| **File name** | Using YouTestMe SVN |
| --- | --- |
| **Author** | YouTestMe |
| **Confidentiality** | Internal |
| **Last save date** | Friday, January-12-2024 at 5:40:28 PM |

Contents

[1 Introduction 2](#_Toc220141558)

[2 SVN Installation 2](#_Toc28067337)

[2.1 Selective checkout (highly recommended) 3](#_Toc655219345)

[2.2 Checkout of the entire repository 3](#_Toc539874439)

[3 Important Commands 5](#_Toc412974153)

[4 SVN Good Practices 6](#_Toc98259449)

[4.1 Do not check files larger than 15 MB 6](#_Toc1270776713)

[4.2 Do not check in temporary files 6](#_Toc410417732)

[4.2.1 How to Automate Ignoring of the MS Office Temporary Files 6](#_Toc2003310863)

[4.3 General Naming Convention 7](#_Toc1955502718)

[5 Development Procedure using SVN 8](#_Toc2144377558)

[5.1 Using utility script to merge code (recommended) 9](#_Toc902962150)

[6 Creating a branch 10](#_Toc803887240)

[6.1 Creating a full project branch 11](#_Toc574960075)

[6.1.1 Naming convention 11](#_Toc298998092)

[6.1.2 Branching the desired folder on server 11](#_Toc828629320)

[6.1.3 Checkout the newly created branch 13](#_Toc199874188)

[6.2 Creating a CODE only branch 15](#_Toc921199653)

[6.2.1 Branching out the “www\_source” folder 15](#_Toc1473573762)

[6.2.2 Checking out the CODE branch 18](#_Toc428090734)

[7 Updating the “Projects.xls” file 19](#_Toc738472131)

[8 Manual Branch Merge 20](#_Toc1568222814)

[8.1 Branch to trunk merge 24](#_Toc518606062)

[9 Release procedure 24](#_Toc1924766050)

[9.1 Release standard 27](#_Toc173858055)

[10 Huge commits rule 28](#_Toc829979637)

[11 Remote SVN Connection 31](#_Toc1619061449)

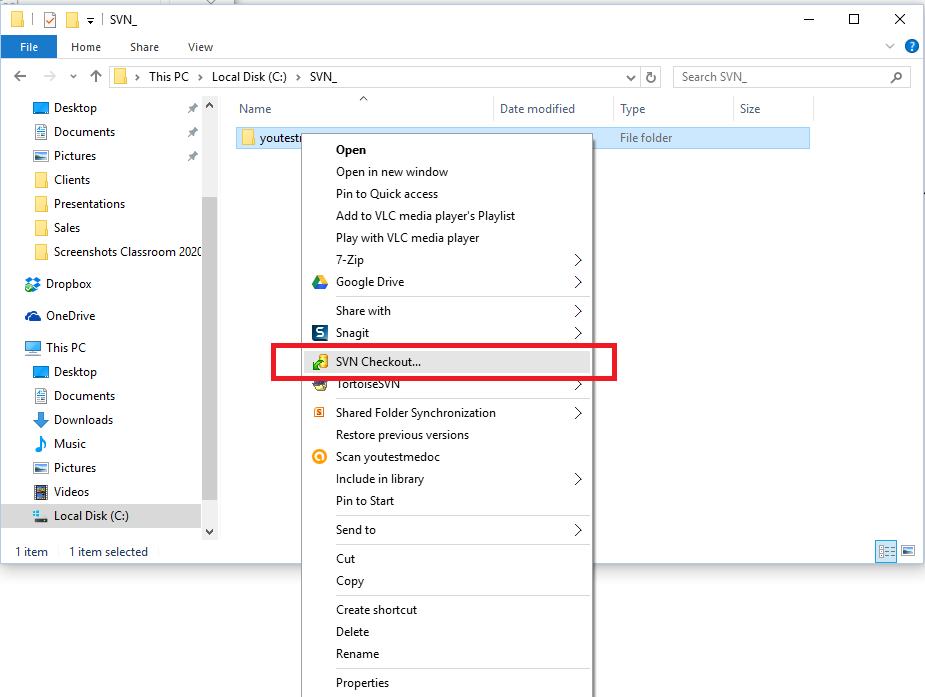
# Introduction

This document serves as a instructional manual how to use SVN in YouTestMe.

# SVN Installation

Go to <http://tortoisesvn.net/downloads.html> and choose the appropriate version for your PC (32 bit or 64 bit). Once it is downloaded, go to one of your local hard drives and create a folder called SVN. In it, create a new folder called “youtestmedoc”.

Right click on the youtestmedoc folder and there will be an option “SVN Checkout” (picture). If the option doesn’t appear immediately, try restarting your PC.



## Selective checkout (highly recommended)

This method is suitable for all users who encounter SVN for the first time and need to use only certain repositories.

* Find the necessary repositories for your team: [SVN repositories by team](https://docs.google.com/spreadsheets/d/1EITXpg9TOJmW322smPt1a_sLQfL1c99UHPV9wK4KHPs/edit" \l "gid=0).
* And **follow the instruction for selective checkout**: [Selective Checkout](https://owncloud.youtestme.com/owncloud/s/7XOHWyUNLn0hJ69).

Graphical user interface, text, application, email

Description automatically generated

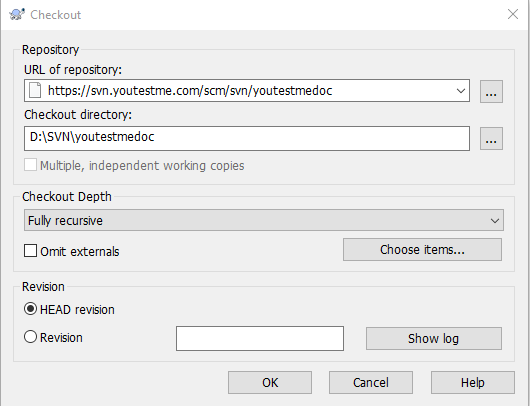
Once you type in the address, click OK. You will be prompted to type in the username and password you were provided, specific to this SVN repository group. If you are working on a shared computer DO NOT check the option “Save authentication”.

## Checkout of the entire repository

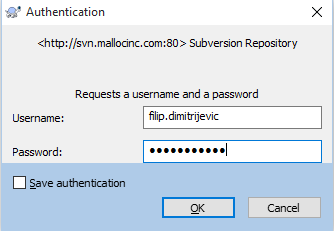
If you need to checkout the entire youtestmedoc with all repositories, follow the instruction below:

* Type in the server URL for repository “youtestmedoc” (picture):

<https://svn.youtestme.com/scm/svn/youtestmedoc>



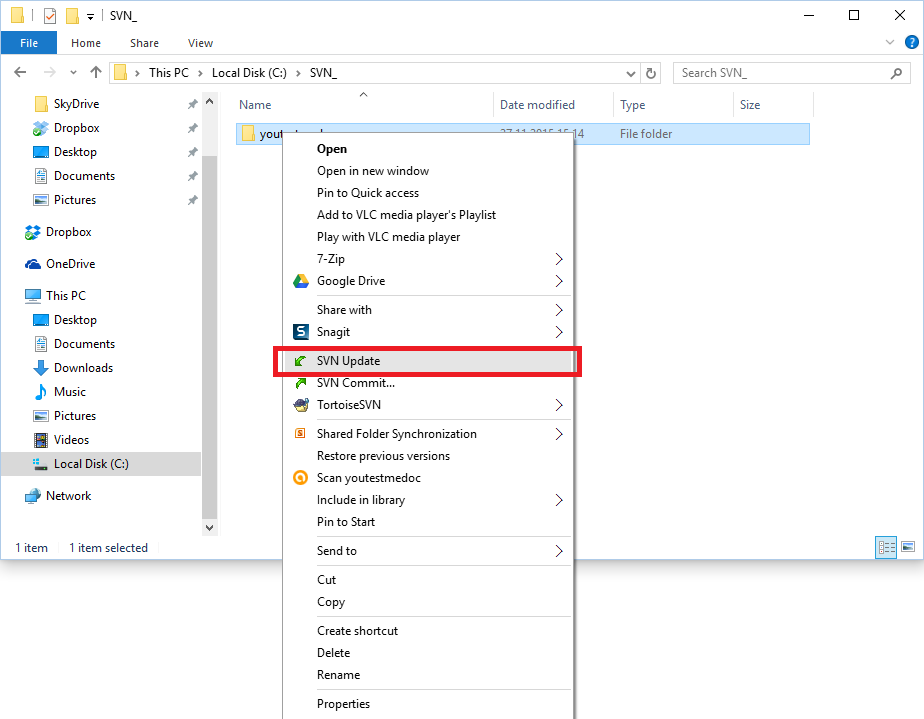
Once you type in the address, click OK. You will be prompted to type in the username and password you were provided, specific to this SVN repository group. If you are working on a shared computer DO NOT check the option “Save authentication”.

