foundation-server/) ²⁰.

Cuando usamos feature toggling, podemos adoptar la estrategia de branching **Main only**, habilitando y deshabilitando funcionalidades como ya hemos visto. Así, el branch principal se convierte en una fuente de confianza para nuestro pipeline de automatización.

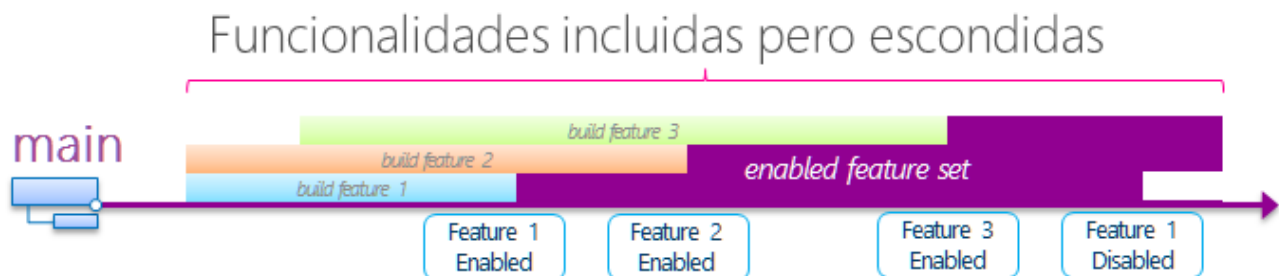


Figura 11 – Integración continua: Feature toggling

¹⁷ <http://david.gardiner.net.au/2012/07/feature-toggle-libraries-for-net.html>

¹⁸ [http://msdn.microsoft.com/en-us/library/ms252587\(v=vs.100\).aspx](http://msdn.microsoft.com/en-us/library/ms252587(v=vs.100).aspx)

¹⁹ <http://aka.ms/treasure54>

²⁰ <http://aka.ms/treasure53>

Release isolation también es factible si usamos los branches como snapshots de archivos y bloqueamos los branches de manera adecuada.

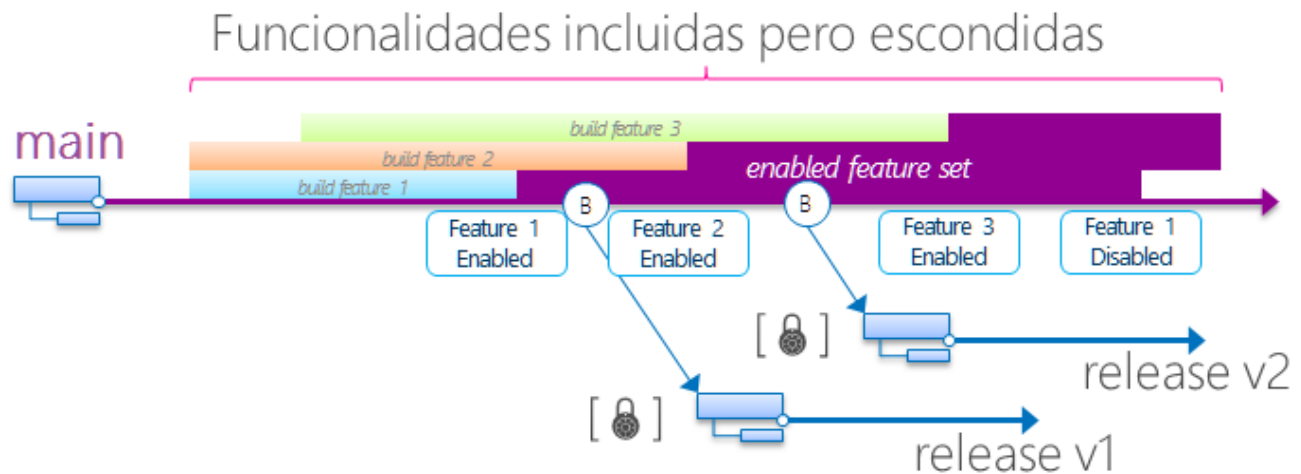


Figura 12 – Integración continua: Feature toggling con release isolation

Si optamos por el uso de **feature isolation**, es importante mantener el alcance de la funcionalidad corto al igual que la duración. Haciendo que los branches tengan una vida corta, las operaciones de merge y el impacto potencial es lo más pequeño posible. Restringe el tiempo de vida de un branch de funcionalidad a unas horas, mejor que días, y borra el branch después de hacer el merge (RI) en el branch principal.

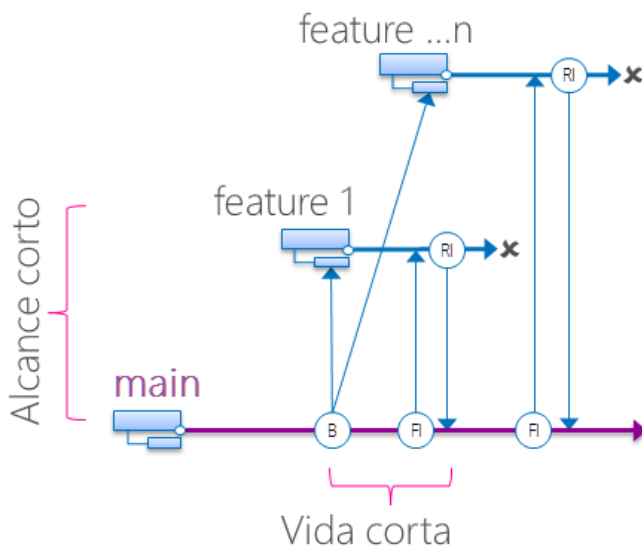



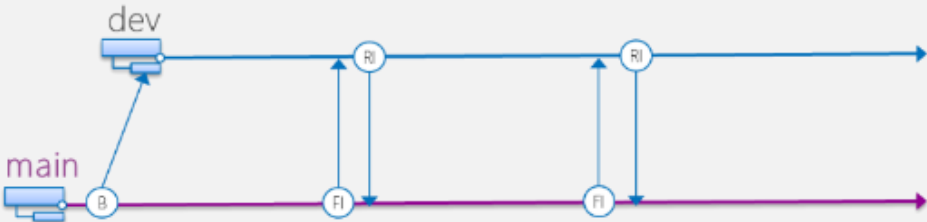
Figure 13 – Integración continua: Feature isolation


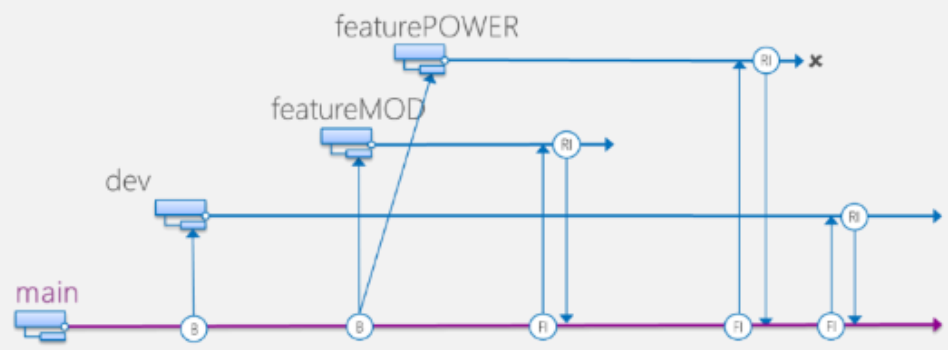
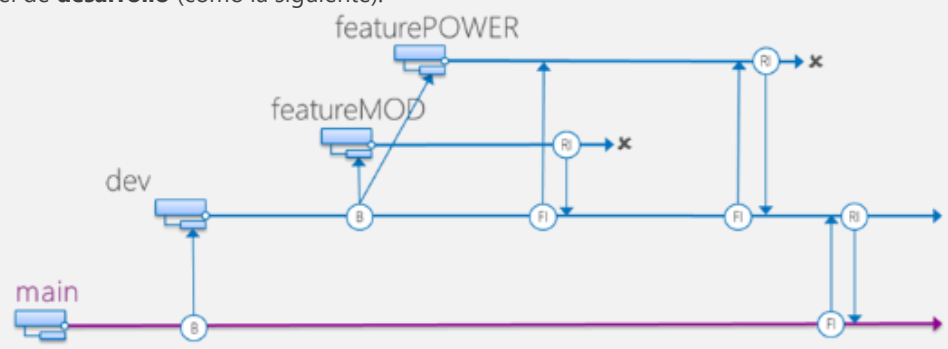


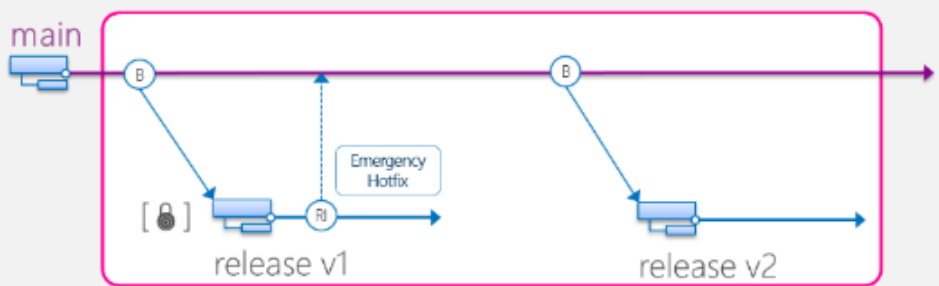
Tutoriales

De la nada a la complejidad o no

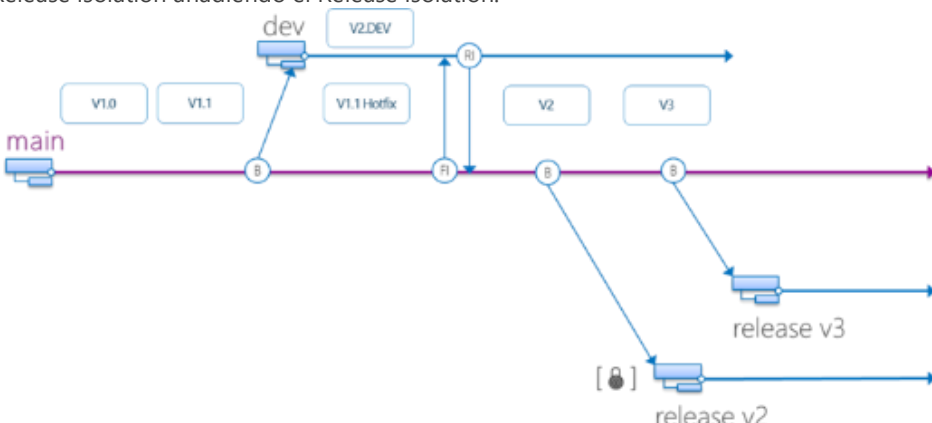
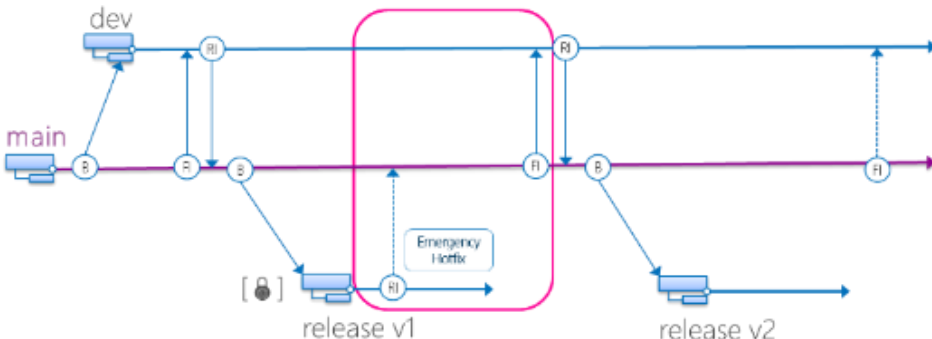
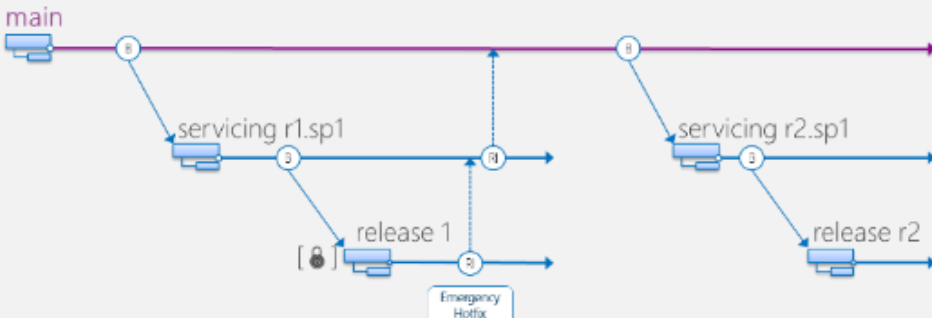
NOTA

El tutorial se basa en el Hands-on Lab (HOL) “De la simplicidad a la complejidad o no”, página [37](#). Leed el HOL para ver todos los detalles y ver cómo hemos llegado a la solución en el tutorial.

Pasos	Instrucciones
1 Empezando con Main Only <input type="checkbox"/> - Hecho	 <p>Recomendamos que empieces por esta estrategia, evita las complejidades de los branch y los merges.</p> <ul style="list-style-type: none"> Tenlo en cuenta cuando no necesitas aislamiento y te sientes cómodo con los labels. Opcionalmente puedes convertir el directorio principal a branch principal para permitir las funcionalidades de los branch, como la visualización y la habilidad de guardar propiedades como propietario y comentarios. <p>COMPLEJIDAD → Baja 😊 Aislamiento: No Releases: 1</p>
2.1 Development Isolation – Empezando <input type="checkbox"/> - Hecho	 <ul style="list-style-type: none"> Considera la estrategia de Development Isolation si necesitas aislar el desarrollo de nuevas funcionalidades, experimentos o corrección de bugs. Haz un branch de la principal a una de dev cuando necesites soportar el desarrollo concurrente. Antes de hacer un merge con las nuevas funcionalidades (Reverse Integration), haz un merge desde Main a dev (Forward Integration) para obtener los cambios del branch principal. <p>COMPLEJIDAD → Moderada 😊 Aislamiento: Desarrollo Releases: 1</p> <p>NOTA Considera crear un segundo espacio de trabajo para el branch de dev para aislarlo del principal.</p>
2.2 Development Isolation – Correcciones <input type="checkbox"/> - Hecho	<ul style="list-style-type: none"> Desarrolla las correcciones en el branch de dev si el flujo de desarrollo tiene que ser concurrente. De manera alternativa (no recomendado) desarrolla correcciones urgentes en el branch principal y haz merges (FI) al branch de dev lo antes posible. <p>COMPLEJIDAD → Moderada 😊 Aislamiento: Desarrollo Releases: 1</p>

Pasos	Instrucciones
3.1 Feature Isolation – Empezando  - Hecho	 <ul style="list-style-type: none"> • Considera el uso de la estrategia Feature Isolation si necesitas desarrollar funcionalidades claras de manera concurrente. • Estudia y decide si sacar los branches de funcionalidad fuera del principal (como la anterior) o en el de desarrollo (como la siguiente).  <ul style="list-style-type: none"> • Si te encuentras corrigiendo cosas urgentes en el branch principal, considera usar Feature Isolation para las correcciones, branches de funcionalidad de la principal. • Antes de hacer merges de las nuevas funcionalidades al branch padre (Main o dev), haz un merge del padre al branch de funcionalidad (FI) para obtener los últimos cambios del branch padre. <p>COMPLEJIDAD → Moderada ☺ Aislamiento: Desarrollo Releases: 1</p>
3.2 Feature Isolation – Limpieza  - Hecho	<ul style="list-style-type: none"> • Opcionalmente borra y/o destruye los branches cuando ya no los necesitas para reducir el ruido. • Ten cuidado al destruir ya que es una acción irreversible y debe usarse con cuidado.
4 Release Isolation  - Hecho	 <ul style="list-style-type: none"> • La estrategia Release Isolation es útil cuando necesitas tener snapshots y varias release principales, por ejemplo: v1 y v2 • De manera opcional (no recomendado) haz correcciones de urgencia en el branch de release y hacer merges (FI) de los cambios al branch principal lo antes posible. <p>COMPLEJIDAD → Moderada ☺ Aislamiento: Release Releases: 2+</p>

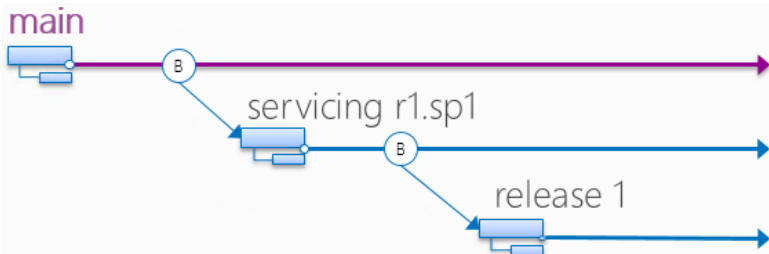
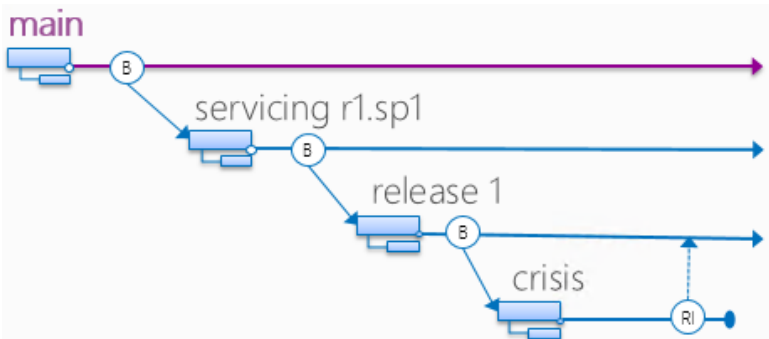
Estrategias de Branching – Tutoriales

Pasos	Instrucciones
<p>5</p> <p>Development & Release Isolation</p> <p>☐ - Hecho</p>	<ul style="list-style-type: none"> Normalmente evolucionaremos la estrategia de Development Isolation a la de Development & Release Isolation añadiendo el Release Isolation.  <ul style="list-style-type: none"> Cuando hacemos merge de los cambios desde un branch de release al branch principal, como una corrección urgente, asegúrate de hacer un merge (FI) de esos cambios a los branches de desarrollo y/o de funcionalidades lo más rápido posible.  <p>COMPLEJIDAD → Moderado ☺ a Alto ☹ Isolation: Development & Release Releases: 2+</p>
<p>6</p> <p>Servicing y Release Isolation</p> <p>☐ - Hecho</p>	<ul style="list-style-type: none"> Si y sólo si necesitas una estrategia de release con administración de servicios concurrentes para corregir bugs y crear service packs, podemos evolucionar a la estrategia de Servicing and Release Isolation. Analiza y define la convención de nombrado de release y servicing. Renombra tu branch de release a servicing, y haz un branch de servicing a release, como definiste en tu convención de nombrado.  <ul style="list-style-type: none"> Seguramente (no recomendado) necesitaremos incluir varios branches, por ejemplo, añadir branch para una administración más fina de los servicios. <p>COMPLEJIDAD → [Muy] Alta ☹ Aislamiento: [Dev &] Release Releases: 2+ y Service Packs</p>
<p>7</p> <p>Otras opciones</p>	<ul style="list-style-type: none"> Considera el uso de la estrategia Main Only por defecto. Antes de usar otras estrategias, considera otras opciones como Feature Toggling.

Pasos	Instrucciones
	<ul style="list-style-type: none"> EVITA modelos de branching complejos que pueden ser increíbles, pero que son muy costosos de mantener

Tabla 4 – Tutoriales: De la nada a la complejidad

Adapta tu proceso de branching para los casos excepcionales.

Pasos	Instrucciones
1 Punto de partida ☐ - Hecho	 <p>Empezaremos con un equipo que está usando la estrategia de <i>servicing and release isolation</i>. Hay tres equipos trabajando en el código. Algunos están trabajando en el branch principal para la siguiente release, otros están trabajando en uno de servicing para crear una pequeña actualización y otros están trabajando en el branch de release corrigiendo errores.</p> <p>NOTA El equipo que está trabajando en las correcciones tiene una cadencia de 1 corrección por semana, sin embargo le entregan esas correcciones al cliente para testarlo; así que hay un pequeño retraso desde que se corrige hasta que se incluye en producción.</p>
2 Urgencia – sin branch ☐ - Hecho	<p>El equipo ha mandado 5 correcciones al cliente y están trabajando en la sexta. La corrección 3 está en producción. El cliente ha lanzado un bug de alta prioridad, que necesita estar en producción lo antes posible. No pueden comprobar las correcciones 4, 5 o 6. Recuerda los pasos,</p> <p>1. Analiza el nuevo requisito y que no entre el pánico</p> <p>Sabemos que el cliente necesita corregir este nuevo bug. Después de analizarlo, vemos que hay una parte del código que no ha cambiado en las correcciones 4, 5 y 6. Esto significa que podemos construir el código y ofrecer una corrección desde el branch de release. Lo importante es que cuando testemos este parche, vamos a llamarlo 3a por simplicidad, en un entorno que tiene la corrección 3.</p> <p>Siempre ten un entorno o snapshot, que esté en el mismo nivel que el de producción.</p>
3 Urgencia – con branch ☐ - Hecho	 <p>2. Haz un branch sólo si es necesario</p> <p>Vamos a usar el mismo escenario del caso 2. Sin embargo, esta vez cuando analizamos el bug descubrimos que en las correcciones 4, 5 y 6 han cambiado el código de la misma área. Tendremos que crear un branch para poder trabajar en ello, pero esto lleva a más preguntas: ¿Desde qué branch debemos partir? ¿Qué versión debería tener este branch?</p> <p>Vamos a ver primero desde dónde. Nuestro branch de desarrollo ha progresado mucho desde la release y los branches de servicing también están muy lejos y están a punto de publicar la próxima versión. El mejor branch de partida es el branch de release.</p>

Estrategias de Branching – Tutoriales

Pasos	Instrucciones
	<p>Determinar la versión de branch dependerá de lo que esté en producción. Como en producción está la corrección 3, tenemos que identificar el changeset que incluyó la corrección 3 y hacer un branch a partir de ahí. Llamaremos a este branch 'crisis' en lugar de 3a por razones que veremos ahora. Ahora ya podemos crear las builds del branch, crear el parche, testarlo y entregarlo al cliente.</p> <p>4. Borrar la rama</p> <p>Si mergeamos los cambios a release, pueden incluirse en el branch de la corrección 6 para asegurarnos de que no vuelva a ocurrir. Sin embargo, en esta situación tenemos un problema mayor. Alguien podría decir que estamos ofreciendo demasiadas correcciones y que el tiempo desde que ofrecemos la corrección y la implementamos es demasiado largo. Esto puede ser verdad, sin embargo, este escenario volverá a ocurrir y tendremos que reevaluar la cadencia de nuestras releases para evitar que esta situación vuelva a ocurrir.</p> <p>En este escenario, en lugar de hacer un merge (RI) a release, tendremos que hacer un merge (FI) de todos los changesets que se hicieron para la corrección 4 desde release al branch 'crisis'. Ahora podemos usar este branch para producir la corrección 4a (por esto hemos llamado al branch crisis no 3a). Seguiremos el mismo proceso para generar la corrección 5a y luego hacemos merge (RI) a release. Borraremos el branch sólo después de que la corrección 6 llegue a producción. Siempre se comprensivo con el impacto de una corrección, el cliente se sentirá mejor si puede desechar una serie de correcciones y obtener un parche acumulativo si los beneficios tienen sentido. Ha tenido cierto coste pero la crisis ha pasado.</p>

Tabla 5 – Tutorial: Adapta tu proceso de branching para los casos excepcionales

Casos reales

Entregar software en intervalos desde días a meses

Los siguientes escenarios son implementaciones llevadas a cabo en proyectos reales que han entregado software a producción en intervalos tanto a largo plazo – 6 a 9 meses – como a medio plazo – 1 a 3 meses – y como en corto plazo – 1 a 14 días. El tamaño de los equipos va desde los 20 a más de 100 y normalmente necesitan un soporte de infraestructura de varios cientos de servidores. El código fuente es normalmente de unas 5 millones de líneas de código. Lo importante es que estos son sistemas grandes; sin embargo, el escenario puede ser aplicable a proyectos más pequeños.

