Всем привет!
Сегодня у нас будет с большей части практический вебинар по гиту и автотестам, мы будем много повторять пройденный материал, чтобы лучше закрепилось, поэтому если вы в своем сознании настолько преисполнились что вам все понятно и скучно, не обессудьте, для вас могу сделать отдельный мини вебинар про ветки в гите и как с ними жить
В целом рекомендую всем вместе со мной делать - для этого вам нужен компуктер, аккаунт на гитхабе и настроенная пара ssh ключей. Если у вас нет аккаунта в гитхабе и там не настроены ssh - сделает это сейчас пока будет небольшая вводная и косучек теории, чтобы вы потом согли присоедениться к практике.
По регламентам все как обычно у нас - хотите спросить - пишите в чат и Стас будет мониторить либо тяните руку. По дефолту все замьючены пожалуйста)
После каждого небольшого блока я буду останавливаться для ответов на эти вопросы.

1. **Предыстория GIT**

Если нам командой нужно работать над одним файлом нам нужна какая-то система, чтобы мы могли работать вместе не мешая друг другу - не удалять и не перезаписывать куски.
Зачем вообще система версионного контроля:
- уметь создавать бэкап (резервная копия)
- синхронизация работы нескольких разработчиков (хоть 2, хоть 50)
- отмена изменений
- отслеживание изменений и авторов
- возможность осздания изолированных “мест” для экспериментов - напрмиер проверить новую библиототеку не показывая заказчику или команде

Почему GIT
9 из 10 разработчиков используют git
- время операции в git быстрее чем в svn
- гарантия целосности данных - при сохранении гит делает снепшот файла - по сути хеширует его
- можно делать ветки! (указатели) и за счет этого есть много разных стратегий управения проектом, механизмов работы команды

Создание Github repo
- Зарегистрированы и авторизованы в https://github.com/
- Нажимаем + -> New repository
- Вводим имя - webinar autotests 13-10-23
- Включить чекбокс для ридми - таким образом создасться свежий комит
- Create
Показать первый комит из гитхаба

Теперь клонируем репозиторий себе на локальную машину:
Можно сделать через https можно через ssh - у меня уже настроена пара ключей ssh, я буду делать через ssh
- Открыть terminal
- Перейти в директорию в которую я хочу склонировать проект - cd Downloads
- Выполняем команду git clone <путь>
- Проверяем в нашей папке что появился проект - cd <webinar autotests 13-10-23>

если вы первый раз клонируете себе проект то может появится вопрос разрешения установки связи с github - там вводим yes / y

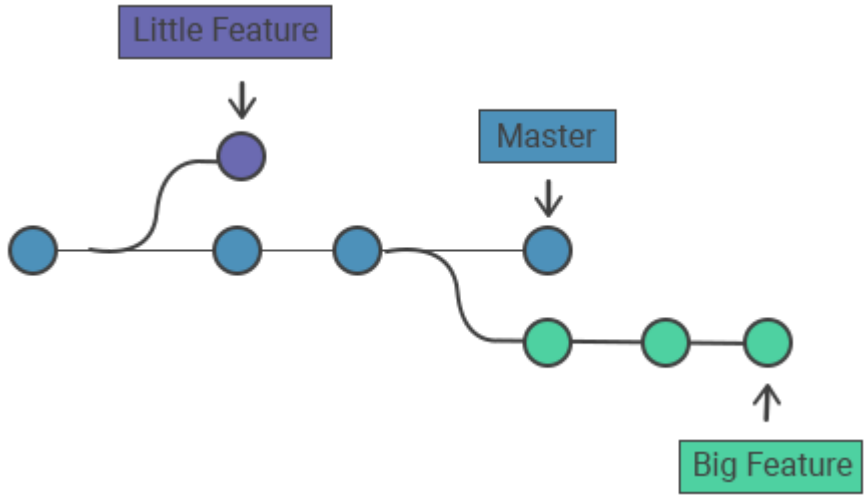
Локальный файл в удаленный репозиторий:
- создаем файл в папке проекта - echo This is some text > myfile.txt
теперь у нас получилось что на локальном компьютере в проекте есть файл, которого нет в гитхабе
- вызываем git status
нам подсказываем что у нас есть недобавленный файл и если мы хотим добавить его в комит нам нужно выполнить git add
- выполняем git add . — точка значит что мы добавляем все в комит, можно писать конретный файл
- git status
- git commit -m “first commit with txt file”
видим что комит создан
посмотрим историю комитов с помощью команды
- git log
дальше нам нужно синхронизировать локальный и удаленный репозиторий с помощью команды
- git push
-открываем гитхаб, обновляем страницу и убеждаемся что у нас повился наш локлаьный текстовый файл

