**О чем эта статья**

Эта статья предназначена для облегчения работы с гитом и нашим локальным сервером.

Все операции с гитом можно грубо разделить на четыре категории:

* Операции, которые нужно выполнить вообще один раз - [установить git на свой комп, зарегистрироваться на сервере и т.д.](http://wiki.dep111.rtc.local/manuals:git#%D0%BE%D0%B4%D0%BD%D0%BE%D0%BA%D1%80%D0%B0%D1%82%D0%BD%D1%8B%D0%B5_%D0%B4%D0%B5%D0%B9%D1%81%D1%82%D0%B2%D0%B8%D1%8F).
* Операции, которые нужно выполнять один раз при создании нового проекта - [создать и настроить репозиторий на компе и на сервере.](http://wiki.dep111.rtc.local/manuals:git#%D1%81%D0%BE%D0%B7%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5_%D0%BD%D0%BE%D0%B2%D0%BE%D0%B3%D0%BE_%D1%80%D0%B5%D0%BF%D0%BE%D0%B7%D0%B8%D1%82%D0%BE%D1%80%D0%B8%D1%8F)
* Операции, которые нужно выполнять регулярно - [коммит и пуш](http://wiki.dep111.rtc.local/manuals:git#%D1%87%D1%82%D0%BE_%D0%BD%D0%B0%D0%B4%D0%BE_%D0%B4%D0%B5%D0%BB%D0%B0%D1%82%D1%8C_%D1%80%D0%B5%D0%B3%D1%83%D0%BB%D1%8F%D1%80%D0%BD%D0%BE).
* “Извращения” - ветвление истории, изменения прошлого, вызов сатаны. Эту категорию данный мануал не покрывает.

Если вам что-то не нравится, что-то неудобно, что-то не гудит или не свистит, не бойтесь гуглить или спрашивать у администраторов гита.

**Зачем вообще нужен GIT?**

Возможно вы хотите спросить: “Но зачем мне этот ваш гит? Мне и так норм, у меня есть гугель-драйв/дропбокс/резервные копии в архивах/божественная защита!”. Гит нужен по нескольким причинам:

* Для облегчения совместной работы над одним проектом. Гит надежнее нашего локального обмена, т.к. в нем сохраняются все версии файлов.
* Если вас собьет троллейбус, разобраться в ваших резервных копиях другим людям будет **значительно** тяжелее, чем вам сейчас. Гит - стандартизированный способ хранения.
* Вполне вероятно, что через полгода и вам будет тяжело вспомнить, где что лежит и зачем.

**Логика GIT**

В кратце суть такова. Вы создаете папку (или берете существующую) у себя на компе и делаете ее “репозиторием”. Это означает, что гит будет отслеживать изменения файлов в ней. Гитом можно пользоваться и без сервера, при этом история изменений сохраняется только локально. Когда вы делаете папку репозиторием, в ней создается скрытая папка .git, внутри которой сохраняются все локальные версии. Это означает, что:

* если папку .git удалить, то вся локальная история пропадет
* если вы хотите поработать дома, вы можете спокойно скопировать репозиторий на флешку и вся локальная история будет у вас с собой.

В отличие от дропбокса и тому подобных средств синхронизации **гит никогда ничего не делает сам**. Если вы хотите сохранить текущую версию файлов, вам нужно сделать *коммит* (commit). Коммит сохраняет версию локально.

Если вы хотите сохранить версию на сервере, то вам нужно создать репозиторий на сервере (один раз), привязать репозиторий на компе к серверу (один раз) и потом сделать *пуш* (push). Пуш отправляет всю локальную историю на сервер; т.е. можно сделать много коммитов и всего один пуш.

Если несколько человек редактирует один репозиторий на сервере, то чтобы забрать изменения, которые сделал другой человек, вам нужно сделать *пулл* (pull).

Если вы хотите скопировать с сервера чужой репозиторий, вам нужно сделать *клон* (clone).

Желательно, чтобы один ваш репозиторий соответствовал одной “единице работы”. Что является единицей работы - вопрос дискуссионный, но **крайне** желательно не сваливать в один репозиторий платы/код/что-угодно, которые никак друг с другом не связаны. Пусть это будут файлы, хотя бы относящиеся к одному проекту.

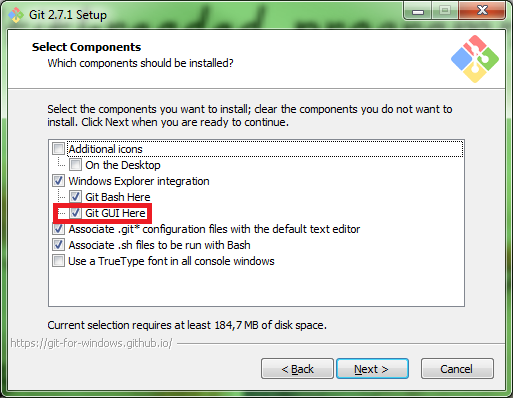
В идеале - одна плата - один проект для IDE - один  репозиторий.

**Однократные действия**

**Установка Гита**

Для гита существует множество клиентов, которые делают одни и те же вещи, но интерфейс у них разный. Данный мануал покрывает только один клиент - tortoise git. Если вы хотите использовать другой клиент, то вам с ним и разбираться.

Сначала нужно установить консольное приложение git:

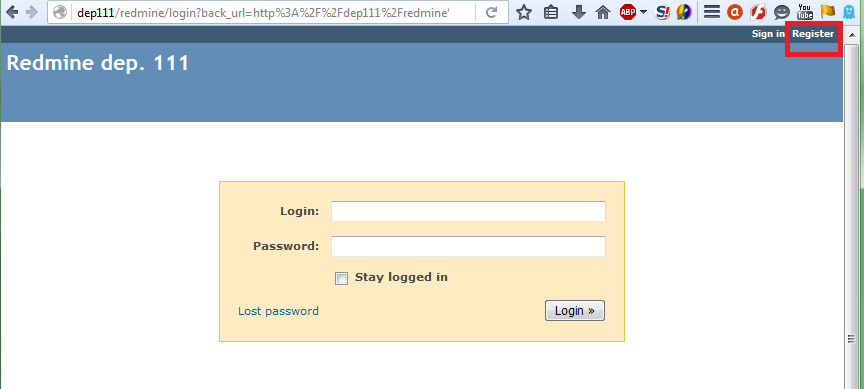
1. Скачайте и запустите установщик <https://git-scm.com/download/win>
2. Нажимайте далее-далее-далее-далее пока не дойдете до вот этого окна. **Снимите** галку “Git GUI here”
3. 
4. Нажимайте далее-далее-далее до окончания установки

Теперь нужно установить tortoise git - клиент для гита, который интегрируется в проводник.

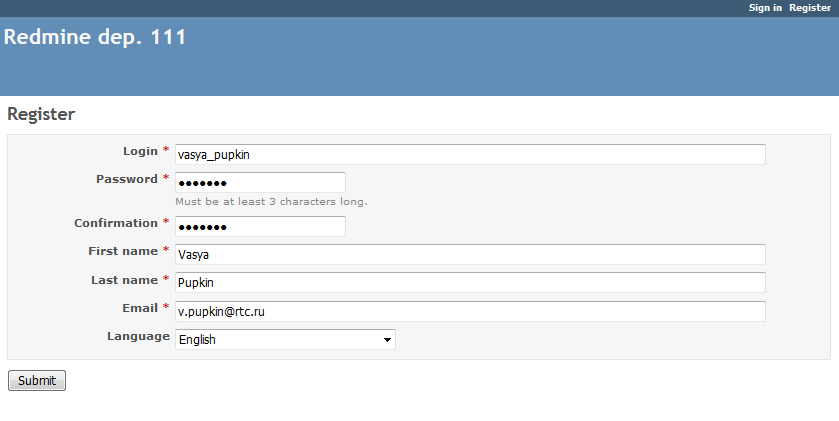
1. Скачайте и запустите установщик <https://tortoisegit.org/download/>
2. Бездумно нажимайте далее до окончания установки

**Регистрация на сервере**

Все серверные игрища нашего отдела находятся по адресу [http://dep111](http://dep111/). Авторизация в сервер гита происходит через redmine, поэтому (как это ни странно), зарегистрироваться нужно в redmine. Для этого, зайдите в <http://dep111/redmine/> и нажмите **Register**.