NOTA

En este escenario, vamos a *doblar* alguna de las reglas, que hemos visto en la guía. Estaremos programando directamente en los branches de Main y Release e ilustraremos el uso de los conceptos que hemos visto en la guía para ofrecer una solución que funciona.

Tutorial

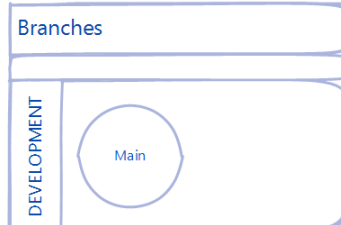
Este escenario tiene tres releases. Usaremos una secuencia numérica de tres partes con el siguiente formato: r.s.hh, donde r=Release, s=Servicing y h=Hotfix. Por ejemplo, nuestras tres release serán 1.1.0, 1.2.0 y 2.1.0

Las releases 1.1.0 y 1.2.0 nacerán del branch Servicing 1.0 y la 2.1.0 del branch servicing 2.0. Cuando los equipos debaten sobre las versiones/branches, está claro que un nombre con dos números es un branch de Servicing mientras que un nombre con tres números indican un branch de release. También está claro de qué branch de Servicing nació un branch de Release.

Etapas del escenario y sus detalles

Empezando – El equipo comienza con un branch principal. Los requisitos están claros y tenemos una fecha de entrega. En este punto, no sabemos qué versión será, pero el equipo se refiere a ella como release 1.

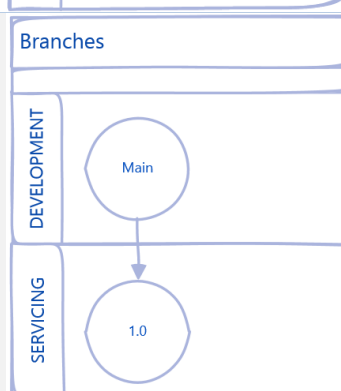
Los equipos de desarrollo, test y construcción trabajan juntos para desarrollar la solución.



Branch para estabilizar – Cuando el código está estable el equipo decide hacer un branch desde Main a una branch de servicio para arreglos finales de estabilidad.

Dos equipos pueden trabajar ahora en paralelo. En Main algunos equipos pueden comenzar a trabajar en la próxima versión, mientras que en Servicing 1.0 otros miembros del equipo pueden arreglar los últimos detalles.

Es de esperar que el equipo que está trabajando en el branch Servicing tarde un poco más para asegurarse de que se cumplen los requisitos de calidad de release, mientras que un subgrupo puede empezar a trabajar en los tests del branch principal. Dependiendo de tu nivel de automatización puede ocurrir lo contrario



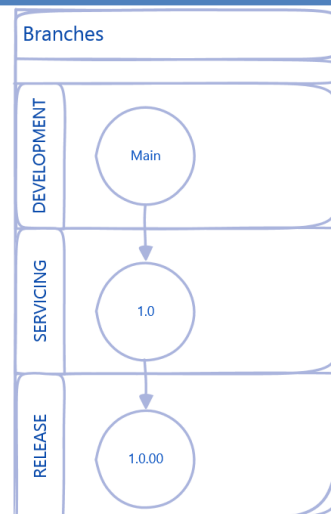
Etapas del escenario y sus detalles

Crear un branch para Release – Cuando consideres que el código es de una calidad suficiente, el equipo puede crear el branch de Release a partir de la de Servicing. En este punto, conviene pensar un poco el nombre exacto que le vas a dar a este branch de Release. Esperamos que no se hagan más de uno o dos branches de release a partir del branch Servicing 1.0, así que sólo añadiremos un dígito al branch de Servicing.

NOTA

Si esperas liberar 10 o más versiones a release, el nombre del branch de Servicing debería tener dos dígitos empezando con dos ceros, por ejemplo: 1.00.00. ¿Por qué? Sólo porque en el Source Control Explorer las releases se listarán en el orden correcto.

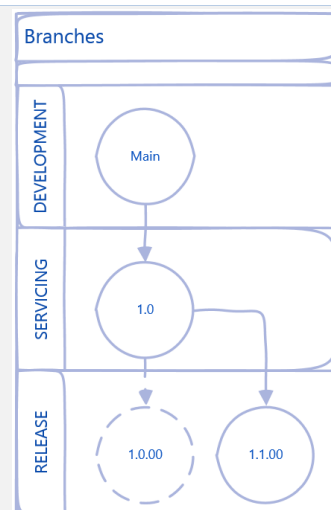
El branch Release 1.0.00 reúne los mínimos requisitos de calidad, que no tienen que ser los mismos requisitos de calidad que los de producción. Mientras la release está en el cliente, crea todas las correcciones en el branch de Release para corregirlos. Al mismo tiempo, un equipo está trabajando en el branch Servicing para estabilizar el código y reducir el número de bugs hasta que se consiga el nivel de calidad de producción. Cualquier corrección, por ejemplo, 1.0.01, 1.0.02 se vuelve a integrar en el branch Servicing 1.0 y eventualmente al branch padre y a los branches hijos.



Crear un branch para Release (otra vez) – En nuestro ejemplo, la Release 1.0.00 nunca llegará a producción. Ofrecemos esta release al cliente para que puedan testarla con otros partners.

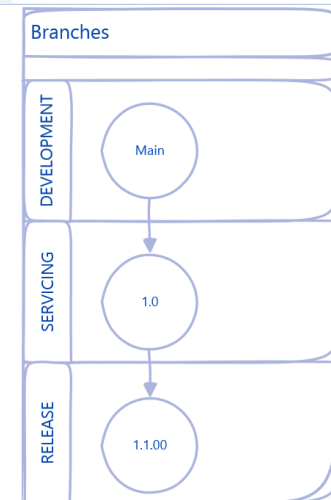
La Release 1.0.00 ha sido entregada con varias correcciones que han sido mergeadas a Servicing 1.0 y el equipo que trabaja en ese branch ha conseguido que alcance la calidad necesaria para producción. Se crea un branch llamado Release 1.1.00 a partir del branch Servicing 1.0.

Sólo es una cuestión de recursos, pero intenta limitar el número de branches que crees, especialmente las de Release ya que cualquier cambio que se haga en ellas tendrán que revertirse en otros branches. Idealmente usaremos el branch de Servicing como un buffer. Si un bug no es lo suficientemente prioritario, intenta convencer al cliente de que es mejor esperar a la próxima release para corregirlo. De esta manera no tendrás que cerrar y testar una corrección y el cambio puede hacerse en el branch de servicing donde habrá más tiempo para testar.



System Live – La Release 1.1.00 ya está en producción. Cualquier corrección se hace en el branch de Release, se testea y se integra en producción lo antes posible. Así se evitan situaciones en las que una corrección que se hace en Release y hace falta tiempo para lanzarla, otro tema es saber qué es más prioritario. En este caso suele ser más fácil deshacer el primer cambio, implementar la corrección crítica y rehacer el cambio. Otra manera es hacer un branch por changeset desde el branch de release, pero esto conlleva costes adicionales como configurar las builds y mover a un equipo para que trabaje en ese branch.

También usamos el branch de Servicing para correcciones a largo plazo de una release. Por ejemplo, si una corrección va a tardar dos semanas en implementarse y esperas corregir otras cosas en ese tiempo, el equipo puede trabajar en el branch de servicing y cuando terminen se mergean en release. Estamos asumiendo que el branch de Servicing no se usa para estabilizar cambios de código entre branches de release. Si ese es tu caso necesitarás hacer un branch de funcionalidad desde el branch de Release para tener el aislamiento adecuado.



Etapas del escenario y sus detalles

Madurez – Hablemos sobre el siguiente diagrama que muestra cómo poder soportar ocho branches. La de Producción es la Release 1.1.00

Main – aquí tenemos a un equipo que trabaja en 3 funcionalidades.

Servicing 1.0 – esta es sólo un puente de branches en este momento

Release 1.0.00 – ha sido borrada y la mostramos para ilustrar cómo podemos tener varios branch de release a partir de la de Servicing.

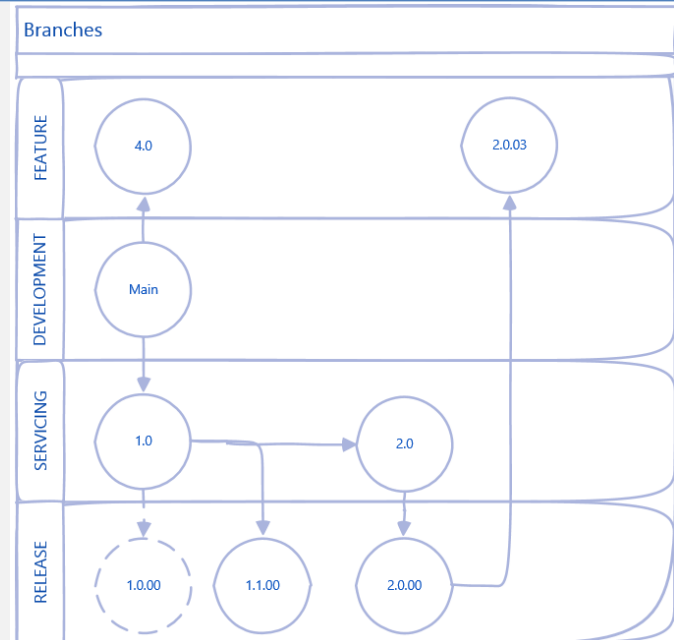
Release 1.1.00 – aquí es donde trabajamos para las correcciones de producción.

Servicing 2.0 – tiene un equipo estabilizando el código para el próximo branch 2.1.00 y que reemplazará al branch 1.1.00 en producción.

Release 2.0.00 – está ahí para soportar la release que el cliente está usando para testar la integración con otros sistemas.

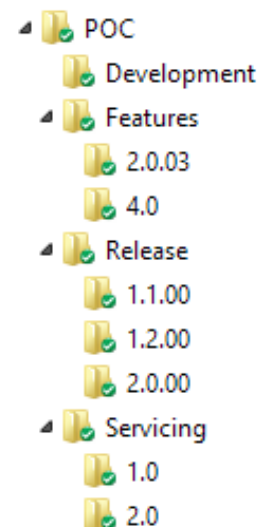
Feature 2.0.03 – es una corrección en la que estamos trabajando que sabemos que nos llevará algún tiempo en implementar y que no podemos trabajar en el branch Servicing mientras se estabiliza el código para la version 2.1.0. Una vez que el código esté estabilizado lo mergearemos al branch 2.0.00, y seguramente borrarémos el branch 2.0.03.

Feature 4.0 – aquí se está trabajando en una funcionalidad que se espera sólo estará lista en la próxima mayor release del branch principal. Es importante recordar que un branch de funcionalidad se puede crear desde el branch principal para trabajar en algo que no se ha planeado para la release; de esta manera el branch principal se puede liberar sin la funcionalidad en vez de tener que deshacer los cambios de main porque el cliente haya cambiado de opinión.



Puntos clave

- Trabajamos en Main para mayor releases pero debemos usar branches de funcionalidad para protegernos de cancelaciones o cambios de prioridad en las funcionalidades.
- Trabajamos en los branches de Release y no las tratamos como si fuesen inmutables. Usamos branches de funcionalidad adicionales y de Servicing cuando esperemos que una corrección se lleve mucho tiempo. De esta manera dejamos el branch de Release disponible para corregir bugs de alta prioridad.
- Todo lo que necesitamos para garantizar que podemos repetir una build es un changeset ya que es inmutable. Los labels y branches son sólo una conveniencia. Úsalas a tu favor para simplificar el soporte de una entrega de tu software.
- Mergeamos de Release a Servicing tan pronto como se haga la corrección. Normalmente mergearemos el cambio a Main y otros branches posteriores para no retroceder. Mergeamos desde Servicing a Main cuando creemos un branch de Release.
- Dejaremos el branch de Servicing aunque sólo sea como pasarela de branches durante un tiempo. Nos ofrece un lugar natural del que partir para hacer correcciones que vayan a ser especialmente largas y podemos usarlas para crear otra release completa si cambian las circunstancias del proyecto.
- Usaremos un sistema de nombrado consistente en nuestros branches.
- Intentaremos limitar el número de branches que necesitamos para hacer correcciones ya que suelen tener un alto costo de entrega.
- Organizaremos nuestros branches en directorios para una navegación más sencilla como vemos aquí...



FAQ

Estrategias

¿Qué diferencia hay entre branches y directorios?

A partir de TFS 2010 hay una diferencia entre branches y directorios en el control de versiones. Un branch habilita ciertas funcionalidades relacionadas con los branches como la visualización, así como poder guardar propiedades como el propietario y comentarios a cada branch. Para más información leed.

- [Branch Folders and Files](#) ²¹
- [View the Branch Hierarchy of a Team Project](#) ²²
- [View Where and When Changesets Have Been Merged](#) ²³
- [Team Foundation Server Permissions](#) ²⁴

¿Por qué hemos cambiado el nombre de las estrategias?

Hemos oído cómo los ingenieros y los usuarios hablaban sobre el branching en diálogos casuales y técnicos. La decisión fue incluir nombres de estrategias más significativos que podamos visualizar en nuestra mente.

Por ejemplo, visualiza un plan de branch plano. Ahora repite el ejercicio, visualiza uno desarrollo, luego uno de release, y finalmente la estrategia de **development y release** isolation. ¿Qué fue más fácil de visualizar?

Operaciones

¿Los work ítems forman parte del modelo de branching/merging?

No, los work ítems existen a un nivel lógico muy diferente del repositorio de control de código. Cuando haces una operación en un branch: no se duplican los work ítems, no se traslada la historia de un work ítem, y no se actualizan los enlaces de un work ítem. Para más información leed: [Copy a Work Item](#) ²⁵.

¿Se pueden crear branches entre team projects?

Sí, puedes pero no es recomendable a menos que dos equipos necesiten compartir el código y no puedan compartir un proceso común.

¿Cómo administramos los bugs entre los branches?

Como vimos en [¿Los work ítems forman parte del modelo de branching/merging?](#), página 34, crear branches no tiene ningún impacto en los work ítems, incluidos los bugs. Considera quién es el responsable del código y de los bugs, es una responsabilidad del que ha escrito el código o del que ha “movido” el código.

Para más información, leed [Copy a Work Item](#) ²⁶ y [Create or Delete Relationships Between Work Items](#) ²⁷.

¿Hay algo de valor en el “Branch desde una versión” cuando creamos un branch?

Cuando creamos un branch, hay cinco opciones disponibles para elegir el punto de partida:

²¹ [http://msdn.microsoft.com/en-us/library/ms181425\(VS.110\).aspx](http://msdn.microsoft.com/en-us/library/ms181425(VS.110).aspx)

²² [http://msdn.microsoft.com/en-us/library/dd465202\(VS.110\).aspx](http://msdn.microsoft.com/en-us/library/dd465202(VS.110).aspx)

²³ [http://msdn.microsoft.com/en-us/library/dd405662\(VS.110\).aspx](http://msdn.microsoft.com/en-us/library/dd405662(VS.110).aspx)

²⁴ [http://msdn.microsoft.com/en-us/library/ms252587\(VS.110\).aspx](http://msdn.microsoft.com/en-us/library/ms252587(VS.110).aspx)

²⁵ [http://msdn.microsoft.com/en-us/library/ms181321\(VS.110\).aspx](http://msdn.microsoft.com/en-us/library/ms181321(VS.110).aspx)

²⁶ [http://msdn.microsoft.com/en-us/library/ms181321\(VS.110\).aspx](http://msdn.microsoft.com/en-us/library/ms181321(VS.110).aspx)

²⁷ [http://msdn.microsoft.com/en-us/library/dd286694\(v=vs.110\).aspx](http://msdn.microsoft.com/en-us/library/dd286694(v=vs.110).aspx)

- **Labels y workspaces** no se suelen referir a un punto en el tiempo si no a una colección de archivos agrupados por el usuario.
- **Date, Latest Version o Changeset** se refiere normalmente a un punto en el tiempo y soluciona las necesidades de equipos más grandes. Por ejemplo, para la resolución de bugs saber cuándo se introdujo suele ser una fecha dada. Cuando nos movemos a otro sprint o iteración las opciones más adecuadas suelen ser Latest Version o Changeset.

¿Se pueden borrar los branches?

Sí. Tienes que entender tu jerarquía de branches y saber las implicaciones que tendrá en un futuro borrar un branch. Cuando borras un branch, estás eliminando una relación entre padre e hijos, estás añadiendo la necesidad de hacer merge sin un fundamento claro.

Por ejemplo, el diagrama de la izquierda permite que los cambios se propaguen de A -> B, B->C y viceversa a través de la jerarquía de branches. Si borras el branch B (la imagen de la derecha), sólo podrás hacer cambios de A->C y de C->A podrás hacer merges sin sincronización.

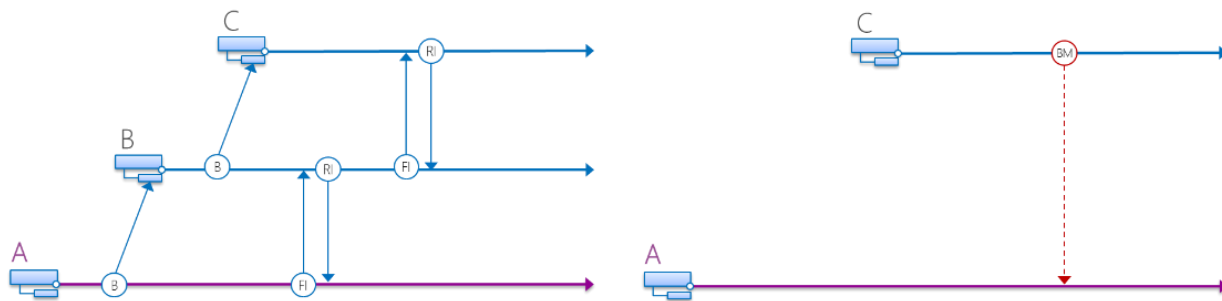


Figura 14 – Jerarquía de branches A -> B -> C

Para más información leed [Branch Re-parenting](#)²⁸ y [Team Foundation Server Permissions](#)²⁹

¿Qué hace falta para consolidar o hacer transiciones entre branches?

Antes de considerar una consolidación o transición entre branches hazte las siguientes preguntas:

- ¿Cuáles son las verdaderas razones para hacer esto?
- ¿Cuáles son los beneficios?
- ¿Cuáles son las ventajas de mantener el status quo?
- ¿Cuáles son los riesgos de la consolidación?
- ¿Qué recursos son necesarios para completar la consolidación?
- ¿Cuál es el plan de escape?
- ¿Qué efectos tendrá en el futuro?
- ¿Cómo incluiremos las mejores prácticas?
- ¿Necesitamos mantener el histórico y los work ítems asociados al código?

Una vez que hayamos respondido a estas preguntas, podremos saber cuáles son los siguientes pasos que queremos para consolidar nuestros branches o crear un nuevo entorno. Cambiar a un nuevo branch puede ser la más fácil de las dos opciones. Puedes bloquear todos los branches existentes y hacer que todos trabajen en el nuevo entorno desde el punto de comienzo designado. Esto suele ser más apropiado cuando las aplicaciones han sido creadas y mantenidas por equipos que estaban separados y que no tenían nada en común.

²⁸ <http://blogs.msdn.com/b/hakane/archive/2009/05/19/enable-branch-visualization-in-upgraded-team-projects-tfs-2010-beta1.aspx>

²⁹ [http://msdn.microsoft.com/en-us/library/ms252587\(VS.110\).aspx](http://msdn.microsoft.com/en-us/library/ms252587(VS.110).aspx)

Para consolidar branches, tendrás que revisar tus branches actuales y decidir qué se debe incluir. Una vez que hayas tomado esas decisiones, podrás crear una nueva línea principal y proceder a migrar cada branch designado. Junto a cada paso, deberás verificar que cada branch migrada no rompe la línea principal. Si es posible, haz esto en un entorno controlado antes de hacerlo en un entorno de producción.

Para más información leed [Branch Folders and Files](#)³⁰ y [Merge Folders and Files](#)³¹.

¿Qué es la funcionalidad “Re-parent Branch” y cuándo debería usarla?

La funcionalidad de Re-parent Branch podemos usarla para establecer una relación padre-hijo entre branches que no son padre e hijo y para alterar relaciones padre-hijo ya existentes en una jerarquía de branches.

Sin embargo, para “revertir” una relación existente padre-hijo, tenemos que desconectar el branch hijo de su padre usando la opción “No parent”, y después hacer el “re-parent” entre los branches padre e hijo que queramos.

Leed [Branch Re-parenting](#)³² para más información.

¿Cuándo deberíamos usar labels?

Recomendamos los labels en aquellos casos en los que necesitemos un snapshot del código para referencias futuras y/o cuando la mutabilidad de los labels no sea un problema.

Seguridad

¿Puede ser realmente inmutable un branch?

¿Puedes cerrar tu casa para asegurarte de que no entra nadie? La respuesta suele ser “no”, aunque podemos bloquear un branch con permisos, la verdadera inmutabilidad depende de nuestra arquitectura de seguridad.

¿Cómo administro los permisos de las ramas en mi equipo?

Deberías evaluar con cuidado cómo administrar los permisos de acceso al sistema de control de código. Tómate un tiempo para evaluar los niveles de acceso que necesitas entre los roles y definir los roles y responsabilidades en una matriz puede ayudarte a elegir la solución adecuada. Considera el uso de grupos de seguridad, basados en Active Directory, ya que TFS se actualizará automáticamente. Recuerda incluir a aquellas personas fuera de tu equipo que puedan necesitar acceso, como los de IT o Soporte. También tendrás que tener en cuenta las políticas de seguridad corporativa o gubernamentales que apliquen.

Leed [Securing Team Foundation Server](#) para más información.

