**Основные понятия:**

**custom-ветка –** локальнаяветка в git, соответствующая модулю в SVN. Эта ветка из git синхронизируется только с помощью SVN. Информацию из неё не поступает в общий репозиторий git.

**сustom-версия** – файлы модуля, соответствующие SVN ( custom-ветке).

**master-ветка** – основная open-source ветка разработки модуля. Может отличаться от custom-ветки.

В каждом пункте есть вариант для командной строки git; далее указаны эквивалентные действия, которые можно проделать с помощью визуального клиента TortoiseGit или другими визуальными средствами.

**I. Подготовка**

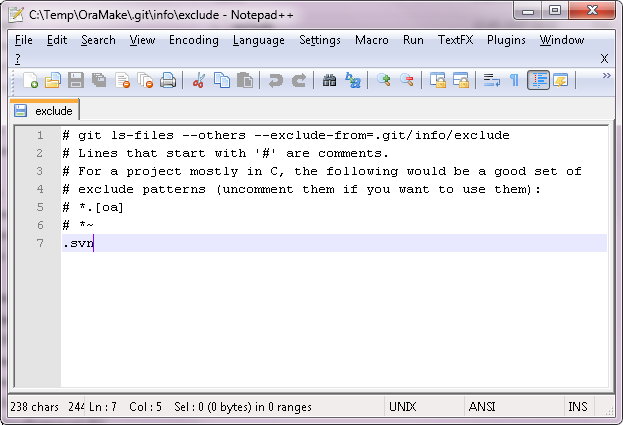
Добавляем игнорирование служебных каталогов Subversion:

echo '.svn' >> .git/info/exclude

либо меняем файл .git/info/exclude в текстовом редакторе

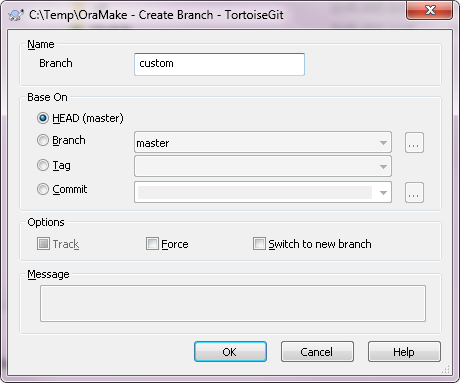
( этот файл можно также поменять позже с помощью меню Git -> Add to ignore list,

на папке .svn)

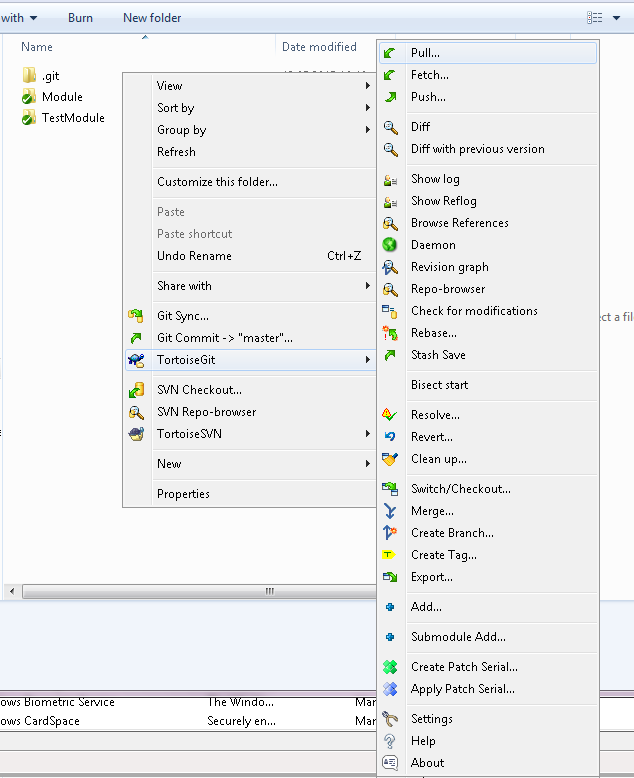


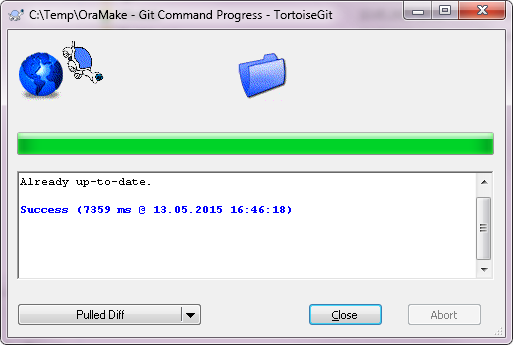
В git-репозитарии проекта создаем локальную ветку custom для custom-версий модулей.