Заполните все поля по образцу. Логин **обязательно** должен быть в виде имя\_фамилия транслитом. Имя и фамилия должны тоже быть транслитом.

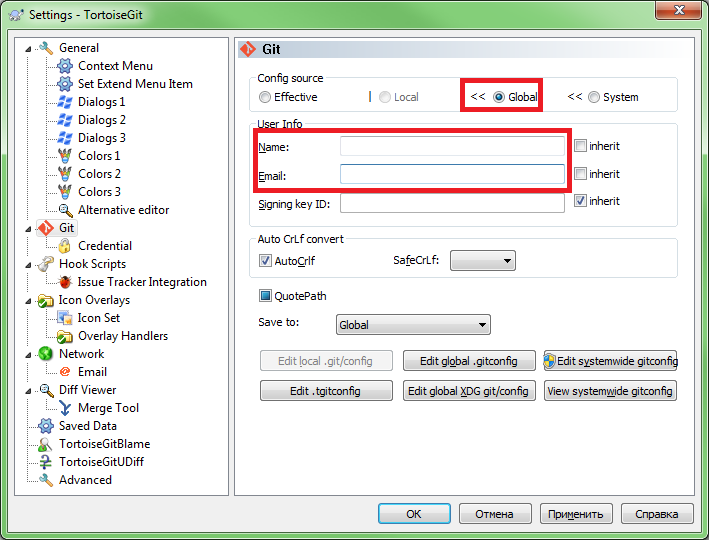


Нажмите Submit и дождитесь подтверждения вашего аккаунта администратором.

После подтверждения зайдите под тем же логином и паролем на сервер гита - [http://dep111:9090](http://dep111:9090/)

**Сохранение логина и почты на компе**

Чтобы делать коммиты и пуши вам необходимо внести свой логин для сервера и почту в tortoisegit. Зачем почту? А хрен его знает. Но когда великий гит желает, мы не спрашиваем, мы повинуемся. Почту пишем на домене rtc.ru. Кликаем в папке правой кнопкой, выбираем tortoisegit → settings → Git

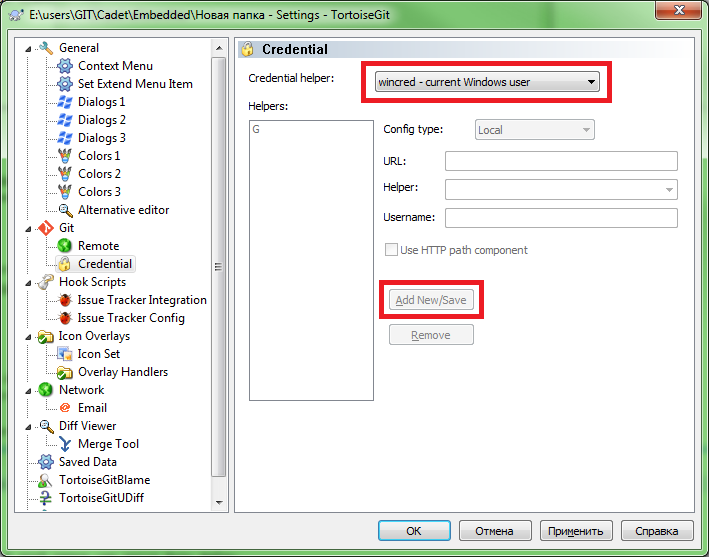


Убеждаемся, что выбрано Global. Заполняем только поле Name (логин для сервера) и Email, больше никаких галочек не снимаем и не ставим. Скриншот может отличаться от того, что видите вы! Почту пишем на домене rtc.ru.

**Сохранение пароля**

Чтобы при каждом пуше гит не спрашивал ваш логин и пароль для сервера, их можно сохранить. Для этого в любой папке кликните правой кнопкой мыши по пустому месту и выберите tortoise git → settings → credential

В выпадающем списке выберите wincred - current windows user. Add new/Save должна быть неактивна.



При следующем пуше гит спросит пароль в первый и последний раз (если вы его правильно введете, конечно).

**Дополнительные манипуляции**

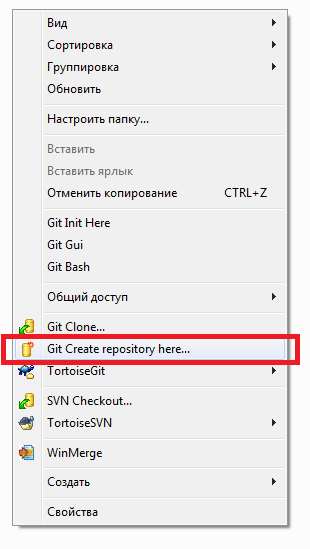
Чтобы мердж-коммиты были чуть более осмысленными, выполните вот это:

git config –global merge.log yes

Это заставит команду git merge по-умолчанию принимать опцию –log.

**Создание нового репозитория**

**Создание репозитория на компе**

1. Выберите папку, в которой хотите создать репозиторий. В этой папке уже могут быть файлы.
2. Откройте ее в проводнике и нажмите на пустом месте правую кнопку мыши и выберите пункт Git create repository here:
3. 
4. Появится вспылавающее окно. Не ставьте галок и просто нажмите ОК.
5. Появится еще одно всплывающее окно. Нажмите ОК.

После этого в папке появиться скрытая папка .git. Не удаляйте ее, в ней хранится вся локальная история. Не открывайте эту папку, если вам дорог ваш разум!

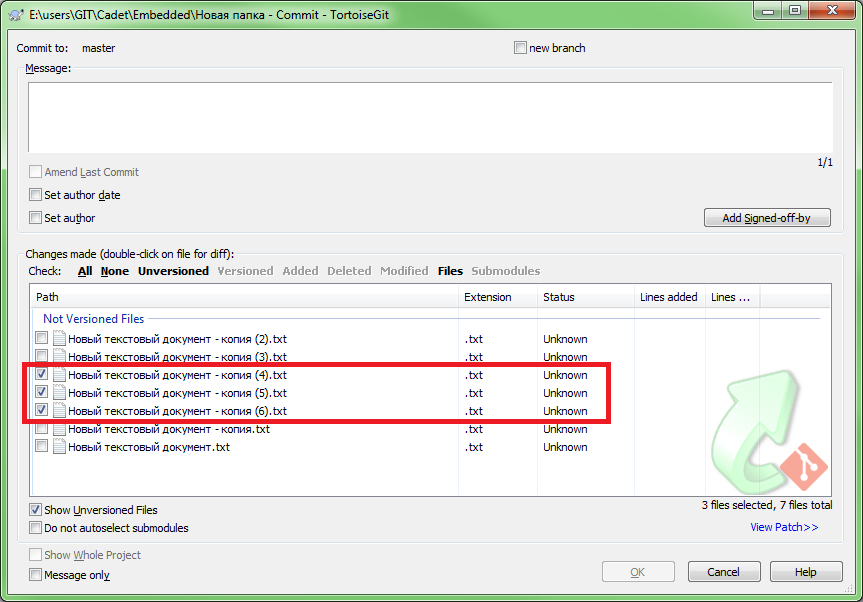
Теперь папка стала репозиторием.