³⁰ [http://msdn.microsoft.com/en-us/library/ms181425\(VS.110\).aspx](http://msdn.microsoft.com/en-us/library/ms181425(VS.110).aspx)

³¹ [http://msdn.microsoft.com/en-us/library/ms181428\(VS.110\).aspx](http://msdn.microsoft.com/en-us/library/ms181428(VS.110).aspx)

³² <http://blogs.msdn.com/b/hakane/archive/2009/05/19/enable-branch-visualization-in-upgraded-team-projects-tfs-2010-beta1.aspx>

Hands-on Lab (HOL) – De la simplicidad a la complejidad, ¿o no?

Prerrequisitos

Necesitas lo siguiente para completar este lab:

- La solución de ejemplo, documentada a continuación.
- La última [Brian Keller VM](#) ³³ o un entorno con:
 - Visual Studio 2012 Professional o superior
 - Team Foundation Server 2012 o superior

Nuestra solución de ejemplo y su contexto

La solución de nuestro ejemplo, que se entrega junto a este HOL en el sitio de CodePlex [Version Control \(formerly the Branching and Merging\) Guide](#) ³⁴ se basa en el ejemplo de Managed Extensions Framework (MEF) cubierto en detalle en [Managed Extensions Framework \(MEF\)... simplicity rules](#) ³⁵.

Empezaremos con nuestra solución favorita de la calculadora que nos permitirá sumar y restar y evolucionará para incluir la multiplicación y la división durante el lab. El foco no está en MEF, C#, .NET o en el excitante mundo del debugging, sino en la cuestión “*creamos un branch o no*”.

Esperamos que os guste el viaje y os pedimos que leáis las guías que acompañan a esta para ver más detalles de branching y las funcionalidades de Team Foundation Version Control

Ejercicio 1: Configuración del entorno

META

En este lab, asumimos que no tenemos un entorno pre configurado, crea un team Project, descarga y haz check-in de la solución de ejemplo.

Tarea 1: Lógate en tu entorno

AVISO

Por favor **no** uses un entorno de producción para estos HOLs, ¡no queremos team projects o branches no deseados en producción!

Paso	Instrucciones
1 Logon <input type="checkbox"/> - Hecho	<ul style="list-style-type: none">• Si te estás logando en una instancia de BK's VM, por ejemplo en TechReady usa las credenciales administrator P2ssw0rd.• Si no, usa las credenciales de tu entorno con las credenciales que te permitan crear un Team Project.

Tabla 6 – Lab 1, Tarea 1

³³ <http://aka.ms/almvms>

³⁴ <http://aka.ms/treasure18>

³⁵ http://blogs.msdn.com/b/willy-peter_schaub/archive/2012/11/23/willy-s-cave-dwelling-notes-10-managed-extensions-framework-mef-simplicity-rules.aspx

Tarea 2: Crea un Team Project

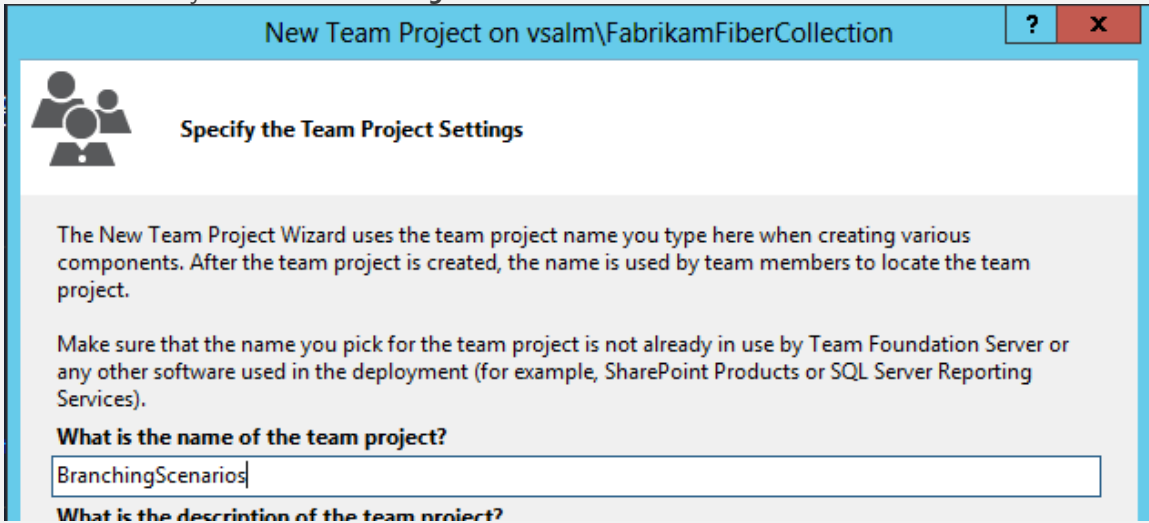
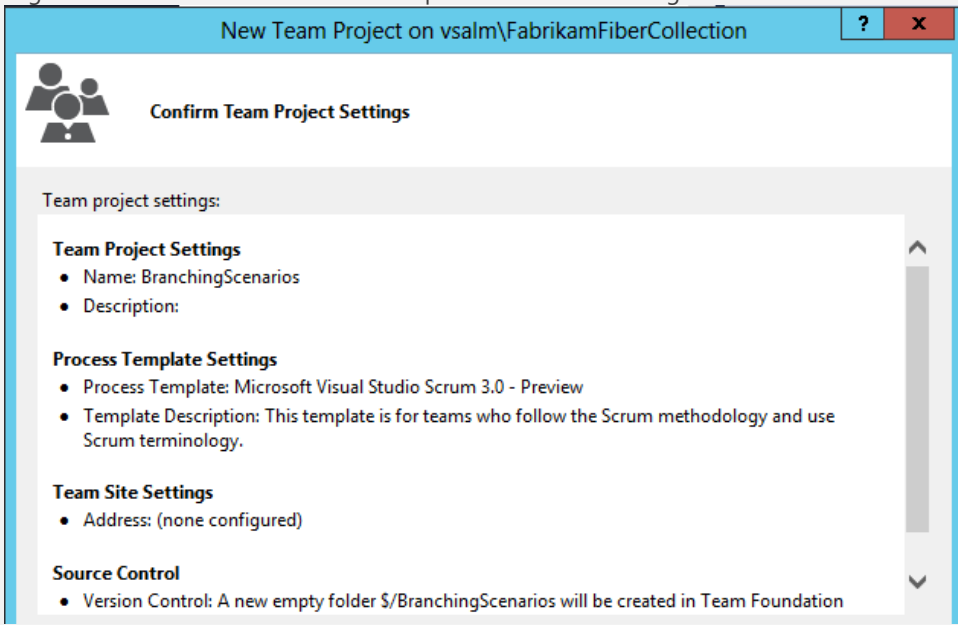
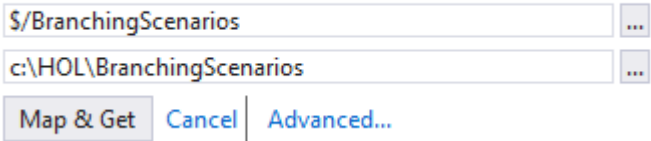
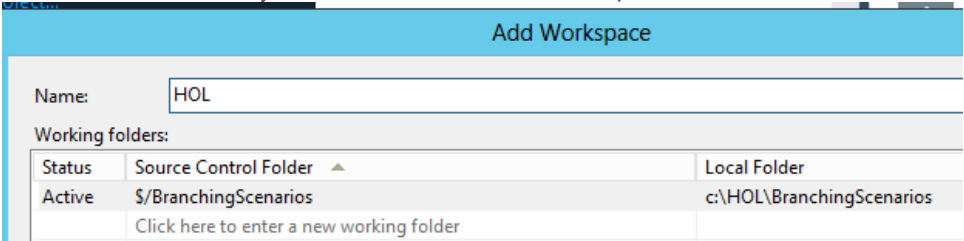
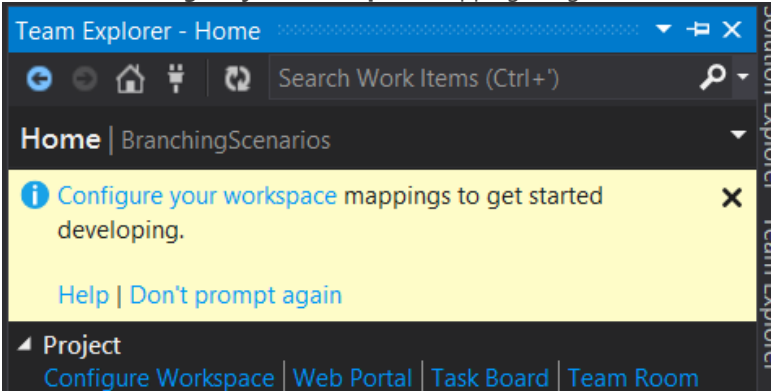
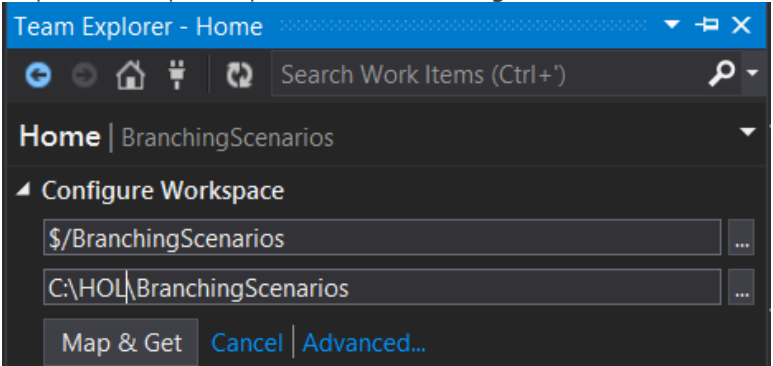
Paso	Instrucciones
1 Abre Visual Studio □ - Hecho	<ul style="list-style-type: none"> Abre Visual Studio
2 Crea un Team Project (TP) □ - Hecho	<ul style="list-style-type: none"> Crea un Team Project llamado BranchingScenarios  <ul style="list-style-type: none"> Selecciona la plantilla que quieras. No te hará falta un portal de SharePoint Elige Team Foundation Version Control para tu control de código fuente. 

Tabla 7 – Lab 1, Tarea 2

Tarea 3: Crea tu sandbox

Paso	Instrucciones
1 Configura los mappings para tu espacio de trabajo ☐ - Hecho	<p>Si usas Visual Studio 2012 (ve más abajo si usas VS 2013)</p> <ul style="list-style-type: none"> Configura los mappings de tu espacio de trabajo Ayuda: en el Source Control Explorer, selecciona el desplegable Workspaces, edita el diálogo Configure Workspaces. Cambia el path de tu workspace a c:\HOL\BranchingScenarios <p>Configure Workspace</p>  <ul style="list-style-type: none"> Selecciona Advanced... y dale el nombre HOL a tu workspace.  <p>Si usas Visual Studio 2013</p> <ul style="list-style-type: none"> Conéctate al nuevo team Project llamado BranchingScenarios Selecciona configure your workspace mappings to get started  <ul style="list-style-type: none"> Mapea el workspace al path c:\HOL\BranchingScenarios  <ul style="list-style-type: none"> Selecciona Advanced... y dale el nombre HOL a tu workspace Selecciona Map & Get

Estrategias de Branching – Hands-on Lab (HOL) – De la simplicidad a la complejidad, ¿o no?

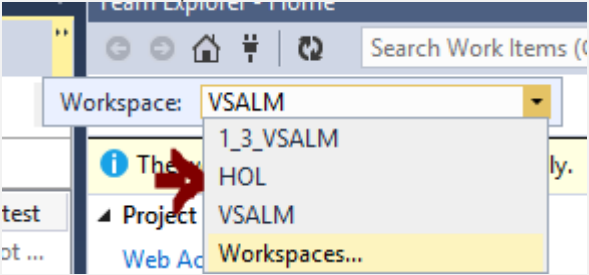
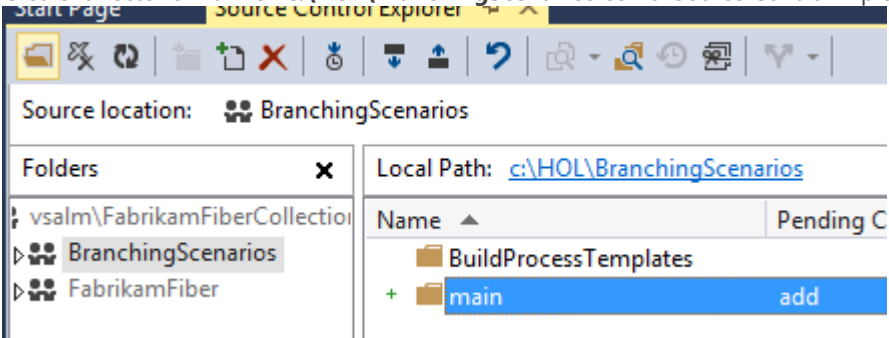
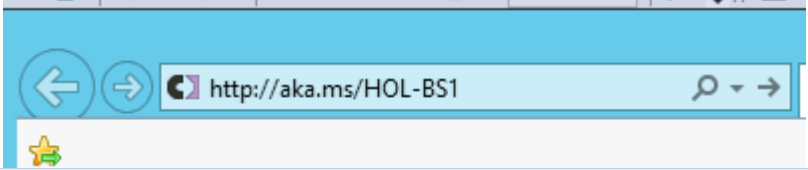
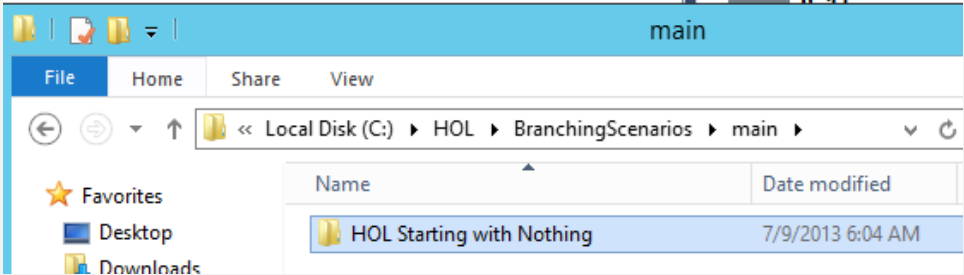
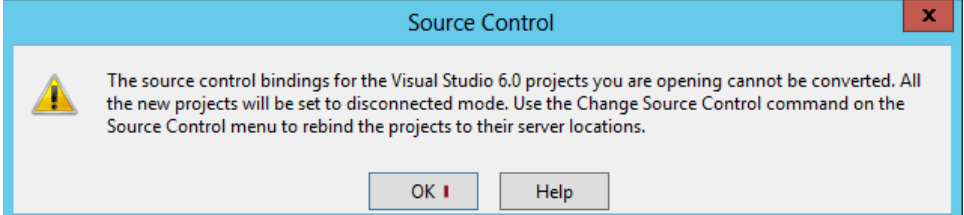
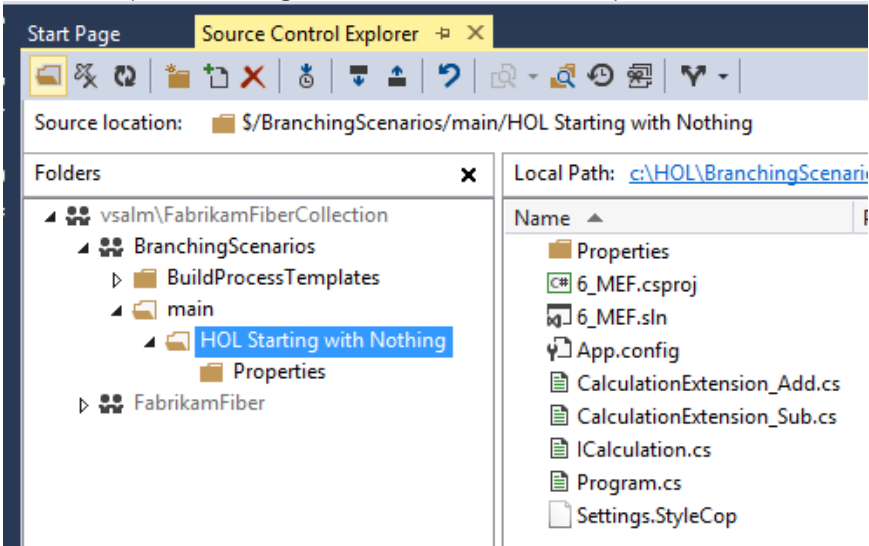
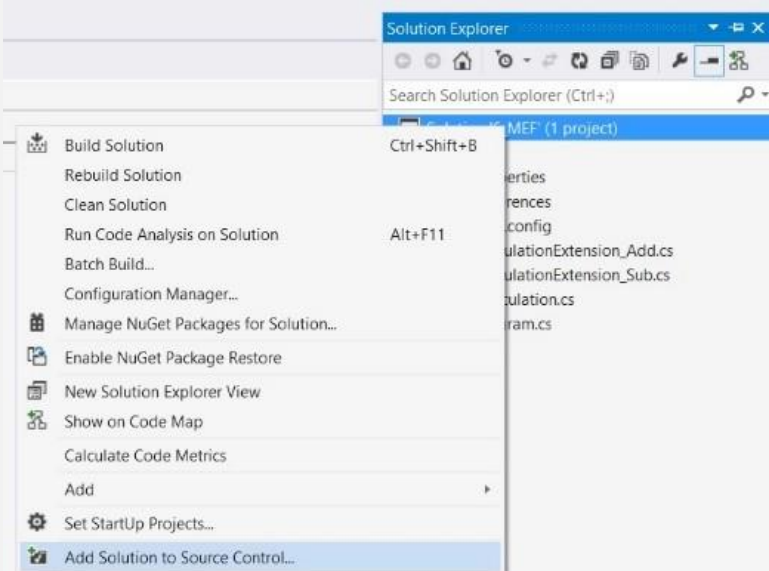
Paso	Instrucciones
2 Mapea el directorio local ☐ - Hecho	<ul style="list-style-type: none"> Ve al Source Control Explorer Cambia tu workspace a HOL 
3 Crea un directorio main ☐ - Hecho	<ul style="list-style-type: none"> Crea el directorio main en c:\HOL\BranchingScenarios con el Source Control Explorer 

Tabla 8 – Lab 1, Tarea 3

Tarea 4: Descarga y haz check in de la solución de ejemplo

Paso	Instrucciones
1 Descarga el código de ejemplo ☐ - Hecho	<ul style="list-style-type: none"> Descarga el código de ejemplo de http://aka.ms/hol-bs1 
2 Descomprime el código de ejemplo ☐ - Hecho	<ul style="list-style-type: none"> Descomprime el proyecto de ejemplo en c:\hol\BranchingScenarios\main 
3 Explora ☐ - Hecho	<ul style="list-style-type: none"> Abre la solución 6_MEF de c:\hol\BranchingScenarios\main. Selecciona OK cuando veas el siguiente aviso de los proyectos de Visual Studio 6.0 

Estrategias de Branching – Hands-on Lab (HOL) – De la simplicidad a la complejidad, ¿o no?

Paso	Instrucciones
	<ul style="list-style-type: none"> Explora la solución de ejemplo. Es una aplicación de consola Usaremos el parámetro Debug Command para definir datos de test, por ejemplo: 1 1 + para sumar 1 y 1. Asegúrate de que compila y se ejecuta sin errores. Verifica de que tienes lo siguiente en el Source Control Explorer 
<p>4</p> <p>Añádelo al control de código</p> <p>☐ - Hecho</p>	<ul style="list-style-type: none"> Haz clic derecho en la solución en el Solution Explorer y selecciona Add Solution to Source Control... 

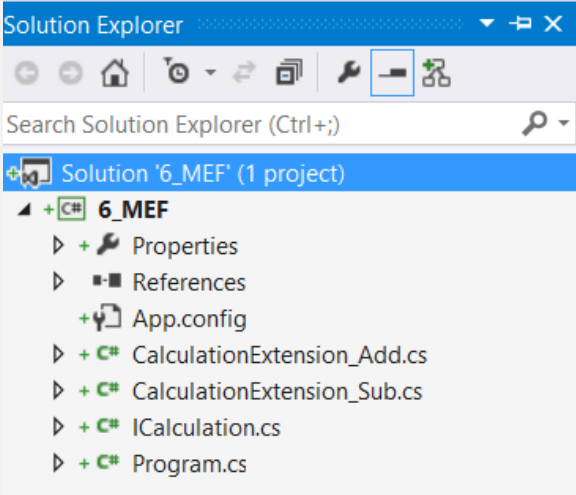

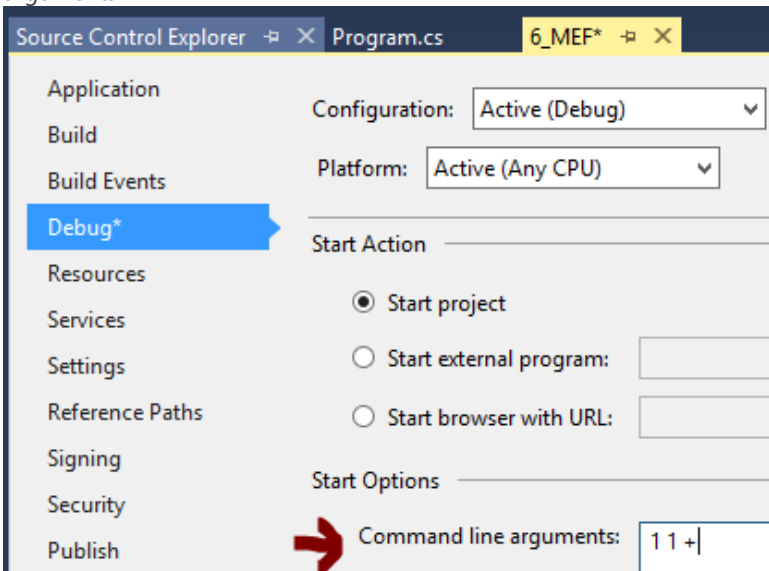
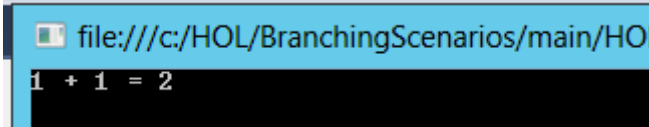

Paso	Instrucciones
	<ul style="list-style-type: none"> La solución, el proyecto y los archivos asociados ya están enlazados al control de código. 
5 Compila y ejecuta  - Hecho	<ul style="list-style-type: none"> Haz clic derecho en el proyecto 6_MEF en el Solution Explorer y selecciona Properties Selecciona la pestaña de Debug y define los siguientes argumentos en el textbox "Command line arguments" 1 1 +  <ul style="list-style-type: none"> Compila y ejecuta el programa (F5) y comprueba que obtienes este resultado:  <ul style="list-style-type: none"> Guárdalo todo
6 Check-in  - Hecho	<ul style="list-style-type: none"> Haz clic derecho en el Solution Explorer y selecciona Check In. Añade un comentario al check-in Haz clic en el botón Check In para enviar los cambios al control de código.

