Getting Started

Table of Contents

Powershell setup

See the PC setup guide for details of installing and setting up Powershell.

Among other things the profile automatically imports two modules, `TFS` and `TFS2`. These modules provide a number of "Get-TFS*" cmdlets useful for interacting with the git repos hosted on TFS.

Open up your profile file:

```
notepad $profile
```

and locate the section:

```
$global:tfs = @{
    root_url    = 'http://vssdmlivetfs:8080/tfs'
    collection  = 'BomiLiveTFS'
    project     = 'ReleaseEngineering'
}
```

Adjust `$global:tfs.project` as necessary for the project you usually work on.

Then save.

Credentials

To access TFS' git repositories (as used by the `TFS` and `TFS2` modules, above), the current credentials must be accessible. This can be done for each session by running:

```
$global:tfs.credential = Get-Credential
```

Alternatively, you can use this script:

```
$mycreds = $global:tfs.credential

if(! $mycreds -a  ! $password ) {
    $password = Read-host "Password" -AsSecureString
    $mycreds = New-Object System.Management.Automation.PSCredential ($env:UserName,
$password)
    $global:tfs.credential = $mycreds
}
```

Alternatively, can store using:

```
new-storedcredential -target $global:tfs.root_url -username $env:Username -Pass XXXXX
```

If you forget to do this, then running any of the TFS commands will prompt for your username/password.

git setup

Whereas using TFVC (TFS' "native" source code) required using Visual Studio to check in files, git is open source and supports many different clients. This section describes how to configure Visual Studio to use git, and how you can also interact with git from the command line and using a number of other freely available tools.

> To learn more about git concepts, see here.
>
> To learn more about why we're using git in the first place, see here.

Visual Studio

Git is automatically part of Visual Studio 2015 and 2013, though support in earlier versions is limited. We therefore recommend using VS2015 if working in the new git repositories.

The git support in VS2015 is reasonable, though some of the more advanced functions are missing. Assuming you understand how git works, you will be able to get by perfectly well (though Microsoft in their usual fashion have given new names to some concepts, eg "sync" meaning both git push and git pull. Sigh).

The support for git can be found in the Team Explorer pane:

[VS2015 Team Explorer for git repositories] | *_images/_setting-up-git/vs2015-team-explorer-git.png*

However, you will find the Source Control Explorer does *not* support git. Instead, just use the Windows Explorer to navigate.

Command line (Git for Windows)

The PC setup guide describes how to install git using Chocolatey; basically just run:

```
choco install git
```

This installs `git.exe` command line utility, accessible from both regular Windows command prompt and also from (mSys/MinGW) Linux bash shell.

SourceTree

If you prefer a GUI (as most Windows users probably will), then a tool we recommend is Atlassian's SourceTree. This is free to use, though you do need to register on their site to download it.

To install, the PC setup guide describes how to install SourceTree using Chocolatey; basically just run:

```
choco install SourceTree
```

Here's a screenshot to give you an idea:

[SourceTree] | *_images/_setting-up-git/sourcetree-screenshot.png*

Along the left hand side are the git repositories; these can be grouped into folders. If you have already cloned a repository, you can drag and drop the folder from (Windows) Explorer into this area.

Along the top toolbar you can `Clone` new repositories and use git `Fetch`/`Pull`/`Push` with the remote repository, while the `Explorer` button launches Windows Explorer so you can easily access the files in the repository. If you know the `git` command line then the `Terminal` to open up a terminal console.

One thing that's nice is that most of the more powerful git commands are available, but surfaced through easy-to-find Windows actions/context menus.