Удаленный файл в локальный репозиторий:
Дальше представим ситуацию что мы работаем вместе с кем-то над проектом и он внес изменения в удаленный репозиторий, которые анм могут пригодиться, допустим написал какую-то нужную нам функцию, с которой мы могли бы завершить свой кусочек работы. Для этого нам эти измнения нужно подтянуть к себе на локальный компьютер.
Мы такую ситуацию смоделируем так - создадим комит в веб версии гитхаба и потом это подтянем к себе локлально:
- в гитхабе есть опция edit file выполняем ее с нашим файлом
- добавляем текст
- комитим изменения в веб интерфейсе
- видим в истории 3 комита
- смотрим что происходит локально - git log - там пока только два комита
- есть в гите команда git fetch которая синхронизирует два репозитория, но она не устанавливает изменения в файлах для этого есть команда git merge
мы будем сейчас использовать команду git pull которая как раз комбинирует в себе эти две команды
дословно как раз pull тянуть, а push толкать
- выполняем git pull
- выполняем git log
- проверяем наш файл

Вот в общем-то самые **базовые операции** мы с вами рассмотрели
- git clone
- git status
- git add
- git commit
- git push
- git pull
- git log

Ветки:

Если вы обратили внимание мы с вами все эти изменения выполняли в одной ветке с названием main
Коротко вам расскажу, но уже показывать не буду
По сути гит это граф, где каждый комит знает своего родителя, и гит позволяет создавать ответвления как в дереве. то есть ветка - это у нескольких коммитов есть один общий предок
единственное в дереве ветки не могу врасти обратно в ствол, а в гите могут.
Зачем ветки нужны - писать фичи не ломая стабильную версию, не мешая другим в команде



Дополнительно:
вот тут вам поиграться https://learngitbranching.js.org
если вам не нравится работать в терминале есть Git Gui & gitk где все действия кнопочками + там карсиво отображается как раз это самое ветвление и наглядно видно какой комит откуда выходит

Автотесты на API
Итак, переходим к написанию автотестов
Тесты будем писать для api магазина питомцев https://petstore.swagger.io/#/
что-то из того что я вам буду показывать вам на прошлом вебинаре уже показал Стас, может быть будет что-то новое
Главное нам с вами сегодня научиться понимать откуда что берется и как оно с друг другом связано, потому что до этого вы многое в тренажере копировали вставляли и так мне кажется сложнее понять логику
- Открываем Pycharm
- Можно создать новый проект с удобным вам названием, я открою вот то что мы с вами создали пока работали с гитом и буду работать тут
- дальше нам нужно установить pytest — пишем в терминале пайчарма pip3 install pytest
- создаем файл - у нас сервис это petstore там файл и назовем (Слева название проекта - контекстное меню New - petstore.py)
- дальше нам нужно импортировать библиотеку запросов, чтобы мы собственно могли с нашими зарпосами работать - пишем в файле impost requests - появляется красная лапомчка, которая нам говорит от том что у нас он еще не установлен и мы нажимаем install
мы начнем с **создания нового животного в магазине** - то есть это post метод /pet и нам его нужно как-то в нашем проекте вызвать
- пишем

```
def create_new_pet():
```

```
    return requests.post()
```
наш метод будет возвращать какой-то результат, его мы будем брать из requests
дальше нам нужно добавить атрибуты вызова в post, первый атрибут это собственно адрес, можно напрямую, но лучше передвать его через переменную в другом файле
для этого создаем python файл configutation
и в нем создаем наш `BASE_URL = "https://petstore.swagger.io/v2"`
наш бейс урл у нас прямо на сайте и прописан (показать)
зачем это делать - вот вы видите у нас в урле прям зашита версия v2 - представьте у нас есть сони тестов и во всех мы адрес прописывали напрямую - нам тогда в каждом из этой сотни тестов нужно проходить и менять v2 на v3 например
а если мы действуем через config файл нам тогда один раз нужно поменять и все ок
смотрите мы с вами сегодня будем работать только с блоком pet - поэтому мы могли бы даже и добавить наш /pet внутрь базового урла, потому что как видите тут все действия выполняются внутри пути /pet но разными методами
но для наглядности я этого делать не буду + вы потом сможете дальше красио дописать тесты если хотите на store и user
поэтому добавляем путь /pet в переменную
```
PET_API_PATH = "/pet"
```
окей, возвращаемся в наш основной файл и используем там наш созданный адрес через переменные
```
def create_new_pet():
    return requests.post(configuration.BASE_URL + configuration.PET_API_PATH)
```
видим что нам автоматически импортировался наш файл configuration
вот этот плюс между нашими переменными пути позволяет нам складывать две строки. - это называется конкатенация - то есть сложение двух строк в одну
дальше в наш метод нужно добавить тело
тут точно также, можно добавить его сразу же в нашу функцию
а можно создать отдельный файл с тестовыми данными - все по тем же причинам + такая запись у нас будет короче потому что это тело мы будем сегодня еще несколько раз использовать
создаем файл data.py
в нем создаем PET_BODY = {}
в скобки вставляем значение
id как видим мы задаем сами, фото и тэги нам не нужны мы из оставим пустыми
получается
```
PET_BODY = {
        "id": 345,
        "category": {
            "id": 1,
            "name": "dogs"
        },
        "name": "Totoshka",
        "photoUrls": [],
        "tags": [],
        "status": "available"
}
```
возвращаемся к нашему запросу и передаем туда наше созданное тело

