**Git Wiki**

**Building a Solution from GIT**



Simpson, Tabatha (RT-PHL)

May 28

**Contents**

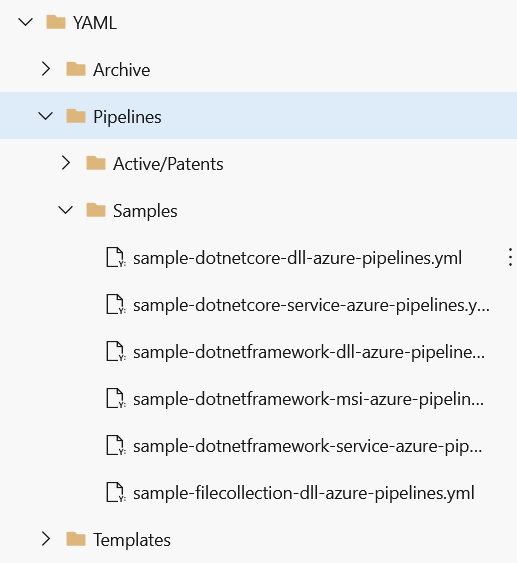
* [Using YAML Pipeline (preferred)](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/508/Building-a-Solution-from-GIT#using-yaml-pipeline-(preferred))
  + [Create AzureDevOps Pipeline](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/508/Building-a-Solution-from-GIT#create-azuredevops-pipeline)
  + [First Run](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/508/Building-a-Solution-from-GIT#first-run)
* [Desktop Application using Build Template (DON'T DO THIS)!](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/508/Building-a-Solution-from-GIT#desktop-application-using-build-template-(don't-do-this)!)

**Using YAML Pipeline (preferred)**

Here is a yaml guide from Microsoft: <https://docs.microsoft.com/en-us/azure/devops/pipelines/yaml-schema>

1. Based upon the type of project for which you are creating an Azure DevOps pipeline, choose the appropriate YAML template from the list of available YAML pipeline samples:

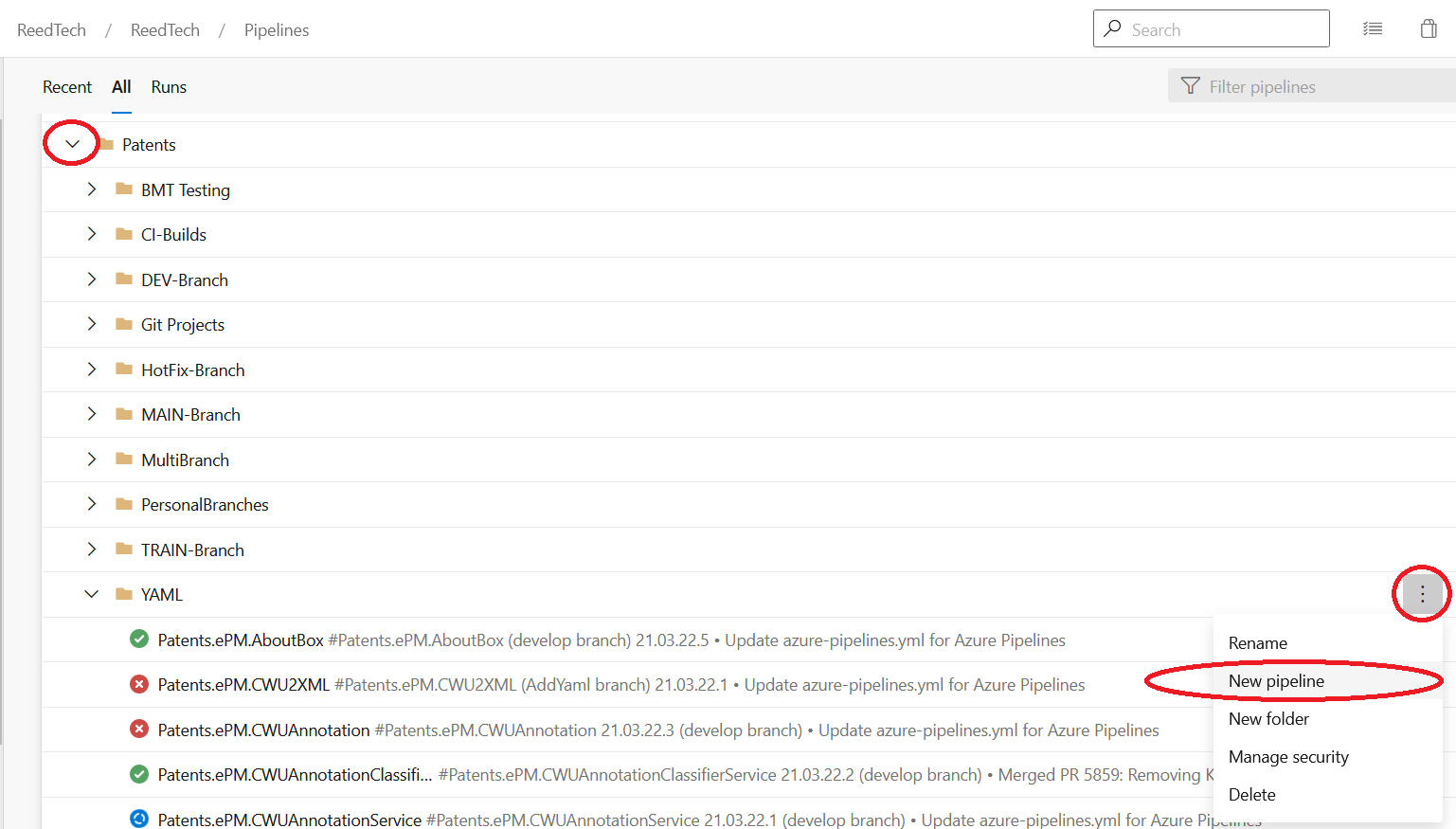
<https://tfsprod.reedtech.com/tfs/ReedTech/ReedTech/_git/Patents.DevOps?path=%2FAzureDevOps%2FYAML%2FPipelines%2FSamples&version=GBmaster>



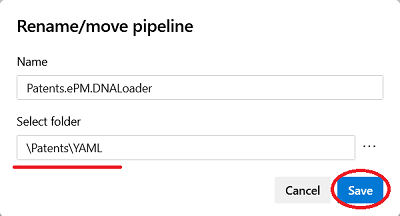
1. Download a copy of the chosen yml file to the solution folder level of your project and rename it as "azure-pipelines.yml".
2. Check the contents of the yml file to make sure it looks good - there may be things to change specific to the project.
3. Check in the new yml file to your remote branch.
   * As a suggestion, you may want to create a new feature branch from develop for your pipeline yml file while you work to get a successful build. You can continue to push changes to the remote copy of your feature branch without needing a pull request to merge each iterative change into develop. Once you have a successful build from your feature branch, you can make one pull request merging all your changes into the develop branch.

**Create AzureDevOps Pipeline**

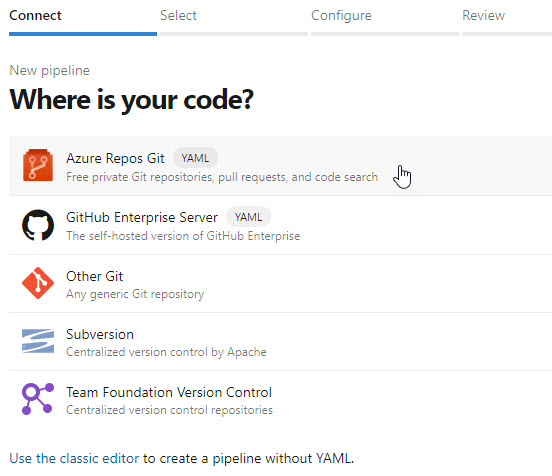
1. Navigate to the [Pipelines "All" view](https://tfsprod.reedtech.com/tfs/ReedTech/ReedTech/_build?view=folders), then expand "Patents". Click the ellipses button to the right of the "YAML" folder, and click on "New Pipeline":



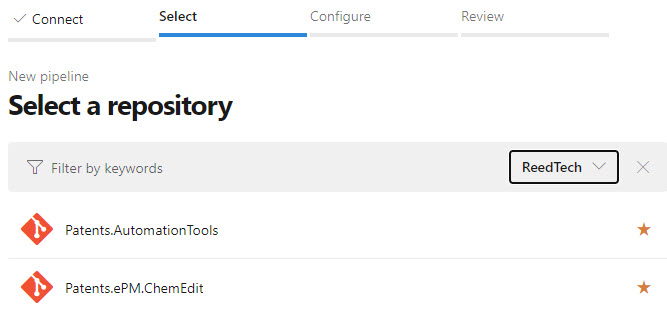
NOTE: This will specifically save the pipeline in this folder. If you miss this step, the pipeline automatically gets saved at the root level. In that case, you can move the pipeline after it's been created by clicking the ellipses button, and going to Rename/move. Make sure "\Patents\YAML" is the Select folder, and click "Save":



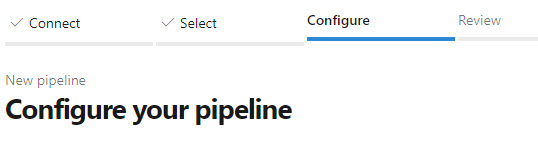
1. Under "Connect", click "Azure Repos Git":



1. Under "Select", click on your project repository:

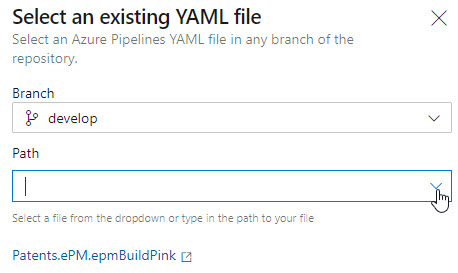


1. Under "Configure", select "Existing Azure Pipelines YAML File":

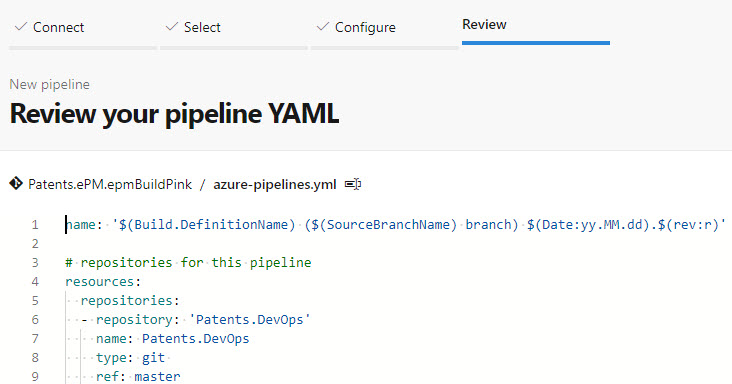
  
...

C:\Users\ghan\Downloads\Pic\06-1.jfif

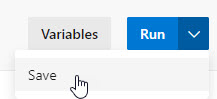
1. A frame will appear for you to pick the yml file saved to your project repository. Click the "Continue" button at the bottom when done.



1. The "Review" tab will show the YAML file selected:

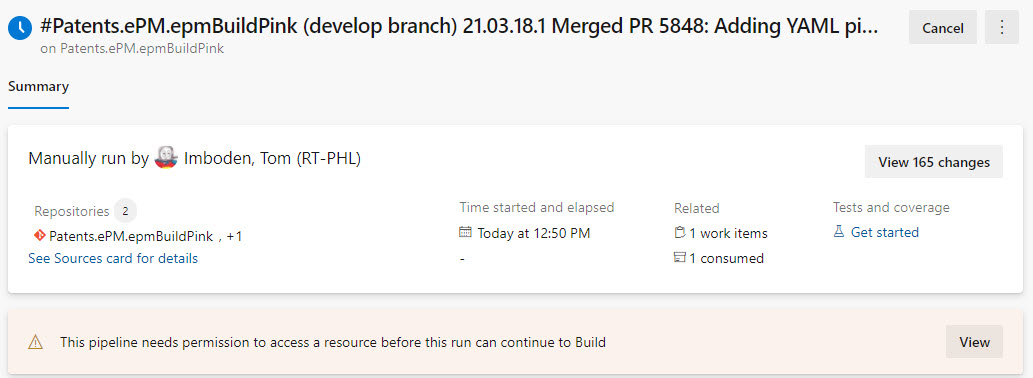


1. You can save your progress under the "Run" button dropdown to the right:

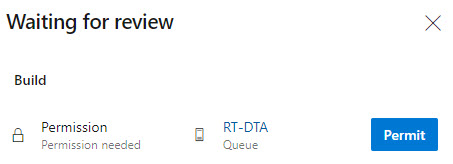


**First Run**

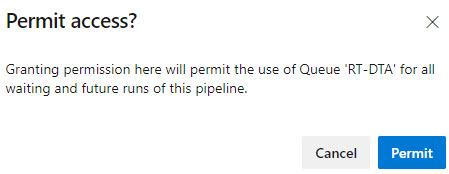
1. Test run your new pipeline. You will encounter some permission request.



1. Click "View":



1. Click "Permit" to allow the pipeline to use the build pool agent.



1. Click "Permit" to avoid this request for future runs.

Note: If you get a "Result: False" in the Run Checkmarx Scan step, do not worry about it.

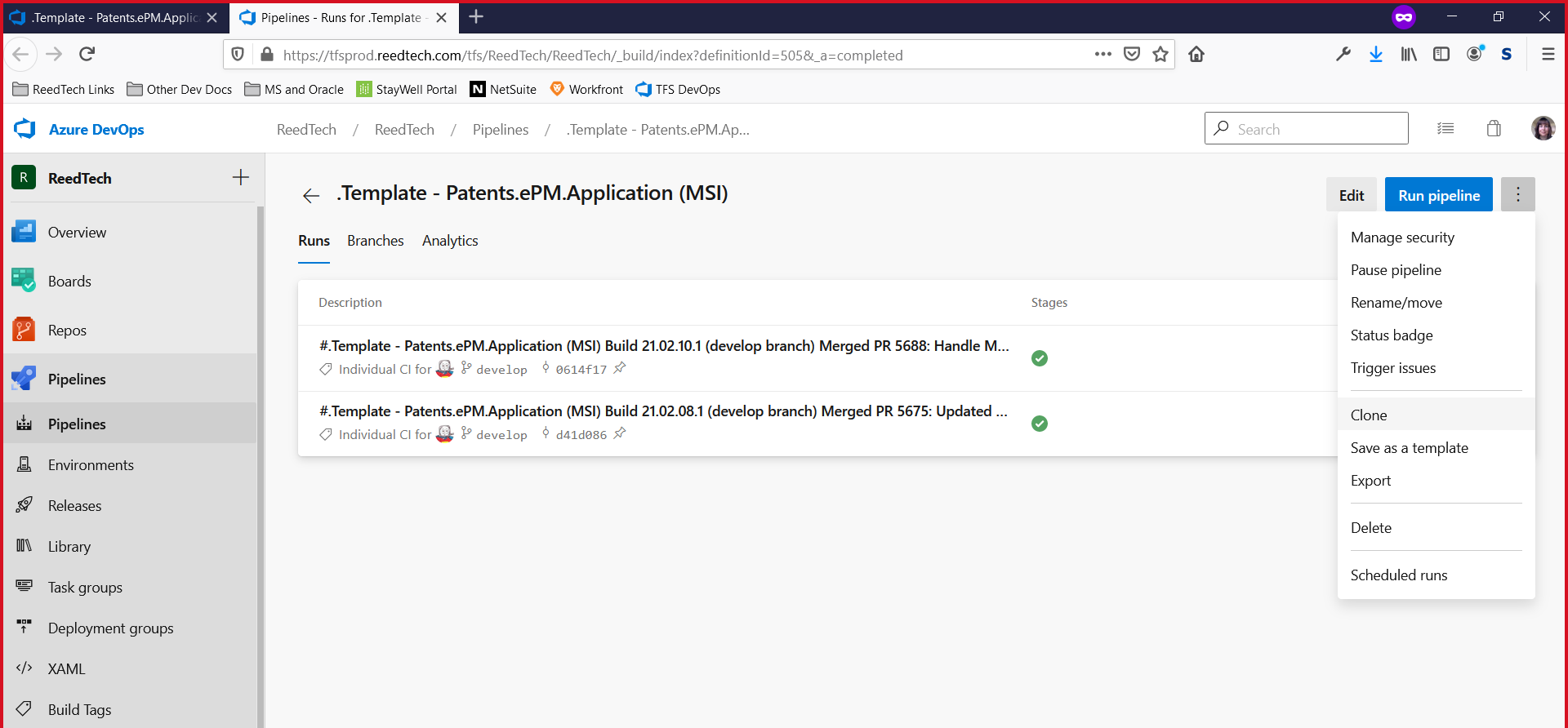
[Troubleshoot Pipeline Run Issues](https://docs.microsoft.com/en-us/azure/devops/pipelines/troubleshooting/troubleshooting?view=azure-devops)

**Desktop Application using Build Template (DON'T DO THIS)!**

1. Be sure to have the AutomationTools submodule included in the project(s) to be built.

[Add AutomationTools submodule to Repository](https://tfsprod.reedtech.com/tfs/ReedTech/ReedTech/_wiki/wikis/ReedTech.wiki/446/Adding-a-Submodule-to-Your-Git-Repository)

1. Clone the build template: <https://tfsprod.reedtech.com/tfs/ReedTech/ReedTech/_apps/hub/ms.vss-ciworkflow.build-ci-hub?_a=edit-build-definition&id=505>



1. Rename the clone to the same as the repo name for the project (Ex: Patents.FEP.fepSCOREBuild), and then click Save. Save it in the default Patents\Git Projects.
   * If you get an error about CheckmarxService, disable the Security Scans step.
2. Click on "Get Sources" at the top and change the repository selection to the appropriate repo. Make sure the other stuff there looks good, especially that "checkout submodules" is checked.
3. Make sure the info in the other tabs and steps all look appropriate.
4. Click Save and Queue to run the build. Fix any errors as necessary, saving changes made to the build.
5. Subsequent builds of the same project can then simply use the build pipeline that was created!

