Оглавление

[Git 2](#_Toc25297994)

[Работа с репозиторием 3](#_Toc25297995)

[Получение репозитория 3](#_Toc25297996)

[Основные используемые функции 3](#_Toc25297997)

[1. Обновление текущей ветки из центрального репозитория: 3](#_Toc25297998)

[2. Отправка текущей ветки в центральный репозиторий: 4](#_Toc25297999)

[3. Переключение на некоторую ветку: 4](#_Toc25298000)

[4. Создание новой ветки 5](#_Toc25298001)

[5. Удаление веток 6](#_Toc25298002)

[6. Слияние ветки с текущей 6](#_Toc25298003)

[Стандартные процедуры работы 6](#_Toc25298004)

[1. «Начало работы над задачей» 6](#_Toc25298005)

[2. «Коммит очередного кусочка работы» 6](#_Toc25298006)

[3. «Отправка ветки на центральный репозиторий» 6](#_Toc25298007)

[4. «Ребейз относительно мастера» 7](#_Toc25298008)

[5. «Кратковременное сохранение состояния изменений» 7](#_Toc25298009)

[6. «Длительное сохранение состояния изменений» 7](#_Toc25298010)

[7. «Ревью ветки» 8](#_Toc25298011)

[Работа с Sublime merge 9](#_Toc25298012)

# Git

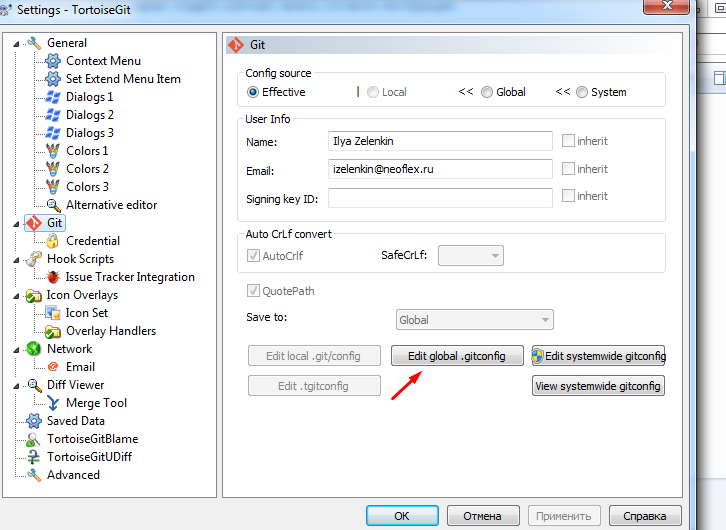
Установить GIT, https://git-scm.com/

Установить tortoiseGIT, <https://tortoisegit.org/download/>

Установить sublimemerge <https://www.sublimemerge.com/>

Адрес репозитория https://github.com/NeoPractice/Repo.git

В клиенте tortoiseGITнастроить



Добавить параметры

[http]

sslVerify = false

[user]

name = [Имя Фамилия латиницей]

email = [email]

[credential "github.com/NeoPractice/Repo"]

git = .helper

git = .username

[credential "https://NeoPractice:NeoStudent123$@github.com/NeoPractice/Repo"]

git = .helper

git = .username

Выкачать репозиторий проекта:

• https://github.com/NeoPractice/Repo.git - репозиторий учебного центра

В URL для выкачки добавить NeoPractice:NeoStudent123$: https://NeoPractice:NeoStudent123$@github.com/NeoPractice/Repo.git

Общая информация по технологии Git <https://www.youtube.com/watch?v=RiYz59qVypg>

## Терминология GIT

**Git или Гит** — система контроля и управления версиями файлов.

**GitHub или Гитхаб** — веб-сервис для размещения репозиториев и совместной разработки проектов.

**Репозиторий Git** — каталог файловой системы, в котором находятся: файлы конфигурации, файлы журналов операций, выполняемых над репозиторием, индекс расположения файлов и хранилище, содержащее сами контролируемые файлы.

**Локальный репозиторий** — репозиторий, расположенный на локальном компьютере разработчика в каталоге. Именно в нём происходит разработка и фиксация изменений, которые отправляются на удалённый репозиторий.

**Удалённый репозиторий** — репозиторий, находящийся на удалённом сервере. Это общий репозиторий, в который приходят все изменения и из которого забираются все обновления.

**Форк (Fork)** — копия репозитория. Его также можно рассматривать как внешнюю ветку для текущего репозитория. Копия вашего открытого репозитория на Гитхабе может быть сделана любым пользователем, после чего он может прислать изменения в ваш репозиторий через пулреквест.

**Обновиться из апстрима** — обновить свою локальную версию форка до последней версии основного репозитория, от которого сделан форк.

