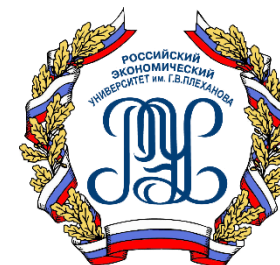


BACKEND - РАЗРАБОТКА НА ЯЗЫКЕ PYTHON

РЭУ ИМ. Г.В. ПЛЕХАНОВА МПТ



<https://ucmpt.ru/>

<https://do.rea.ru/>

DESKTOP ПРИЛОЖЕНИЯ

- **Настольное (desktop) приложение - это программа, обрабатываемая на стороне клиента и запускаемая в виде обыкновенного исполняемого файла на устройстве пользователя. В качестве такого устройства может быть компьютер, коммуникатор или смартфон.**

МОДУЛИ ДЛЯ СОЗДАНИЯ DESKTOP ПРИЛОЖЕНИЙ

- **Tkinter**
- **PyQt**
- **PySide 2**
- **Kivy**
- **wxPython**
- **Flexx**
- **CEF Python**
- **И др.**

TKINTER

- Для создания окна приложения надо вызвать метод `Tk()`.
- Также можно прописать заголовок приложения с помощью метода `title` и установить размер/местоположение с помощью метода `geometry`.
- Для вывода созданного приложения имеется метод `mainloop`.

TKINTER BUTTON

- **Элемент кнопки вызывается при помощи конструктора `Button`, при написании конструктора можно указать параметры для видоизменения нашей кнопки.**
- **Стоит помнить, перед запуском приложения кнопку надо сделать видимой, при помощи метода `pack`.**
- **Для того чтобы добавить обработчик событий на кнопку, необходимо сначала создать метод, и его записать в параметр `command` при создании кнопки.**

TKINTER LABEL

- Для написания текста, пометок, названия элементов можно воспользоваться текстовой меткой или же Label.
- Как и у кнопки, у метки есть параметры для изменения отображения.

TKINTER ENTRY

- Для ввода различных значений используется элемент **Entry**.
- Для передачи различных можно воспользоваться компонентами **StringVar**, **IntVar**, **BooleanVar**, **DoubleVar**.

SQL В PYTHON

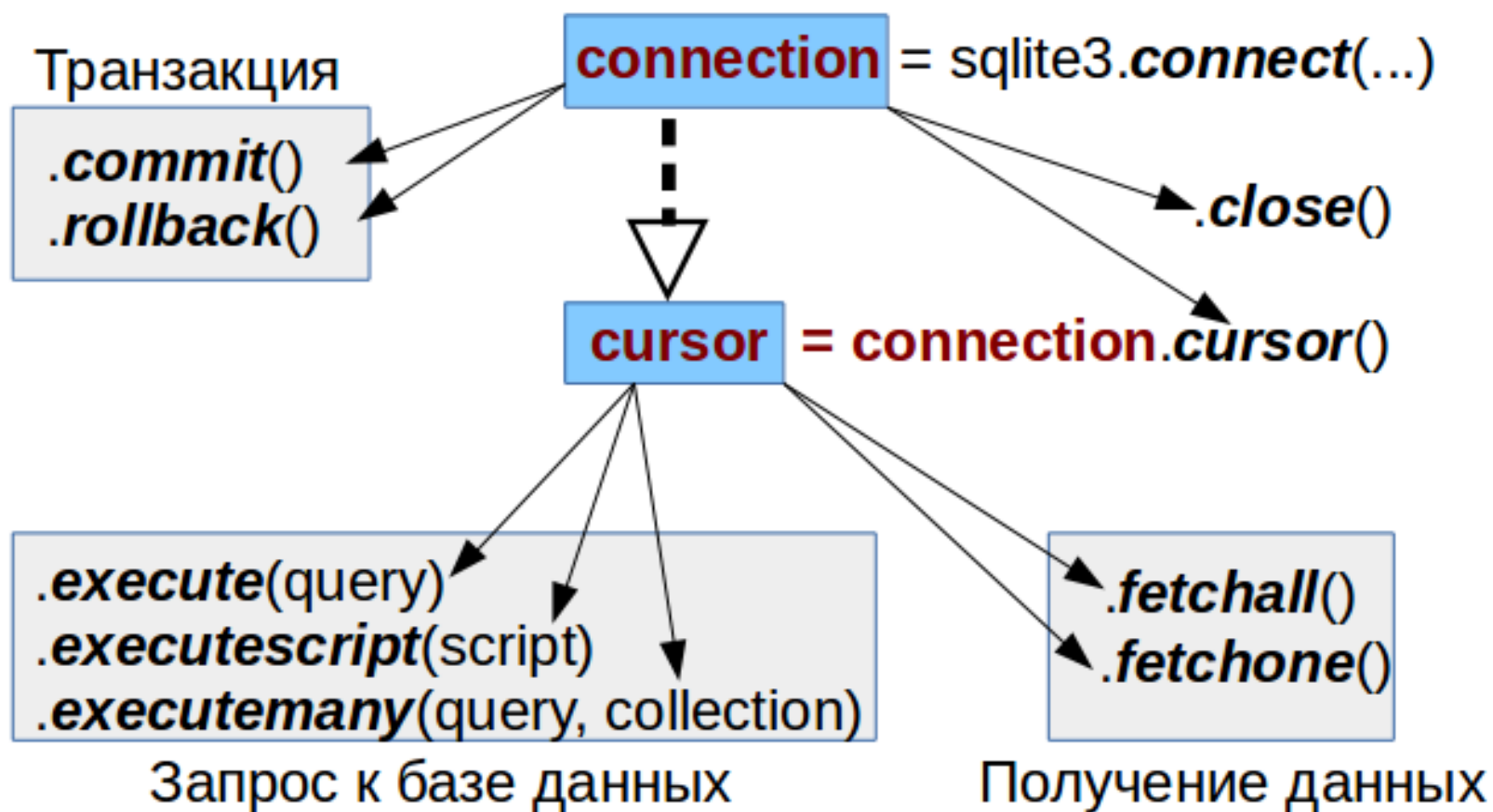
- **Python Database API (DB-API)** определяет стандартный интерфейс для модулей доступа к базе данных Python. Это задокументировано в PEP 249. Почти все модули баз данных Python, такие как `sqlite3`, `psycopg2` и `mysql-python`, соответствуют этому интерфейсу.