**Installing Sourcetree**

Follow

1

Edit



Carnes, Terry (RT-PHL)

Jul 6, 2020

**Installing Sourcetree Notes**

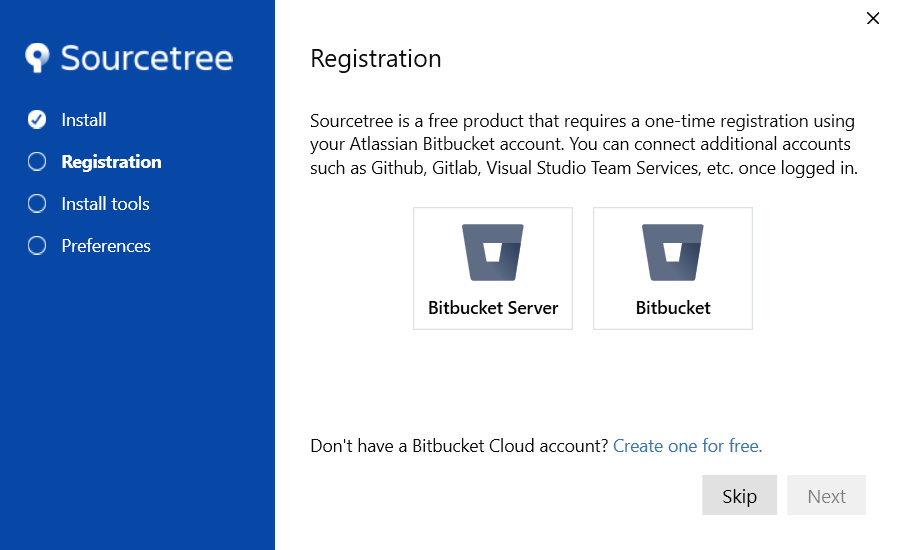
1. **General**

a. [Software and Tool Availability Wiki](https://tfsprod.reedtech.com/tfs/ReedTech/ReedTech/_wiki/wikis/ReedTech.wiki?pagePath=%2FReed%20Tech%20Developer%20Wiki%2FSoftware%20Development%20Department%2FSoftware%20and%20Tool%20Availability&wikiVersion=GBwikiMaster&_a=edit&pageId=452)

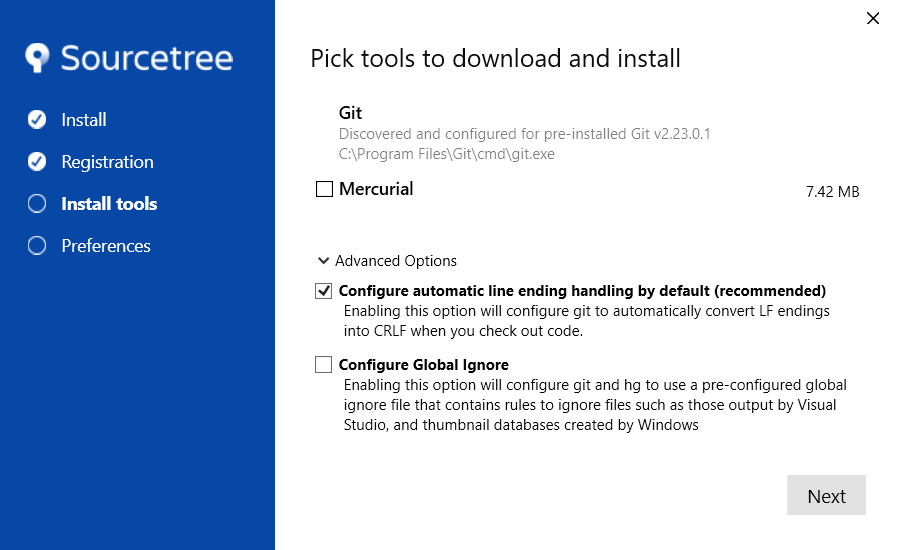
b. [Software and Tool Approval Requests Wiki](https://tfsprod.reedtech.com/tfs/ReedTech/ReedTech/_wiki/wikis/ReedTech.wiki?pagePath=%2FReed%20Tech%20Developer%20Wiki%2FSoftware%20Development%20Department%2FSoftware%20and%20Tool%20Approval%20Requests&wikiVersion=GBwikiMaster&pageId=188)

1. **Installing Sourcetree**

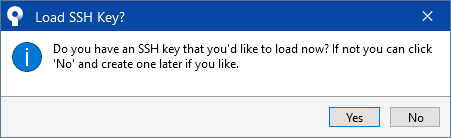
a. **Register/Sign-In for an Atlassian Bitbucket account –** During the installation process, you will be asked to either sign-in or register for a free Atlassian Bitbucket account. This enables you to open the program after installation where you will then be able to hook up the the Reed Tech repos.



b. \*\*Options -\*\* Since Mercurial was not on our approved list, I deselected it, and after conferring with others, I checked Configure automatic line ending handling, and did not check Configure Global Ignore.

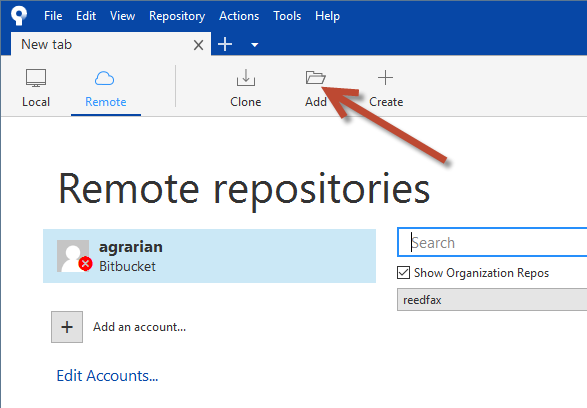


c. \*\*SSH Key -\*\* I answered “No” figuring I could always change it later.

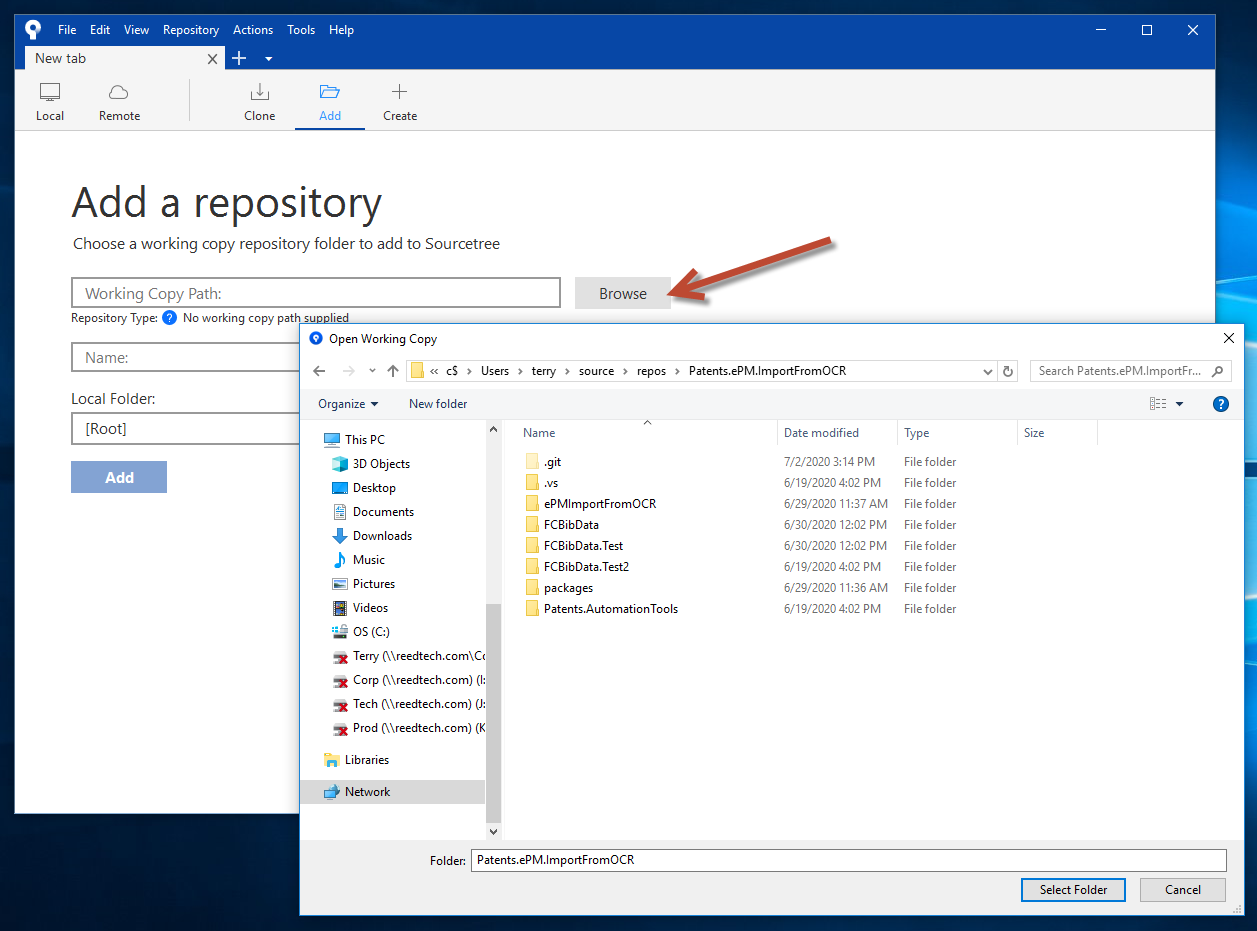


1. **Starting Sourcetree and Adding a Repository -** Once installation is finished you can start up Sourcetree. (This is assuming that you already have a repository cloned onto your local drive.)

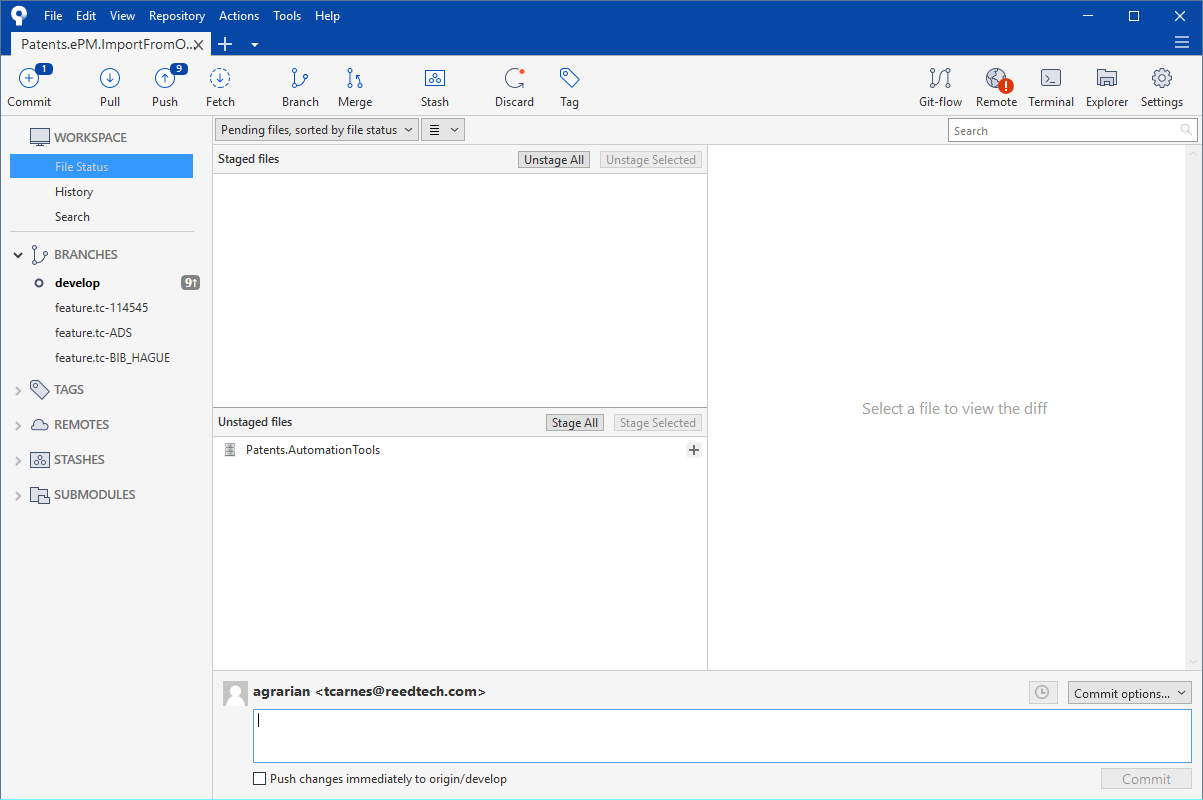
a. Click “Add” to add a repository



b. In the Add a repository dialog, Click “Browse” and browse to the local folder where your repository resides. Then click “Select Folder” and “Add” and you are all done!



1. This will take you back to the main screen and you are ready to start using Sourcetree!



**Adding a Submodule to Your Git Repository**

https://tfsprod.reedtech.com/tfs/ReedTech/_apis/GraphProfile/MemberAvatars/win.Uy0xLTUtMjEtNzM4ODcyMzUyLTIwODU1MTQ0OTQtNzUxODU5MzgzLTM0OTkx

Morin, Nikita (RT-PHL)

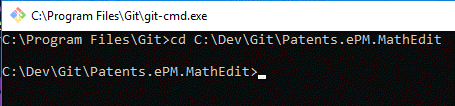
Jul 2, 2020

One nice use for a Git submodule is for our build pipelines to bring down a centralized repository for automation tools. We use several PowerShell scripts to perform a variety of functions in our builds. We store these in a separate repository under

<https://tfsprod.reedtech.com/tfs/ReedTech/ReedTech/_git/Patents.AutomationTools>

Thanks to **@Morin, Nikita (RT-PHL)** and the [Internet](https://www.benday.com/2016/11/04/one-tfs-build-multiple-git-repositories-with-submodules/) , here are steps to incorporate a submodule for use with your build pipeline.

1. Launch your Git command window.
2. Navigate to where you want to clone your project. For example, I want to clone Patents.ePM.MathEdit to my local folder C:\Dev\Git\Patents.ePM.MathEdit



1. Clone your repository here. You can either do this through Visual Studio or with the Git command

git clone <project\_repository\_url>

For example,

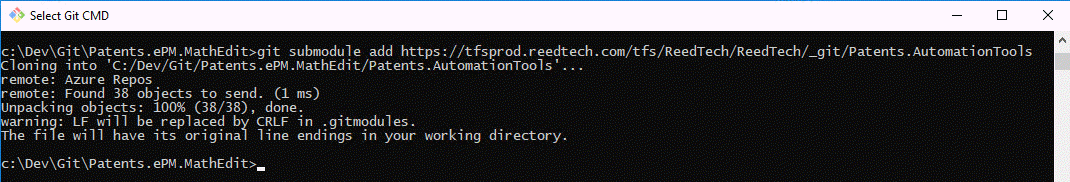
git clone https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/\_git/Patents.ePM.MathEdit

1. Add the submodule to this local cloned repository. Use the Git command

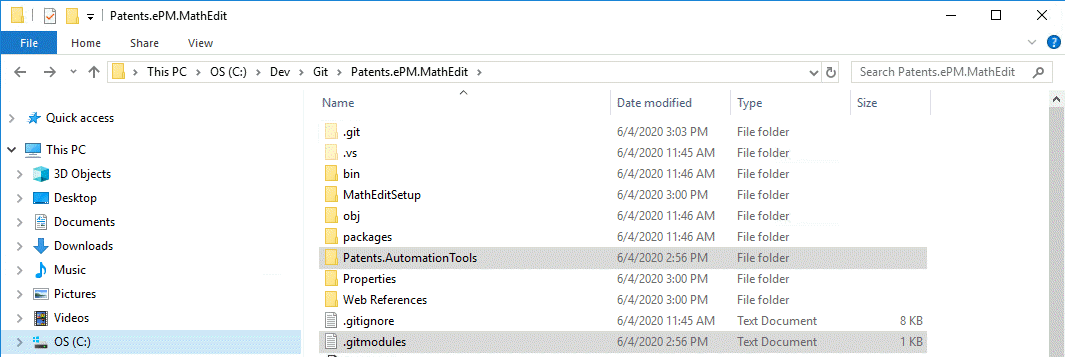
git submodule add <submodule\_repository\_url>

For example

git submodule add https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/\_git/Patents.AutomationTools



Now you will see in your local repo files:



The .gitmodule file is nothing more than a quick reference to identify the submodule:

[submodule "Patents.AutomationTools"]

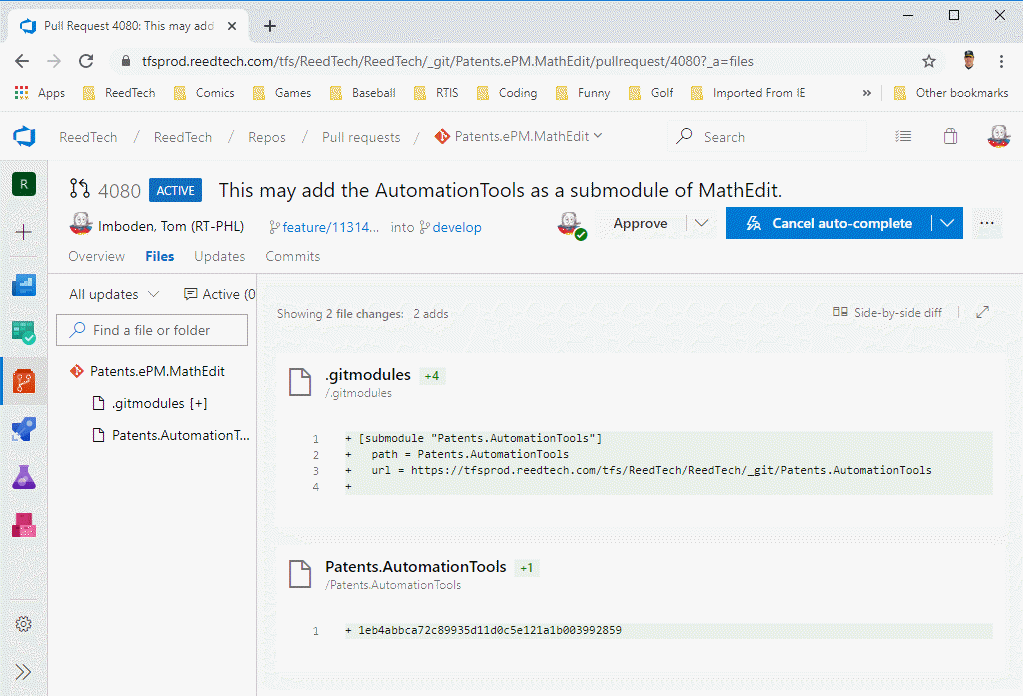
path = Patents.AutomationTools

url = https://tfsprod.reedtech.com/tfs/ReedTech/ReedTech/\_git/Patents.AutomationTools

1. Create a branch from develop, stage the changes, push the branch, and make a pull request to eventually complete this commit to the remote develop branch.