**Добавление нужных файлов**

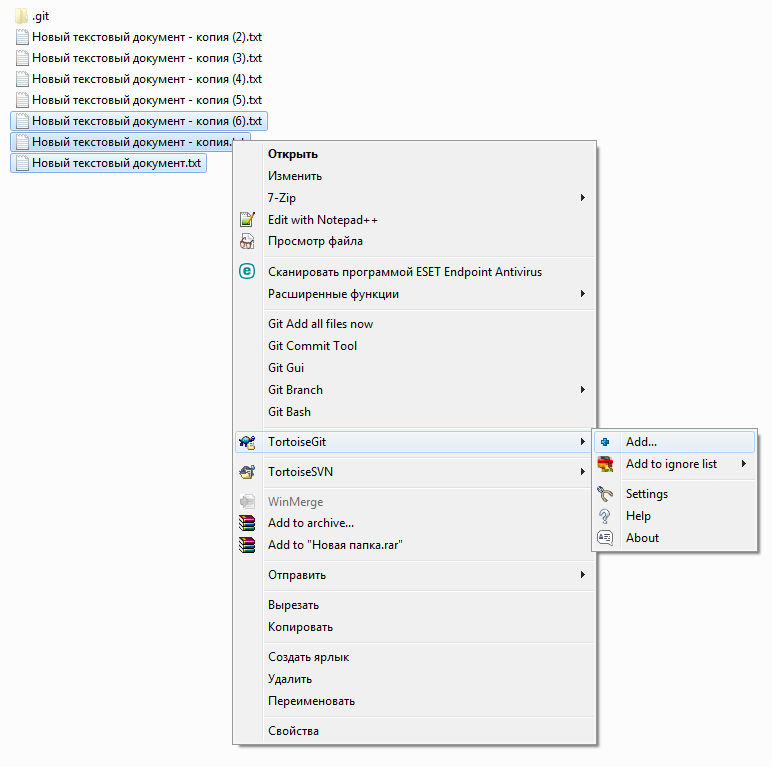
По-умолчанию, все файлы в свежесозданном репозитории гитом высокомерно игнорируются (unversioned, untracked, unknown..) и на них пририсовывается иконка со знаком вопроса. Чтобы гит начал отслеживать изменения в файле, этот файл нужно *добавить* (add).

Добавить файлы можно двумя способами:

1) При коммите отметить их галочками. Этот способ удобен, если у вас мало вложенных в репозиторий папок (потому что в окне коммита эти папки разворачиваются). В этом окне можно выделять группы файлов через шифт+клик или контрол+клик.



2) Если файлов и папок много, то их можно просто выделить в проводнике, кликнуть правой кнопкой мыши и в контекстном меню выбрать tortoise git → add. Если такого пункта в меню нет, то файл либо добавлен в игнор, либо [что-то пошло не так](http://wiki.dep111.rtc.local/manuals:git#%D1%87%D1%82%D0%BE_%D0%B4%D0%B5%D0%BB%D0%B0%D1%82%D1%8C_%D0%B5%D1%81%D0%BB%D0%B8_%D1%87%D1%82%D0%BE-%D1%82%D0%BE_%D0%BF%D0%BE%D1%88%D0%BB%D0%BE_%D0%BD%D0%B5_%D1%82%D0%B0%D0%BA).



После этого откроется диалоговое окно добавления файлов. В нем можно проконтролировать, что вы добавляете все файлы, которые хотите. Если вы случайно выделили лишний файл, галку с него можно снять.

Если вы случайно добавили лишний файл, то его можно “раздобавить” через контекстное меню tortoise git → revert.

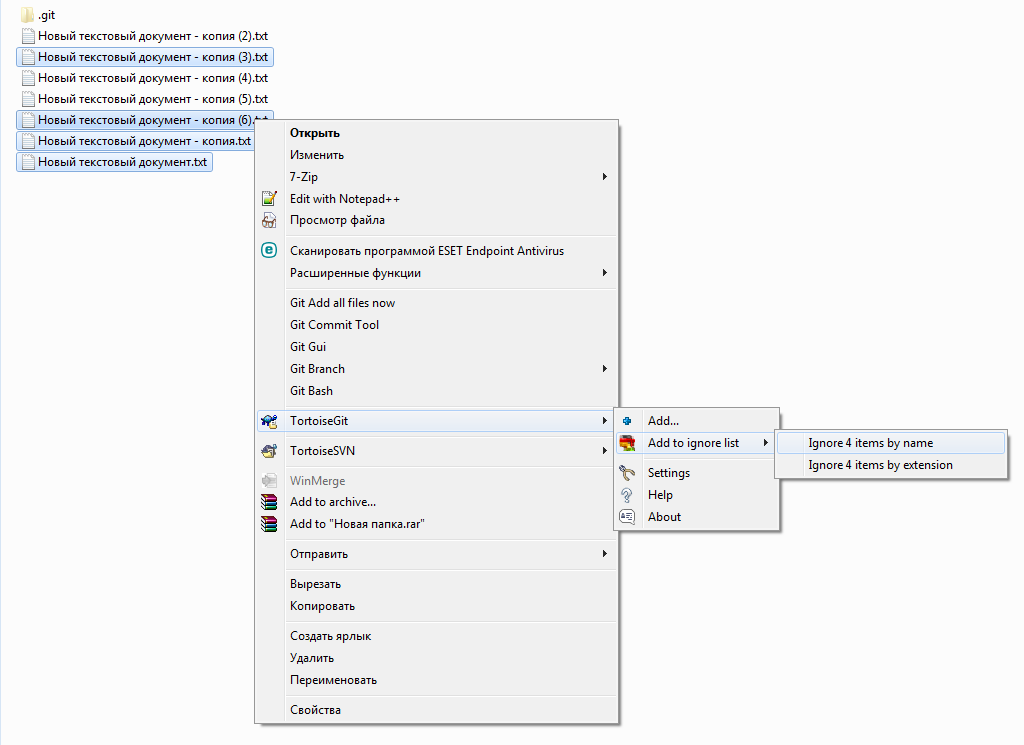
**Игнорирование ненужных файлов**

Как правило, в папке с проектом есть куча файлов, которые никому не нужны. Это могут быть файлы, которые генерирует альтиум или кейл, это могут быть просто какие-то ваши заметки, файлы .bak, бинарники и т.д.

Эти файлы *можно* хранить и версировать в гите, но как правило это совершенно ни к чему. Такие файлы любят меняться сами по себе и гит будет помечать их как измененные. Эти файлы часто много весят.

Разумеется, такие файлы можно просто не добавлять при коммите (т.е. не отмечать их галочкой), но тогда диалог коммита будет ими засоряться. А когда диалог засоряется, очень легко забыть добавить какой-нибудь важный файл.