**Обновиться из ориджина** — обновить свою локальную версию репозитория до последней удалённой версии этого репозитория.

**Клонирование (Clone)** — скачивание репозитория с удалённого сервера на локальный компьютер в определённый каталог для дальнейшей работы с этим каталогом как с репозиторием.

**Ветка (Branch)** — это параллельная версия репозитория. Она включена в этот репозиторий, но не влияет на главную версию, тем самым позволяя свободно работать в параллельной. Когда вы внесли нужные изменения, то вы можете объединить их с главной версией.

**Мастер (Master)** — главная или основная ветка репозитория.

**Коммит (Commit)** — фиксация изменений или запись изменений в репозиторий. Коммит происходит на локальной машине.

**Пул (Pull)** — получение последних изменений с удалённого сервера репозитория.

**Пуш (Push)** — отправка всех неотправленных коммитов на удалённый сервер репозитория.

**Пулреквест (Pull Request)** — запрос на слияние форка репозитория с основным репозиторием. Пулреквест может быть принят или отклонён вами, как владельцем репозитория.

**Мёрдж (Merge)** — слияние изменений из какой-либо ветки репозитория с любой веткой этого же репозитория. Чаще всего слияние изменений из ветки репозитория с основной веткой репозитория.

**Кодревью** — процесс проверки кода на соответствие определённым требованиям, задачам и внешнему виду.

## Работа с репозиторием

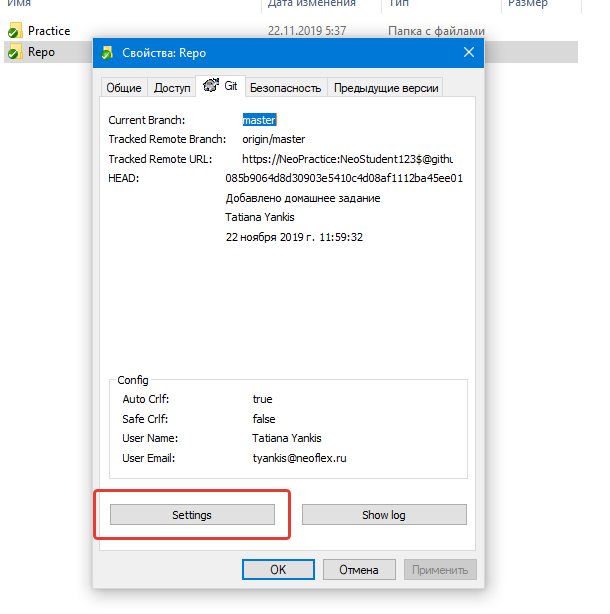
Общая информация по технологии Git <https://www.youtube.com/watch?v=RiYz59qVypg>

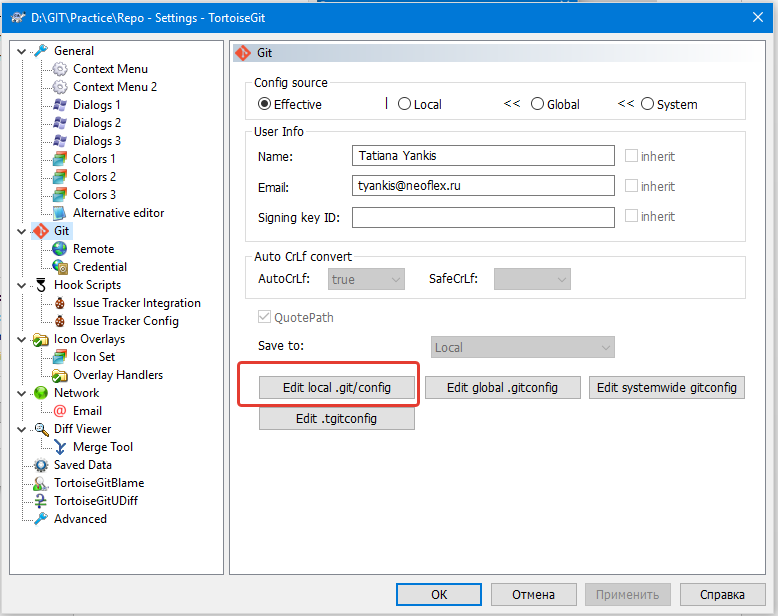
## Получение репозитория

В папке, где будут размещаться все рабочие проекты, жмём  
Правой кнопкой→TortoiseGit→Clone, вводим адрес центрального репозитория

