Git Flow Implementation in NPP

Navetti AB
E-mail:info@navetti.com
Phone: +46 8 44 00 120

Contents

1 Introduction ... 2
1.1 Purpose of the document ... 2
2 Git flow processes ... 3
3 Branching ... 5
3.1 Create new branch ... 5
3.1.1 Create new branch with TortoiseGit ... 5
3.1.2 Create new branch with Git Bush ... 6
3.2 Checkout branch and switch to the branch ... 7
3.2.1 Checkout branch and switch to the branch with TortoiseGit ... 7
3.2.2 Checkout branch and switch to the branch with Git Bush ... 10
3.3 Merging ... 11
3.3.1 Pull Request ... 11
3.3.2 Merging with Git Bash ... 12
3.3.3 Merging with TortoiseGit ... 13
3.4 Delete branch ... 15
3.4.1 Delete branch with TortoiseGit ... 15
3.4.2 Delete branch with Git Bush ... 17
4 Type of branches in NPP ... 18
4.1 Feature branch ... 18
4.2 Hotfix branch ... 18
4.2.1 Hotfix made by Git Bash ... 19
4.2.2 Hotfix made by TortoiseGit ... 19
4.3 Release Branch ... 21
4.4 Tagging ... 22
4.4.1 Create Tag by TortoiseGit ... 22
4.4.2 Create Tag by Git Bush ... 24
4.4.3 Get Remote Tags ... 24
5 Tracking Branches ... 25
6 Document history ... 26

1 Introduction

Navetti Business Navigator – Git flow process manual.

1.1 Purpose of the document

The purpose of this document is to describe guide for effectively using of Git Flow.

2 Git flow processes

Git flow process is based on branching. Branching incorporates:

1. **Create branch** (click on following link)
2. **Merge branch** (click on following link)
3. **Delete branch** (click on following link)

Git flow process for NPP4 consisted of few different branches

1. **Master branch-** one of the main branches that never dies. Hotfix branches has to be derived from master branch. Direct changes on master branch are not recommended. Changes from release and hotfix branches are merging on master branch.

2. **Develop branch** – one of the main branches that never dies. Feature and release branches has to be derived from develop branch. Direct changes on develop branch are not recommended. Changes from feature branches, release branches, and hotfix branches are merging on master branch.

3. **Feature branch** – one of the short term branches, which are alive while they are needed. This branch has to be derived from develop branch at the moment when new functionality has to be develop.

4. **Release branch** – one of the short term branches, which are alive while they are needed. This branch has to be derived from develop branch at the moment when all functionalities from the sprint are done.

5. **Hotfix branch** – one of the short term branches, which are alive while they are needed. This branch has to be derived from master branch at the moment when some small fix has to be done on the specific tagged version on master branch.

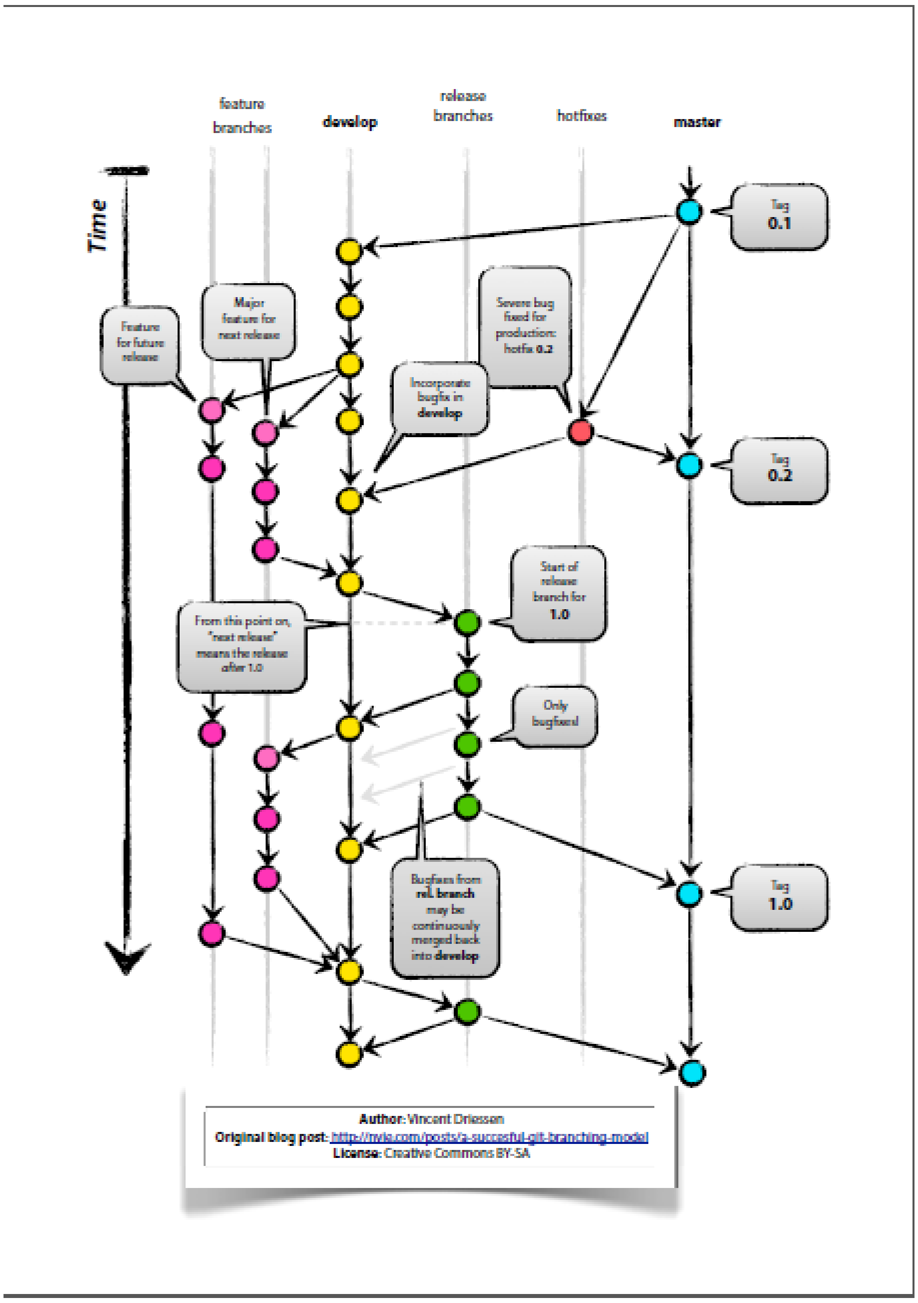

feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Tag 1.0
Author: Vincent Driessen
Original blog post: http://nvie.com/posts/a-succesful-git-branching-model
License: Creative Commons BY-SA

Figure 1: Git flow process

3 Branching

3.1 Create new branch

3.1.1 Create new branch with TortoiseGit

New branch can be created with Tortoise Git from Windows Explorer.

Please follow the instructions below:

1. Locate the folder where the project is checked out.
2. Right click with the mouse and choose **TortoiseGit -> Create Branch…**

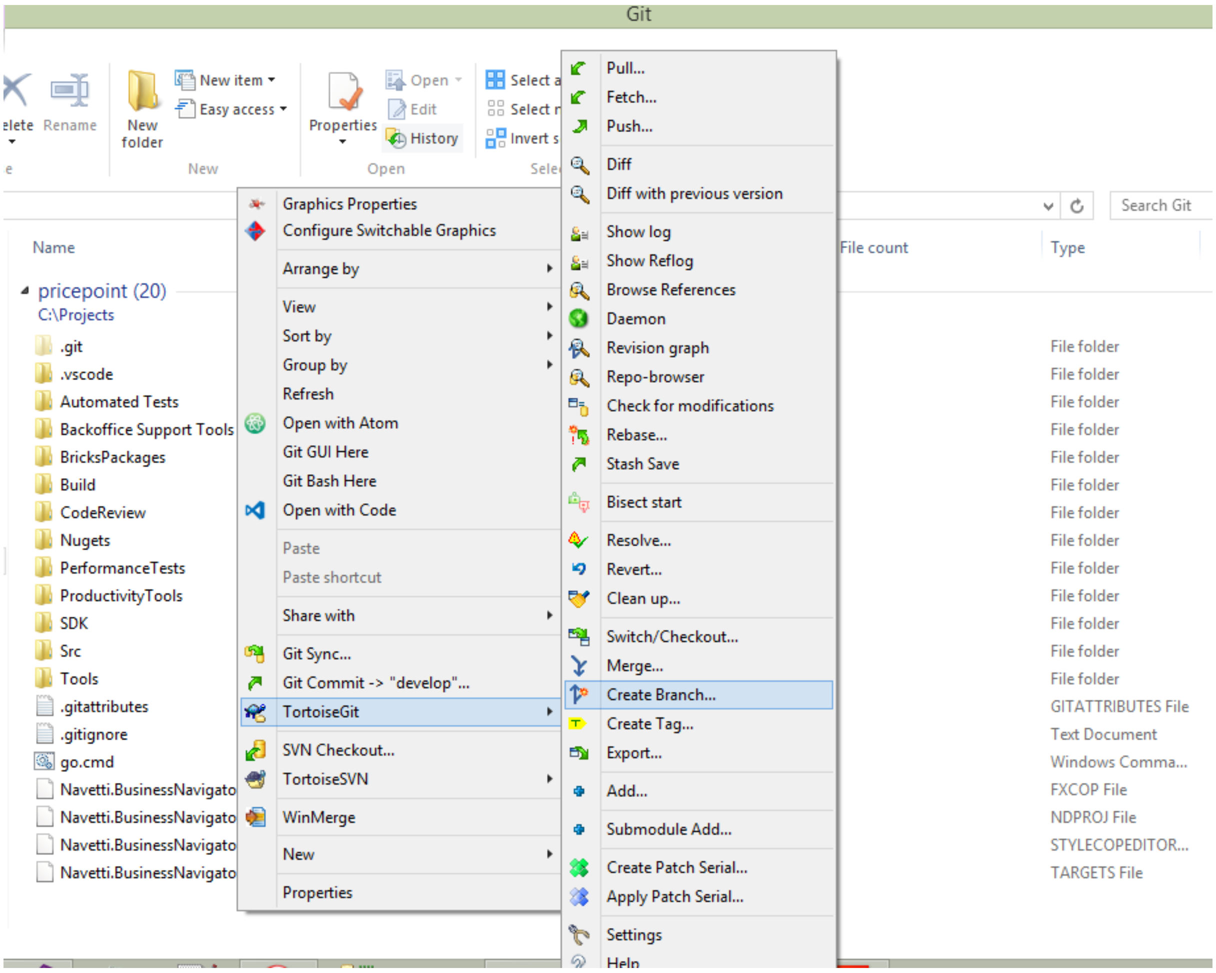
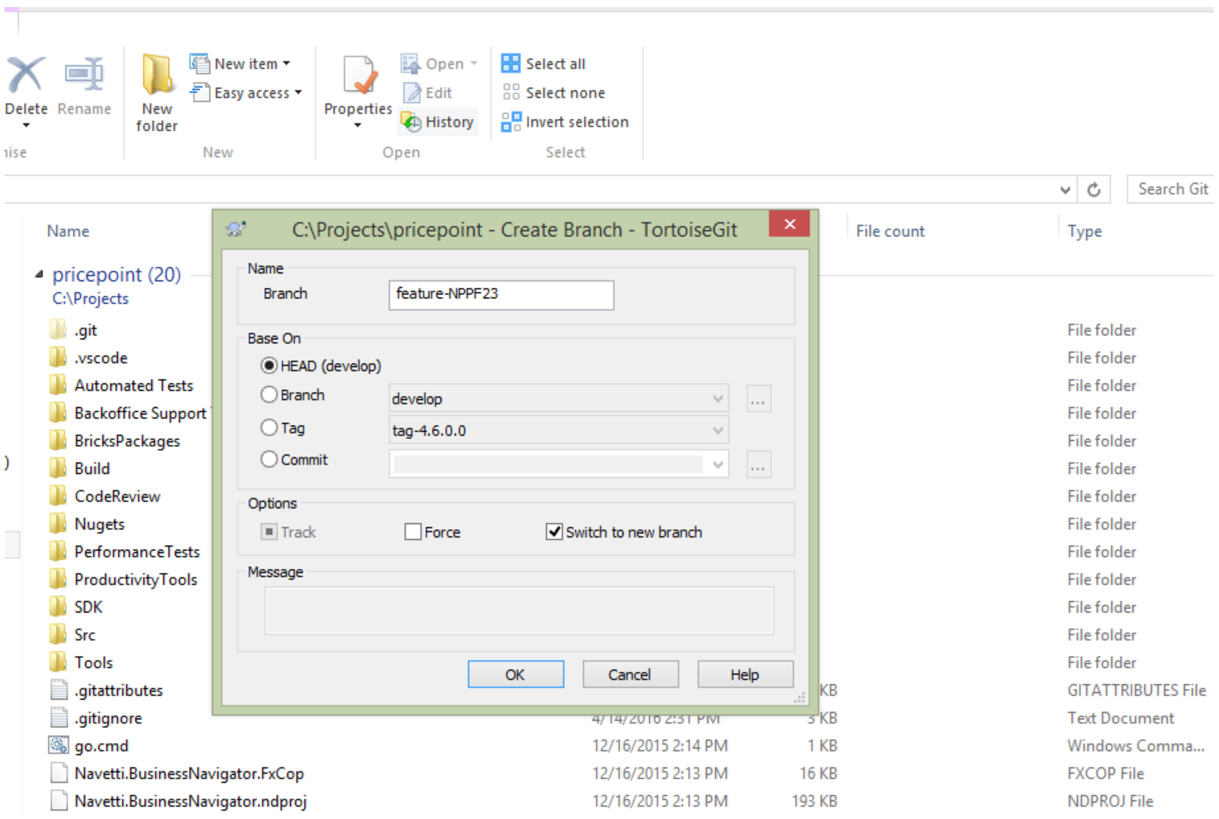


