

TFS Version Control

Part 4

Git for TFVC Users

Visual Studio ALM Rangers

The MIT License (MIT)

Copyright (c) 2015 Microsoft Corporation

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.

Table of Contents

Foreword	4
Introduction	5
Centralized and Distributed VC Systems	6
TFVC == Centralized Version Control System	6
Git == Distributed Version Control System	6
Choosing between TFVC and Git	7
Team project or repo granularity	8
Workflows (basics)	9
Branching	9
Making Changes	11
Reviewing and acting on changes with Pull Requests	14
Converting a TFVC Repository	16
What to migrate?	16
Why not?	16
Useful migration resources	16
TFVC feature with Git	18
Git DOs and DON'Ts	19
Links to more information	20
Appendix	22
In Conclusion	23
Quick Reference Sheets / Posters	24

Foreword

Since the first writing of the TFS branching guide, a lot has changed in the world of version control. Hosted version control solutions are everywhere, and many of them include integration with build, project tracking, and other services. Distributed Version Control is no longer a niche, and has changed the way that many developers think about what it means to version their code. More developers are using version control than ever before – and this is a great thing for the billions of end users of those software development projects.

More developers are using version control. This means developers need solid, practical, and easy-to-digest guidance that is industry proven now more than ever. This guide, and those that came before it, strive to do just that – provide the version control guidance that development teams need to be effective while being approachable and flexible. In this latest edition, we have streamlined the guidance and simplified concepts with the goal of helping teams of all shapes and sizes to develop strategies that enable the flexibility and agility that modern development team's demand.

I also need to mention that this guide would not be in its current form without the readers. Thanks to all of you whom have contributed your feedback and ideas that have helped shape this guide over the years. As is in the past, if you see something in this guide that you would like to see changed or improved, please let us know!

Happy versioning!

Matthew Mitrik – Program Manager, Visual Studio Cloud Services

Introduction

This guide aims to serve as a starting point for learning about [Git](#)¹ and how it compares to TFVC from the perspective of a reader familiar with TFS. Choosing a version control system for your project or organization is a foundational decision which will affect how development is performed, and should not be made without consideration.

The intention of this guide is to provide you the differences between TFSVC and Git so you can decide what version control system is best suited for your needs. We have also included some DOs and DONTs to help assist you along your Git journey to make implementation as painless as possible. Note that we do not believe there is a 'best' version control system, as both have strengths and weaknesses in different scenarios. This guide is one of a set of companion guides as outlined in chapter "What's New" in the Part 1 – **Branching Strategies** companion guide.

NOTE Refer to [Use Visual Studio with Git](#)² and [Set up Git](#)³ and to familiarize yourself with the Visual Studio tooling.

Intended audience

In this guide, we primarily target **Garry**, the development lead, **Doris**, the developer, and **Dave**, the TFS administrator, all of whom are TFVC users and are considering Git. See [ALM Rangers Personas and Customer Profiles](#)⁴ for more information on these and other personas.

Visual Studio ALM Rangers

The Visual Studio ALM Rangers includes members from the Visual Studio Product group, Microsoft Services, Microsoft Most Valuable Professionals (MVP) and Visual Studio Community Leads. Their mission is to provide out-of-band solutions to missing features and guidance. A growing Rangers Index is available online.

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>

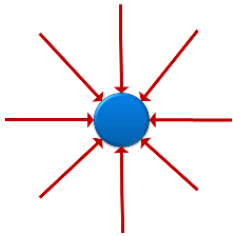
Centralized and Distributed VC Systems

GOAL

Typical TFVC User questions:

- *What is the difference between a centralized and distributed version control system?*
- *Why would I chose one version control system over the other?*

TFVC == Centralized Version Control System

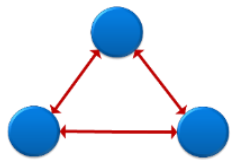


Most version control systems are centralized version control systems (CVCS). In a CVCS, a single, shared repository is used to maintain an authoritative copy of the source code, and all changes are made against the central copy. Individuals have local copies of the files which they modify and send back to the server to share with others.

Team Foundation Version Control (TFVC) is a centralized, server-based version control system. TFVC enables two modes of operating against central repository. The first mode uses Server Workspaces, where the server must be contacted before any developer can add, edit, rename, delete, or even diff their changes against the original. The second (default) mode uses Local Workspaces, where the additional metadata on the developer's machine tracks changes to files, but the server must still be contacted to update files and submit changes. Local workspaces offer greater flexibility for working with source control while offline, and without good communication can lead to additional merging before changes can be checked in.

In addition to TFVC, there are numerous other CVCSs – including systems such as [Visual SourceSafe](http://go.microsoft.com/fwlink/?LinkId=183114)⁵, [CVS](http://cvs.nongnu.org/)⁶ and [Subversion](http://subversion.tigris.org/)⁷.

Git == Distributed Version Control System



Git is an [open source](http://www.git-scm.com/about/free-and-open-source)⁸ distributed version control system. Git enables developers to work in two modes, by setting up a central copy and working in a hub and spoke model, or by creating a true peer-to-peer model. The former is by far the more common and the model recommended with Visual Studio Online (VSO).