For example, here was the pull request for adding the AutomationTools submodule to Patents.ePM.MathEdit:

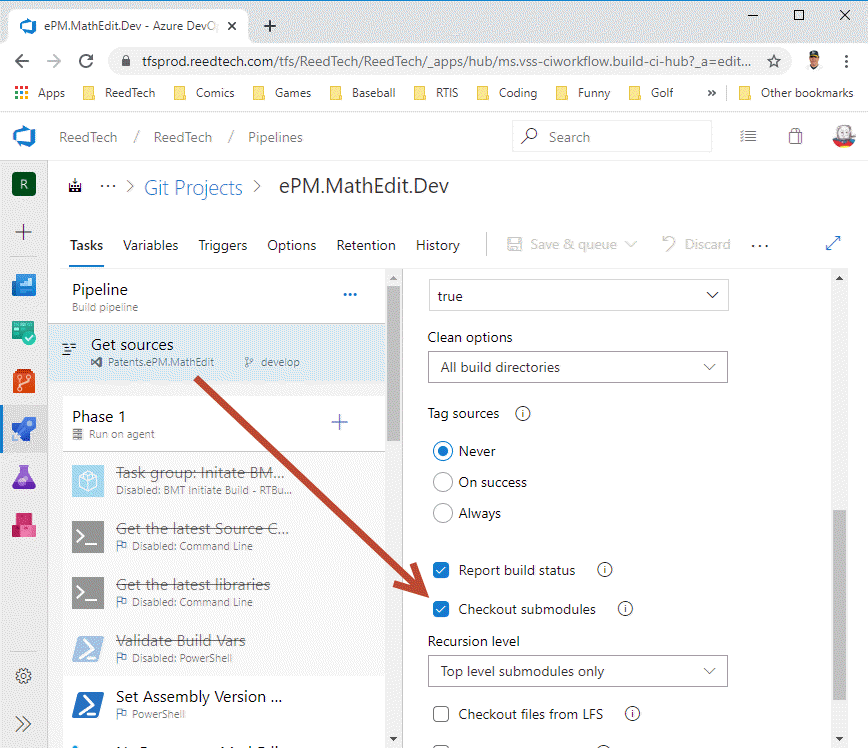
<https://tfsprod.reedtech.com/tfs/ReedTech/ReedTech/_git/Patents.ePM.MathEdit/pullrequest/4080?_a=files>



1. Check the "Checkout submodules" checkbox on your project's build pipeline. This is under the "Get sources" task.

For example, to finish with the MathEdit build:

<https://tfsprod.reedtech.com/tfs/ReedTech/ReedTech/_apps/hub/ms.vss-ciworkflow.build-ci-hub?_a=edit-build-definition&id=472>

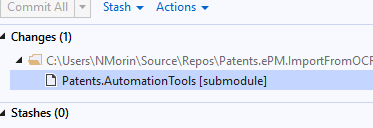


**Updating Submodules**

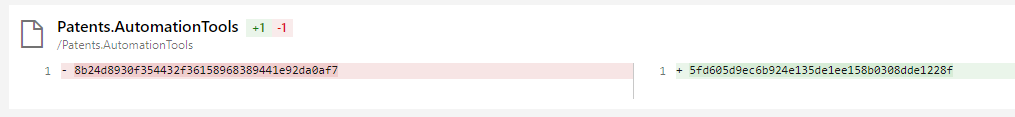
When there is a new script that is needed from a submodule such as Patents.AutomationTools in a project that has already added the submodule previously, the reference to the submodule must be updated.

1. Launch your Git command window.
2. Navigate to where the .gitmodules file is located in the solution.
3. Run this command: git submodule update --remote --merge

This should update the reference to the submodule's latest commit.  
It will appear as a pending change, which can then be committed and pushed and merged to the remote branches.



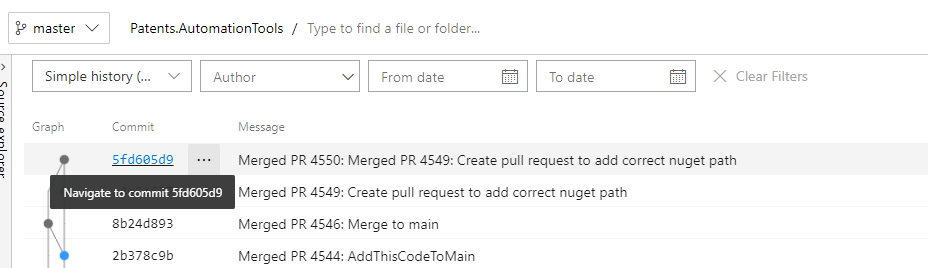
Once it is successfully in the Azure Devops Git repo, this change will appear as follows:



The SHA-1 hash that is updated is the Git commit id of the submodule's latest commit.

Notice that the first 8 characters match the latest commit ID in the submodule.

5fd605d9ec6b924e135de1ee158b0308dde1228f



**Overwrite branch with another branch**



Spotts, Stan (RT-PHL)

May 29, 2020

Sometimes you follow a different path and end up with the develop branch being more complete and current than master. You can overwrite it using these commands:

git checkout develop

git merge -s ours master

git checkout master

git merge develop

This happens with your local branches, of course. But to push it to the remote (DevOps Server) repo you will need to be able to override the policy that requires a pull request so you can push it to the remote. This is a special security setting that only the architects and specific people that the development managers have authorized for it.

**git Workshop**



Spotts, Stan (RT-PHL)

Mar 20, 2019

This workshop walks through the basic usage of Visual Studio and git when working on a project. It relies on a repo called **gitWorkshop** on the dev DevOps Server.

[Reed Tech git Workshop script](https://tfsprod.reedtech.com/tfs/reedtech/c81424e9-cee7-459e-ac5a-c7d20cb4ea09/_apis/git/repositories/803c21fb-03dd-45af-8f19-bf7f1d4f1cf8/Items?path=%2F.attachments%2FReed%20Tech%20git%20Workshop-777d7093-14d2-4a56-8ffc-cec81de383cf.docx&download=false&resolveLfs=true&%24format=octetStream&api-version=5.0-preview.1&sanitize=true&versionDescriptor.version=wikiMaster)

### Resetting the workshop repo (Admin)

Unlike "real" code repos, this one does not have branch policies. This is to allow simple resetting of the repo. The state of the repo must be reset after completing a workshop. The steps for a DevOps administrator to do this are:

Go to administrative CMD prompt.  
CD to folder with **gitWorkshop** repo.

git checkout develop

git pull

git checkout --orphan newGitWorkshop

Start Visual Studio and open the gitWorkshop solution. Remove all .cs files except the ***workshop.cs*** file.  
Save the soluton and exit Visual Studio.

Go to administrative CMD prompt.

git add -A

git commit -am "Initializing git workshop repo"

git branch -D develop

git branch -m develop

git push -f origin develop

The repo will then be ready for the next workshop.

**Useful git tricks**



Imboden, Tom (RT-PHL)

Jul 30

**Contents**

* [Links](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/233/Useful-git-tricks#links)
* [Disable cases-sensitivity](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/233/Useful-git-tricks#disable-cases-sensitivity)
* [Authentication Failed when trying to access git repos](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/233/Useful-git-tricks#authentication-failed-when-trying-to-access-git-repos)
* [Create feature branch from User Story or Bug work item](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/233/Useful-git-tricks#create-feature-branch-from-user-story-or-bug-work-item)
* [Pull specific commit(s) into another branch](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/233/Useful-git-tricks#pull-specific-commit(s)-into-another-branch)
* [Modify last commit](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/233/Useful-git-tricks#modify-last-commit)
* [Undo last commit](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/233/Useful-git-tricks#undo-last-commit)
* [Modify last several commits](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/233/Useful-git-tricks#modify-last-several-commits)
* [Remote and local develop out of sync](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/233/Useful-git-tricks#remote-and-local-develop-out-of-sync)
* [Stale local references to remote branches](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/233/Useful-git-tricks#stale-local-references-to-remote-branches)
* [Add a new project to a solution in an existing repo](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/233/Useful-git-tricks#add-a-new-project-to-a-solution-in-an-existing-repo)
* [Process to create a new repo](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/233/Useful-git-tricks#process-to-create-a-new-repo)
  + [Create a new repo (Administrative)](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/233/Useful-git-tricks#create-a-new-repo-(administrative))
    - [Repo for as-yet-to-be-created solution](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/233/Useful-git-tricks#repo-for-as-yet-to-be-created-solution)
    - [Repo for existing solution not yet in a repo](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/233/Useful-git-tricks#repo-for-existing-solution-not-yet-in-a-repo)
* [Clone a remote repo to local workspace](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/233/Useful-git-tricks#clone-a-remote-repo-to-local-workspace)
* [Changed code in branch shows up when checking out other branch](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/233/Useful-git-tricks#changed-code-in-branch-shows-up-when-checking-out-other-branch)
* [Determine remote path for existing local repo](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/233/Useful-git-tricks#determine-remote-path-for-existing-local-repo)
* [Split repo into multiple repos by subfolder](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/233/Useful-git-tricks#split-repo-into-multiple-repos-by-subfolder)
* [Copy specific folder path from different repo into current repo](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/233/Useful-git-tricks#copy-specific-folder-path-from-different-repo-into-current-repo)
* [Determine remote path for existing local repo](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/233/Useful-git-tricks#determine-remote-path-for-existing-local-repo)
* [Tagging a repository before removing it](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/233/Useful-git-tricks#tagging-a-repository-before-removing-it)

# Links

* [On Undoing, Fixing, or Removing Commits in git](http://sethrobertson.github.io/GitFixUm/fixup.html)

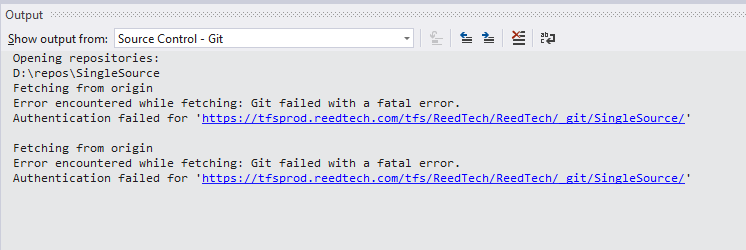
# Disable cases-sensitivity

If you change the case of a filename and check it in, git will by default see two different files. Git started out on Linux, which has a case-sensitive file system. To disable case sensitivity on Windows, go to the command line and type:

git config --global core.ignorecase true

# Authentication Failed when trying to access git repos

If you see an error in your Output window similar to this:



You may have the OpenSSL channel configured instead of the Windows Secure Channel protocol.

Type:  
git config --system -l

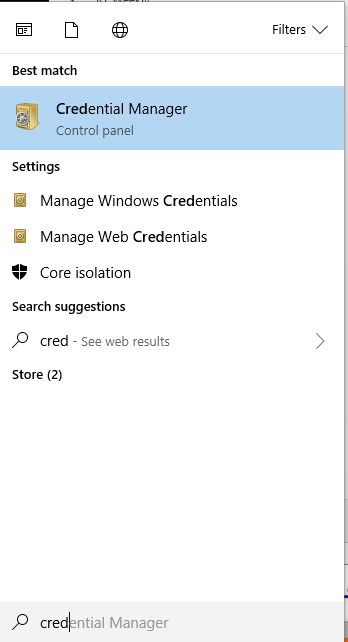
If you see the sslbackend element under http set to OpenSSL, type:

git config --system http.sslbackend schannel

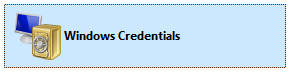
If this doesn't fix your issue, then your stored credentials may be stale. To fix, click the magnifying glass in the lower left of your screen, next to the Windows icon:

**C:\Users\ghan\Downloads\Pic\02.png**

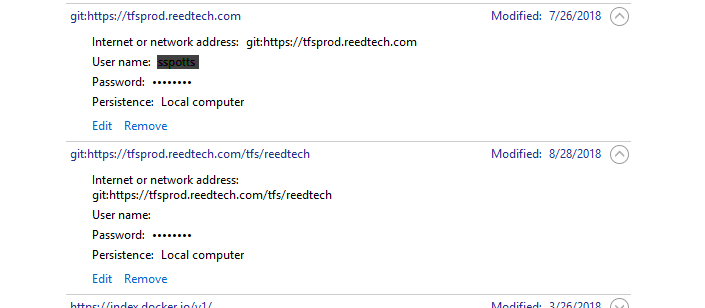
Type **cred** and hit [Enter] when you see Credential Manager.



When Credential Manager opens, click **Windows Credentials**.



Scroll down to see the git:[https://tfsprod.reedtech.com](https://tfsprod.reedtech.com/) entry. There may be more than one, as some may have longer paths.

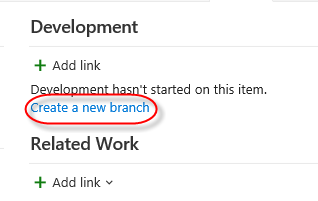


Click **Edit** and change the password for your account. Once you save this, you can exit, and you should not see the authentication failed error again.

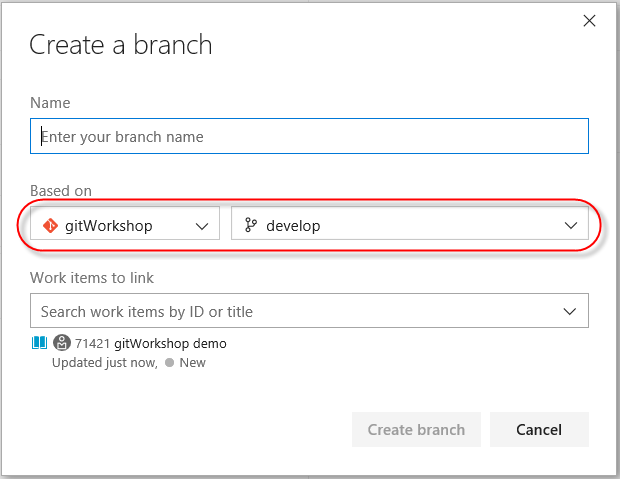
[Back to Top](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/233/Useful-git-tricks###)

# Create feature branch from User Story or Bug work item

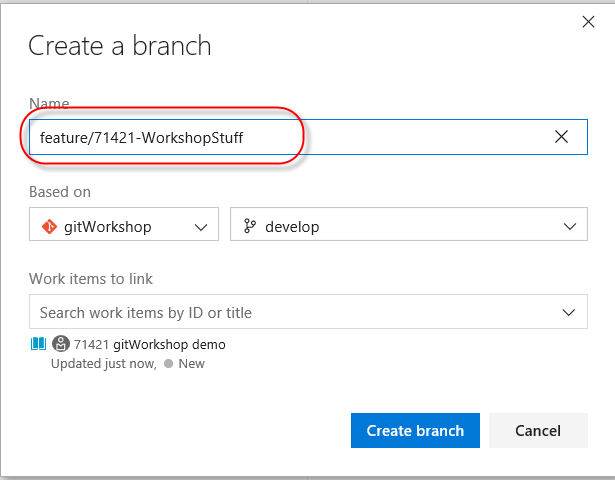
You can create a branch on a repo right from the work item you're going to work on. On the work item, under the Development area, click on **Create a new branch**.



On the create a branch form, click the drop down to select the repo you want to create a branch from. Regardless of the repo, you should select the **develop** branch as the source branch.

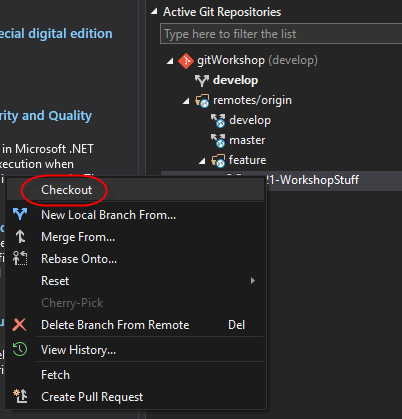


Enter the name of the feature ("feature/{work item id}-title", remember, no spaces). Note that the work item is already set to link to the branch, and you can link to other work items if needed. This helps when you're done with development for this story and are making a pull request.



Click the **Create Branch** button.

Now you have a feature branch on the server and the browser interface will open to it. The next step is to get it on your workstation. Refresh the repo by clicking the refresh icon ( ) on the **Branches** pane of **Team Explorer**. You'll see the feature under remotes/origin. Right click on the feature branch and click **Checkout** on the context menu.



You can also use the command line and simply go to the repo's folder on your hard drive, then click:

checkout develop

git fetch

git checkout feature/{work item id}-title

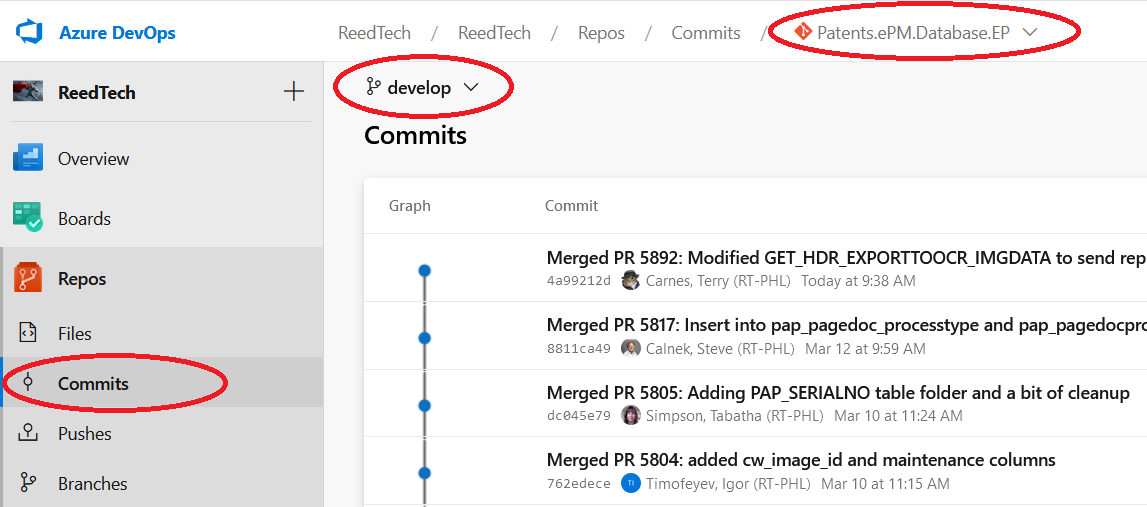
and it will pull down the branch as a local branch, ready for you to work in.

