

Control de versiones con TFS

Parte 4

Git para usuarios de TFVC

Spanish translation
by Juan María Laó Ramos

Visual Studio ALM Rangers

The MIT License (MIT)

Copyright (c) 2015 Microsoft Corporation

Se concede permiso, de forma gratuita, a cualquier persona que obtenga una copia de este software y de los archivos de documentación asociados (el "Software"), para usar el Software sin ninguna restricción, sin ninguna limitación sobre los derechos de uso, copia, modificación, fusión, distribución, sublicenciación, y/o vender copias del Software, y para permitir que las personas que obtengan el Software puedan hacerlo, acogiéndose a las siguientes condiciones:

El aviso de copyright y sus permisos deben incluirse en todas las copias del software.

EL SOFTWARE SE ENTREGA "TAL CUAL", SIN GARANTÍA DE NINGÚN TIPO, EXPLÍCITA O IMPLÍCITA, INCLUYENDO PERO NO LIMITANDO LAS GARANTÍAS DE COMERCIALIZACIÓN, ADECUACIÓN PARA UN OBJETIVO PARTICULAR Y NO INFRACCIÓN. EN NINGÚN CASO, LOS AUTORES NI TITULARES DEL COPYRIGHT SERÁN RESPONSABLES DE NINGUNA RECLAMACIÓN, DAÑOS U OTRA RESPONSABILIDAD, YA SEA EN FORMA DE CONTRATO, AGRAVIO O CUALQUIER OTRA FORMA QUE ESTÉ LIGADA AL SOFTWARE O AL USO U OTRAS TRATOS QUE SE LE DE AL SOFTWARE

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Índice

Prólogo	5
Introducción	6
Sistemas de Control de Versiones Centralizados y Distribuidos	7
TFVC == Sistema de Control de Versiones Centralizado	7
Git == Sistema de Control de Versiones Distribuido	7
Elegir entre TFVC y Git	8
Team project o granularidad de repositorio	9
Flujo de trabajo (lo básico)	11
Branching	11
Haciendo Cambios	13
Revisando cambios con Pull Requests	16
Migrando un repositorio TFVC	18
¿Qué migrar?	18
¿Por qué no?	18
Recursos útiles para la migración	18
Funcionalidades de TFVC en Git	20
Buenas y malas prácticas en Git	21
Enlaces a más información	23
Apéndice	25
Conclusión	26
Posters	27

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

Cada vez más desarrolladores están utilizando control de versiones. Esto significa que los desarrolladores necesitan ahora más que nunca guías sólidas, prácticas, fáciles de entender y probadas por la industria.. 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 han 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 Mitrik – Program Manager, Visual Studio Cloud Services

Introducción

Esta guía intenta ser un punto de partida para aprender [Git](#)¹ y cómo se compara con TFVC desde el punto de vista de alguien que está familiarizado con TFS. Elegir el sistema de control de versiones para un proyecto u organización es una decisión muy importante que afectará al rendimiento del desarrollo, y no debe elegirse a la ligera.

La intención de esta guía es ver las diferencias entre TFVC y Git de manera que pueda decidir qué sistema de control de versiones se ajusta mejor a sus necesidades. También hemos incluido una serie de recomendaciones en forma de buenas y malas prácticas que les ayudarán a que el viaje de implementar Git sea lo menos doloroso posible. No pensamos que haya un "mejor" sistema de control de versiones, cada uno tiene sus ventajas e inconvenientes en diferentes escenarios.

Esta guía forma parte de un conjunto de guías como se dijo en "Qué hay de nuevo" en la guía Parte 1 – **Estrategias de Branching**

NOTA Ver [Use Visual Studio with Git](#)² y [Set up Git](#)³ para familiarizarse con las herramientas para Visual Studio.

A quién va dirigida

En esta guía, nos hemos centrado en **Garry**, el jefe de desarrollo, **Doris**, la desarrolladora, y **Dave**, el administrador de TFS, todos son usuarios de TFVC y están considerando la opción de Git. Leed [ALM Rangers Personas and Customer Profiles](#)⁴ para más información sobre estos y otros personajes.

Visual Studio ALM Rangers

Los Visual Studio ALM Rangers lo forman miembros del grupo de producto de Visual Studio, Microsoft Services, Microsoft Most Valuable Professionals (MVP) y Visual Studio Community Leads. Su misión es ofrecer una guía profesional y soluciones para características que aún no estén disponibles. Hay disponible una lista online con los Rangers.

Home aka.ms/vsarunderstand

Solutions aka.ms/vsarsolutions

Membership aka.ms/vsarindex

Contributing ALM Rangers

Anisha Pindoria, Bill Heys, Dan Hellem, Dave McKinstry, David V. Corbin, Donovan Brown, Fabio Stawinski, James Waletzky, Matthew Mitrik, Richard Albrecht, Tommy Sundling, Willy-Peter Schaub

¹ <http://www.git-scm.com/>

² <https://msdn.microsoft.com/en-us/library/hh850437.aspx>

³ <https://msdn.microsoft.com/en-us/library/hh850445.aspx>

⁴ <http://vsarguidance.codeplex.com/releases/view/88001>

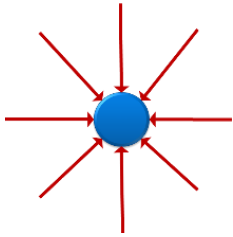
Sistemas de Control de Versiones Centralizados y Distribuidos

META

Típicas preguntas de un usuario de TFVC:

- ¿Cuál es la diferencia entre un sistema de control de versiones distribuido y uno centralizado?
- ¿Por qué debería elegir un sistema de control de versiones sobre otro?

TFVC == Sistema de Control de Versiones Centralizado

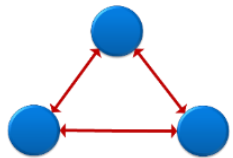


La mayoría de los sistemas de control de versiones son sistemas centralizados (CVCS, Centralized Version Control System). En un CVCS, se usa un repositorio único y compartido para mantener una copia del código fuente, y todos los cambios se hacen contra esa copia central. Los individuos tienen copias locales de los archivos que modifican y mandan de nuevo al servidor para compartirlo con los demás.