Configuring

Configuring is fairly painless…

- `Tools > Options > General`:

  Enter your welfare.ie email address:

  [SourceTree] | *_images/_setting-up-git/sourcetree-options-general.png*

- `Tools > Options > Git:`

  Use system git:

  [SourceTree] | *_images/_setting-up-git/sourcetree-options-general.png*

- `Tools > Options > Authentication`:

  Use your regular Active Directory credentials:

  [SourceTree] | *_images/_setting-up-git/sourcetree-options-authentication.png*

Other Clients

Some other git clients you might want to consider are:

- git extensions

  This is similar to SourceTree, also with a graphical UI. (It's probably more fully-featured than SourceTree, actually)

```
choco install gitextensions
```

- VS Code

  Microsoft's lightweight cross-platform editor, also comes with Markdown editor support.

  ```
  choco install VisualStudioCode
  ```

- Github Desktop

  Although this has specific features for github.com, it is possible to use with any git repo

  ```
  choco install GitHub
  ```

And you can find even more clients here.

TFS Power Tools

Some of the powershell commands provided by the REM team require TFS power tools to be installed.

There are various versions available, we recommend you install:

- TFS PowerTools 2013 Update 2 (download msi)

The procedure we recommend is:

- uninstall any previous versions of TFPT that you might already have installed* restart computer if so afterwards
- install the above MSI* *include* powershell cmdlets* Windows Explorer integration is also recommended
- restart computer once more.

There is also a TFS Power tools 2015, but we've had mixed results using it. Keep with 2013 for now.

setting up nuget

NuGet is the package manager for the Microsoft development platform including .NET. The NuGet client tools provide the ability to produce and consume packages.

The NuGet Gallery is the central package repository used by all package authors and consumers, including Microsoft and vendors such as Naked Objects Group.

NuGet.exe

The `NuGet.exe` utility is used to download (aka "restore") Nuget packages from the NuGet feed.

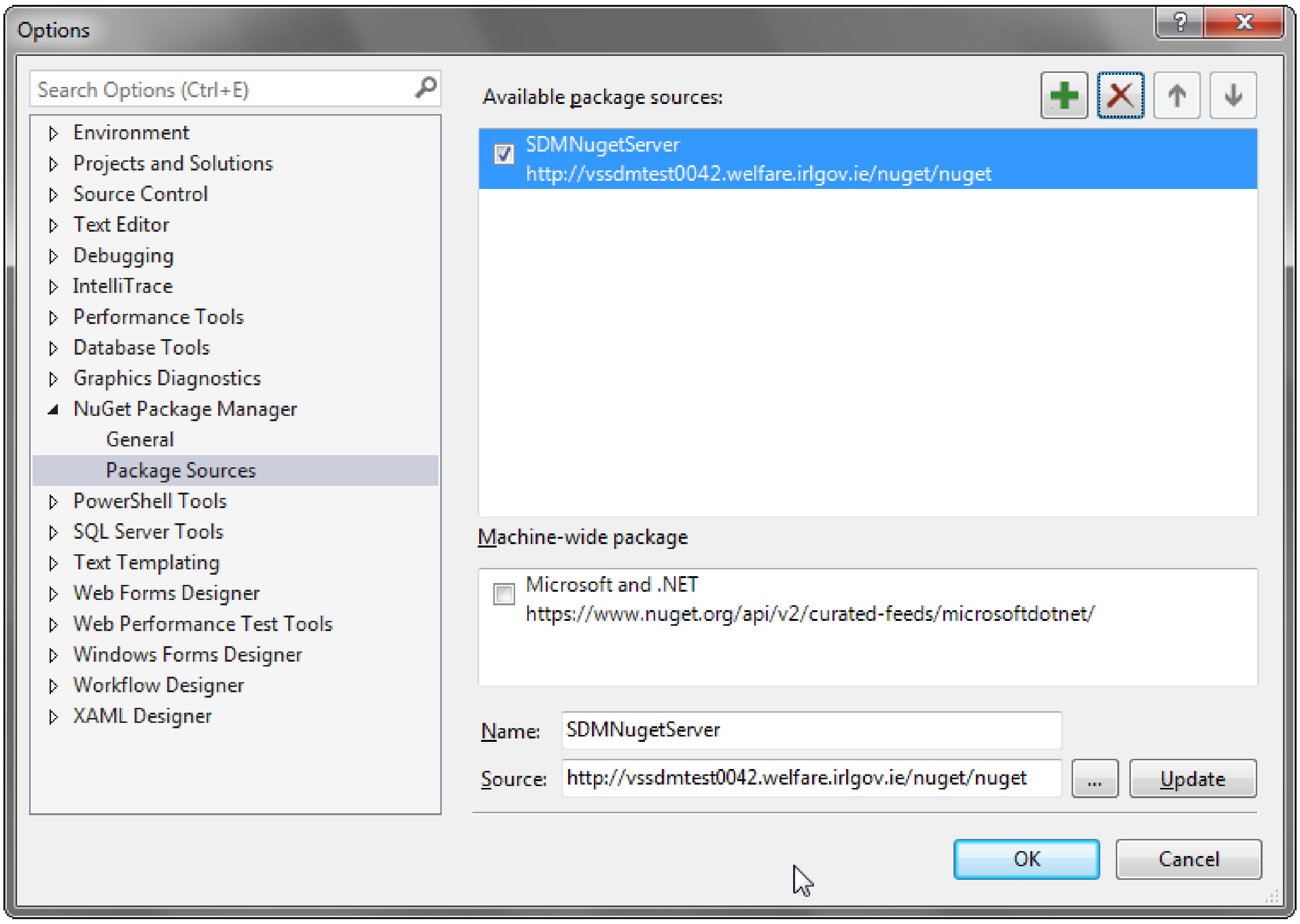
Previously REM scripts expect this file to reside in a well-known location (`C:\NuGet\NuGet.exe`). However, because `NuGet.exe` itself is updated by Microsoft quite recently (there were 11 releases in a year) the new platform's build scripts will automatically update this from a central copy (at `\\vssdmrelease\Tools\NuGet\Exe`) if the local copy is missing or out of date).