[Back to Top](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/233/Useful-git-tricks#Index)

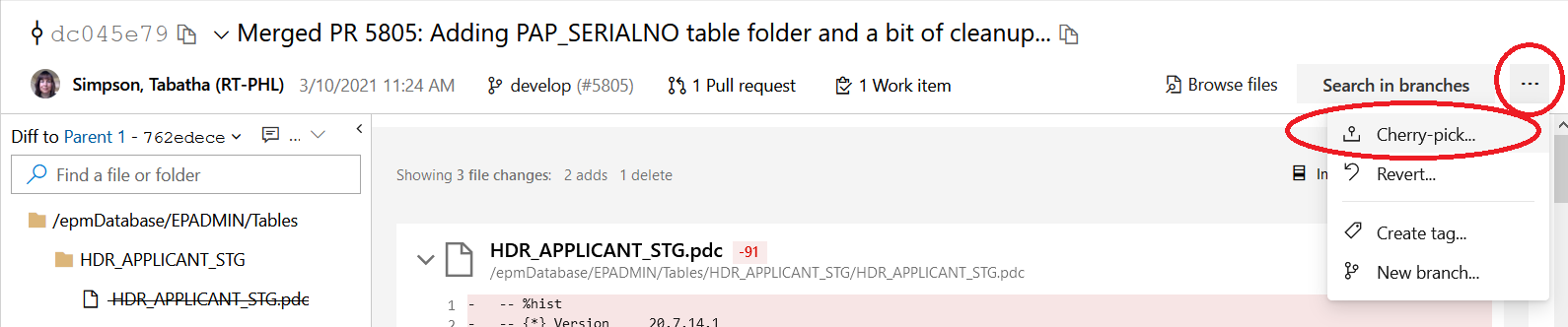
# Pull specific commit(s) into another branch

Say several developers have been pulling their local branches into 'Develop' for the same repository for various changes. But then you need to specifically move your changes from 'Develop' into 'Master' without including other developers' changes.

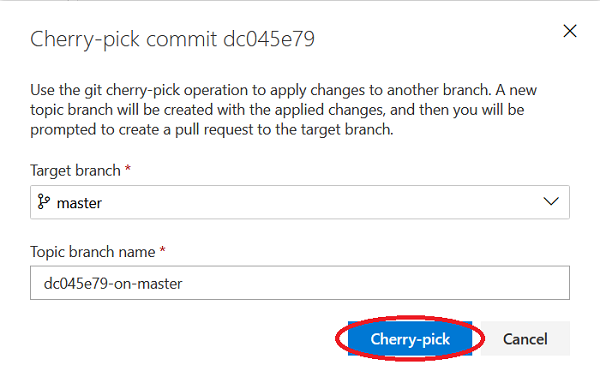
1. In DevOps, go to "Repos" > "Commits". Select the repository you want, and the "develop" branch. This gives you the list of commits made to the develop branch:



1. Click on the commit you want to pull (or one of them). In the page that opens, click the ellipses button, then "Cherry-pick...":



1. In the box that pops up, select the "master" branch in the dropdown. It automatically gives the the source "topic branch" a name using the commit. If this is the only commit to pull, then the default is fine...if not, give it a meaningful name. Then click "Cherry-pick":



1. When the cherry-pick is complete, another box pops up to say so. If this is the last commmit to be included, click on "Create Pull Request" and go through the pull request...if not, click "Close".
2. If you have more commits to add to the pull request, repeat steps 2 - 4 for the additional commits, making sure to enter the same "Topic branch name" for each.

[Back to Top](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/233/Useful-git-tricks#Index)

# Modify last commit

Sometimes you'll make a commit and need to change it. Perhaps you made misspellings, or need to add more to the description. Maybe you need to remove files you mistakenly included, or add some you forgot about.

The following command will open an editor (vi by default, but you can set it to notepad++ or notepad easily), showing the last commit message. Edit and save, and it will replace the commit with the new message.

git commit --amend

You can stage changes and add them to the previous commit by using add or rm (remove) commands prior to the commit amend.

git add file5.cs file6.cs src\file7.cs

git rm file3.cs

git commit --amend

[Back to Top](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/233/Useful-git-tricks#Index)

# Undo last commit

Sometimes you'll make a commit to a local repo and need to completely undo it. You can fix this at the command line.

This will undo the last commit but keep the changes:

git reset HEAD~1

This will undo the last commit and discard the changes:

git reset --hard HEAD~1

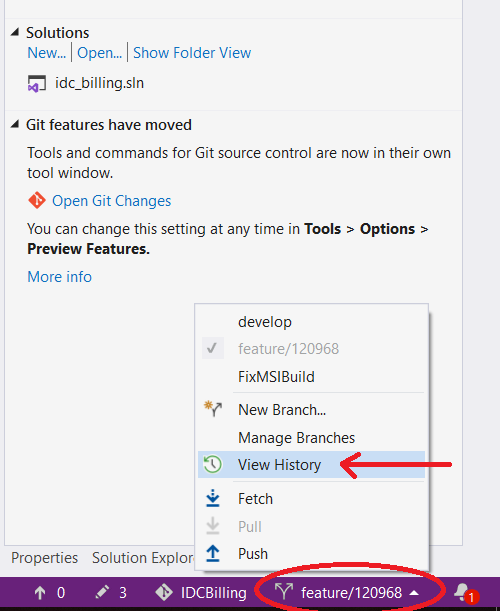
Yes, ~2 will undo last two commits, etc.

If you already did a pull request and need to reset a commit to a remote (TFS) repo, follow the local steps then type:

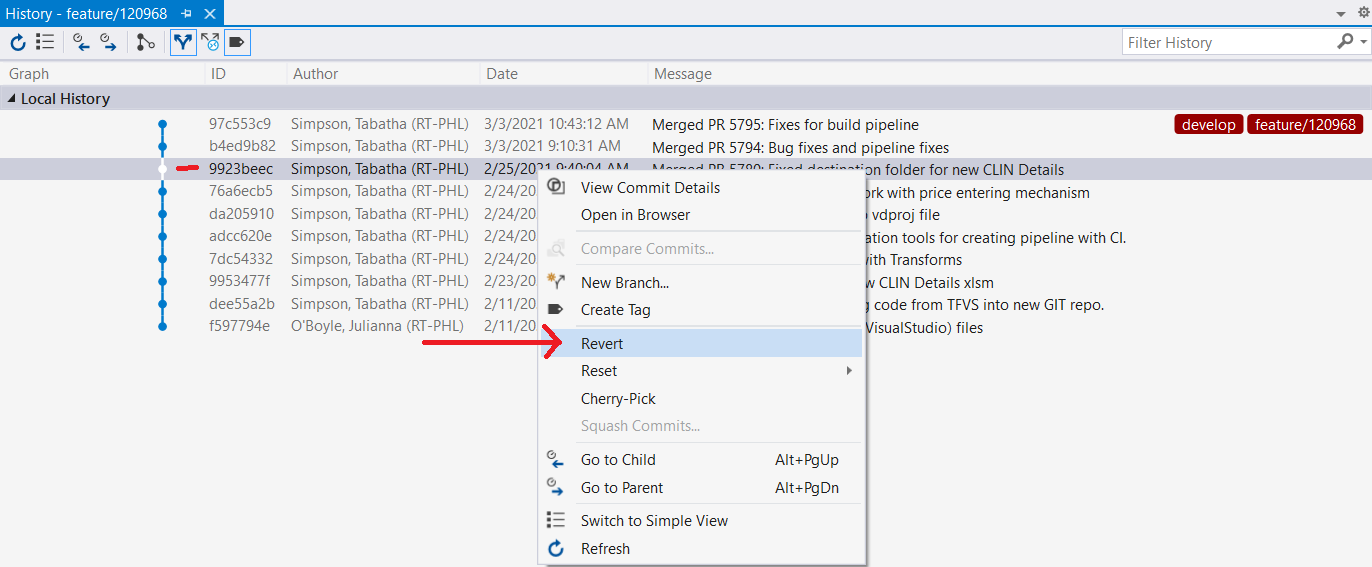
git push --force

You can also revert commits within Visual Studio....

Click on your branch at the bottom and go to View History:



Find the commit you want to revert TO. For example, if you want to undo the last 2 commits, select the third from last commit. Right-click it and then click "Revert":



This will create a commit for undoing those commits.

Just note the difference between Revert and Reset is Reset will actually remove those commits as opposed to creating an "undo" commit, altering the commit history. Therefore Revert is safer.

[Back to Top](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/233/Useful-git-tricks#Index)

# Modify last several commits

Use git rebase to modify your last several commits. You can change commit messages, squash (combine) commits together, and more. You may want to do this to change your local commit history to make it easier to remove a change made with multiple commits by making it one. You may want to cherry-pick a commit to apply to another branch. You may want to add a user story reference to the messages of several commits.

For example, you want to rewrite the commit messages for one of your last three commits.

git rebase -I HEAD~3

HEAD is the top of your commit history. Adding the "~3" refers to the HEAD and 2 previous commits. The "-i" indicates that you want an interactive, i.e. editor, session.

You'll see something like this, with commits listed from older to newer:

pick 3a80e1b So much fun

pick fb5c14c Merged PR 7: So much fun

pick 099349b Merged PR 8: another test

# Rebase 87688f4..a6d5c65 onto 87688f4 (3 commands)

#

# Commands:

# p, pick <commit> = use commit

# r, reword <commit> = use commit, but edit the commit message

# e, edit <commit> = use commit, but stop for amending

# s, squash <commit> = use commit, but meld into previous commit

# f, fixup <commit> = like "squash", but discard this commit's log message

# x, exec <command> = run command (the rest of the line) using shell

# d, drop <commit> = remove commit

# l, label <label> = label current HEAD with a name

# t, reset <label> = reset HEAD to a label

# m, merge [-C <commit> | -c <commit>] <label> [# <oneline>]

# . create a merge commit using the original merge commit's

# . message (or the oneline, if no original merge commit was

# . specified). Use -c <commit> to reword the commit message.

#

# These lines can be re-ordered; they are executed from top to bottom.

#

# If you remove a line here THAT COMMIT WILL BE LOST.

#

# However, if you remove everything, the rebase will be aborted.

#

# Note that empty commits are commented out

git is nice enough to show you a list of actions you can take on all the commits. You can edit the items at the top that all start with the "pick" command. So if you wanted to change the last commit message and combine the first and second into one with a new message, you could change them to this:

r 3a80e1b So much fun

pick fb5c14c Merged PR 7: So much fun

s 099349b Merged PR 8: another test

The editor will close then reopen with the commit message you want to change:

So much fun

# Please enter the commit message for your changes. Lines starting

# with '#' will be ignored, and an empty message aborts the commit.

#

# Date: Fri Oct 19 09:57:19 2018 -0400

#

# interactive rebase in progress; onto 87688f4

# Last command done (1 command done):

# reword 3a80e1b So much fun

# Next commands to do (2 remaining commands):

# squash fb5c14c Merged PR 7: So much fun

# squash 099349b Merged PR 8: another test

# You are currently editing a commit during a rebase.

#

# Changes to be committed:

# new file: MyNewFile.cs

#

After you change the message and save and exit the editor, it will open again, this time with the two other commit messages.

# This is a combination of 2 commits.

# This is the 1st commit message:

So much fun

# This is the commit message \#2:

Merged PR 8: another test

another test

# Please enter the commit message for your changes. Lines starting

# with '#' will be ignored, and an empty message aborts the commit.

#

# Date: Fri Oct 19 09:57:19 2018 -0400

#

# interactive rebase in progress; onto 87688f4

# Last commands done (2 commands done):

# squash 099349b Merged PR 8: another test

# No commands remaining.

# You are currently rebasing.

Again, edit the messages and save. It will squash the two commits into one and use the message you changed.

[Back to Top](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/233/Useful-git-tricks#Index)

# Remote and local develop out of sync

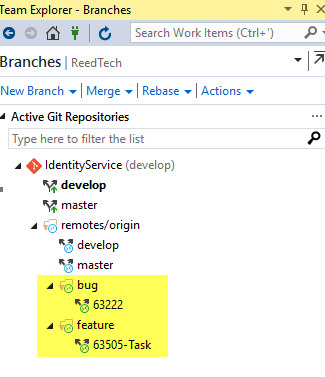
In our workflow, the remote copy of develop should be the "of record" repo while your local copy of develop is a copy. So reset your local develop to match the remote one.

git reset --hard origin/develop

[Back to Top](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/233/Useful-git-tricks#Index)

# Stale local references to remote branches

If you pushed a feature, bug, or hotfix to TFS, and the branch was deleted (maybe automatically after a pull request was completed), and you still see the feature/bug/hotfix branch in your view of remote/origin, like this:



You can use "prune" to remove these stale local references to the remote branches from the command line. This can be done using:

git remote prune origin

or, if you also want to pull down any changes to your tracking branches,

git fetch -p

Or you can delete local branches in Visual Studio simply by right-clicking on a branch and selecting "delete".

[Back to Top](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/233/Useful-git-tricks#Index)

# Add a new project to a solution in an existing repo

If you have a solution in an existing repo and need to add a project to it, you'll add it to your local branch then do a pull request to promote it to the server repo.

1. Create a new feature branch. You can use the "feature/{user story}" name format.
2. Add the new project to the solution while you have this branch checked out.
3. Create a pull request for this branch.
4. When the pull request is approved and the new project is in the server develop repo, checkout the develop branch and do a fetch.
5. Delete the local feature branch you used for the pull request.

[Back to Top](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/233/Useful-git-tricks#Index)

# Process to create a new repo

The person who creates the repo may not be the person using it to write code. In some cases an Architect or a Lead Developer may create the repo for a new project for developers. So this is split into two parts, one for creating the repo on the server, and the other for cloning it locally.

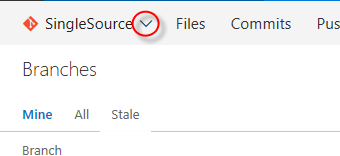
## Create a new repo (Administrative)

This is currently done by TFS administrators to ensure constinstancy.

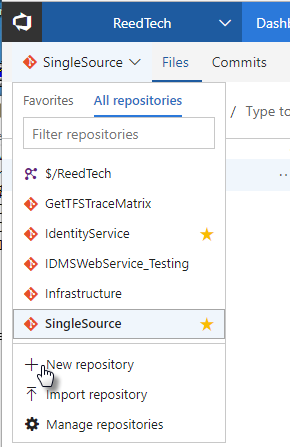
Developers will follow the instructions at [Clone a remote repo to local workspace](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/233/Useful-git-tricks#Clone-a-remote-repo-to-local-workspace) to use the repo.

Typically, we equate a repo with a solution, with projects under the solution.

Browse to TFS and click on the Code link in the top toolbar. Click the downward arrow to the left of Files to open the repo list.



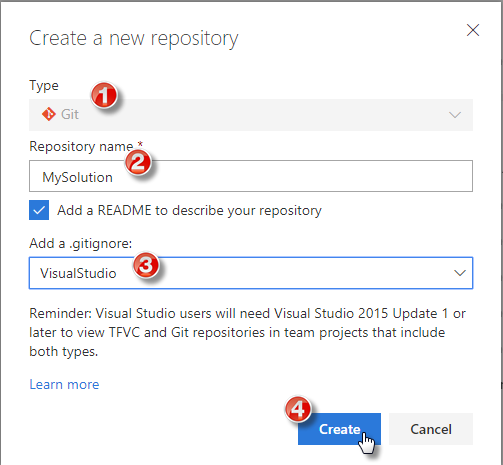
Then click on the New repository link.



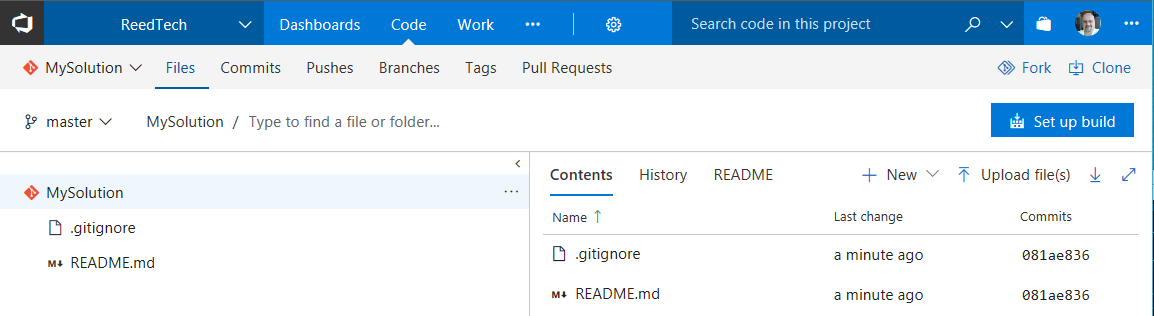
### Repo for as-yet-to-be-created solution

Make sure (1) git is the selected type, (2) you name your repo, (3) you add the .gitignore file for Visual Studio, and if you wish you can check the box to add a README to your repo. The README file is useful for information a developer will need to work on the project, such as a list of tools or controls to be installed. See **Repo for existing solution not yet in a repo** below to add a solution you've already got on your workstation that isn't yet in source control.

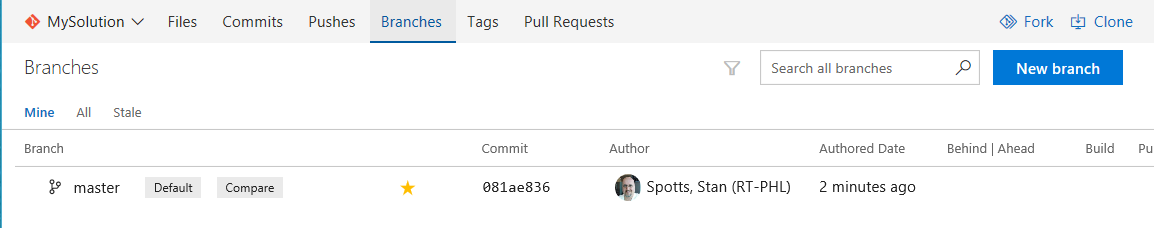
Click the Create button.



