LINK Data Dispatcher



ETFS TFVC and Git

Prepared for

3M

1/21/2015

Version 1 Draft

Prepared by

**Tony Feissle**

Senior Consultant

[tfeissle@micrososoft.com](mailto:tfeissle@micrososoft.com)

Contributors

**Louis Weinstein**

|  |  |
| --- | --- |
|  | Prepared for 3M |

Revision and Signoff Sheet

Change Record

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Author | Version | Change reference |
| 1/9/2015 | Tony Feissle | 1.0 | Draft for review/discussion |
|  |  |  |  |
|  |  |  |  |

Reviewers

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Version approved | Position | Date |
| Louis Weinstein | 1.0 | Principal Consultant | 1/9/2015 |
| Michael O’Brien |  | Software Engineer Specialist |  |
|  |  |  |  |

Table of Contents

[1 3M Recommendations 3](#_Toc409597346)

[Git: 4](#_Toc409597347)

[2 Git Concepts 5](#_Toc409597348)

[2.1 Git Commands 6](#_Toc409597349)

[Tips & Tricks 7](#_Toc409597350)

[2.2 Using Git 8](#_Toc409597351)

[Appendix A – References 10](#_Toc409597352)

1. 3M Recommendations

General version control recommendations for the TFS environment @ 3M

There are currently two distinct repositories supported by TFS 2013:

* TFVC
* Git

The key differentiator between TFSVC and Git rests in where the source is located. TFVC is centralized and the code is located on the SQL server in 3M which supports TFS. With Git, the source is distributed amongst the various machines which access the data.

As Brian Harry [says](http://blogs.msdn.com/b/bharry/archive/2013/01/30/git-init-vs.aspx): “*I like to say our goal is to have the best available centralized version control solution and the best available distributed version control solution.*”

#### Team Foundation Version Control (TFVC) provides a very capable and scalable version control solution.

There are two models available within TFVC:

#### Check-in Check-out with server workspaces

This, original, model provides tremendous scalability for very large codebases together with fine level control and auditability. Checking out requires a connection to the server and offline editing is more difficult, and outside of Visual Studio even more so.

#### Edit Commit with local workspaces

Local workspaces were introduced in TFS 2012 alongside server workspaces and allow for much improved offline support. With a local workspace you can checkout locally, edit, rename and delete all without a connection to the server. It’s also much easier to work outside of Visual Studio in another tool as this model doesn’t rely on file attributes but instead on metadata held within the local workspace.

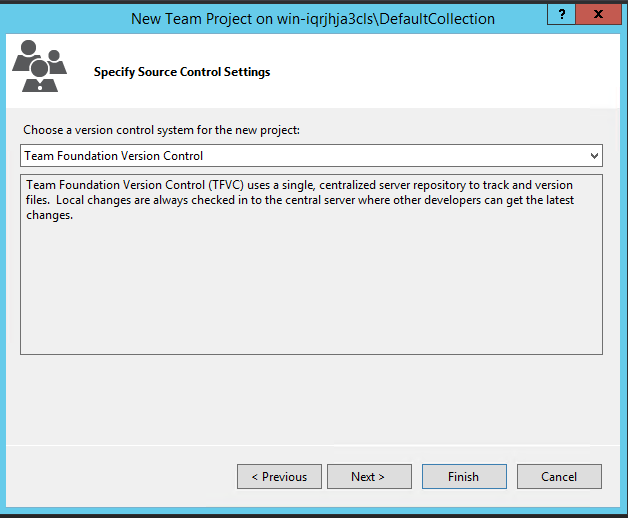
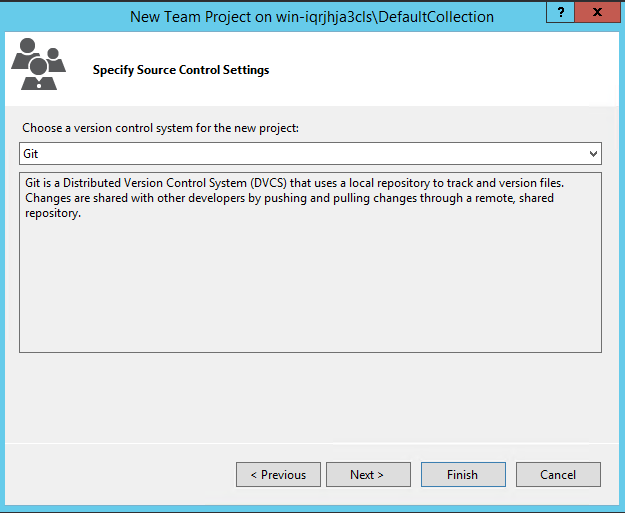
TFS using TFVC will suit teams that require any combination of large codebase support, fine permission control and where working offline is the exception rather than the rule. It will support any mix of Visual Studio and Eclipse developers, allowing them to share the same repository and ALM capabilities.