git branch custom



На всех этапах в этом описании считаем что все изменения ветки master были взяты из SF.

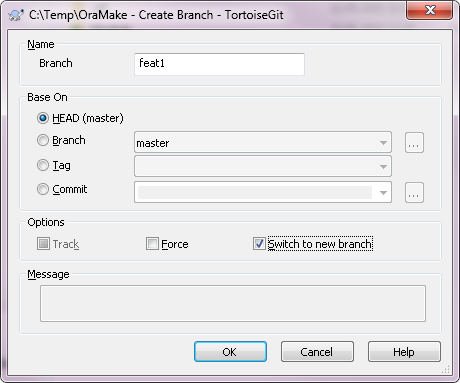




**II. Внесение изменений в основной проект с последующим слиянием их в custom-версию.**

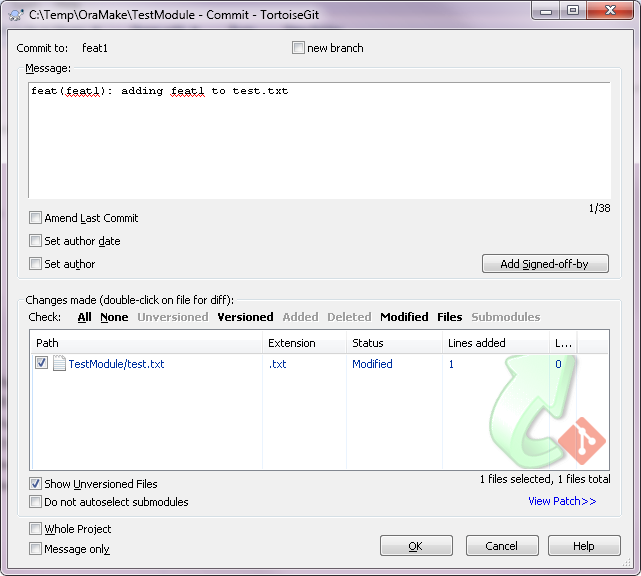
1. Создаем новую рабочую ветку и переходим на нее.

git checkout -b feat1 master

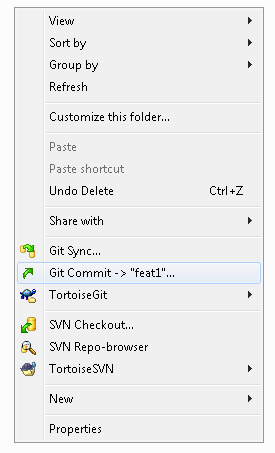


2. Вносим необходимые изменения

Вносим изменения, затем все сразу коммитим.

git add .

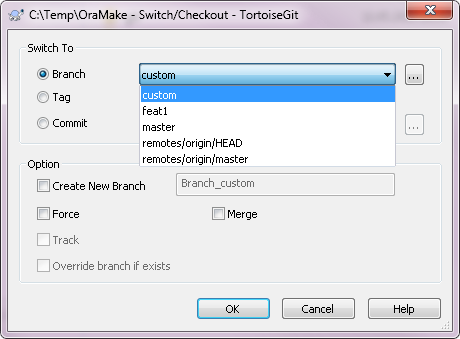
git commit -m '<commit message>'



3. Обновляем custom-версию модуля из SVN

Переключаемся на ветку custom

git checkout custom

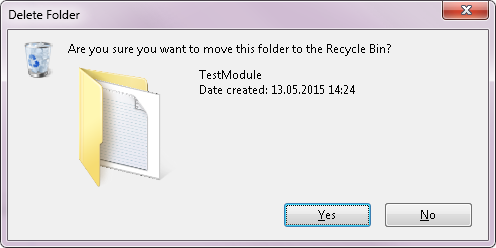


Далее 3a или 3б:

3a. Если это делается для модуля ( TestModule) в первый раз

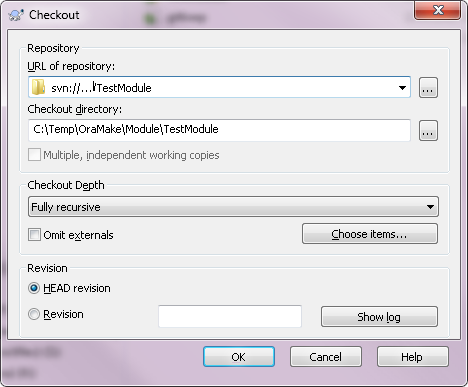
cd Module

rm -rf TestModule



Получаем текущие файлы модуля из Trunk в SVN.

svn co -q svn://.../TestModule/Trunk TestModule



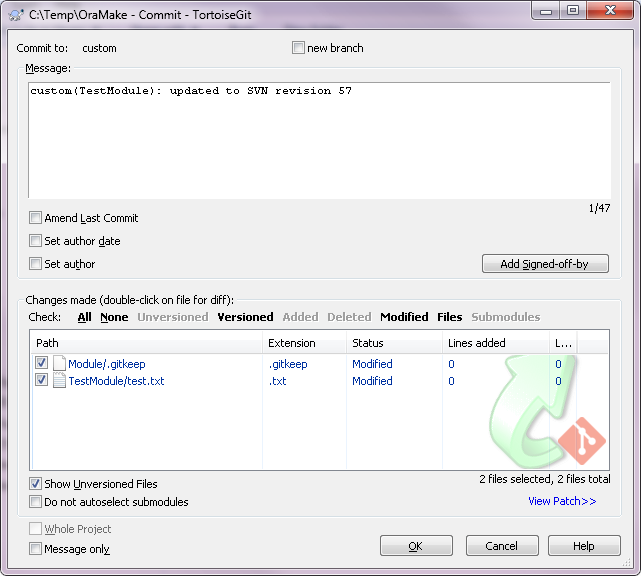
И комитим изменения в git.

cd ..

git add .

git commit -m 'custom(TestModule): updated to SVN revision rNNN'

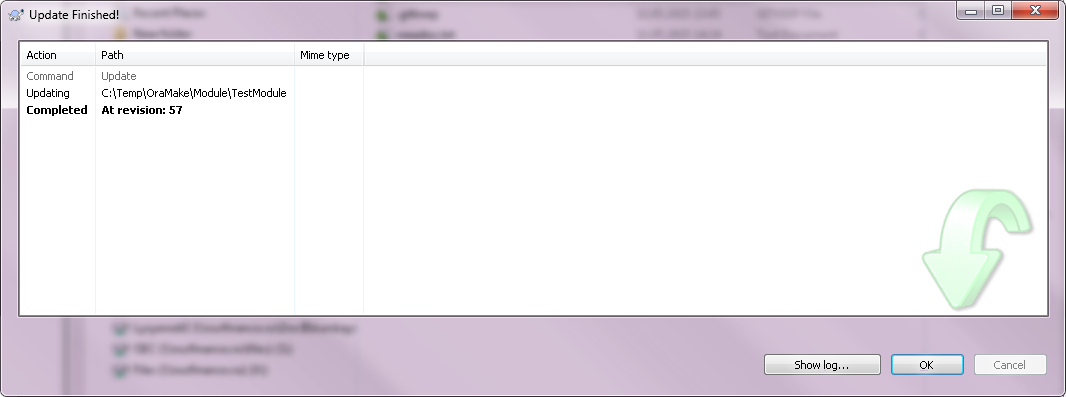
( вместо NNN указывается номер ревизии из SVN).