МОДУЛИ ДЛЯ РАБОТЫ С SQL

- **SQLite – sqlite3**
- **PostgreSQL – psycopg2**
- **MySQL – mysql.connector**
- **ODBC – pyodbc**

ФОРМА DB-API

Python DB-API методы



ПРОЦЕСС РАБОТЫ С DB-API

- Для создания соединения необходимо вызвать метод `connect()`, который создает сессию подключения к базе данных и вернет объект `connection`, для взаимодействия с БД.
- В свою очередь `connection` реализует инкапсуляцию сеанса БД.
- `cursor()` конструктор возвращающий объект `cursor`. `Cursor` используется для выполнения запросов и скриптов.

ПРОЦЕСС РАБОТЫ С DB-API

- С помощью методов `cursor` можно как отправлять запросы, так и получать различные данные.
- `execute()`, `executemany()` - Отправление запросов в БД.
- `fetchone()`, `fetchmany()`, `fetchall()` - Получение данных из БД из выполненного запроса.

ПРОЦЕСС РАБОТЫ С DB-API

- После написанных и выполненных запросов необходимо применить эти самые изменения или откатить, если что-то пошло не так. Как в транзакциях.
- `connection.commit()` метод применения транзакции, сохраняет изменения в БД.
- `connection.rollback()` метод отката транзакции, откатывается в состояние до применения `cursor()`.
- `connection.close()` после всей проделанной работы с БД необходимо, как и с файлами, закрывать запущенный процесс.

GIT

- **Git – это распределенная система управления версиями.**
- **Система управления версиями – программное обеспечение для облегчения работы с изменяющейся информацией. Система управления версиями позволяет хранить несколько версий одного и того же документа, при необходимости возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение, и многое другое.**

РЕПОЗИТОРИЙ GIT

- **Локальный репозиторий Git – это папка, в которой в Git отслеживаются изменения. На компьютере может быть любое количество репозиториев, каждый из которых хранится в собственной папке. Каждый репозиторий Git в системе не зависит, поэтому изменения, сохраненные в одном репозитории Git, не влияют на содержимое другого.**

ПЕРВОНАЧАЛЬНАЯ НАСТРОЙКА

- После установки git следует его настроить.
- `git config --global user.email "(Ваша почта)"`
- `git config --global user.name "(Ваше имя)"`
- В ином случае, при сохранении состояния git будет требовать вписать почту и имя. Это необходимо в качестве информации, кто данный коммит сделал и как с ним связаться.