****

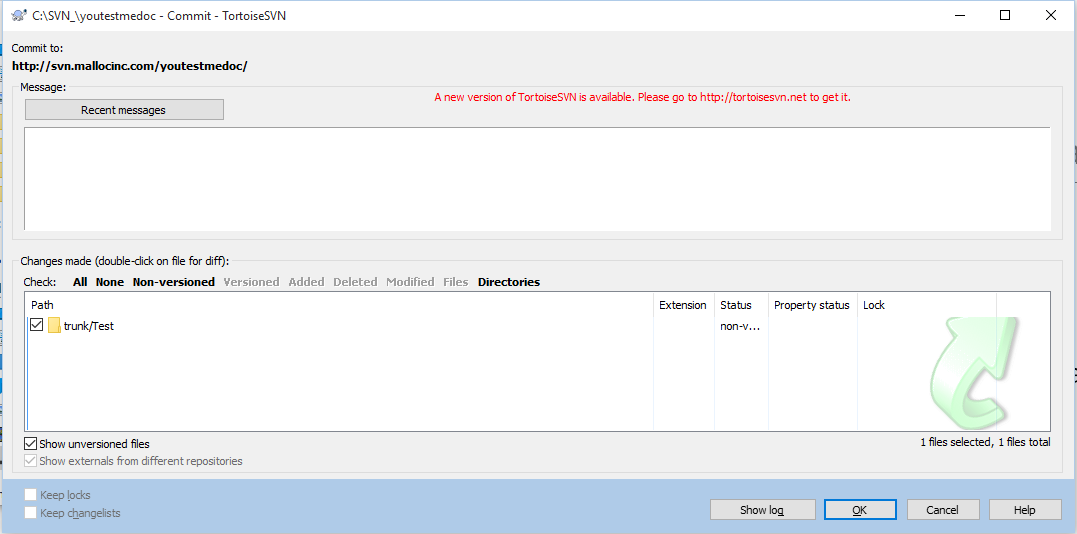
**Important: The username and password for this SVN is not the same as your email password!**

# Important Commands

Before making any changes to the SVN, update the entire folder. It is highly advisable to perform updates at least once a day to avoid any conflicts. Right click on the youtestmedoc folder and choose the “**SVN Update**” option. You can also target some of the subfolders for more specific updates.



Another important command is “**SVN Commit**”. This one sends the content you created to the shared SVN folder. In other words, when you add or update a file in the SVN folder it is also necessary to commit it using this command. Once you click the SVN Commit option, you will see a list of files that you updated or added. In case of adding new files it is necessary to check the box on the left side of them (picture).



# SVN Good Practices

## Do not check files larger than 15 MB

SVN is not a backup system. There should be no videos, voice messages, database and website backups, WAR files, program installations, etc.

## Do not check in temporary files

Please don't check these files in SVN:

* ~$\*.docx
* ~$\*.xls
* ~$\*.xlsx

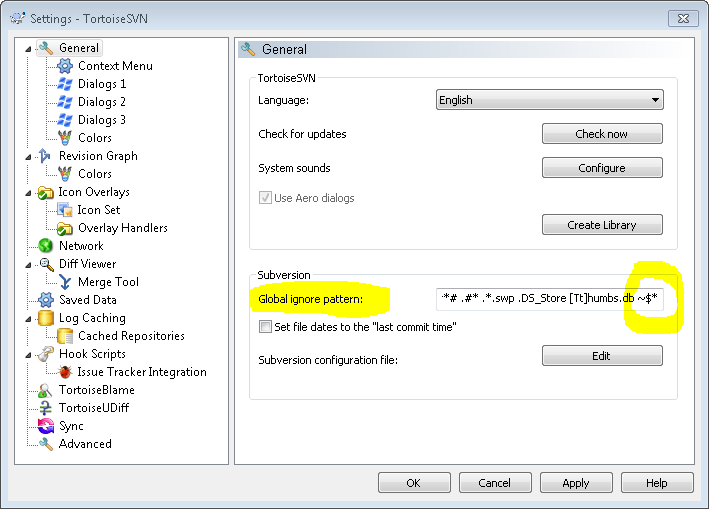
Those are temporary MS WORD and Excel files that exist only while documents are open.

Close documents before checking them into SVN. That will also make ~$\* files disappear.

### How to Automate Ignoring of the MS Office Temporary Files

To set up SVN to automatically ignore temporary MS Office Files so you don't check them in by mistake, add "~$\*" to "Global Ignore Pattern" as shown on the picture below.

Go to "Tortoise SVN -> Settings"



## General Naming Convention

1. Name characters should be in this range [A-Z][a-z][-\_.], without other special characters.
2. Names should be shorter than 30 characters. Avoid full sentences for description in name. If you need to explain the purpose of a directory, place a readme file in it.
3. Use **ONLY** English letters for naming. Using Serbian Latin is not compatible with all systems. Don’t even think about Cyrillic.

# Development Procedure using SVN

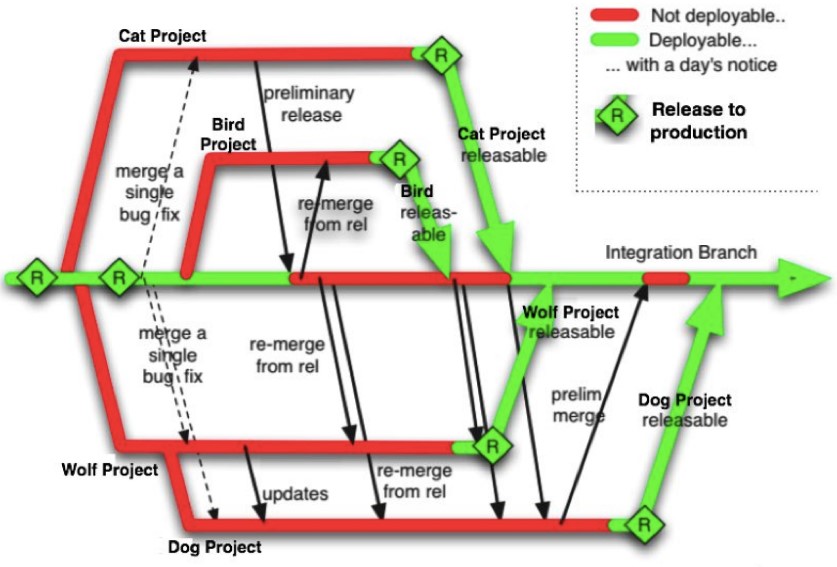
Never do the following:

1. Do not do development in “trunk”
2. Merge code with conflicts – solve conflicts first.

Step by Step procedure:

1. Create a branch for your project. Give branch a meaningful name. SVN log description should have:
   1. Source branch name used to create a branch
   2. Description about the purpose of the branch. Example: This branch is created from “trunk” and it is used as a Release 6.4 main development branch.
2. Merge “source” branch to your new branch at least once a day to pick up changes from source branch and to avoid conflicts when merging back to “source” branch. Utility script template:  
   \res\Scripts\SVN\DOS\For Developers\Merging\ytm\_svn\_merge.bat  
   Note that above script will commit all changes in your working copy/your branch.
3. Do all development and testing in your branch
4. When your development and testing is completed:
   1. Merge source branch one more time to your branch. Note: merging script will commit all your changes in working copy.
   2. Produce SVN diff between your branch and “source” branch and send to your manager for review and approval. Your manager has to verify and approve the code changes your are implementing. Utility script template:   
      \res\Scripts\SVN\DOS\For Developers\Merging\ytm\_svn\_diff.bat
   3. Merge your branch into “source” branch. Important: when you are merging back to “source” branch you need to use SVN flag “—reintegrate”. Please see sample script “ytm\_svn\_merge\_back\_to\_source.bat” in \res\Scripts\SVN\DOS\For Developers\Merging\Used by Zoran\

Graphical representation of using branches in YTM development process.



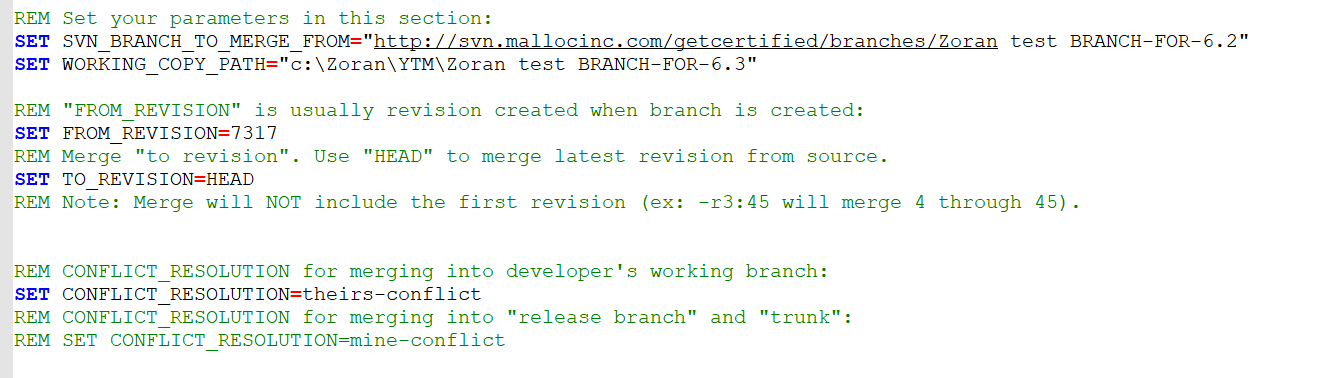
## Using utility script to merge code (recommended)

Use script to do the merging. It is much easier and faster than manual procedure which is also described in text further below.