3б. Если это делается повторно

Обновляем файлы модуля из SVN.

svn update Module/TestModule



git status

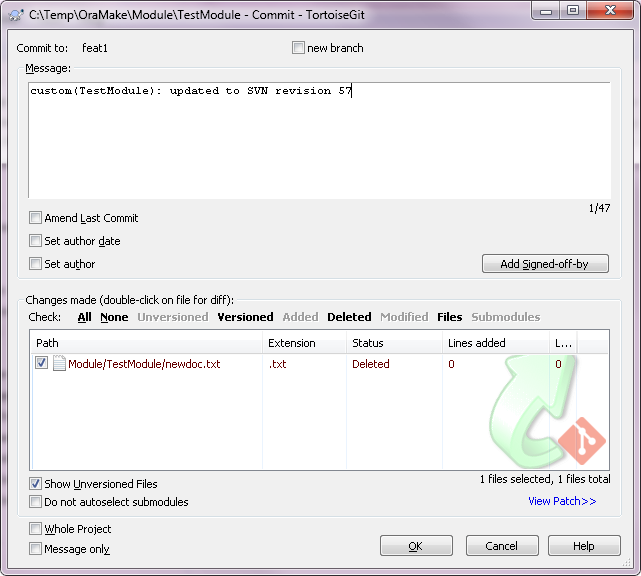
В случае наличия изменения нужно добавить изменения в индекс и

выполнить коммит в git:

git add .

git commit -m 'custom(TestModule): updated to SVN revision rNNN'

( вместо NNN указывается номер ревизии из SVN).

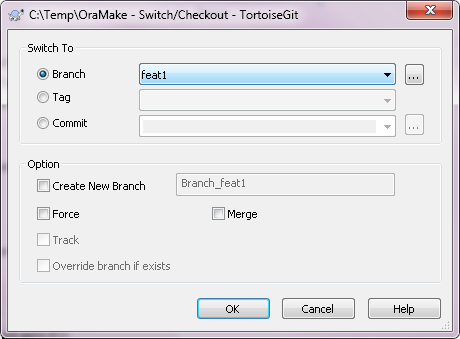


4. Переносим изменения из feat1 в custom

1) Обеспечиваем базирование feat1 на последнем коммите в master

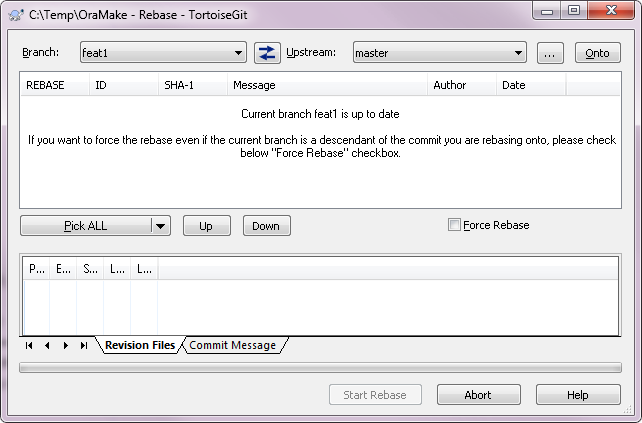
git checkout feat1

git rebase master



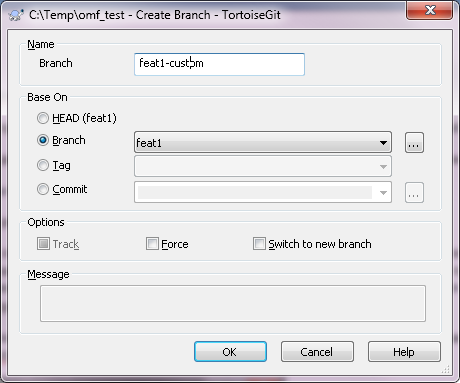
В данном обычном случае с момента пункта 1) с веткой master ничего не происходило, поэтому

feat1 итак базируется на master.



2) Создаем ветку-синоним для feat1. Это означает что после следующей команды ветки feat1 и feat1-custom будут указывать на тот же commit. Такой временный указатель требуется для следующей команды rebase, чтобы указать диапазон commit-ов, которые будут применяться к custom.