<https://github.com/NeoPractice/Repo.git>

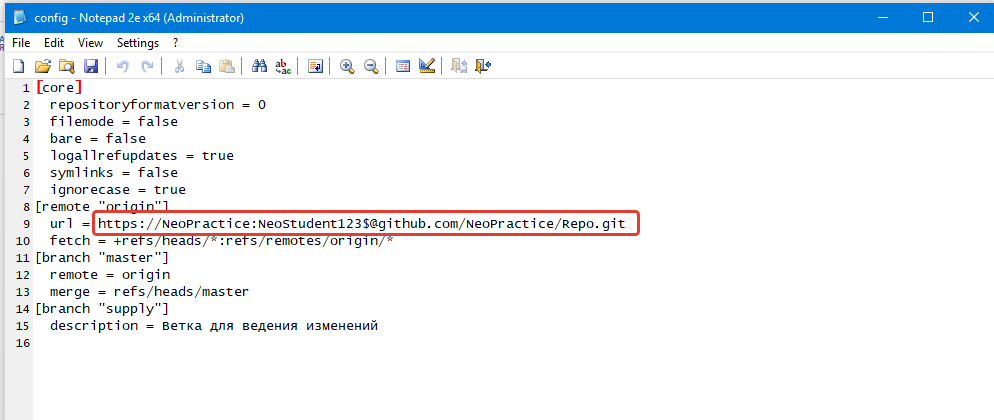
Создается каталог, правой клавишей мыши открываем свойство папки и переходим на закладку GIT. Далее настраиваем свойства гита

  
В открывшемся окне выбираем Edit local config



В открывшемся блокноте редактируем строку на

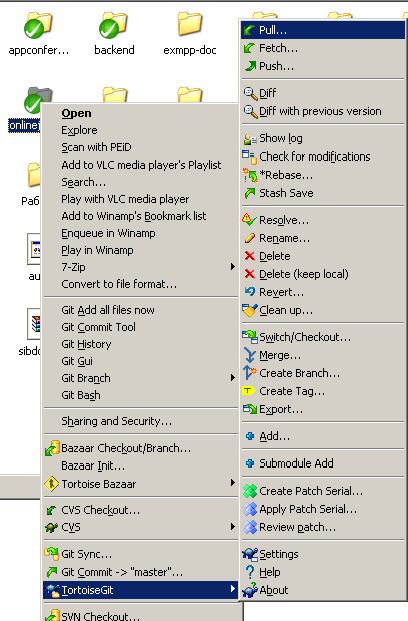
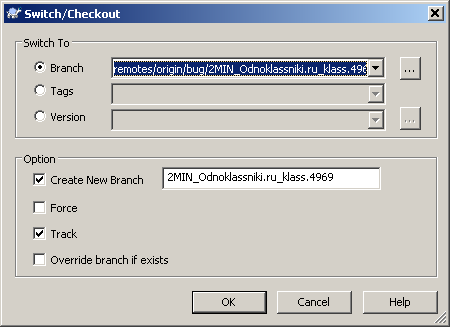
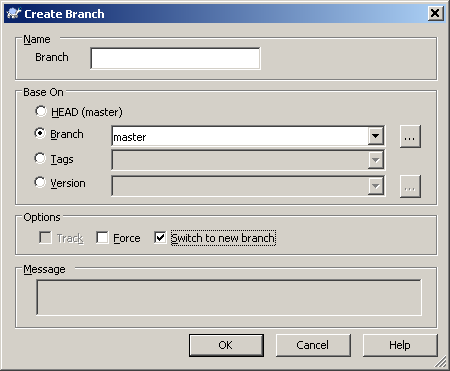
https://NeoPractice:NeoStudent123$@github.com/NeoPractice/Repo.git



Сохраняем настройки

## Основные используемые функции

При работе используются либо консольные утилиты, либо графический интерфейс.

1. Обновление текущей ветки из центрального репозитория:  
   В контекстном меню папки с исходниками TortoiseGit->Pull  
     
     
   
2. Отправка текущей ветки в центральный репозиторий:  
   В контекстном меню TortoiseGit→Push  
   Выбираем в Local сперва master, потом нашу текущую ветку.  
   Remote заполняется при этом автоматически.  
   Название remote ветки должно быть идентично local
3. Переключение на некоторую ветку:  
   В контекстном меню TortoiseGit→Switch/Checkout  
   В меню Branch выбрать remotes/origin/<нужная ветка>  
   [v] «Create new branch», название <нужная ветка>, [v] «Track»   
     
     
   
4. Создание новой ветки  
   Контекстное меню→TortoiseGit→Create Branch  
   В Name в Branch вводим нужное название ветки  
   Чтобы ветка базировалась от текущей, в Base On выбираем HEAD  
   Чтобы ветка базировалась от мастера, выбираем Branch, в списке master.  
   Ставим [v] Switch to new branch чтобы сразу переключиться на ветку.  
     
     
   
5. Удаление веток  
   Открываем меню зажав кнопку Shift → TortoiseGit → Browse Reference  
   в разделе heads лежат локальные ветки, в разделе remotes\origin удалённые.  
   Выбираем нужную ветку, жмём правой кнопкой → Delete Remote Branch  
   Для локальной ветки, соответственно, Delete Branch.
6. Слияние ветки с текущей  
   Контекстное меню → TortoiseGit → Merge  
   выбираем Branch который нужно слить с текущей веткой  
   ставим обязательно галочку «No fast forward», Merge message не трогаем.
7. Файлы с изменениями помечены красными восклицательными знаками.  
   Чтобы посмотреть общую картину изменений,   
   Меню→Git Commit -> “ветка”  
   Внизу список изменений в репозитории, обязательно нажимать галочку «View patch» и проверять изменения на предмет соответствия Правилам ведения чистых коммитов.