Team Foundation Version Control (TFVC) es un sistema de control de versiones centralizado en un servidor. TFVC permite dos modos de operación contra el repositorio central. El primer modo usa espacios de trabajo en el servidor (Server Workspaces), en ese modo, debe contactarse con el servidor antes de que cualquier desarrollador pueda añadir, editar, renombrar, borrar o incluso comparar cambios contra el original. El segundo modo (por defecto) usa espacios de trabajo locales (Local Workspaces), en las máquinas de los desarrolladores se guardan ciertos metadatos que permiten hacer un seguimiento de los cambios en los archivos, pero hay que contactar con el servidor para actualizar los archivos y subir los cambios. Los espacios de trabajo locales ofrecen una mayor flexibilidad para trabajar con control de versiones mientras estamos desconectados, y si no hay una buena comunicación pueden ser necesarias algunas operaciones de fusión para poder subir los cambios.

Además de TFVC, hay otros CVCS –como [Visual SourceSafe](#)⁵, [CVS](#)⁶ y [Subversion](#)⁷.

Git == Sistema de Control de Versiones Distribuido



Git es un sistema de control de versiones distribuido [open source](#)⁸. Git permite que los desarrolladores trabajen en dos modos, bien configurando una copia central y trabajar en modo hub, o bien creando un modelo totalmente punto a punto. El primero es, con diferencia, el más común y el modelo recomendado para Visual Studio Online (VSO).

⁵ <http://go.microsoft.com/fwlink/?LinkId=183114>

⁶ <http://cvs.nongnu.org/>

⁷ <http://subversion.tigris.org/>

⁸ <http://www.git-scm.com/about/free-and-open-source>

El primer modo usa un repositorio central. Una vez que los desarrolladores clonan el repositorio, no es necesario contactar con el repositorio central para ser capaces de hacer commit y ver el historial completo de los cambios del repositorio local. Los desarrolladores pueden trabajar de manera totalmente aislada y desconectada del repositorio central hasta que estén listos para fusionar sus cambios al repositorio central.

El segundo modo no usa un repositorio central y los cambios se sincronizan por los desarrolladores o usando soluciones de sincronización punto a punto como [gittorrent](https://code.google.com/p/gittorrent/)⁹.

Además de Git, hay otros DVCS – como [Monotone](http://www.monotone.ca/)¹⁰ y [Mercurial](http://mercurial.selenic.com/)¹¹.

JOYA

¿Preocupado por la recuperación de desastres?

- Como se clona todo el historial en la máquina de cada desarrollador, tendrá varias copias de seguridad completa en diferentes máquinas.
Importante: Los cambios locales siguen en riesgo hasta que se envíen al repositorio central o a otro repositorio distribuido.
- Se puede migrar un clon muy fácilmente volviéndolo a clonar.

Elegir entre TFVC y Git

Ver [use version control](#) ¹² y use los posters con las referencias rápidas, como ayuda para decidir qué sistema de control de versiones puede encajar mejor en su TFS Team Project. Unas veces convendrá usar un sistema centralizado y otras veces un sistema distribuido.

Escalabilidad, localización, experiencia y políticas son algunos de los factores que hay que tener en cuenta cuando estéis planeando vuestro repositorio. Por ejemplo:

- Si su equipo tiene **experiencia** con sistemas de control centralizados, TFVC puede mitigar los costes de adopción y aprendizaje. Del mismo modo, si tu equipo tiene **experiencia** con sistemas de control de versiones distribuidos, Git puede ser mejor.
- Si necesitas **escalar** a un repositorio muy grande deberías considerar TFVC.
- TFVC tiene un mejor soporte para otorgar **permisos y políticas** en TFS 2013.
- Git puede ser mejor para equipos **geográficamente** distribuidos, especialmente cuando hay problemas de conectividad.
- Para equipos que estén realizando desarrollos multiplataforma, hay herramientas de terceros que soportan repositorios Git. Por ejemplo, si estáis usando XCode, soporta Git desde fábrica.

JOYA

¡Evite artefactos binarios cuando uséis repositorios distribuidos!

- Como se clona todo el historial en cada una de las máquinas de los desarrolladores tenemos que ser cuidadosos cuando subimos recursos que no son código fuente.
- Para recursos grandes/binarios como videos, librerías o assemblies, DVCS no son la mejor opción para guardar contenido binario.

⁹ <https://code.google.com/p/gittorrent/>

¹⁰ <http://www.monotone.ca/>

¹¹ <http://mercurial.selenic.com/>

¹² <http://msdn.microsoft.com/en-us/library/ms181368.aspx>

Team project o granularidad de repositorio

META

Típicas preguntas de un usuario de TFVC:

- ¿Cuál es la diferencia entre TFVC y Git desde la perspectiva de un Team Project?
- ¿Puedo tener más de un repositorio Git por Team Project? Si es así, ¿por qué debería tener uno o más?

Cuando seleccione **TFVC** su repositorio se comparte entre otros Team Projects. Puede escalar el repositorio desde uno pequeño hasta un repositorio con millones de archivos por rama y grandes archivos binarios e identificar fácilmente los cambios que han hecho los desarrolladores.

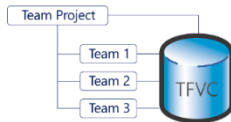


Figura 1 - Team Project con repositorio TFVC

Usando **Git** tendremos un repositorio por Team Project, pero podemos añadir más si es necesario.

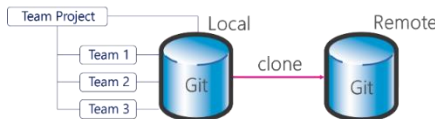


Figura 2 - Team Project con un repositorio Git centralizado y otro Git remoto.

No es necesario crear más Team Projects para tener más repositorios Git. Puede administrar los permisos para todos los repositorios Git en un solo team project en un solo lugar – y esta es una buena razón para no crear Team Projects (TP) separados para cada repositorio.

La decisión entre un solo repositorio y varios debería basarse en la forma y arquitectura del producto. Algunas cosas a tener en cuenta:

- El código con dependencias puede beneficiarse de un único repositorio.
- Varios repositorios pequeños y cohesivos puede reducir el coste de clonado y de almacenamiento, pero complica la trazabilidad y el mantenimiento.
- Tener varios repositorios puede ser útil para DevOps, por ejemplo, usar Dev en uno y Ops en otro.
- El repositorio TFVC o Git define la atomicidad.
 - No se pueden hacer commit en dos repositorios de manera atómica.
 - No podemos sincronizar el código entre repositorios, complicando los procesos de build/release/versionado.
- Aunque no es lo recomendado, podemos combinar y embeber varios repositorios usando [Git Tools - Submodules](http://git-scm.com/book/en/v2/Git-Tools-Submodules)¹³. Los repositorios Git separados no se actualizarán con un pull desde el repositorio principal, pero todos los Submódulos pueden actualizarse durante el clonado.