Figure 2: Branching with Tortoise GIT from Windows explorer

3. Set the name of the new branch in the **Branch** field. From the **Base On** section, chose from which branch the newly created branch will be derived (branch head/branch/tag/commit). If you check **Switch to new branch**, after click on **OK**, your local copy will be switched to newly created branch.



C:\Projects\pricepoint - Create Branch - TortoiseGit

Name
Branch: feature-NPPF23

Base On
(●) HEAD (develop)
( ) Branch: develop
( ) Tag: tag-4.6.0.0
( ) Commit:

Options
[■] Track
[ ] Force
[✓] Switch to new branch

Message

OK | Cancel | Help

Name			File count	Type
pricepoint (20) C:\Projects				
.git				File folder
.vscode				File folder
Automated Tests				File folder
Backoffice Support				File folder
BricksPackages				File folder
Build				File folder
CodeReview				File folder
Nugets				File folder
PerformanceTests				File folder
ProductivityTools				File folder
SDK				File folder
Src				File folder
Tools				File folder
.gitattributes		KB		GITATTRIBUTES File
.gitignore	4/14/2016 2:31 PM	3 KB		Text Document
go.cmd	12/16/2015 2:14 PM	1 KB		Windows Comma...
Navetti.BusinessNavigator.FxCop	12/16/2015 2:13 PM	16 KB		FXCOP File
Navetti.BusinessNavigator.ndproj	12/16/2015 2:13 PM	193 KB		NDPROJ File

Figure 3: Branching with Tortoise Git from Windows explorer

With these steps new branch will be created in the local repository. If this branch should be shared with others, it should also exist in the remote repository. For this purpose follow the steps bellow:

4. Switch to the newly created branch (with **Switch to new branch** from previous step, or follow instruction in this section)
5. Right click with the mouse and choose **TortoiseGit -> Push…**

3.1.2 Create new branch with Git Bush

New branch can be created with Git Bush command line. Commands and instruction used for creating new branch are listed below:

1. Switch to the branch from where the new branch will be derived:
   ```
   git checkout BaseBranchName
   ```
2. Create the new branch.
   ```
   git branch NewBranchName
   ```
3. Switch to the newly created branch.
   ```
   git checkout NewBranchName
   ```
4. Push the new branch to the remote repository
   ```
   git push origin NewBranchName
   ```

3.2 Checkout branch and switch to the branch

3.2.1 Checkout branch and switch to the branch with TortoiseGit

Local Branches can be switched with Tortoise Git from Windows Explorer.

Please follow the instructions:

1. Locate the folder where the project is checked out.
2. Right click with the mouse and choose **TortoiseGit -> Switch/Checkout…**

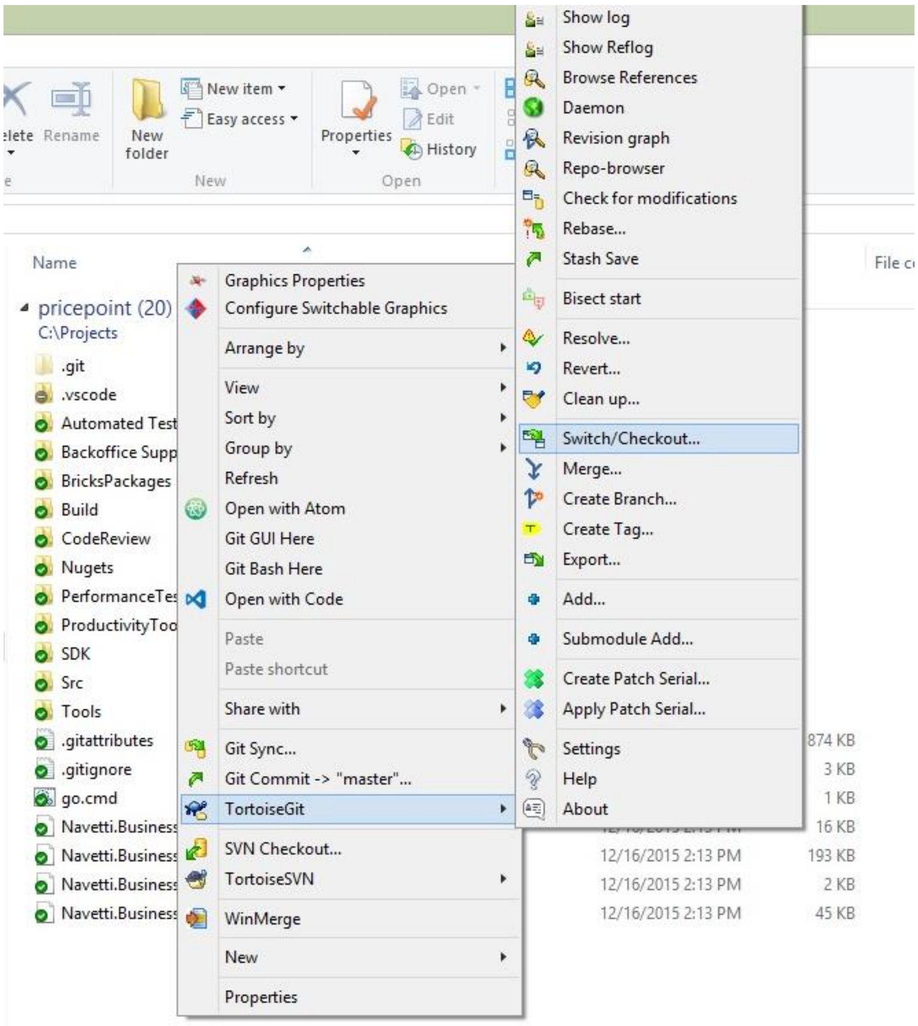
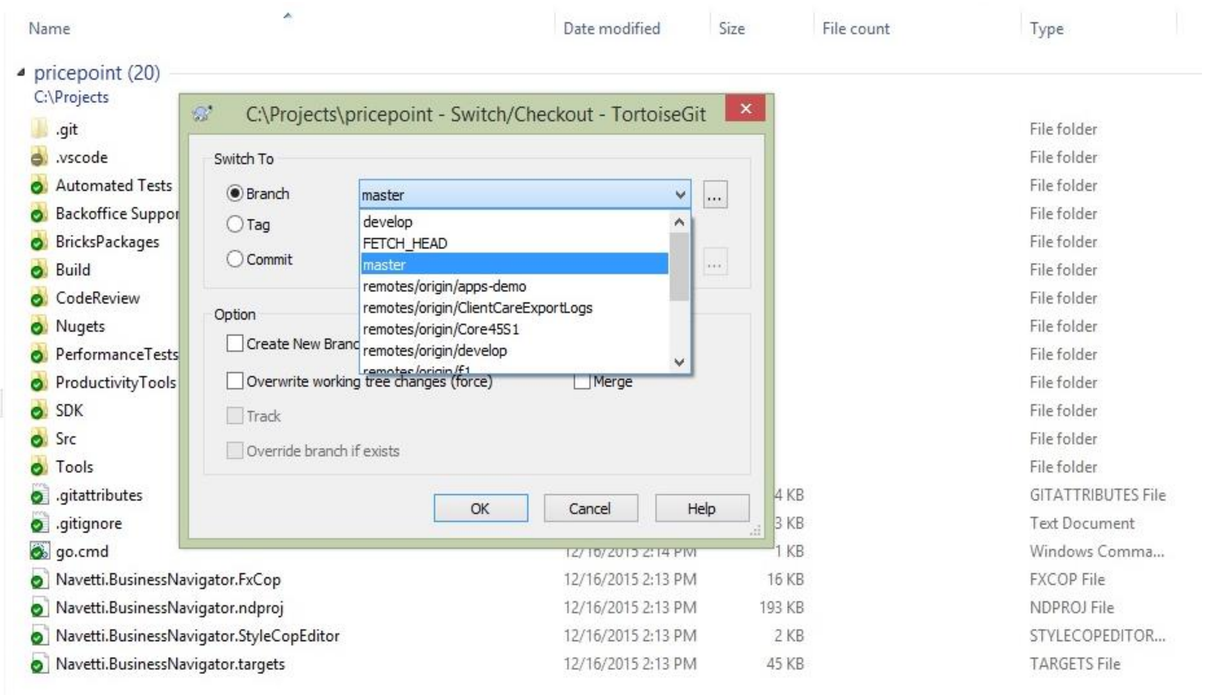