Package Sources/Feeds

While NuGet Gallery provides a central package repository, it is also possible to set up additional repositories.

Within the DSP projects we use a local nuget server on \vssdmtest0042\NugetPackages to hold nuget packages built locally. (It also has a local caches of packages downloaded from `nuget.org`).

In Visual Studio, the DSP nuget server should be configured as a "package source" (sometimes also called a "nuget feed"):



Available package sources:

☑ SDMNugetServer
http://vssdmtest0042.welfare.irlgov.ie/nuget/nuget

Machine-wide package

☐ Microsoft and .NET
https://www.nuget.org/api/v2/curated-feeds/microsoftdotnet/

Name: SDMNugetServer

Source: http://vssdmtest0042.welfare.irlgov.ie/nuget/nuget

Local Feed

The `Build-Locally` script is used to build, test and package up code from a given repo. If that code itself uses nuget packages, then these are obtained using the package sources on the PC.

When the developer pushes a new version of the code to git, a CI pipeline will run and (if all tests pass) result in a new version of the package being published to the DSP Nuget server. This new version of the package is **immutable and cannot be changed**.

While this is great for traceability, it would be highly inconvenient/irritating for developers trying to create and test a new version of a package. Therefore the `Build-Locally` script allows the package to be built and published locally. This can be done as many times as necessary, overwriting any previously built version each time. Only when the software is finally pushed to git/CI does the version become fixed.

