
DevOpsGroup
www.devopsgroup.com | Phone: 0800 368 7378 | e-mail: team@devopsgroup.com | 2019
Migrating 1 TFVC Repo to 668 Git Repos
WinOps 2019
© DevOpsGroup DOGPublic

James Reed

Lead Engineer @DevOpsGroup

Migrating 1 TFVC Repo to 668 Git Repos

1. Why?
2. How to investigate a TFVC repo
3. Migration planning

Why did we end up doing this and what were our options?

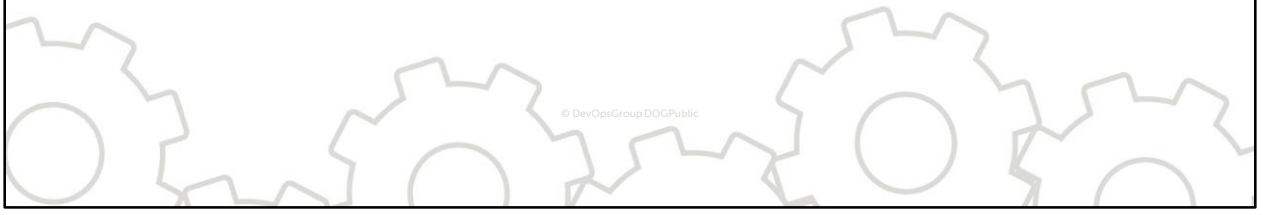
How do you analyse and investigate your repo to perform the migration

How should you plan your migration?

What does the process look like?

Why?

And what are our options?



Why

In early 2019 a client contacted us for help migrating from TFS 2013 to Azure DevOps

Initial Investigation

- Source control
- Builds
- Work Items
- Test Plans
- SharePoint
- SSRS and SSIS reporting

We asked them what features of TFS they were using.
Source control
Builds
Work Items
Test Plans
Sharepoint integration
SSRS and SSIS integrations / reporting

Initial Investigation

- Source control
- Builds
- Work Items
- Test Plans
- SharePoint
- SSRS and SSIS reporting

The client was only using Source Control and Work Items

They didn't have any automated builds or manual test plans

Sharepoint had already been moved

They weren't using any of the SSRS / SSIS reports (note this functionality is built in to AzD or replaced by Power BI. We just wanted to know so we could tell the client

Why move to Azure DevOps

- The client wanted to take advantage of Pipelines
- The client liked the advantages of SaaS
  - New Features released regularly
  - Wouldn't end up on an out of date platform again

We asked them why they wanted to move to Azure DevOps

They wanted to reset the way they use work items
Customised Agile template that wasn't working for them anymore
They didn't want to end up many versions behind the latest again (SaaS!)
They already had a couple of team using Azure DevOps and they wanted to get everyone on the same platform

Why move to Azure DevOps

- The client already had an active organisation in Azure DevOps
  - The client didn't want to have 2 separate organisations
- The client wanted to preserve as much source control history as possible
- The client wanted to reset their work item templates

We asked them why they wanted to move to Azure DevOps

They wanted to reset the way they use work items
Customised Agile template that wasn't working for them anymore
They didn't want to end up many versions behind the latest again (SaaS!)
They already had a couple of team using Azure DevOps and they wanted to get everyone on the same platform

What are the options to migrate

- Upgrade to Azure DevOps Server 2019 then use the Azure DevOps migration tool
  - Must be in a new organisation for each TFS collection database

This is the best way to migrate to Azure DevOps. It's fully supported by Microsoft and allows you to perform a full fidelity migration.

You may have to perform some fixes to work items as part of the upgrade and migration if you've customised a process template but this is well documented and pretty easy to do.

The only downside is that the process can take a while so downtime will be required

What are the options to migrate

- Use something like OpsHub
  - Will rewrite TFVC history

Tools such as OpsHub support TFVC to TFVC migrations, but the history will be replayed in to the new repo and checkin users and timestamps will reflect the account and time of migration. The old dates and users will be prepended to the checkin comments

OpsHub also supports a whole bunch of other features such as work item sync and synchronisation with other tools such as service desks, check it out it's good. Just not appropriate for our specific requirements

What are the options to migrate

- Use a “Tip” migration
  - Lose all history
  - Recreating branches could get messy

Just get the latest from TFS then Git Add (or TF add depending on the target)

If you want to maintain the branching structure then you might have to mess about a bit with creating empty branches and moving stuff around, or performing baseless merges.

Or you could just migrate Could just migrate a main branch but then you’d need to sync your migration to a time when everything was in Main and you could afford to lose child branches.