Tabla 9 – Lab 1, Tarea 4 – Descarga y haz check in de la solución de ejemplo

Ejercicio 2: MAIN Only – Reglas simples

META

Explorar la estrategia de branching **Main Only**. Recomendamos esta estrategia y evitar el advenimiento de branches y sus estrategias de merge asociadas, nosotros usamos esta estrategia para todos los proyectos nuevos de los proyectos de los ALM Ranger.

Contexto

El equipo necesita implementar la funcionalidad de DIVISIÓN, no necesitan aislamiento (aún) y se sienten cómodos usando labels para indicar sus hitos. Usando la estrategia Main Only evitas tener que hacer branches y los subsiguientes merges, minimizando así la complejidad del mantenimiento y los costes asociados.



Figura 15 – Estrategia Main Only

Tarea 1: Desarrollando en Main

AVISO

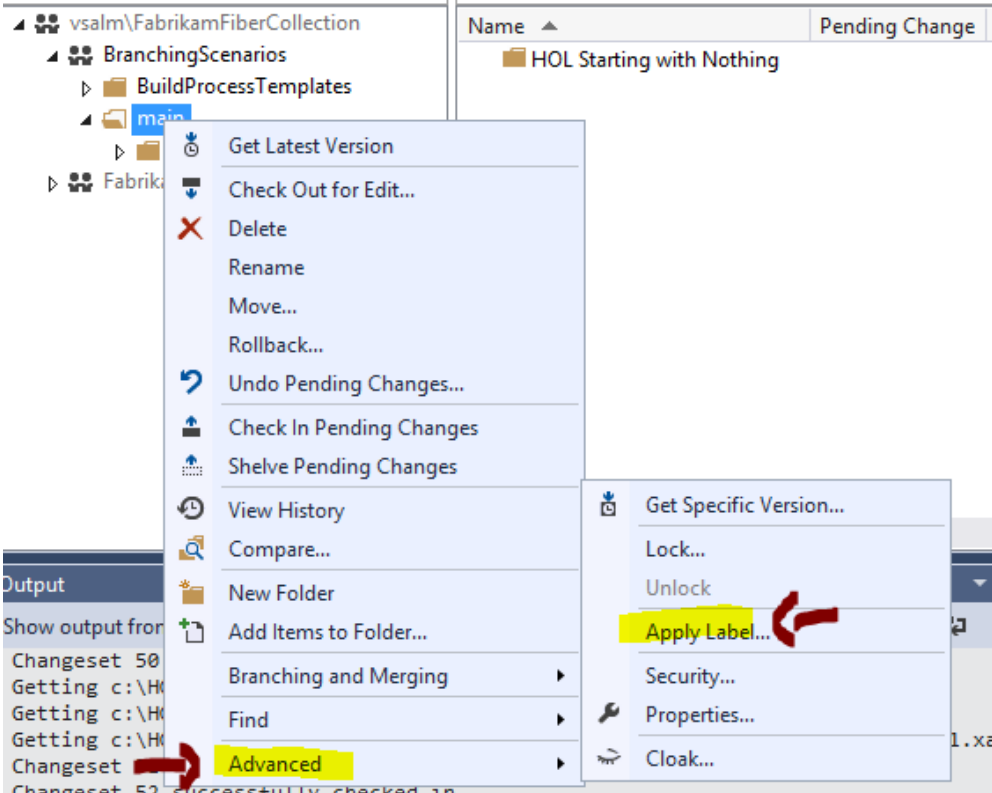
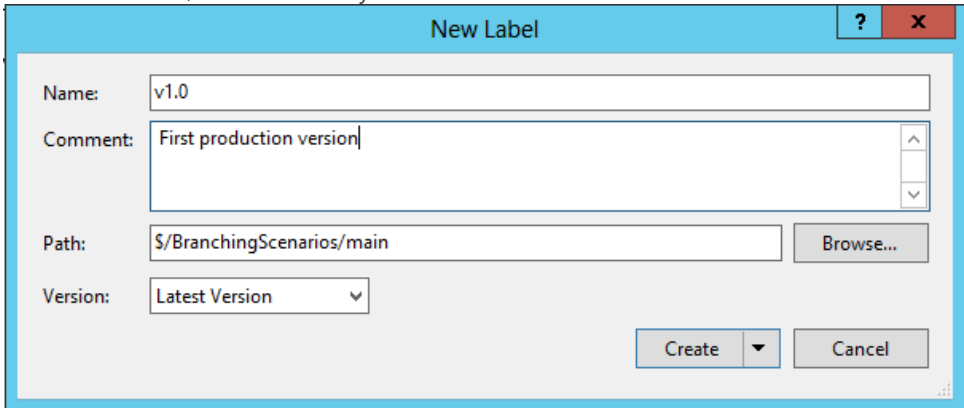
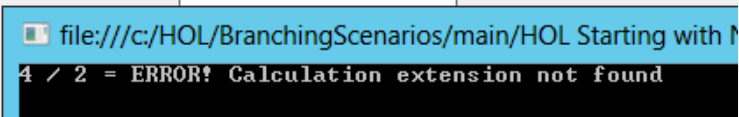
En el mundo real, debes implementar estrategias de BVT (Build-Verify-test), para proteger la calidad del branch principal y otros branches. Leer [Visual Studio Test Tooling Guide](#) ³⁶, [Team Foundation Build Guidance](#) ³⁷ y [Unit Test Generator](#) ³⁸ para más información sobre este tema, ya que no se cubre en este tutorial para mantener el foco en las estrategias de branching y evitar distracciones que puedan ser causadas por las builds y la ejecución de los tests.

Paso	Instrucciones
1 Label ☐ - Hecho	<ul style="list-style-type: none">Para empezar crearemos un label para la última versión del código que acabamos de subir, lo llamaremos v1.0

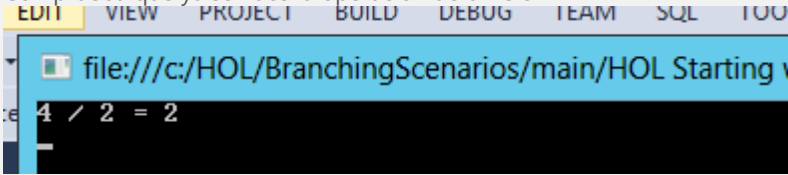
³⁶ <http://aka.ms/treasure27>

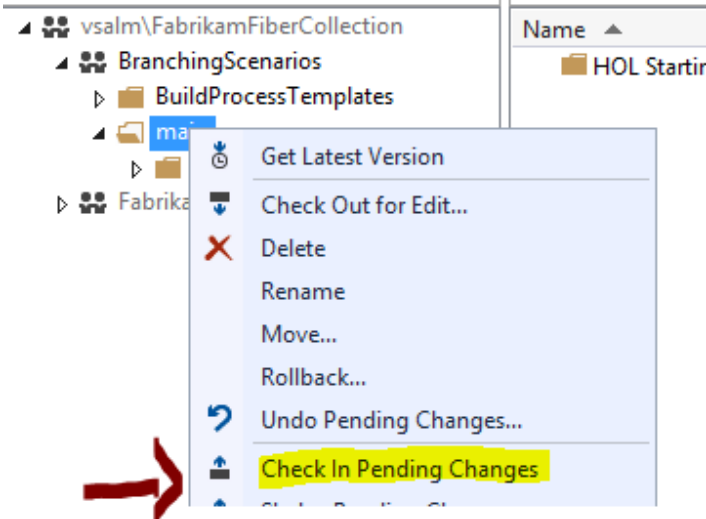
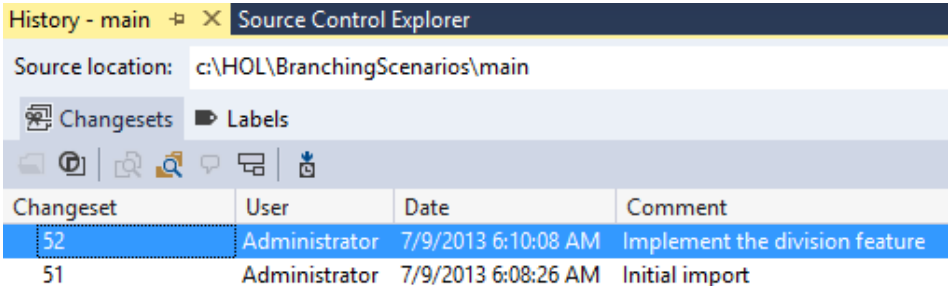
³⁷ <http://aka.ms/treasure23>

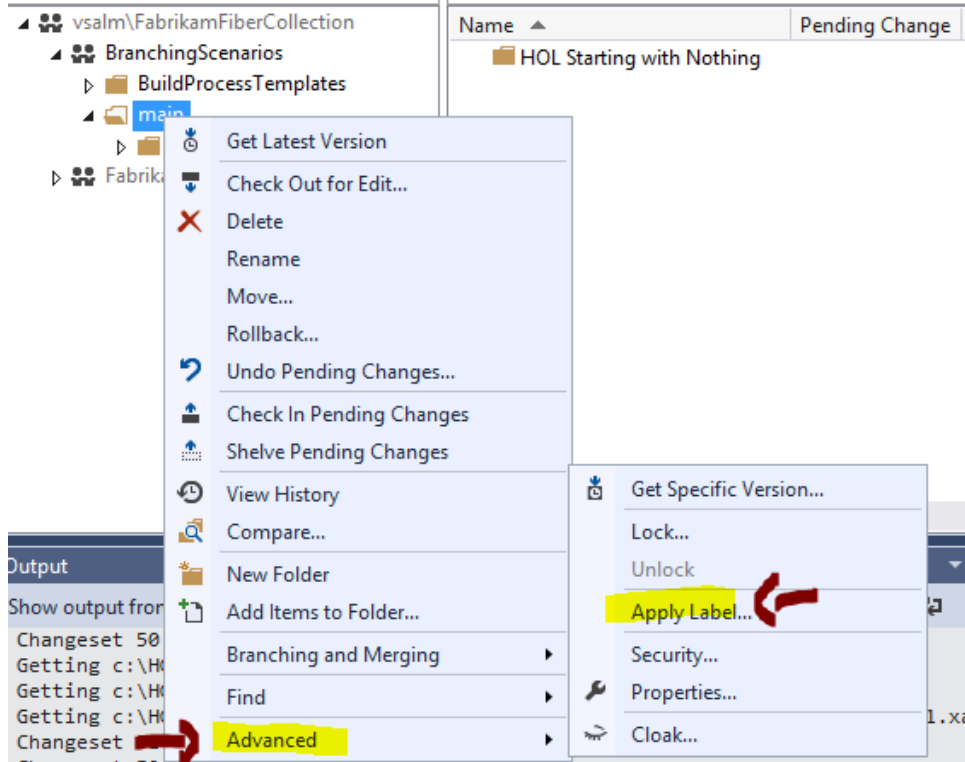
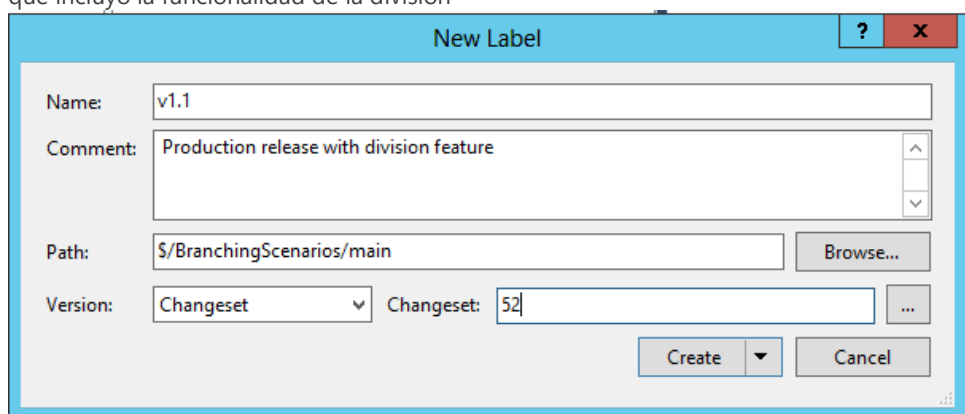
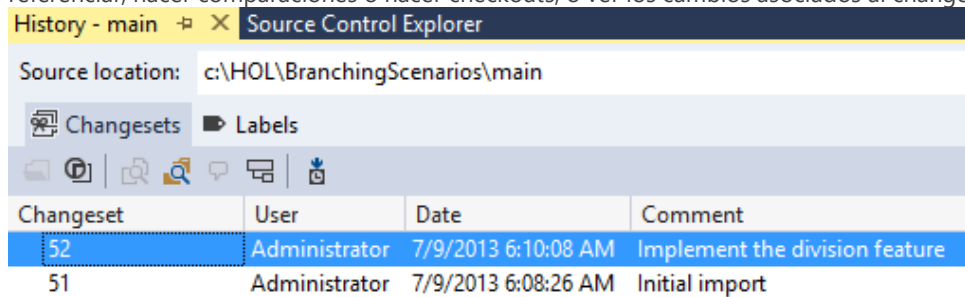
³⁸ <http://aka.ms/treasure50>

Paso	Instrucciones
	<ul style="list-style-type: none"> Haz clic derecho en el directorio principal en el Source Control Explorer, selecciona Advanced y Apply Label  <ul style="list-style-type: none"> Define el nombre, un comentario y selecciona Latest Version 
<p>2</p> <p>Ejecuta para validar</p> <p>☐ - Hecho</p>	<ul style="list-style-type: none"> Haz clic derecho en el proyecto 6_MEF en el Solution Explorer y selecciona Properties Selecciona la pestaña Debug y define los siguientes argumentos para la línea de comandos 4 2 / <p>Start Options</p> <p>Command line arguments: <input type="text" value="4 2 /"/></p> <ul style="list-style-type: none"> Compila y ejecuta el programa (F5) y comprueba que la calculadora falla ya que aún no puede dividir: 

Estrategias de Branching – Hands-on Lab (HOL) – De la simplicidad a la complejidad, ¿o no?

Paso	Instrucciones
3 Extiende la funcionalidad ☐ - Hecho	<ul style="list-style-type: none"> Añade la clase CalculationExtension_Div al proyecto y la lógica de la división escribiendo o haciendo copiar-pegar de la clase *_Add Debes definir el símbolo ExportMetadata, el nombre de la clase y la división <pre>[Export(typeof(ICalculation))] [ExportMetadata("Symbol", '/')] 0 references public class CalculationExtension_Div : ICalculation { 4 references public long Calculation(long valueOne, long valueTwo) { return valueOne / valueTwo; } }</pre>
4 Testa la funcionalidad ☐ - Hecho	<ul style="list-style-type: none"> Compila y ejecuta (F5) la solución Comprueba que ya se hace la operación de división  La funcionalidad ha sido implementada y testeada por un desarrollador Ya podemos hacer check-in de los cambios sin comprometer la calidad del directorio main

Paso	Instrucciones												
5 Check in y label ☐ - Hecho	<ul style="list-style-type: none"> Haz Check in de los cambios del código haciendo clic derecho en el directorio Main en el Source Control Explorer y selecciona Check In Pending Changes  <p>Haz clic derecho en el directorio Main en el Source Control Explorer, selecciona la opción View History y toma nota de los changesets. Deberías tener uno por la importación inicial y otro por el checkin de la funcionalidad de la división</p>  <table border="1"> <thead> <tr> <th>Changeset</th> <th>User</th> <th>Date</th> <th>Comment</th> </tr> </thead> <tbody> <tr> <td>52</td> <td>Administrator</td> <td>7/9/2013 6:10:08 AM</td> <td>Implement the division feature</td> </tr> <tr> <td>51</td> <td>Administrator</td> <td>7/9/2013 6:08:26 AM</td> <td>Initial import</td> </tr> </tbody> </table>	Changeset	User	Date	Comment	52	Administrator	7/9/2013 6:10:08 AM	Implement the division feature	51	Administrator	7/9/2013 6:08:26 AM	Initial import
Changeset	User	Date	Comment										
52	Administrator	7/9/2013 6:10:08 AM	Implement the division feature										
51	Administrator	7/9/2013 6:08:26 AM	Initial import										

Paso	Instrucciones												
	<ul style="list-style-type: none"> Haz clic derecho en el directorio Main en el Source Control Explorer, selecciona Advanced y Apply Label 												
	<ul style="list-style-type: none"> Puedes seleccionar Latest Changeset como antes para definir el número de Changeset del checkin que incluyó la funcionalidad de la división 												
	<ul style="list-style-type: none"> Haz clic en el directorio Main en el Source Control Explorer, selecciona View History y luego selecciona la pestaña Labels Ahora tienes dos labels que indican hitos específicos en tu branch principal, que puedes usar para referenciar, hacer comparaciones o hacer checkouts, o ver los cambios asociados al changeset:  <table border="1"> <thead> <tr> <th>Changeset</th> <th>User</th> <th>Date</th> <th>Comment</th> </tr> </thead> <tbody> <tr> <td>52</td> <td>Administrator</td> <td>7/9/2013 6:10:08 AM</td> <td>Implement the division feature</td> </tr> <tr> <td>51</td> <td>Administrator</td> <td>7/9/2013 6:08:26 AM</td> <td>Initial import</td> </tr> </tbody> </table>	Changeset	User	Date	Comment	52	Administrator	7/9/2013 6:10:08 AM	Implement the division feature	51	Administrator	7/9/2013 6:08:26 AM	Initial import
Changeset	User	Date	Comment										
52	Administrator	7/9/2013 6:10:08 AM	Implement the division feature										
51	Administrator	7/9/2013 6:08:26 AM	Initial import										

Estrategias de Branching – Hands-on Lab (HOL) – De la simplicidad a la complejidad, ¿o no?

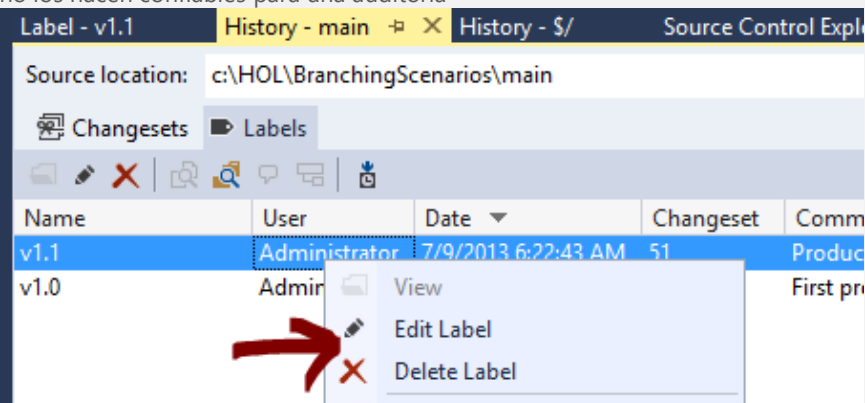
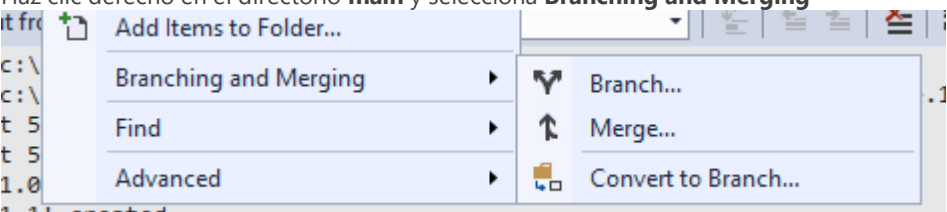
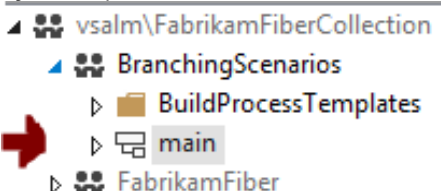
Paso	Instrucciones
6 "Manchando" el Label issue ☐ - Hecho	<ul style="list-style-type: none"> Haz clic derecho en cualquiera de los labels Fíjate en las opciones de edición y borrado de los labels, convierten al label en mutable y por tanto no los hacen confiables para una auditoría 

Tabla 10 – Lab 2, Tarea 1

Tarea 2: Opcionalmente convierte el directorio Main en un branch principal

Convertir el directorio principal en un branch es un paso opcional. El proceso de branch realiza este paso cuando creamos branches de la principal. Los ALM Rangers usan frecuentemente la utilidad [TFS Branch Tool](#)³⁹ para implementar su entorno de control de versiones, que realiza este paso automáticamente como parte de la configuración.

Paso	Instrucciones
1 Comprueba las funcionalidades ☐ - Hecho antes de la conversión	<ul style="list-style-type: none"> Haz clic derecho en el directorio main y selecciona Branching and Merging 
2 Convertir un directorio en un branch ☐ - Hecho	<ul style="list-style-type: none"> Fíjate en que podemos hacer un Branch, Merge y Convert to Branch Haz clic derecho en el directorio main y selecciona Branching and Merging, Convert to Branch. Opcionalmente añade una descripción y selecciona Convert
3 Comprueba las funcionalidades tras la conversión ☐ - Hecho	<ul style="list-style-type: none"> Fíjate en que el icono ha cambiado a un icono de branch 

³⁹ <http://aka.ms/treasure35>

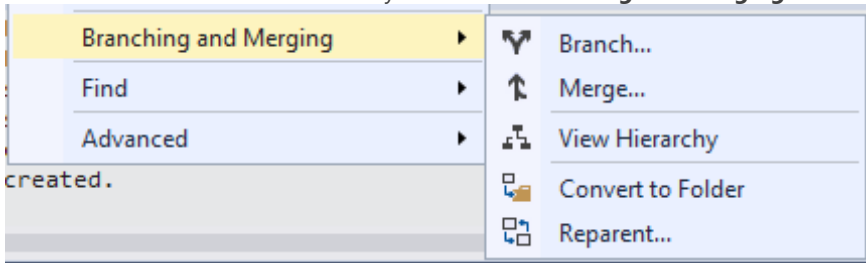
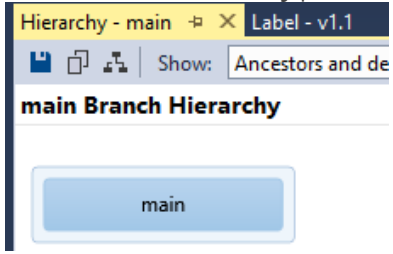
Paso	Instrucciones
	<ul style="list-style-type: none"> Haz clic derecho en el branch main y selecciona Branching and Merging  Fíjate que la opción Convert To Branch ha desaparecido, pero han aparecido las opciones View Hierarchy y Reparent Selecciona View Hierarchy para ver el nodo simple (main only) 

Tabla 11 – Lab 2, Tarea 1

REVISIÓN

Hemos explorado la estrategia de branching Main Only y las implicaciones de promover el directorio principal a un branch. Hemos trabajado sólo en un lugar, el directorio | branch principal y hemos usado labels para marcar las versiones de nuestro código.