Sample/template scripts for SVN merging is in this location:

\res\Scripts\SVN\DOS\For Developers\Merging\Used by Zoran\

Step by step procedure:

1. Make a copy of template Script for SVN merging into your working directory (working branch).
2. Make changes to your local copy of merging script. Below is section of the script that should be changed. Comments in the script are self explanatory however if unsure – ask your manager.  
   
3. Try your merge script with minor code changes before you start doing development to make sure that merging procedure works fine
4. Do your development in YOUR branch
5. Merge “source” branch” to your branch at least once a day (with script is now just one click). Note that utility script will commit all changes in your working copy/your branch.

# Creating a branch

We will use trunk as an example in this document to create a branch.

We will also distinguish the process of creating a branch with all projects (database and code files) from just a branch that has only code files.

\* Note (important):

Code must be formatted before creating a branch!

## Creating a full project branch

### Naming convention

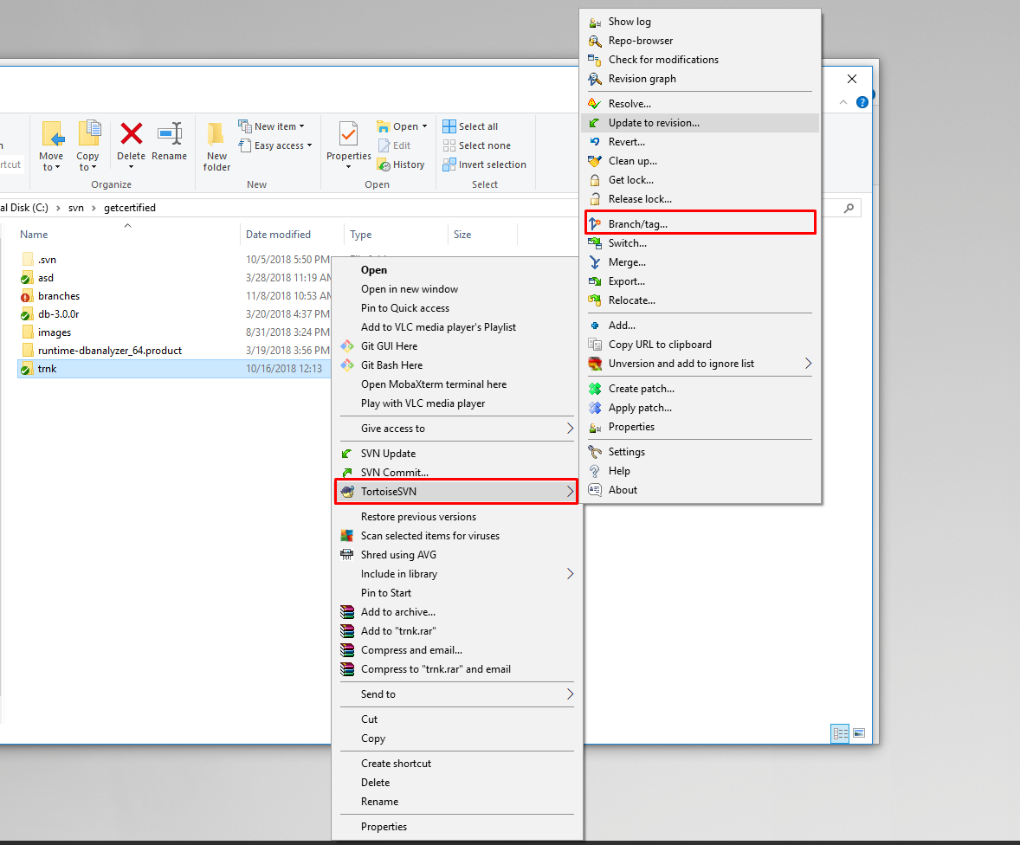
All branches should strictly follow the naming convention listed below:

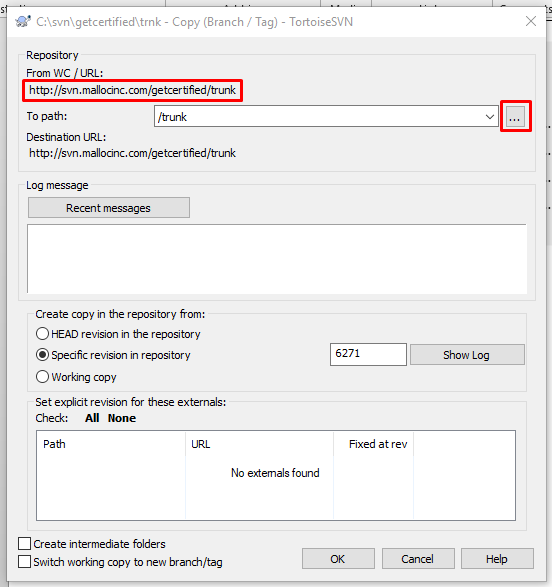
1. Main release branch should contain only the application name and planned version (e.g GC-6.3.0)
2. All branches should start with the application name followed by planned version (i.e GC-6.3.0-BLOBS)
3. Branches used for experimental purposes should have EXP tag before the functionality (i.e GC-6.3.0-EXP-ZAPP)
4. Branches used for integration or manual merge of certain functionalities or versions before should have INT tag before the functionality (i.e GC-6.3.0-INT-SAML)

### Branching the desired folder on server

Firstly, we need to create a folder on the server that will hold the branch for us (essentially, creating a new copy with the same data as that folder but on a different path).

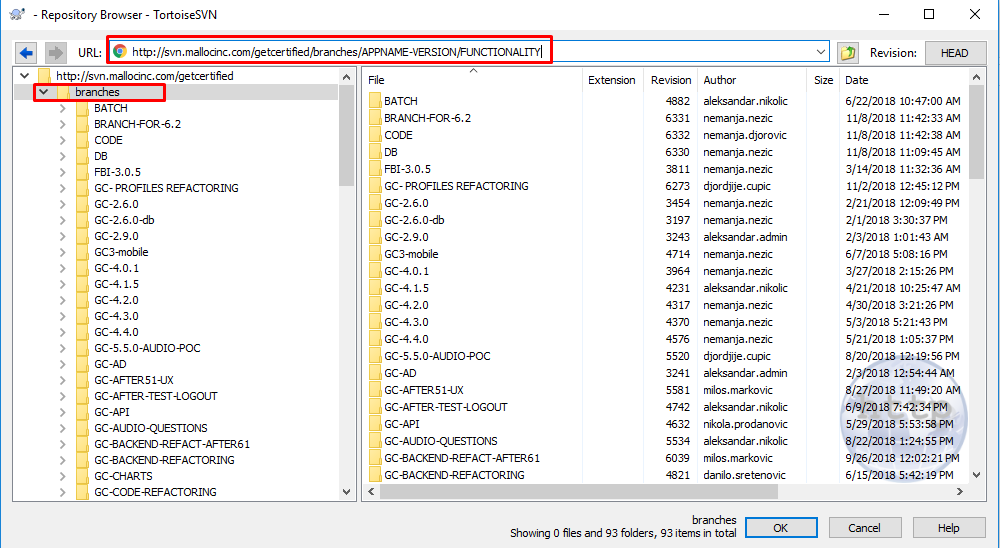
To do this, right click on the desired folder you wish to branch from and select the following options highlighted on the picture below.





The folder will be located on server under the “Branches” folder. In the URL highlighted on top, enter the desired name for your branch following the predefined standard explained below:

* APPNAME – Name of the application (GC for GetCertified, CL for Classroom2020)
* VERSION/FUNCTIONALITY – version or name of the functionality that will be developed on this branch (i.e GC-6.2.0 for version, GC-UX-Polishing for functionality)



When completed, press “OK” and you should see the following screen as shown on picture below.