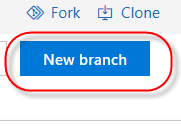
You'll now have a repo with a master branch in TFS. A new page will display in the browser showing the new repo info.



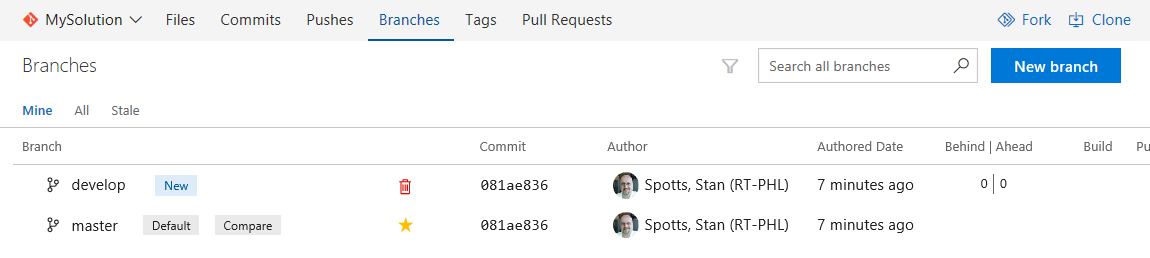
Click the Branches menu option to the right of the Files option. You'll see the default **master** branch.



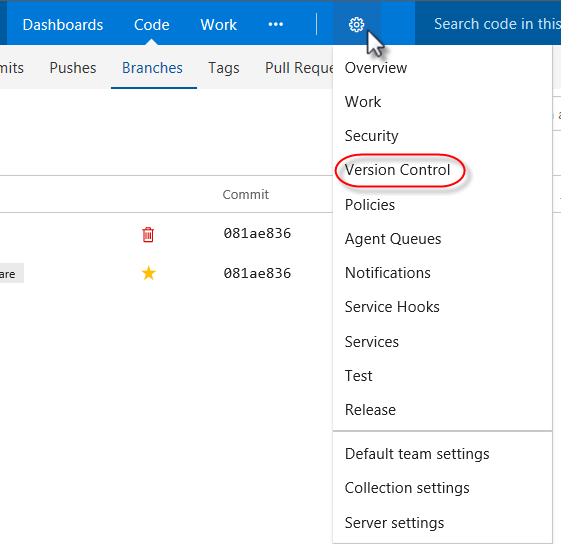
We need to add the develop branch and make it the default branch. Click the **New Branch** button.



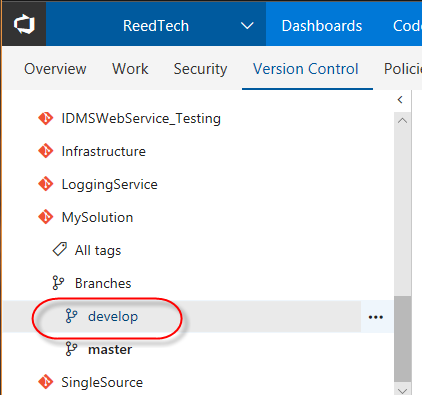
Type "develop" in the **Name** field and click the **Create branch** button. We now see a master and a develop branch.



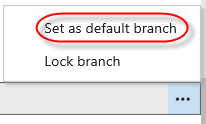
However, we want **develop** to be the default branch. Mouse hover over the gears icon to the right of **Dashboards** and click on **Version Control** from the drop down menu.

****

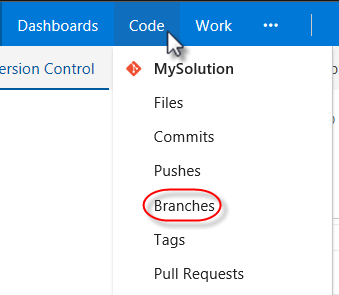
Find your repo and expand it, then expand **Branches** and click on the **develop** branch.

****

Click on the ellipses to the right of the **develop** branch. Click **Set as default branch** from the context menu.



Go back to the repo page by mouse hovering over the **Code** menu option to the right of **Dashboards** and clicking **Branches** from the drop down menu.

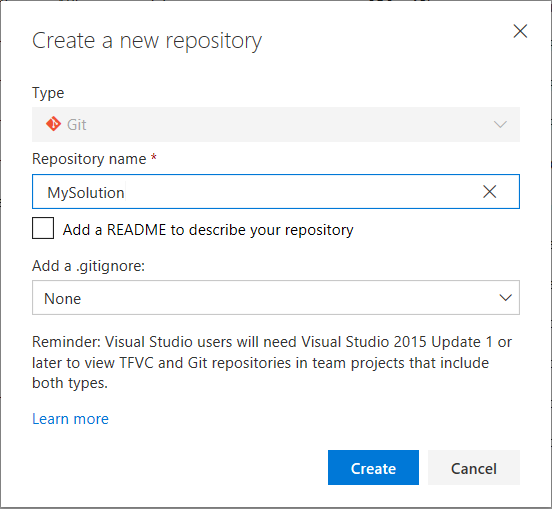


You'll see that **develop** is now the default repo. At this point, the TFS administrator has to be asked to set the policies for the two branches.

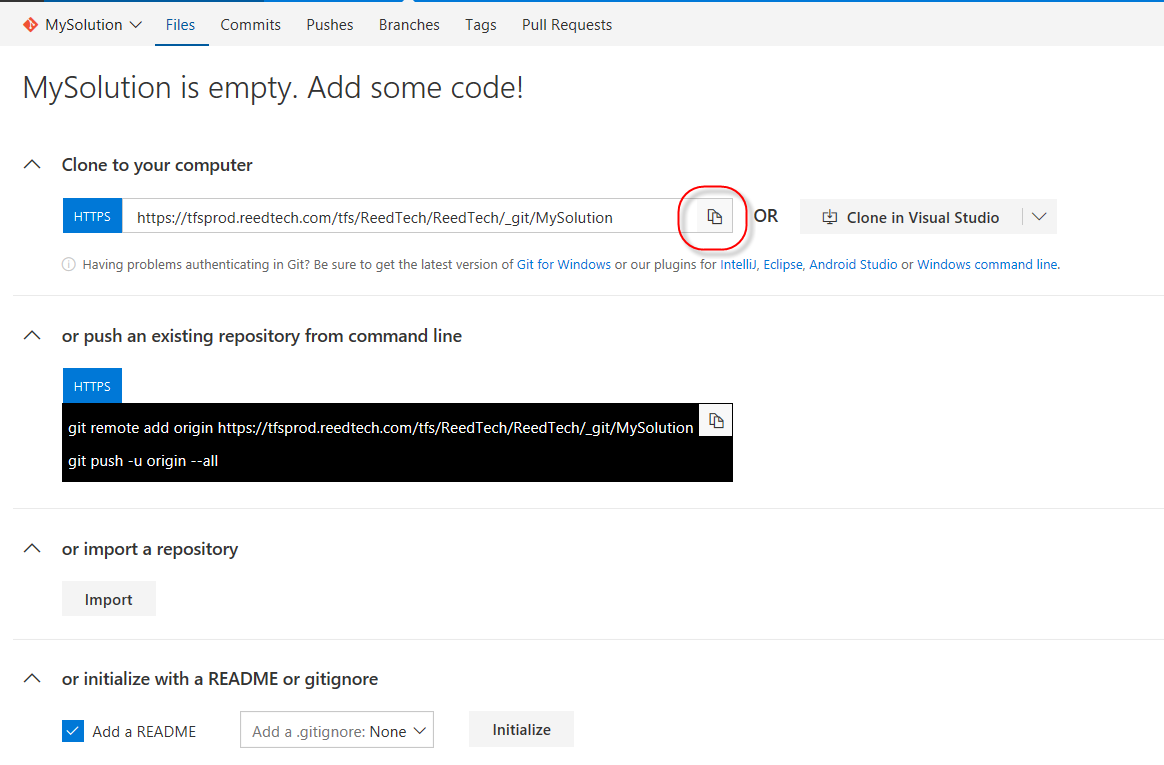
### Repo for existing solution not yet in a repo

If you do not tell TFS to add a README file and add a .gitignore file, the repo will be created but there will be no branches created. This is useful if you already have a solution written and want to use it to create a new repo.

Click the Create button.



You now have an empty repo. A new page will display in the browser showing the new repo info. Click on the little copy icon to the right of the url under "Clone to your computer."



Open a command prompt and navigate to the folder that contains your git repos. From another repo, copy a .gitignore file to this folder. Create a README.md file in notepad and save it. If you want to get more points, add information to the README.md file that other developers can use to see what they have to do to work with, deploy, and test this solution.

Now you need to initialize this as a git workspace, stage your files, and commit.

Type:

git init

git add -A

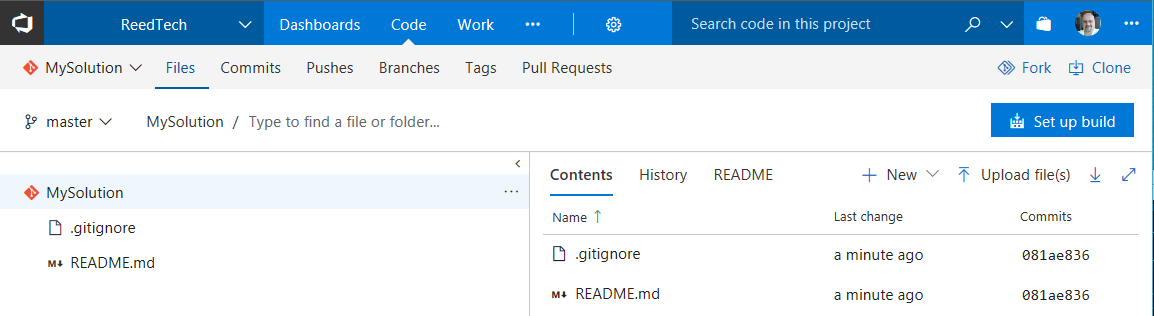
git commit -m"Initial commit"

Great! We have a local repo all configured. Now we need to push this to the remote. Type **git push** and then paste the url you just copied, and append to that **master:master**. This will create the remote master branch on the empty repo you created on TFS.

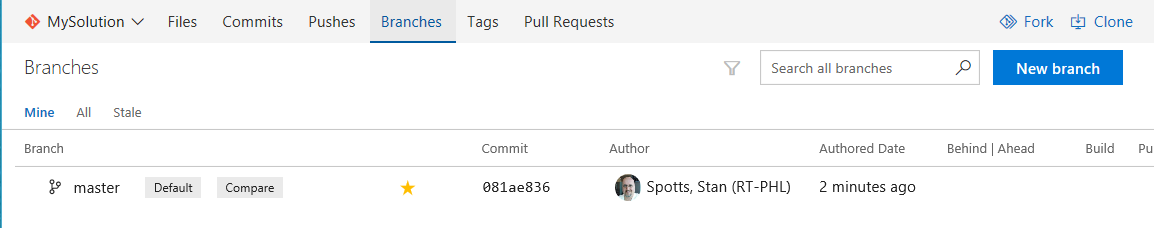
E.g.,  
git push https://tfsprod.reedtech.com/tfs/ReedTech/ReedTech/\_git/MySolution master:master

If you get a validation/permission error and you've already fixed your credentials, try replacing [**https://tfsprod.reedtech.com**](https://tfsprod.reedtech.com/) with [**http://tfs:8080**](http://tfs:8080/) and it should work.

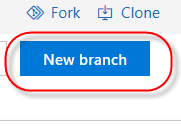
You'll now have a repo with a master branch in TFS. In your browser, on the page where you saw the empty repo, refresh the page. You'll see that you've got "stuff" in the repo now.



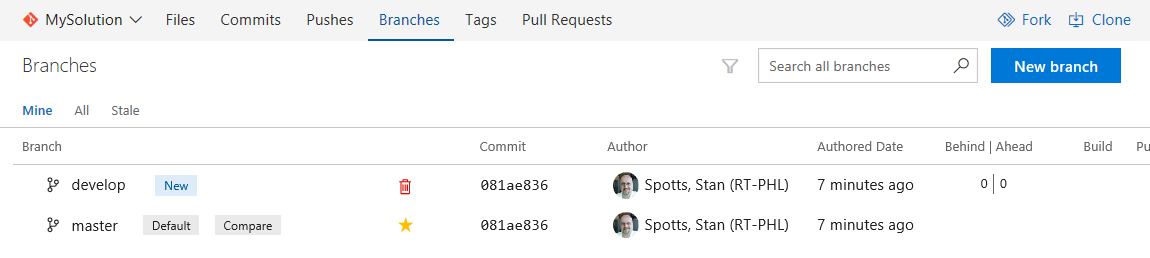
Click the Branches menu option to the right of the Files option. You'll see the default **master** branch.



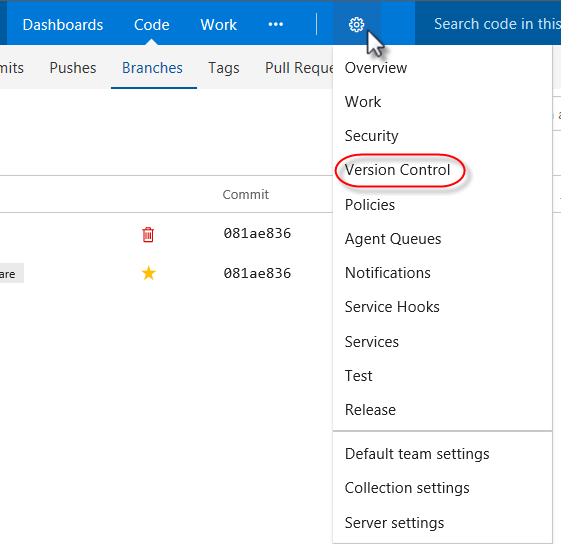
We need to add the develop branch and make it the default branch. Click the **New Branch** button.



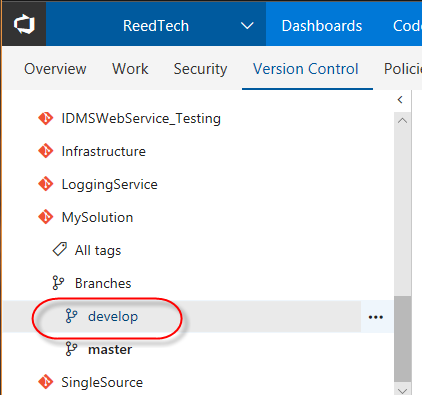
Type "develop" in the **Name** field and click the **Create branch** button. We now see a master and a develop branch.

****

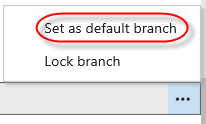
However, we want **develop** to be the default branch. Mouse hover over the gears icon to the right of **Dashboards** and click on **Version Control** from the drop down menu.



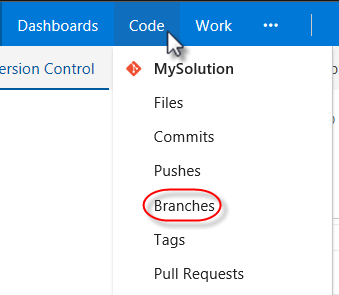
Find your repo and expand it, then expand **Branches** and click on the **develop** branch.



Click on the ellipses to the right of the **develop** branch. Click **Set as default branch** from the context menu.



Go back to the repo page by mouse hovering over the **Code** menu option to the right of **Dashboards** and clicking **Branches** from the drop down menu.



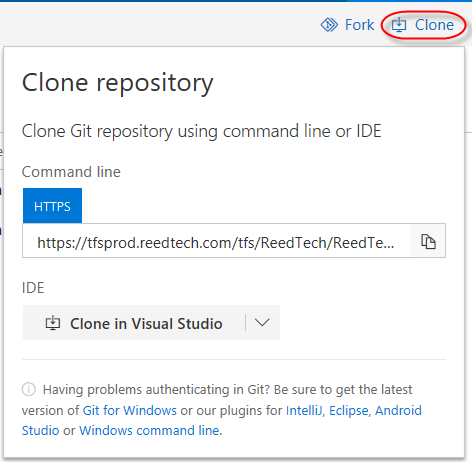
You'll see that **develop** is now the default repo. At this point, the TFS administrator has to be asked to set the policies for the two branches.

[Back to Top](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/233/Useful-git-tricks#Index)

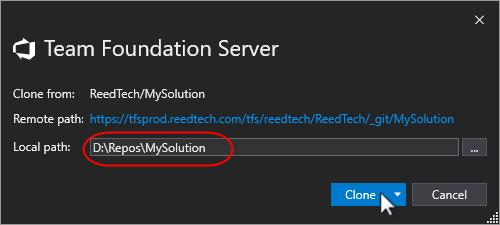
# Clone a remote repo to local workspace

To work with a remote repo, you need to clone it to your local workspace. This is what developers will do for any existing repo on TFS that they want to work on.

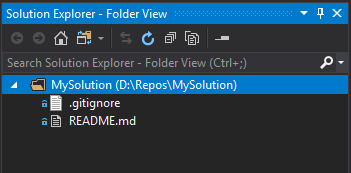
Click the Clone link on the top right corner of the page, then click the **Clone in Visual Studio** button.



You may be asked if you mean to switch apps to open Visual Studio, so answer **yes**. A new instance of Visual Studio will open, and you'll be shown a dialog box to perform the cloning. Change your local path if necessary, and click the **Clone** button.



Solution Explorer will show the solution with the .gitignore and README.md files.

****

You will see at the right on the status line that the **develop** branch is checked out.

C:\Users\ghan\Downloads\Pic\39.png

You may now create a feature branch and add projects and files and create a pull request to update the server branch per the Reed Tech git workflow.

If you would rather use the command-line to clone the repo, copy the link from the Clone repository dialog box and do the following:

1. Open an administrative command prompt.
2. Change to the folder that you want to clone the repo under.
3. Type **git clone** then paste the repo's url, then hit [Enter].
4. Type **git branch -A** to see your develop branch and remote branches.