En este ejercicio hemos creado:

- 0 branches
- 0 merges
- 2 labels

Ejercicio 3: Development Isolation... bienvenido al branching

META

Explorar la estrategia Development Isolation, que introduce uno o más branches de desarrollo a partir del principal, permitiendo el desarrollo concurrente para la próxima release, experimentación o corrección de bugs de manera **aislada** en cada branch de desarrollo.

Contexto

Tu equipo necesita mantener un branch principal estable para la próxima mayor release en producción pero es necesario el desarrollo concurrente para añadir la funcionalidad de multiplicación a la solución de la calculadora.

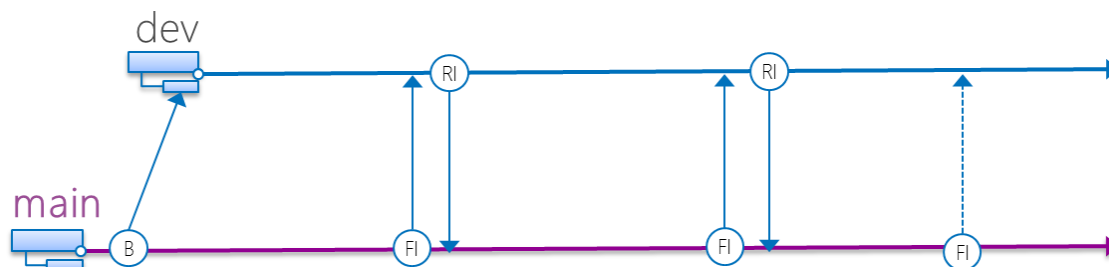
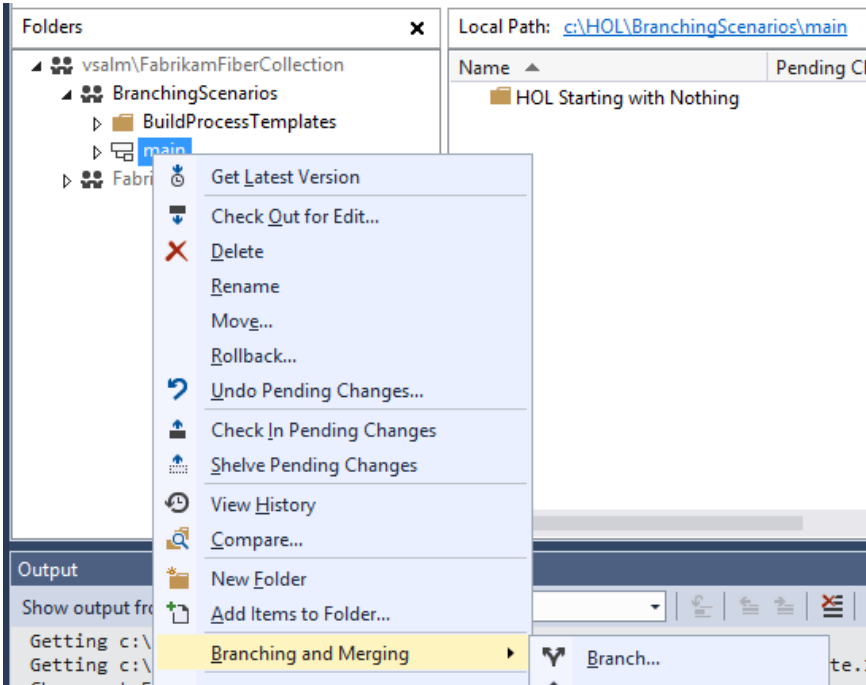
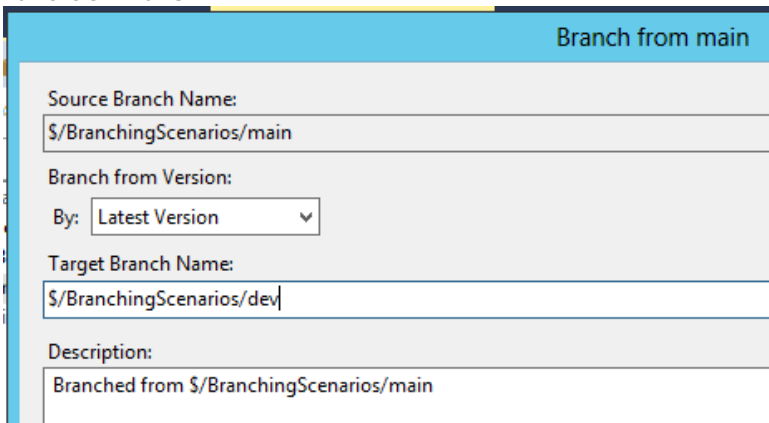
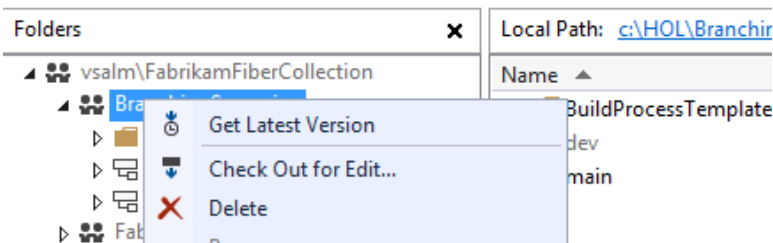

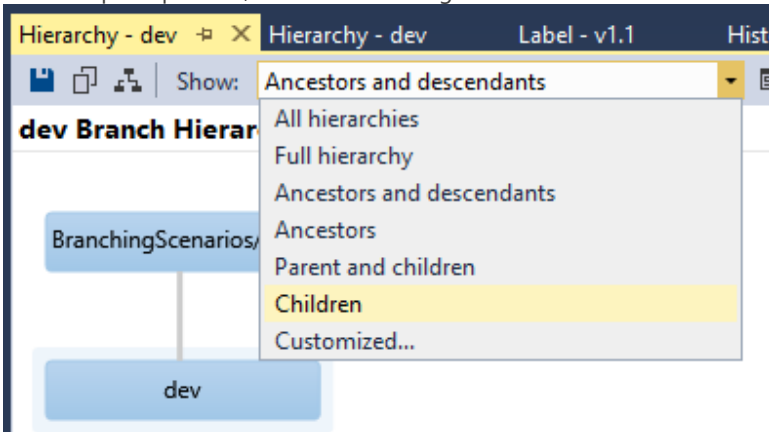
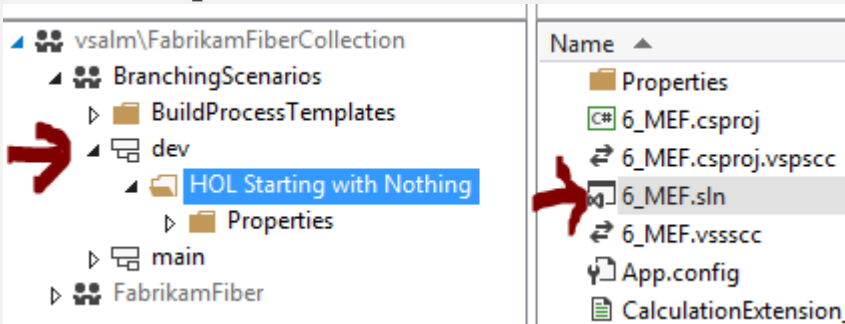


Figura 16 – Estrategia Development Isolation

Tarea 1: Crea el branch de desarrollo para aislar el nuevo desarrollo

Paso	Instrucciones
1 Crea el branch de desarrollo ☐ - Hecho	<ul style="list-style-type: none"> Haz clic derecho en main en el Source Control Explorer, selecciona Branching and Merging, Branch  Dale el nombre al Target Branch Name siguiente <code>\$/BranchingScenarios/dev</code>, como en la imagen, y haz clic en Branch  Puedes usar cualquier nombre para el branch dev, por ejemplo, dev, research, o desarrollo, pero mantén siempre una consistencia para evitar confusiones y errores Haz clic derecho en el directorio BranchingScenarios en el Source Control Explorer y selecciona Get Latest Version
	<p>NOTA En el mundo real, debes considerar crear un segundo workspace para el branch dev para aislarlo del principal.</p>

Paso	Instrucciones
	 <ul style="list-style-type: none"> Haz clic derecho en el branch dev en el Source Control Explorer, selecciona Branching and Merging y View Hierarchy para tener una visualización de los branches main y dev  <ul style="list-style-type: none"> Si el tiempo lo permite, mira los otros diagramas 
2 Extiende la funcionalidad □ - Hecho	<ul style="list-style-type: none"> Abre la solución 6_MEF.sln en el branch dev  <ul style="list-style-type: none"> Añade la clase CalculationExtension_Mul y la lógica de la división escribiendo el código o haciendo copiar-pegar editando la clase *_Add

Estrategias de Branching – Hands-on Lab (HOL) – De la simplicidad a la complejidad, ¿o no?

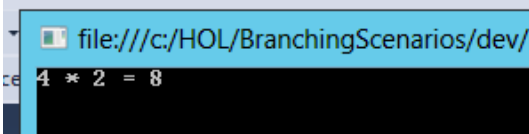
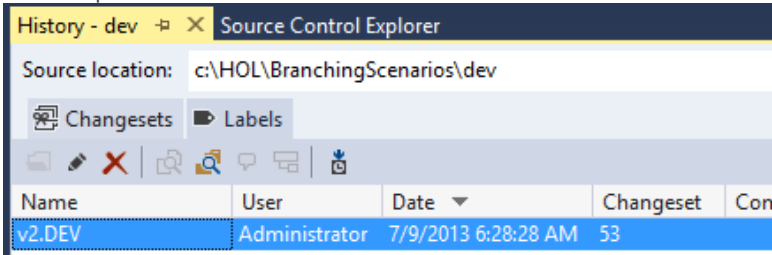
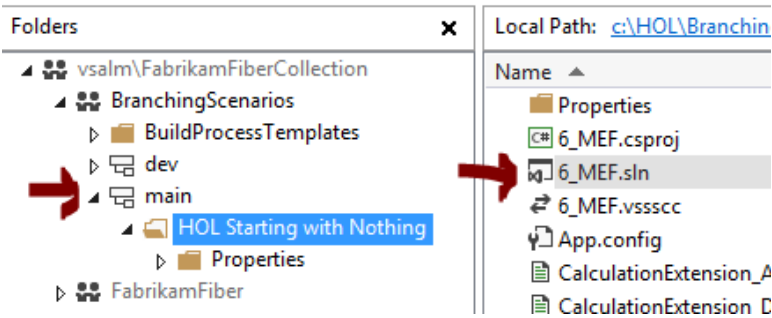
Paso	Instrucciones
	<ul style="list-style-type: none"> Debes definir el símbolo ExportMetadata, el nombre de la clase y la multiplicación <pre> [Export(typeof(ICalculation))] [ExportMetadata("Symbol", '*')] public class CalculationExtension_Mul : ICalculation { public long Calculation(long valueOne, long valueTwo) { return valueOne * valueTwo; } } </pre>
<p>3</p> <p>Testa la nueva funcionalidad</p> <p>☐ - Hecho</p>	<ul style="list-style-type: none"> Haz clic derecho en el proyecto 6_MEF en el Solution Explorer y selecciona Properties Selecciona la pestaña Debug y define los siguientes argumentos para la línea de comandos: 4 2 * Compila y ejecuta (F5) la solución Comprueba que la funcionalidad de la multiplicación funciona como sigue  <ul style="list-style-type: none"> Haz checkin de los cambios en el branch dev Añade un label v2.DEV a esta nueva funcionalidad como una marca para indicar que las pruebas se han completado  <ul style="list-style-type: none"> Ya hemos completado la funcionalidad y estamos listos para integrar los cambios de dev en el branch main

Tabla 12 – Lab 3, Tarea 1

Tarea 2: Corrige un bug en Main

Antes de poder hacer un merge y liberar las nuevas funcionalidades, tenemos que investigar y corregir un bug en producción que tiene una prioridad muy alta.

Paso	Instrucciones
<p>1</p> <p>Investiga el bug</p> <p>☐ - Hecho</p>	<ul style="list-style-type: none"> Los usuarios han reportado que la operación de resta de la versión v1.1 no funciona. Cambiamos de contexto y nos vamos al branch de main, que contiene el código de producción Abrimos la solución 6_MEF.sln del branch dev  <ul style="list-style-type: none"> Hacemos clic derecho en el proyecto 6_MEF en el Solution Explorer y seleccionamos Properties.

Estrategias de Branching – Hands-on Lab (HOL) – De la simplicidad a la complejidad, ¿o no?

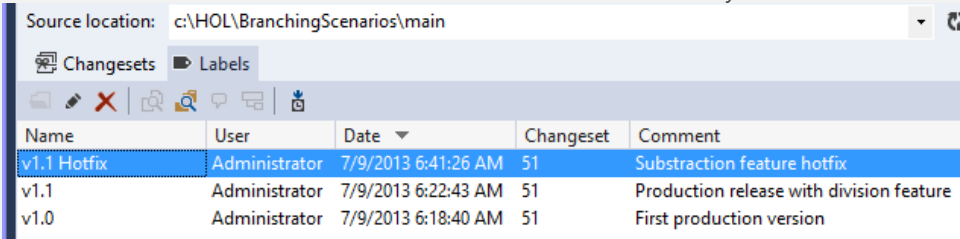
Paso	Instrucciones
	<ul style="list-style-type: none"> Seleccionamos la pestaña Debug y definimos los siguientes argumentos de la línea de comandos: 4 2 -  Compilamos y ejecutamos (F5) la solución Fijaos que el usuario tiene razón, la resta parece que está sumando los valores 
2 Corrige el bug ☐ - Hecho	<ul style="list-style-type: none"> Vamos a modificar el archivo ...main\CalculationExtension_Sub.cs y encondrad el problema <pre>public class CalculationExtension_Sub : ICalculation { 4 references Administrator 1 change public long Calculation(long valueOne, long valueTwo) { // Intentional bug for HOL should be - return valueOne + valueTwo; } }</pre> Cambiamos el + a un – para corregir el bug y si queremos podemos borrar el comentario <pre>public class CalculationExtension_Sub : ICalculation { 4 references Administrator 1 change public long Calculation(long valueOne, long valueTwo) { return valueOne - valueTwo; } }</pre>
3 Valida y testa ☐ - Hecho	<ul style="list-style-type: none"> Compila y ejecuta (F5) la solución Comprueba que la funcionalidad de resta funciona de la siguiente forma 
4 Hacemos Check-in, creamos un label y publicamos la corrección v1.1 ☐ - Hecho	<ul style="list-style-type: none"> Haz checkin de los cambios en el branch main Creamos el label con los últimos cambios en el branch main branch y lo llamamos v1.1 hotfix 

Tabla 13 – Lab 3, Tarea 2

Contexto – Intermedio

Como hemos visto tenemos una versión estable v1.1 Hotfix en el branch principal y una versión estable v2.DEV en el branch de desarrollo en este momento...

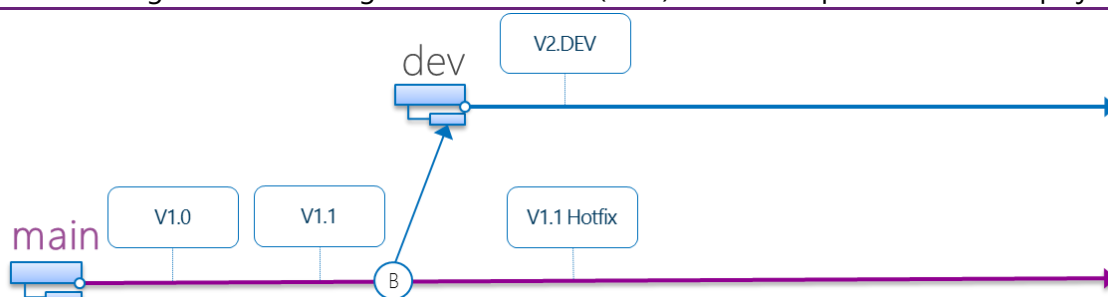
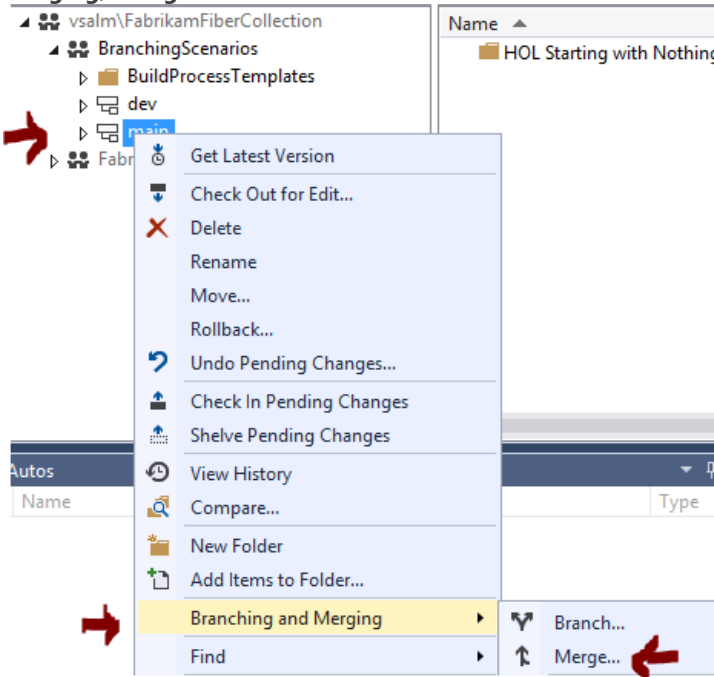
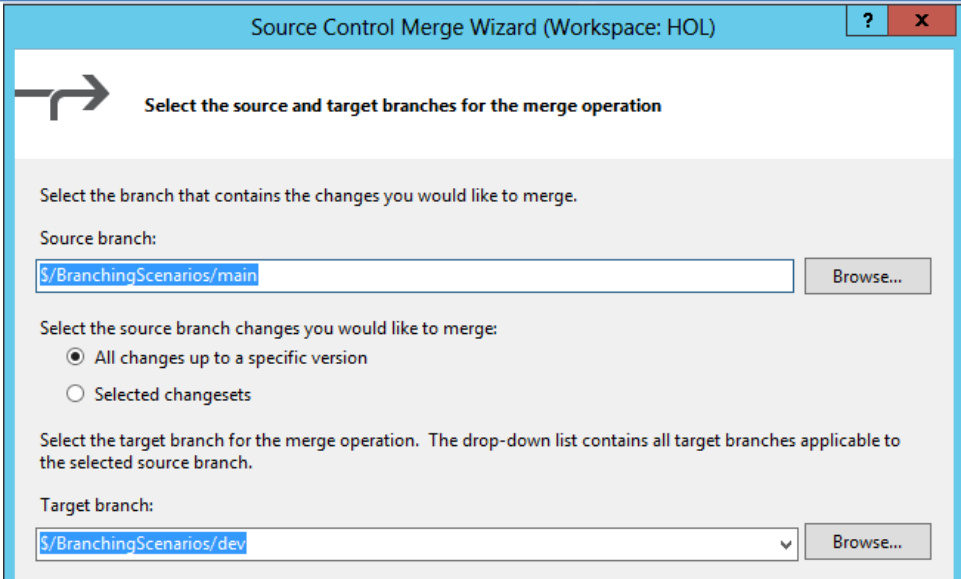
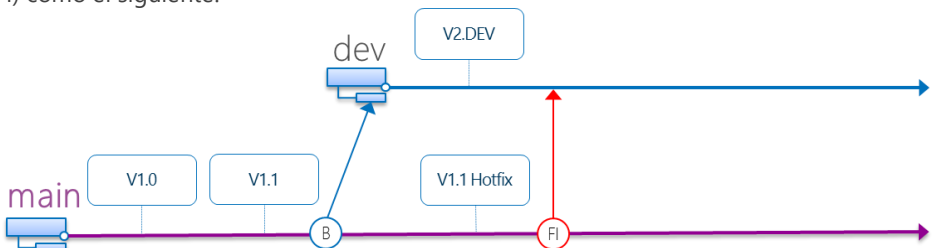
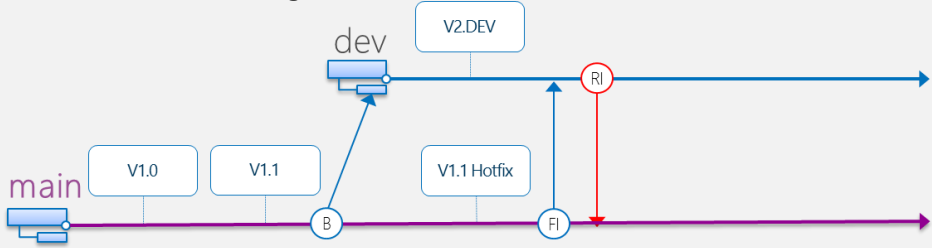


Figura 17 – Revisión del ejercicio de la estrategia Development Isolation

Lo que debemos hacer es compartir la corrección con el equipo de desarrollo e incluir todos los cambios nuevos al branch principal para que podamos tener lo último de lo último, incluyendo las funcionalidades de + - * / para nuestros clientes.