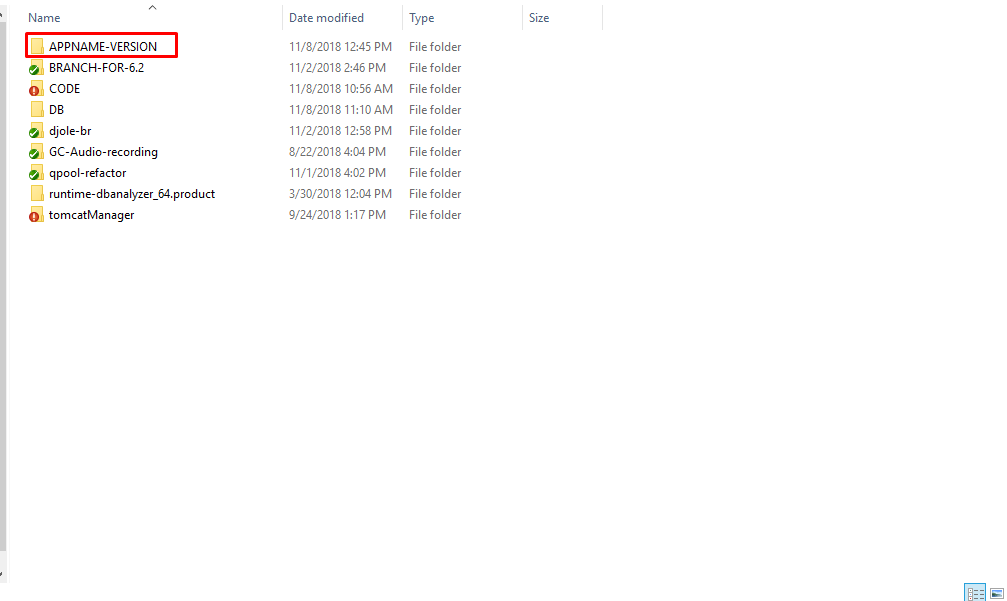


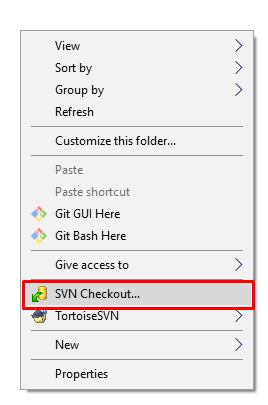
Finally, press “OK” and wait a few moments for system to create the folder on server.

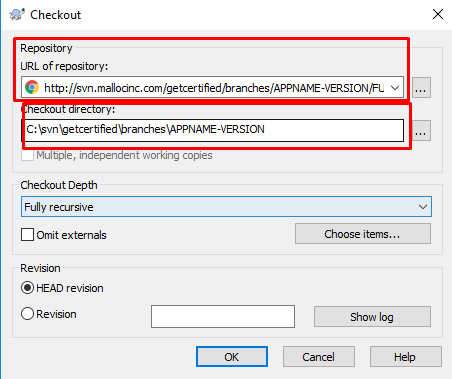
### Checkout the newly created branch

Last step is to check out the branch from the server onto your PC. To do this, we’ll firstly create a folder in which we’ll checkout (“download”, if you will) the files from the server. This will work as our local copy of data located on the server. You can name the folder anything you like, but a rule of thumb is to use the same naming convention as with branches on server (refer to the picture below).

When finished, go inside that folder right click and select “SVNCheckout option”. Make sure the paths are correct (highlighted in the picture) and click ok “OK” button.







That’s it, we’re done. You have successfully created a branch.

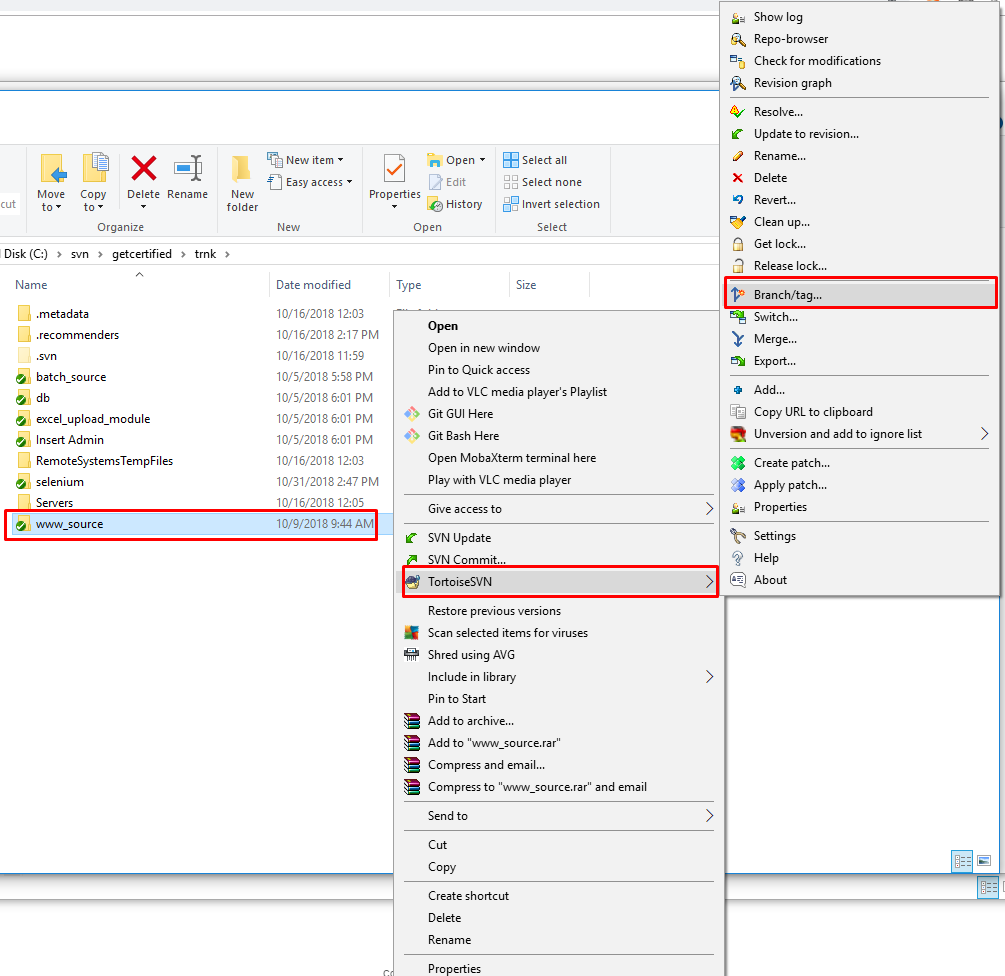
## Creating a CODE only branch

### Branching out the “www\_source” folder

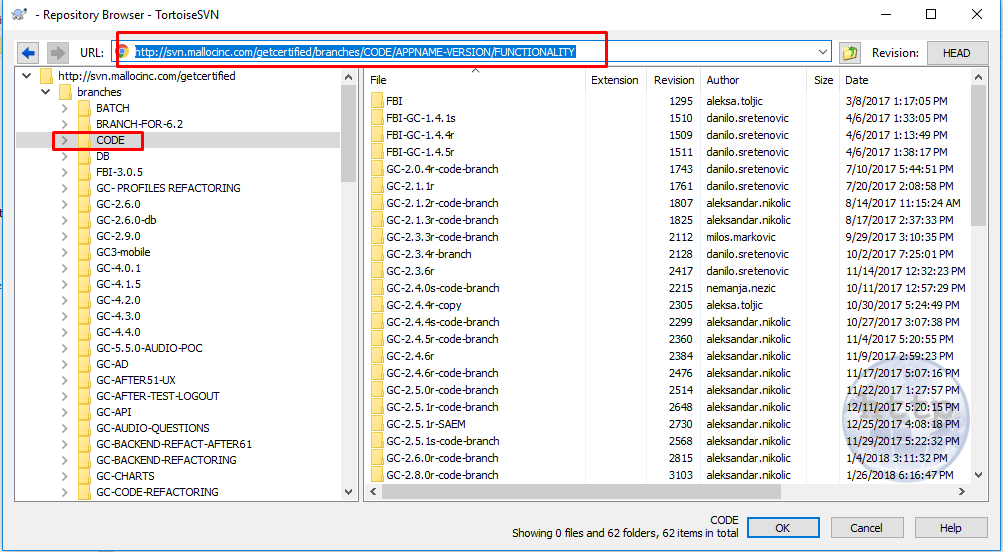
In this example we’ll user trunk to branch only the code.

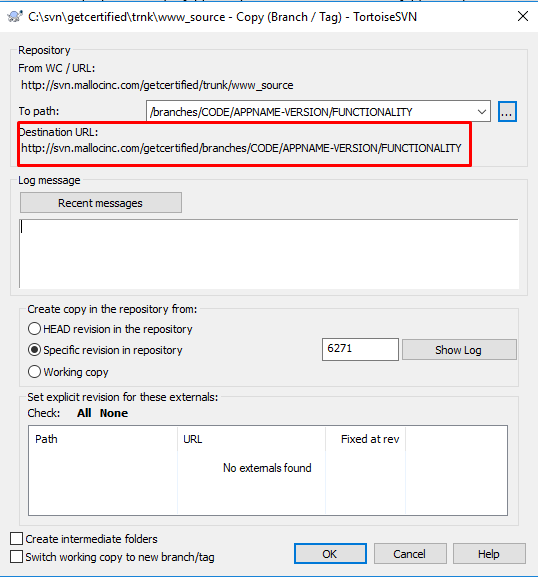
To do this, open the folder and navigate to www\_source folder inside.

Right click on the www\_source folder, select the options highlighted in the picture below.



Once the new window opens, navigate to the CODE folder on the right. Inside, create a new folder following the naming standard explained above in section 2.1.1 of this document.