The first mode uses a central repository. Once the repo is cloned by developers, the central repo need not be contacted to be able to commit and view full history from their local repo. Developers can work in complete isolation and disconnected from the central repo until they are ready to merge their changes with the central repo.

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>

The second uses no central repo and changes are synchronized by the developers or peer-to-peer synchronization solutions such as [gittorrent](https://code.google.com/p/gittorrent/)⁹.

In addition to Git, there are numerous other DVCSs – including [Monotone](http://www.monotone.ca/)¹⁰ and [Mercurial](http://mercurial.selenic.com/)¹¹.

GEM

Worried about disaster recovery?

- As all history is cloned to every developer's machine you have multiple full backups of the repository on different hardware.
Important: Local changes are still at risk until pushed to a central or other distributed repo.
- A clone is easily migrated by re-cloning.

Choosing between TFVC and Git

See [use version control](#)¹² and use the companion quick reference posters, as an aid to decide which version control system to associate with your TFS Team Project. There is a place for both the centralized and distributed version control system.

Scalability, location, experience and **policies** are some of the common factors you need to consider when planning your repository. For example:

- If your team is **experienced** with centralized version control systems, TFVC might avoid adoption and training costs. Similarly if your team is **experienced** with distributed version control systems, Git might be better.
- If you need to **scale** to a very large repository you may want to consider TFVC.
- TFVC currently has better support for fine grain **permissions** and **policies** in TFS 2013.
- Git might be better for **geographically** distributed teams, especially combined with poor connectivity.
- For teams doing cross-platform development, there may be more 3rd party-support for Git-based repositories. For example, if you are using XCode, support for Git is built in.

GEM

Avoid binary artifacts when using a distributed repository!

- As all history is cloned to every developer's machine you need to be careful about committing non-code assets.
- For large/binary assets, such as videos, libraries and assemblies, DVCS is not the best option for storing binary content.

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 or repo granularity

GOAL Typical TFVC User questions:

- *What is the difference between TFVC and Git from a Team Project perspective?*
- *Can I have more than one Git repo per Team project? If yes, why would I have one or more?*

When you select **TFVC** your repository is shared across other Team Projects. You can scale the repository from small to large codebases with millions of files per branch and large binary files and easily identify changes made by all developers.



Figure 1 - Team Project with TFVC repo

Using **Git** as your version control repository you get one repo per Team Project, but you can add more if needed.

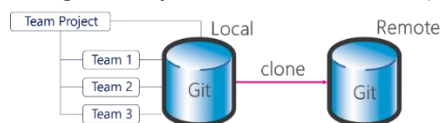


Figure 2 - Team Project with single centralized and one remote Git repo

You do not need to create more Team Projects to get more Git repos. You can manage permissions for all Git repos in a single team project in one place – which is a good reason not to create separate Team Projects (TPs) for each repo.

The decision whether to use a single repository, or multiple repositories should be based on the shape and architecture of the product. Some considerations:

- Code with dependencies may benefit from a single repository.
- Multiple small and cohesive repositories can reduce cloning and storage costs, but complicates repo traceability and maintenance.
- Multiple repositories can be useful for DevOps, to arrange Dev in one and Ops in another, for example.
- Your TFVC or Git repo is the boundary of atomicity.
 - You cannot commit to two repos atomically.
 - You cannot coordinate code between repos, complicating build/release/versioning processes.
- Although not recommended, you could combine and embed multiple using [Git Tools - Submodules](#)¹³. Separate Git repos will not be fetched with a default pull from the main repo, but all of the Submodules can be pulled down during cloning.

GEM Start single, small and keep cohesive

- Start with a single repository where possible.
- Using multiple repositories or submodules can be very complex.

GEM Plan your TPC – TP - Teams - VC

- Planning your TFS, Team Project Collections, Team Projects and Teams is covered in [TFS Planning, Disaster Avoidance and Recovery, and TFS on Azure Iaas Guide](#)¹⁴ and beyond the scope of this guide.

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

¹⁴ <http://aka.ms/treasure5>

Workflows (basics)

Branching

GOAL

Typical TFVC User questions:

- *Are the branching strategies covered in the companion guides relevant to Git?*
- *Why is Git branching regarded as non-expensive and recommended?*
- *Which branching strategy can we start with when exploring Git?*
- *Which branching strategy should we consider for team development environment?*

NOTE

Learn about how [Git branching](#) ¹⁵ works, understand it by trying it out, and embrace the model.

Do not simulate branching hierarchies in Git as covered in the Part 1 – **Branching Strategies** companion guide. You may not need them.

Branching is different

The concepts covered in the companion Branching Strategies guide are still valid. However, in the context of Git, the value of branching is higher, the complexity and cost lower, and development teams are encouraged to branch (often) when diverging from the main development stream when using the Git version control system.

You need to take a step back and examine how Git stores its data and how it does branching and merging. Every user has a local repository, making status, commit, branch, merge and diff operations fast. A branch is really nothing more than a pointer to a specific commit, and switching branches in Git will **swap out** the content of your working directory which is fundamentally different from the TFVC experience where your branches live in different folders. Deleting any branch is just deleting the pointer, which helps keeping the branch list clean and bloat of repository minimal.