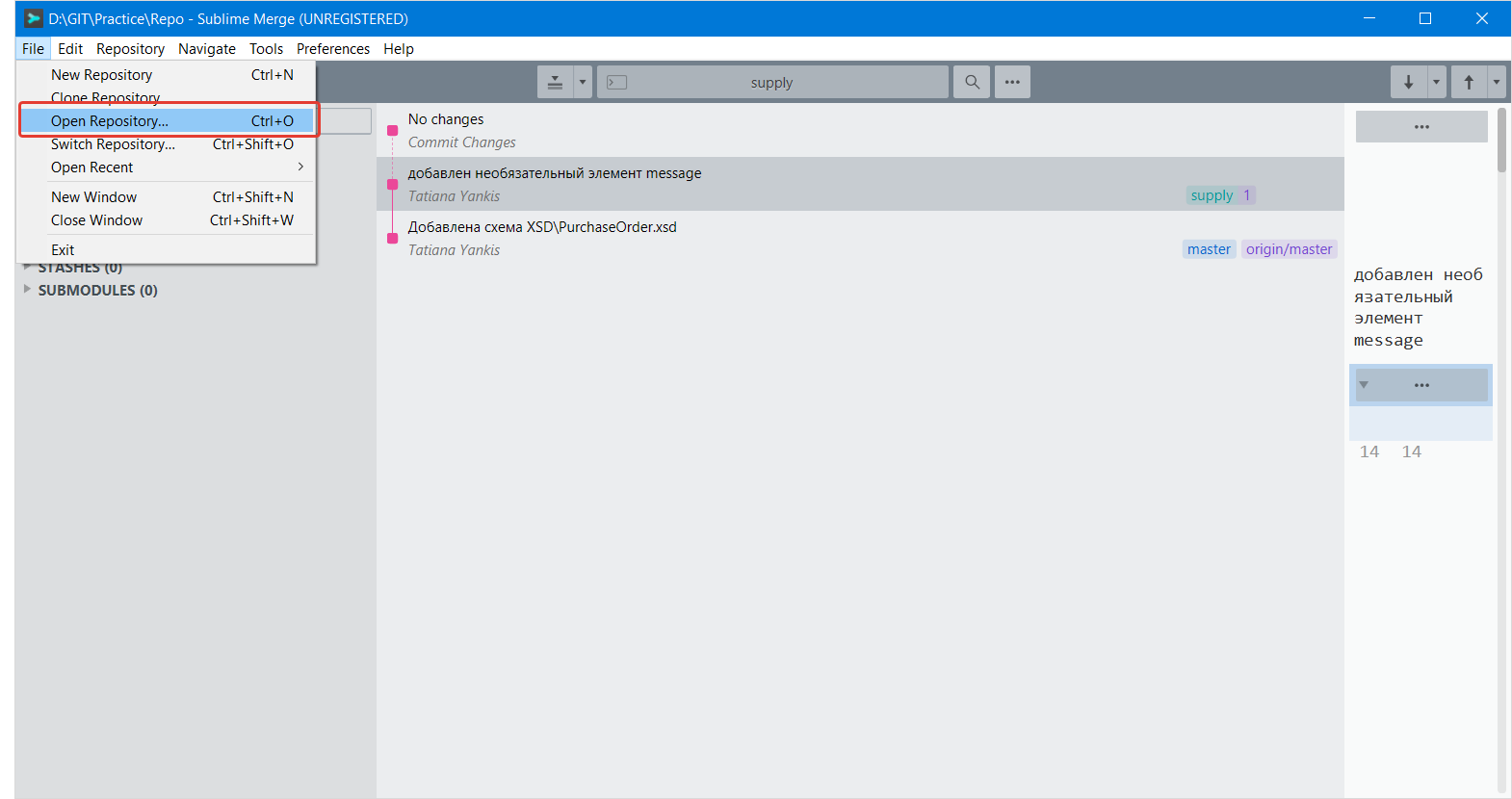
## Стандартные процедуры работы

1. «Начало работы над задачей»  
   Выполняется перед началом работы над задачей. Дерево должно быть без изменений.  
   Меню → TortoiseGit → Switch/Checkout,  
   Branch: master  
   Меню → TortoiseGit → Pull  
   Меню → TortoiseGit → Create Branch  
   Name Branch: название новой ветки  
   Base On: HEAD (master)  
   [x] Switch to new branch
2. «Коммит очередного кусочка работы»  
   Выполняется после выполнения некого изменения, суть которого целостная.  
   Меню → Git commit -> “имя ветки”  
   Отметить файлы, только нужные для данного конкретного коммита  
   Обязательно щелкнуть на «View Patch», и убедиться   
   в соответствии правилам ведения чистых коммита  
   В message ввести описание изменения, соответствующее правилам
3. «Отправка ветки на центральный репозиторий»  
   Выполняется после завершения работы, либо в конце каждого дня (чтобы был бакап на сервере), либо если нужно какие-то изменения показать коллеге.  
   Меню → TortoiseGit → Push   
   Выбираем в Local сперва master, потом нашу текущую ветку.  
   Remote заполняется при этом автоматически.  
   Название remote ветки должно быть идентично local  
   Не следует делать push после каждого коммита, так как это потребует доступа до удалённого сервера, и, соответственно, времени, потраченного впустую.
4. «Ребейз относительно мастера»  
   Выполняется перед заливкой на сервер законченной задачи, когда все изменения уже закоммичены.  
   Меню → Git Sync  
   Local branch: master  
   Remote branch: master  
   Правая стрелочка вниз у первой кнопки (Pull по умолчанию), Fetch  
   Меню → TortoiseGit → \*Rebase  
   Branch: текущая ветка  
   UpStream: master  
   Если будут конфликты – разобраться с ними через вкладку «Conflict File»  
   На выбор, правой кнопкой файла, утилиты  
   Edit Conflicts – позволяет по каждому расхождению выбрать  
   использовать версию какую блока  
   Resolve Conflicts Using  
   theirs – использовать версию мастера  
   mine – использовать версию текущей ветки  
   Open – открыть в редакторе, и исправить вручную  
   После исправления сделать Resolve  
   После исправления всех конфликтов нажать Commit  
   После ребейза нажать Done
5. «Кратковременное сохранение состояния изменений»  
   Выполняется, если требуется временно приостановить работу над текущей веткой на короткое время (например, на ревью, или чтобы сделать какую-либо двухминутную задачу).  