JOYA

Empiece por lo simple, pequeño y mantenga la cohesión

- Empiece con un sólo repositorio siempre que sea posible.
- Usar varios repositorios o submódulos puede ser muy complejo.

¹³ <http://git-scm.com/book/en/v2/Git-Tools-Submodules>

JOYA

Planifique sus TPC – TP - Teams - VC

- Planifique su TFS, Team Project Collections, Team Projects y Teams está cubierto en la guía [TFS Planning, Disaster Avoidance and Recovery, and TFS on Azure Iaas Guide](#)¹⁴ y está fuera del alcance de esta guía.

¹⁴ <http://aka.ms/treasure5>

Flujo de trabajo (lo básico)

Branching

META

Típicas preguntas de un usuario de TFVC:

- ¿Las estrategias de branching que se han visto en la guía sirven para Git?
- ¿Por qué el branching en Git no es costoso y es recomendable usarlo?
- ¿Qué estrategia de branching es la recomendada cuando se empieza con Git?
- ¿Qué estrategia de branching deberíamos considerar para un equipo de desarrollo?

NOTA

Aprenda cómo funciona el [Git branching](#) ¹⁵, entiéndelo probándolo, y abraza el modelo.

No simules las estrategias de branching que hemos visto en la Parte 1 – **Estrategias de Branching**. No las necesitarás.

Branching es diferente

Los conceptos tratados en la guía de Estrategias de Branching siguen siendo válidos. Sin embargo, en el contexto de Git, el valor del branching es mucho mayor, la complejidad y el coste disminuyen, y se pide encarecidamente a los equipos de desarrollo que creen ramas (muy a menudo) a partir de la rama principal cuando vayan a desarrollar funcionalidades cuando usan Git como sistema de control de versiones.

Debemos dar un paso atrás y examinar cómo guarda Git la información y cómo hace los procesos de branching y merging. Cada usuario tiene un repositorio local con un estado, consiguiendo así que las operaciones de commit, branch, merge y comparación sean muy rápidas. Una rama o branch no es más que un puntero a un commit concreto, y cuando se cambia de rama en Git sólo **se cambia el contenido** de tu directorio de trabajo al estado de ese commit, algo que es muy diferente a cómo funciona TFVC en el que cada rama tiene un directorio diferente. Borrar una rama es sólo borrar un puntero, esto ayuda a mantener la lista de ramas limpia y hace que el tamaño del repositorio sea el mínimo posible.

Para más información visita:

- [Git Branching – Branches in a Nutshell](#) ¹⁶
- [A pragmatic guide to the Branch Per Feature git branching strategy](#) ¹⁷
- [A successful Git branching model](#) ¹⁸
- [Comparing Workflows](#) ¹⁹

¿Por qué las ramas en Git son poco costosas?

En la Parte 2 – **Joyas de TFVC** vimos que los directorios son una forma de organizar las ramas en TFVC.

TFVC

- Cambiar entre directorios de ramas es **rápido** y **simple**.
- Las ramas son una “**copia**”.
- Las ramas son “**baratas**”, pero mapear varias ramas en tu máquina local es “**costoso**”.

¹⁵ <http://www.git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging>

¹⁶ <http://www.git-scm.com/book/en/v2/Git-Branching-Branched-in-a-Nutshell>

¹⁷ <https://www.acquia.com/blog/pragmatic-guide-branch-feature-git-branching-strategy>

¹⁸ <http://nvie.com/posts/a-successful-git-branching-model/>

¹⁹ <https://www.atlassian.com/git/tutorials/comparing-workflows/feature-branch-workflow>

En **Git** una rama es algo tan liviano como un puntero a un commit. Cuando creamos una rama, creamos un nuevo puntero que se mueve por el histórico, sin copiar nada del histórico. Ver [Git Branching - What a Branch Is](#)²⁰ para más detalles.

Git

- Cambiar entre ramas es **rápido** y **simple**.
- Una rama es un "**puntero**".
- El almacenamiento es "**barato**", ya que no se duplican datos.

Main Only

Si quieres investigarlo o usarlo en proyectos personales puedes empezar con la estrategia **main only**. Para ver en qué consiste la estrategia main only, pásate por Control de Versiones con TFS Parte 1 - Estrategias de Branching.



Figura 3 – Visualización de la estrategia Main Only de la guía

En Git se usa el formato Grafo Dirigido Acíclico (del inglés: Directed Acyclic Graph (DAG)) para representar el historial y la estrategia Main Only podría representarse como se indica en la Figura 4

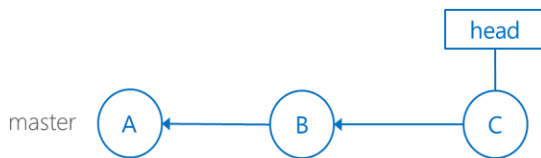


Figura 4 – Visualización de la estrategia Main Only usando el diagrama Grafo Dirigido Acíclico (DAG)

Un concepto muy importante en el grafo dirigido Acíclico es que todo nodo referencia a otro nodo, sin ciclos. Todo apunta a un commit padre, y las ramas son bifurcaciones en el historial. En el grafo anterior, el commit **raíz** puede ser 1.0. Master se refiere al repositorio remoto original, normalmente el repositorio centralizado, y **head** es un puntero a la rama que está actualmente activa (= checked out).

Rama por funcionalidad

Introducción

Recomendamos a los equipos de desarrollo que amplíen la estrategia de main-only con ramas **por funcionalidad** y que hagan lo que se describe en [Conduct a Git pull request on Visual Studio Online](#)²¹ para incluir los cambios en la línea principal.

Una funcionalidad suele ser resolver un bug individual o una funcionalidad en la que un desarrollador está trabajando hasta que esté listo para incluirlo en la rama central. Es normal crear y borrar varias ramas durante un día.

Es fácil visualizar las ramas y el trabajo asociado usando la vista jerárquica.