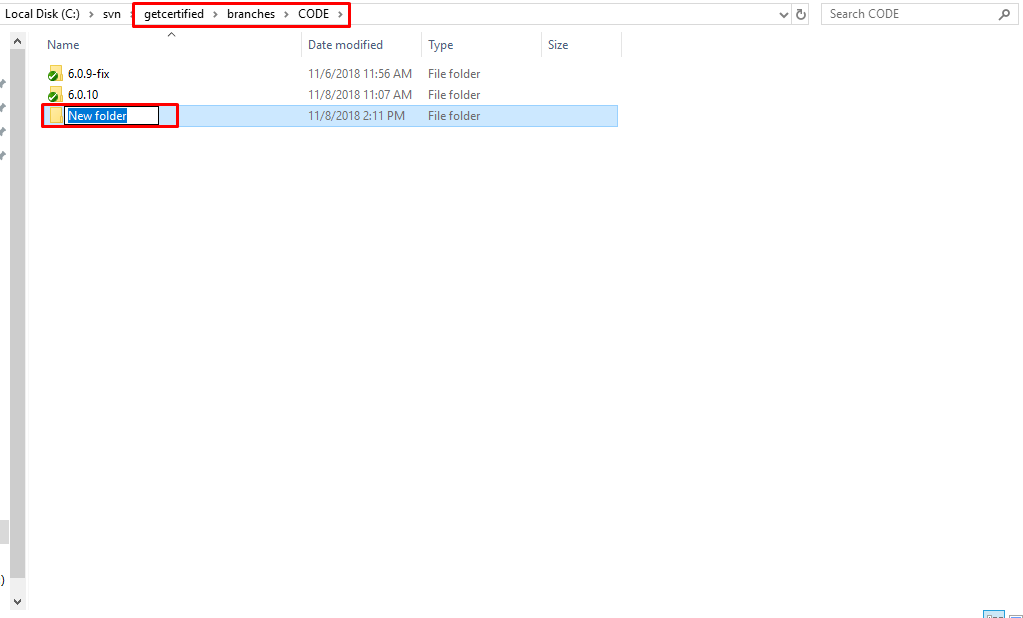


### Checking out the CODE branch

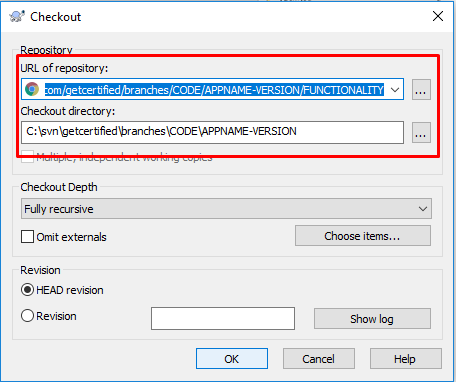
Finally let’s checkout the branch we have just created.

To do this, navigate to the desired folder you wish to store this branch (NOTE: if there aren’t any, you should create a CODE folder in which you will store only CODE branches).

In the example shown in the picture below we will create a folder inside CODE folder, located under the “Branches” subfolder.



When done, navigate inside that folder. To initialize the checkout process, right click inside the folder and select SVN Checkout option. Verify that the parameters highlighted in the picture below are correct, and press OK.



# Updating the “Projects.xls” file

After every new branch is created, the Projects file needs to be updated with the corresponding information about the branch.

The Projects file is located on the following path:

* <http://svn.mallocinc.com/youtestmedoc1/trunk/Projects/Projects.xlsx>

Once you open the file, switch to the “Branches” sheet and enter the following information:

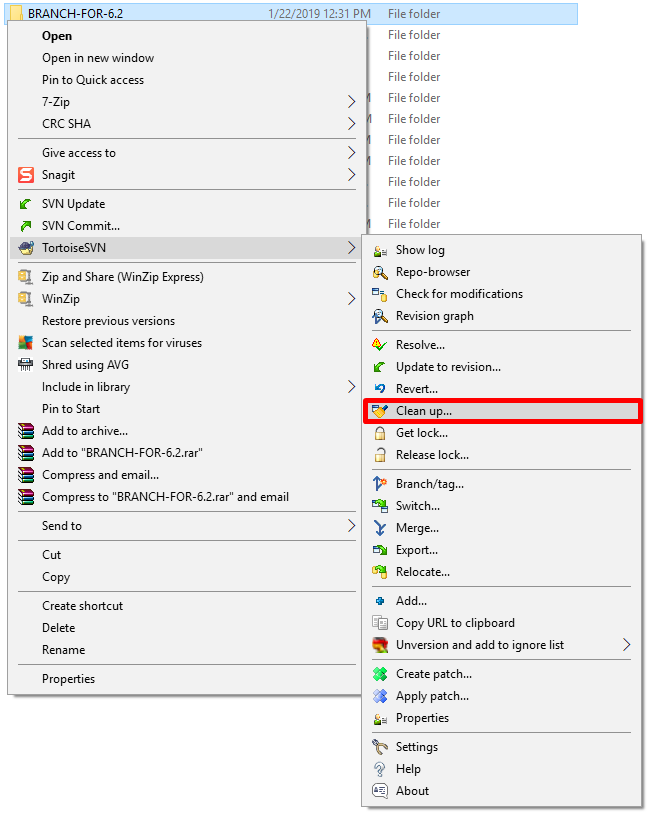
* Application – Name of the application to which the branch is associated with
* Created by – Name of the person who created the branch
* Date created – Date of branch creation
* Branch/Tag/Trunk – SVN path to the created branch
* Comment – Brief description of the purpose of branch
* Branch type – CODE only or Full project branch
* From tag/branch – SVN location of the root folder that was branched out
* Compatible with DB version – Database version that is compatible with the code

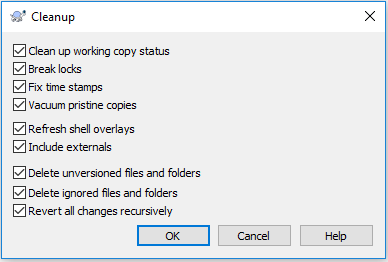
**NOTE**

**Update of the above mentioned information is mandatory, so please do not leave any of the fields listed empty.**

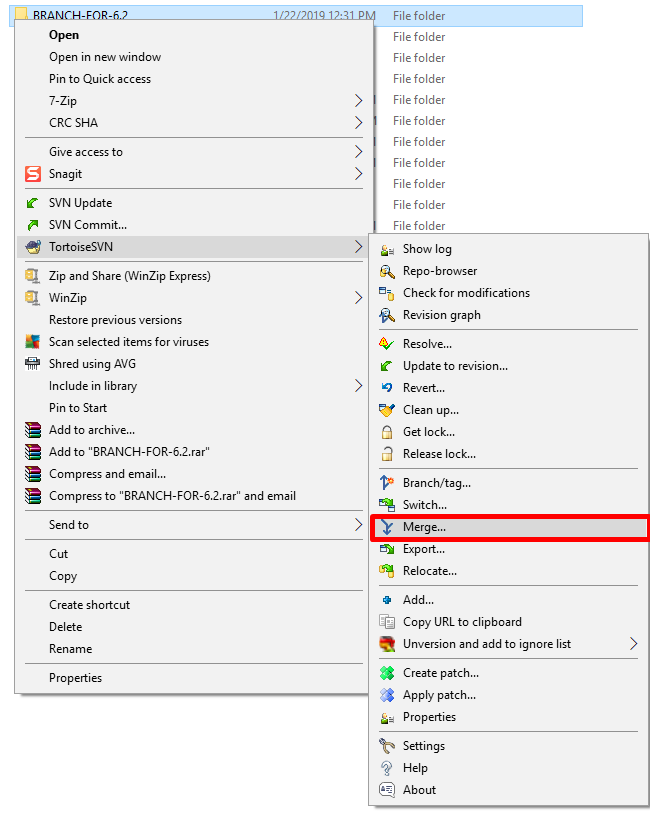
# Manual Branch Merge

Before doing a merge process, it is good to perform a full clean-up of both folders (branch/trunk) we are doing merge on.

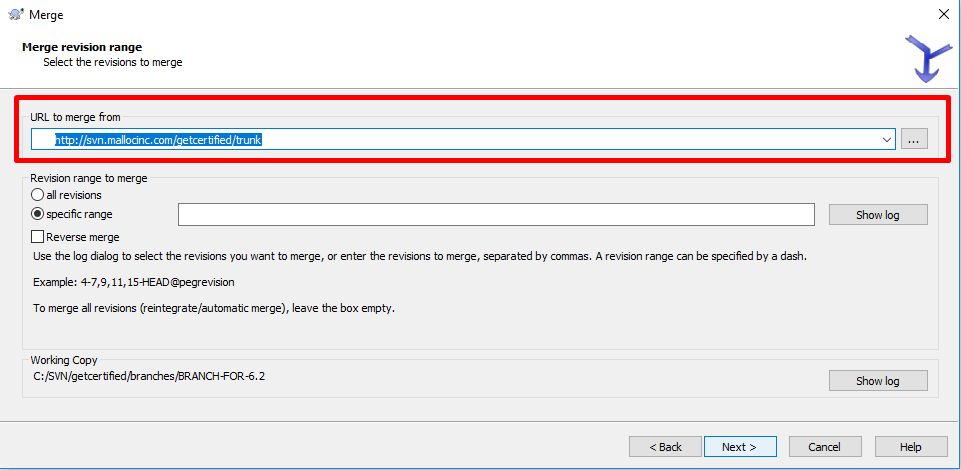




To do a merge, first select a merge folder (ex. BRANCH-FOR-6.2).