Figure 4: Switch branches with Tortoise Git from Windows explorer

3. Choose **Branch** option from **Switch To** section, then from the dropdown select the desired branch. Note, the branches which do not start with “remotes/origin/…” are branches from your local repository. The other ones are the remote branches.



pricepoint (20)
C:\Projects

Name	Date modified	Size	File count	Type
.git				File folder
.vscode				File folder
Automated Tests				File folder
Backoffice Suppor				File folder
BricksPackages				File folder
Build				File folder
CodeReview				File folder
Nugets				File folder
PerformanceTests				File folder
ProductivityTools				File folder
SDK				File folder
Src				File folder
Tools				File folder
.gitattributes		4 KB		GITATTRIBUTES File
.gitignore		3 KB		Text Document
go.cmd	12/16/2015 2:14 PM	1 KB		Windows Comma...
Navetti.BusinessNavigator.FxCop	12/16/2015 2:13 PM	16 KB		FXCOP File
Navetti.BusinessNavigator.ndproj	12/16/2015 2:13 PM	193 KB		NDPROJ File
Navetti.BusinessNavigator.StyleCopEditor	12/16/2015 2:13 PM	2 KB		STYLECOPEDITOR...
Navetti.BusinessNavigator.targets	12/16/2015 2:13 PM	45 KB		TARGETS File

C:\Projects\pricepoint - Switch/Checkout - TortoiseGit

Switch To

[✓ Branch] master
Tag
Commit

develop
FETCH_HEAD
master
remotes/origin/apps-demo
remotes/origin/ClientCareExportLogs
remotes/origin/Core45S1
remotes/origin/develop

Option

Create New Branc
Overwrite working tree changes (force)
Merge
Track
Override branch if exists

Figure 5: Switch to local branches with Tortoise Git from Windows explorer

Checkout and switch to remote the branch can be done with Tortoise Git from Windows Explorer.
Please follow the instruction:

1. Locate the folder where the project is checked out.
2. Right click with the mouse and choose **TortoiseGit -> Switch/Checkout…**

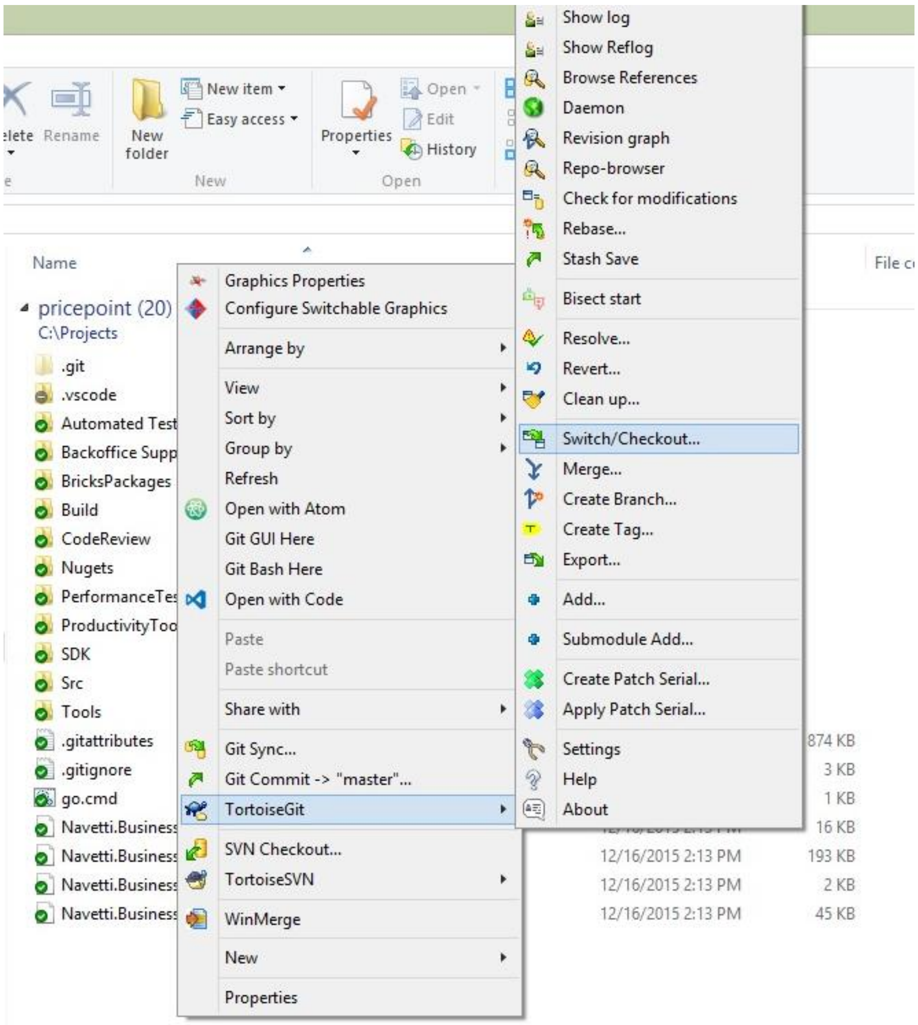
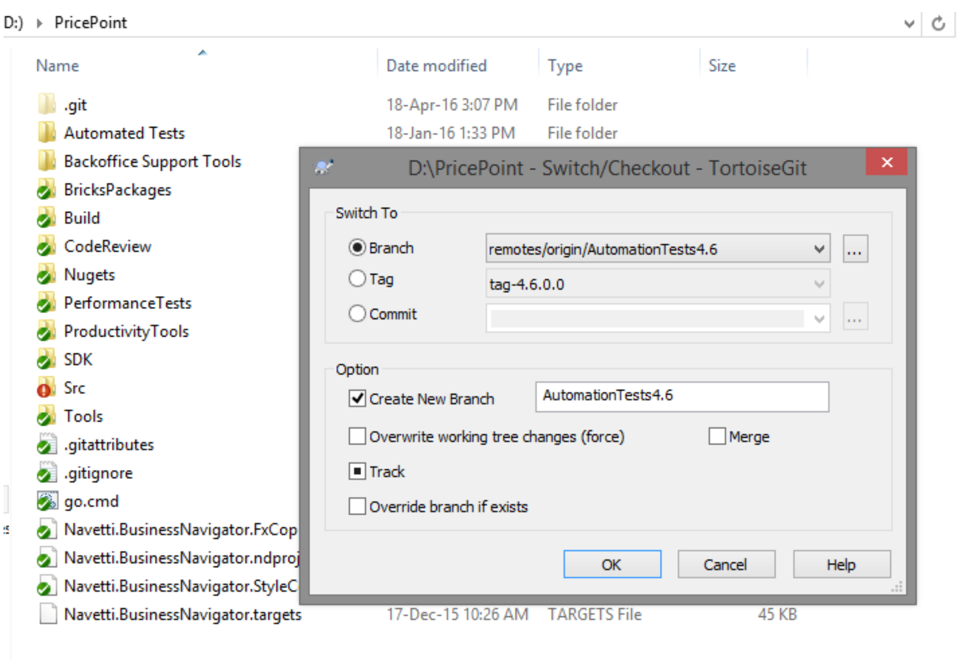


Figure 6: Switch branches with Tortoise Git from Windows explorer

3. Choose **Branch** option from **Switch To** section, then from the dropdown select the desired branch (remote branches names start with “remotes/origin/…”). Note that **Create New Branch** option is selected and the text filed is populated with the name of the branch. After pressing **OK**, the local repository will be switched to AutomationTests4.6 branch.



D:) ▸ PricePoint

Name	Date modified	Type	Size
.git	18-Apr-16 3:07 PM	File folder	
Automated Tests	18-Jan-16 1:33 PM	File folder	
Backoffice Support Tools			
BricksPackages			
Build			
CodeReview			
Nugets			
PerformanceTests			
ProductivityTools			
SDK			
Src			
Tools			
.gitattributes			
.gitignore			
go.cmd			
Navetti.BusinessNavigator.FxCop			
Navetti.BusinessNavigator.ndproj			
Navetti.BusinessNavigator.StyleC			
Navetti.BusinessNavigator.targets	17-Dec-15 10:26 AM	TARGETS File	45 KB

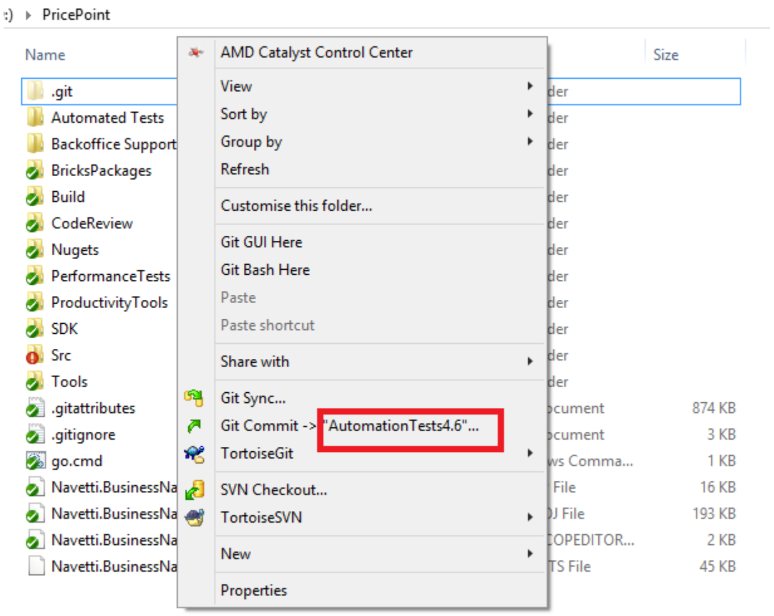
D:\PricePoint - Switch/Checkout - TortoiseGit

Switch To
- [✓ Branch] remotes/origin/AutomationTests4.6
- Tag: tag-4.6.0.0
- Commit

Option
- [✓] Create New Branch: AutomationTests4.6
- [ ] Overwrite working tree changes (force)
- [ ] Merge
- [■] Track
- [ ] Override branch if exists

Figure 7: Checkout and switch to new branch with Tortoise Git from Windows explorer

With Tortoise Git can be validated which is the current branch.