²⁰ <http://git-scm.com/book/en/v1/Git-Branching-What-a-Branch-Is>

²¹ <http://blogs.msdn.com/b/visualstudioalm/archive/2014/06/10/git-pull-request-visual-studio-online.aspx>

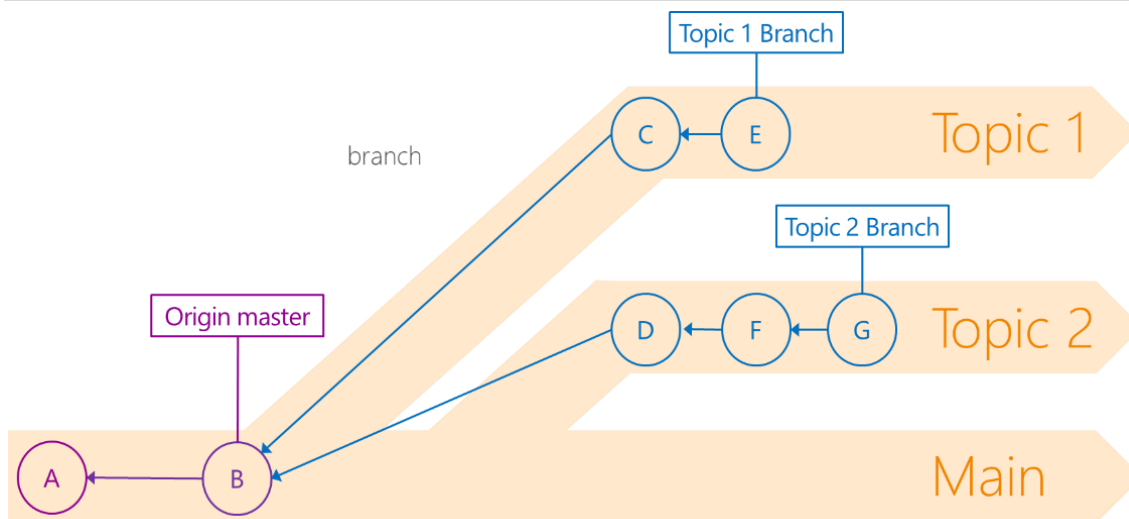


Figura 5 – Ramas por funcionalidad usando una vista jerárquica y el diagrama del Grafo Dirigido Acíclico (DAG)

¿Por qué usar ramas por funcionalidad?

- Las ramas por funcionalidad nos permiten cambiar de contexto de manera rápida y completa.
- Es más fácil identificar el historial y los cambios durante una revisión de código.
- Permiten hacer los merge cuando se esté listo, sin importar el orden en el que se crearon o cuándo se trabajó en ellos.

JOYA

¡Haz merge antes de borrar!

- Borra la rama cuando hayas terminado.

Haciendo Cambios

GOAL

Típicas preguntas de un usuario de TFVC:

- ¿Con qué flujo de trabajo podemos empezar cuando evaluamos Git?
- ¿Qué flujo de trabajo debemos tener en cuenta para un equipo de desarrollo?
- ¿Qué pasos a alto nivel tenemos que considerar en cada flujo de trabajo y dónde podemos encontrar más detalles?

Empieza por un flujo de trabajo centralizado

Te recomendamos que comiences con un flujo de trabajo centralizado, simplificando así el paso de TFVC a Git. Este flujo de trabajo usa un repositorio central, hosteado en TFS/VSO, como único punto de verdad para el historial de tu código y es efectivo con la estrategia de branching de master-only.

¿Por qué?

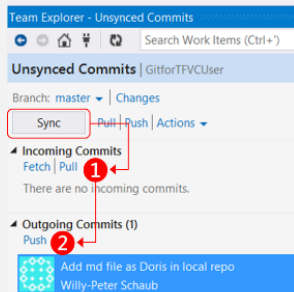
- Reduce la fricción durante el cambio de TFVC a Git.
- Pone a un lado el aprendizaje de las funcionalidades de los sistemas de control de código distribuido.
- Permite que los desarrolladores tengan su copia local del repositorio.
- Aísla los cambios en repositorios locales hasta que es necesario integrarlos.

Pasos básicos en el flujo de trabajo

NOTA Ver [use Visual Studio with Git](#) ²² para más detalles de las herramientas de Visual Studio.

1. [Clona](#) ²³ un repositorio local desde el repositorio remoto.
2. Haga cambios locales, añadiendo²⁴, eliminando²⁵ o modificando recursos en la rama master.
3. Haga [Commit](#) ²⁶ de sus cambios a tu repositorio local.
4. Haga [Pull](#) ²⁷ de los últimos cambios desde el repositorio central y resuelva los conflictos localmente.
5. Haga [Push](#) ²⁸ de sus cambios locales en el repositorio remoto.

GOYA La funcionalidad de **sync** de Visual Studio combina una operación de **pull** **1** y una operación **push** **2**.



El flujo centralizado tiene algunos matices en Git, como el **clonado**, **pull** y **push**, pero son familiares y fáciles de aprender para usuarios de TFVC.

Evoluciona al flujo de trabajo a ramas por funcionalidad

Una vez que su equipo se sienta cómodo con el flujo de trabajo centralizado, los matices de Git y se familiaricen con la visualización del historial, es tiempo de explorar otros flujos de trabajo. El flujo de ramas por funcionalidad es similar al de **ramas por funcionalidad** que vimos en la guía de **Estrategias de Branching** y es otro peldaño más en el paso de TFVC a Git.

¿Por qué?

- Aísla los bugs y el desarrollo de funcionalidad en ramas separadas.
- Protege a la rama master de código roto.
- Permite entornos de integración continua.

Pasos básicos en el flujo de trabajo

NOTA Ver [use branches](#) ²⁹ para conocer más detalles sobre los conceptos y las herramientas para ramas en Visual Studio.

²² <https://msdn.microsoft.com/en-us/library/hh850437.aspx>

²³ <http://git-scm.com/docs/git-clone>

²⁴ <http://git-scm.com/docs/git-add>

²⁵ <http://git-scm.com/docs/git-rm>

²⁶ <http://git-scm.com/docs/git-commit>

²⁷ <http://git-scm.com/docs/git-pull>

²⁸ <http://git-scm.com/docs/git-push>

²⁹ <https://msdn.microsoft.com/en-us/library/jj190809.aspx>

1. Cree una [rama](#)³⁰ a partir de tu rama master, esa será tu rama de funcionalidad.
2. Haga [Checkout](#)³¹ de la rama de funcionalidad.
3. Haz
4. ga cambios locales, añadiendo³², eliminando³³ o modificando recursos en la rama de funcionalidad.
5. Haga [Commit](#)³⁴ de los cambios en el repositorio local.
6. Publique la rama de funcionalidad para crear una referencia en el repositorio remoto.
7. Haga [Push](#)³⁵ de los cambios de la rama de funcionalidad local al repositorio remoto.

NOTA A medida que el equipo se familiarice y se sienta cómodo con Git, puede evaluar flujos de trabajo más avanzados como gitflow, Forking y otros.