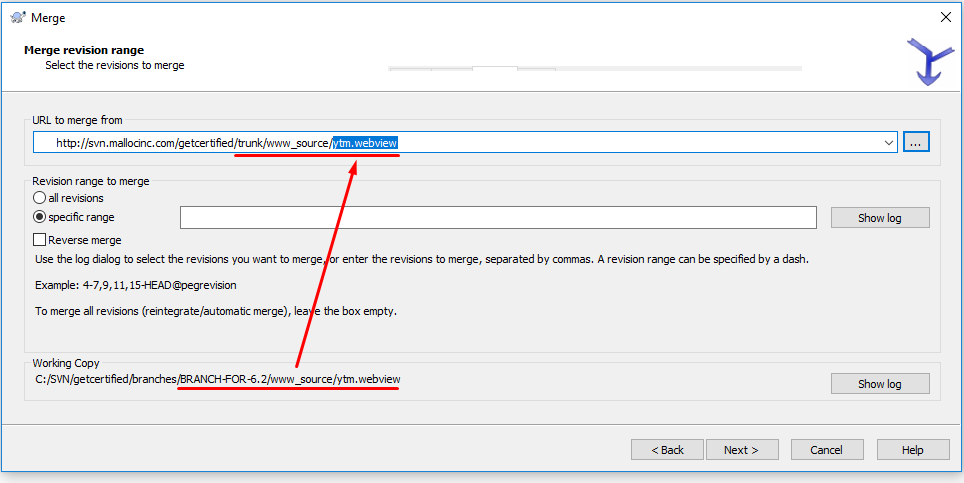
After the type of merge is chosen (which is “Merge a range of revisions” in our case), it is required first to select the URL of the folder from which will merge be done (ex. trunk), and to select the range of revisions to merge.



If only code is going to be merged, then select and choose URL for a source folder (*www\_source*) instead of a whole folder.

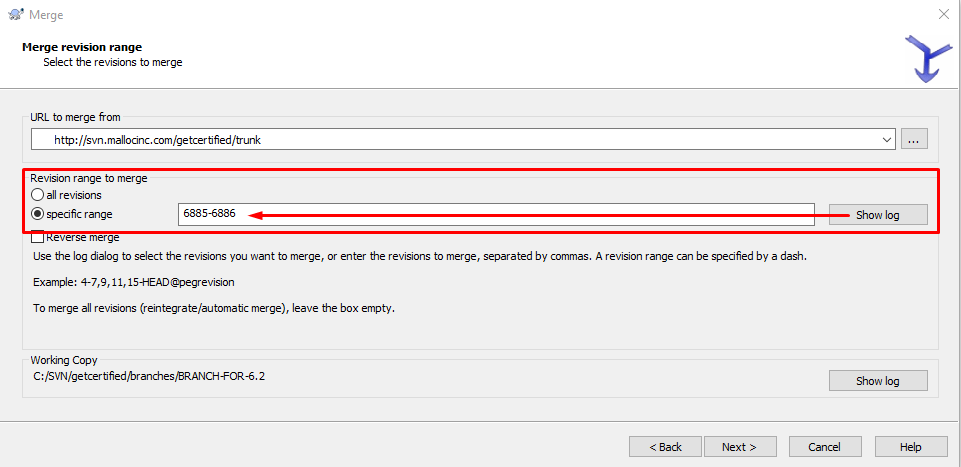
\* Note (optional):

If it is known that there is a huge amount of differences between branches, merge should be done project by project, not the whole (or source) branch folder.

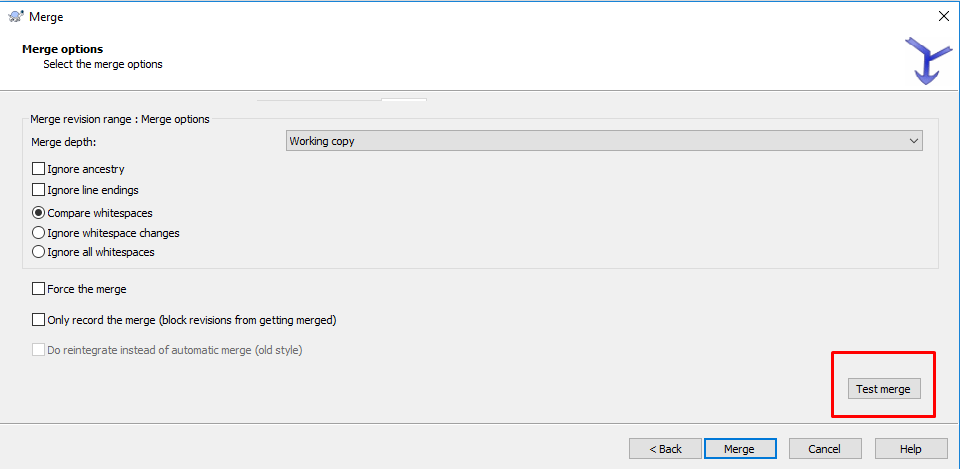


\* Note (optional):

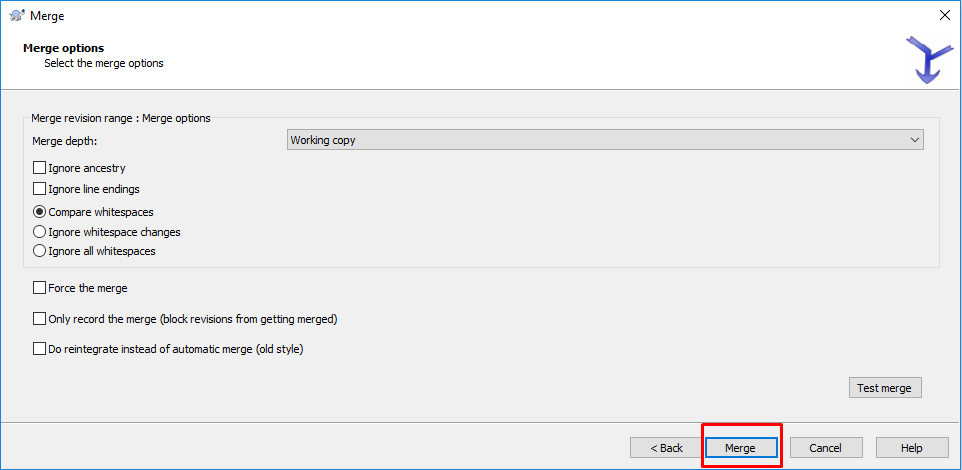
It would be good to perform a merge revision by revision, and note suspicious conflicts somewhere separately. To do so, choose a specific range option, check log and write certain range.



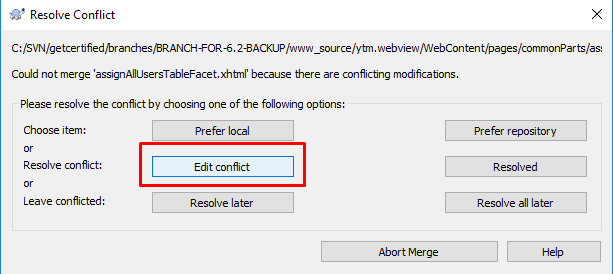
Just before starting Merge process, it is mandatory to perform a Test merge to preview the results of merge simulation. This way, merge success will be checked. Also, potential conflicts will be shown.



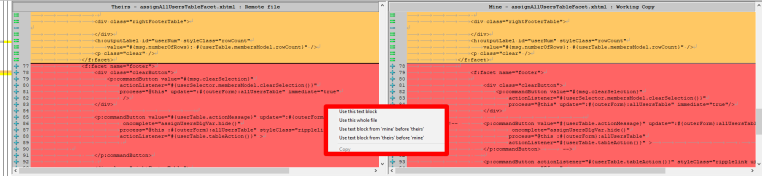
After Test merge is completed successfully, Merge process can start.

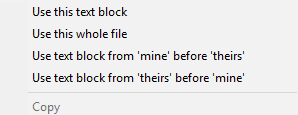


If conflict occurs, it needs to be Edited carefully and then Resolved.



There are few important options to choose which of conflicted blocks (or whole files) will be merged –certain block, whole file, or both blocks (‘theirs’ and ‘mine’ which represent remote file block and working copy block respectively). This must be done for all conflicted blocks.





After finishing with all the conflicted blocks, file should be saved and conflict resolved.

Important rules for resolving certain types of files:

* .*css* should be resolved anyway -> use any of two whole files (repository or local)
* *.scss* must be resolved merging both sides of conflict -> ‘mine’ before ‘theirs’, or ‘theirs before ‘mine’

After the merge process it is mandatory to re-compile css.