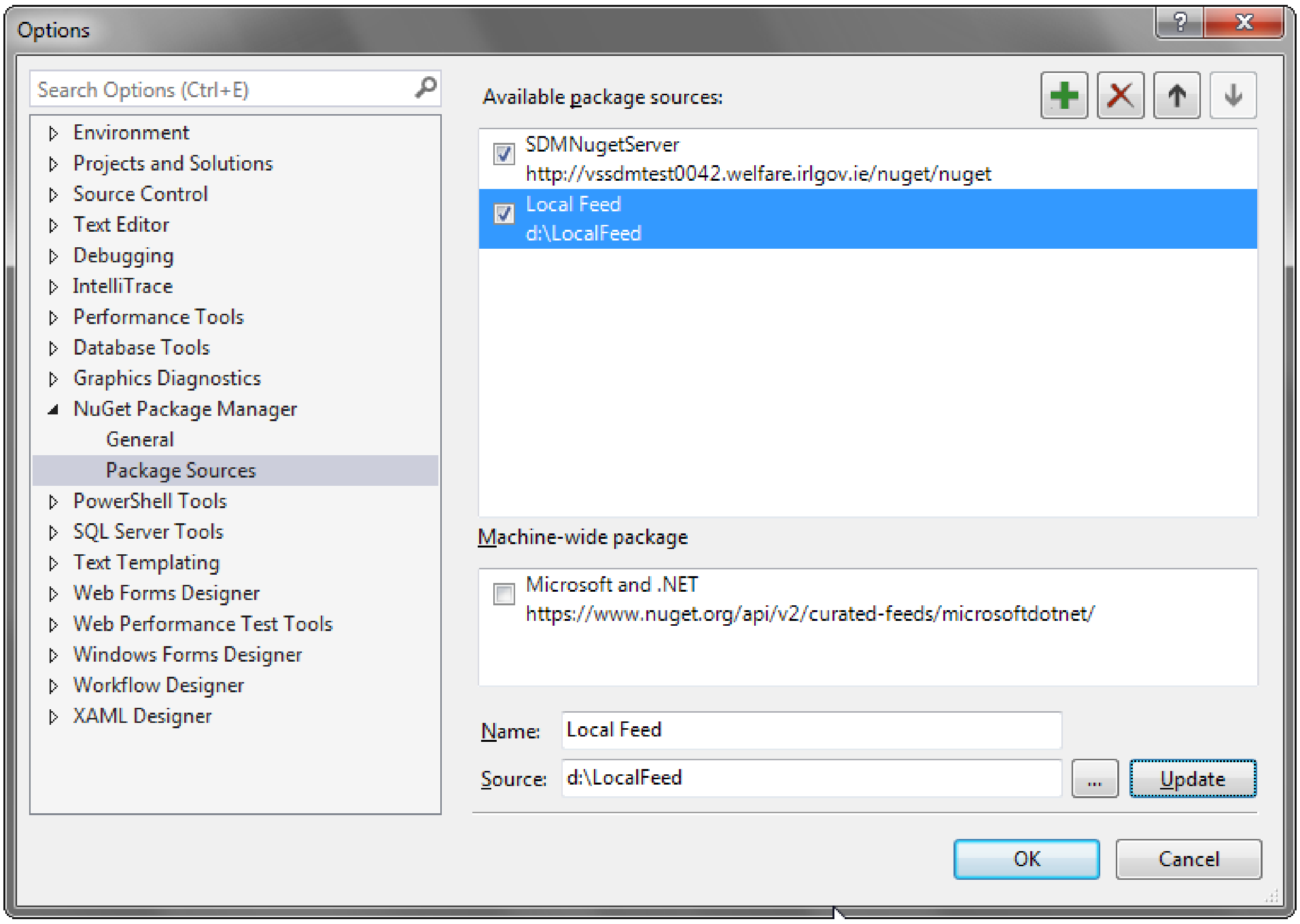
The local feed is set up in artifact.yml using the `publish.publish_local` property, eg:

```
publish:
  publish_local: D:\LocalFeed
```

TODO: this screenshot is out-of-date, and references the original physical server. Instead, use "nugetserver" as the host name.

Note that `Build-Locally` also uses this local feed directory as a package source itself (to restore from). This therefore means that the developer can work on two inter-dependent projects.

This feed directory should therefore also be set up as a package source within Visual Studio:



☑ SDMNugetServer
http://vssdmtest0042.welfare.irlgov.ie/nuget/nuget

☑ Local Feed
d:\LocalFeed

☐ Microsoft and .NET
https://www.nuget.org/api/v2/curated-feeds/microsoftdotnet/

Name: Local Feed

Source: d:\LocalFeed

The REM team generally recommends using a local directory, such as `D:\LocalFeed`. However, if you could configure some other directory, eg a read/write network share, if you wish to share a package between a group of developers before formally pushing a new version.