³⁰ <http://git-scm.com/docs/git-branch>

³¹ <http://git-scm.com/docs/git-checkout>

³² <http://git-scm.com/docs/git-add>

³³ <http://git-scm.com/docs/git-rm>

³⁴ <http://git-scm.com/docs/git-commit>

³⁵ <http://git-scm.com/docs/git-push>

Revisando cambios con Pull Requests

META

Típicas preguntas de un usuario de TFVC:

- ¿Qué es un Pull Request?
- ¿Cómo podemos hacer revisiones de código en Git en VSO?
- ¿Cuáles son los pasos de alto nivel que tenemos que considerar en el flujo de trabajo de revisión y dónde encontramos más detalles?

¿Qué es un Pull Request?

Los pull requests en Git son similares a las revisiones de código en TFVC. Permiten que los desarrolladores en sus propias ramas pidan al equipo feedback sobre sus cambios antes de incluirlos en la rama master. Los desarrolladores pueden revisar los cambios, dejar comentarios, y aprobarlos si es necesario.

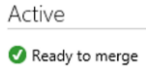
¿Por qué?

- Es la forma más simple de hacer revisiones de código en Git.
- Evita que los cambios o las nuevas funcionalidades ensucien la rama master antes de comprobar que el código reúne las métricas de calidad de tu equipo.

Pasos básicos en el flujo de trabajo – Sin conflictos

NOTA

Ver el post [Conduct a Git pull request on Visual Studio Online](#)³⁶ como tutorial más detallado.

1. Haga [Push](#)³⁷ de sus cambios de su rama local al repositorio remoto.
Vea la sección **Evoluciona al flujo de trabajo de ramas por funcionalidad**, en la página 14, para más detalles.
2. **Cree** un pull request para solicitar una revisión y hacer un merge desde su rama a la rama **master** en el repositorio centralizado.
3. **Revise** un pull request añadiendo comentarios de feedback.
4. Actúe en el feedback y haga [push](#) de sus cambios locales en el repositorio remoto.
5. **Complete** el pull request.
 - Si no hay conflictos, el pull request se mostrará como “ready to merge”.

 - Una vez que se apruebe, haga el [merge](#)³⁸ de los cambios desde su rama a la rama master.
6. Opcionalmente puede **borrar** su rama para evitar el ruido.

Pasos básicos en el flujo de trabajo – Con conflictos

1. Haga [Push](#)³⁹ de sus cambios de su rama local al repositorio remoto.
Vea la sección **Evoluciona al flujo de trabajo de ramas por funcionalidad**, en la página 14, para más detalles.
2. **Cree** un pull request para solicitar una revisión y hacer un merge desde su rama a la rama **master** en el repositorio centralizado.
3. **Revise** un pull request añadiendo comentarios de feedback.
4. Actúe en el feedback y haga [push](#) de sus cambios locales en el repositorio remoto.

³⁶ <http://blogs.msdn.com/b/visualstudioalm/archive/2014/06/10/git-pull-request-visual-studio-online.aspx>

³⁷ <http://git-scm.com/docs/git-push>

³⁸ <http://git-scm.com/docs/git-merge>

³⁹ <http://git-scm.com/docs/git-push>

5. Si la rama de destino ha cambiado, el pull request mostrará el mensaje:

Active

🟡 Target branch updated, re-evaluate
6. Si hay conflictos en el merge, el pull request mostrará **"Merge conflicts"**.
7. Haga [Pull](#)⁴⁰ de los últimos cambios de master.
8. Haga [Merge](#) de la rama master a su rama y resuelva los conflictos localmente.
9. Haga [Push](#) de su rama al repositorio remoto.
10. **Re-evaluate** el Pull Request.
11. **Complete** el pull request.
 - Si no hay conflictos, el pull request mostrará **"ready to merge"**.

Active

🟢 Ready to merge
 - Una vez aprobado, haga merge de los cambios desde su rama a la rama **master**.
12. Opcionalmente puede **borrar** su rama para evitar el ruido.

⁴⁰ <http://git-scm.com/docs/git-pull>

Migrando un repositorio TFVC

META

Típicas preguntas de un usuario de TFVC:

- ¿Cuáles son los conceptos y recomendaciones para migrar de TFVC a Git?
- ¿Qué datos pueden y deben ser migrados?
 - ¿Qué herramientas hay disponibles para ayudar en las migraciones?

NOTA

Ver los whitepapers de la guía [Understanding TFS migrations from on-premises to Visual Studio Online](#)⁴¹ para ver los pasos y conceptos para entender, planificar e implementar migraciones de Team Foundation Server.

Convertir un repositorio de TFS a Git es similar a cualquier esfuerzo de migración a Team Foundation Server (TFS). Debe decidir qué va a migrar y cuanta historia necesita mantener tras la conversión a un repositorio Git.

¿Qué migrar?

Recomendamos migraciones simples para evitar el coste y la complejidad de entender y migrar el historial, y para evitar las complicaciones que puedan surgir por tener un historial corrupto. Evite peligros siguiendo estas recomendaciones:

- Mantenga su antiguo repositorio TFVC para poder ver el pasado
- Mantenga los archivos binarios, como videos, en el repositorio anterior.
- Re-evalúe su estrategia de branching y **no** migre sus ramas de TFVC a Git.

¿Por qué no?

- Las migraciones son complejas, costosas y difíciles de planificar e implementar.
- Las migraciones están plagadas de casos extremos.
- La experiencia nos ha enseñado que no hay manera de mover nuestros datos con una fidelidad del 100%.

Recursos útiles para la migración

Artículos – Código fuente

- [Git-TFS / doc / commands / branch.md](#)⁴²
- [Git-TFS – Where Have You Been All My Life](#)⁴³
- [Git-TFS - Work with your Team \(Foundation Server\) with Git](#)⁴⁴
- [How Do I Use git-tfs idiomatically?](#)⁴⁵
- [I dropped a large binary object in Git ... now what?](#)⁴⁶
- [Trimming Git Commits/Squashing Git History](#)⁴⁷

⁴¹ <http://vsarguidance.codeplex.com/>

⁴² <https://github.com/git-tfs/git-tfs/blob/master/doc/commands/branch.md>

⁴³ <http://elegantcode.com/2011/03/15/git-tfs-where-have-you-been-all-my-life/>

⁴⁴ <http://www.methodsandtools.com/tools/git-tfs.php>

⁴⁵ <http://stackoverflow.com/questions/5129959/how-do-i-use-git-tfs-and-idiomatic-git-branching-against-a-tfs-repository>

⁴⁶ <http://www.sadev.co.za/content/i-dropped-large-binary-object-git-now-what>

⁴⁷ <http://stackoverflow.com/questions/2302736/trimming-git-commits-squashing-git-history>

Artículos – Work Items

- [Understanding TFS migrations from on-premises to VSO - Part 2 - Walkthrough](#)⁴⁸ le guiará sobre un proceso de migración de on-premises, usando las máquinas virtuales de [Brian Keller's](#)⁴⁹, a Visual Studio Online (VSO), incluyendo tanto el código fuente como los **work items**.
- [Migrating a TFS TFVC based team project to a Git team project - a practical example](#)⁵⁰ es un ejemplo de migración de TFVC a Git, incluyendo work ítems y varias personalizaciones.
- [Migrating a TFS TFVC based team project to a Git team project retaining as much source and work item history as possible](#)⁵¹.