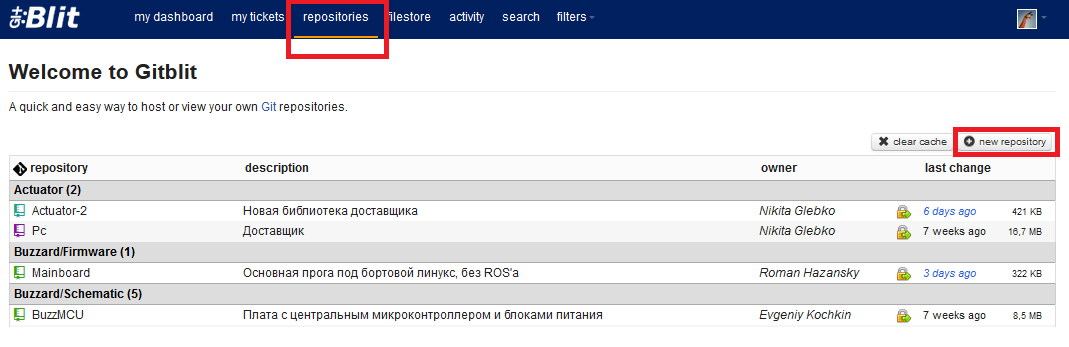
Более правильный выход - добавить ненужные файлы в игнор. Для этого выделите все ненужные вам файлы и папки, сделайте правый клик и в меню выберите пункт tortoise git → add to ignore → ignore items by name:



После этого в папке создаться текстовый файл .gitignore, в котором будет просто список имен. При следующем коммите файл .gitignore нужно добавить и закоммитить.

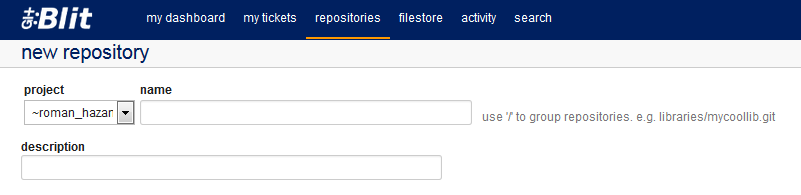
Если вы хотите игнорировать все файлы с определенным расширением, то вы можете сделать аналогичные действия, но выбрать ignore items by extension.

**Создание репозитория на сервере**

1. Зайдите на наш локальный сервер гита - <http://dep111:9090/>
2. Залогиньтесь
3. Перейдите на вкладку repositories и нажмите new repository:
4. 

Вы попадаете на страницу создания репозитория. На ней нужно выбрать проект, имя нового репозитория и заполнить поле description (описания проекта). Все остальные поля можно не трогать. В названии проекта можно использовать только латинцу, цифры и некоторые спецсимволы. **Пробелы использовать нельзя.** В поле description можно использовать кириллицу. Описание желательно заполнить, т.к. оно отображается в списке репозиториев и облегчает понимание, что же это за хреновину вы туда положили.

На странице создания репозитория вам нужно выбрать проект, к которому принадлежит репозиторий или создать новый проект. Проекты на сервере - это просто папки, но отображает он их, к сожалению, через анус.



* Чтобы создать новую папку, ее имя нужно вписать в поле **name** перед именем проекта, например *Kukan/mainboard* - создаст папку *Kukan* и в ней создаст репозиторий *mainboard*. Вложенные папки допустимы.
* Чтобы создать новый проект, в качестве текущего проекта нужно выбрать корень (т.е. /).
* Если проект (т.е. папка) уже существует, то его можно выбрать в выпадающем списке **project**.
* Если же в выпадающем списке можно выбрать только проект *Kukan/pukan*, а вы хотите создать репозиторий *mainboard*в папке *Kukan*, то вам придется написать *Kukan/mainboard* в поле name.

Да, мы знаем, что это идиотизм. Мы работаем над этим.

Если вы хотите создать какой-то свой личный репозиторий, не имеющий прямого отношения к проектам, то в выпадающем списке нужно выбрать пункт *~ваш\_логин* - это ваша приватная папка. Обычно этот пункт находится в самом конце списка или выбран по-умолчанию. Репозитории в вашей личной папке не отображаются в общем списке, но их все равно можно достать и склонировать.

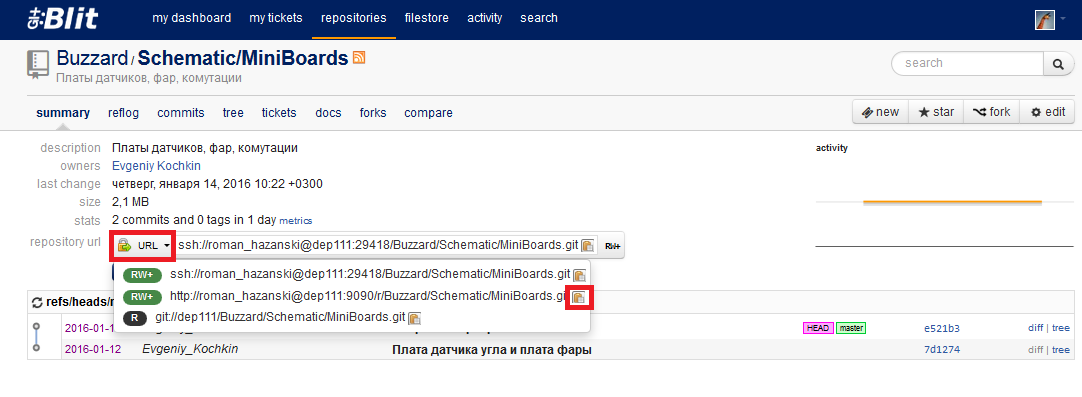
Если вы не можете выбрать никакой проект, кроме своего приватного (и он при этом засерен), если вы не видите кнопку new repository или гигантская птица рух не дает вам зайти на сервер, то вам необходимо [обратиться к администратору](http://wiki.dep111.rtc.local/manuals:git#%D1%87%D1%82%D0%BE_%D0%B4%D0%B5%D0%BB%D0%B0%D1%82%D1%8C_%D0%B5%D1%81%D0%BB%D0%B8_%D1%87%D1%82%D0%BE-%D1%82%D0%BE_%D0%BF%D0%BE%D1%88%D0%BB%D0%BE_%D0%BD%D0%B5_%D1%82%D0%B0%D0%BA)

**Привязка репозитория на сервере к локальному**

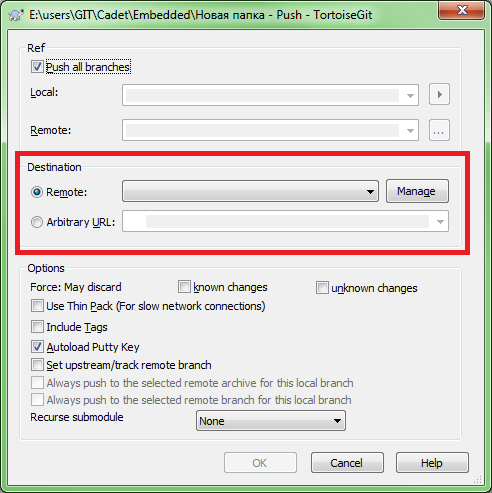
Если вы хотите сделать пуш (то бишь, отправить вашу историю куда-то) нужно сначала создать место, куда вы хотите ее отправить. Для этого смотрите пункт [cоздание репозитория на сервере](http://wiki.dep111.rtc.local/manuals:git#%D1%81%D0%BE%D0%B7%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5_%D1%80%D0%B5%D0%BF%D0%BE%D0%B7%D0%B8%D1%82%D0%BE%D1%80%D0%B8%D1%8F_%D0%BD%D0%B0_%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80%D0%B5).

Привязанный репозиторий на сервере называется *ремоут* (remote). Добавить новый ремоут можно либо прямо при пуше, либо в произвольный момент.