:) ▸ PricePoint

Git Commit -> "AutomationTests4.6"...

Figure 8: Check the current branch with Tortoise Git from Windows explorer

3.2.2 Checkout branch and switch to the branch with Git Bush

Follow the instructions to checkout the branch and switch to with Git Bush

1. Switch to branch, with name branchName:

```
git checkout branchName
```

2. Checkout remote branch, with name branchName, and switch to it:

```
git pull origin branchName
git checkout branchName
```

3. Make sure that current loaded branch is designated branch. Following command show the working tree status and the current branch

```
git status
```

4. List all branches. The current branch in the list is marked with *

```
git branch
```

3.3 Merging

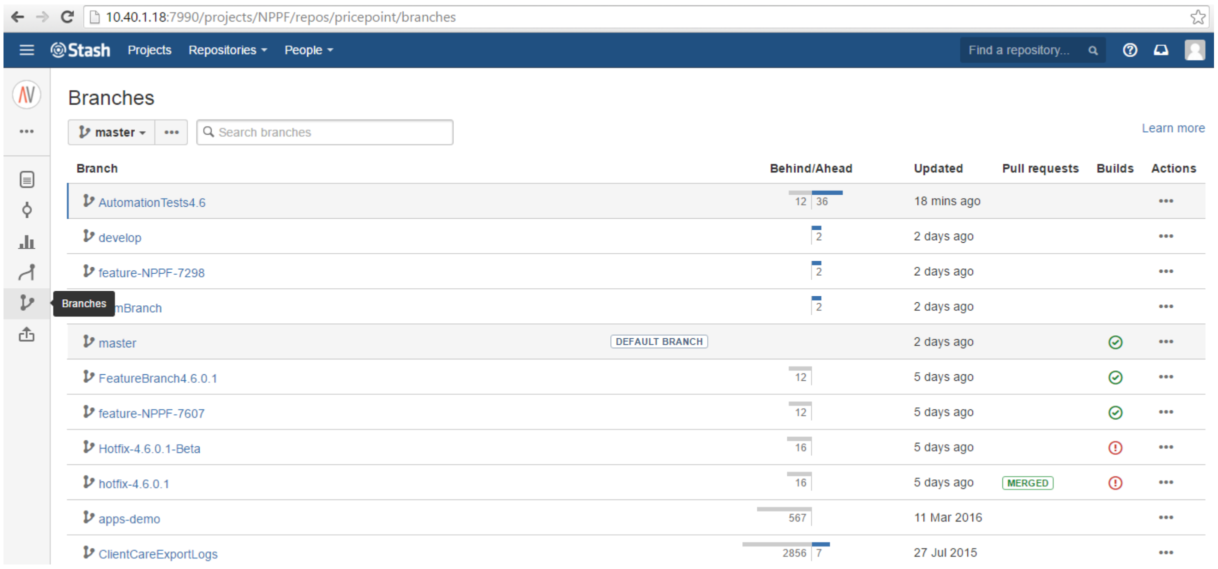
Merging allows two branches to be integrated into one single branch.

3.3.1 Pull Request

Pull request is highly recommended step before merging two branches. This step only can be done on remote server on Stash (0.40.1.18:7990/projects/NPPF/repos/pricepoint). Please follow instructions bellow:

1. Open Stash, open the link http://10.40.1.18:7990/projects/NPPF/repos/pricepoint/browse

2. From the left menu select branches

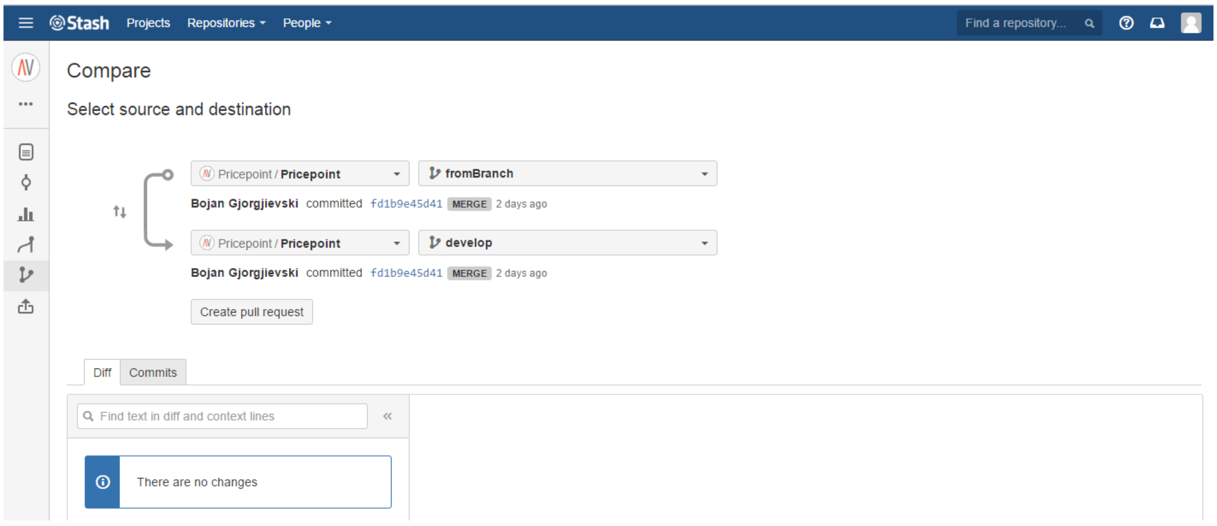


Branch		Behind/Ahead	Updated	Pull requests	Builds	Actions
AutomationTests4.6		12 | 36	18 mins ago			•••
develop		2	2 days ago			•••
feature-NPPF-7298		2	2 days ago			•••
mBranch		2	2 days ago			•••
master	DEFAULT BRANCH		2 days ago		✓	•••
FeatureBranch4.6.0.1		12	5 days ago		✓	•••
feature-NPPF-7607		12	5 days ago		✓	•••
Hotfix-4.6.0.1-Beta		16	5 days ago		!	•••
hotfix-4.6.0.1		16	5 days ago	MERGED	!	•••
apps-demo		567	11 Mar 2016			•••
ClientCareExportLogs		2856 | 7	27 Jul 2015			•••

Figure 9: List branches on Stash

3. From the list with branches select the branch which need to be merged

4. In the Compare page make sure that both branches (“from” branch and “to” branch) are appropriately selected



Compare

Select source and destination

Pricepoint / Pricepoint — fromBranch

Bojan Gjorgjievski committed fd1b9e45d41 MERGE 2 days ago

Pricepoint / Pricepoint — develop

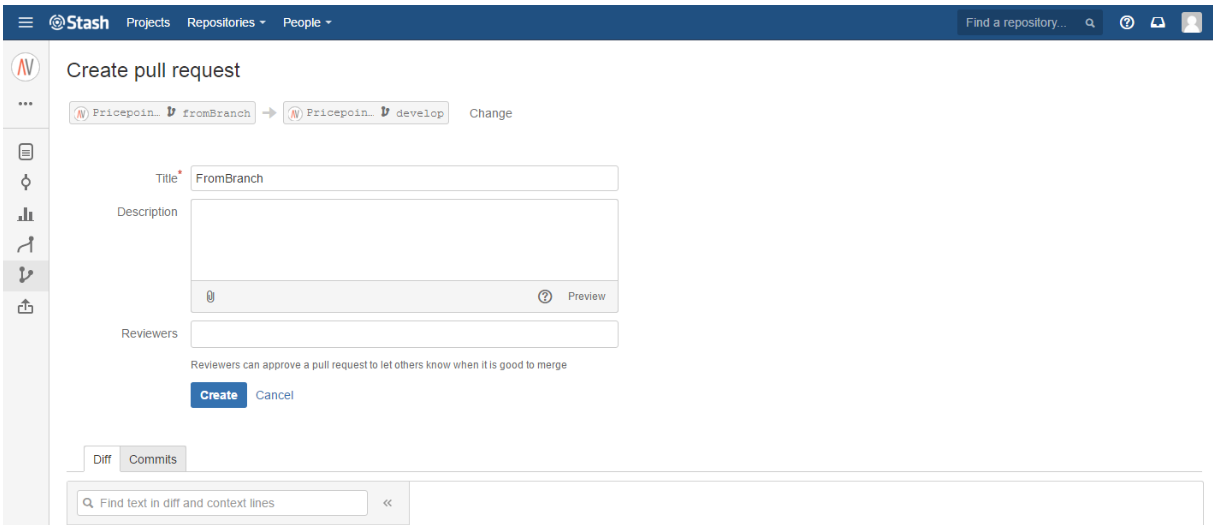
Bojan Gjorgjievski committed fd1b9e45d41 MERGE 2 days ago

There are no changes

Figure 10: Review changes between two branches

5. Click on Diff tab in the table below and revise changes, make sure that conflicts are not present
6. Check that following changes should not be merged
    a. Connection Strings form config files
    b. Msmq instance
    c. Thumbprint
    d. URLs of the application

7. Click on Button Create pull request
8. Set the reviewers of the pull request



Create pull request

Pricepoin... fromBranch → Pricepoin... develop

Title*: FromBranch

Description:

Reviewers:

Reviewers can approve a pull request to let others know when it is good to merge

Figure 11: Set Title, Description and Reviewer for Pull request

9. List Pull requests to check if the pull request is approved by reviewers. As pull request is approved the merging can be started (refer to section for merging branches).