СОЗДАНИЕ РЕПОЗИТОРИЯ

- Для инициализации репозитория по пути в котором находимся прописываем следующую команду:
- `git init`

РАБОТА С РЕПОЗИТОРИЕМ

- Для подготовки файлов необходимо прописать команду
- `git add .`
- Для проверки состояния имеется команда
- `git status`
- Для того чтобы сохранить состояние нашего проекта используется команда
- `git commit -m "<Название коммита>"`
- Ключ `-m(message)` нужен для того прописать название коммита, в котором пишется основные изменения.

ОТОБРАЖЕНИЕ ИНФОРМАЦИИ

- Для просмотра истории коммитов необходимо прописать команду
- `git log`
- Также имеются ключи:
 - `--pretty=` - формат отображения информации
 - `online` - компактное отображение
 - `short`
 - `medium`
 - `full`
 - `fuller`
 - `format: %h => %s (%ad) - %an` - кастомный формат
 - `--max-count=`
 - `--all`
 - `--author=`
 - `--graph`

ТЕГ

- Теги представляют из себя ссылки на ключевые моменты в момент создания проекта.
- `git tag v1` – Создание тега
- `git checkout v1` – Перемещение на коммит тега
- `git tag` – просмотр всех тегов
- `git log --all` – Вывод информации о коммитах с тегом

ОТМЕНА ДЕЙСТВИЙ

- Отмена локальных изменений (до индексации)
- `git checkout <файл>`

- Отмена проиндексированных изменений (перед коммитом)
- `git reset HEAD <файл>`

- Отмена коммитов
- `git revert HEAD`

ОТМЕНА ДЕЙСТВИЙ

- Сброс коммитов к предыдущему тегу
- `git reset --hard v1`

- Удаление сброшенного тега
- `git tag -d v1`

ВЕТКИ

- Для отображения существующих веток
 - `git branch`
- Для создания новой ветки
 - `git branch <Название ветки>`
- Для переключения на ветку используется `checkout`
 - `git checkout <Название ветки>`

ВЕТКИ

- Также можно одновременно создать ветку и на неё переключится
- `git checkout -b <Название ветки>`
- Для того чтобы объединить ветки
- `git merge <Название ветки> / git merge <Название ветки> -m <сообщение>`
- Следует помнить, что слияние идет из указанного в команде ветки в ветку в которой находитесь.
- Если нужно выйти из состояния слияния
- `git merge --abort`

ПЕРЕБАЗИРОВАНИЕ

- Есть иной способ перенести все изменения из одной ветки в другую, с помощью перебазирования.
- `git rebase master` (из другой ветки)

ПРОСМОТР РАЗНИЦЫ КОММИТОВ

- Можно посмотреть в чем разница между несколькими версиями.
- `git diff <коммит>`
- `git diff <коммит> <коммит>`
- `git diff <ветка> <ветка>`

ПРОСМОТР ХЭШЕЙ

- **git cat-file -t <hash> - -t(type) тип хэша (commit, branch и т.д.)**
- **git cat-file -p <hash> - -p(rint) вывод информации о хэше**
- **git cat-file -p <treehash> - вывод дерева**

.GITIGNORE

- **Git рассматривает каждый файл в вашей рабочей копии как файл одного из трех нижеуказанных типов.**
 - **Отслеживаемый файл – файл, который был предварительно проиндексирован или зафиксирован в коммите.**
 - **Неотслеживаемый файл – файл, который не был проиндексирован или зафиксирован в коммите.**
 - **Игнорируемый файл – файл, явным образом помеченный для Git как файл, который необходимо игнорировать.**
-
- **Игнорируемые файлы – это, как правило, артефакты сборки и файлы, генерируемые машиной из исходных файлов в вашем репозитории, либо файлы, которые по какой-либо иной причине не должны попадать в коммиты.**