If tree conflict occurs, developer who worked on that part should be contacted to check if the current working copy state can be used, or another solution must be provided.

\* NOTE (IMPORTANT):

If you are not sure what are you doing, call your manager!

## Branch to trunk merge

To be sure that the trunk is being kept ‘safe’ whole the time, few processes should be done before merging specific branch to trunk in the following order:

1. Create new branch from trunk
2. Merge your specific branch to the new one
3. Merge trunk to the new one

Now it is safe to perform a final merge to the trunk.

\* Note:

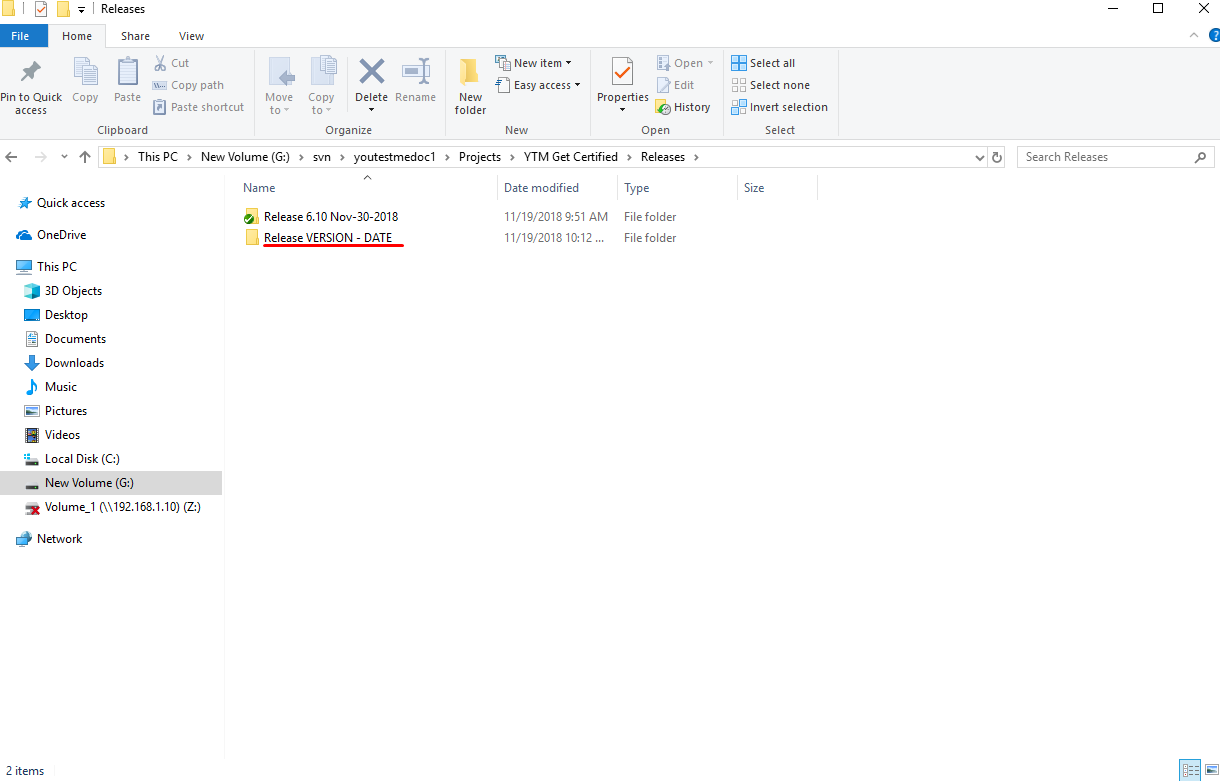
Depending on the situation, merge sometimes needs to be done manually – by copy/paste.

\* Example (SAML implementation merge to trunk):

1. GC-SAML-SINGLE-SIGN-ON (outdated branch made from trunk)
2. create new branch from trunk (GC-NEW-BRANCH)
3. GC-SAML-SINGLE-SIGN-ON merge to GC-NEW-BRANCH
4. merge trunk to GC-NEW-BRANCH
5. merge GC-NEW-BRANCH to trunk

# Release procedure

When planning and working on a new release, the new directory should be created on the following location: youtestmedoc1/trunk/Projects/YTM Get Certified/Releases.



The directory name should follow the predefined standard explained below:

* VERSION – Application version of that particular release (i.e 6.2.0)
* DATE – Expected release date

The release document should be saved inside the following the same naming standard as the directory.

Trunk is used to create a branch for the new release. It is necessary to merge trunk into the branch for the next release daily in order to avoid conflicts and problems.

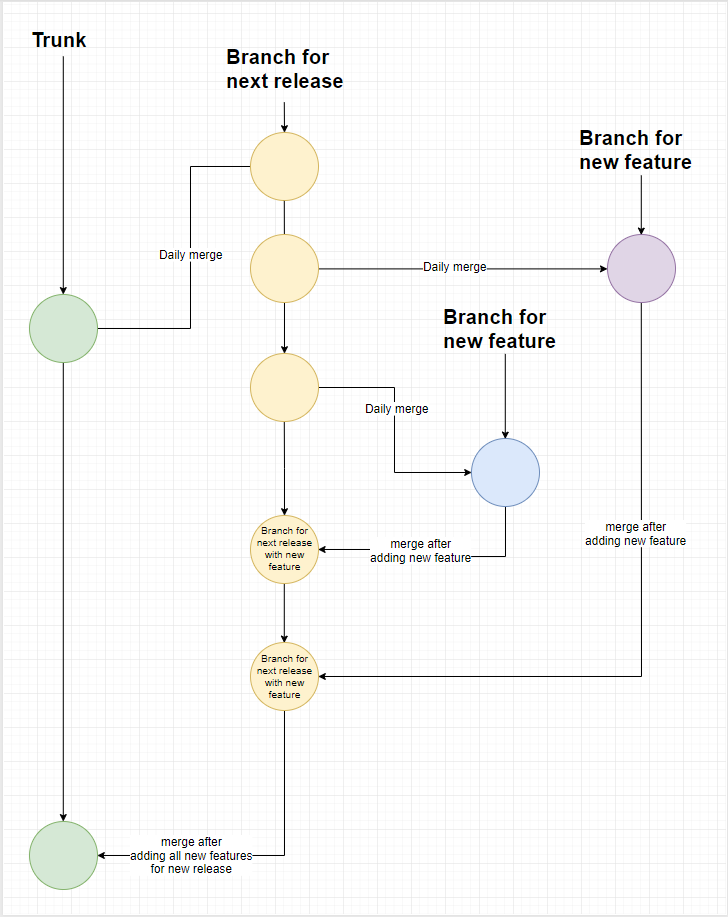
The release supervisor is responsible for the branch for the next release.

To add a new feature, a new branch must be created from the branch for the next release.

It is necessary to merge the branch for the next release into the branch for the new feature daily and solve the conflicts and problems, in order to avoid conflicts and problems during merging branch for a new feature back to the branch for a new release.

All conflicts must be resolved at the branch for the new feature.

When that branch is problem free (added feature and there are no conflict after merge for the next release in that branch), the branch with the new feature can be merged into the branch for the next release and there should be no conflicts and problems.



The developer working on this branch is responsible for the merge from the branch for the next release in the branch and for solving conflicts and problems.

A project/module should never endanger the branch for the next release or trunk. Conflicts should be solved exclusively outside the branch for the next release and trunk.

The document should be created using the template located on the following location:

[GeCertified Release Document Template](http://svn.mallocinc.com/youtestmedoc1/trunk/Projects/YTM%20Get%20Certified/Releases/GetCertified%20Release%20Document%20Template.docx)

The document should be filled with all relevant information about the release.

## Release standard

Release should be limited to a cycle of maximum 3 weeks of coding, to ensure that the releases are as simple as short as possible.

Initially, the release coordinator must be determined, who will be responsible for a release to be clean. Release coordinator has few important responsibilities:

* To create a release plan with the list of new features and its information
* To estimate the release deadline
  + deadline may be prolonged for a period of 1 week maximum (total time - 4 weeks)
  + if task cannot be done in 4 weeks, it should be delegated to another future release depending on a required time, and it must be done on the separate branch
* Not to allow endangering of a release, coordinator must protect the release from the possible conflicts

Responsibilities for the Quality assurance team release members:

* Must be familiar with what will be in a release from the beginning, and should make necessary documentation and “unpleasant” scenarios
* Testing must be at least at 80% of total completion at last week of a deadline

Possible scenarios:

1. special client requests for a demo - development manager (Danilo), seniors and the release coordinator must check if it is the same code that is being changed on a release branch. If it is, then the new branch is being created from the release branch. Developer who is working on that special feature branch must merge release branch to his branch few times a day. This way, possible conflicts will be avoided, or reduced to a minimum.
2. emergency fix - the same procedure (above) goes for creation of a branch from trunk. If an emergency fix is required for example, then it must be merged to the release branch at least once a day.
3. long period task - it must be done on a separate branch, and will be merged in some future release
4. refactoring process conflicts - merge sometimes needs to be done manually (copy-paste) if it is not consistent with a current part of the refactoring process.