git branch feat1-custom

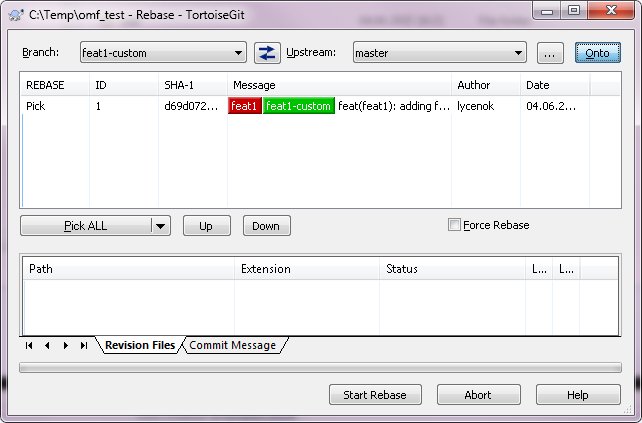


и переносим созданную ветку на custom.

git rebase --onto custom master feat1-custom

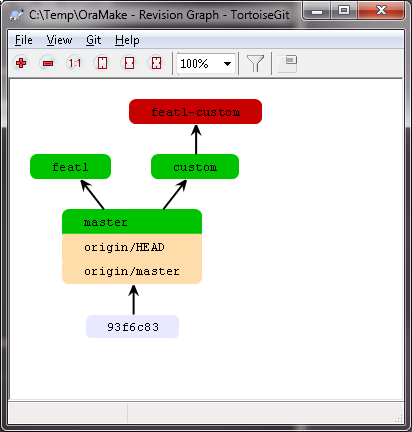
При этом на базе custom перевыполняются коммиты, сделанные на feat1-custom в диапазоне [ master; feat1-custom] и ветка feat1-custom начинает указывать на последний перевыполненный коммит

В Tortoise Git нужно выбрать ветку custom с помощью кнопки “Onto”.



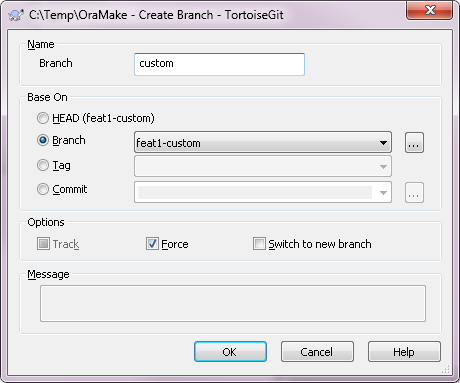
В результате находимся на ветке feat1-custom

После указанной команды должна быть схема “Revision Graph” ( вызывается из контекстного меню Tortoise Git), аналогичная изображённой на скриншоте.



git branch -f custom

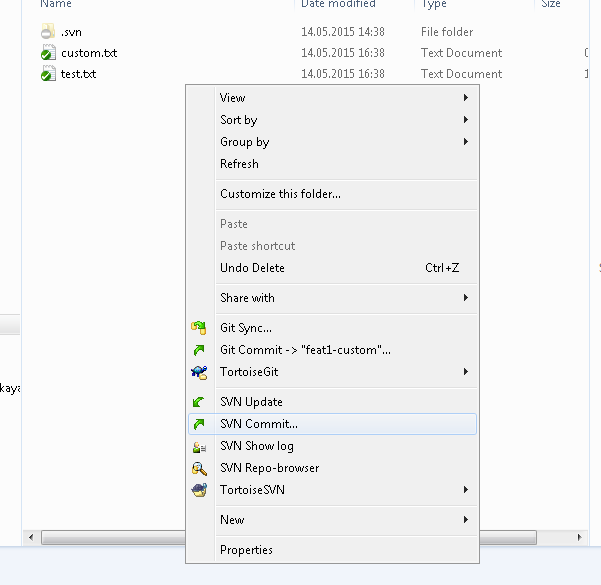
Переводим custom на последние изменения ( Create Branch, отмечаем checkbox “Force”).