Herramientas

- [Git-TF on Codeplex](#)⁵²
- [Git-TFS on GitHub](#)⁵³
- [Git Tool on Git-SCM](#)⁵⁴

⁴⁸ <https://vsarguidance.codeplex.com/downloads/get/1421797>

⁴⁹ <http://aka.ms/ALMVMs>

⁵⁰ <http://aka.ms/ju0tcj>

⁵¹ <http://aka.ms/w8n1ka>

⁵² <http://gittf.codeplex.com/>

⁵³ <https://github.com/git-tfs/git-tfs>

⁵⁴ <http://git-scm.com/download/win>

Funcionalidades de TFVC en Git

META

Típicas preguntas de un usuario de TFVC:

- Si usaba una funcionalidad de TFVC, ¿Cuál es la correspondiente en Git?

Si has usado esta funcionalidad de TFVC ...	Considera esta funcionalidad de Git ...
Fusión de ramas hermanas	Merge ⁵⁵
Políticas de Check-in	Pull requests ⁵⁶ , Branch Policies ⁵⁷
Revisiones de código	Pull requests
Múltiples workspaces locales	Clone ⁵⁸ (para varios repositorios locales) Checkout ⁵⁹ (para varias ramas locales)
Shelvesets	Ramas por funcionalidad (page 12) o stashes ⁶⁰

Tabla 1 – Correspondencias de funcionalidades entre TFVC y Git

⁵⁵ <http://www.git-scm.com/docs/git-merge>

⁵⁶ <http://blogs.msdn.com/b/visualstudioalm/archive/2014/06/10/git-pull-request-visual-studio-online.aspx>

⁵⁷ TBD, when available.

⁵⁸ <http://www.git-scm.com/docs/git-clone>

⁵⁹ <http://www.git-scm.com/docs/git-checkout>

⁶⁰ <http://www.git-scm.com/docs/git-stash>

Buenas y malas prácticas en Git

META

Típicas preguntas de un usuario de TFVC:

- *A partir de lo aprendido, cuáles son:*
 - ¿Las buenas prácticas que debemos promover?
 - ¿Las prácticas que debemos evitar?

Buenas prácticas

Haga commit de sus cambios frecuentemente.

- Siempre puede arreglarlos más tarde con los comandos [amend](#)⁶¹, [squash](#)⁶² o [rebase](#)⁶³
- Siempre debe compilar y probar sus commit antes de hacer push.

Use ramas para aislar sus cambios.

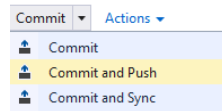
- Las ramas son baratas y privadas, y mergearlas es simple.
- Puede hacer un backup de los cambios en una rama subiéndolos al servidor.
- Las ramas por funcionalidad permiten el workflow de Pull Request.
- Limpie sus ramas cuando las fusione en la rama principal para limpiar el repositorio.

Establezca una convención de nombres para publicar ramas por funcionalidad.

- Nombre las ramas con la convención users/alias/branchname por ejemplo: users/doris/newfeature
- Esto ayudará a agrupar las ramas y hará más fácil a los demás identificar el "dueño" de la rama.

Recuerde hacer push de sus cambios

- Commit != Checkin. Un commit es una operación local.
- (Commit + Push) == Checkin. Recuerde hacer push de sus cambios locales cuando esté listo para compartirlos.
- No es necesario hacer commit y push a la vez, pero Visual Studio tiene herramientas que te ayudan con eso.



Evite

No haga rebase después de un push.

- Esto es un pecado mortal en Git, ya que fuerza a todos los demás a reajustar sus cambios locales.
- Si hace un rebase después de un push, hará que cualquiera que tenga una dependencia con su código entre en un callejón sin salida. Para salir de ese callejón tendrán que sufrir varios merge y rebase muy estresantes – ¡Y no estarán contentos con usted!

No suba binarios al repositorio.

- Como todos los repositorios tienen un historial, añadir un binario hace que crezca el repositorio.
- Evite subir videos, herramientas y assemblies. Puede que merezca la pena guardar algún contenido binario con su código, siempre que sea pequeño, y que no cambie a menudo.
- Es mejor usar NuGet para el versionado y administración de binarios.

⁶¹ <http://www.git-scm.com/book/en/v2/Git-Basics-Undoing-Things>

⁶² <http://www.git-scm.com/book/en/v2/Git-Tools-Rewriting-History>

⁶³ <http://www.git-scm.com/book/en/v2/Git-Branching-Rebasing>

JOYA

Aprenda haciendo, pero reacciona rápido

- Los conceptos de Git se aprenden mejor usando sus funcionalidades.
- Todo puede deshacerse, pero cuanto antes mejor.

Enlaces a más información

NOTA

Esta lista de materiales y herramientas no es una lista completa. Son pequeñas joyas que hemos encontrado a medida que hemos ido creando esa guía.