For more information see:

- [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

Why is Git branching cheap?

As covered in Part 2 – **TFVC Gems** companion guide folders are an organizational tool for TFVC that provide a container for branches, other folders, or files.

TFVC

- Switching between branch folders is **quick** and **simple**.
- Branches are a “**copy**”.
- Branches are “**cheap**”, but mapping multiple branches to your local machine is “**costly**”.

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>

In **Git** a branch is simply a lightweight movable pointer to one of the commits. When you branch you create a new pointer to move around in history, without copying any of the history. See [Git Branching - What a Branch Is](http://blogs.msdn.com/b/visualstudioalm/archive/2014/06/10/git-pull-request-visual-studio-online.aspx)²⁰ for details.

Git

- Switching between branches is **quick** and **simple**.
- Branches are “**pointers**”.
- Storage is “**cheap**” as no version control data is duplicated.

Main Only branch

For research or personal projects you can start with the **main** only strategy. For an introduction into the main only strategy, please refer to the TFS Version Control, Part 1 -Branching Strategies companion guide.



Figure 3 – Main Only branching strategy visualization from branching strategy companion guide

With Git the Directed Acyclic Graph (DAG) format is used to represent history and the Main Only strategy would be represented as shown in Figure 4

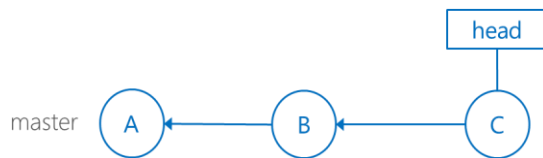


Figure 4 – Main Only branching and history visualized using the Directed Acyclic Graph (DAG) diagram

Important concepts of the directed acyclic graph representation is that every node references another node in the graph, without cycles. Everything points back to one root commit, and branches are forks in history. In the graph above the **root** commit would be 1.0. **Master** refers to the original remote repository, typically the centralized repository, and **head** is a pointer to the branch you currently have active (=checked out).

Topic branch

Overview

We encourage your development teams to supplement the main-only branching strategy with **topic** branches and [Conduct a Git pull request on Visual Studio Online](http://blogs.msdn.com/b/visualstudioalm/archive/2014/06/10/git-pull-request-visual-studio-online.aspx)²¹ to submit changes to the mainline.

A topic is often an individual bug or feature that a developer is working on until it's ready to be merged into a central branch. It is not uncommon to create and delete multiple topic branches during a single day.

It is easier to visualize the branches and associated work streams using the hierarchical view.

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>

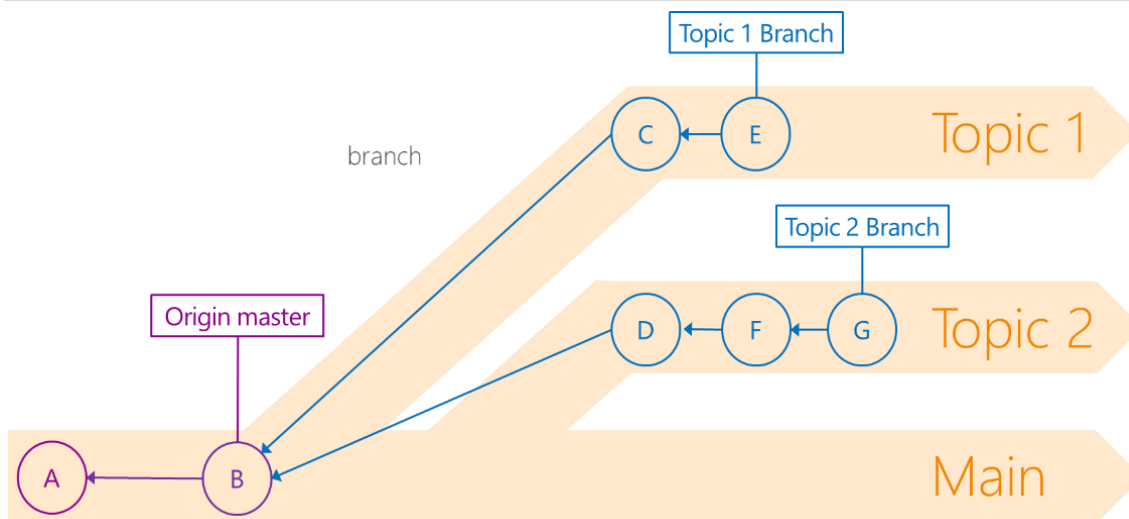


Figure 5 – Topic branch using a hierarchical view and the Directed Acyclic Graph (DAG) diagram

Why use topic branches?

- Topic branches allow you to context-switch quickly and completely.
- It is easier to identify history and changes during code review.
- Allows for merge when ready, regardless of the order in which they were created or worked on.

GEM

Merge before delete or bloat!

- Delete topic branches when done.

Making Changes

GOAL

Typical TFVC User questions:

- Which workflow can we start with when exploring Git?
- Which workflow should we consider for team development environment?
- What are the high-level steps to consider with each workflow and where do we find more details?