   Меню → TortoiseGit → Stash Save  
   После этого дерево чисто, можно переключиться на другую ветку/мастер и так далее, поработать, после чего восстановить состояние, если переключиться обратно на рабочую ветку, и сделать  
   Меню → TortoiseGit → Stash Pop  
   Тем самым восстановив состояние изменения.
6. «Длительное сохранение состояния изменений»  
   Выполняется в конце рабочих суток, чтобы даже частичные изменения были забакаплены; либо при необходимости срочно переключиться на решение другой задачи, которая может занять значительно больше 5-10 минут.  
   Меню → Git Commit -> “ветка”  
   Отмечаем все-все изменения (Select/Deselect All)  
   В текст сообщения пишем «Partial commit»  
   Позже, для возврата к тому же состоянию как было до, переключаемся на рабочую ветку, и делаем  
   Меню → TortoiseGit → Show Log  
   Выделяем коммит, который идет в дереве сразу перед «Partial commit»  
   Правой кнопкой → Reset <ветка> to this  
   Reset type: Mixed
7. «Ревью ветки»  
   Выполняется на чистом дереве, временно сохраните изменения согласно пункта 5, если требуется.  
   Меню → TortoiseGit → Switch/Checkout  
   Branch: master  
   Меню → TortoiseGit → Pull  
   Меню → TortoiseGit → Switch/Checkout  
   Branch: remotes/origin/нужнаяветка  
   [x] Create new branch: нужнаяветка  
   [x] Force  
   [x] Track  
   [x] Override branch if exists  
   Меню → TortoiseGit → \*Rebase  
   Branch: нужнаяветка  
   UpStream: master  
   Ветка ветка должна быть «up to date» или заребейзится без конфликтов.  
   == Анализируем изменения просмотром лога изменений через  
   Меню → TortoiseGit → Show log  
   и смотрим изменения от master до последнего  
   == Смотрим общее изменение относительно мастера  
   Меню → TortoiseGit → Diff with previous version  
   Version 1: HEAD  
   Version 2: master  
   == если всё хорошо, делаем  
   Меню → TortoiseGit → Switch/Checkout  
   Branch: master  
   Меню → TortoiseGit → Merge  
   From: нужнаяветка  
   [x] No fast forward  
   Сообщение не редактируем.  
   Меню → TortoiseGit → Push  
   И удаляем ветки:  
   Shift + Меню → TortoiseGit → Browser Reference  
   в дереве слева Refs => heads => находим ветку, правой кнопкой, Delete branch  
   в дереве слева remotes => origin => находим ветку, правой кнопкой,  
   Delete remote branch