Tarea 3: Incluye todos los cambios y publiquemos una mayor release

Paso	Instrucciones
1 Añade la corrección al branch de desarrollo ☐ - Hecho	<ul style="list-style-type: none"> Haz clic derecho en el branch main en el Source Control Explorer, selecciona Branching and Merging, Merge  Comprueba que en source tenemos <code>\$/BranchingScenarios/main</code> y en target <code>\$/BranchingScenarios/dev</code>

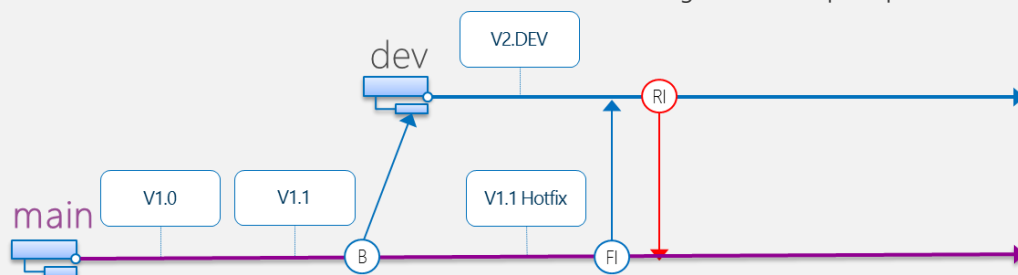
Paso	Instrucciones
	<div data-bbox="402 197 1357 772">  <p>Selecciona Next, selecciona como Version type a Latest Version y selecciona Next</p> <ul style="list-style-type: none"> Lee el resumen de la operación de merge y selecciona Finish para realizar un forward integration (FI) como el siguiente:  </div>
<p>2</p> <p>Haz un merge desde el branch de dev al branch main</p> <p>☐ - Hecho</p>	<ul style="list-style-type: none"> No deberías tener conflictos en esta operación ya que no hemos incluido cambios que puedan generarlos Haz checkin de tus cambios en el branch de dev, compuesto por CalculationExtension_Sub Por seguridad, vuelve a comprobar que las funcionalidades de división y resta siguen funcionando Clic derecho en el branch dev en el Source Control Explorer, selecciona Branching and Merging, y Merge. Comprueba que en source tenemos \$/BranchingScenarios/dev y en target \$/BranchingScenarios/main Selecciona Next, pon el Version type a Latest Version y selecciona Next Lee el resumen de la operación y selecciona Finish para realizar la operación de reverse integration (RI) como se muestra en la figura  <ul style="list-style-type: none"> Haz clic derecho en el branch main en el Source Control Explorer y selecciona Check In Pending Changes, veremos que se incluyen el archivo de proyecto 6_MEF.csproj y el archivo CalculationExtension_Mul.cs
<p>3</p> <p>Valida y testa</p>	<ul style="list-style-type: none"> Abre la solución en el branch principal y configura los argumentos para la línea de comandos como antes y prueba los siguientes cálculos:

Paso	Instrucciones																									
<div><div></div> - Hecho</div>	<div><div><div><div></div></div><div>Test 4 2 +</div></div><div><div></div><div>Test 4 2 -</div></div><div><div></div><div>Test 4 2 *</div></div><div><div></div><div>Test 4 2 /</div></div></div>																									
4 Crea un label y publica la v2 <div><div></div> - Hecho</div>	<div><div><div><div><div></div></div><div>Haz un label para los últimos cambios en el branch principal y llámala V2</div></div><div><div><div>History - main</div><div>Source Control Explorer</div></div><div><div>Source location:</div><div>c:\HOL\BranchingScenarios\main</div></div><div><div>Changesets</div><div>Labels</div></div><div><div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div></div><table><thead><tr><th>Name</th><th>User</th><th>Date</th><th>Changeset</th><th>Comment</th></tr></thead><tbody><tr><td>V2</td><td>Administrator</td><td>7/9/2013 6:39:34 AM</td><td>51</td><td>Feature complete calculator with + - * /</td></tr><tr><td>V1.1 Hotfix</td><td>Administrator</td><td>7/9/2013 6:22:23 AM</td><td>51</td><td>MOve subtraction hotfix to production</td></tr><tr><td>V1.1</td><td>Administrator</td><td>7/9/2013 6:14:43 AM</td><td>51</td><td>Production release with division feature</td></tr><tr><td>v1.0</td><td>Administrator</td><td>7/9/2013 6:13:58 AM</td><td>51</td><td>First production release</td></tr></tbody></table></div></div></div>	Name	User	Date	Changeset	Comment	V2	Administrator	7/9/2013 6:39:34 AM	51	Feature complete calculator with + - * /	V1.1 Hotfix	Administrator	7/9/2013 6:22:23 AM	51	MOve subtraction hotfix to production	V1.1	Administrator	7/9/2013 6:14:43 AM	51	Production release with division feature	v1.0	Administrator	7/9/2013 6:13:58 AM	51	First production release
Name	User	Date	Changeset	Comment																						
V2	Administrator	7/9/2013 6:39:34 AM	51	Feature complete calculator with + - * /																						
V1.1 Hotfix	Administrator	7/9/2013 6:22:23 AM	51	MOve subtraction hotfix to production																						
V1.1	Administrator	7/9/2013 6:14:43 AM	51	Production release with division feature																						
v1.0	Administrator	7/9/2013 6:13:58 AM	51	First production release																						

Tabla 14 – Lab 3, Tarea 3

REVISIÓN

Hemos explorado la estrategia development isolation. Hemos mostrado cómo implementar esta estrategia. Finalmente, hemos explicado por qué deberíamos usar esta estrategia. Hemos explorado cómo trabajar en un bug y en el desarrollo de nuevas funcionalidades a la vez, haciendo merges al branch principal al final del ejercicio.



En este ejercicio hemos creado:

- 1 branch
- 2 merges, uno FI y otro RI
- 3 labels

Ejercicio 4: Feature Isolation... ¡un especial!

META

Explorar la estrategia Feature Isolation, que incluye uno o más branches desde el principal, permitiendo el desarrollo concurrente en funcionalidades claras, que podremos “mergear” y publicar cuando sea necesario.

Contexto

Como antes, necesitamos aislamiento, tenemos que desarrollar las funcionalidades claras de Mod (Módulo) y Power (Potencia), con la flexibilidad necesaria para publicar estas funcionalidades a producción. Desarrollarlo en otro branch de desarrollo o en un branch con el nombre de la funcionalidad es cuestión de gustos, sin embargo nosotros lo implementaremos con un modelo híbrido para este ejercicio de la siguiente forma:

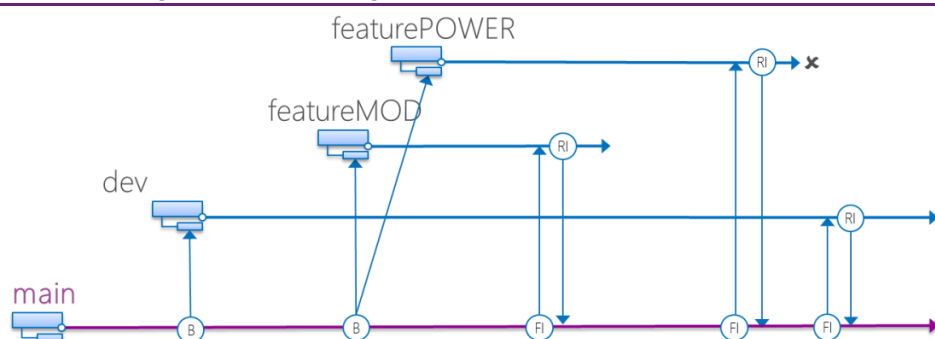


Figura 18 – Estrategia híbrida de Development and Feature Isolation Ejemplo 1

Una estrategia diferente podría ser hacer un branch de dev y enlazar las funcionalidades y su implementación al branch de desarrollo. Esta estrategia usará el branch de desarrollo como branch de integración, haciendo merge de todas las funcionalidades en el branch padre.

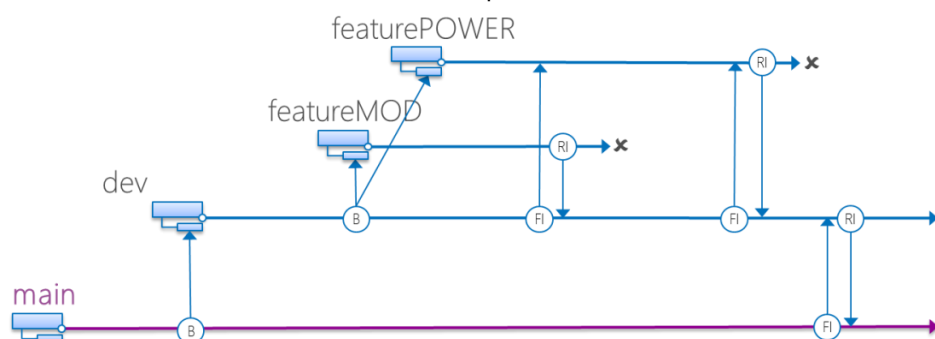


Figura 19 – Estrategia híbrida Development y Feature Isolation Ejemplo 2

Tarea 1: Crea los branch de funcionalidad

Paso	Instrucciones
1 Crea el branch para la funcionalidad POWER <input type="checkbox"/> - Hecho	<ul style="list-style-type: none"> Haz clic derecho en el branch main en el Source Control Explorer, selecciona Branching and Merging, Branch Define como target <code>\$/BranchingScenarios/featurePOWER</code> <div data-bbox="406 1270 1263 1711"> <p>Branch from main</p> <p>Source Branch Name: \$/BranchingScenarios/main</p> <p>Branch from Version: By: Latest Version</p> <p>Target Branch Name: \$/BranchingScenarios/featurePOWER</p> <p>Description: Branched from \$/BranchingScenarios/main</p> </div> <div data-bbox="357 1743 1458 1854"> <p>NOTA Considera el uso de directorios para organizar los branches cuando trabajes con branches de funcionalidad, por ejemplo <code>\$/BranchingScenarios/features/featurePOWER</code></p> </div> <ul style="list-style-type: none"> Selecciona Branch y confirma el diálogo "Continue to branch?" haciendo clic en Yes.

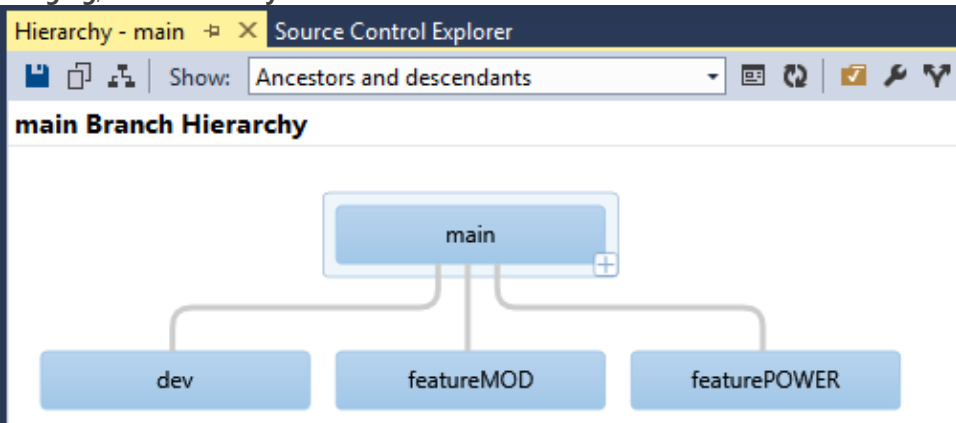
Paso	Instrucciones
2 Crea el branch para la funcionalidad MOD ☐ - Hecho	<ul style="list-style-type: none"> Haz clic en el branch main en el Source Control Explorer, selecciona Branching and Merging, Branch Define el target como \$/BranchingScenarios/featureMOD o \$/BranchingScenarios/features/featureMOD si quieres organizar los branches de funcionalidad. Selecciona Branch y confirma el diálogo "Continue to branch?" haciendo clic en Yes.
3 Obtén la última versión y mira la jerarquía ☐ - Hecho	<ul style="list-style-type: none"> Haz clic derecho en el directorio BranchingScenarios en el Source Control Explorer, selecciona Get Latest Version Haz clic derecho en el branch main en el Source Control Explorer, selecciona Branching and Merging, View Hierarchy  <pre> graph TD main[main] --> dev[dev] main --> featureMOD[featureMOD] main --> featurePOWER[featurePOWER] </pre> <ul style="list-style-type: none"> La jerarquía debe ser como el de la Figura 18 - Estrategia híbrida de Development and Feature Isolation Ejemplo 1 en la página 57

Tabla 15 – Lab 4, Tarea 1

Tarea 2: Crea la funcionalidad MOD

Paso	Instrucciones
1 Implementa la funcionalidad ☐ - Hecho	<ul style="list-style-type: none"> Abre la solución 6_MEF.sln en el branch featureMOD Añade la clase CalculationExtension_Mod y la lógica para la operación Mod escribiendo la clase o copiando, pegando y editando la clase *_Add class Debes definir el símbolo ExportMetadata, el nombre de la clase y la operación Mod <pre> [Export(typeof(ICalculation))] [ExportMetadata("Symbol", '%')] 0 references public class CalculationExtension_Mod : ICalculation { /// <summary> ... 6 references public long Calculation(long valueOne, long valueTwo) { return valueOne % valueTwo; } } </pre>
2 Prueba la funcionalidad ☐ - Hecho	<ul style="list-style-type: none"> Haz clic derecho en el proyecto 6_MEF en el Solution Explorer y selecciona Properties Selecciona la pestaña Debug y define los siguientes argumentos para la línea de comandos: 5 2 % Compila y ejecuta (F5) la solución Comprueba que la funcionalidad Mod funciona así

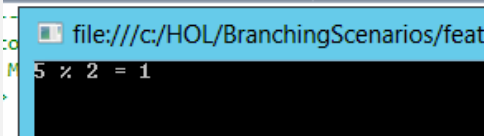
Paso	Instrucciones
	
3 Haz checkin ☐ - Hecho	<ul style="list-style-type: none"> Haz checkin de tus cambios en el branch featureMOD

Tabla 16 – Lab 4, Tarea 2

Tarea 3: Crea la funcionalidad POWER

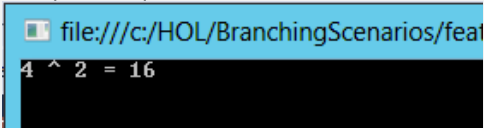
Paso	Instrucciones
1 Implementa la funcionalidad ☐ - Hecho	<ul style="list-style-type: none"> Abre la solución 6_MEF.sln en el branch featurePOWER Añade la clase CalculationExtension_Pow y la lógica para la operación Power escribiendo la clase o copiando, pegando y editando la clase *_Add class Debes definir el símbolo ExportMetadata, el nombre de la clase y la operación Power <pre> [Export(typeof(ICalculation))] [ExportMetadata("Symbol", '^')] 0 references public class CalculationExtension_Pow : ICalculation { /// <summary> ... 6 references public long Calculation(long valueOne, long valueTwo) { long valueResult = 0; if (0 != valueTwo) { valueResult = valueOne; for (long i = --valueTwo; 0 != valueTwo; valueTwo--) { valueResult *= valueResult; } } return valueResult; } } </pre> <p>El código anterior no es el más óptimo y si quieres puedes mejorarlo ☺</p>
2 Prueba la funcionalidad ☐ - Hecho	<ul style="list-style-type: none"> Haz clic derecho en el proyecto 6_MEF en el Solution Explorer y selecciona Properties Selecciona la pestaña Debug y define los siguientes argumentos para la línea de comandos: 4 2 ^ Compila y ejecuta (F5) la solución Comprueba que la funcionalidad Power funciona así 
3 Haz checkin ☐ - Hecho	<ul style="list-style-type: none"> Haz checkin de tus cambios en el branch featurePOWER

Tabla 17 – Lab 4, Tarea 3

Contexto – Intermedio

Como hemos visto, tenemos un branch estable **main**, un branch de desarrollo (**dev**) con una actividad desconocida y las dos funcionalidades ya están listas, pero aisladas.

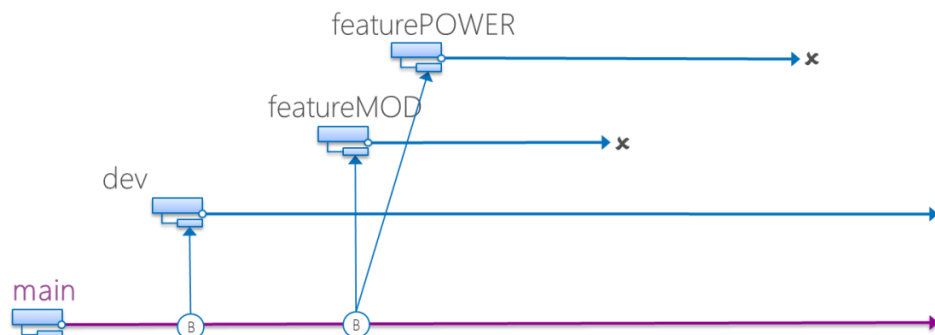

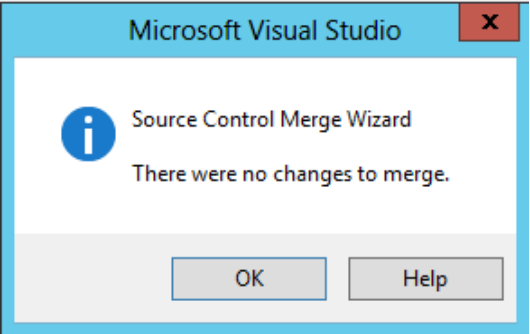


Figura 20 – Revisión del ejercicio de la estrategia Feature Isolation

Lo que debemos hacer es asegurarnos de que los branches de funcionalidad están actualizadas con cualquier actividad que pueda haber ocurrido durante su período de desarrollo y hacer los merges para que vuelvan al branch principal y publicar v3.

Tarea 4: Haz un merge con todos los cambios y publica una mayor release

Paso	Instrucciones
1 Haz el merge de featureMOD ☐ - Hecho	<ul style="list-style-type: none"> Forward-Integration (FI) merge <ul style="list-style-type: none"> Haz clic derecho en el branch main en el Source Control Explorer, selecciona Branching and Merging, y Merge. Comprueba que en source tenemos <code>\$/BranchingScenarios/main</code> y que en target está <code>\$/BranchingScenarios/featureMOD</code> Selecciona Next, pon el Version type a All changes up to a specific y selecciona Next Selecciona Label, encuentra el label V2 como hemos visto y selecciona Next  <p>Select the versions of the source items</p> <div style="border: 1px solid #ccc; padding: 5px; margin: 10px 0;"> <p>Specify the version up to which source branch changes should be merged. In most cases, changes merged previously will not be merged again.</p> <p>Version type: Label Label: V2@\$/BranchingScenarios ...</p> </div> <ul style="list-style-type: none"> Lee el resumen de la operación de merge y selecciona Finish para realizar el reverse integration (RI) como hemos visto. Como no ha habido cambios en el branch principal, veremos el diálogo "no changes to merge"  <ul style="list-style-type: none"> Reverse-Integration (RI) merge <ul style="list-style-type: none"> Haz clic derecho en el branch featureMOD en el Source Control Explorer, selecciona Branching and Merging, y Merge.

Paso	Instrucciones
	<ul style="list-style-type: none"> Comprueba que en source tenemos <code>\$/BranchingScenarios/featureMOD</code> y en target <code>\$/BranchingScenarios/main</code> Selecciona Next, pon en Version type Latest Version y selecciona Next Lee el resumen de la operación de merge y selecciona Finish para realizar el merge reverse integration (RI) Haz checkin de tus cambios en el branch main <p>Hemos realizado los siguientes merges FI y RI</p>
<p>2</p> <p>Haz merge de featurePOWER</p> <p>☐ - Hecho</p>	<ul style="list-style-type: none"> Igual que antes, pero ahora nos centraremos en la otra funcionalidad Haz un merge Forward-Integration (FI) para obtener los últimos cambios del branch principal, resuelve todos los posibles conflictos en el branch de funcionalidad, y estabiliza antes de hacer el merge con el branch principal. <ul style="list-style-type: none"> Haz clic derecho en el branch main en el Source Control Explorer, selecciona Branching and Merging, y Merge. Comprueba que en source tenemos <code>\$/BranchingScenarios/main</code> y en target <code>\$/BranchingScenarios/featurePOWER</code> Selecciona Next, pon en Version type All changes up to a specific y selecciona Next Selecciona Label, encuentra el label V2 como hemos visto y haz clic en Next <ul style="list-style-type: none"> Lee el resumen de la operación de merge y selecciona Finish para realizar un merge de reverse integration (RI) Como no ha habido cambios en el branch principal, veremos el diálogo "no changes to merge" <ul style="list-style-type: none"> Reverse-Integration (RI) merge <ul style="list-style-type: none"> Haz clic en el branch featurePOWER en el Source Control Explorer, selecciona Branching and Merging, y Merge. Comprueba que en source tenemos <code>\$/BranchingScenarios/featurePOWER</code> y en target <code>\$/BranchingScenarios/main</code>

Estrategias de Branching – Hands-on Lab (HOL) – De la simplicidad a la complejidad, ¿o no?

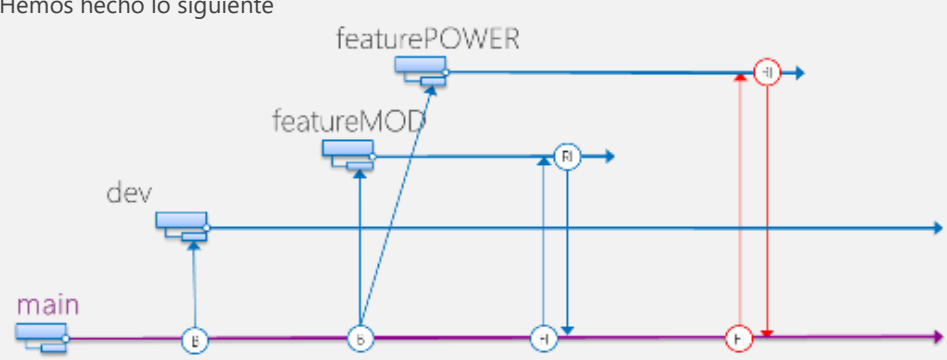
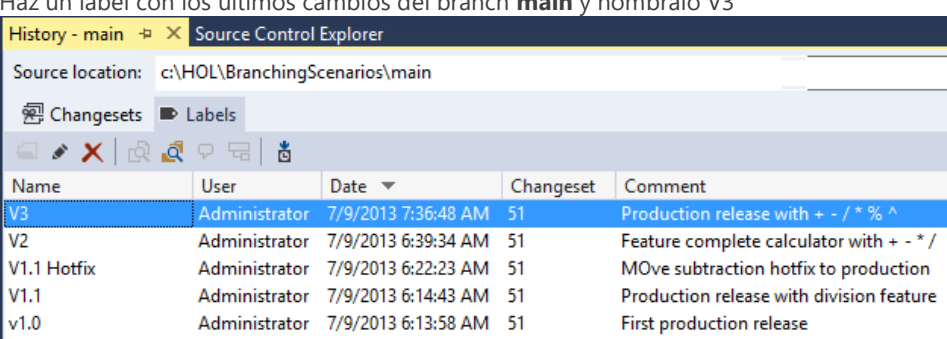
Paso	Instrucciones
	<ul style="list-style-type: none"> o Selecciona Next, y pon en Version type Latest Version y selecciona Next o Lee el resumen de la operación de merge y selecciona Finish para realizar el reverse integration (RI) • Hemos hecho lo siguiente 
3 Comprueba y publica ☐ - Hecho	<ul style="list-style-type: none"> • Abre la solución del branch principal y configura los argumentos de la línea de comandos como antes, y verifica los siguientes cálculos: <ul style="list-style-type: none"> o Test 4 2 + o Test 4 2 - o Test 4 2 * o Test 4 2 / o Test 5 2 % o Test 4 2 ^ • Haz un label con los últimos cambios del branch main y nómbralo V3 

Tabla 18 – Lab 4, Tarea 4

Tarea 5: De manera opcional (no recomendado) borra los branch de funcionalidades

AVISO

No nos hemos vuelto locos con esta tarea opcional, como en la mayoría de los equipos queremos auditar y trazar. Borrar o peor aún, destruir todo va en contra del sentido del control de versiones.

Borramos el branch de funcionalidad, el directorio y los artefactos asociados, manteniendo el histórico en el control de versiones para auditorías y por motivos de seguridad. Si te sientes bien borrando permanentemente los artefactos, puedes usar la línea de comandos **tf** para **destruirlos** permanentemente. Cuidado, esta operación no puede deshacerse.

tf destroy → Destruye, o borra permanentemente los archivos del control de versiones de Team Foundation.