Start with the Centralized Workflow

We recommend that you start with the centralized workflow, simplifying the switch from the centralized TFVC to the distributed Git source control system. This workflow uses the central repository, hosted by TFS/VSO, as the single point-of-truth for the history of your source and is effective using the master-only branching strategy.

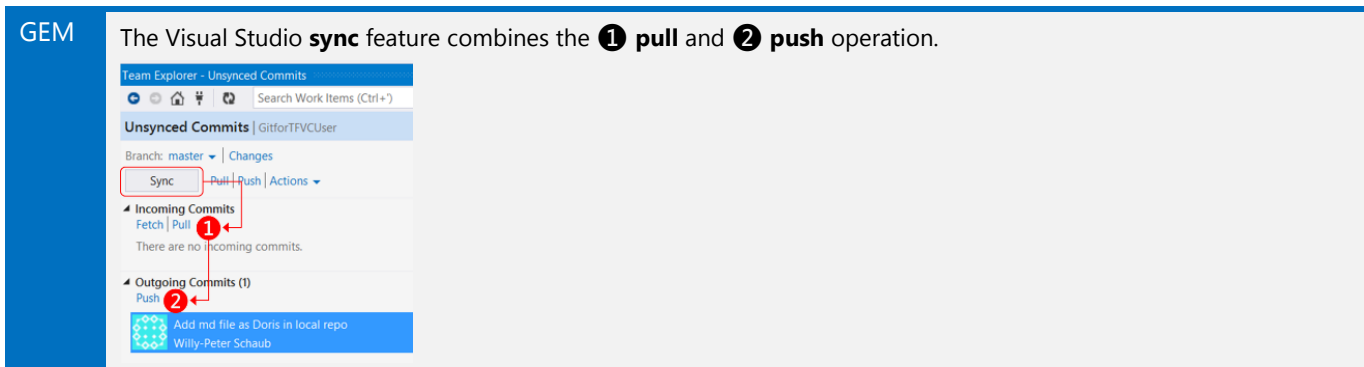
Why?

- Reduce a potential daunting context switch from TFVC to Git.
- Postpone the need to master distributed source control features.
- Allow developers to have their own local copy of the repository.
- Isolate changes to local repositories until it is convenient to integrate.

Basic workflow steps

NOTE Refer to [use Visual Studio with Git](#) ²² for details on the Visual Studio tooling.

1. [Clone](#) ²³ a local repo from the centralized remote repo.
2. Make local changes, by [adding](#) ²⁴, removing ²⁵ or modifying assets in the master branch.
3. [Commit](#) ²⁶ the changes to your local repo.
4. [Pull](#) ²⁷ latest changes from the centralized repo and resolve any conflicts locally.
5. [Push](#) ²⁸ your local changes to the centralized remote repo.



The centralized workflow has a few Git nuances, such as **cloning**, **pull** and **push**, but should otherwise look familiar and be easy to master for TFVC users.

Evolve to the Topic Branch Workflow

Once your team is comfortable with the centralized workflow, nuances of Git, and familiar with the visualization of and tracking of history, it is time to explore other workflows. The Topic Branch Workflow is similar to the **feature branching** covered in the companion **Branching Strategies** guide and thus another familiar stepping stone for the TFVC users transitioning to Git.

Why?

- Isolate topics such as bugs and feature development in a separate branch.
- Protect the master branch from broken code.
- Enable continuous integration environments.

Basic workflow steps

NOTE Refer to [use branches](#) ²⁹ for details on Visual Studio branching tools and concepts.

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. [Branch](#)³⁰ your local master branch a topic branch
2. [Checkout](#)³¹ the topic branch.
3. Make local changes, by [adding](#)³², removing³³ or modifying assets in the topic branch.
4. [Commit](#)³⁴ the changes to your local repo.
5. Publish your topic branch to create a reference on the remote repo.
6. [Push](#)³⁵ your local topic branch changes to the remote repo.

NOTE

As your team become familiar and comfortable with Git features, you can evaluate more advanced workflows, such as the Gitflow, Forking, and many other workflows.

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>

Reviewing and acting on changes with Pull Requests

GOAL

Typical TFVC User questions:

- *What is a Pull Request?*
- *How can we perform code reviews in Git on VSO?*
- *What are the high-level steps to consider with the review workflow and where do we find more details?*

What is a Pull Request?

Pull requests in Git are similar to code reviews in TFVC. They enable developers to work in their own topic branches and submit to the team for feedback on their changes prior to merging code into the master branch. Developers can then review the code changes, leave inline comments, and give approval if they are satisfied with those changes.

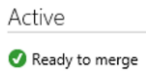
Why?

- The simplest way of conducting code reviews in Git.
- Isolate features or changes from polluting your master branch until the code meets your team quality requirements


Basic workflow steps – No conflicts

NOTE

See [Conduct a Git pull request on Visual Studio Online](#)³⁶ blog post for a detailed walkthrough.

1. [Push](#)³⁷ your local topic branch changes to the remote repo.
See **Evolve to the Topic Branch Workflow**, page 12, for details.
2. **Create** a pull request to request a review and merge from **topic** to **master** Branch on the centralized repo.
3. **Conduct** a pull request by adding review feedback comments.
4. Act on feedback and [push](#) your local changes to the remote repo.
5. **Complete** the pull request.
 - If there are no conflicts, the pull request will show “ready to merge”.