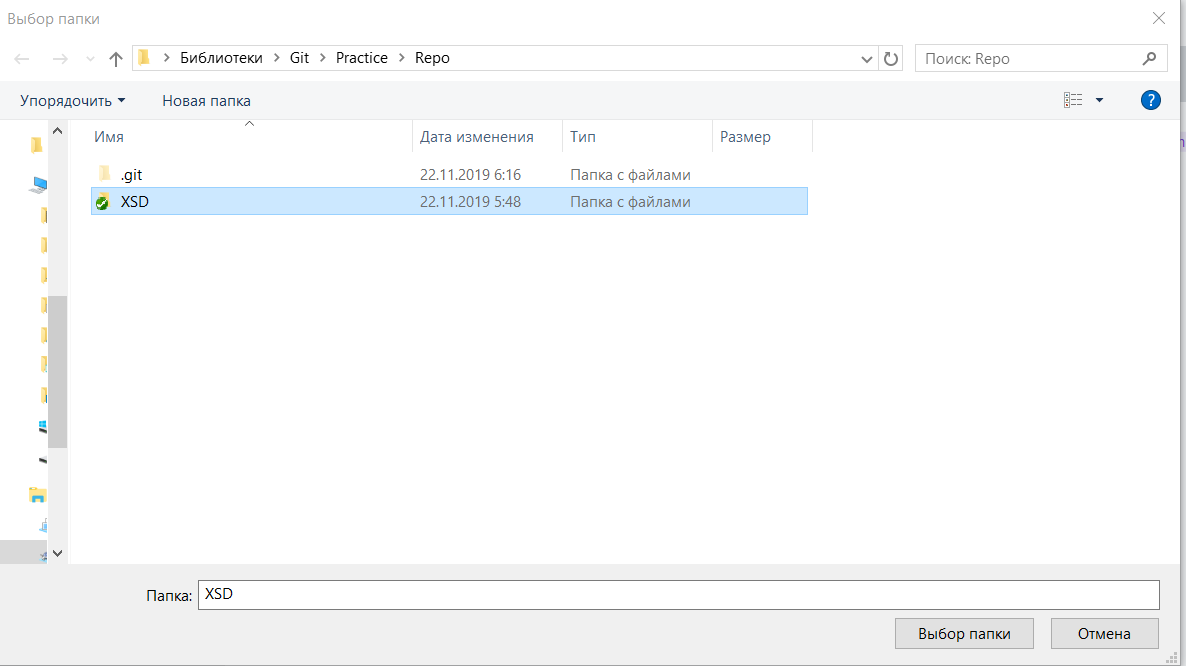
## Работа с Sublime merge

Запустить программу

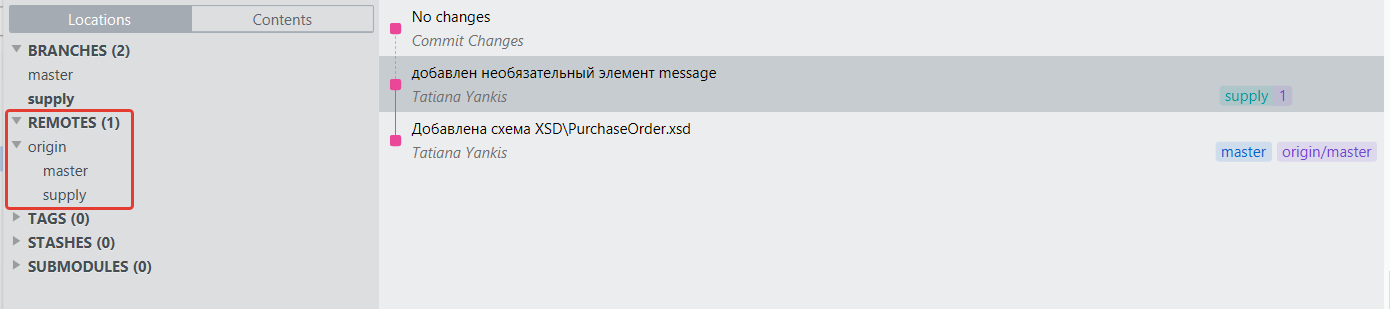
Открыть File – Open Repository



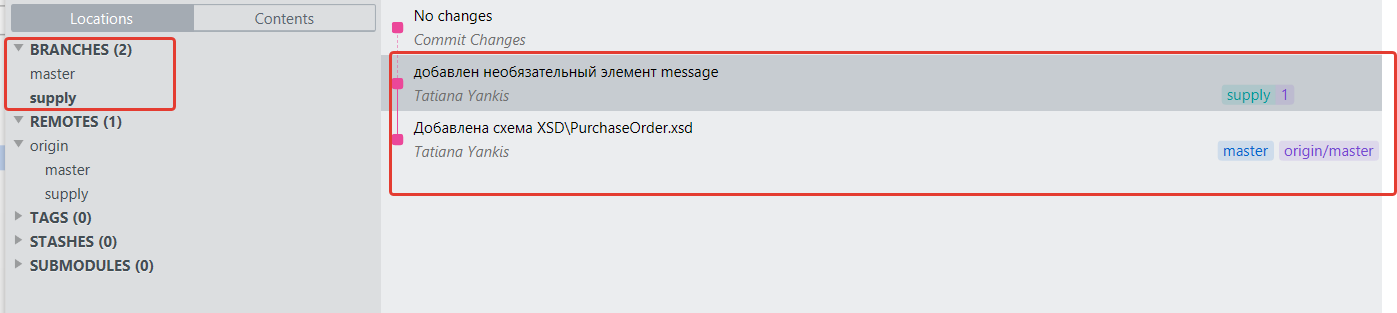
Выбрать папку репозитория



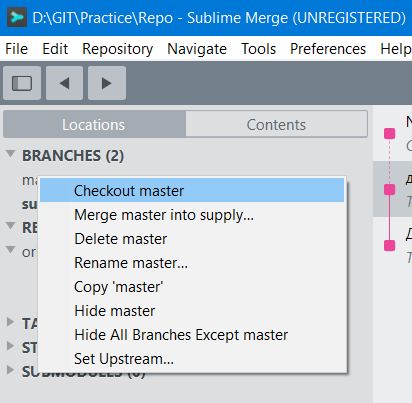
После того, как откроется репозиторий можно будет на левой панели увидеть все удаленные ветки



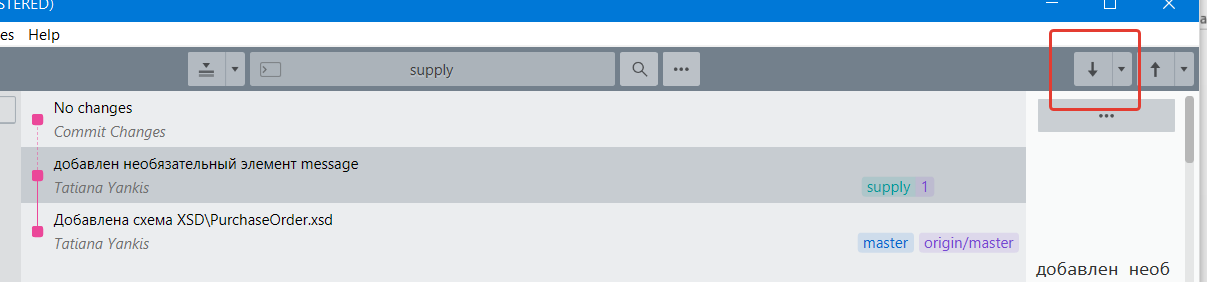
И локальные ветки, а также увидеть историю коммитов



Для переключения между локальными ветками используем команду checkout <имя ветки>



Для команды pull используем кнопки справа



# Домашнее задание

Создать схему XSD в соответствии со спецификацией. При этом необходимо разместить файл в локальную папку, синхронизированную с репозиторием GIT «https://github.com/NeoPractice/Repo.git».

* Вы должны отслеживать всю свою работу в Git, включая код.
* Выполните или попробуйте 1 -7 шаги из Стандартные процедуры работы
* Для этого создайте свою ветку на основе master-ветки (шаг 1 – Начало работы над задачей)
* Делайте коммит каждый раз, когда вы отдыхаете или останавливаетесь (шаг 2 - «Коммит очередного кусочка работы»)
* Создавайте комментарии при выполнении коммита, начиная с глагола (например, «созданы элементы Request и Response»).
* Сделать хотя бы 5 коммитов на протяжении выполнения задания.
* Когда XSD схема будет готова, перенесите созданную ветку в центральный репозиторий (шаг 3 «Отправка ветки на центральный репозиторий»)

## Спецификация на точку интеграции «Список депозитов».

### Формат запроса

**Кратность (Обязательность)**

* **Кратность –** определяет обязательность элемента и допустимое количество таких элементов в документе.   
   [0] –Необязательный элемент, т.е. сообщение может не содержать этот элемент или этот элемент может встретиться в сообщении не более одного раза.
* [1] – Обязательный элемент. Сообщение обязательно должно содержать такой элемент и этот элемент не может встретиться в документе повторно.
* [0-N] – Необязательный элемент, который может записан в сообщении несколько раз. При необходимости можно конкретизировать значение N.