TFS with TFVC, with either model, is still centralized, so a developer working locally without a connection to the server has some limitations such as not being able to create branches, merge changes, view history or perform compares without the server connection. In other words, TFVC is not a DVCS.

### Git:

Git arose from the Linux kernel development community (including and in particular Linus Torvalds) in 2005 and has become the leading DVCS tool. It has a totally different model to centralized version control systems like TFVC by keeping a full repository locally on the developer’s machine. The repository is essentially a file based system with metadata in hidden files. The developer can then perform any task offline, including checking out, editing, committing, viewing and diffing on the full history, creating and destroying branches, merging and more.

In TFS, when creating a Team Project, you can choose which repository to use:



Most TFS features are fully integrated with Git. Good examples are found [here](http://www.visualstudio.com/en-us/get-started/share-your-code-in-GIT-vs).

After the creation of the team project, you will need to install Git tools:

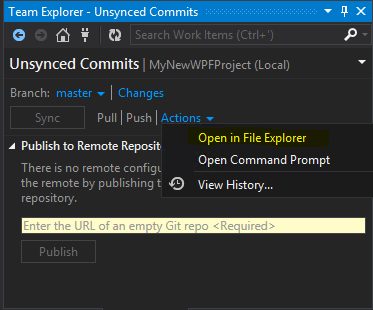
|  |  |
| --- | --- |
|  |  |

While this document will explain how Git is currently implemented in TFS, it is improving with every version. Both version control systems are fully supported by TFS, as new TFS versions are released, the version control choices will reach full parity with regards to all TFS features. For most 3M teams TFVC will be the **preferred** choice.