Setting up your local workspace

In the short-to-medium term there will be two "types" of git repositories:

- BOMi "artifact" repos: those that contain the source code for a single packaged artifact (usually a nuget component, but could in principle be anything)
- everything else, hosted in SDM_DEV_MIRROR.

The plan is to steadily factor out the existing codebase from the latter into the former; eventually there will be 100s of the former. However, this is an "eat the elephant" type of undertaking, not something that can be done overnight. See the transition plan for a more detailed view as to how we'll get there.

Workspace Layout

As discussed in more detail in the concepts docs, TFS team projects are containers of multiple (git) repositories. Since you are likely to work on multiple git repos, we highly recommend that you check out git repositories following this hierarchy:

```
D:\git\XXX\Yyy
```

where

- `git` is a useful prefix to group all the cloned repos together
- `XXX` is the containing TFS team repository
- `YYY` is the git repository

You can use either `C:` or `D:` drive, wherever you have room.

The one exception to this scheme is SDM_DEV_MIRROR team project, which contains two git repos, `Trunk` and `NonDelivery`. Because of the 260 character path limit you'll find that cloning the git repo will not fully succeed if you use `D:\git\SDM_DEV_MIRROR\Trunk` as the base directory. For these two repositories we therefore abbreviate the team project to `SDM`.

The suggested layout is therefore:

```
D:\git\
    SDM\
        Trunk\
        NonDelivery\
    ReleaseEngineering\
        BuildScripts\
        DeveloperGuide\
    BOMi_Infrastructure\
        Sdm.Cluster.Root.Api\
        Sdm.Infrastructure.Api
        Sdm.Services.Bomi4.Api\
    BOMi\
        ri.calculator\
        SdmBasisSearch\
    TestingFrameworks\
        Sdm.Test.Common\
        Sdm.Test.RandomData\
```

and so on.

Eventually we expect there to be 10~20 team projects (one per major DSP "product" built on .NET), with each team project having maybe 10 to 50 git repositories. This directory structure should therefore be reasonably manageable.

Cloning all git repos

Run the following script:

```
Get-TFSGitRepositoriesAll | % {
    $projectName = $_.ProjectName
    $repoName = $_.RepoName
    $repoUrl = $_.RepoUrl
    "$projectName|$repoName|$repoUrl"
}  > repos.txt
```

This will generate a `repos.txt` file in the current directory.

Comment out (with a `#`) any repos you don’t care about.

Then run:

```
cat repos.txt | ? { ! $_.StartsWith("#") } | % {
    $parts = $_.Split("|")
    $projectName = $parts[0]
    $repoName = $parts[1]
    $repoUrl = $parts[2]
    $exists = get-item "$projectName/$repoName/.git/HEAD" -ea silently
    if (! $exists) {
        $x = new-item -ItemType directory -path $projectName/$repoName -force -ea
silently
        if ($?) {
            pushd $projectName
            echo "git clone $repoUrl"
            invoke-expression "git clone $repoUrl"
            popd
        }
    }
}
```