3.3.2 Merging with Git Bash

Follow the instructions bellow for merging branches, merging on develop or master branch will be very common situation in future. In following text will be distinguished branches that need to be merged with “from” branch and “to” branch

1. Make sure the “from” branch have already gotten approved by reviewers of Pull Request, if not make Pull Request for merging (follow the instructions for Pull Request)

2. Checkout the “to” branch (for ex. master or develop branch)

3. Make sure that local copy of the “to” branch is up to date otherwise make pull then commit and push all uncommitted changes

4. Merge “from” branch to “to” branch
```
$ git merge --no-ff fromBranch toBranch –m “Merge: Merge message”
```

6. Check if merging passed without conflicts
```
$ git status
```

7. Push merged changes of “to” branch
```
$ git push origin toBranch
```

3.3.3 Merging with TortoiseGit

1. Locate the folder where the project is checked out.

2. Make Pull request for the branch that need to be merged (follow the instructions for Pull Request) and wait to be approved by reviewers

3. Review the changes between two branches to avoid the conflicts (check difference between the branches on Stash). Make sure that both branches are up to date on remote server before reviewing differences

4. Checkout the “to” branch (follow the instructions for Checkout Branch)

5. Make sure that local copy of the “to” branch is up to date otherwise make pull then commit and push all uncommitted changes

6. Merge “from” branch to “to” branch

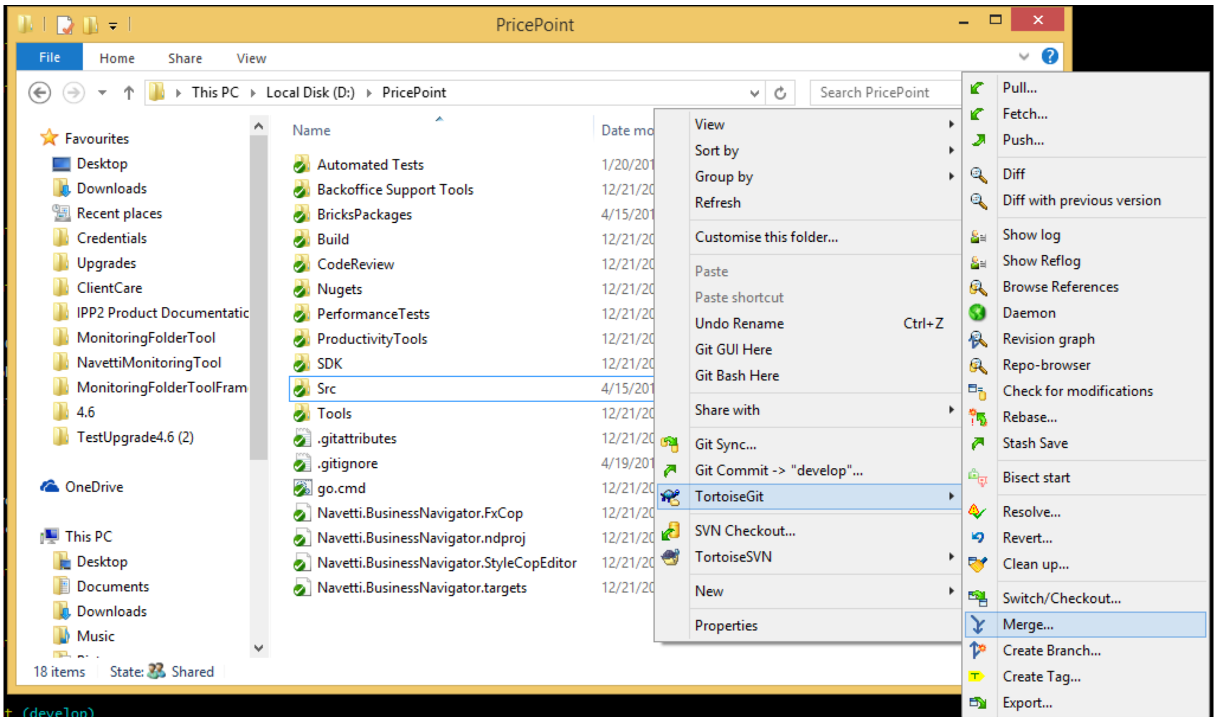
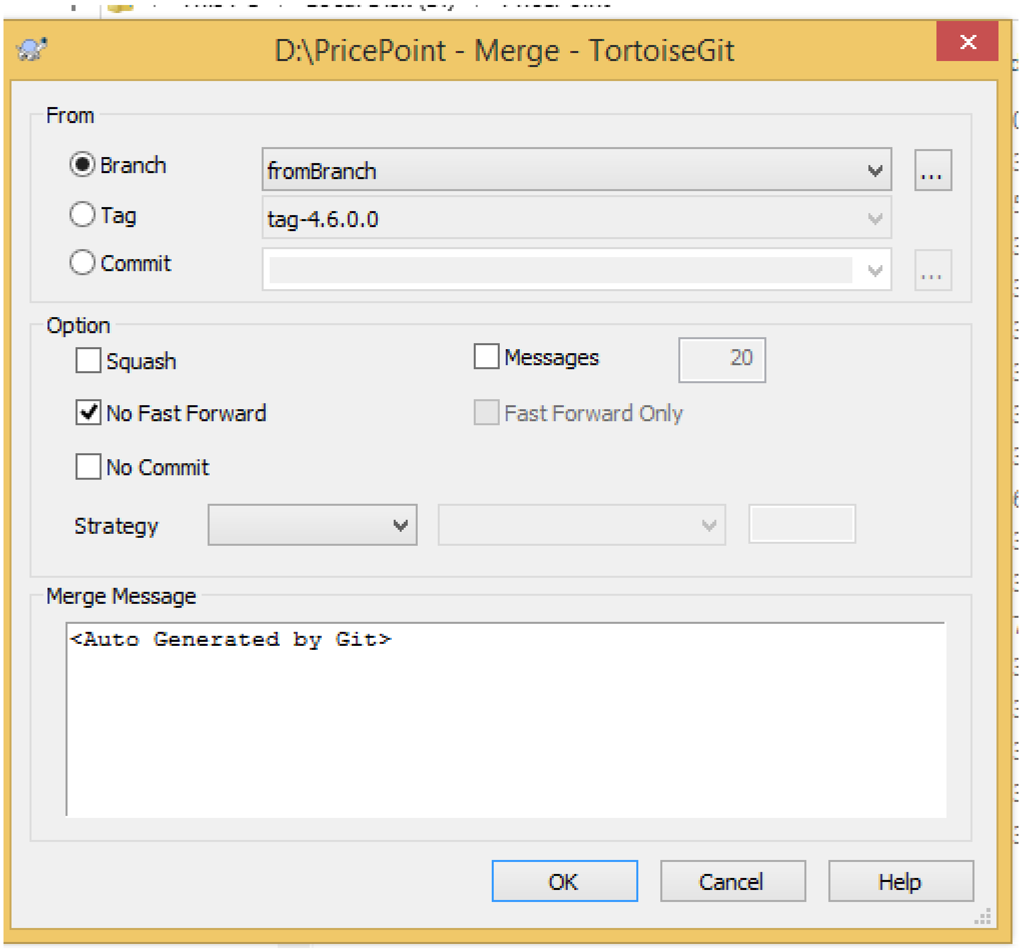


Figure 12: Merge with TortoiseGit from Windows Explorer



(●) Branch: fromBranch
( ) Tag: tag-4.6.0.0
( ) Commit

[ ] Squash
[✓] No Fast Forward
[ ] No Commit
[ ] Messages: 20
[ ] Fast Forward Only

```
<Auto Generated by Git>
```

Figure 13: Merge settings of TortoiseGit from Windows Explorer

7. (Optional) Resolve conflicts, if conflicts were occurred during merging

8. Push merged changes of “to” branch

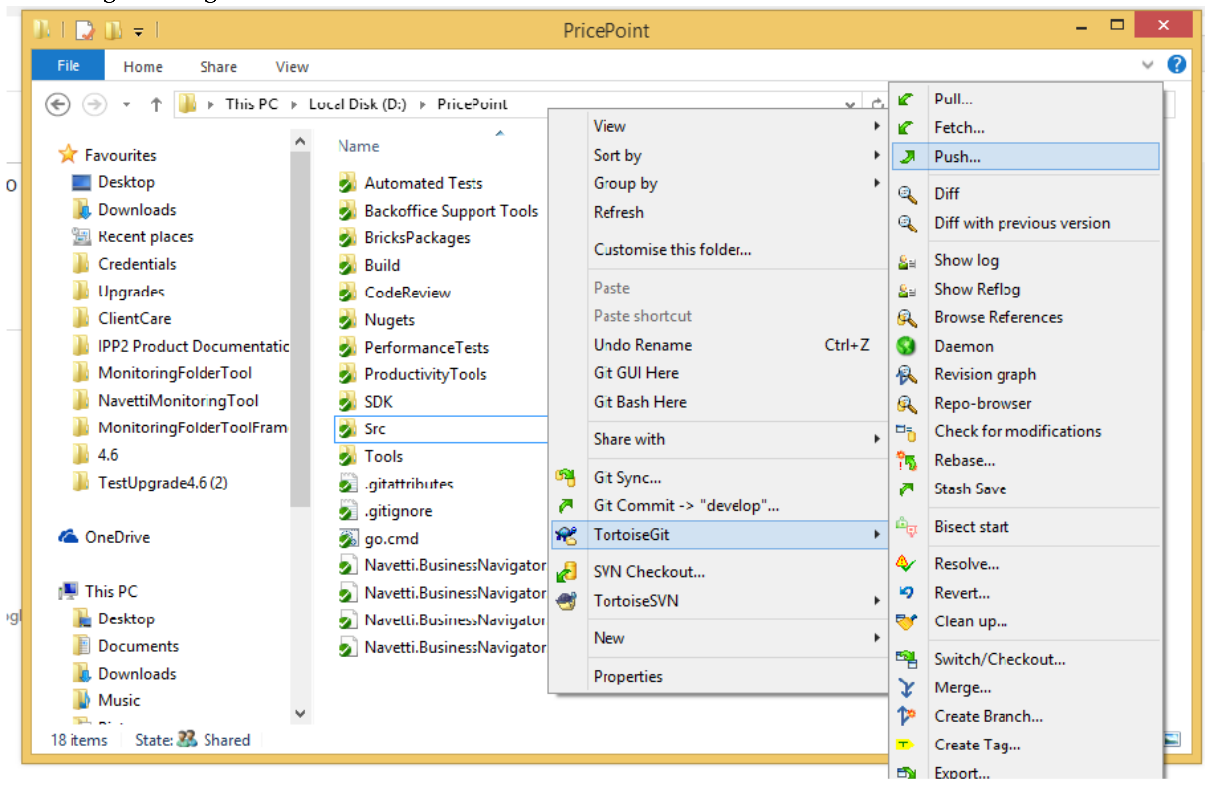


Figure 14: Push with TortoiseGit on Windows Explorer

3.4 Delete branch

3.4.1 Delete branch with TortoiseGit

To delete a branch from local repository with Tortoise Git, please follow the instruction bellow:

1. Locate the folder where the project is checked out.
2. Right click with the mouse and choose **TortoiseGit -> Switch/Checkout...**