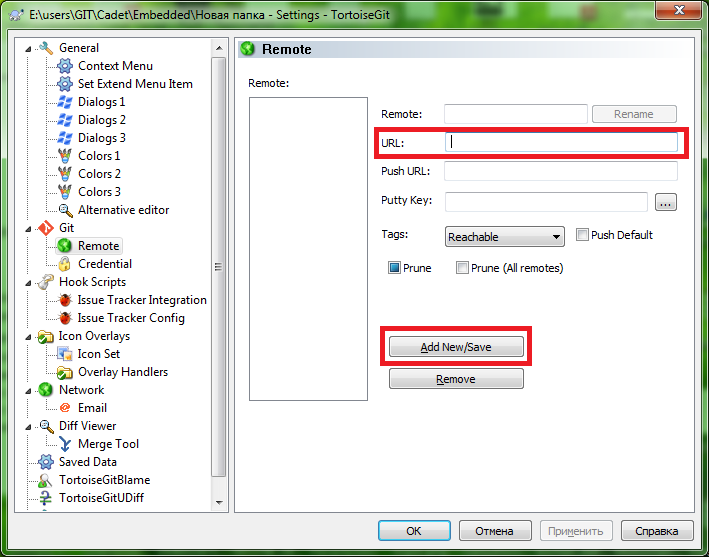
В любом случае, вам нужен будет URL для ремоута. Чтобы его взять, зайдите в свой репозиторий на сервере, нажмите на кнопку URL и скопируйте **среднюю** строку. Чтобы ее скопировать, можно либо выделить текст либо нажать на картиночку справа.



При пуше:

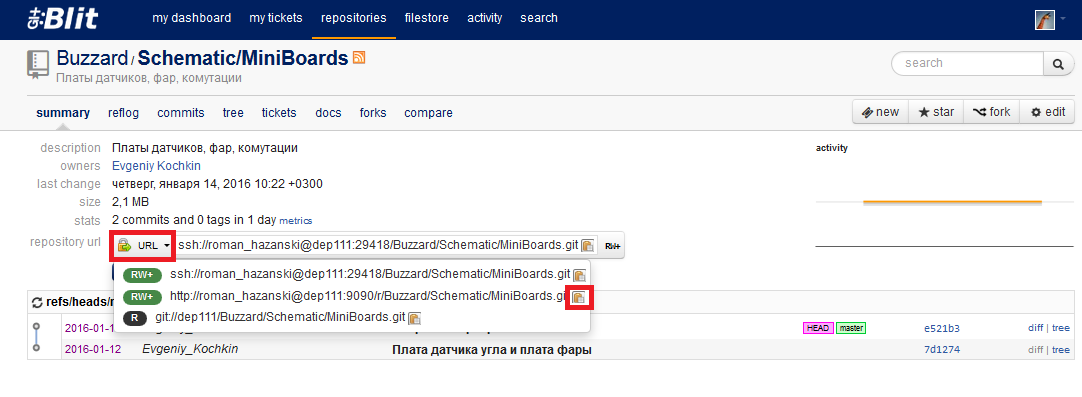
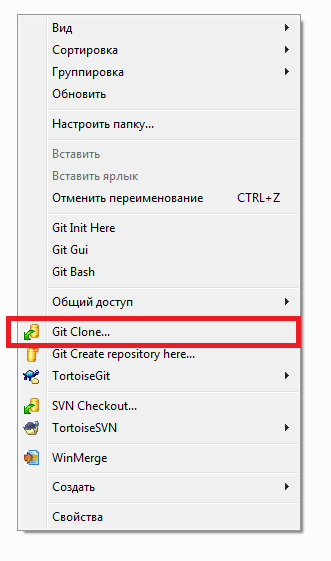
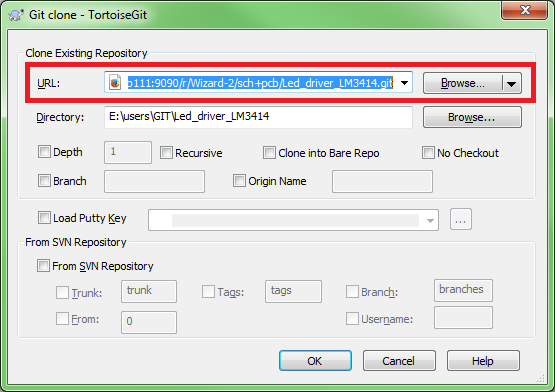
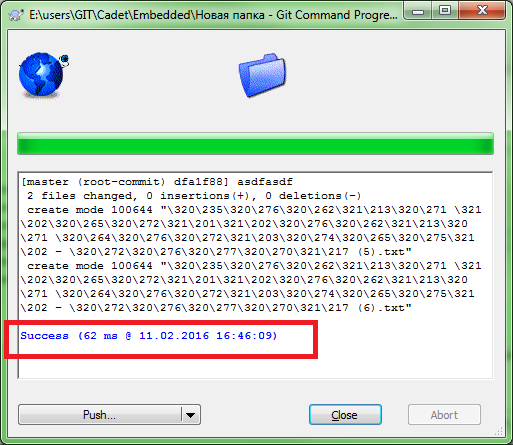
1. вызываете диалог пуша и видите пустой список remote.
2. 
3. Нажимаете на кнопку Manage.
4. Попадаете в настройку ремоутов.

В любой момент вызываете контекстное меню и выбираете tortoise git → settings → remote и попадаете в настройку ремоутов.

В поле URL нужно вставить скопированный вами URL и нажать на Add New/Save. При этом автоматически добавляется ремоут с именем **origin**. 

При каждом следующем пуше origin выбирается автоматически и можно просто жать ОК.

**Как клонировать репозиторий с сервера**

1. Зайдите на страницу проекта на сервере.
2. Нажмите на URL и скопируйте среднюю ссылку (для этого можно нажать на картиночку справа)
3. 
4. Зайдите в папку, в которую вы хотите склонировать репозиторий. При клоне папка с репозиторием создаться автоматически.
5. В контекстном меню выберите Git Clone.
6. 
7. В появившемся диалоговом окне вставьте скопированный URL в строку URL и нажмите ОК.
8. 
9. Дождитесь появления синей надписи success.
10. 

**Что надо делать регулярно**

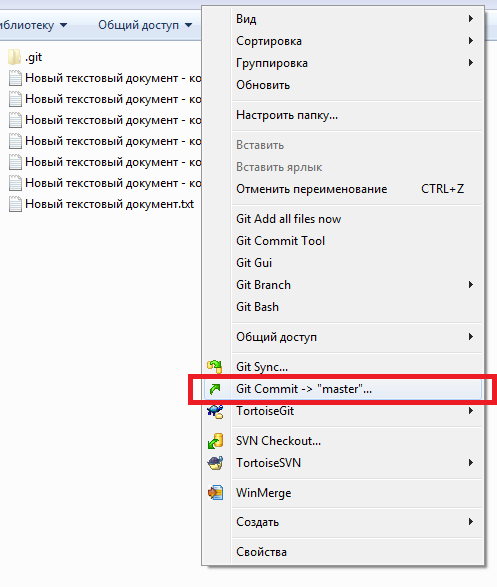
Каждое существенное изменение желательно сохранять. Для этого необходимо сделать коммит (сохранить версию локально) и пуш (сохранить историю на сервере).

Совершенно необязательно делать коммиты на каждый чих, но весьма желательно делать коммит и пуш хотя бы раз в день - чтобы в худшем случае потерять только сегодняшнюю работу.