[Back to Top](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/233/Useful-git-tricks#Index)

# Changed code in branch shows up when checking out other branch

If you changed code in a file in a branch, and then switched branches, the code change follows you to the other branch.

Checking out a branch only reflects what's been committed. If you made changes in a branch, before checking out another branch, either commit the changes or use the command-line **git stash** command to store them.

The git stash command is useful because you can pick up where you left off when you go back to your original working branch. You simply checkout the branch, then use the command-line **git stash apply** to restore your uncommitted, working code.

[Back to Top](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/233/Useful-git-tricks#Index)

# Determine remote path for existing local repo

In case you forget what the url to the remote/origin repo is for the current local repo

git remote get-url origin

[Back to Top](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/233/Useful-git-tricks#Index)

# Split repo into multiple repos by subfolder

This is all done at the command prompt, as Visual Studio doesn't have this level of git tooling. What we need to do is to clone the current repo from origin/remote, then keep only the structure we want.

git clone -b develop --single-branch https://tfsprod.reedtech.com/tfs/reedtech/reedtech/\_git/\_original repo\_ \_new repo\_

E.g.:  
git clone -b develop --single-branch https://tfsprod.reedtech.com/tfs/reedtech/reedtech/\_git/SingleSource LS.SS.ChannelService

Change directory to the folder for the new branch.

E.g.  
cd d:/repos/LS.SS.ChannelService

Per RT repo standard practices, the default branch is "develop." You need to ensure you're in the develop for the next step, so use this to check:

git branch

You should see \* develop" If it's "\* master," stop now and get Stan. Now we use a handy - but potentially dangerous - git command and filter. Note there is a space between "--" and "develop" and not between "--" and "subdirectory." Also make sure that you use forward slashes ("/") and a final slash on the folder. If you use backslashes ("\") you'll find that your folder is empty.

git filter-branch --prune-empty --subdirectory-filter \_folder to keep for new repo\_ -- develop

E.g.  
git filter-branch --prune-empty --subdirectory-filter src/Services/ChannelService/ -- develop

Type "dir" to see what you have in your root folder. The filter-branch directive will have moved your subfolder to the top of the structure. You can copy a solution file from the original repo or just create a new one for these folders. If you copy, you'll want to edit the solution file in a text editor and change the folders to match the new structure.

For instance, you might see this in the original .sln file:

Project("{2150E333-8FDC-42A3-9474-1A3956D46DE8}") = "src", "src", "{785474E9-7255-4C34-B5A6-C28556A83C4C}"

EndProject

Project("{2150E333-8FDC-42A3-9474-1A3956D46DE8}") = "Services", "Services", "{57FD1AAE-9552-4585-AA32-1092B62DD7C4}"

EndProject

Project("{2150E333-8FDC-42A3-9474-1A3956D46DE8}") = "ChannelService", "ChannelService", "{3D21DB1F-1C92-45AF-AC34-32CC31D1CC8D}"

EndProject

Project("{9A19103F-16F7-4668-BE54-9A1E7A4F7556}") = "ChannelService.Api", "src\Services\ChannelService\ChannelService.Api\ChannelService.Api.csproj", "{D54C4B67-8F53-424C-9C45-6E7B61EA34AA}"

EndProject

Project("{9A19103F-16F7-4668-BE54-9A1E7A4F7556}") = "ChannelService.Infrastructure", "src\Services\ChannelService\ChannelService.Infrastructure\ChannelService.Infrastructure.csproj", "{1D3A2415-BACD-4546-923D-DECF70FEB69F}"

EndProject

Project("{9A19103F-16F7-4668-BE54-9A1E7A4F7556}") = "ChannelService.Domain", "src\Services\ChannelService\ChannelService.Domain\ChannelService.Domain.csproj", "{EFC55D66-D0AF-4416-9AEC-92A91B971129}"

EndProject

Project("{FAE04EC0-301F-11D3-BF4B-00C04F79EFBC}") = "ChannelService.Api.Tests", "src\Services\ChannelService\ChannelService.Api.Tests\ChannelService.Api.Tests.csproj", "{E97695CA-A813-4F67-989B-8ACEEB3B7041}"

EndProject

You would have to remove the src and services pointers, plus the one for the next level folder for the old location, and change the four pointers to the .csproj files, like so:

Project("{9A19103F-16F7-4668-BE54-9A1E7A4F7556}") = "ChannelService.Api", "ChannelService.Api\ChannelService.Api.csproj", "{D54C4B67-8F53-424C-9C45-6E7B61EA34AA}"

EndProject

Project("{9A19103F-16F7-4668-BE54-9A1E7A4F7556}") = "ChannelService.Infrastructure", "ChannelService.Infrastructure\ChannelService.Infrastructure.csproj", "{1D3A2415-BACD-4546-923D-DECF70FEB69F}"

EndProject

Project("{9A19103F-16F7-4668-BE54-9A1E7A4F7556}") = "ChannelService.Domain", "ChannelService.Domain\ChannelService.Domain.csproj", "{EFC55D66-D0AF-4416-9AEC-92A91B971129}"

EndProject

Project("{FAE04EC0-301F-11D3-BF4B-00C04F79EFBC}") = "ChannelService.Api.Tests", "ChannelService.Api.Tests\ChannelService.Api.Tests.csproj", "{E97695CA-A813-4F67-989B-8ACEEB3B7041}"

EndProject

Remove the line items under the section GlobalSection(ProjectConfigurationPlatforms) = postSolution that have the same guid as the pointers you just removed, and do the same under the GlobalSection(NestedProjects) = preSolution section so you aren't referencing missing folders.

Also copy the **.gitignore**, **.gitattributes**, and **README.md** file from the original repo to the top level of the new repo. It would be a good idea to edit the README.md file and add any information a developer will need to know to begin working on the repo.

Add the files to the repo with:

git add -A

git commit -m"Refactored from \_original repo\_"

Once this is all complete, we need to push this to origin/remote to create the new remote develop branch.

git push https://tfsprod.reedtech.com/tfs/reedtech/reedtech/\_git/\_remote repo name\_ develop:develop

E.g.:

git push https://tfsprod.reedtech.com/tfs/reedtech/reedtech/\_git/LS.SS.ChannelService develop:develop

This creates sets up your current code as the develop branch in origin/remote.

git remote set-url origin https://tfsprod.reedtech.com/tfs/reedtech/reedtech/\_git/\_new repo\_

E.g.:  
git remote set-url origin https://tfsprod.reedtech.com/tfs/reedtech/reedtech/\_git/LS.SS.ChannelService

Verify this was done with:

git remote -v

Notify Stan you got this far so he can create the master branch, set it as the default branch, and set the branch policies.

[Back to Top](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/233/Useful-git-tricks#Index)

# Copy specific folder path from different repo into current repo

This is all done at the command prompt, as Visual Studio doesn't have this level of git tooling. What we need to do is to clone the repo that has the folder you want from origin/remote, then merge that into the current repo. The first part is similar to [Split repo into multiple repos by subfolder](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/233/Useful-git-tricks#Split-repo-into-multiple-repos-by-subfolder), but adds additional special new fetch, merge, and commit directives.

First, we need to clone the repo that contains the folder we want. We'll pull down the whole develop branch of the repo.

git clone -b develop --single-branch https://tfsprod.reedtech.com/tfs/reedtech/reedtech/\_git/\_original repo\_ \_new repo\_

E.g.:  
git clone -b develop --single-branch https://tfsprod.reedtech.com/tfs/reedtech/reedtech/\_git/SingleSource temp\_repo

Change directory to the folder for this temp repo.

E.g.  
cd d:/repos/temp\_repo

Per RT repo standard practices, the default branch is "develop." You need to ensure you're in the develop for the next step, so use this to check:

git branch

You should see \* develop" If it's "\* master," stop now and get Stan. Now we use a handy - but potentially dangerous - git command and filter. Note there is a space between "--" and "develop" and not between "--" and "subdirectory." Also make sure that you use forward slashes ("/") and a final slash on the folder. If you use backslashes ("\") you'll find that your folder is empty.

git filter-branch --prune-empty --subdirectory-filter \_folder to keep for new repo\_ -- develop

E.g.,  
git filter-branch --prune-empty --subdirectory-filter src/Database/SingleSoruceDb/ -- develop

Type "dir" to see what you have in your root folder. The filter-branch directive will have moved your subfolder to the top of the structure. You can copy a solution file from the original repo or just create a new one for these folders. If you copy, you'll want to edit the solution file in a text editor and change the folders to match the new structure.

We'll set up this temp repo as a remote with:

git remote add temp ../\_new repo\_

E.g.,

git remote add temp ../temp\_repo

Now we'll do a fetch to get the history of the remote into the local .git folder.

git fetch \_temp repo\_

E.g.,

git fetch temp

We'll set up a merge point - don't fret that this fails, as it does what we need.

git merge -s ours --no-commit \_temp repo\_/develop

E.g.,

git merge -s ours --no-commit temp/develop

Note the **-s ours** specifies we want the "ours" merge strategy. This resolves any number of heads, but the resulting tree of the merge is always that of the current branch head, effectively ignoring all changes from all other branches. It is meant to be used to supersede old development history of side branches.

Now we'll get the folder added to our current repo. Make sure you are in the folder that you want the new folder under.

git read-tree --prefix=\_my\_new\_subfolder\_ -u \_temp repo\_/develop

E.g.,

git read-tree --prefix=FileMgmtServiceDB -u temp/develop

You can **git status** to see the staged changes. You may want to see that the new folder is in your local workspaces. If you want to edit things like project or solution files to reflect any structural changes, you can do that. Then we commit the new folder.

git commit -m"Refactored to include \_folder name\_ from \_source repo\_"

[Back to Top](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/233/Useful-git-tricks#Index)

# Determine remote path for existing local repo

In case you forget what the url to the remote/origin repo is for the current local repo

git remote get-url origin

[Back to Top](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/233/Useful-git-tricks#Index)

# Tagging a repository before removing it

If you want to remove a repository, but still be able to get it back from history, consider tagging the current repo before deleting and committing. You can search for tags and even restore by referencing them.

Not a VS-accessible command. Command line is:

git tag –a AngularCore –m “some message about superfluous folder/solution”

This is an annotated tag, and keep a lot of info in git. If you just want a lightweight tag that is just a pointer to the last commit this solution is in (if you do it just before you delete it then commit), just use:

git tag AngularCore

You can use whatever word you want if you don’t like AngularCore, of course.

If you already deleted it, just run git log and look for the commit where you deleted it, and add the first 7 characters of the previous commit checksum to the command.

To see the commits and checksums, run:

git log --pretty =oneline

You’ll see a list of checksums and commit messages. If the checksum before the commit was 2ea9873e8affba3013dd0a509190ac5489230055, you can add the tag using:

git tag AngularCore 2ea9873

Tags will show up in git log.

**git Merge conflict resolution**



Simpson, Tabatha (RT-PHL)

Jul 8

**Contents**

* [Configure git to use Visual Studio merge tool](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/236/git-Merge-conflict-resolution#configure-git-to-use-visual-studio-merge-tool)
* [The following untracked working tree files would be overwritten by merge](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/236/git-Merge-conflict-resolution#the-following-untracked-working-tree-files-would-be-overwritten-by-merge)
* [Fixing Merge Conflicts from Pull Requests](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/236/git-Merge-conflict-resolution#fixing-merge-conflicts-from-pull-requests)

**Configure git to use Visual Studio merge tool**

First, get to an administrative command prompt and type:

git config --system -e

This will open the system config for git. Cut and past the following into the file:

[diff]

tool = vsdiffmerge

[difftool]

prompt = true

[difftool "vsdiffmerge"]

cmd = \"C:\\Program Files (x86)\\Microsoft Visual Studio\\2017\\Enterprise\\Common7\\IDE\\CommonExtensions\\Microsoft\\TeamFoundation\\Team Explorer\\vsDiffMerge.exe\" \"$LOCAL\" \"$REMOTE\" //t

keepbackup = false

trustexistcode = true

[merge]

tool = vsdiffmerge

[mergetool]

prompt = true

[mergetool "vsdiffmerge"]

cmd = \"C:\\Program Files (x86)\\Microsoft Visual Studio\\2017\\Enterprise\\Common7\\IDE\\CommonExtensions\\Microsoft\\TeamFoundation\\Team Explorer\\vsDiffMerge.exe\" \"$REMOTE\" \"$LOCAL\" \"$BASE\" \"$MERGED\" //m

keepbackup = false

trustexistcode = true

You may need to change "Enterprise" in the "cmd=" lines to reflect a different version. Check your path.

Save this file. When you do a merge and encounter conflicts, you will see the diff/merge tool in visual studio.

**The following untracked working tree files would be overwritten by merge**

If you see this error, you probably have files that are tracked remotely that aren't tracked locally. Try this:

git add \*

git stash

git pull

If you want to get back the changes, use:

git stash apply

otherwise get rid of them with:

git stash clear

**Fixing Merge Conflicts from Pull Requests**

1. Open the repo in Visual Studio.
2. Do a Pull on the local 'develop' branch (or the source branch).
3. Make a local 'master' branch (or whatever the target branch is if you don't already have a local version that tracks the origin branch).
4. Right-click the source branch and choose the option to merge into the target branch.  
   NOTE: The option might say to merge it into itself (ex: Merge 'develop' into 'develop') - choose it anyway, and the confirmation box that pops up should say the correct target branch.
5. In the Git changes window, the merge conflicts should be listed. Right click on a file with a conflict and choose "Merge..." to open the merge conflict window and manually fix the merge.
6. Once all merge conflicts are fixed, commit the changes.
7. NOW...Git will likely not let you push the commits directly. So instead, create a new branch based on the target branch. What that should do is create the new branch where there is 1 outgoing commit, which is the fixed merge. Simply Push that branch, then do the Pull Request into the target branch.

**Import TFVC code into git**

Follow

0

Edit



Mondelblatt, Aaron (RT-PHL)

Aug 3

**Prep**

The code in TFVC should be cleaned up as much as possible prior to the import. There is no way to filter or select folders or files from the import. Some files should be deleted from TFVC prior to the import. You might not see any of these if the project has good .tfignore and nuget.config files.

There are some guidelines, use them to guide how you clean solution folders prior to import:

* The \debug folder and all folders under it (\obj, \bin) must be deleted.
  + If the solution has a Setup project that includes files from any such folder in the output, those files must be moved to a location under the main project that gets included in the repo, such as the "Resources" folder.
* The \packages folder contents that are downloaded nuget packages must be deleted. They will be recreated locally when you build.
* Any dll's or exe's must be deleted. These should be in nuget packages.
* In the Common/NetSuite folder path, there is an 8Gb References.cs file that says it was autogenerated. Large files that get generated during the build should be removed.
* Sisence .ecdata files should contain only schema, not data.

Note also that the SharedLib location used by Patents will *not* be imported. Any dll's here should be moved to nuget if they're still in use.

The person performing the import must ensure that they add a new [.gitignore](https://tfsprod.reedtech.com/tfs/reedtech/c81424e9-cee7-459e-ac5a-c7d20cb4ea09/_apis/git/repositories/803c21fb-03dd-45af-8f19-bf7f1d4f1cf8/Items?path=%2F.attachments%2Ft-39af82aa-1bba-43dd-9f13-157c01ae0515.txt&download=false&resolveLfs=true&%24format=octetStream&api-version=5.0-preview.1&sanitize=true&versionDescriptor.version=wikiMaster) file to the root of master prior to creating the develop branch. The attached file has a .txt extension that must be removed prior to adding to the master branch.

**Steps**

1. Clone the newly created repo
2. Create a feature branch (like "feature/new")
3. Copy the files from the TFS location into new feature branch using File Explorer
   * Repo folder should look something like this:

* If this project is a desktop app that includes a Setup project which generates .msi and/or .exe files, add to the .gitignore to ignore these.  
  Under "# Build results", it should look like this:  
  [Dd]ebug/  
  [Dd]ebugPublic/  
  [Rr]elease/  
  [Rr]eleases/  
  qac/  
  dev/  
  qat/  
  qav/  
  uat/  
  trn/  
  prod/  
  x64/  
  x86/  
  [Ww][Ii][Nn]32/  
  [Aa][Rr][Mm]/  
  [Aa][Rr][Mm]64/  
  bld/  
  [Bb]in/  
  [Oo]bj/  
  [Oo]ut/  
  [Ll]og/  
  [Ll]ogs/
* Make sure the repo gets organized according to our standardized [new project architecture](https://tfsprod.reedtech.com/tfs/ReedTech/ReedTech/_wiki/wikis/ReedTech.wiki/540/New-Project-Architecture).

1. Check changes into the GIT repo
2. Make sure the solution compiles / builds (checking in any changes made to get it working)
3. Push the changes and create a pull request
4. Once a reviewer has approved it, you can manually complete the request or auto-complete on approval.

**Using Git with Visual Studio**

Follow

0

Edit



Simpson, Tabatha (RT-PHL)

Jul 20

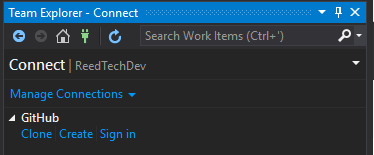
Visual Studio updates usually include additional Git support. However, the command line is more expressive. This documentation explains how to get repositories ("repos") onto your workstation and properly create branches for both new features and hotfixes; check in code; and manage your local repos following our variant of Gitflow (which we'll call "rtflow").