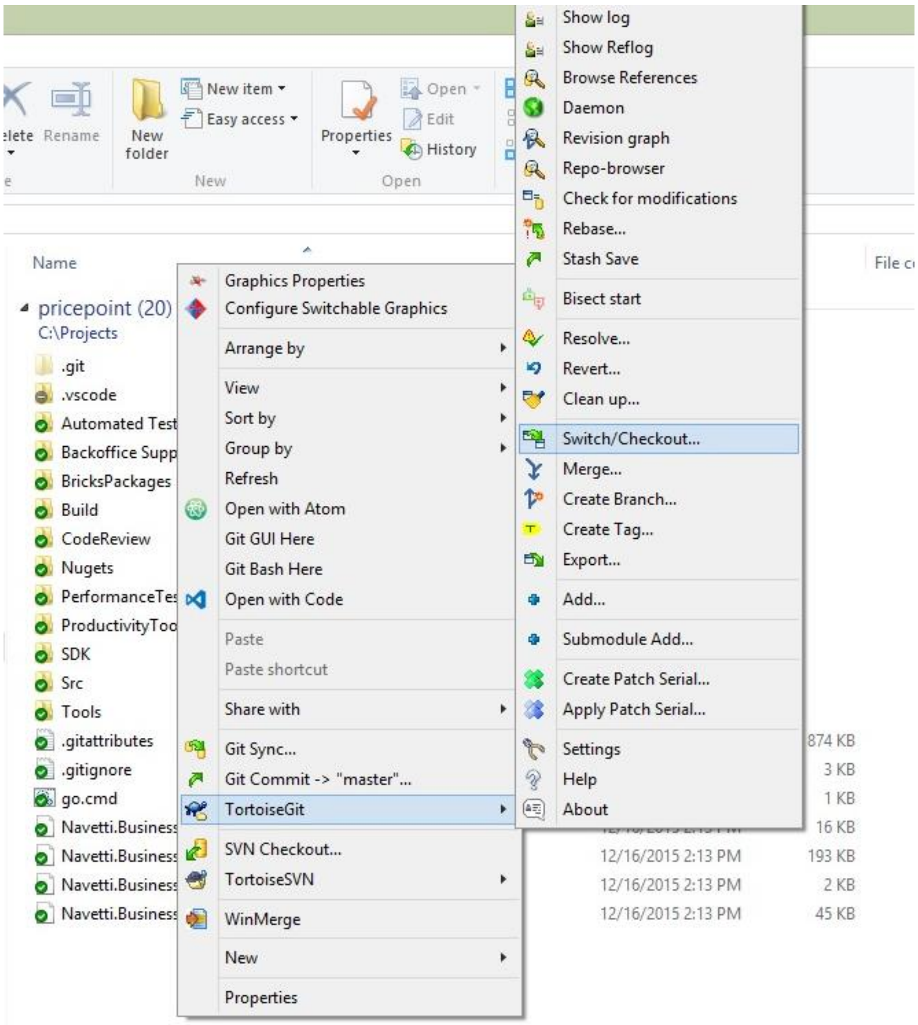
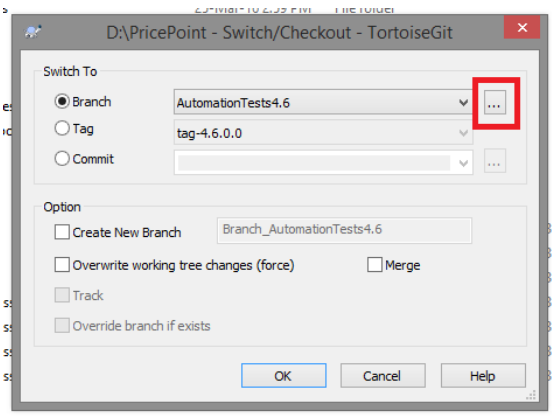


Figure 15: Switch branches with Tortoise Git from Windows explorer


D:\PricePoint - Switch/Checkout - TortoiseGit

Switch To

[✓ Branch] AutomationTests4.6 ...
Tag tag-4.6.0.0
Commit

Option

Create New Branch Branch_AutomationTests4.6
Overwrite working tree changes (force)
Merge
Track
Override branch if exists

OK Cancel Help

Figure 16: Delete branch with Tortoise Git from Windows explorer

3. Delete local branch.

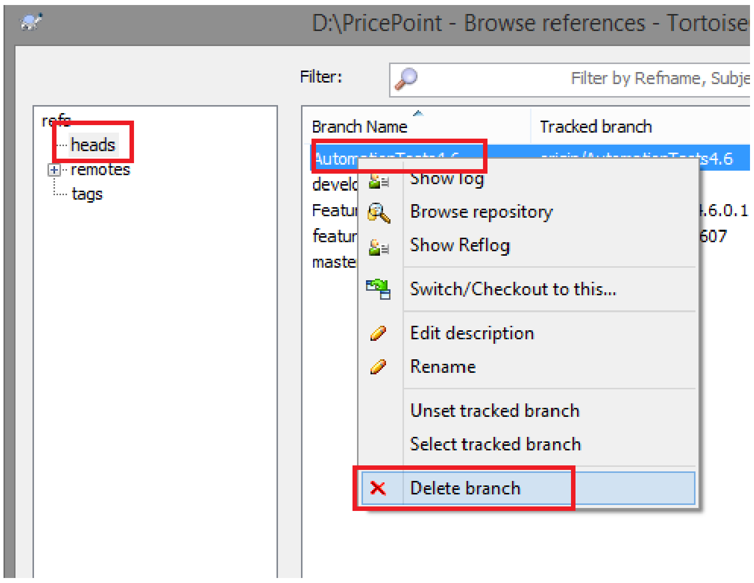


Figure 17: Delete local branch with Tortoise Git from Windows explorer

4. Delete remote branch.

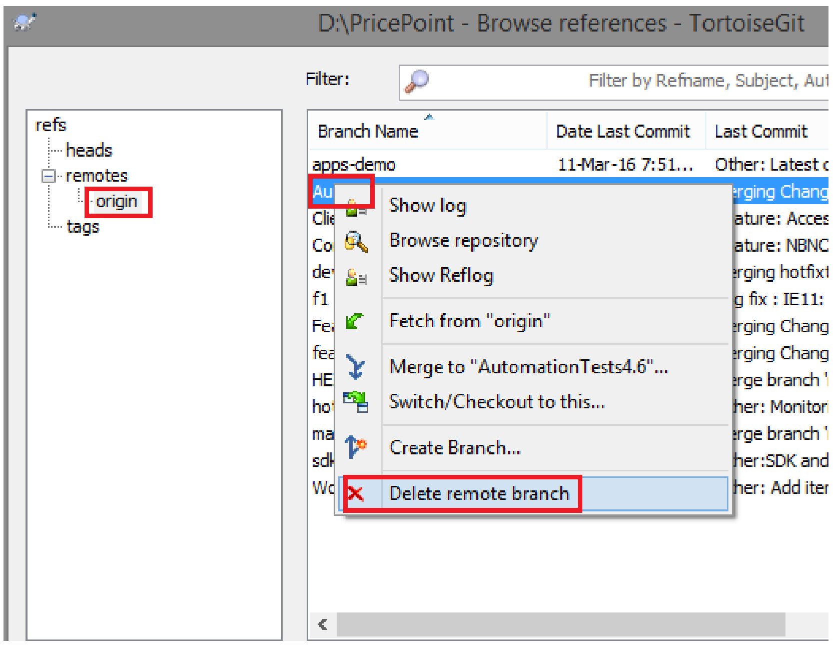


Figure 18: Delete remote branch with Tortoise Git from Windows explorer

3.4.2 Delete branch with Git Bush

Deleting branches can be performed with Git Bash command line. Follow the instructions bellow

1. Delete local branch with name branchName:

```
git branch -d branchName
```

2. Delete remote branch (branch from remote repository) with name branchName:

```
git push origin --delete branchName
```

4 Type of branches in NPP

4.1 Feature branch

Feature branches are used to develop one or more new features for the upcoming or a distant future release.

Every feature branch must be created from **develop** branch. Before creating branch, local **develop** branch should be up to date.

Recommended naming convention for the feature branches is: “feature-*“. Instead of *(star) sign put some information for the feature. For example: feature-NPPF-2323 or feature-FullImport.

Instructions for creating new branch can be found in this section.

When the new branch is created additional configuration is required. <connectionStrings> node in web.config file (Host and ProviderBasedSTS projects) and app.config file (Scheduler) should be changed to point to the database created for this feature.

When the feature is finished, the branch which was created, must to be merged back to the **develop** branch. Before merging must be run automation tests and unit tests. Instruction for merging can be found in this section.

After merging, if this branch is not going to be used any more should be deleted from the remote server. Instruction can be found in this section.

Follow the instructions bellow how to make feature branch with name “FeatureBranch” from “develop” branch:

1. Checkout on develop branch
```
git checkout develop
```
2. Make sure that local version of develop branch is up to date with remote version
```
git status
git pull
```
3. Create feature branch from develop branch
```
git branch FeatureBranch
```
4. Checkout on feature branch
```
git checkout FeatureBranch
```
5. Push the newly created branch on remote server
```
git push origin FeatureBranch
```

Or, new branch can be created also with these commands, “FeatureBranch” from “develop” branch:

```
git checkout -b FeatureBranch develop
git push origin FeatureBranch
```

4.2 Hotfix branch

Hotfix is a special branch that should derived from the tagged version on master branch. If some issue occurred on specific version on client environments and the issue has to be resolved on that version, then hotfix branch should be created from that tagged version on master branch. In the hotfix branch the issue has to be resolved, tested and afterwards has to be merged back to master branch with newly tagged version, and also has to be merged to develop branch.

4.2.1 Hotfix made by Git Bash

Follow the instructions bellow how to make hotfix from some tagged version on master branch

1. Checkout on master branch
   `$ git checkout master`
2. Pull all tags from master branch
   `$ git pull --tags`
3. Checkout the tagged version
   `$ git checkout tag-1.0.0.0`
4. Make hotfix branch from the tagged version.
   `$ git branch hotfix-1.0.0.1`
5. Checkout the hotfix version
   `$ git checkout hotfix-1.0.0.1`
6. Push the hotfix on remote server and afterward check if exists on Stash (http://10.40.1.18:7990/projects/NPPF/repos/pricepoint/branches)
   `$ git push origin hotfix-1.0.0.1`
7. Make changes on hotfix
8. Change VersionInfo file in the hotfix accordingly. If hotfix is derived from tagged version 1.0,0.0, set the Version info of the hotfix as 1.0.0.1. For the naming convention for the hotfixes should be followed incremental rule only of the last digit.
9. Commit changes locally and then push on the remote server
10. Create pull request before merging hotfix on the master and develop branch (follow instructions for Pull Request)
11. Merge hotfix on master branch (follow the instructions for merging branches)
12. Create tag on master branch. As hotfix is merged on the master branch a new tag should be created on the master branch (follow instructions for creating tag)
13. Merge hotfix on the develop branch (follow the instructions for merging branches)
14. Delete hotfix - as hotfix is merged on the master and develop branch the one can be deleted (follow the instructions for deleting branch)

4.2.2 Hotfix made by TortoiseGit

Follow the instructions bellow how to make hotfix from some tagged version on master branch using TortoiseGit from Windows Explorer