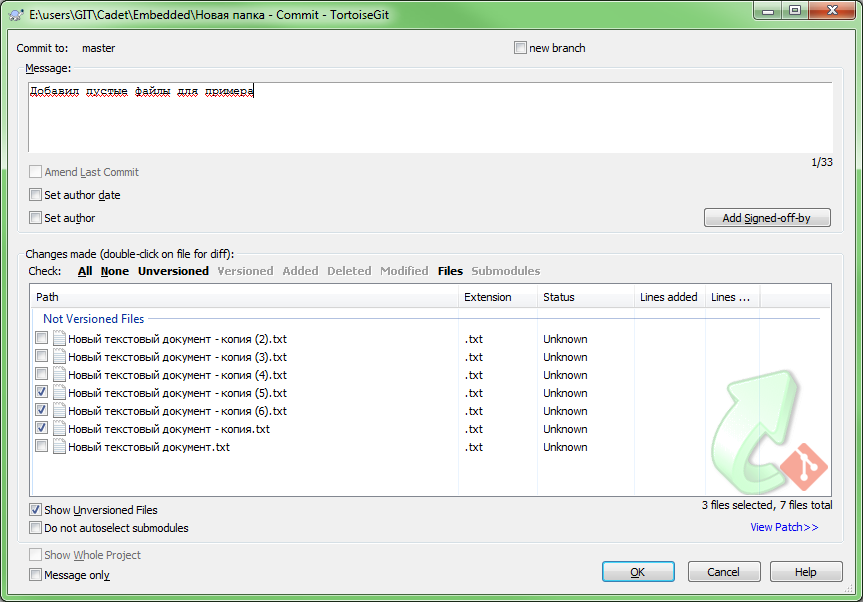
Желательно периодически проверять, что то, что вы запушили на сервер, открывается/собирается/компилируется. Для этого полезно склонировать репозиторий с сервера в произвольную папку.

**Как сделать коммит**

Кликните правой кнопкой на пустом месте в репозитории и выберите Git Commit → <имя ветки>.



Отметьте галочками те изменения, которые вы хотите сохранить в данной версии (по умолчанию выбраны все изменения в добавленных файлах).



Напишите вменяемый комментарий в поле комментария. Без комментария коммит сделать нельзя. Комментарий может быть многострочным (но в быстром логе видна только первая строчка). Коммент должен объяснять **что изменилось в этой версии**, а не что это за проект или что вы ели на обед.

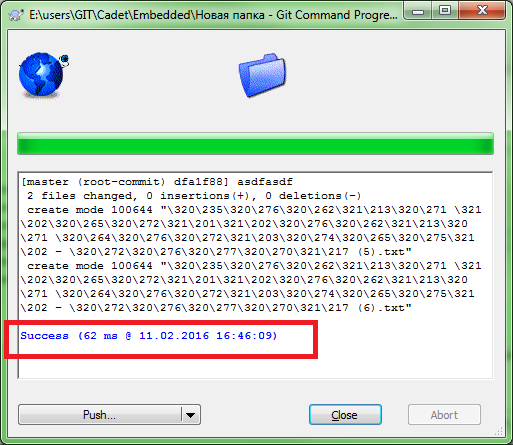
Плохие комментарии (приведены целиком)

* Добавил
* Поменял
* Какие-то изменения
* Децл-лох
* вылаофжыдврафыгшрашфцгурашгфцардшфгцуардшгар

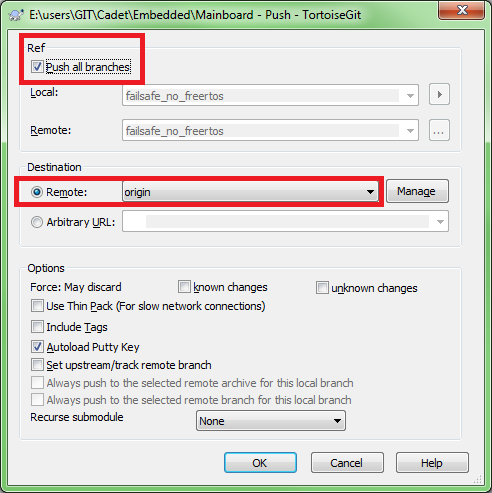
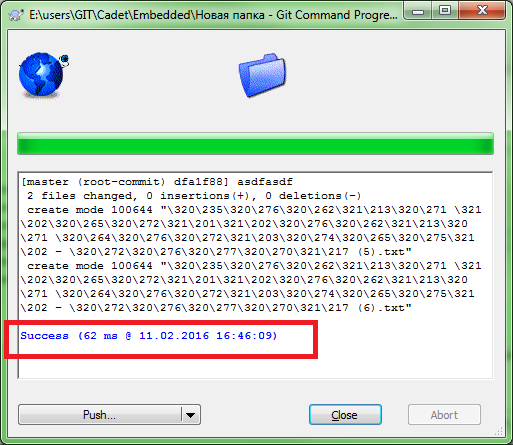
Хорошие комментарии:

* Поправил гербер
* Исправления на монтажном слое
* Значения тока теперь интерпретируются как знаковые
* Движение теперь относительное

Нажмите ОК и дождитесь синей надписи Success.



**Как сделать пуш**

1. Чтобы сделать пуш, нужно сначала сделать хотя бы один коммит (но можно больше).
2. Зайти в контекстное мену tortoise git → push
3. Убедиться, что в открывшемся окне стоит галка push all branches и выбран ремоут origin. Если origin нельзя выбрать, значит его нужно сначала добавить (см. ААААААААААААА)
4. 
5. Нажать ОК.
6. Дождаться появления синей надписи Success.
7. 

**Мерджи**

Слияния у нас происходят, в основном, по нескольким типовым сценариям.

**Просто pull**

Если вы делаете платы, то в основном вы делаете просто pull. Pull - это сначала fetch (т.е. стянуть с сервера все изменения, но никуда не вливать; в логе обновляются ветки origin/что-нибудь), а потом merge удаленной ветки в текущую.

В большинстве случаев можно просто бездумно делать pull. Но иногда возникают разного рода проблемы, текст которых никто не читает. А надо читать!  
Читать надо не “git did not exit cleanly” или “exit code 1” - потому что это просто означает “произошла ошибка”, а то, что написано выше. Чтобы понять причину, достаточно просто прочесть английский текст, но ниже таки приведено разжевывание для ленивых :)  
Там может быть написано:

**Your local changes to the following files would be overwritten by merge**

error: Your local changes to the following files would be overwritten by merge:

stationary\_rcu/mother\_board.SchDoc

Please commit your changes or stash them before you merge.

Aborting

Перевод:

Ваши локальные изменения будут перезаписаны мерджом. Изменения в файлах:

stationary\_rcu/mother\_board.SchDoc

Пожалуйста, сделайте коммит или stash перед мерджом

Почему это произошло? Потому что вы поменяли какой-то файл и ваш коллега поменял какой-то файл. Если речь о платах, то такого какбы вообще быть не должно; два человека не должны менять одну и ту же плату/схему.  
Что же делать?

1) Если изменения, которые внесли в файл вы, не должны там присутствовать, то их нужно откатить. Если они еще не были закоммичены, то можно сделать revert на конкретном файле; при этом изменения просто пропадут. А потом сделать pull еще раз

2) Если же ваш коллега внес изменения, которые не должен был вносить, то нужно сначала дать ему пизды, чтобы он больше так не делал, а потом делать мердж вручную.

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

error: The following untracked working tree files would be overwritten by merge:

stationary\_rcu/mother\_board.SchDoc

Please move or remove them before you merge.

Aborting

Перевод:

У вас есть неверсируемые файлы, которые будут перезаписаны мерджом:

stationary\_rcu/mother\_board.SchDoc

Пожалуйста, уберите их куда-нибудь.