 - Once approved, [merge](#)³⁸ the changes from the topic to the master branch.
6. Optionally **delete** the **topic** branch to avoid clutter.

Basic workflow steps – With conflicts

1. [Push](#) your local topic branch changes to the remote repo.
See **Evolve to the Topic Branch Workflow**, page 12, for details.
2. **Create** a pull request to request a review and merge from **topic** to **master** Branch on the centralized repo.
3. **Conduct** a pull request by adding review feedback comments.
4. Act on feedback and [push](#) your local changes to the remote repo.
5. If the target branch has changed, the pull request will show “Target branch updated, re-evaluate”

6. If there are merge conflicts, the pull request will show “**Merge conflicts**”.
7. [Pull](#)³⁹ latest changes from master branch from the centralized repo.
8. [Merge](#) master to topic branch and resolve any conflicts locally.

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-pull>

9. [Push](#) your local topic branch changes to the remote repo.
10. **Re-evaluate** the Pull Request.
11. **Complete** the pull request.
 - If there are no conflicts, the pull request will show “**ready to merge**”.
12. Optionally **delete** the **topic** branch to avoid clutter.

Active

✓ Ready to merge

Converting a TFVC Repository

GOAL

Typical TFVC User questions:

- *What are the concepts and recommendation around migrating TFVC to Git?*
- *What version control data can and should be migrated?*
 - *What tooling is available to assist me with migrations?*

NOTE

Refer to the whitepapers in [Understanding TFS migrations from on-premises to Visual Studio Online](#)⁴⁰ guidance for the concepts and a walkthrough of understanding, planning, and implementing Team Foundation Server migrations.

Converting a TFS TFVC repository to Git repository is similar to any migration effort to Team Foundation Server (TFS). One has to decide on what is migrated and how much history is required in the conversion to a Git repository.

What to migrate?

We recommend simple 'tip' migrations to avoid the cost and complexity of understanding and migrating history, as well as exceptions caused by missing or corrupt history. Avoid potential pitfalls by:

- Keeping your old TFVC repository to look at the past.
- Keeping your large binary files, for example videos, in your old or another TFVC repo.
- Re-evaluating your branching strategy and do **not** migrate your TFVC branches to Git.

Why not?

- Migrations are complex, expensive and hard to plan and implement effectively.
- Migrations are plagued with edge cases.
- Our experience shows that no solution will move your data with full fidelity.

Useful migration resources

Articles - Source

- [Git-TFS / doc / commands / branch.md](#)⁴¹
- [Git-TFS – Where Have You Been All My Life](#)⁴²
- [Git-TFS - Work with your Team \(Foundation Server\) with Git](#)⁴³
- [How Do I Use git-tfs idiomatically?](#)⁴⁴
- [I dropped a large binary object in Git ... now what?](#)⁴⁵
- [Trimming Git Commits/Squashing Git History](#)⁴⁶

⁴⁰ <http://vsarguidance.codeplex.com/>

⁴¹ <https://github.com/git-tfs/git-tfs/blob/master/doc/commands/branch.md>

⁴² <http://elegantcode.com/2011/03/15/git-tfs-where-have-you-been-all-my-life/>

⁴³ <http://www.methodsandtools.com/tools/git-tfs.php>

⁴⁴ <http://stackoverflow.com/questions/5129959/how-do-i-use-git-tfs-and-idiomatic-git-branching-against-a-tfs-repository>

⁴⁵ <http://www.sadev.co.za/content/i-dropped-large-binary-object-git-now-what>

⁴⁶ <http://stackoverflow.com/questions/2302736/trimming-git-commits-squashing-git-history>

Articles – Work Items

- [Understanding TFS migrations from on-premises to VSO - Part 2 - Walkthrough](#) ⁴⁷ walks you through the migration of a simulated on-premises environment, using [Brian Keller's VM](#) ⁴⁸, to Visual Studio Online (VSO), including both version control and **work items**.
- [Migrating a TFS TFVC based team project to a Git team project - a practical example](#) ⁴⁹ of a TFVC to Git migration, including work items and customization hereof.
- [Migrating a TFS TFVC based team project to a Git team project retaining as much source and work item history as possible](#) ⁵⁰.

Tools

- [Git-TF on Codeplex](#) ⁵¹
- [Git-TFS on GitHub](#) ⁵²
- [Git Tool on Git-SCM](#) ⁵³

⁴⁷ <https://vsarguidance.codeplex.com/downloads/get/1421797>

⁴⁸ <http://aka.ms/ALMVMs>

⁴⁹ <http://aka.ms/ju0tcj>

⁵⁰ <http://aka.ms/w8n1ka>

⁵¹ <http://gittf.codeplex.com/>

⁵² <https://github.com/git-tfs/git-tfs>

⁵³ <http://git-scm.com/download/win>

TFVC feature with Git

GOAL

Typical TFVC User questions:

- *If I used a feature in TFVC, what would I consider in Git?*

If you have been using TFVC feature ...	Consider the Git feature ...
Directly merging sibling branches	Merge ⁵⁴
Check-in policies	Pull requests ⁵⁵ , Branch Policies ⁵⁶
Code Reviews	Pull requests
Multiple local workspaces	Clone ⁵⁷ (multiple local repo) Checkout ⁵⁸ (multiple local branches)
Shelvesets	Topic branches (page 10) or stashes ⁵⁹

Table 1 - TFVC to Git feature mapping

⁵⁴ <http://www.git-scm.com/docs/git-merge>

⁵⁵ <http://blogs.msdn.com/b/visualstudioalm/archive/2014/06/10/git-pull-request-visual-studio-online.aspx>

⁵⁶ TBD, when available.