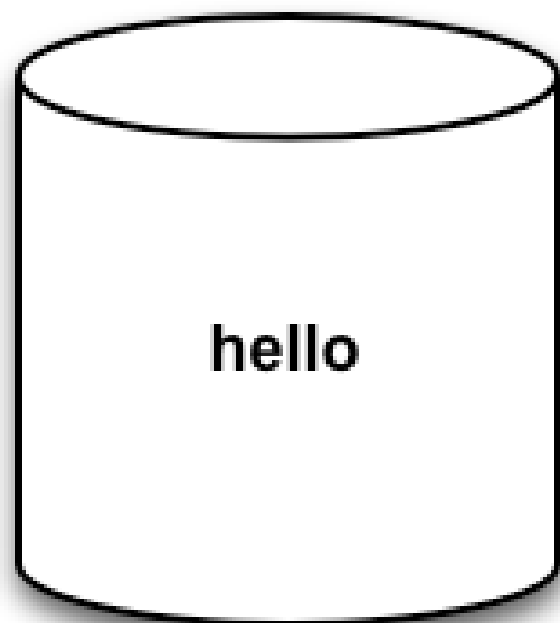
УДАЛЕННЫЙ РЕПОЗИТОРИЙ

- **Удаленный репозиторий git.**
- **Если локальный репозиторий Git — это папка, то удаленный репозиторий та же папка, только, понятное дело, не на локальном компьютере, а на удаленном сервере или иной машине. Такое размещение позволяет обращаться к git сразу нескольким людям. Именно поэтому при первоначальной настройке необходимо вписать имя и почту. Для того чтобы другой человек смог к вам обратиться и с вами связаться.**

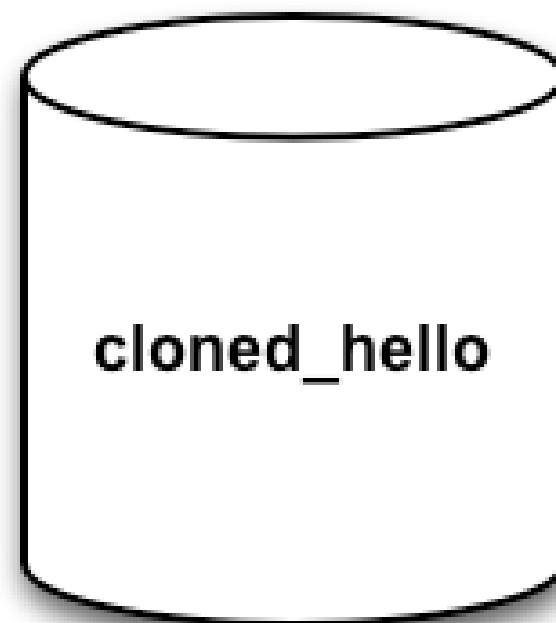
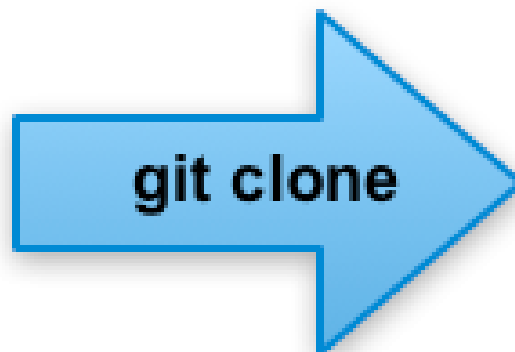
СЕРВИСЫ GIT

- **GitHub**
- **GitLab**
- **Bitbucket**
- **Gogs**

НЕСКОЛЬКО РЕПОЗИТОРИЕВ



*Оригинальный
репозиторий*



*Копия оригинального
репозитория*

ЗАГРУЗКА УДАЛЕННОГО РЕПОЗИТОРИЯ

- Для загрузки проекта с удаленного репозитория используется команда
- `git clone <ссылка>`
- Для того чтобы каждый раз не прописывать полностью путь до репозитория можно воспользоваться установкой псевдонима.
- `git remote add origin <ссылка>`
- В этом случае слово «origin» будет играть роль псевдонима при взаимодействии с удалённым репозиторием. В наименование origin записывается сама ссылка репозитория.

ОТПРАВКА И ЗАГРУЗКА ДАННЫХ

- Для отправки коммитов в удаленный репозиторий
 - `git push origin`
- Для загрузки изменений из удаленного репозитория
 - `git fetch origin`
 - `git merge origin/master` – слияние изменений из удаленного репозитория в локальную ветку мастер
- `git pull` - `git fetch` и `git merge` в одном лице. Загружает изменения из удаленного репозитория и переносит в локальную ветку

ЛОКАЛЬНЫЕ И УДАЛЕННЫЕ ВЕТКИ

- **git branch** – вывод локальных ветвей
- **git branch -a** – вывод всех ветвей
- **git branch --track <ветка> origin/<ветка>** – установка слежения локальной ветки с удаленной