Почему это произошло? Скорее всего, потому что ваш коллега закоммитил временный файл (который вообще не надо коммитить), а у вас он просто лежит.  
Что делать?

Опять же, сначала нужно дать пизды человеку, который закоммитил временный файл.  
Затем, нужно этот файл удалить у себя и сделать pull еще раз. Если таких файлов много, то можно, помолясь, сделать git clean -d -f - это удалит все неверсируемые файлы в репозитории. Безвозвратно удалит!

**Мердж в ветку для внешнего подключения (external)**

Этот мердж делается так же, как любой другой:

1. Переключится в ветку external
2. Сделать merge другой ветки **В** нее
3. Разрешить конфликты
4. Сделать commit

В ветке external должны лежать только те файлы, которые буду использованы в других проектах.  
Например, если ветка master в вашей библиотеке выглядит, просто как проект, где в src\super\_liba лежит, собственно, библиотека, то в ветке external должны быть только файлы из super\_liba, прямо в корне репозитория.

Из-за этого при мердже из master в external постоянно возникают ошибки, связанные с тем, что в external такой файл был удален, а в master - изменился:

CONFLICT (modify/delete): src/uart/uart\_handle\_mock.h deleted in HEAD and modified in develop. Version develop of src/uart/uart\_handle\_mock.h left in tree.

CONFLICT (modify/delete): src/uart/uart\_handle.cpp deleted in HEAD and modified in develop. Version develop of src/uart/uart\_handle.cpp left in tree.

CONFLICT (modify/delete): src/uart/i\_serial\_port.h deleted in HEAD and modified in develop. Version develop of src/uart/i\_serial\_port.h left in tree.

CONFLICT (modify/delete): src/main/main\_free\_rtos.cpp deleted in HEAD and modified in develop. Version develop of src/main/main\_free\_rtos.cpp left in tree.

CONFLICT (modify/delete): src/freertos/i\_freertos\_task.h deleted in HEAD and modified in develop. Version develop of src/freertos/i\_freertos\_task.h left in tree.

CONFLICT (modify/delete): src/freertos/freertos\_wrapper.h deleted in HEAD and modified in develop. Version develop of src/freertos/freertos\_wrapper.h left in tree.

Auto-merging milliganjubus\_freertos\_reg\_table.h

CONFLICT (modify/delete): .cproject deleted in HEAD and modified in develop. Version develop of .cproject left in tree.

Automatic merge failed; fix conflicts and then commit the result.

Решать эти проблемы можно следующим образом:

1. Зайти в папку src/super\_liba, которая образовалась после мерджа, и перенести в корень все файлы, которые должны быть в корне; заменяя существующие.
2. Все файлы, которых в external ветке быть не должно, просто удалить
3. Нажать git → resolve и удалить там все конфликтующие файлы, которых быть не должно
4. Разрешить истинные конфликты, как в обычном мердже (хотя тут их вроде как быть не должно вообще)
5. Сделать коммит, из коммит-сообщения убрать генерированный текст про конфликты.

**Темные искусства**

**Redmine**

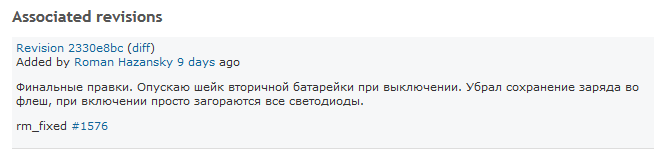
Если вы пользуетесь редмайном, то можно закрывать задачи с помощью гита. Или просто оставлять в задаче ссылку на коммит, который имеет к ней какое-то отношение. Для этого нужно привязать репозиторий с кодом к проекту в редмайне (Открыть проект - Settings - Repositories - New repository). После этого нужно просто оставить в сообщении коммита команду для редмайна и номер задачи. И сделать пуш.

Например:

Финальные правки. Опускаю шейк вторичной батарейки при выключении. Убрал сохранение заряда во флеш, при включении просто загораются все светодиоды.

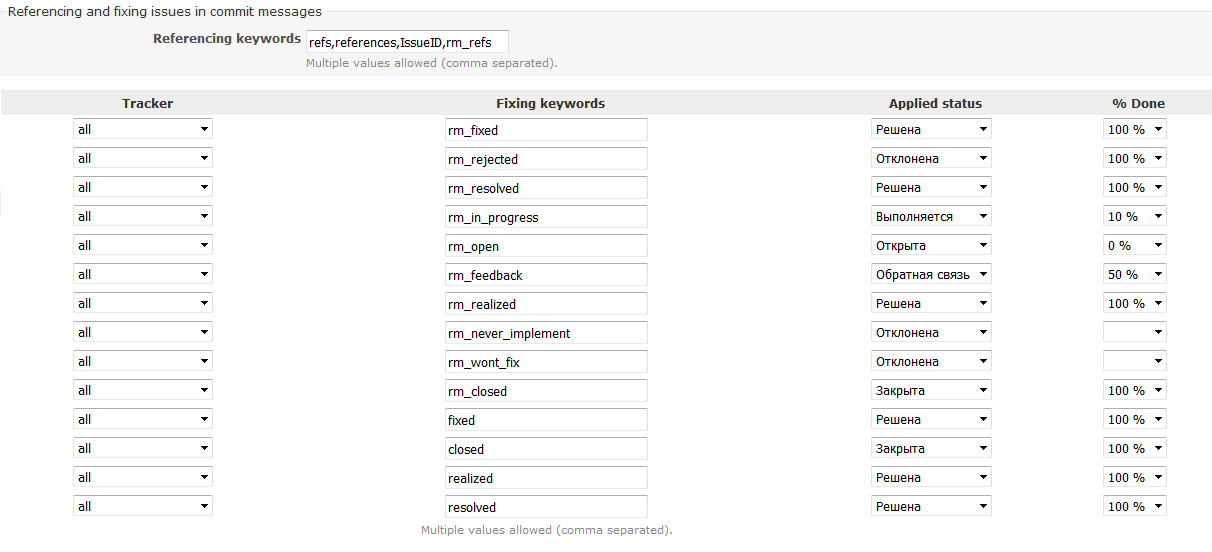
rm\_fixed #1576

После этого в задаче появится вот такая приписка:

[](http://wiki.dep111.rtc.local/_media/manuals:redmine_assoc_revision.png)

Эти команды не являются фиксированными, в настройках редмайна можно добавлять новые или менять их. Разные команды могут менять статус задачи и процент выполнения или ничего не менять.

Текущий перечень команд представлен ниже. Referencing keywords не меняют в задаче ничего, только делают приписку.

[](http://wiki.dep111.rtc.local/_media/manuals:redmine_git_commands.png)

**Поддеревья**

В нашем отделе для переиспользования чужого кода настоятельно рекомендуется использовать не копипасту, а поддеревья. С помощью этого механизма вы получаете локальную копию нужных файлов и отметку в логе гита, которая содержит хеш коммита, который вы себе забрали. Вы всегда знаете, какую версию файлов используете и можете достаточно легко ее обновить. Ваша копия файлов в поддереве не зависит от существования оригинального репозитория, из которого вы сделали поддерево.

**Как подключить поддерево**

1. Зайти в репозиторий локально, кликнуть правой кнопкой и выбрать Git bash here (т.е. открыть консоль гита)
2. В консоли вбить команду:

git subtree add --prefix=src/Имя\_Поддерева http://путь\_к\_удаленному\_репозиторию\_поддерева имя\_ветки --squash

Здесь:

* Имя\_Поддерева должно совпадать с тем, как репозиторий называется на сервере
* имя\_ветки - это имя той ветки, которую вы хотите себе присосать. Как правило, ветки для внешнего подключения называются external или external\_что-нибудь
* squash нужен, чтобы слепить всю историю поддерева в один коммит.