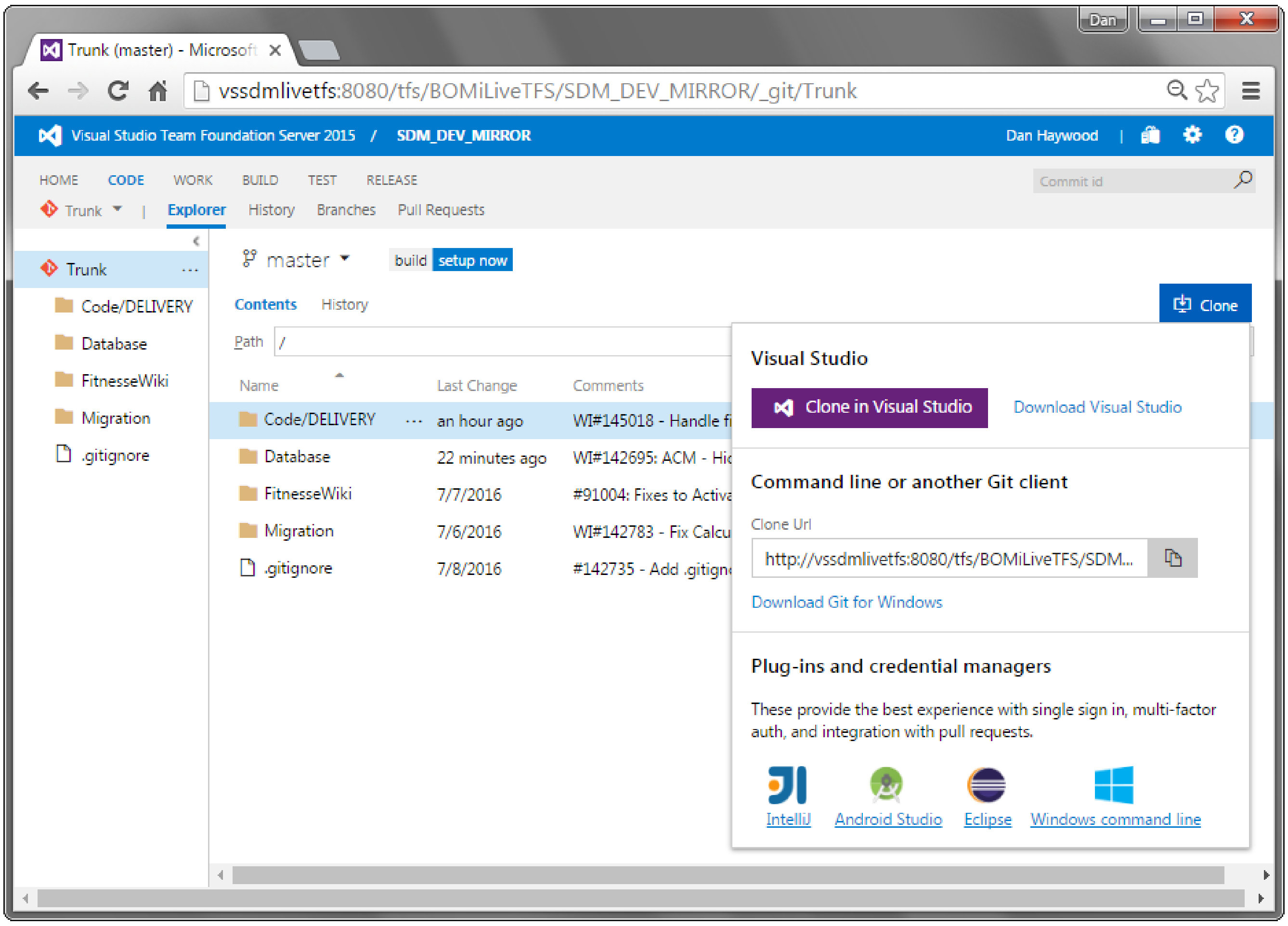
All missing repos will be cloned; any existing repos will be left untouched.

Once you've done this, you'll need to rename `SDM_DEV_MIRROR` to `SDM` and then ensure all files are correctly checked out:

```
cd D:\git
mv SDM_DEV_MIRROR SDM
cd SDM\Trunk
git reset --hard master
```

Manually cloning repos

To manually clone a repository, you can just obtain the URL from the git repo's home page on TFS, eg:


Trunk (master) - Microsoft
Dan
vssdmlivetfs:8080/tfs/BOMiLiveTFS/SDM_DEV_MIRROR/_git/Trunk
Visual Studio Team Foundation Server 2015 / SDM_DEV_MIRROR
Dan Haywood
HOME CODE WORK BUILD TEST RELEASE
Trunk | Explorer History Branches Pull Requests
Commit id
Trunk
Code/DELIVERY
Database
FitnesseWiki
Migration
.gitignore
master
build setup now
Contents History
Path /
Clone

Name	Last Change	Comments
Code/DELIVERY	an hour ago	WI#145018 - Handle fi
Database	22 minutes ago	WI#142695: ACM - Hic
FitnesseWiki	7/7/2016	#91004: Fixes to Activa
Migration	7/6/2016	WI#142783 - Fix Calcu
.gitignore	7/8/2016	#142735 - Add .gitign

Visual Studio

Clone in Visual Studio Download Visual Studio

Command line or another Git client

Clone Url

http://vssdmlivetfs:8080/tfs/BOMiLiveTFS/SDM...