добавляем import data
формат данным у нас json
получается:

```
import requests
import configuration
import data
def create_new_pet():
    return requests.post(configuration.BASE_URL + configuration.PET_API_PATH,
json=data.PET_BODY)
```

вот по сути мы с вами добавляем нового питомца в магазин и теперь нам нужно написать тест
для этого создаем файл test_adding_new_pet
проверим что у нас успешно собственно добавляется новый питомец и добавляем функцию

```
def test_success_adding_pet():
```

в ней создаем переменную нашего нового питомца, в которой мы будем использовать ответ, то есть то что вернулось в результет выполнения нашей функции из petstore

```
new_pet = petstore.create_new_pet()
```

чтобы это сработало нам в тест нужно импортировать petstore
мы можем проверить что статус код у нас 200
для этого пишем наше любимое assert

```
assert new_pet.status_code == 200
```

пробуем запустить через терминал - пишем pytest
Второй тест
Далее давайте убдимся, что в ответе кличка животного сопадает с тем, которое мы передаем
При этом мы ведь не всегда будем передавать именно наш подготовленный json, нам нужно убдеиться независимо от того что внутри того что мы передаем оно совпадает с ответом
то есть мы не можем написать тест проветь проверь что имя равно Тотошка
Нет, вообще можем конечно, но смысла в таком автотесте мало, нам ведь важео чтобы при добавлении новых и разных питомцев они создавались корректно
Для того чтобы это проверить нам нужно создать функцию, которая будем изменять наше тело запроса
Создаем новый файл - helper.py
Внутри создаем функцию

```
def modify_new_pet_body():
```

Далее чтобы внутри функции данные тела запроса изменить для начала их нужно скопировать
для этого перед функцией добавим импорт наших данных

```
import data
```

а внутри функции добавим

```
body = data.PET_BODY.copy()
```

тут мы с вами используем метод copy который копирует нам все данные pet_body в переменную body мы это делаем для того, чтобы менять не оригинальные данные, а внутри переменной, на случай если мы где-то накосячим чтобы не испортить оригинал который может использоваться в других тестах
дальше мы собственно обращаемся к атрибуту имя в нашей переменной body

```
body["name"] =
```

добавляем в нашу функциб агрумент который мы будем передавать, от есть как раз наше имя которые мы можем каждый раз менять

```
def modify_new_pet_body(name):
```

и собственно этот наш аргумент и будет нашим именем в body
ну и соотвественно функция наша должна что-то возвращать
в нашем случае нам нужно чтобы она вернуло все тело запроса с измененным именем
получается:

```
import data
def modify_new_pet_body(name):
    body = data.PET_BODY.copy()
    body["name"] = name

    return body
```

возвращаемся к нашему тесту
и называем его

```
def test_success_adding_pet_with_custom_name():
```

копируем в наш тест из теста выше результат нашего запроса

```
new_pet = petstore.create_new_pet()
```

но теперь мы не хотим использовать тело запроса напрямую из данных, а также оно нам нужно динамичным
поэтому мы возвращаемся в файл petstore
копируем нашу функцию
переименовываем верхнюю в create_pet_with_body
аналогично функции в хелпере мы тут передаем как аругмент наше body - то что мы используем внутри функции, но хотим чтобы его можно было менять
и получается вот так

```
def create_new_pet_with_body(body):
    return requests.post(configuration.BASE_URL + configuration.PET_API_PATH,
json=body)
```

в функции пониже можно не осталвять такую длинную строку а переиспользовать результат нашей функции выше только тут мы в качестве данных передаем наше оригинальное body
получается:

```
import requests
import configuration
import data
def create_new_pet_with_body(body):
    return requests.post(configuration.BASE_URL + configuration.PET_API_PATH,
json=body)
def create_new_pet():
    return create_new_pet_with_body(data.PET_BODY)
```