What are the options to migrate

- Upgrade to Azure DevOps Server 2019 and use the Git import tool
  - Only supports 180 days of history
  - Only a single branch is migrated

This is built in to TFS 2017 update 2 and above, however it only covers some simple scenarios and there are a few restrictions.

What are the options to migrate

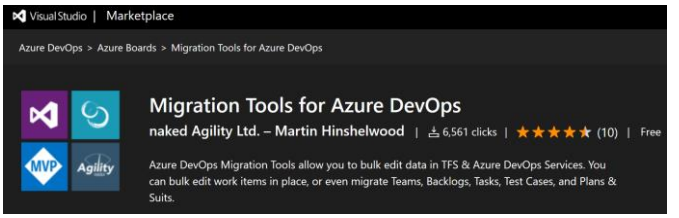
- Use Git-TFS to migrate all of the history to Git
  - There probably will be issues

This was the path we chose, it was the closes solution to the requirement of keeping as much history as possible, and migrate all of the code in the the existing Azure DevOps org.

However it's not perfect. Commit links to work items will be lost and other things can break.

Work item migration

For work item migrations use Martin Hinshelwood's Migration tools for Azure DevOps

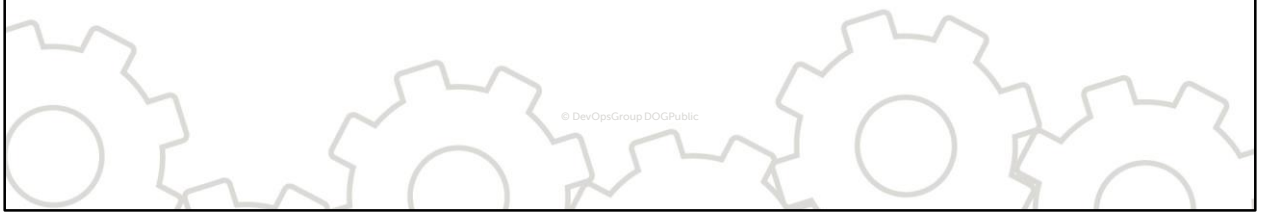


https://marketplace.visualstudio.com/items?itemName=nkdagility.vsts-sync-migration

Note that the talk doesn't cover the work item migration, but shout out to Martin Hinshelwood tools for migrating work items

Set up some Json files to map work items, you can to clever transformations such remap states and turn states in to tags so you can keep metadata about work items whilst making fairly significant changes

How to investigate a TFVC repo



How to investigate a TFVC repository

Talk to the Developers

To get an understanding of how the client was using TFVC we took a road trip to the clients office and spent a day with the development team.

We put TFVC up on a projected screen and got them to walk through the structure of source control with us. We asked them about what the code did and how they worked. This was to ensure that we weren't giving them bad advice.

When we looked at the code there were broadly 4 types of code

1. A couple of web sites, pretty standard stuff
2. A lot of small, discrete back end services (approximately 200)
3. A lot of device specific code not really device drivers, but can be thought of that way (approximately 400)
4. Miscellaneous things, not many and not complex

When we investigated the repo we found that a fair amount of code wasn't under a branch. About 6 of us spend 30 minutes right clicking and "converting to branch" it sucked, but it meant that a human being had made a conscious decision to include this code in to the migration. It prompted some good conversations about what code really needed to be migrated. Should this folder be included? Should this code this is in a branch be excluded?

Once we knew that there would be a lot of repos generated we had a conversation about what the impact would be. Did they really want 668 git repos. Would this be a complete pain or did it fit with their current workflow.

As it turned out, their development workflow would work well with this number of repos the “drivers” were very device specific and not a lot of common code. strategically it aligned with their future plans for adopting a microservices back end.

We also discussed the destination of these repos. Did they want them all in a single team project or did they want them organised in a different way

In the end we decided to migrate everything in to a single project as they wanted to manage the work items that way. It was agreed that they could move the repos around themselves post migration if that wasn’t working for them

We discovered that there were some “Main” branches with duplicate names which might cause issues. We discussed tried to ascertain were they the same things and one could be safely deleted? Some were some weren’t
We didn’t want to rename those branches as we knew it would prevent us migration history.

Finally we discussed what the future might look like in Git. I wanted to understand what experience the dev team had in using git to try to get a feel for any problems they might have. Did they understand the differences between centralised VC and Distributed and how that would affect their workflow. As it happens, most of the devs were already familiar with Git as that was what they were using in AzD and they were looking forward to having a unified workflow across the entire codebase. But if they weren’t familiar with Git then we would have had some discussions around training.