1. Locate the folder where the project is checked out.

2. Create hotfix from the specific tagged version

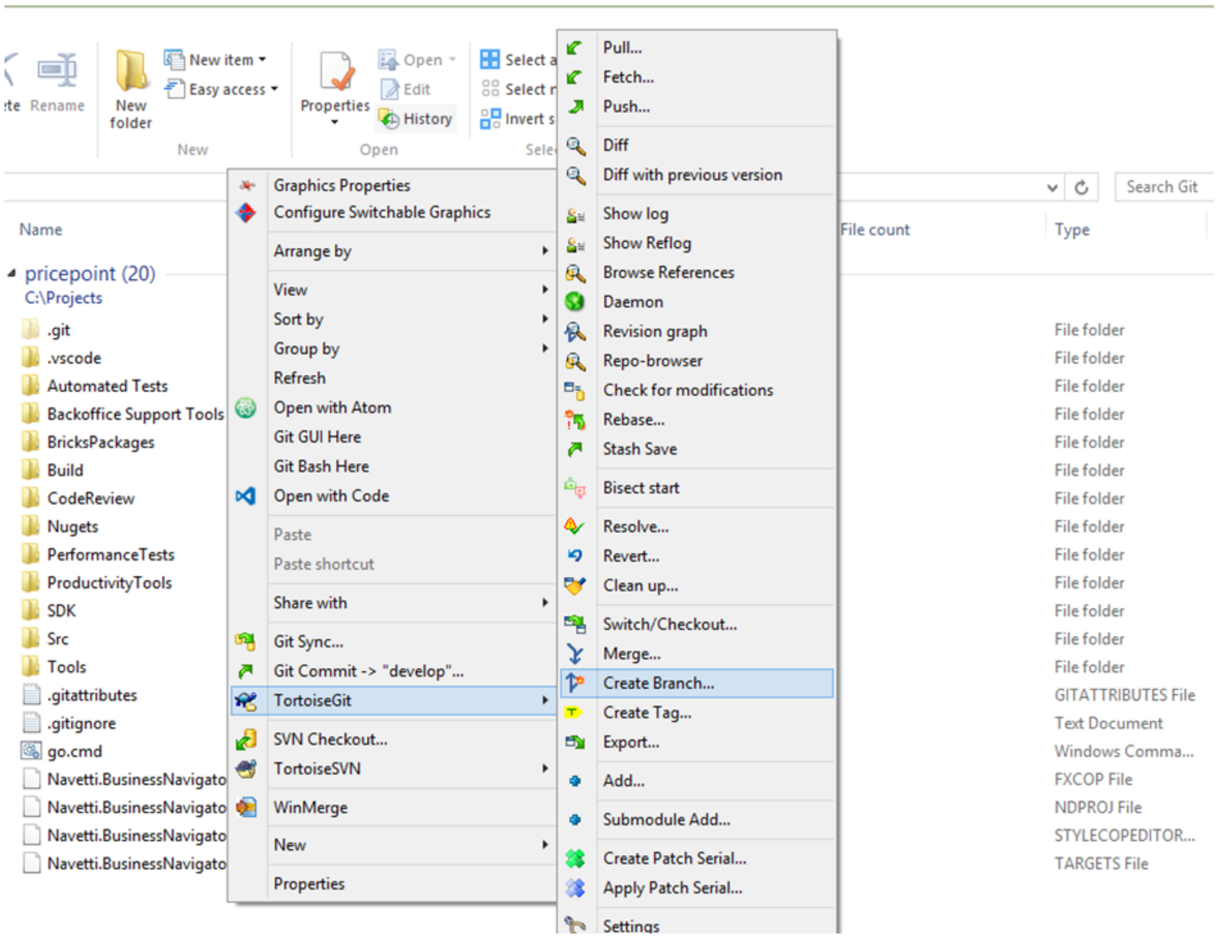
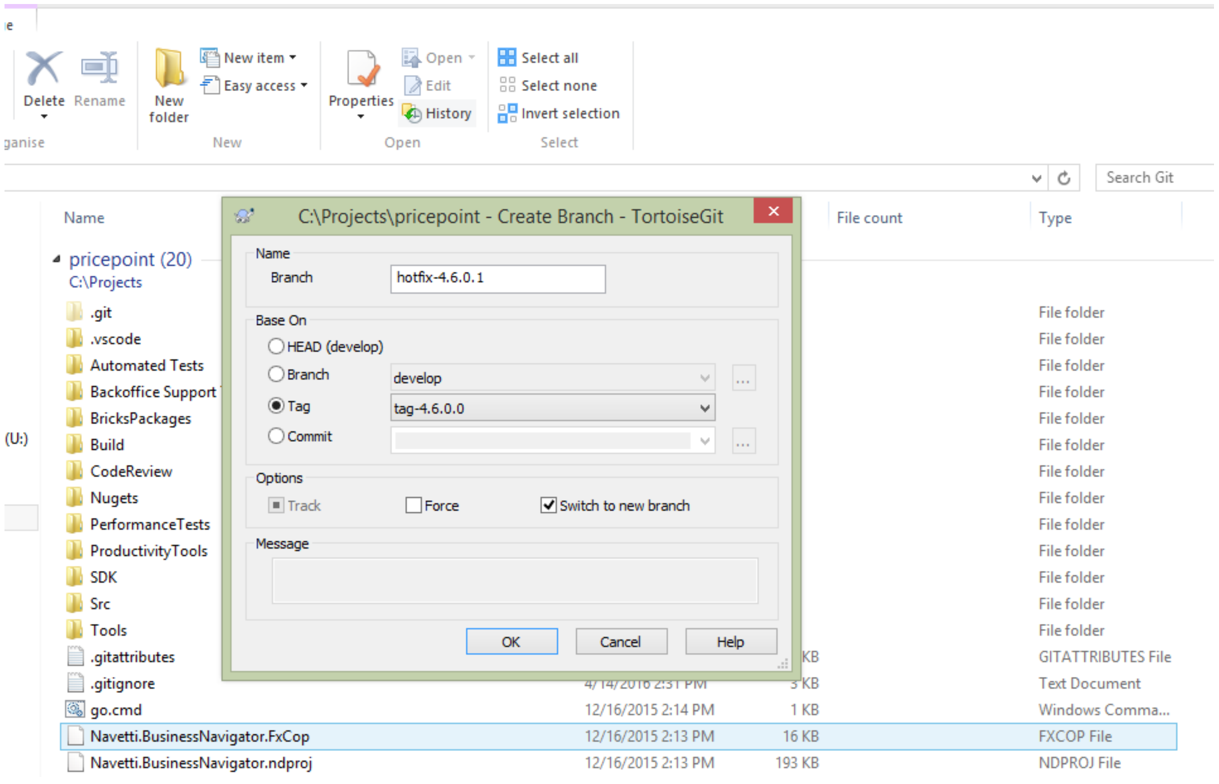


Figure 19: Create hotfix branch from master branch



Name
Branch: hotfix-4.6.0.1

Base On
HEAD (develop)
Branch: develop
[✓ Tag]: tag-4.6.0.0
Commit

Options
Track
Force
[✓ Switch to new branch]

Message

Figure 20: Set options for creating hotfix branch from master branch

3. Push the hotfix on remote server and check if exists on Stash http://10.40.1.18:7990/projects/NPPF/repos/pricepoint/branches

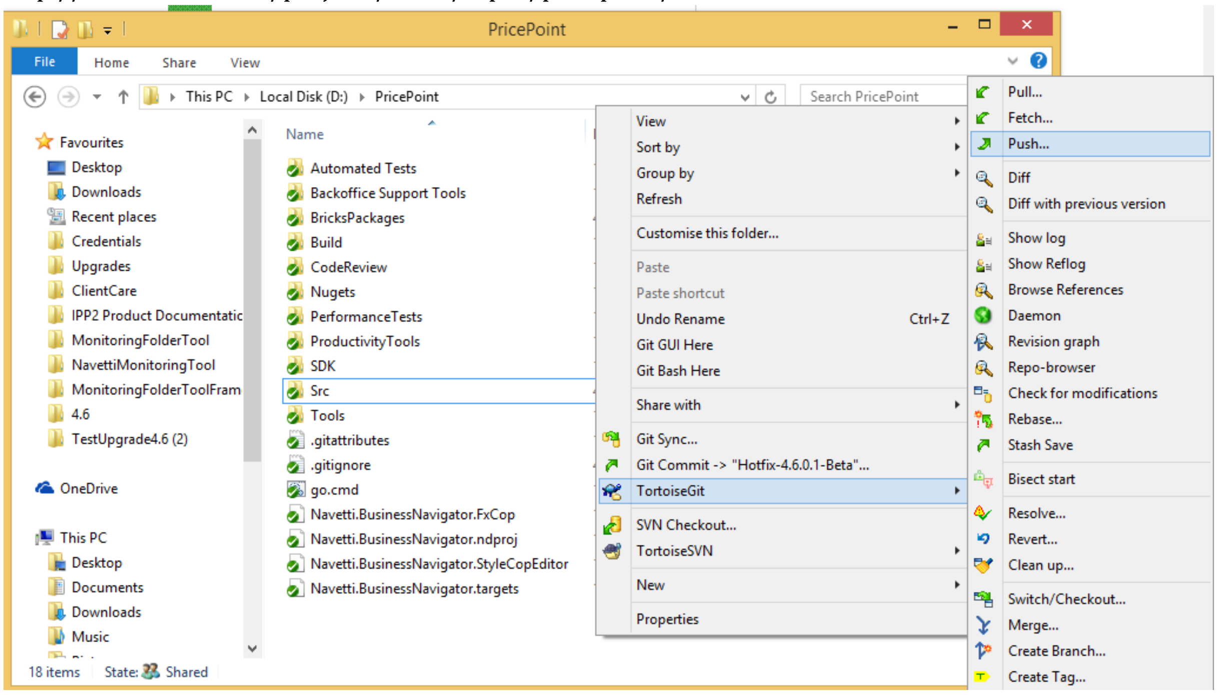


Figure 21: Push hotfix branch on remote server

4. Make changes on hotfix
5. Change VersionInfo file in the hotfix accordingly. If the hotfix is derived from tagged version 1.0.0.0, set the Version info of the hotfix as 1.0.0.1. For the naming convention for the hotfixes should be followed incremental rule only of the last digit.
6. Commit changes locally and then push on the remote server
7. Create pull request before merging hotfix on the master and develop branch (follow instructions for Pull Request)
8. Merge hotfix on master branch (follow the instructions for merging branches with GitTortoise)
9. Create tag on master branch - as hotfix is merged on the master branch a new tag should be created on the master branch (follow instructions for creating tag with GitTortoise)
10. Merge hotfix on the develop branch (follow the instructions for merging branches with GitTortoise)
11. Delete hotfix - as hotfix is merged on the master and develop branch the one can be deleted (follow the instructions for deleting branch with GitTortoise)

4.3 Release Branch

Release branches support preparation of a new production release. Furthermore, they allow for minor bug fixes and preparing meta-data for a release (version number, etc.). By doing all of this work on a release branch, the **develop** branch is cleared to receive features for the next big release.