Braking rules scenario (example from sample):

- courseClass.xhtml is divided onto courseClass.xhtml, courseInfo.xhtml, courseMembers.xhtml, courseStats.xhtml, courseEastContent.xhtml, courseTitleContent.xhtml…

- coursesController.java is divided onto coursesController.java, courseMembers.java, courseStats.java

- new uniform user assignment system is introduced in users, courses, pools and quizzes

- TOs got its new wrapper class (Entity)

- Modification propagation - course entities are used on many different pages, except on the manage course page itself, so it is required to propagate modifications there also

At the same time required changes are being made on a trunk, which deviate from release standard procedure because of the modifications from an example above.

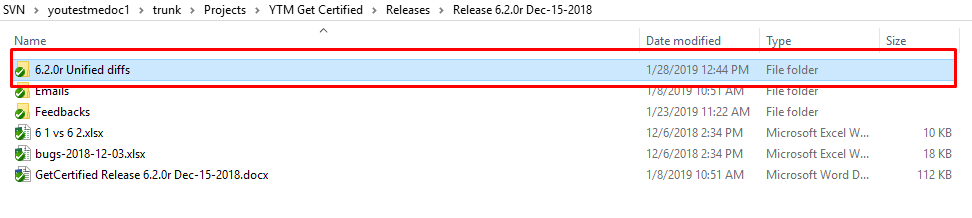
If there is a scenario that is not included in this procedure, then it must be discussed.

# Huge commits rule

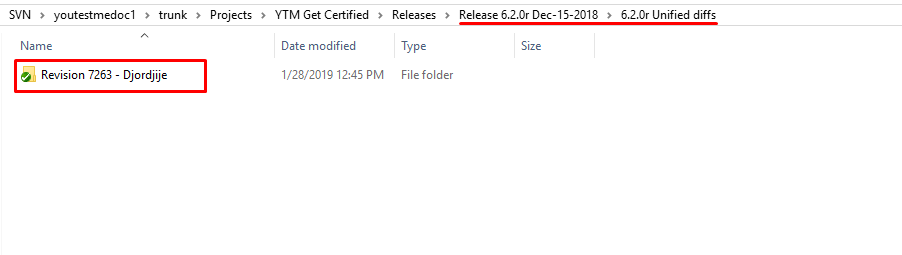
This rule must be respected by all the developers, especially junior ones.

Each time developer is doing a commit with many changes, following steps should be done before a commit in a certain release folder:

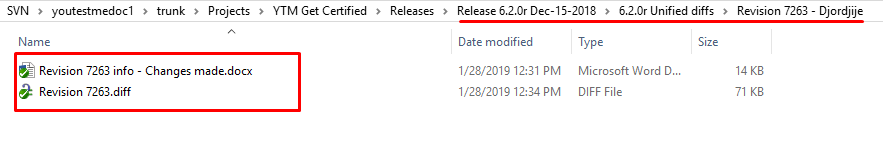
1. Create folder named Unified diffs if it does not exist in a release folder (http://svn.mallocinc.com/youtestmedoc1/trunk/Projects/YTM Get%20Certified/Releases)



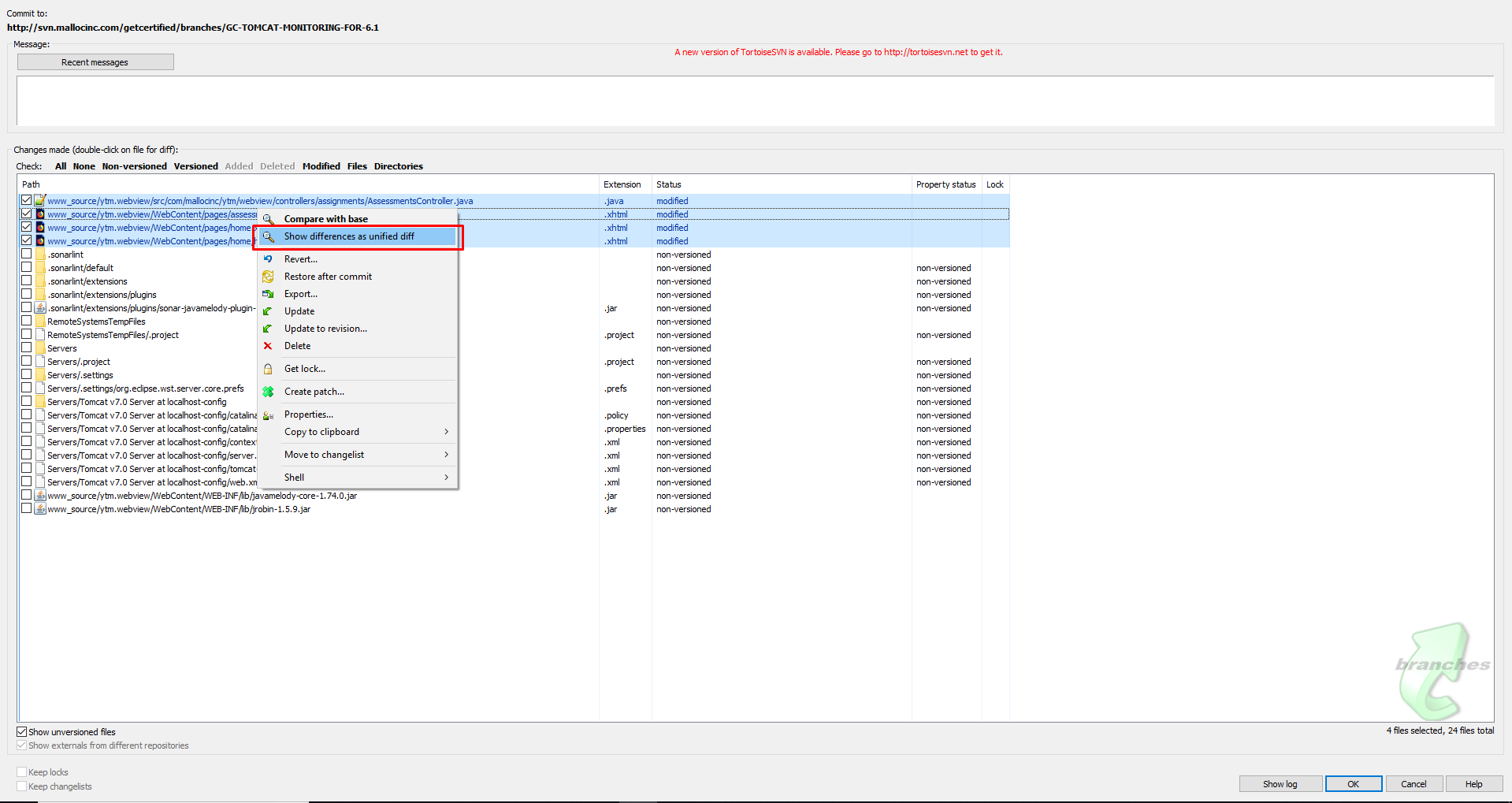
1. Create folder inside Unified diffs folder with a commit revision number (add number after the commit) and commit author name/surname.



1. Inside this folder create a word file with the information about the added/modified/deleted code, and the revision unified diff file. Both should be named with a revision number inside their name.



To save the revision unified diff file, select modified files and “Show differences as unified diff”:



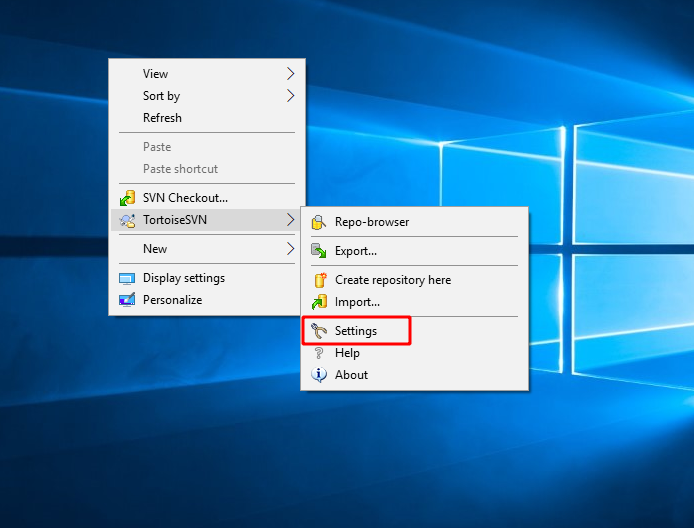
At this time junior developers should go through code with their superiors before doing a commit!

# Remote SVN Connection

Connection to SVN outside office is possible by using proxy or some of the infrastructure VPN services. Infrastructure VPN is any VPN provdided by YouTestMe.

Procedure for connecting to SVN via proxy:

1. Right-click anywhere. Go to Tortoise SVN > Settings



1. In Settings windows, click on Network
2. Check “Enable proxy server”
3. Enter username and password and click OK