Download Git for Windows

Plug-ins and credential managers

These provide the best experience with single sign in, multi-factor auth, and integration with pull requests.

IntelliJ Android Studio Eclipse Windows command line

You can then clone using either the command line, Visual Studio, or a graphical tool such as SourceTree.

Using Command Line

Switch to the appropriate directory and use git clone, eg:

```
cd D:\git\BOMi_Infrastructure
git clone
http://vssdmlivetfs:8080/tfs/BOMiLiveTFS/BOMi_Infrastructure/_git/Sdm.Cluster.Root.Api
```

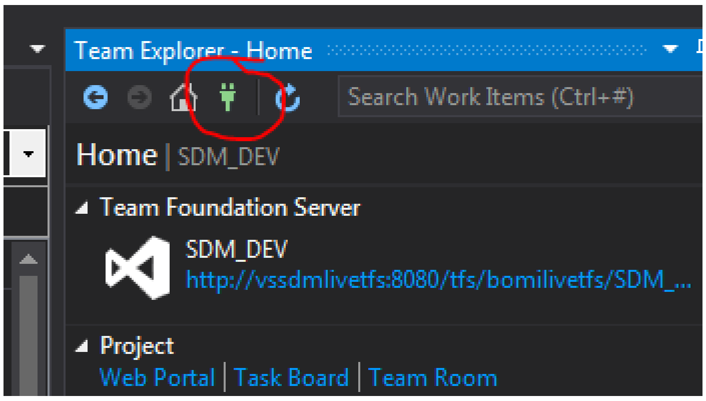
This will create a subdirectory with the same name as the repository cloned (eg `Sdm.Cluster.Root.Api` in the example above).

Using Visual Studio

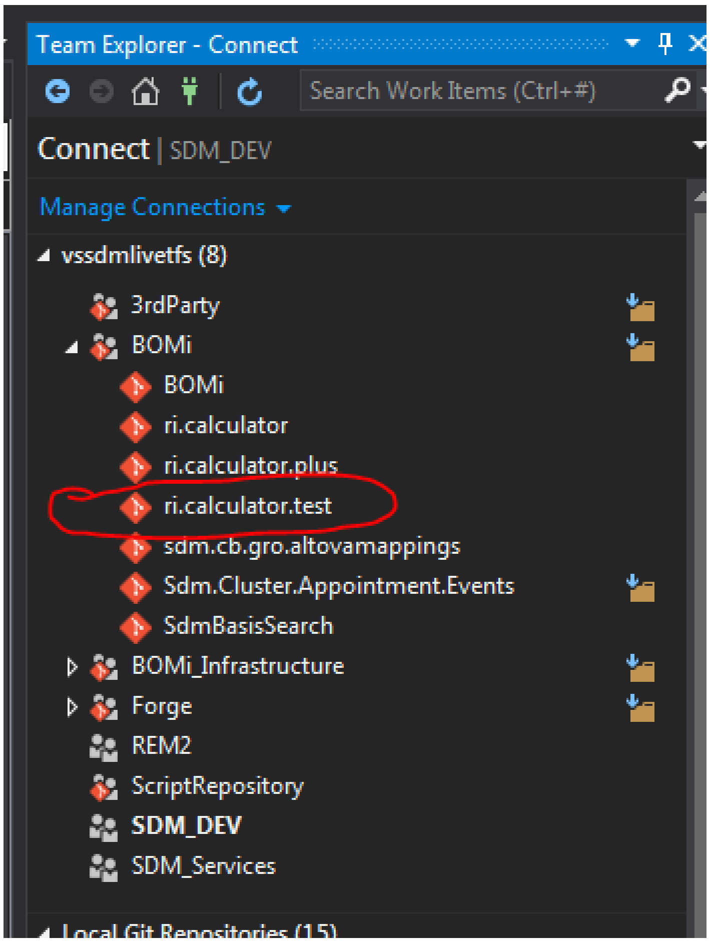
NB: the screenshots below are for a different git repo, not SDM_DEV_MIRROR/Trunk; adapt accordingly