Blogs | Foros

[A successful Git branching model](#)⁶⁴

[Hg Init: a Mercurial tutorial](#)⁶⁵

[Migrating a TFS TFVC based team project to a Git team project - a practical example](#)⁶⁶

[TFS vs. Git...or is it TFS with Git?](#)⁶⁷

[Setting up the Perfect Git Command Line Environment on Windows](#)⁶⁸

[Use Git commandline directly from Visual Studio](#)⁶⁹

Materiales de referencia

[Git Documentation](#)⁷⁰

[Git-flow Cheatsheet](#)⁷¹

[GIT Succinctly](#)⁷²

[Git versus TFVC](#)⁷³

[GitHub Guides](#)⁷⁴

[Scott Chacon's Pro Git book](#)⁷⁵

[Use Visual Studio with Git](#)⁷⁶

Nivel básico y medio

Muestra el uso y el efecto de las operaciones de git

Git Succinctly por Ryan Hodson

MSDN

Información avanzada

La introducción oficial

MSDN

⁶⁴ <http://nvie.com/posts/a-successful-git-branching-model/>

⁶⁵ <http://hginit.com/>

⁶⁶ <http://blogs.blackmarble.co.uk/blogs/rfennell/post/2014/08/25/Reprint-Migrating-a-TFS-TFVC-based-team-project-to-a-Git-team-project-a-practical-example.aspx>

⁶⁷ <http://johanleino.wordpress.com/2013/09/18/tfs-vs-git-or-is-it-tfs-with-git/>

⁶⁸ http://www.woodwardweb.com/git/setting_up_the.html

⁶⁹ <http://blog.jessehouwing.nl/2013/11/use-git-directly-from-visual-studio.html>

⁷⁰ <http://git-scm.com/doc>

⁷¹ <http://danielkummer.github.io/git-flow-cheatsheet/>

⁷² <http://www.syncfusion.com/resources/techportal/ebooks/git>

⁷³ http://msdn.microsoft.com/en-us/library/ms18368.aspx#tfvc_or_git_summary

⁷⁴ <https://guides.github.com/>

⁷⁵ <http://git-scm.com/book/en/v2>

⁷⁶ <http://msdn.microsoft.com/en-us/library/hh850437.aspx>

Herramientas	Atlassian's SourceTree ⁷⁷ Git Diff Margin ⁷⁸ Git for Windows ⁷⁹ GitHub for Windows ⁸⁰ git-tf ⁸¹ git-tfs ⁸² PoshGit ⁸³	Gratuita y muy útil Muestra los cambios en vivo del archivo editado Herramienta de línea de comandos al estilo Bash, sin lujos, control total Libre, sencilla y rápida, lo simplifica todo Facilita compartir cambios con TFS, VSO y Git. Puente Git Entorno PowerShell para Git.
Tutoriales	Comparing Workflows ⁸⁴ Deep Dive into Git with TFS ⁸⁵ Getting Started with Git using TFS 2013 ⁸⁶ Git Videos ⁸⁷ GitHub Training ⁸⁸ Gittutorial ⁸⁹ Learn Version Control with Git ⁹⁰ Using Git with VS 2013 Jump Start ⁹¹ Welcome to LearnGitBranching ⁹²	Lista de flujos posible Sesión de implementación de TFS Git <u>Máquinas virtuales de Brian Keller</u> Video tutoriales de Git Más tutoriales, desde lo básico hasta lo más complejo. Tutoriales básicos de Git Un curso paso a paso para el principante Curso introductorio de Microsoft Virtual Academy Aprende los conceptos que se esconden detrás del branching
Tutoriales de Pluralsight	Git for Visual Studio Developers ⁹³ Git Fundamentals ⁹⁴ GitHub for Windows Developers ⁹⁵	

⁷⁷ <http://www.sourcetreeapp.com>

⁷⁸ <http://visualstudiogallery.msdn.microsoft.com/cf49cf30-2ca6-4ea0-b7cc-6a8e0dad1a8>

⁷⁹ <http://msysgit.github.io>

⁸⁰ <https://windows.github.com>

⁸¹ <https://gittf.codeplex.com/>

⁸² <https://github.com/git-tfs/git-tfs>

⁸³ <https://github.com/dahlbyk/posh-git>

⁸⁴ <https://www.atlassian.com/git/tutorials/comparing-workflows/>

⁸⁵ <http://channel9.msdn.com/Events/Build/2014/3-590>

⁸⁶ <http://download.microsoft.com/download/B/C/8/BC8558E1-192E-4286-B3B0-320A8B7CE49D/Getting%20Started%20with%20Git%20using%20Team%20Foundation%20Server%202013.docx>

⁸⁷ <http://www.lynda.com/Git-training-tutorials/1383-0.html>

⁸⁸ <http://training.github.com/kit/>

⁸⁹ <http://www.git-scm.com/docs/gittutorial>

⁹⁰ <http://www.git-tower.com/learn/ebook/command-line/introduction>

⁹¹ <http://www.microsoftvirtualacademy.com/training-courses/using-git-with-visual-studio-2013-jump-start>

⁹² <http://pcottle.github.io/learnGitBranching/>

⁹³ <http://www.pluralsight.com/courses/git-visual-studio-developers>

⁹⁴ <http://www.pluralsight.com/courses/git-fundamentals>

⁹⁵ <http://www.pluralsight.com/courses/github-windows-developers>

Apéndice

Entornos

La siguiente tabla resume algunos de los sistemas de control de versiones que usamos en nuestro Visual Studio Online:

Team	System	Created	# users	User distribution	Local size	Largest known file	Artifacts
ALM Rangers	TFVC	2011.04	~250	Worldwide	50.2GB	285MB	Source, docs and large binary files
Project Unicorn	Git	2014.11	~50	Worldwide	140MB	5MB	Source and documentation

Tabla 2 – Ejemplos de entornos

Conclusión

Aquí termina nuestra aventura de Git desde el punto de vista de un usuario de TFVC. Hemos visto las principales diferencias entre Git y TFVC, hemos contado algunas buenas prácticas, cosas que debe evitar y algunos consejos para saber elegir qué entorno vendría mejor.

Las últimas páginas contienen unos posters con Guías rápidas. Están disponibles de manera separada y puede que le resulte útil imprimirlas y colgarlas en la pared.

Esperamos que esta guía le haya resultado útil y le deseamos que salga airoso de sus aventuras en el desarrollo de software.

Atentamente

The Microsoft Visual Studio ALM Rangers

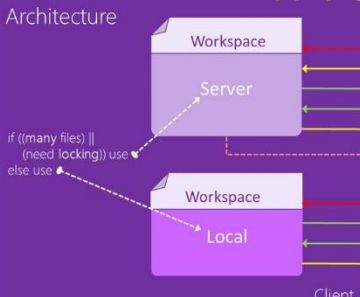


Posters

VC Cheat Sheet for TFVC and Git

Este poster está disponible tanto en JPG como PDF a alta calidad como parte de la guía.

Version Control Cheat Sheet for TFVC and Git



Architecture

Workspace (Server) ↔ Workspace (Local)

TFVC Server

Client ↔ Server

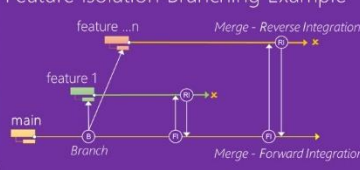
Get, Checkout, Shelf, Unshelve, Check-in, Lock, Get, Shelf, Unshelve, Check-in

if ((many files) || (need locking)) use else use

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.