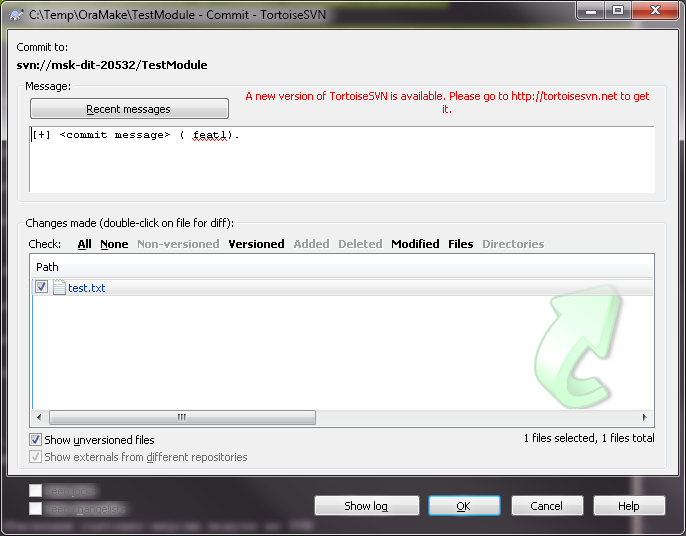


5. Сливаем изменения в SVN.

svn ci Module/TestModule -m '<commit message> ( feat1).'

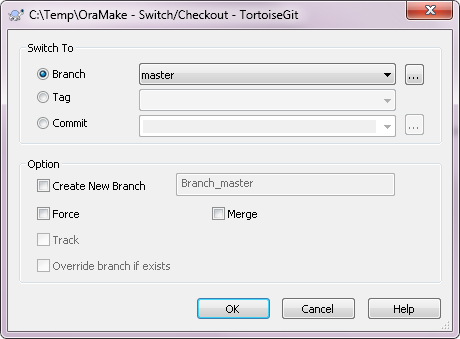
( в скобках можно указывать какое-то условное имя доработки, которое будет соответстовать имени доработки в комментариях git)



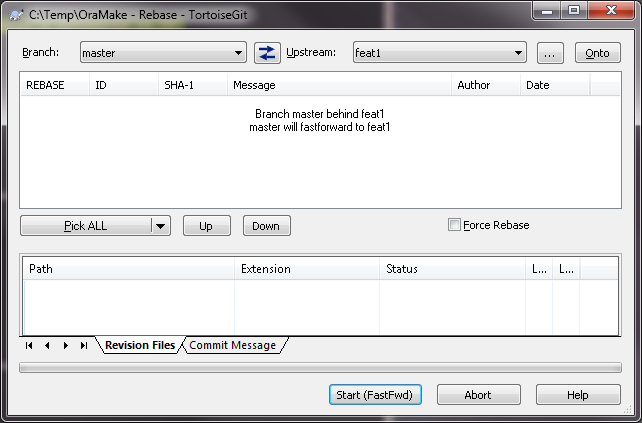


6. Сливаем изменения из feat1 в master и публикуем на SF

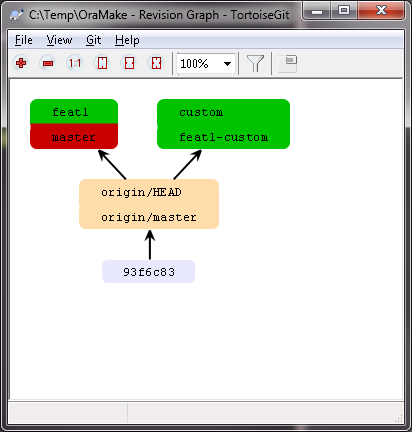
git checkout master



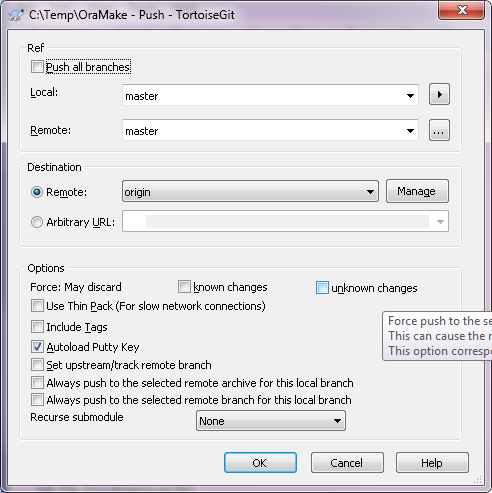
git rebase feat1



В “Revision Graph” будет примерно следующая картина:



Выполняем git push



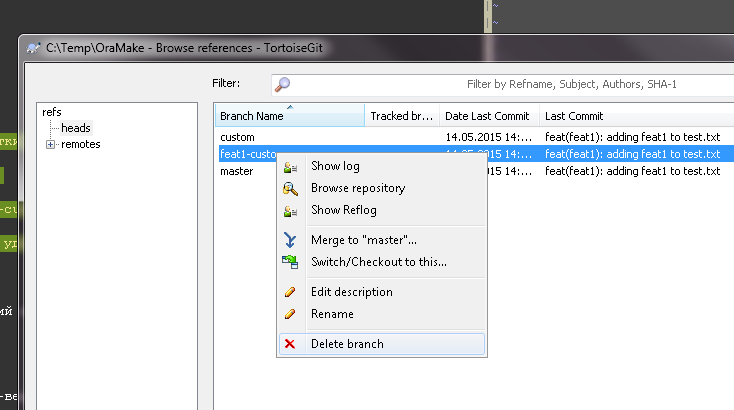
7. Удаляем рабочие ветки ( если они не нужны)

git branch -d feat1

git branch -D feat1-custom

( -D для безусловного удаления)

В TortoiseGit это можно сделать из списка ссылок ( Browse references)



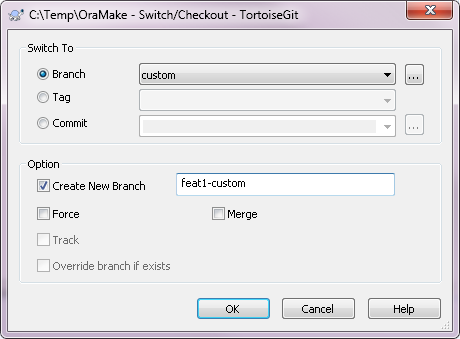
**III. Внесение изменений в custom-версию с последующим переносом их в основной проект.**

1. Обновляем custom-версию модуля из SVN

Согласно п.3 из раздела II ( см. выше).

2. Создаем ( на custom) новую рабочую ветку и переходим на нее.

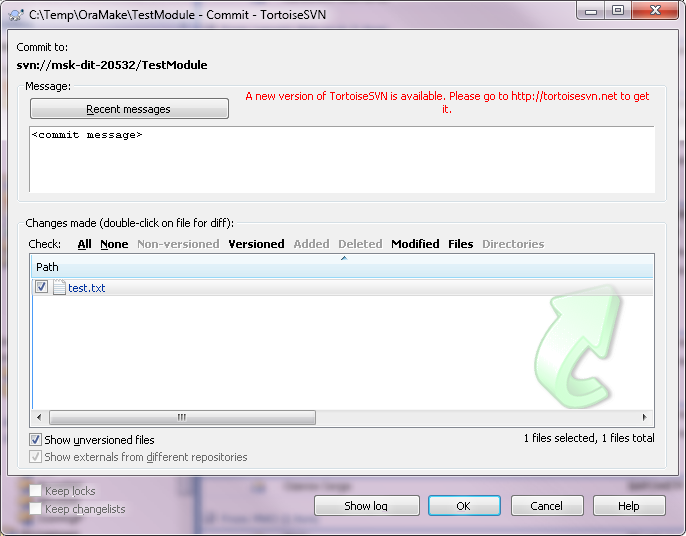
git checkout -b feat1-custom custom



3. Вносим изменения и коммитим их в SVN

Изменили файлы, затем закоммитили:

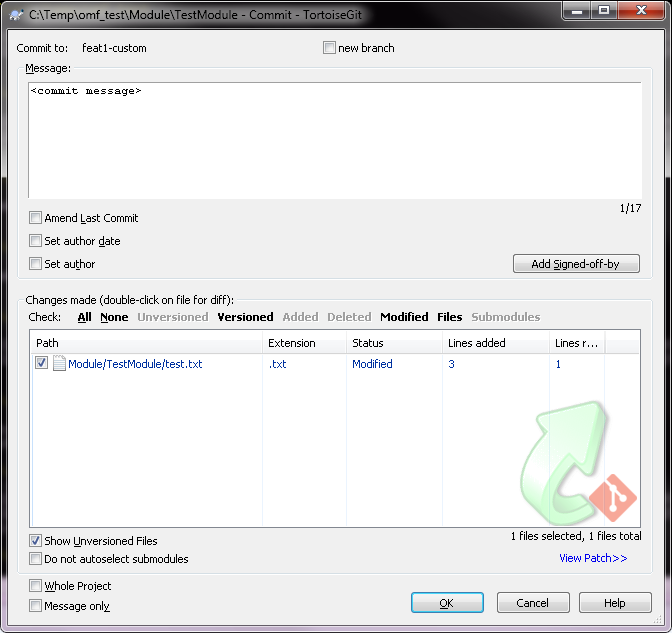
svn ci Module/TestModule -m '<commit message>'



4. Закоммитили изменения в git ( в feat1-custom)

git add .