To checkout an individual git repository using Visual Studio:

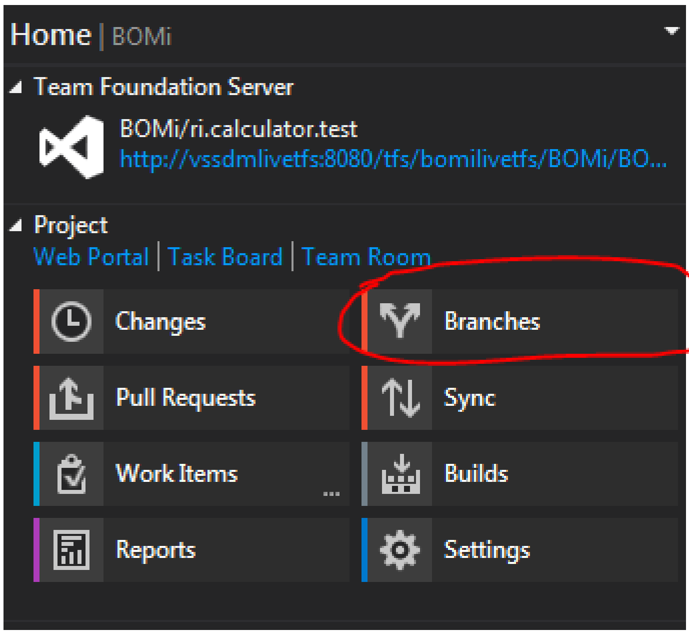
- connect to TFS team project:


Team Explorer - Home
Search Work Items (Ctrl+#)
Home | SDM_DEV
Team Foundation Server
SDM_DEV
http://vssdmlivetfs:8080/tfs/bomilivetfs/SDM_...
Project
Web Portal | Task Board | Team Room

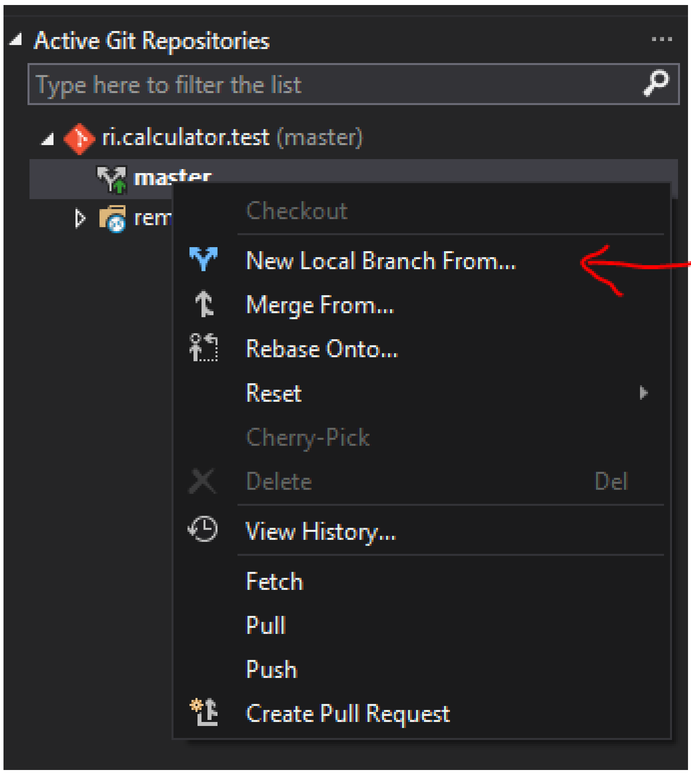
- Select the git repository within:



- view the branches available:



- Finally, clone the repository, checking out the `master` branch locally:



Using SourceTree

TODO.