**Contents**

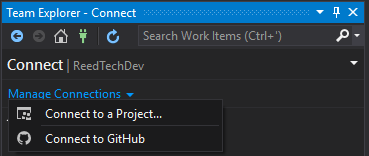
* [Get a copy of a repo from GIT](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/234/Using-Git-with-Visual-Studio#get-a-copy-of-a-repo-from-git)
* [Steps in making a code change](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/234/Using-Git-with-Visual-Studio#steps-in-making-a-code-change)
  + [Create a working branch for a user story or bug](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/234/Using-Git-with-Visual-Studio#create-a-working-branch-for-a-user-story-or-bug)
    - [Create a working branch for a hotfix](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/234/Using-Git-with-Visual-Studio#create-a-working-branch-for-a-hotfix)
  + [Change tracking and committing](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/234/Using-Git-with-Visual-Studio#change-tracking-and-committing)
  + [Updating the local development branch](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/234/Using-Git-with-Visual-Studio#updating-the-local-development-branch)
  + [Committing features and bugs to the TFS (origin) development branch](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/234/Using-Git-with-Visual-Studio#committing-features-and-bugs-to-the-tfs-(origin)-development-branch)
  + [Making changes to a Pull Request](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/234/Using-Git-with-Visual-Studio#making-changes-to-a-pull-request)
  + [Committing hotfixes to TFS Git](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/234/Using-Git-with-Visual-Studio#committing-hotfixes-to-tfs-git)
  + [Switching branch context for hotfix](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/234/Using-Git-with-Visual-Studio#switching-branch-context-for-hotfix)
* [Possible Issues and Their Fixes / Workarounds](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/234/Using-Git-with-Visual-Studio#possible-issues-and-their-fixes-%2F-workarounds)
  + [When attempting to modify a Setup project, you get error "The command you are attempting cannot be completed because the file '[Setup project file]' is under source code control and is not checked out."](https://tfsprod.reedtech.com/tfs/reedtech/ReedTech/_wiki/wikis/ReedTech.wiki/234/Using-Git-with-Visual-Studio#when-attempting-to-modify-a-setup-project%2C-you-get-error-%22the-command-you-are-attempting-cannot-be-completed-because-the-file-'%5Bsetup-project-file%5D'-is-under-source-code-control-and-is-not-checked-out.%22)

# Get a copy of a repo from GIT

From Visual Studio, you can connect to your repo. Go to Team Explorer and click on the Manage Connections icon (  ). Click on the Manage Connections link:

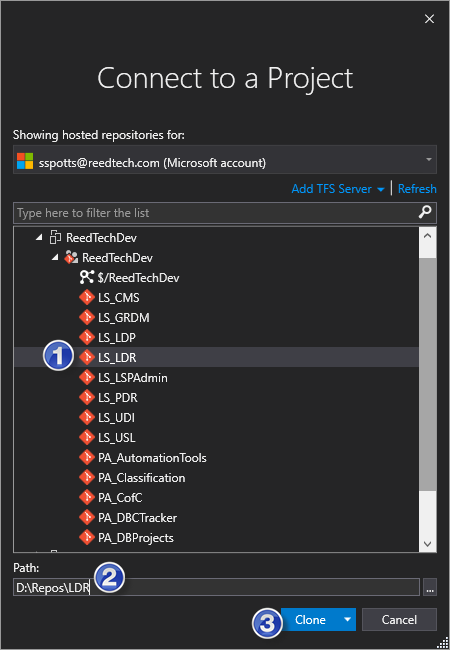


Click on Connect to a Project....

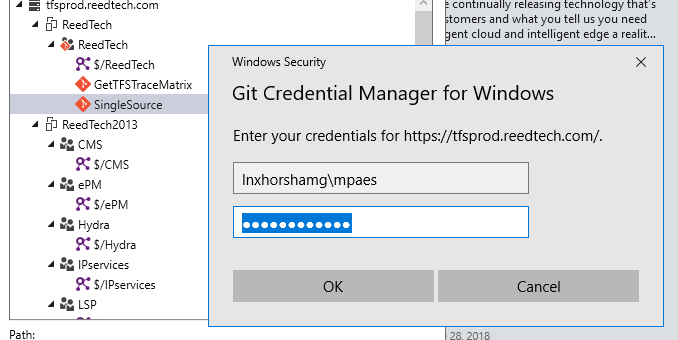


Traverse the hierarchy of repositories and click on the one you want to clone (1). Verify or change the path to where you want this repo cloned (2), then click the Clone button (3).

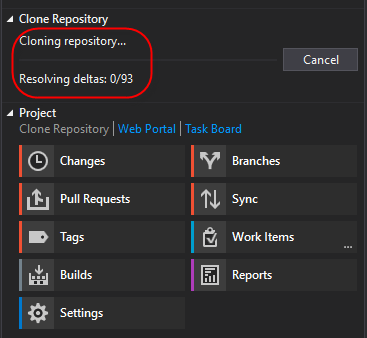
The new path (2) should **not** be the same as your old Team Foundation Version Control (TFVC) workspace path. The recommended path is d:\repos, if you have a d: drive. When IT has rolled out [work folders](https://docs.microsoft.com/en-us/windows-server/storage/work-folders/work-folders-overview)  you will add the entire repos path (e.g., d:\repos) folder structure as a work folder. This will ensure network sync of these local files.



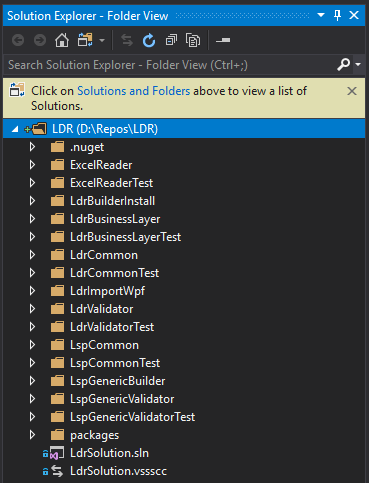
Enter your credentials (if needed).



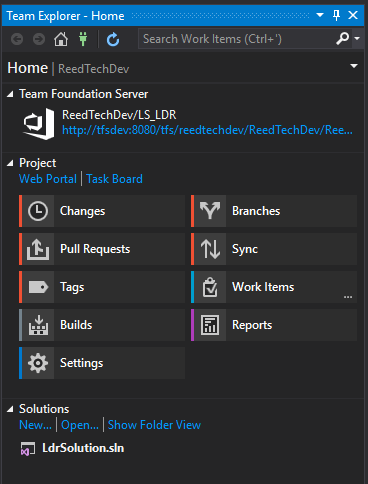
In the Team Explorer pane, you will see the status as the repository is being cloned locally.



The Solution Explorer pane will open when cloning is completed, showing your solution.



Return to the Team Explorer window to see the Git operations you can perform in Visual Studio.

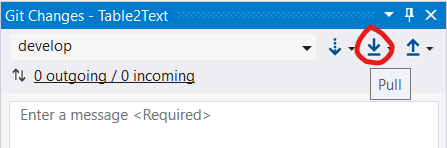


The default branch in "remote" (TFS) for all repos is **"develop"**.

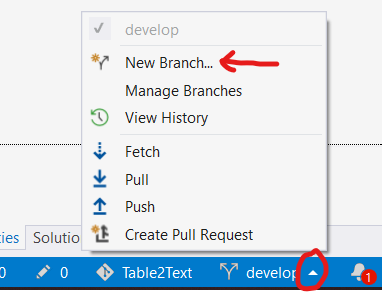
# Steps in making a code change

This is the general cycle of steps to follow in making and "checking in" code changes, with details on the step how-to's following:

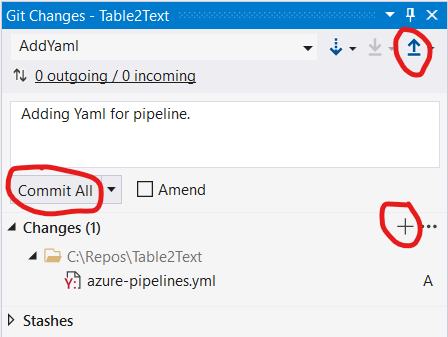
1. In Visual Studio, select the local 'develop' branch, and do a Pull



1. Then create a new local branch based on 'develop'. Keep the box to check out the branch checked.



1. Make your changes, stage the changes, commit the staged changes. When all your changes are ready and committed, push your branch.



1. Then click the link to do a Pull Request, and submit that (into 'develop'). After the Pull Request is complete, in Visual Studio select the 'develop' branch, then go to Manage Branches (as in the screen shot for # 2 above) so you can delete the local branch that was just pulled into 'develop' (do not reuse branch after pull request is done).

## Create a working branch for a user story or bug

Once your local repo is created, you're ready to create a branch off of your local **"develop"** for the user story or bug you on which you will be working. You will also do this if you're in the middle of working on a user story or bug for a TFVC project which becomes imported to Git.

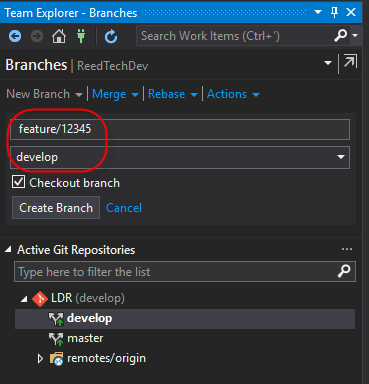
Go to Team Explorer - Branches, click New Branch, and select **"develop"** from the dropdown list for the source parent. It is strongly recommended you use the branch name pattern

**(feature|bug)/<work item ID>{\_description}**

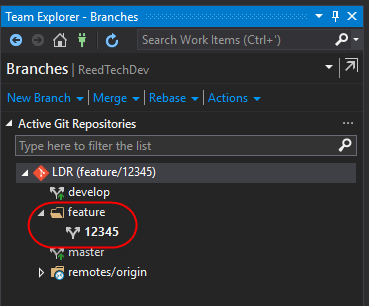
where

* **(feature|bug)** represents the "feature" (user story) or "bug" for which you'll be coding,
* **<work item ID>** is the ID number of the user story or bug work item in TFS, and
* **{\_description}** is an optional short description of your work on this feature or bug.

Feel free to add more descriptive text with **<work item ID>**, such as by appending a short description after an underscore ("\_"). While you don't absolutely need this, it helps visually differentiate what code you're working on when you create multiple concurrent branches. Note that spaces are not allowed in branch names.



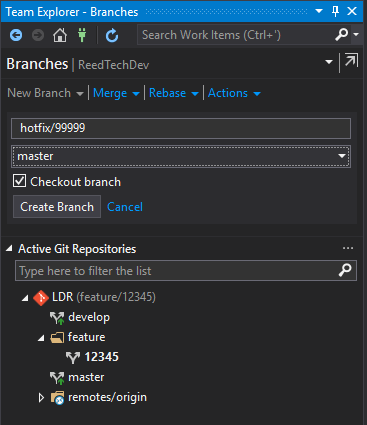
Click the Create Branch button to create your feature or bug branch. With the Checkout branch box checked, the new branch will become your current active branch and will appear bolded in your Active Git Repositories list. If you follow the recommended naming convention, you'll see all your feature branches grouped together.



If you were in the middle of working on a user story or bug during your project's import from TFVC to Git, add your code changes straight to this branch and continue working the story or bug.

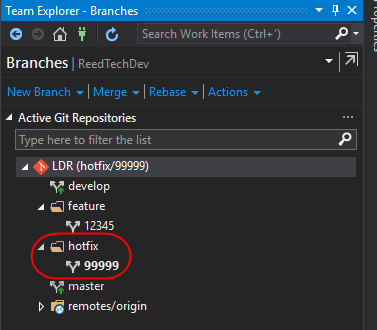
### Create a working branch for a hotfix

If you are working on a hotfix, create a branch named **hotfix/<work item ID>{\_description}** from your **"master"** branch:

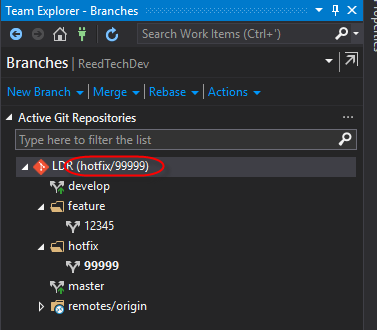


You're branching from master because it is what reflects current production code, not current development code. A hotfix is against production code.

As with features, the **hotfix/** prefix helps group your hotfix branches.



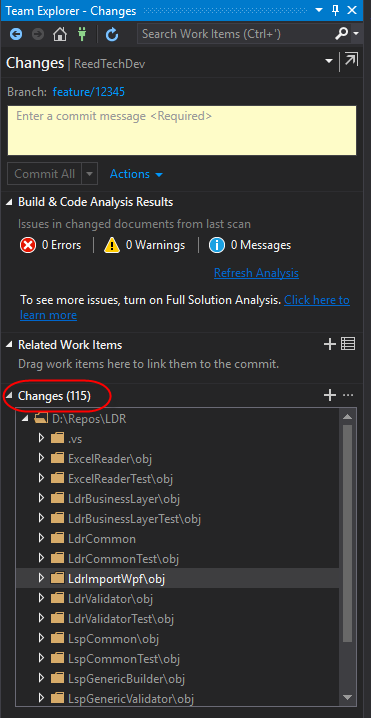
Again, by creating the new branch and having the Checkout branch box checked, your new hotfix branch became your active branch. You can also tell which branch you're working in because it appears next to the repo name under Active Git Repositories in the Team Explorer - Branches window:



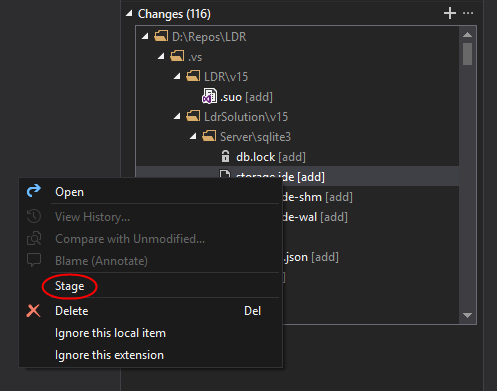
You can also switch the actively checked out branch by double-clicking on the desired branch under Active Git Repositories.

## Change tracking and committing

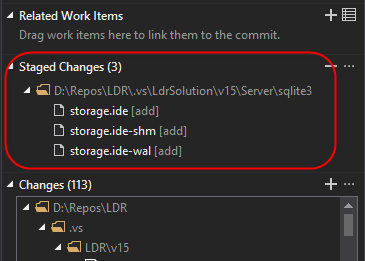
You can see your changes in Team Explorer by clicking on the Changes button from Team Explorer - Home. This looks similar to what you'd see when using TFVC.



However, before committing any changes, you need to stage them. This is like defining a changeset in TFVC. Right click on the changed files under Changes you want to process and click Stage.

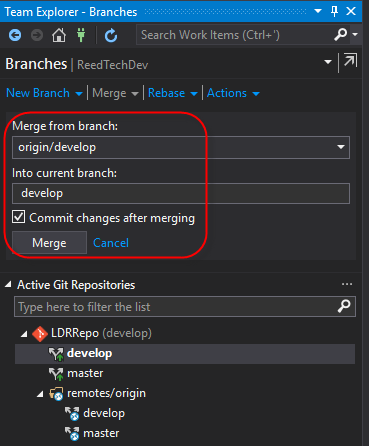


You'll see your staged file changes in their own Staged Changes section. At this point, you can also right click on any staged file and Unstage it if you do not want to commit it with the next commit request.



## Updating the local development branch

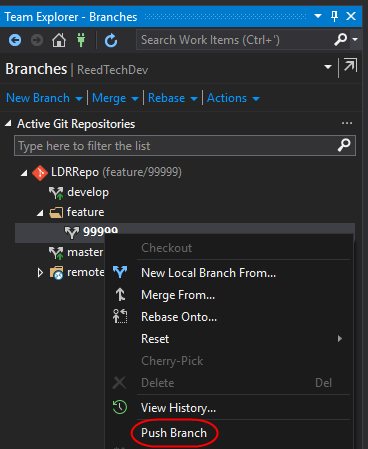
Before creating your feature branch, you'll want to merge from the **"origin/develop"** (server) branch onto your local **"develop"** branch. This is like "Get Latest" with TFVC. Right click your local **"develop"** branch and click on Merge From.... You can also click Merge with the local **"develop"** branch selected (checked out).



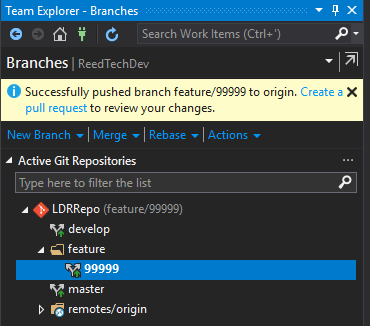
Alternately, you can right click on your local develop branch and click **pull** from the context menu.

## Committing features and bugs to the TFS (origin) development branch

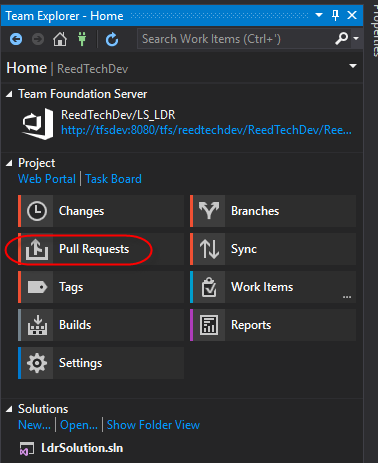
After you've completely coded and tested your feature or bug code changes, you need to merge your feature branch into the **"origin/develop"** branch on TFS. Since we require code review for each change to **"origin/develop"**, we need to push the feature branch to TFS then queue a pull request that enforces the review. Right click on your feature or bug branch and click on Push Branch to push it to TFS.



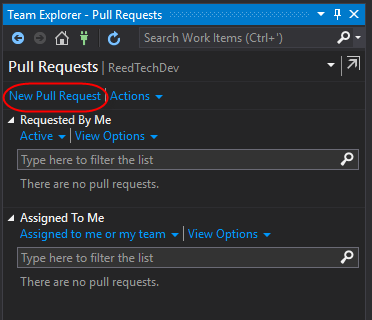
You will be notified when the push was successfully completed.