⁵⁷ <http://www.git-scm.com/docs/git-clone>

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

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

Git DOs and DON'Ts

GOAL

Typical TFVC User questions:

- Based on learnings from the dogfooding, what are the:
 - DO's we should promote?
 - DON'T's we should avoid?

DOs

Do commit changes frequently.

- You can always fix them up later with an [amend](#)⁶⁰, [squash](#)⁶¹ or [rebase](#)⁶²
- You should always build and test your commits before you push them.

Do use branches to isolate your changes.

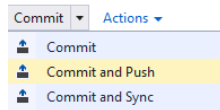
- Branches are cheap and private, and merging is simple.
- You can backup changes on a branch by pushing it to the server.
- Topic branches enable the Pull Request feature workflow.
- Clean up your branches when you've merged the changes into the mainline to avoid UI clutter.

Do use a naming convention when publishing topic branches.

- Name the branch using a common convention such as users/alias/branchname for example: users/doris/newfeature
- This will help group your branches and make it easy for others to identify the "owner".

Do remember to push your changes

- Commit != Checkin. Committing is a local operation.
- (Commit + Push) == Checkin. Remember to push your local commits when you're ready to share.
- You don't need to commit and push at the same time, but VS provides tools to help.



DON'Ts

Don't rebase after pushing.

- This is the cardinal sin in Git because it forces everyone else in the repo to rebase their local changes.
- If you rebase after pushing, you'll cause anyone that has taken a dependency on your code to be heading down a dead-end. To get back on track they'll need to do some painful merging or rebasing – and they won't be happy with you!

Don't commit binaries to your repo.

- As all repos have all of the history, committing large binary files means permanent bloat.
- Avoid adding videos, tooling, and assemblies. It may be appropriate to store some binary content with your code, especially if it is small, limited, and doesn't change very often.
- NuGet is the preferred solution for versioning and managing binaries.

GEM

Learn by doing, but react quickly

- A lot of the Git concepts are best explored by using the features.
- Everything can be undone, but the sooner the better.

⁶⁰ <http://www.git-scm.com/book/en/v2/Git-Basics-Undoing-Things>

⁶¹ <http://www.git-scm.com/book/en/v2/Git-Tools-Rewriting-History>

⁶² <http://www.git-scm.com/book/en/v2/Git-Branching-Rebasing>

Links to more information

NOTE

The list of other material and tools is not a complete list. It represents gems we have encountered while building this content, dogfooding Git or which has been recommended by one of the ALM Rangers.

Blogs | Forums

[A successful Git branching model](#)⁶³

[Hg Init: a Mercurial tutorial](#)⁶⁴

[Migrating a TFS TFVC based team project to a Git team project - a practical example](#)⁶⁵

[TFS vs. Git...or is it TFS with Git?](#)⁶⁶

[Setting up the Perfect Git Command Line Environment on Windows](#)⁶⁷

[Use Git commandline directly from Visual Studio](#)⁶⁸

Reference Materials

[Git Documentation](#)⁶⁹

[Git-flow Cheatsheet](#)⁷⁰

[GIT Succinctly](#)⁷¹

[Git versus TFVC](#)⁷²

[GitHub Guides](#)⁷³

[Scott Chacon's Pro Git book](#)⁷⁴

[Use Visual Studio with Git](#)⁷⁵

Entry & mid-level

Show usage and effect of git-flow operations

With *Git Succinctly* by Ryan Hodson

MSDN

Advanced information

The official introduction

MSDN

⁶³ <http://nvie.com/posts/a-successful-git-branching-model/>

⁶⁴ <http://hginit.com/>

⁶⁵ <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>

⁶⁶ <http://johanleino.wordpress.com/2013/09/18/tfs-vs-git-or-is-it-tfs-with-git/>

⁶⁷ http://www.woodwardweb.com/git/setting_up_the.html

⁶⁸ <http://blog.jessehouwing.nl/2013/11/use-git-directly-from-visual-studio.html>