Issues?

- Code just in a folder not under a branch
  - Git-TFS won't migrate this
- What's happened in the past in the TFVC repo
  - Renamed or deleted and recreated branches
  - Baseless merges
  - Branching hierarchy changes
  - Branches converted to folders
  - Duplicate names

The structure of the code can affect the migration

In TFVC files can just be in a folder, there doesn't need to be any branches at all in a repo any files like this will not be picked up by git-tfs

Some historical changesets will cause problems
If a branch has been renamed, or if a branch has been deleted and then a new branch created with the same name git-tfs will trip up over this when it tries to replay the history in to Git

Baseless merges can cause issues (because they always do) but particularly if the hierarchy has changed.
If a branch has been converted to a folder this will be excluded by git-TFS

If you have Mainline branches with the same name, this can cause issues as git-tfs will try to create repos with the same name and the most recent clone may overwrite the previous repo clone (if you try and automate this

Show

The structure of the code in TFVC – code not in a branch

What's happened in the past in the TFVC repo

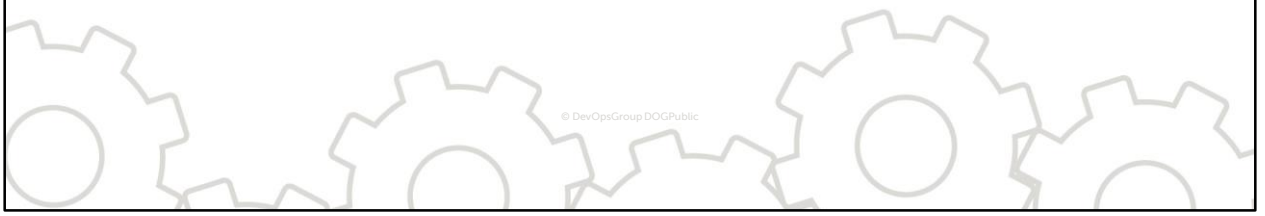
Renamed or deleted and recreated branches – try to migrate a repo with a broken main branch

Baseless merges – may not cause an issue directly but could be a problem if…
Branching hierarchy changes – child of a child where the middle branch has been deleted
Branches converted to folders particularly in the above scenario

Duplicate names

Show some scripts that do analysis and prep will be open source or something

Migration planning



Migration planning

Do a dry run

Practice makes perfect!

It doesn't matter how well you investigate, something will always crop up when you try to do this kind of migration

Make sure to capture the timings. To minimise downtime and to try and reduce the amount of out of hours work try get an understanding of how long things will take.

Take a backup of the collection database so if you need to make structural changes to the repo you can roll back without having a lasting impact on the prod server

Migration Planning

Create an implementation plan

Make sure you have a step of explicit steps for the migration and ensure that this is communicated with all of the stakeholders

Implementation Plan

Stage	Action	Responsible	Timings	Notes
Pre-Migration				
	Detach collection and take full backup	Client	30 mins	
	Re-attach collection	Client	10 mins	
	Fix broken and missing branches	DOG	30 mins	
	Clone Repositories	DOG	6 hours	
	Validate clone	DOG	2 hours	
	Address duplicate repos	DOG	30 mins	Some repos have duplicate names, these will be processed manually
	Address issues	DOG / Client	2 hours	
	Prepare work items for migration	DOG	1 hour	
			12 hours 40 mins	
Cutover	Final check in to TFS	Client	1 hour	Must be a checkin not Shelveset
	Detach collection and take full backup	Client	30 mins	
	Re-attach collection	Client	10 mins	
	Make TFS read only	Client	10 mins	
			1 hour 50 mins	
Migration	Pull repos to bring across latest code	DOG	2 hours	
	Change Origin and push repos	DOG	1 hour	
	Change Origin and push repos for duplicated repos	DOG	10 mins	
	Validate that repos have been pushed	DOG / Client	30 mins	
	Migrate work items	DOG	2 hours	
	Validate work items	DOG / Client	30 mins	
			3 hours 40 mins	Work items can be migrated in parallel to Code
Post Migration	Devs start to use Azure DevOps	Client		
	DOG to provide post implementation support	DOG	1 day	

DEMO

Show

Fixing missing branches
running migration script
Running repoint and push script

Wrap up

- We wanted to migrate to Azure DevOps
  - But the Database import method wasn't suitable
- You need to make sure you understand how the TFVC repo is structured
  - Talk to the Developers and poke around
  - PowerShell is your friend
- Do a dry run to understand the issues and plan your migration

Wrap up

Git is great, but getting there can be painful!

Questions