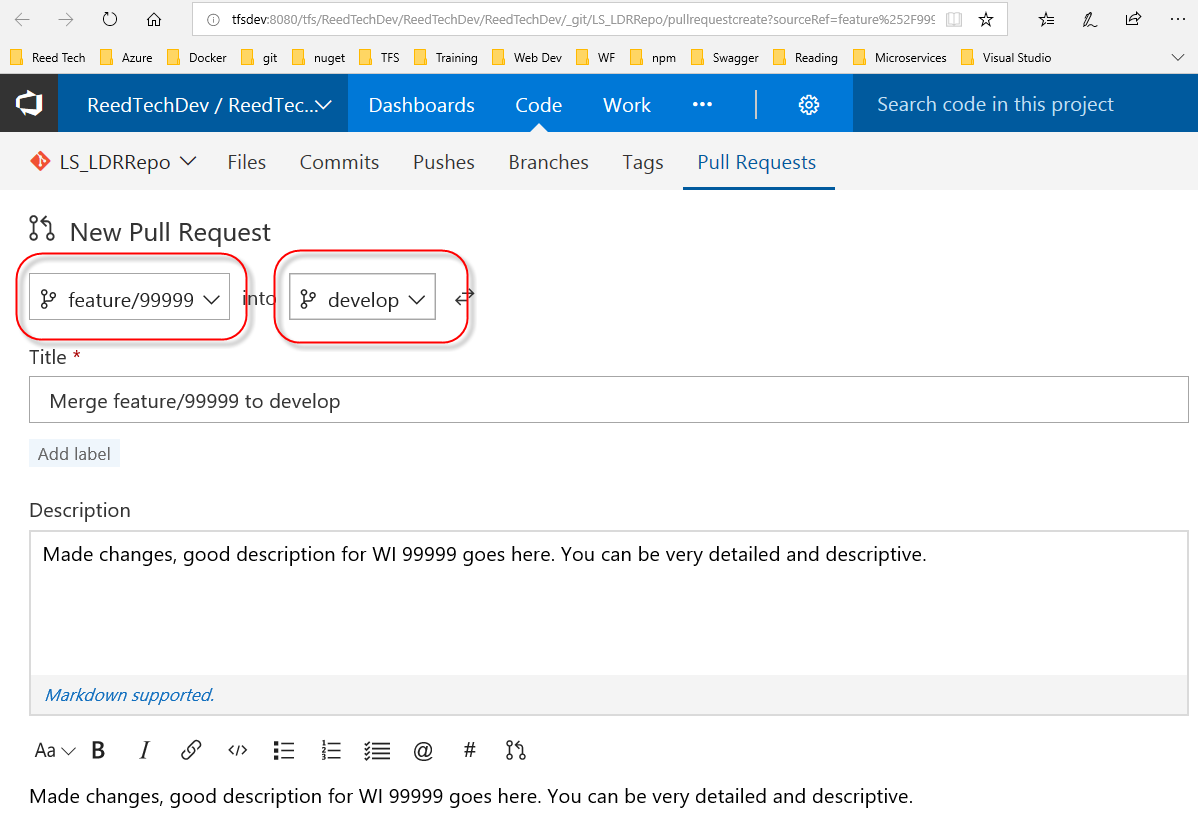
Now go to Pull Requests by clicking Pull Requests from the Team Explorer - Home window.



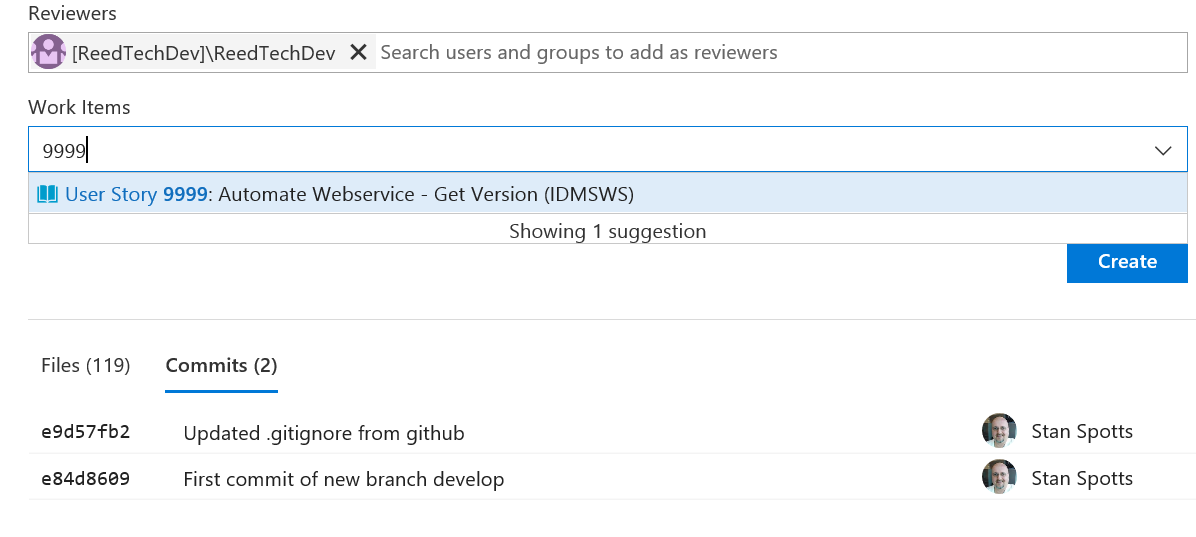
Click on New Pull Request.



Visual Studio will open your default browser to the New Pull Request page in TFS. Make sure that you're pulling from your feature/bug into **"develop"**. Add a useful title and a good description of the changes to be reviewed.



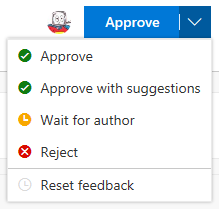
Add reviewer(s), or leave at default, and associate work item(s). Below, your committed files and their contents can be seen, along with the commits and their descriptions associated with this pull request.



Click the Create button to create the pull request to start this workflow.

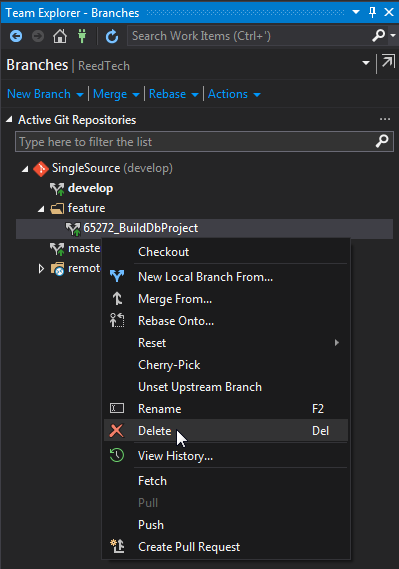
Each reviewer will receive notice of the pull request via email and through the Team Explorer - Pull Requests window under "Assigned to Me". Clicking on the pull request link will bring up the TFS Pull Request Overview tab in your browser.

The reviewer can give overall comment(s), as well as comment(s) inside the code in the differences view under the Files tab. Reviewers can set their feedback status of the pull request with one of the following:



Comments made by reviewers default to a status of "Active", and all comments must receive a response (or status change) before the pull request can be completed.

It is recommended you leave your local branch until your feature or bug changes are pulled into **"origin/develop"**. When you no longer need the local feature/bug branch, you can delete it.



It can be set up to delete the server version of your feature/bug branch after the pull request is approved and completed.

When all the user stories (features) for a release are complete, a lead creates a **"release"** branch. All bug fixes are pulled into the **"release"** branch and **"develop"** branch after that point. Once the release has been deployed to production, the **"release"** branch is tagged and merged into the **"master"** branch, then deleted.

## Making changes to a Pull Request

Sometimes a reviewer finds issues that need to be addressed before approving the merge into **"develop"**. To make changes for an existing pull request:

1. make your changes in the local branch that is the source for the pull request
2. stage and commit your changes
3. do not create a new pull request...the existing pull request will automatically update with the new version of the file(s) committed.

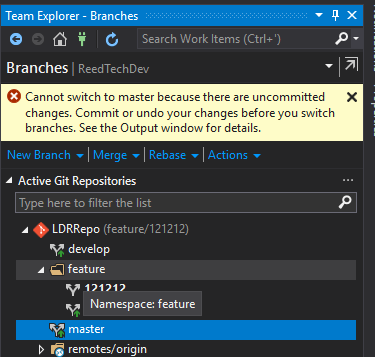
## Committing hotfixes to TFS Git

Hotfix workflow is similar to user story (feature) and bug workflow except:

1. Create your hotfix as **hotfix/<work item ID>{\_description}** from your **"master"** branch.
2. When you complete testing of your hotfix, you must merge with both your local **"master"** and **"develop"** repos, and create a pull request into both **"origin/master"** and **"origin/develop"**.
3. If a **"release"** branch is currently in use, you must also create a pull request into this **"release"** repo to add the hotfix to it.

## Switching branch context for hotfix

At some point you will have to make a hotfix but will be in the middle of working on a feature for an upcoming release. You will checkout the **"master"** branch and create a new hotfix branch. However, when you attempt to check out the **"master"** branch, you'll get an error message:



The Output window will show an error like this:

Cannot complete the operation because of existing changes to the following 2 files:

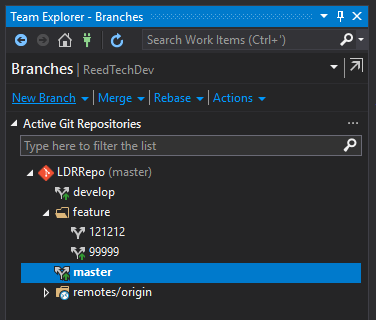
mypath/mysolution/file1.cs

mypath/mysolution/file2.cs

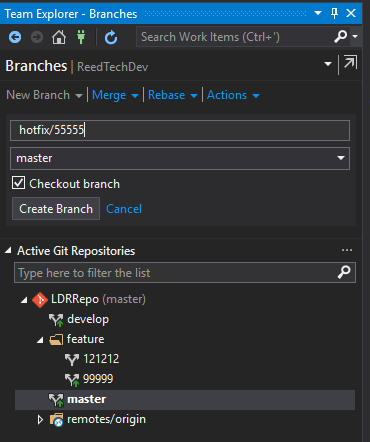
You'll need to either roll back your changes (not good) or stash them (good). Stashing is similar to the shelving in TFVC, except the stashed changes are not stored on the server. Visual Studio does not yet have stash built into its UI, so you will need to go to a command window and use the command **git stash "<description>"**:

git stash "saved work for feature/121212 to work on hotfix/55555"

Now you can change to the **"master"** branch.



And create a hotfix branch from **"master"**.



When you have

* completed coding and testing the hotfix,
* committed the changes to your local **"master"** and **"develop"** branches, and
* created the pull requests,

you can switch back to your feature branch. Then you can pop the stash to get your in-flight changes back.

git stash pop

When you no longer need your hotfix branch, you can delete it.

# Possible Issues and Their Fixes / Workarounds

### When attempting to modify a Setup project, you get error "The command you are attempting cannot be completed because the file '[Setup project file]' is under source code control and is not checked out."

Make sure that in Visual Studio you are working in a personal branch (not 'develop') and close the whole solution. Directly modify the Setup project file itself by navigating to it in file explorer and right-clicking on it and choosing to open it in something like Visual Studio or Notepad++. Make changes to the file and save it - refer to [this wiki page regarding that error.](https://tfsprod.reedtech.com/tfs/ReedTech/ReedTech/_wiki/wikis/ReedTech.wiki/349/Setup-(.vdproj)-Is-Under-Source-Code-Control-and-Is-Not-Checked-Out) Then re-open the solution in Visual Studio - your changes should be there and the project should now be checked out.

**git Hooks**



Spotts, Stan (RT-PHL)

Jan 4, 2019

Git hooks are scripts that are fired during the git code check-in process. Samples are created whenever you do a git init, and are placed in the *.git\hooks* folder off the root of your repo.

* **pre-commit** : runs before you even enter a commit message (unless you use -m switch).
* **prepare-commit-msg** : runs before displaying the commit message to you for editing. (Define template for messages)
* **commit-msg** : runs after you save the commit message, before commit completes. (Enforce message standards)
* **post-commit** : runs after commit is saved (completed). (Notify commit done or update status)
* **pre-base** : runs before rebase. (Check to ensure no push done before rebase)
* **post-rewrite** : runs after a commit is replaced. (Generated docs or copy untracked files in)
* **post-checkout** : runs after git checkout.
* **post-merge** : runs after git merge
* **pre-push** : happens before a push completes. (Can abort the push)

Currently, TFS doesn't support *server-side* git hooks, but does support branch policies. Use these to enforce using pull request for a branch on the server rather than a push, for gated check-in on remote (server) repo. Users cannot push directly to branch when this is configured.

Current policies include

* Require minimum number of reviewers to approve
* Check for linked work items (for traceability)
* Check for code review comment resolution (warn or stop if active)
* Enforce either  
  -- squash (creates single commit)  
  -- no-fast-forward (merges commit history of source branch) merge
* Automatically add users/teams as code reviewers
* Complete pull request only if would result in successful (CI) build
* Require approval via 3rd party service

**Stashing**



Spotts, Stan (RT-PHL)

Apr 8, 2019

Stashing is similar to shelvesets in TFVC, but much simpler to use and fast. Use this if you have work pending but have to load another branch into your workspace. You can get it back when you've completed that work, or even apply the stash to the branch you're now working on.

There is no support in the Visual Studio 2017 UI access to stash, you have to do it from the command line. It *is* supported in Visual Studio 2019.

**git stash** or **git stash push** - saves current tracked files to stash  
**git stash push -u** - saves current tracked and untracked files to stash  
**git stash push "{stash message}"** - saves current files to stash with {stash message}

**git stash list** - shows list of saved stashes, most recent first  
**git stash show** - shows summary of diffs between workspace and latest stash  
**git stash show -p** - shows full diff between workspace and latest stash  
**git stash show {id}** - shows summary of diffs between workspace and stash with id {id}

**git stash apply** - takes the topmost stash and applies to local workspace  
**git stash apply {id}** - takes the stash with id {id} and applies to local workspace  
**git stash pop** - same as git stash apply but deletes the stash from the stack  
**git stash pop {id}** - same as git stash apply {id} but deletes the stash from the stack

**git stash branch {name}** - creates a new branch {name} with the latest stash and delete the stash from the stack  
**git stash branch {name} {id}** - creates a new branch {name} with the stash with id {id} and delete the stash from the stack

**git stash drop** - deletes the latest stash from the stack  
**git stash drop {id}** - deletes the stash with id {id} from the stack  
**git stash clear** - deletes *all* the stashes in the repo

If you're in the middle of work and have to switch context to write a hot fix, you might do the following if you didn't use stash:

git checkout -b my\_current\_work

git commit -a -m "Work in progress"

git checkout master

{change stuff}

git commit -a -m "hotfix ready"

git checkout my\_current\_work

gt reset --soft HEAD^

But with stash, you wouldn't have to make a current work branch and multiple checkouts:

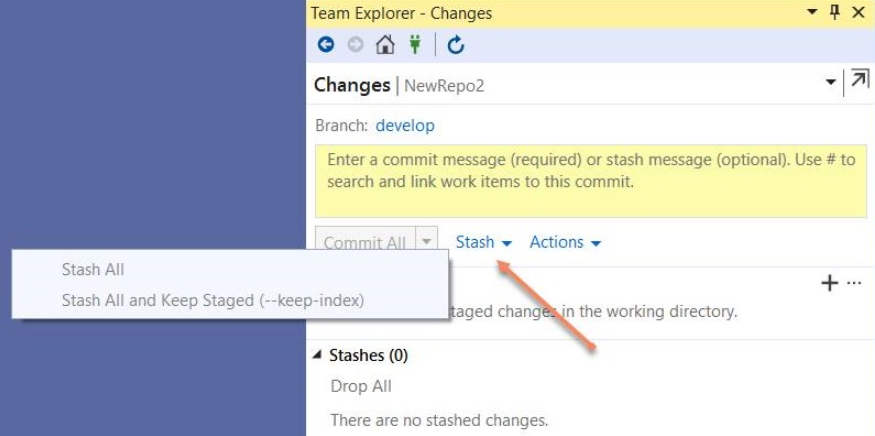
git stash

{change stuff}

git commit -a -m "hotfix ready"

git stash pop

Just to give you a taste from VS2019:



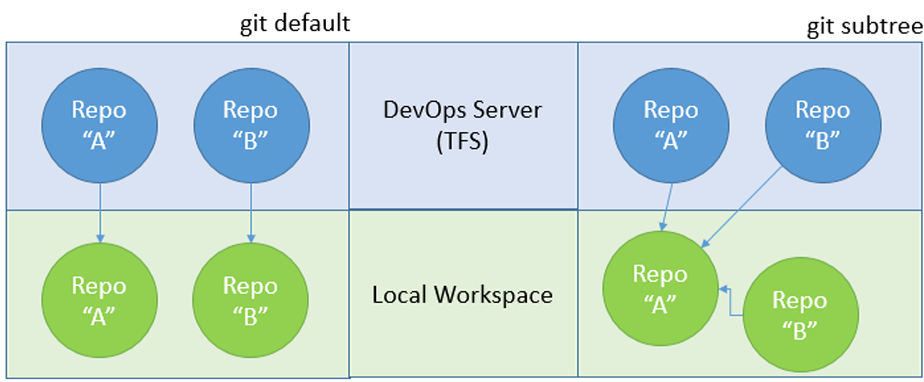
**git subtrees**



Spotts, Stan (RT-PHL)

May 18, 2020

A git subtree allows you to nest one repo inside another as a subfolder. This simplifies things when you have project dependencies. Another method is git submodule, but they are more complex and fit a limited set of use cases. So we'll review only subtree here. And like any other git scenarios, you can make a mess of things if you are not careful.

  
Once the subtree is defined, you will have a slightly more complex merge strategy to learn. You will also want to be careful if you change code in repos that are in subtrees, and commit changes specific only to the subtree or the main repo.

In the "main" local repo that you wish to add to, add the additional repo as a remote, then add the subtree:

### Creating the subtree

git remote add -f {subtreename} https://tfsprod.reedtech.com/tfs/ReedTech/ReedTech/\_git/{reponame}

git subtree add --prefix {subfoldername} {subtreename} develop --squash

**{subtreename**} is what you want to call this subtree rather than specifying the url; It's the alias.  
**{subfoldername}** is the subfolder under your current level in your main repo where you want the subtree to be stored.

### Committing changes to main (parent) repo

Commits will be to the main repo's branch. If you modify the subtree files, the changes will be committed to the parent branch. Be very careful if you are changing code in the subtree and not just using it as an easier way to do local build/unit testing.

If you want to push your changes to the subfolders repo, you need to push the subtree.

git subtree push --prefix={subfoldername} {subtreename} feature/{featureidentifier}

**{featureidentifier}** is the typical work item number and feature title that you use when you're working on a feature. We don't want to push this to develop (since we only allow pull requests to the develop branch), so we push it to DevOps Server as a feature branch. Then you can create a pull request for the feature branch to develop and proceed as normal. The subtree push goes through the commits in your local repo and determines which commits are applicable to the subtree's origin repo, and just pushes those.

### Pulling latest changes from remote into subtree

You will create a pull using the subtree merge strategy.

git subtree pull --squash --prefix={subfoldername} {subtreename} develop