⁶⁹ <http://git-scm.com/doc>

⁷⁰ <http://danielkummer.github.io/git-flow-cheatsheet/>

⁷¹ <http://www.syncfusion.com/resources/techportal/ebooks/git>

⁷² http://msdn.microsoft.com/en-us/library/ms181368.aspx#tfvc_or_git_summary

⁷³ <https://guides.github.com/>

⁷⁴ <http://git-scm.com/book/en/v2>

⁷⁵ <http://msdn.microsoft.com/en-us/library/hh850437.aspx>

Tools	Atlassian's SourceTree ⁷⁶	Free, workflow, fine control
	Git Diff Margin ⁷⁷	Displays live changes of the currently edited file
	Git for Windows ⁷⁸	Bash-like command line, no fancy, full power
	GitHub for Windows ⁷⁹	Free, nice and fast UI, simplifies a lot
	git-tf ⁸⁰	Facilitate sharing of changes with TFS, VSO and Git.
	git-tfs ⁸¹	Git/TFS bridge
	PoshGit ⁸²	PowerShell environment for Git.
Tutorials	Comparing Workflows ⁸³	Array of possible workflows
	Deep Dive into Git with TFS ⁸⁴	Session on TFS Git implementation
	Getting Started with Git using TFS 2013 ⁸⁵	For the Brian Keller VM
	Git Videos ⁸⁶	Video tutorials on Git
	GitHub Training ⁸⁷	More training material, from foundation to edge case.
	Gittutorial ⁸⁸	Entry-level tutorial on using Git.
	Learn Version Control with Git ⁸⁹	A step-by-step course for the complete beginner
	Using Git with VS 2013 Jump Start ⁹⁰	Microsoft Virtual Academy introductory course
	Welcome to LearnGitBranching ⁹¹	Help grasp the powerful concepts behind branching
Tutorials from Pluralsight	Git for Visual Studio Developers ⁹²	
	Git Fundamentals ⁹³	
	GitHub for Windows Developers ⁹⁴	

⁷⁶ <http://www.sourcetreeapp.com>⁷⁷ <http://visualstudiogallery.msdn.microsoft.com/cf49cf30-2ca6-4ea0-b7cc-6a8e0dad1a8>⁷⁸ <http://msysgit.github.io>⁷⁹ <https://windows.github.com>⁸⁰ <https://gittf.codeplex.com/>⁸¹ <https://github.com/git-tfs/git-tfs>⁸² <https://github.com/dahlbyk/posh-git>⁸³ <https://www.atlassian.com/git/tutorials/comparing-workflows/>⁸⁴ <http://channel9.msdn.com/Events/Build/2014/3-590>⁸⁵ <http://download.microsoft.com/download/B/C/8/BC8558E1-192E-4286-B3B0-320A8B7CE49D/Getting%20Started%20with%20Git%20using%20Team%20Foundation%20Server%202013.docx>⁸⁶ <http://www.lynda.com/Git-training-tutorials/1383-0.html>⁸⁷ <http://training.github.com/kit/>⁸⁸ <http://www.git-scm.com/docs/gittutorial>⁸⁹ <http://www.git-tower.com/learn/ebook/command-line/introduction>⁹⁰ <http://www.microsoftvirtualacademy.com/training-courses/using-git-with-visual-studio-2013-jump-start>⁹¹ <http://pcottle.github.io/learnGitBranching/>⁹² <http://www.pluralsight.com/courses/git-visual-studio-developers>⁹³ <http://www.pluralsight.com/courses/git-fundamentals>⁹⁴ <http://www.pluralsight.com/courses/github-windows-developers>

Appendix

Dogfooding environments

The following table summarizes a few of the version control systems used in our Visual Studio Online dogfooding environment:

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

Table 2 – Dogfooding environment examples

In Conclusion

This concludes our adventure into Git from the perspective of a TFVC user. We have touched on some main differences between Git and TFVC and provided advice on DOs and DONTs as well as thoughts on when one version control system should be favored over the other.

The final pages of this guide contains our Quick Reference sheets and posters. These are available separately as part of the guidance and you might find it useful to print these and hang them up in the team area.

We hope you have found this guide useful and wish you success in your software adventures.

Sincerely

The Microsoft Visual Studio ALM Rangers



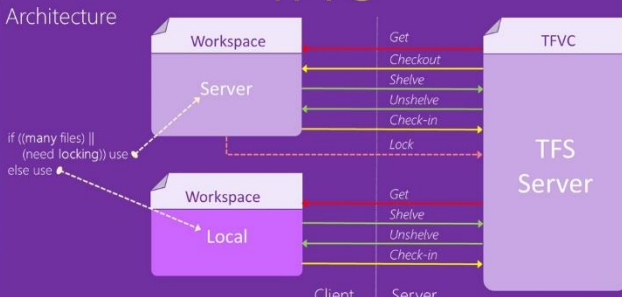
Quick Reference Sheets / Posters

VC Cheat Sheet for TFVC and Git

This cheat sheet is available separately in high-quality JPG and PDF format as part of the guidance.

TFVC

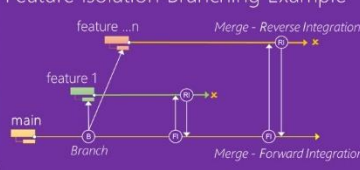
Architecture



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.

Feature Isolation Branching Example



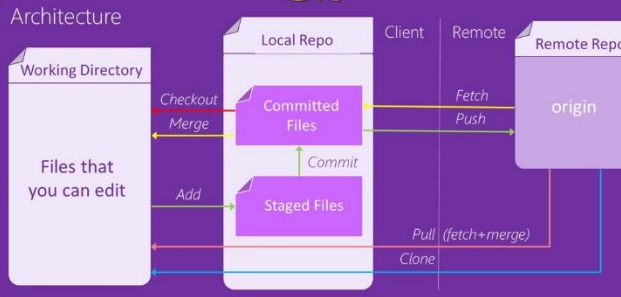
Commands

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**
 - shelveset: stores a set of pending changes in TFVC without a commit
 - shelvesets: 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 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
 - undocheckin: 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.