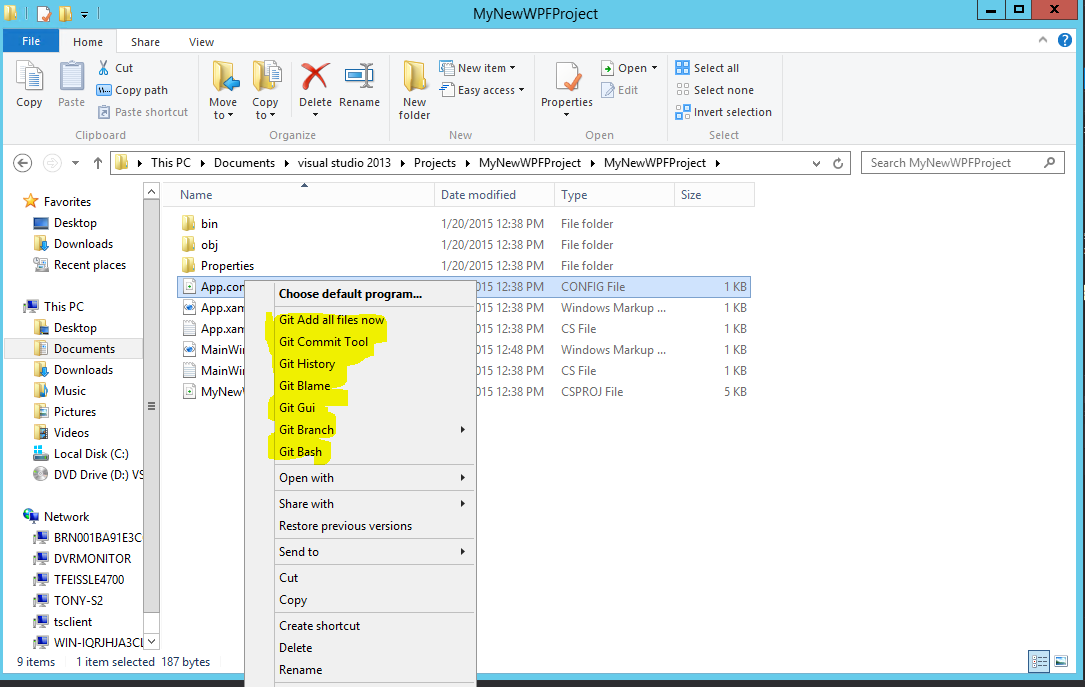
1. Git Concepts

* **repo, repository**: This is your object database were your history and configuration is stored. May contain several branches. Often it contains a worktree too.
* **index, staging area:** This is a 'cache' between your worktree and your repository. You can add changes to the index and build your next commit step by step. When your index content is to your likes you can create a commit from it. Also used to keep information during failed merges (your side, their side and current state)
* **clone:** A clone of a repository ("just another repository") or the act of doing so ("to clone a repository (creates a new clone)")
* **commit:** A state of your project at a certain time. Contains a pointer to its parent commit (in case of a merge: multiple parents) and a pointer to the directory structure at this point in time.
* **branch:** A different line of development. A branch in git is just a "label" which points to a commit. You can get the full history through the parent pointers. A branch by default is only local to your repository.
* **tree:** Basically speaking a directory. It's just a list of files (blobs) and subdirectories (trees). (The list may also contain commits in case you use submodules, but that's an advanced topic)
* **upstream:** After cloning a repository you often call that "original" repository "upstream". In git it's aliased to origin
* **a head:** The top commit of a branch (commit the label points to)
* **HEAD:** A symbolic name to describe the currently checked out commit. Often the topmost commit
* **version:** Might be the same as a commit. Could also mean a released version of your project.
* **tag:** A descriptive name given to one of your commits (or trees, or blobs). Can also contain a message (eg. changelog). Tags can be cryptographically signed with GPG.
* **archive:** An simple archive (.tar, .zip), nothing special with regards to Git.
* **patch:** A commit exported to text format. Can be sent by email and applied by other users. Contains the original author, commit message and file differences
* **changeset:** Synonym for "commit"
* **stash:** Git allows you to "stash away" changes. This gives you a clean working tree without any changes. Later they can be "popped" to be brought back. This can be a life saver if you need to temporarily work on an unrelated change (eg. time critical bug fix)
* **object:** can be one of commit, tree, blob, tag. An object has associated its SHA1 hash by which it is referenced (the commit with id deadbeaf, the tree decaf). The hash is identical between all repositories that share the same object. It also guarantees the integrity of a repository: you cannot change past commits without changing the hashes of all child commits.
* **(module,) submodule:** A repository included in another repository (eg. external library).
* **revspec:** A revspec (or revparse expression) describes a certain git object or a set of commits through what is called the extended SHA1 syntax (eg. HEAD, master~4^2, origin/master..HEAD, deadbeaf^!, …)
* **refspec:** A refspec is pattern describing the mapping to be done between remote and local references during Fetch or Push operations
* **history:** Describes all ancestor commits prior to a commit going back to the first commit.
  1. Git Commands

Git Actions:

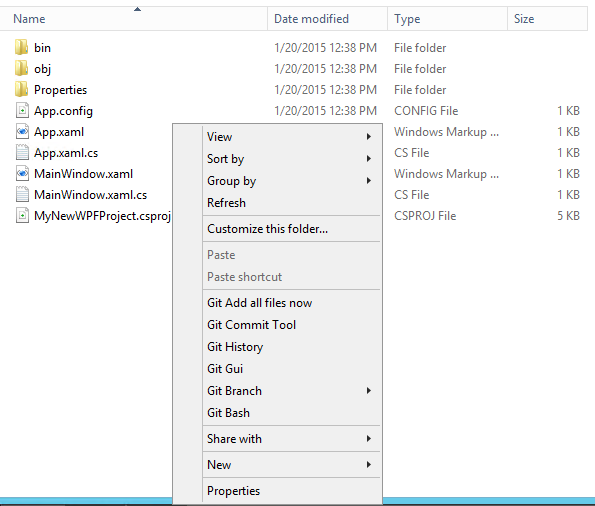


Opening in File Explorer and then right-clicking a file: (as you can see, these tools are installed from “Git tools”



### Tips & Tricks

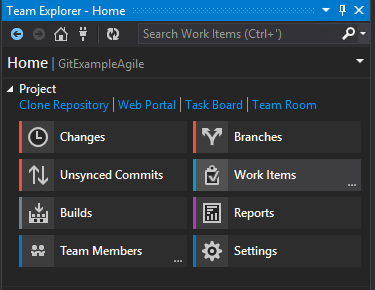
Install this [third-party tool](http://msysgit.github.io/) (it adds Windows Shell integration, Bash and various advanced Git Commands)



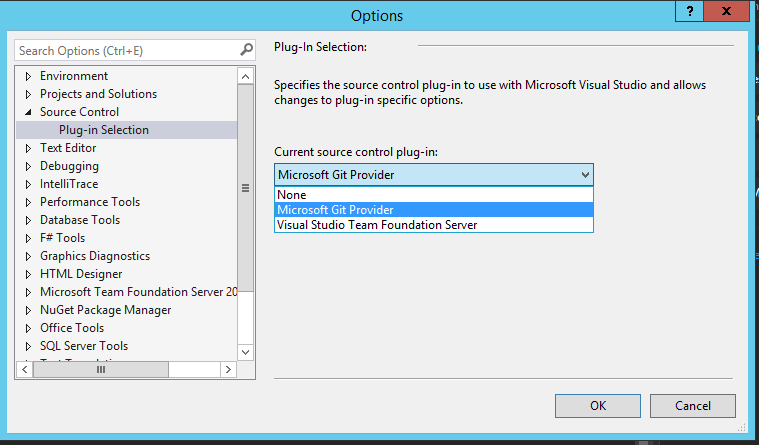
* 1. Using Git

The number one thing for an experienced TFVC user is that you will NOT be using Source Control Explorer (in Visual Studio).

As you can see below:



You have access to all of TFS, but you are connected to the “Microsoft Git Provider”



These links walk you through using Git:

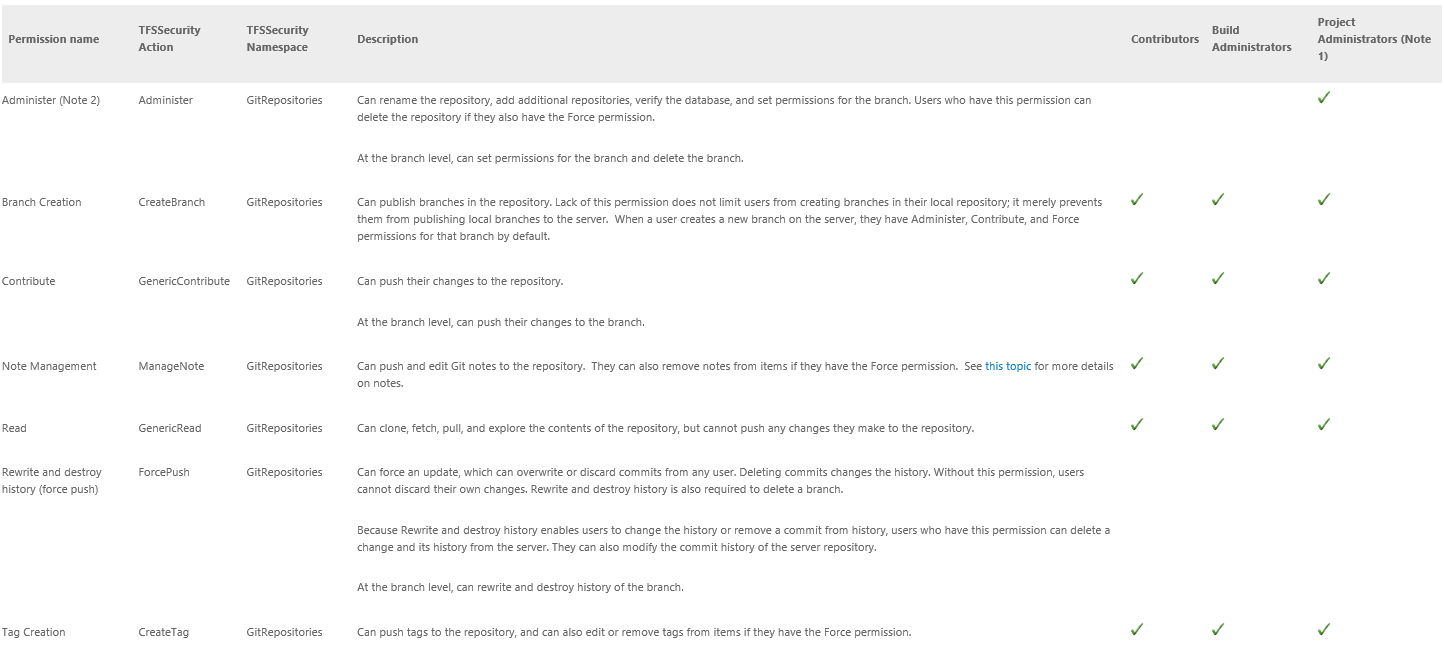
* [Set up Git on your dev machine (configure, create, clone, add)](https://msdn.microsoft.com/en-us/library/vstudio/hh850445.aspx)
* [Develop your app in a Git repository (track, commit)](https://msdn.microsoft.com/en-us/library/vstudio/hh967655.aspx)
* [Collaborate in a Git team project (pull, push)](https://msdn.microsoft.com/en-us/library/vstudio/hh850436.aspx)
* [Use Git branches to switch contexts, suspend work, and isolate risk](https://msdn.microsoft.com/en-us/library/vstudio/jj190809.aspx)
* [Conduct a Git pull request](http://blogs.msdn.com/b/visualstudioalm/archive/2014/06/10/git-pull-request-visual-studio-online.aspx) (for an on-premises team project you must first install Team Foundation Server Update 4)
* [Resolve Git conflicts](https://msdn.microsoft.com/en-us/library/vstudio/dd286559.aspx)
* [View and manage past versions in Git](https://msdn.microsoft.com/en-us/library/vstudio/dn237244.aspx)
* [Work from the Git command prompt](https://msdn.microsoft.com/en-us/library/vstudio/dd286572.aspx)

[Learn about Git permissions](https://msdn.microsoft.com/en-us/library/vstudio/ms252587.aspx#git)

You can set permissions on a [Git project, repository, or branch](https://msdn.microsoft.com/en-us/library/vstudio/hh850437.aspx) from the context menu or from the administration page in TWA, or by using the [TFSSecurity command line tool](https://msdn.microsoft.com/en-us/library/vstudio/ms252504.aspx). These permissions appear only for a team project set up to use Git as the source control system.

You can set all permissions for a project or repository. You can set Administer, Contribute, and Rewrite and destroy history (force push) permissions for a branch. Repositories and branches inherit permissions from assignments made at the project level.

By default, the project-level and collection level Readers groups have only Read permissions.



**Notes:**

1. For Project Collection Administrators and Project Collection Service Accounts, all permissions are set to Inherited allow.

Readers and Project Collection Build Service Accounts groups are assigned view-only permissions: Read.

1. Consider adding all permissions to any manually added users or groups that contribute to the development of the project.

## Appendix A – References

[Using Visual Studio with Git](https://msdn.microsoft.com/en-us/library/vstudio/hh850437.aspx)

[Excellent Third-party Git Tools](http://msysgit.github.io/)

[GitHub on Windows](https://windows.github.com/)