В случае успеха вы увидите нечто подобное:

$ git subtree add --prefix=src/MilliGanjubusCore http://gitblit.dep111.rtc.local/r/EmbeddedLibs/MilliGanjubusCore.git external --squash

git fetch http://roman\_hazanski@gitblit.dep111.rtc.local/r/EmbeddedLibs/MilliGanjubusCore.git external

warning: no common commits

remote: Counting objects: 1999, done

remote: Finding sources: 100% (1999/1999)

remote: Getting sizes: 100% (546/546)

remote: Compressing objects: 76% (7047/9221)

remote: Total 1999 (delta 1282), reused 1998 (delta 1282)

Receiving objects: 100% (1999/1999), 1.47 MiB | 510.00 KiB/s, done.

Resolving deltas: 100% (1282/1282), done.

From http://gitblit.dep111.rtc.local/r/EmbeddedLibs/MilliGanjubusCore

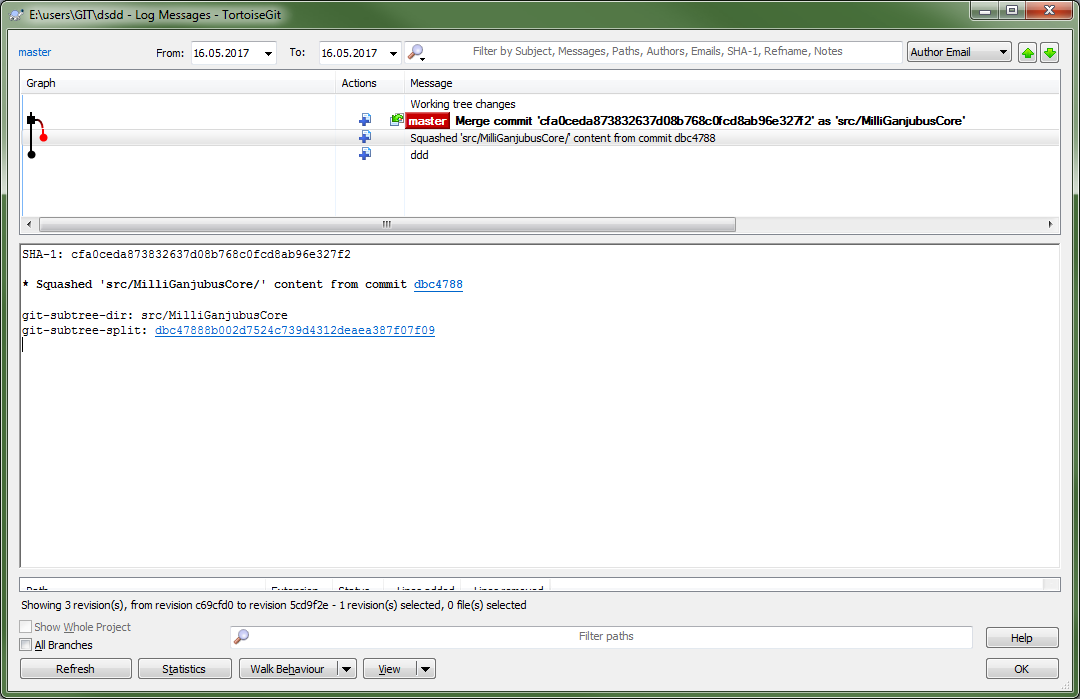
\* branch external -> FETCH\_HEAD

you just checked out: c69cfd09e6507f44817322e513ac24c4c198b52e c69cfd09e6507f44817322e513ac24c4c198b52e 0

submodules:

grep: .gitmodules: No such file or directory

Added dir 'src/MilliGanjubusCore'

После этого в логе должен появиться “писюн”: [](http://wiki.dep111.rtc.local/_media/manuals:tortoise_git_subtree_log.png)

**Как обновить поддерево (подсосать последнюю версию к себе)**

Аналогично подключению, только команда:

git subtree pull

Если вы переименовывали папку с поддеревом, то просто git subtree pull не прокатит. Нужно либо переподключить поддерево вручную (удалить папку и сделать subtree add), либо сделать вот такое:

git subtree split --rejoin --prefix=новый\_путь\_к\_поддереву HEAD

И потом обычный pull по новому префиксу.

Будьте внимательны, subtree split может не справиться и тогда в вашей рабочей копии будет коммит из самого поддерева. В таком случае, нужно сделать hard reset на последний валидный коммит и переподключить дерево вручную.

**Как запушить поддерево (запихать свои изменения в origin поддерева)**

git subtree push --prefix=src/Имя\_Поддерева http://путь\_к\_удаленному\_репозиторию\_поддерева имя\_ветки\_у\_поддерева

Здесь:

* Имя\_Поддерева должно совпадать с тем, как репозиторий называется на сервере
* имя\_ветки\_у\_поддерева- это имя ветки в репозитории поддерева, в которую вы хотите запушить. Как правило, ветки для внешнего подключения называются external или external\_что-нибудь

**Как оформить свою библиотеку для поддревесного подключения**

1. В вашем репозитории должна быть ветка external, прямо в корне которой должны находится только файлы для внешнего подключения.
2. Если веток для внешнего подключения несколько, каждая из них должна начинаться на external\_xxx, причем из ххх должно быть понятно, чем эта ветка отличается от остальных. Например, external\_stm32f10x - ветка для процов серии stm32f10x.
3. **Вам нужно тридцать три с половиной раза проверить, что с кодом во внешних ветках все в порядке, что там не используются табы, кривая кодировка и т.п.**
4. В репозитории должна быть ветка (master или develop), в которой должен лежать проект с примером использования вашей либы.
5. Все инклуды в файлах для внешнего подключения не должны использовать абсолютные пути или пути, которые поднимаются выше уровня папки для внешнего подключения.
6. Если ваша либа зависит от других либ (наших), то вы должны предполагать, что они лежат в папке src, а сама папка src прописана в инклудах. Т.е. вы не должны делать #include “../circular\_buffer/circular\_buffer.h”, вы должны делать #include “circular\_buffer/circular\_buffer.h”.
7. Если в вашей либе есть несколько вложенных папок, то в них допустимо подниматься на нужное количество уровней выше для подключения принадлежащих вашей либе файлов.
8. Если вашей либе ххх необходим заголовочный файл с настройками, то он должен называться xxx\_settings.h и подключаться без путей - #include “xxx\_settings.h”.
9. Поскольку настройки в каждом проекте могут меняться, сам файл с настройками не должен входить в состав либы. Но в состав либы должен входить пример такого файла с названием xxx\_settings\_example.txt.
10. **Предполагайте, что большинство будет использовать этот пример как есть, поэтому настройки по-умолчанию должны либо вызывать static\_assert и таким образом вынуждать пользователя их поменять, либо должны быть сразу максимально универсальными.**
11. Вам необходимо получить репозиторий в EmbeddedLibs и запушить вашу либу туда. Репозиторий в EmbeddedLibs становится основным, вся разработка должна проходить в нем, а не в вашем приватном репозитории.

**Что делать, если что-то пошло не так**

1. Не паниковать
2. Не забивать болт
3. Постараться решить проблему самостоятельно:
   1. Поспрашивать у товарищей
   2. Погуглить
4. Обратиться к администратору гита.