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.

Topic Branch & Pull Request Example




Links

- [msysgit.github.io](#) - Windows client downloads
- [git-scm.com](#) - documentation
- [github.com](#) - code host
- [bit.ly/rfrcz](#) - Git source control provider VS2008-2012
- [bit.ly/yq8at](#) - 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: fetch the latest changes from origin, not merging
 - pull: pull the latest changes from origin and merge into working copies
 - push: commit 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: switch to the id branch
 - diff: shows changes to tracked files
 - merge: two branches
- uncommon**
 - blame: show who changed what and when
 - bisect: find regressions
 - 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.

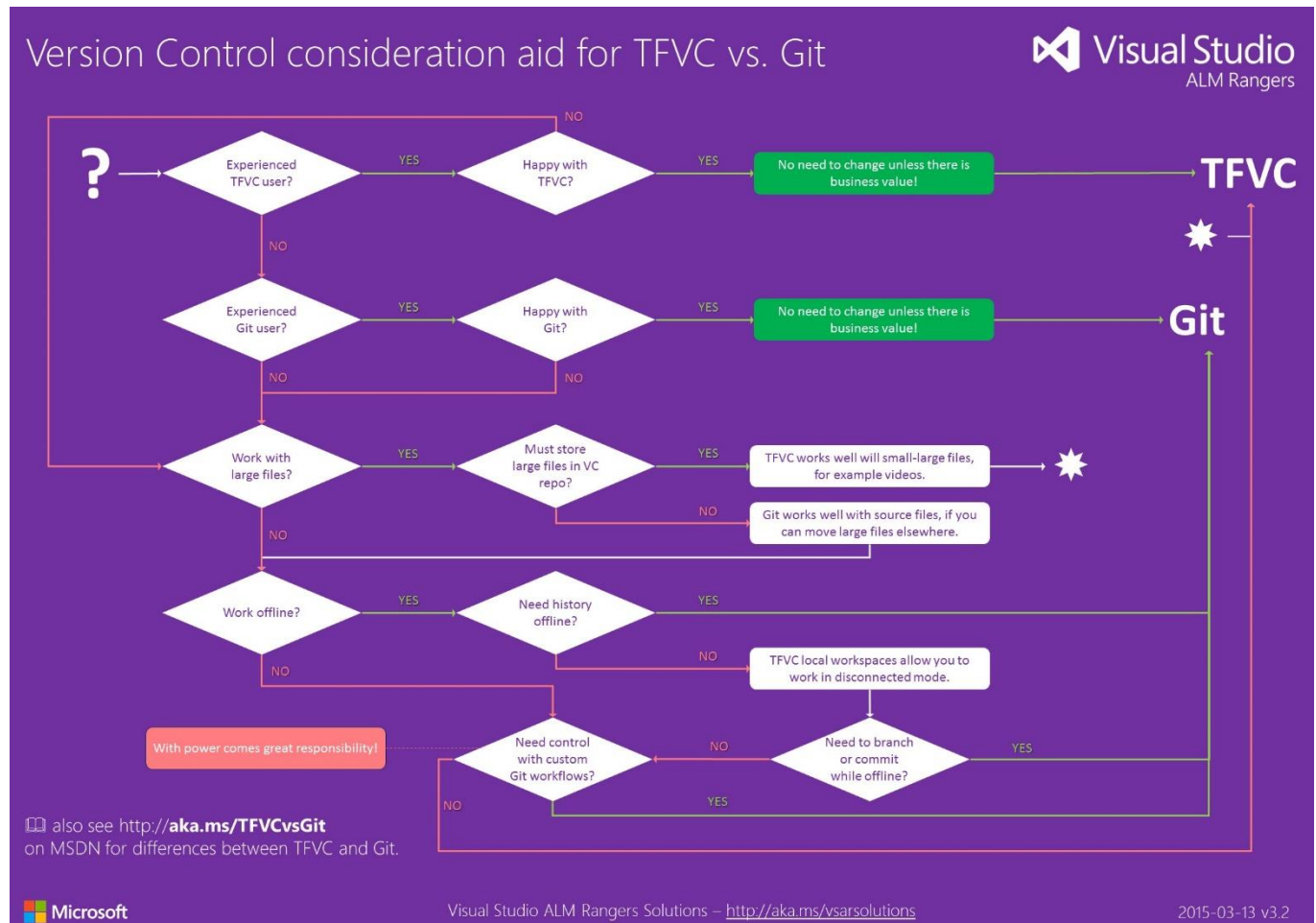


Visual Studio ALM Rangers Solutions – <http://aka.ms/vsarsolutions>

2015-03-13 v3.2

VC consideration aid for TFVC vs. Git

This cheat sheet is available separately in high-quality JPG and PDF format as part of the guidance.



Git Workflows

This cheat sheet is available separately in high-quality JPG and PDF format as part of the guidance.