Feature Isolation Branching Example



main, feature 1, feature ...n, Merge - Reverse Integration, Merge - Forward Integration

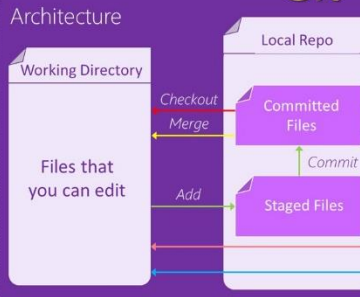
Commands

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**
 - shelves stores a set of pending changes in TFVC without a commit
 - shelvesview used to view details of a shelveset or view shelvesets belonging to 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 TFVC 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
 - undelete 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.
- TFSDelProject**
 - Command line tool which deletes a team project from a TFS team project collection. It is a non-reversible operation.

Git



Architecture

Working Directory ↔ Local Repo (Committed Files, Staged Files) ↔ Remote Repo (origin)


Client ↔ Remote

Checkout, Merge, Add, Commit, Fetch, Push, Pull (fetch+merge), Clone

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.

Topic Branch & Pull Request Example



origin, X, Y, Z, A, B, C, D, E, F, G, H

Y requests a pull request, X approves and initiates merge

Links


- msysgit.github.io - Windows client downloads
- git-scm.com - documentation
- github.com - code host
- bit.ly/tfvc - Git source control provider VS2008-2012
- bit.ly/tfvc - Git extensions

Commands

git command

git command line tool with a variety of commands.

- everyday**
 - add adds the current content to existing paths
 - clone create a working copy from existing repository
 - commit (verb) all the staged local changes
 - fetch the latest changes from origin and merge
 - pull 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 (verb) creates branch called name based on HEAD
 - branch (verb) deletes branch called name
 - checkout (verb) switch to the id branch
 - diff shows changes to tracked files
 - merge merge two branches
- uncommon**
 - blame show who changed what and when
 - branch (verb) find repository
 - grep search working directory
 - log show history of changes
 - show (verb) show a specific file from a specific id
- help**
 - Type **git -help** on the command line for a complete list of commands and arguments.

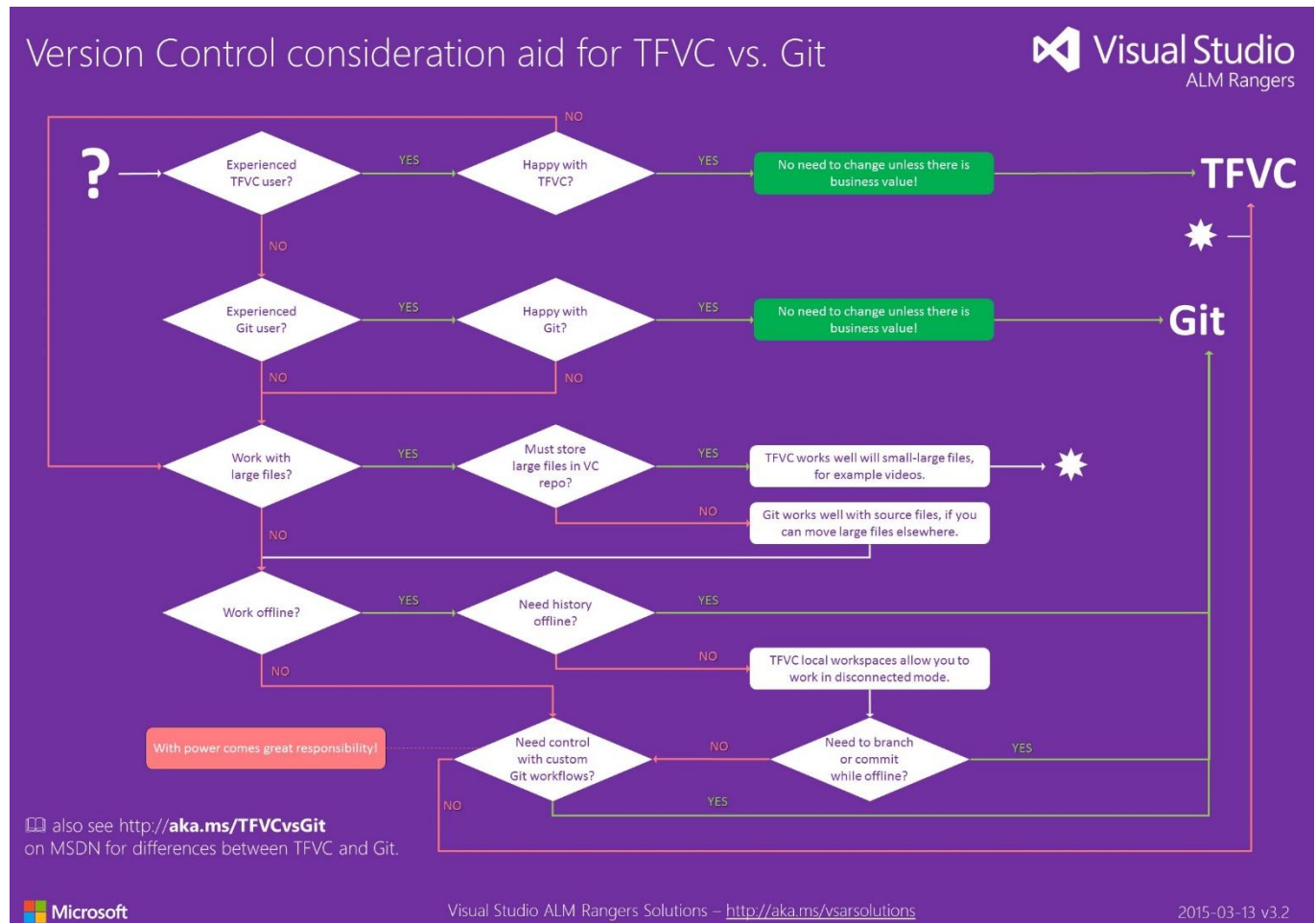


Visual Studio ALM Rangers Solutions – <http://aka.ms/vsarsolutions>

2015-03-13 v3.2

VC consideration aid for TFVC vs. Git

Este poster está disponible tanto en JPG como PDF a alta calidad como parte de la guía.



Git Workflows

Este poster está disponible tanto en JPG como PDF a alta calidad como parte de la guía.