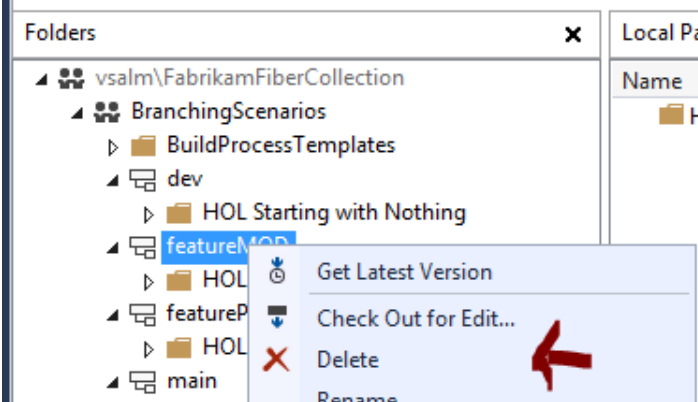
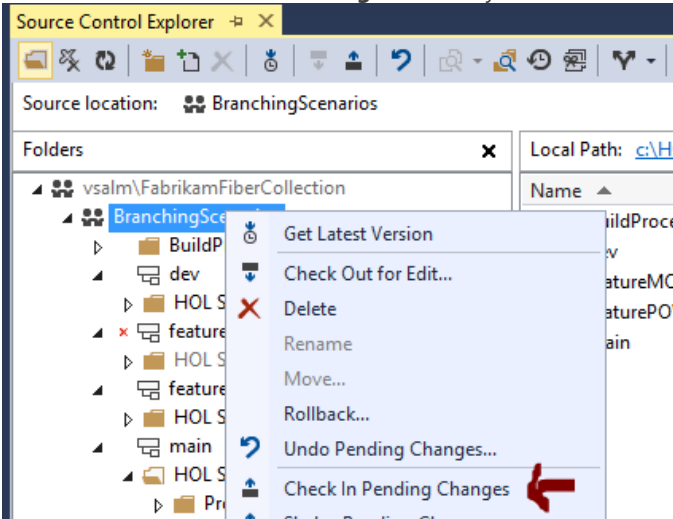
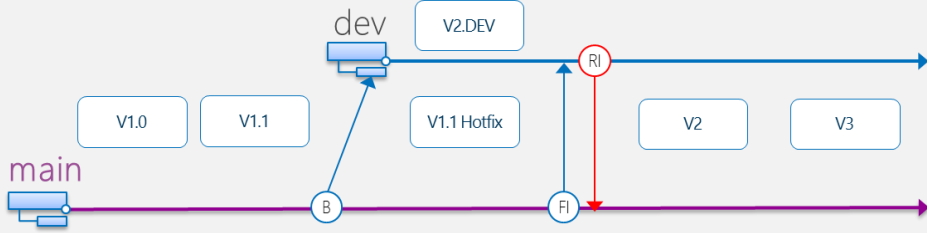
Paso	Instrucciones
1 Borra el branch featureMOD ☐ - Hecho	<ul style="list-style-type: none"> Haz clic en el branch featureMOD en el Source Control Explorer y selecciona Delete  Haz clic en el directorio BranchingScenarios y selecciona Check in Pending Changes  Selecciona Check In, así borraremos el branch featureMOD.
2 Borra el branch featurePOWER ☐ - Hecho	<ul style="list-style-type: none"> Haz clic en el branch featurePOWER en el Source Control Explorer y selecciona Delete Haz clic en el directorio BranchingScenarios y selecciona Check in Pending Changes Selecciona Check In, así borraremos el branch featurePOWER.

Tabla 19 – Lab 4, Tarea 5

REVISIÓN

Hemos explorado la estrategia feature isolation. Hemos demostrado cómo implementar esta estrategia, y por qué usarla. Hemos implementado dos funcionalidades claras y aisladas y hemos publicado v3



En este ejercicio hemos creado:

- 2 branches
- 4 merges, dos FI's y dos RI's
- 1 label

Ejercicio 5: Release Isolation... alarma de auditoría

META

Veremos la estrategia Release Isolation, que introduce uno o varios branches desde el principal, permitiendo la gestión concurrente de releases. Técnicamente hablando estamos viendo una introducción a la estrategia Development & Release isolation en este tutorial, pero veremos sólo la parte de release en este ejercicio.

Contexto

La administración ha notificado al equipo que la organización necesita dar soporte a las releases v2, v3 y a futuras releases a la vez y por lo que el equipo necesita un snapshot de los fuentes de cada release.

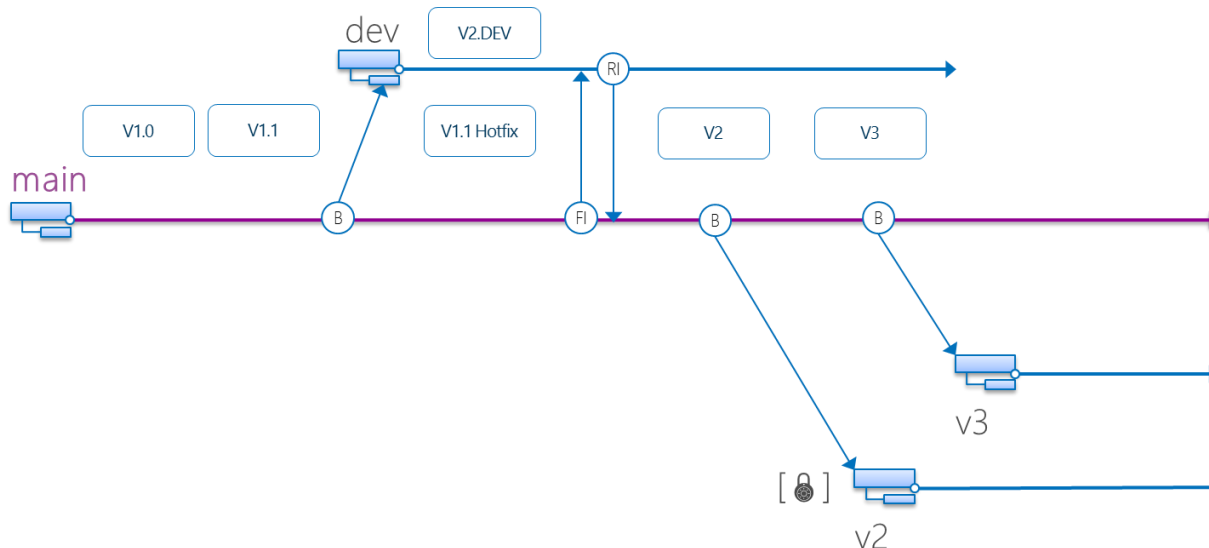
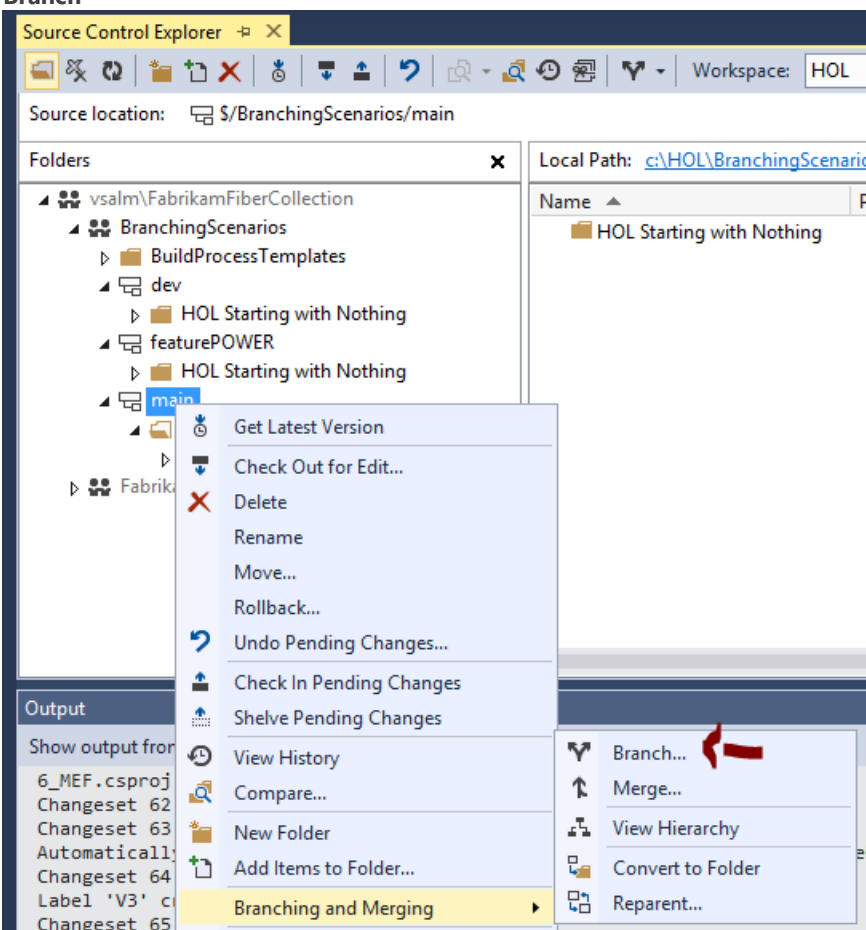
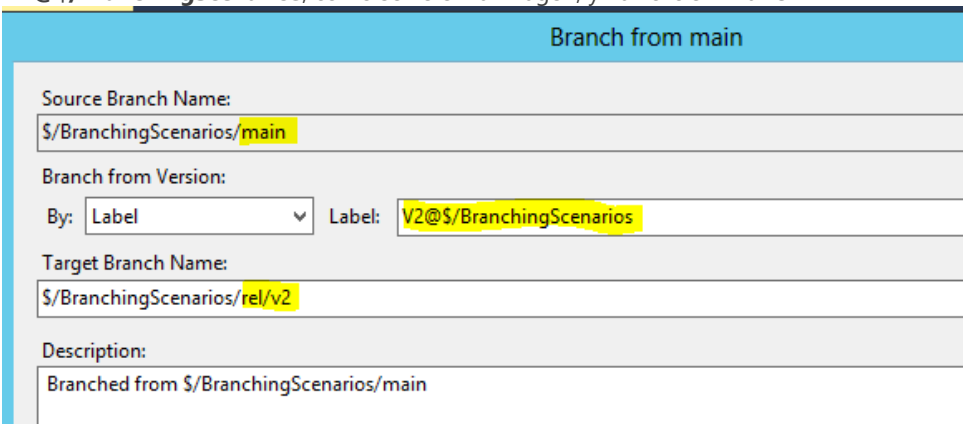


Figura 21 – Release Isolation

Tarea 1: Crea los branches de release

Paso	Instrucciones
1 Release v2 ☐ - Hecho	<ul style="list-style-type: none"> Haz clic en el branch main en el Source Control Explorer, selecciona Branching and Merging, Branch  Usa como Target Branch Name <code>\$/BranchingScenarios/rel/v2</code>, haz un branch a partir del label V2@\$/BranchingScenarios, como se ve en la imagen, y haz clic en Branch  Haz clic en el directorio BranchingScenarios en el Source Control Explorer y selecciona Get Latest Version
2 Release v3 ☐ - Hecho	<ul style="list-style-type: none"> Haz clic derecho en el branch main en el Source Control Explorer, selecciona Branching and Merging, Branch Usa como Target Branch Name <code>\$/BranchingScenarios/rel/v3</code> y haz clic en Branch.

Estrategias de Branching – Hands-on Lab (HOL) – De la simplicidad a la complejidad, ¿o no?

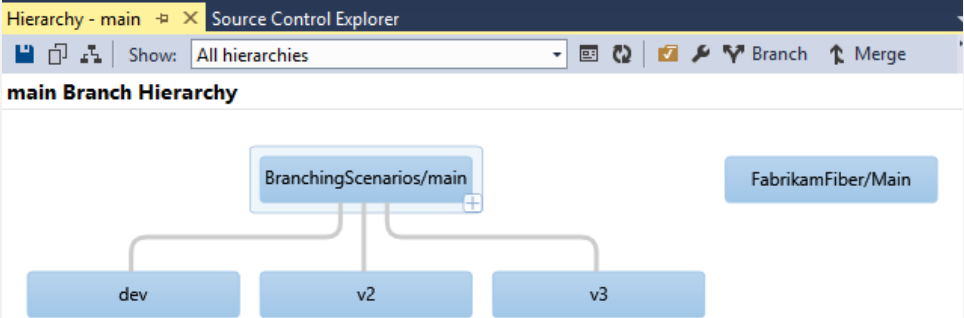
Paso	Instrucciones
	<ul style="list-style-type: none"> Haz clic derecho en el directorio BranchingScenarios en el Source Control Explorer y selecciona Get Latest Version.
3 Recapitulemos ☐ - Hecho	<ul style="list-style-type: none"> Puede parecer que hemos creado dos branches desde el mismo punto en el tiempo. Recuerda que creamos v2 desde un label y después v3 desde la última versión. Esperemos que esto ayude a clarificar cómo se diferencian estos branches y porqué son importantes los labels.
4 Veamos la jerarquía ☐ - Hecho	<ul style="list-style-type: none"> Haz clic derecho en el branch main en el Source Control Explorer, selecciona Branching and Merging y View Hierarchy para tener una visualización de todos los branches 

Tabla 20 – Lab 5, Tarea 1

¿Qué significa el símbolo de bloqueo?

Ahora que hemos aislado los branches que representan las diferentes releases podemos bloquearlas poniéndolas como sólo lectura. El símbolo de bloqueo que vemos en algunos diagramas de la guía indica esas restricciones de sólo lectura. TFS aún no visualiza el estado de bloqueo en los modelos de jerarquía de branches.

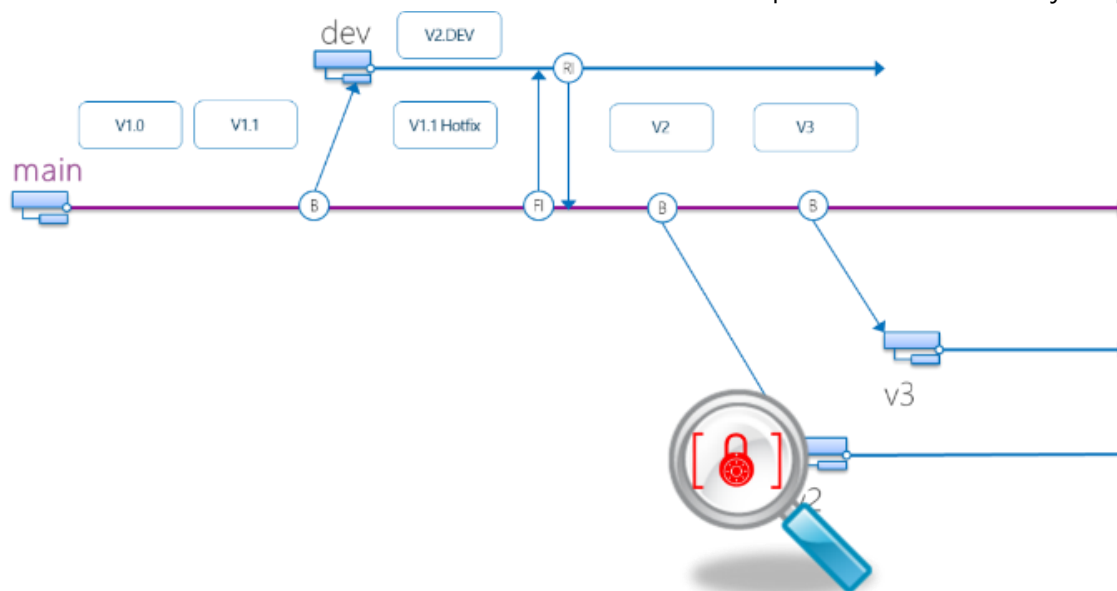


Figura 22 – Release Isolation: Bloqueando

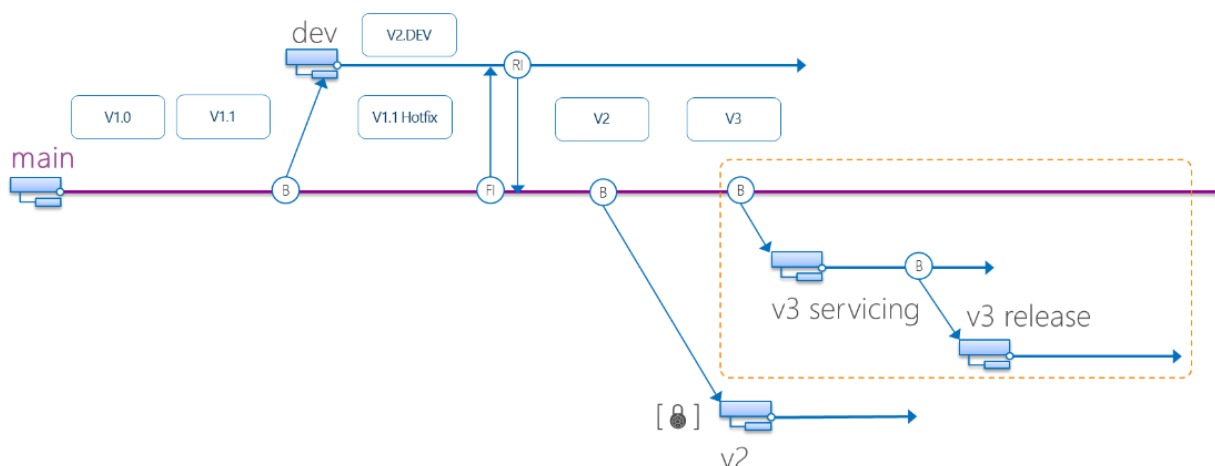
REVISIÓN

Hemos explorado la estrategia de release isolation. Hemos visto cómo implementar esta estrategia. Finalmente hemos explicado porqué usar la estrategia de release isolation.

No hemos resaltado el hecho de poder securizar o bloquear los branch de release. Leed las discusiones de la guía sobre este tema tan caliente ☺

Veremos la estrategia de Servicing y Release Isolation que añade branches de servicing, que permite la gestión de bugs service packs. Esta estrategia es una evolución de la de Release Isolation, o más exactamente de Development and Release Isolation.

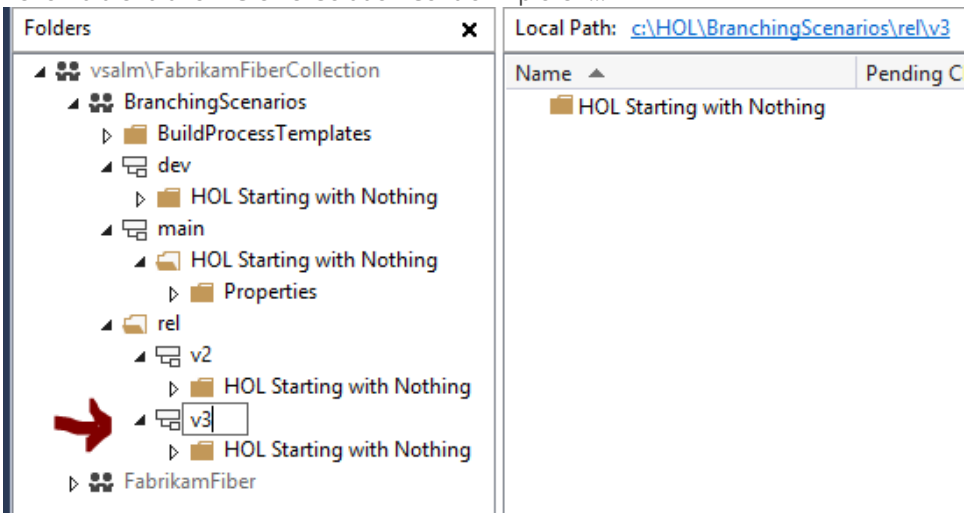
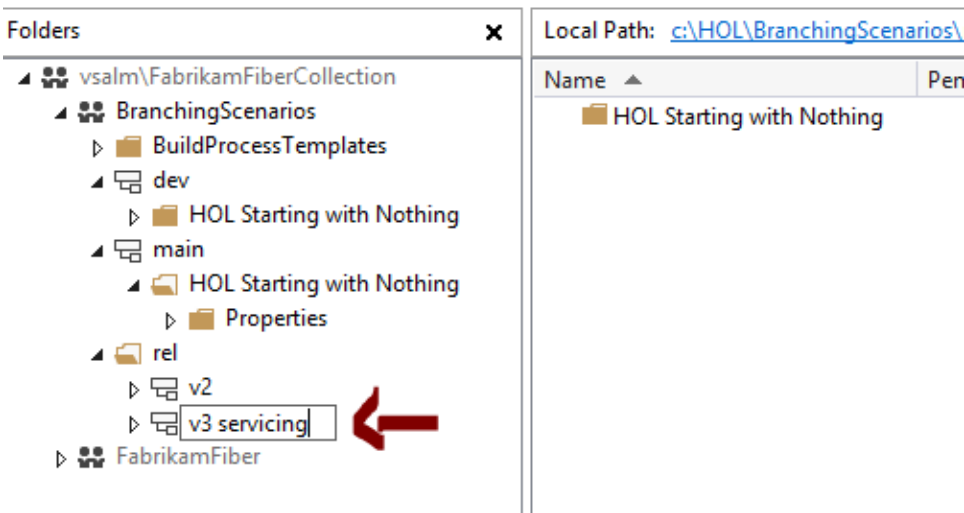
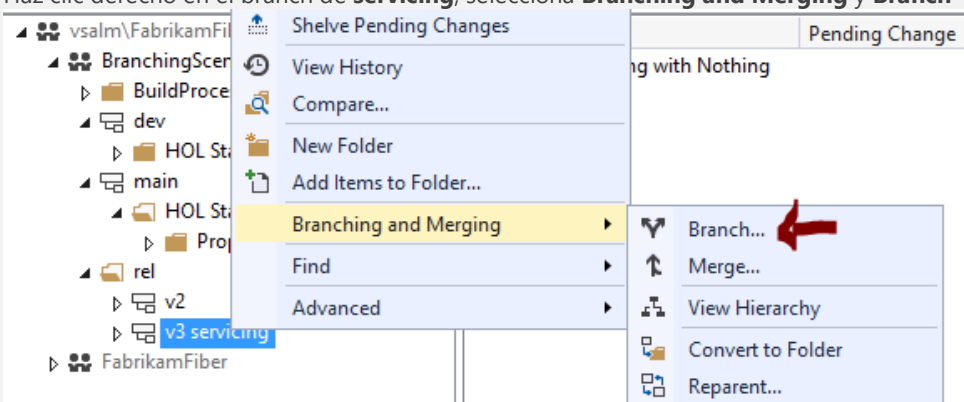
El equipo ha aprendido que habilitar el modelo de servicing permite que los clientes se actualicen a la **próxima major release**, por ejemplo, v1 → v2 y, además, este modelo soporta **service packs adicionales** por release, a partir de v3.