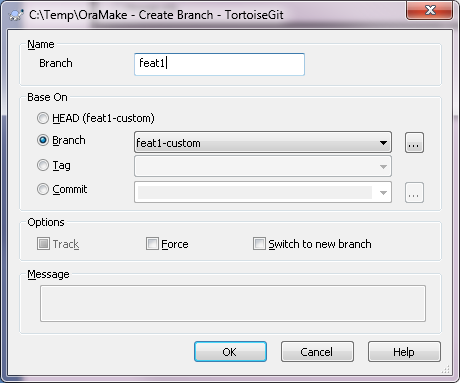
git commit -m '<commit message>'



5. Переносим изменения в master

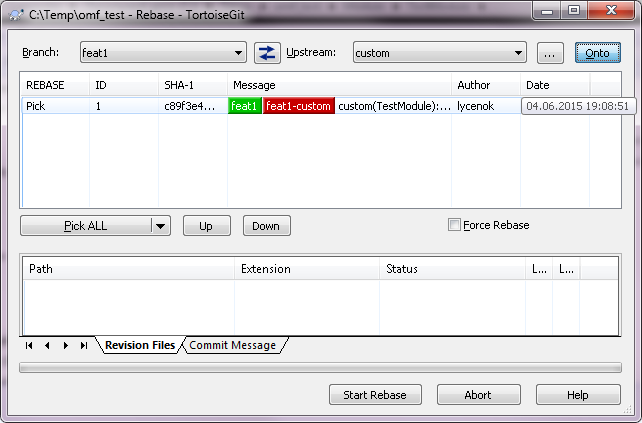
Создаем ветку-синоним для feat1 ( ветку, временно указывающую на тот же commit что и feat1-custom)

git branch feat1 feat1-custom



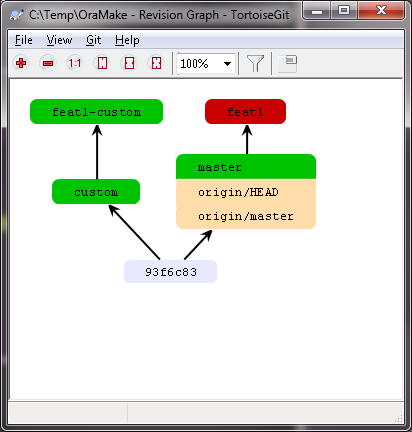
и переносим ее на master

git rebase --onto master custom feat1



В некотором смысле это симметрично пункту 4 раздела II. На master перевыполняются коммиты диапазона [ custom – feat1 ] и в итоге текущей становится ветка feat1.

После указанной команды должна быть схема “Revision Graph” ( вызывается из контекстного меню Tortoise Git), аналогичная изображённой на скриншоте.

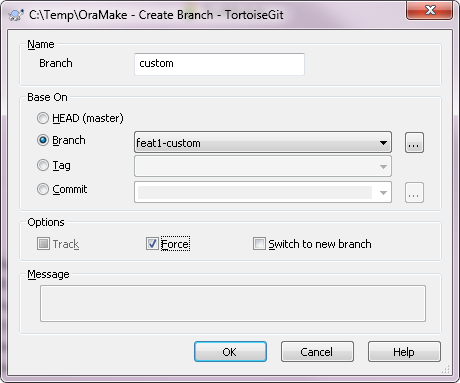


6. Сливаем изменения из feat1 в master и публикуем на SF

Согласно п.6 из раздела II ( см. выше).

7. Перемещаем custom на последние изменения

git branch -f custom feat1-custom



8. Удаляем рабочие ветки ( если они не нужны)

Согласно п.7 из раздела II ( см. выше).