The key moment to branch off a new release branch from **develop** is when **develop** (almost) reflects the desired state of the new release. All features, which are targeted for the release, which are developed on separate feature branches must be merged in to **develop** at this point in time. All features targeted at future releases may not—they must wait until after the release branch is branched off.

Follow these instructions for creating release branch:

1. Every release branch must be created from **develop** branch. Before creating branch, local **develop** branch should be up to date.
2. Recommended naming convention for the feature branches is: “release-*“, instead of star (*) sign should be stated the version number of the upcoming release. For example: release-4.6.0.0.

3. Instructions for creating new branch can be found in this section.
4. VersionInfo file on the newly created release branch should be updated with the corresponding version.
5. All found bugs on this branch, should be fixed on it.

When the state of the release branch is ready to become a real release, some actions need to be carried out:

1. The release branch should be merged into **master** branch. (Instruction for merging can be found in this section)
2. Tag should be created after merging into master. (Instruction for merging can be found in this section)
3. The release branch should also be merged into **develop** branch.
4. When all steps above are finished, the release branch should be deleted, from local and remote repository. (Instruction for deleting can be found in this section)

4.4 Tagging

Tagging is intended to mark the master branch with concrete version at that point. Master branch is marking with tag when release branch or hotfix branch is merging to master branch. Recommended naming convention for tags is: “tag-*“, instead of star (*) sign should be stated the version of the release or hotfix. For example: tag-4.6.0.0.

4.4.1 Create Tag by TortoiseGit

Please follow the instructions how to create tag with Tortoise Git:

1. Locate the folder where the project is checked out.
2. Switch to **master** branch (follow instructions in this section)
3. Make sure that the local **master** branch is up to date (**TortoiseGit -> Pull…)**

4. Right click with the mouse and choose **TortoiseGit -> Create Tag…**

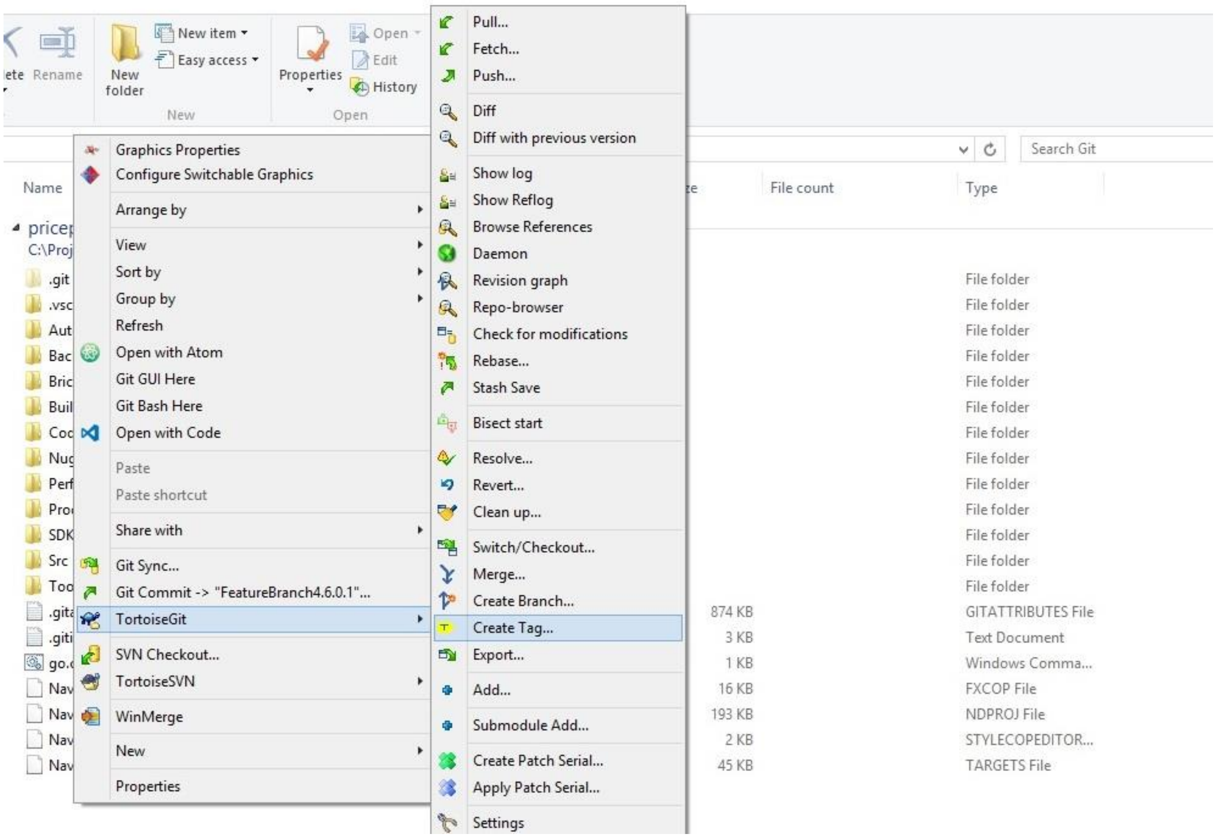
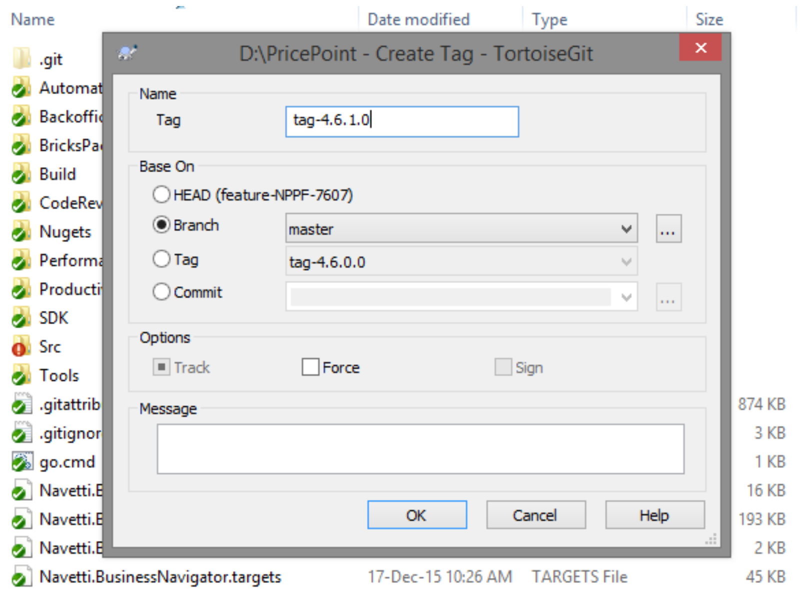


Figure 22: Create tag with Tortoise Git from Windows Explorer

5. Set the name of the new tag in the **Tag** field.
6. From the **Base On** section, from the **Branch** dropdown, select the **master** branch.
7. Enter message in the **Message** field.



D:\PricePoint - Create Tag - TortoiseGit

Name
Tag: tag-4.6.1.0

Base On
HEAD (feature-NPPF-7607)
[✓ Branch]: master
Tag: tag-4.6.0.0
Commit

Options
Track
Force
Sign

Message

OK | Cancel | Help

Figure 23: Create tag with Tortoise Git from Windows Explorer

With these steps the new tag is created in the local repository. The tag should also exist in the remote repository:

8. Switch to the newly created tag (follow instructions from this section)
9. Right click with the mouse and choose **TortoiseGit -> Push…**

4.4.2 Create Tag by Git Bush

New tag can be created with Git Bush command line. Commands and instruction used for creating new tags are listed below:

1. Switch to the master branch

   ```
   git checkout master
   ```

2. Get the latest version of master

   ```
   git pull origin master
   ```

3. Create tag with name "tag-version" and message "Create new tag". –a option is for annotate tag.

   ```
   git tag -a tag- version -m "Create new tag"
   ```

4. Push the new tag to the remote repository

   ```
   git push origin tag- version
   ```

4.4.3 Get Remote Tags

Below are given some commands for listing and getting tags:

1. List remote tags

   ```
   git ls-remote -tags
   ```

2. List tags in local repository

   ```
   git tag
   ```

Or

   ```
   git tag -l
   ```

3. Checkout all remote tags

   ```
   git pull --tags
   ```

4. Switch to some tag with name tagName

   ```
   git checkout tagName
   ```

5 Tracking Branches

Making a connection between local and remote branches is known as “tracking” branches.

Tracking is important because it will make your everyday work with branches a lot easier.

```
$ git branch -u origin/feature-NPPF-4732
```

Your local branch now has a "counterpart" on the remote server.

5.1 Why should you set up tracking connections?

Let's say your current local HEAD branch is named "dev". And let's also say that you have set it up to track the "dev" branch on the remote named "origin". This relationship is invaluable for two reasons:

1. Pushing and pulling becomes a lot easier. You can simply use the shorthand commands "git pull" and "git push" - instead of having to think about the exact parameters like in "git push origin dev". Even more importantly than being "easier", this also prevents you from making mistakes!

2. Git can now inform you about "unpushed" and "unpulled" commits. Let's make an example:
   a. if you have 2 commits only locally that you haven't pushed to the remote yet, your local branch is "2 commits ahead" of its remote counterpart branch.
   b. if, on the other hand, there are 4 commits on the remote branch that you haven't downloaded yet, then your local branch is "4 commits behind" its remote counterpart branch.

This information helps tremendously in staying up-to-date. Git tells you about this right in the output for "git status":

```
$ On branch dev # Your branch and 'origin/dev' have diverged,
$ and have 1 and 2 different commits each, respectively.
$ nothing to commit (working directory clean)
```

5.2 How do you track a remote branch?

There are three main scenarios for creating a tracking connection.

1. **When you're starting to work on an existing remote branch**
   Let's say one of your colleagues has already started and published a branch on your remote server. You now want to chime in and start working on that topic, too. In that scenario, simply use the --track flag with the "git checkout" command:
   ```
   $ git checkout --track origin/dev
   ```
   This creates a new local branch with the same name as the remote one - and directly establishes a tracking connection between the two.

2. **When you're publishing a local branch**
   Let's now look at the opposite scenario: you started a new local branch and now want to publish it on the remote for the first time:
   ```
   $ git push -u origin dev
   ```
   You can tell Git to track the newly created remote branch simply by using the -u flag with "git push".

3. **When you decide at a later point in time**
   In cases when you simply forgot, you can set (or change) a tracking relationship for your current HEAD branch at any time:
   ```
   $ git branch -u origin/dev
   ```

6 Document history

Table 1 Document history

Edition:	Date:	Created by:	Description:
PA1	2016-4-21	Meri Smikova, Valntina Popatanasovic	First version
PA2	2016-04-25	Bojan Gjorgjievski	Added Tracking Branches section
PA3	2016-04-26	Meri Smilkova	Change git merge command