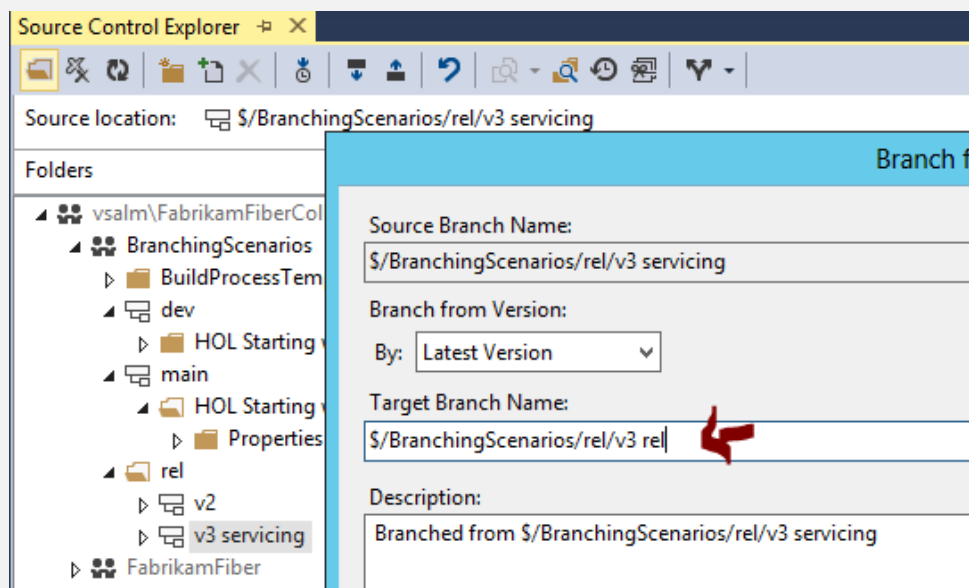
Vamos a usar una convención de nombres arbitraria para los branches de este ejercicio, como hemos hecho antes. Debes invertir tiempo en definir una convención de nombres adecuada para tu organización y entorno antes de realizar cualquier acción con los branches. Renombrar branches no es típico y como alternativa, puedes planificar e introducir branch de servicing con v4 en lugar de inyectarla en la jerarquía v3.

Renombrar branches es técnicamente posible, pero puede introducir problemas de trazabilidad y migración en el futuro si no se hace de manera consistente. Este ejercicio corto y simple muestra lo fácil que es añadir branches rápidamente, lo cual **aumentará considerablemente los costes de mantenimiento y de merge**.

Tarea 1: Introduce un branch de servicing.

Paso	Instrucciones
1 Renombra el branch de release a servicing	<ul style="list-style-type: none"> Renombra el branch v3 en el Solution Control Explorer ...  <p>Local Path: c:\HOL\BranchingScenarios\rel\v3</p> <p>Name Pending C</p> <p>HOL Starting with Nothing</p> <p>... a v3 servicing</p> <p>Source Location: c:\HOL\BranchingScenarios\rel\v3</p>  <p>Local Path: c:\HOL\BranchingScenarios\rel\v3</p> <p>Name Pending C</p> <p>HOL Starting with Nothing</p>
2 Añade un branch de release ☐ - Hecho	<ul style="list-style-type: none"> Haz clic en el branch servicing, selecciona Check In Pending Changes y haz checkin. Haz clic derecho en el branch de servicing, selecciona Branching and Merging y Branch  <p>Local Path: c:\HOL\BranchingScenarios\rel\v3</p> <p>Name Pending Change</p> <p>HOL Starting with Nothing</p>

- Usa como Target Branch Name `$/BranchingScenarios/rel/v3 rel`



- Selecciona **Branch** y **Yes** cuando te pregunte si quieres continuar
- Haz clic derecho en el directorio **BranchingScenarios** y haz un **Get Latest Version**.

3
Veamos la
jerarquía
- Hecho

- Haz clic en el branch **main** en el Solution Control Explorer, selecciona **Branching and Merging**, y **View Hierarchy**

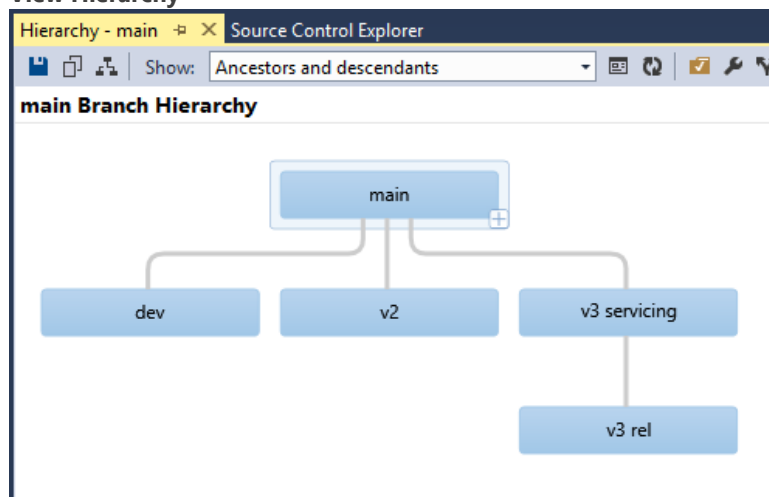


Tabla 21 – Lab 6, Tarea 1

REVISIÓN

Hemos explorado la estrategia servicing and release isolation, cómo implementarla, usado un escenario de branching simple deliberadamente. Recuerda que desaconsejamos el uso de branches para aislar servicing, pero tus requisitos pueden forzarte a desplegar esta estrategia de branching y liberar un service pack.

Conclusión

Aquí concluye nuestra aventura sobre las Estrategias de Branching. Hemos tocado la teoría básica y te hemos introducido en las estrategias y alternativas más comunes al branching. Hemos visto varios ejercicios en los tutoriales y hemos complementado esta guía con un Hands-on Lab y varias guías auxiliares.

En las últimas páginas de la guía, encontrarás una Referencia con los trucos y algún poster. Están disponibles aparte de la guía y puede serte útil imprimirlos y tenerlos disponibles en el área del equipo.

Esperamos que esta guía te sea útil

Atentamente

The Microsoft Visual Studio ALM Rangers



Estrategias de Branching

No hay pociones mágicas ni balas de plata con el branching y el merging. Debes descubrir dependiendo de tus requisitos cuál es la mejor opción para empezar. Este resumen te introduce a algunos de los planes y estrategias que hemos visto en la guía junto a una matriz que te permitirá centrarte en el plan más relevante. Este resumen es un punto de partida – no una guía exhaustiva – de tu decisión para ver qué plan seguir...

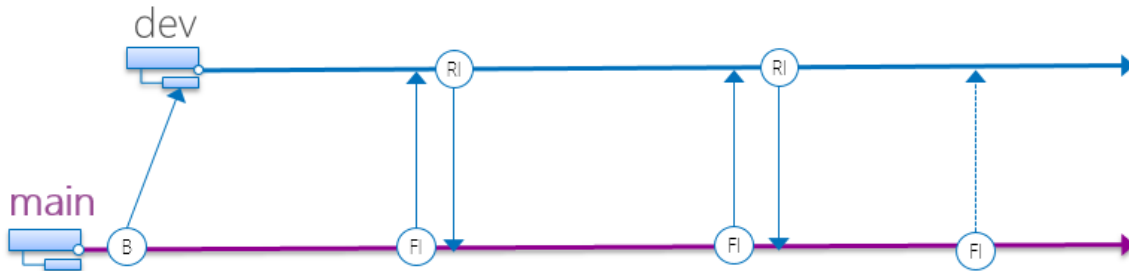
¿Por qué hacer branches?

- Gestionar el trabajo concurrente/paralelo
- Aislar los riesgos de cambios concurrentes
- Hacer snapshots para poder dar soporte
- Compartir recursos
-

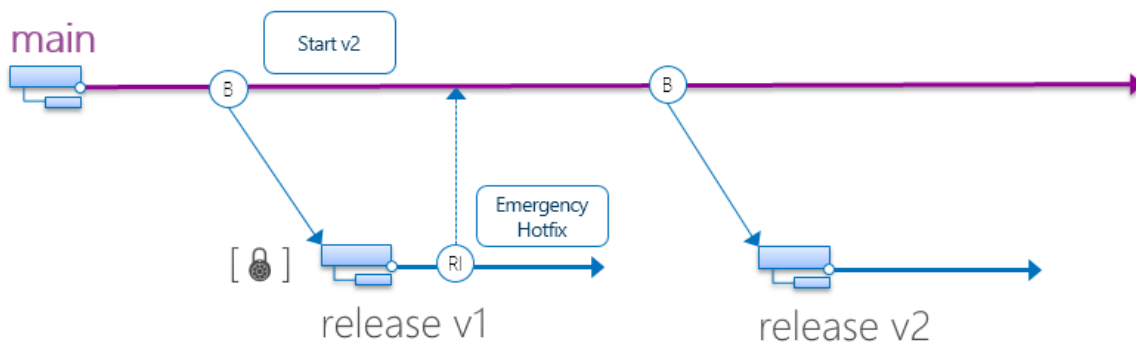
Requisito	Main Only	Development Isolation	Development and Release Isolation	Release Isolation	Servicing and Release Isolation	Servicing, hotfix and Release Isolation	Feature Isolation	Code Promotion	Feature Toggling
Mantenlo simple y minimiza los costes	*	*	*	*			*	*	*
Development isolation		*	*				*		
Feature isolation		*	*				*		*
Release isolation			*	*	*	*			
Ciclos de testing largos			*					*	
Habilitación dinámica de funcionalidades									*
Modelo con una sola release				*	*	*			
Modelo con varias releases				*	*	*			
Modelo con varias release y service packs					*	*			
Modelo con múltiples releases, corrección de bugs y service packs						*			
Hacen falta snapshot de las releases				*	*	*			

Development and Release Isolation Branching

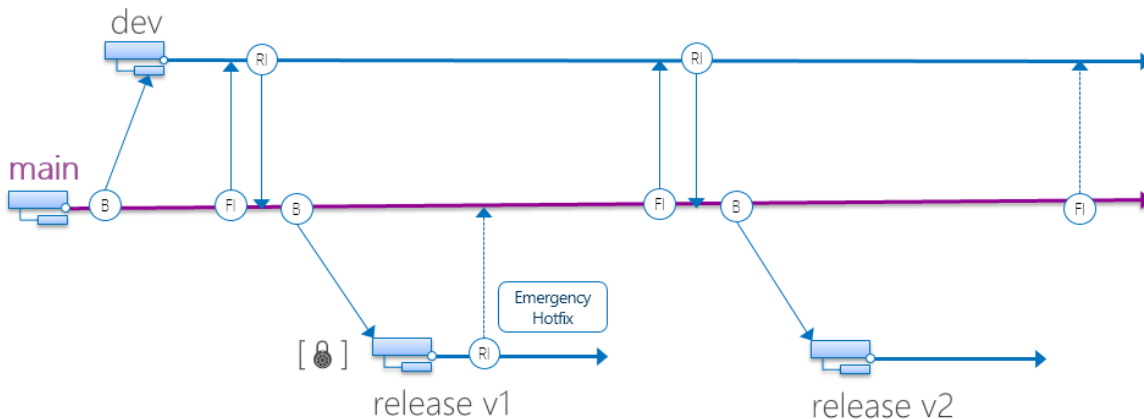
La estrategia **Development Isolation** introduce uno o varios branches de desarrollo a partir de la principal, permite el desarrollo concurrente para la próxima versión, experimentación y corrección de bugs en branches aisladas de desarrollo.



La estrategia **Release Isolation** introduce uno o varios branches de release a partir de la principal, permitiendo la gestión de release concurrente.



La estrategia **Development and Release Isolation** combina las estrategias de Development Isolation y Release Isolation, uniendo los escenarios de uso y las consideraciones de ambas estrategias.



Clave: **(B)** Branch **(FI)** Forward Integrate **(RI)** Reverse Integrate 🔒 read-only

Cuándo considerarla

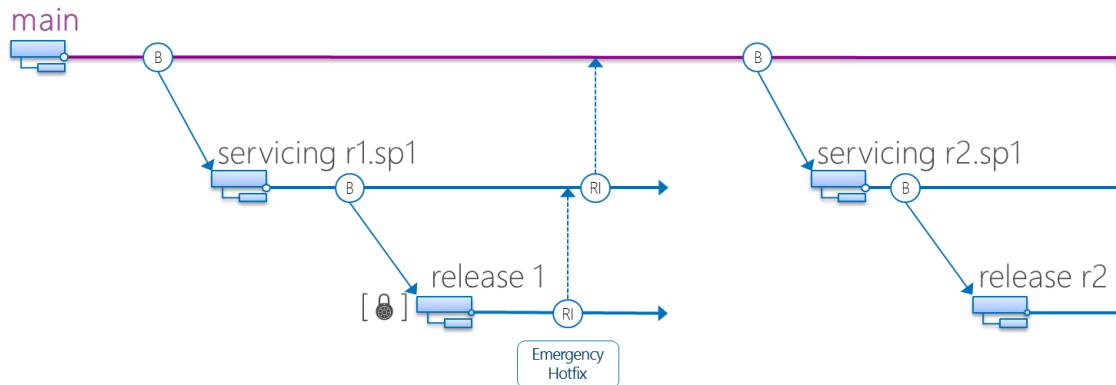
- Desarrollo concurrente y aislado
- Varias major releases
- Requisitos que hay que cumplir

Service and Release Isolation Branching

La estrategia **Servicing and Release** Isolation incluye branches de servicing, lo que permite una gestión concurrente de bugs y service packs. Esta estrategia es una evolución de la Release Isolation, o más técnicamente hablando de la estrategia Development and Release Isolation

Cuándo considerarla

- Release aisladas y concurrentes
- Varios major releases y service packs
- Requisitos que hay que cumplir



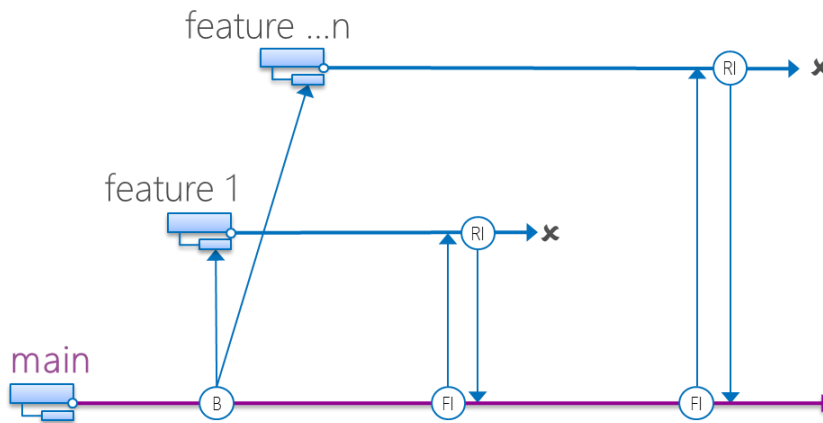
Claves: **(B)** Branch **(FI)** Forward Integrate **(RI)** Reverse Integrate 🔒 read-only

Feature Isolation Branching

En muchos proyectos software, las funcionalidades a implementar pueden estar muy bien definidas. Pueden coexistir varios equipos dedicados que desarrollan varias funcionalidades para un mismo producto. Cuando tengamos este aislamiento tan claro en nuestro proceso podemos usar la estrategia **Feature Isolation**. Haciendo un branch por funcionalidad en un plan razonable de branches para productos grandes que consecuentemente tendrán muchas funcionalidades y varios equipos trabajando en el producto.

Cuándo considerarla

- Desarrollo concurrente de funcionalidades
- Funcionalidades claramente aisladas
- Producto grande con muchas funcionalidades
- Las funcionalidades no son las mismas en cada ciclo de release
- Posibilidad de desechar funcionalidades



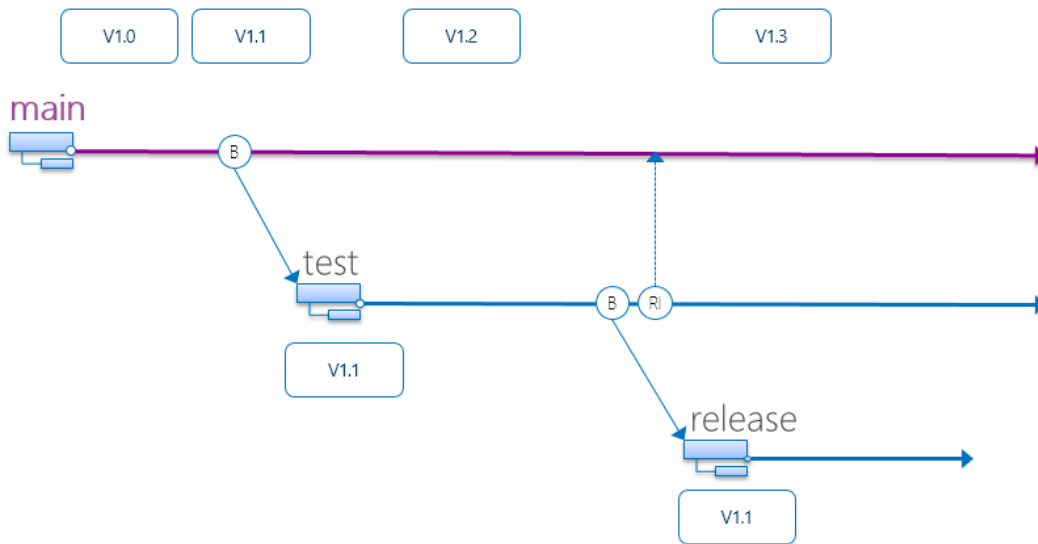
Claves: **(B)** Branch **(FI)** Forward Integrate **(RI)** Reverse Integrate **x** Delete

Code Promotion Branching

El **Code Promotion** es un plan que promueve versiones del código a diferentes niveles a medida que se va volviendo más estable. Otros sistemas de control de versiones han implementado técnicas similares con el uso de propiedades de archivo para indicar en qué nivel se encuentra un archivo. A medida que el código principal se vuelve estable, lo promovemos a un branch para testing.

Cuándo considerarla

- Una sola major release en producción
- Largos ciclos de testing
- Es necesario evitar la congelación del código, pero necesitamos aislamiento para las releases



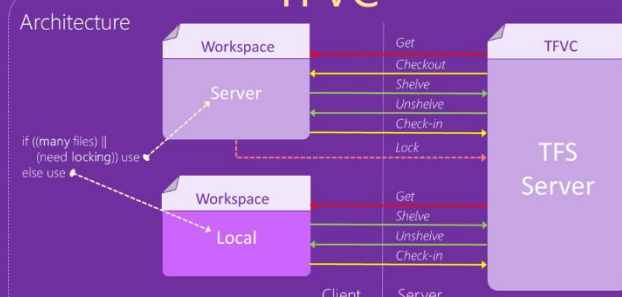
Clave: **(B)** Branch **(RI)** Reverse Integrate

Trucos para el control de versiones en TFVC y Git

Estos trucos están disponibles a parte en JPG y PDF

TFVC

Architecture



Concepts

- Branch** is an isolated copy of item metadata and version control history.
- Changeset** is a logical container for changes of a single check-in.
- Check-in** commits pending changes in workspace to server as a changeset.
- Label** mutable grouping of specific version of a set of source files.
- Shelveset** is a set of pending changes are temporarily saved on the server.
- Workspace** is a local copy of your team's codebase. Create multiple workspaces and switch among them to work on different branches or copies of the codebase.

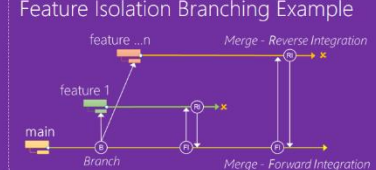
Commands

TF command
Team Foundation Version Control (tfvc) command line tool with a variety of commands.

- everyday**
 - add** adds new files and folders from a local file system location
 - get** retrieves a copy of items from TFVC to the workspace
 - checkin** commits pending changes in current workspace to TFVC
 - checkout** makes local file writable and changes status to "edit"
 - delete** removes files and folders from TFVC and disk
 - history** displays the revision history for files and folders
- branching**
 - branch** creates a copy of items, preserving a relationship to the original items
 - merge** applies changes from one branch into another
- shelving**
 - shelve** stores a set of pending changes in TFVC without a commit
 - unshelve** restores details of a shelveset or view shelvesets belonging to a specific user
 - unshelve** restores shelved changes from TFVC to current workspace
- uncommon**
 - changeset** displays information about a changeset
 - delete** permanently deletes, version-controlled files from TFS. Admin only.
 - lock** locks or unlocks to prevent against checkout of items
 - rename** changes the name or the path of items
 - rollback** reverts the changes of one or more changesets
 - undo** restores items that were previously deleted
 - workspace** creates, deletes, displays, or modifies properties and mappings associated with a workspace
- help**
Type **tf /?** on the command line for a complete list of commands and arguments.

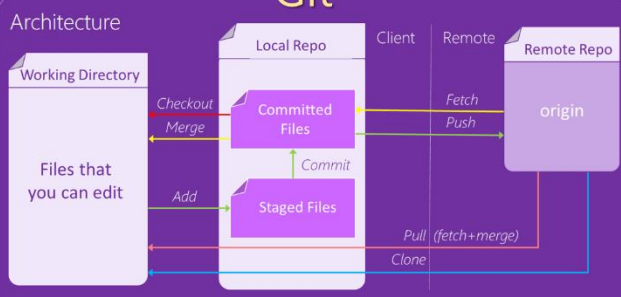
TFSDeleteProject
Command line tool which deletes a team project from a TFS team project collection. It is a non-reversible operation!

Feature Isolation Branching Example



Git


Architecture



Concepts

- Branch** is a named pointer to a commit history in the repository. Used to isolate changes from each other.
- Commit** (noun) is equivalent (mostly) to Changeset, but can also be an action.
- HEAD** is a pointer to the branch your commits will be associated with.
- Stash** is a set of pending changes are temporarily saved in the repository.
- Tag** is a named pointer to a commit in the repository. Useful for marking a point-in-time in your repository.

Feature Isolation Branching Example




Links

- [microsoft.github.io](#) - Windows client downloads
- [git-scm.com](#) - documentation
- [github.com](#) - code host
- [bit.ly/gitsc](#) - Get source control provider VS2008-2012
- [bit.ly/gitee](#) - Get extensions

Commands

git command
git command line tool with a variety of commands.

- everyday**
 - add** adds the current content of existing paths
 - clone** creates a working copy from existing repository
 - commit** (verb) all the staged local changes
 - fetch** fetches the latest changes from origin, not merging
 - pull** fetches the latest changes from origin and merge into working copies
 - push** commits changes to origin
 - rm** removes files from the working tree and from the index
 - status** shows uncommitted changes in the working directory
 - tag** mark a version or milestone
- branching**
 - branch** (noun) creates branch called name based on HEAD
 - branch** (-f) creates branch called name
 - checkout** (-t) switch to the id branch
 - diff** shows changes to tracked files
 - merge** merge two branches
- uncommon**
 - ls-tree** show who changed what and when
 - ls-tree** -r show who changed what and when
 - grep** search working directory
 - log** show history of changes
 - show** (id) show a specific file from a specific id
- help**
Type **git help** on the command line for a complete list of commands and arguments.



Visual Studio ALM Rangers Solutions – <http://aka.ms/vsarsolutions>

2014-03-20 v3

Consideraciones para el control de versiones para TFVC y Git

Estos trucos están disponibles a parte en JPG y PDF