| **Имя элемента** | **Входит в** | **Тип** | **Крат./**  **Обяз.** | **Элемент/**  **Атрибут** | **Назначение** | **Комментарий** |
| --- | --- | --- | --- | --- | --- | --- |
| Request |  |  | [1] | Элемент | Основной охватывающий блок запроса. | |
| App-Data | Request |  | **[1]** | Элемент | Блок | |
| Type | App-Data | string | [1] | Атрибут | Тип запроса | Константа «dimGetCliObjectList» |
| CIMVersion | App-Data | string | [0] | Атрибут | Номер версии | Константа «3» |
| ObjectType | App-Data | string | [0] | Атрибут | Тип Объекта | Константа «Deposit» |
| Client | App-Data |  | [1] | Элемент | Блок данных клиента | |
| UNI | Client |  | [1] | Элемент | Идентификатор объекта |  |
| Globalid | UNI | String | [1] | Атрибут | Глобальный межсистемный идентификатор | Пустая строка |
| System | UNI | String | [1] | Атрибут | Код системы |  |
| Type | UNI | String | [1] | Атрибут | Тип объекта |  |
| Number | UNI | String | [1] | Атрибут | Идентификаторобъекта в системе | Идентификаторклиента в СИФ |
| Identification | Client |  | [1] | Элемент | Элементидентификацииклиента |  |
| IdntList | Identification |  | [1] | Элемент | Элементспискаидентификаторов |  |
| IdntRecord | IdntList |  | [n] | Элемент | Элементидентификатораклиента | Обязательно наличие |
| Status | IdntRecord | String | [1] | Атрибут | Статус | Константа «Actual» |
| System | IdntRecord | String | [1] | Атрибут | Система |  |
| OrgUnit | IdntRecord | String | [1] | Атрибут | Подразделение |  |
| Id | IdntRecord | String | [1] | Атрибут | Идентификатор |  |
| NotClosed | Request |  | [0] | Элемент | Элемент параметра запроса «не включать закрытые» |  |
| Value | NotClosed | String | [1] | Атрибут | Значение параметра запроса «не включать закрытые» | Константа«false» |

### Формат ответа

| **Имя элемента** | **Входит в** | **Тип** | **Крат./**  **Обяз.** | **Элемент/Атрибут** | **Назначение** | **Комментарий** |
| --- | --- | --- | --- | --- | --- | --- |
| Response |  |  | [1] | Элемент | Основной охватывающий блок ответа. | |
| Bis-Data | Response |  | [1] | Элемент | Блок служебной транспортной информации исходного запроса. Используется шиной для маршрутизации ответа.Содержание блока описывается в отдельной документации. | |
| App-Data | Response |  | [1] | Элемент | Охватывающий блок прикладных данных исходного запроса. | |
| Response | App-Data |  | [1] | Элемент | блок ответа. | |
| ResultCode | Response | Целое число | [1] | Атрибут | Код результата | «0» при успешном выполнении.  Коды ошибок см. ниже в таблице |
| ErrorString | Response | string | [0] | Атрибут | Сообщение об ошибке | Обязательно при ненулевом результате.  Примеры см. ниже в таблице |
| ObjectList | Response |  | [0] | Элемент | Элемент прикладных данных ответа |  |
| cifID | ObjectList | string | [0] | Атрибут | Идентификатор клиента в СИФ | Из запроса |
| AccountList | ObjectList |  | [0] | Элемент | Элемент списка счетов |  |
| LoanList | ObjectList |  | [0] | Элемент | Элемент списка кредитов |
| LoanApplicationList | ObjectList |  | [0] | Элемент | Элемент списка |
| DepositList | ObjectList |  | [0] | Элемент | Элемент списка депозитов |  |
| Deposit | DepositList |  | [n] | Элемент | Элемент депозита |  |
| Value | Deposit | String | [1] | Атрибут | Вид депозит/счет | Для вклада физ лица возвращается «Deposit» |
| DepositContract | Deposit |  | [1] | Элемент | Блок данных договора |  |
| Status | DepositContract |  | [1] | Элемент | Состояние |  |
| Value | Status | String | [1] | Атрибут |  |  |
| Number | DepositContract |  | [1] | Элемент | Номер договора |  |
| Value | Number | String | [1] | Атрибут |  |  |
| OpenDate | DepositContract |  | [1] | Элемент | Дата открытия договора |  |
| Value | OpenDate | String | [1] | Атрибут |  |  |
| EndDate | DepositContract |  | [1] | Элемент | Дата окончания договора |  |
| Comment | EndDate | String | [0] | Атрибут | Комментарий |  |
| Value | EndDate | String | [1] | Атрибут | Дата в формате «гггг-мм-дд» | Может быть пустой |
| CloseDate | DepositContract |  | [1] | Элемент | Датазакрытиядоговора |  |
| Value | CloseDate | String(10) | [1] | Атрибут | Дата в формате «гггг-мм-дд» | Может быть пустой |
| Currency | DepositContract |  | [1] | Элемент | Валюта депозита |  |
| Value | Currency | String(3) | [1] | Атрибут |  |  |
| Period | DepositContract |  | [1] | Элемент | Срок вклада |  |
| Unit | Period | String | [1] | Атрибут | Единица измерения | Допустимые значения :  «Day», «year» или «mounth» |
| Value | Period | int | [1] | Атрибут | Срок вклада в заданных единицах изменения |  |