возвращаемся к нашему тесту
и теперь мы там должны использовать ту функцию что мы создали с измененяемым телом запроса

```
def test_success_adding_pet_with_custom_name():
    new_pet = petstore.create_new_pet_with_body()
```

но пока собственно наше нове тело запроса мы еще не создали
исправляем:

создаем внутри функции переменную new_pet_body =
которая будет равна результату выполнения нашей функциив хелпере
если помните она нам как раз измененное тело и возвращает
и внутри скобок мы как раз указываем наше name измененное
это наш аргумент функции, помните?
а так, чтобы это все сработало не забываем импорты, импортируем наш хелпер

```
new_pet_body = helper.modify_new_pet_body("Bim")
```

теперь нам нужно передать это новое тело запроса в нашу переменную new_pet
вот так выглядит наш новый тест, в котором мы изменили данные

```
def test_success_adding_pet_with_custom_name():
    new_pet_body = helper.modify_new_pet_body("Bim")
    new_pet = petstore.create_new_pet_with_body(new_pet_body)
```

дальше нам нужно собственно проверить соотвествие переданным и полученным данным
для этого добавляем assert
для начала такой же как и в первом тесте мы просто проверим что успешно запрос дошел

```
assert new_pet.status_code == 200
```

для проверки имени мы расшифровываем наш результат / ответ и оттуда получаем имя

```
assert new_pet.json()["name"]
```

и проверяем что оно равно нашему значению Бим

```
assert new_pet.json()["name"] == "Bim"
```

запускаем тесты pytest -v
и убеждаемся что они проходят
можем намеренно завалить тест, чтобы убедиться тчо мы во всех праивльных местах передаем нужные значения
передадим другое имя в функцию, а в асерте оставим то же иу бедимся тчо тест у нас тогда провалится
итого мы с вами уже написали 2 теста, вопросы?
дальше сделаем немного интресней и напишем параметризованный тест
например для статуса
параметризованный тест это тест, который позволяет нам проверять разные значения при это не копируя много раз один и тот же тест
на примере статуса мы хотим проверить что у нас создастся запись о питомце с со статусов available, unavailable, численной форме или с русскими буквами
помечаем что наш тест параметризованный и импортируем вначале файла pytest

```
@pytest.mark.parametrize()
```

создаем новый тест с именем

```
def test_success_adding_pet_with_different_category():
```

дальше указываем наш параметр

```
def test_success_adding_pet_with_different_category(statusvalue):
```

передаем название нашего аргумента

```
@pytest.mark.parametrize(“statusvalue)
```

передаем данные, у нас будет несколько перечислений, значит это будет массив
указываем пайтест парам, сколько будет таких параметров, столько и будет запусков этого теста

```
@pytest.mark.parametrize(“statusvalue”, [
    pytest.param()
])
```

внутри указываем наше значение, напрмиер “Dogs” и дальше через запятую указываем id - это у нас по сути название нашего подтеста именно этим параметром

```
@pytest.mark.parametrize("statusvalue", [
    pytest.param(
        "Available", id="Testing status value with string"
    ),
    pytest.param(
        "23", id="Testing status with number value"
    ),
    pytest.param(
        "Недоступно", id="Testing status with russian alphabet"
    )
])
```

Так, дальше нам собственно нужно передать тело запроса с этими разными значениями, для этого возвращаемся в хелпер и напишем универсальную функцию, которая сможет менять любой параметр, а не только одно имя
создаем функцию, которая принимает параметр и его значение как аргументы

```
def modify_pet_body(key,value):
```

дальше можем скопировать тело функции из нашей прошлой модифай, только теперь мы не будем указывать конрктеные атрибуты запроса, а заменим их на нашу пару ключ значение

```
def modify_pet_body(key,value):
    body = data.PET_BODY.copy()
    body[key] = value
    return body
```

возвращаемся к нашему тесту
аналогично прошлым тестам, подготавливаем наше тело запроса с помощью новой функции хелпера и кладем его в переменную
передаем туда ключ ?
а значение это наш параметр statusvalue
дальше выполняем запрос с этим новым телом

```
new_pet = petstore.create_new_pet_with_body(new_pet_body)
```

и делаем асерты
наш стандрантый на ответ 200
и на то что статус соотвествует переданному в параметре

```
assert new_pet.status_code == 200
assert new_pet.json()["status"] == statusvalue
```

запускаем проверяем, видим что с каждым парамтером у нас проходит отдельный тест
Дальше к каждому асерту можно добавлять кастомную комментарий ошибку, чтобы было понятнее что пошло не так
добавим такой комент ко второму тесту с именем у нас
и сломаем тест чтобы увидеть наше собщение в результатах прогона

```
assert new_pet.status_code == 201, \
f"Expected 201, but actual is {new_pet.status_code}"
```

и последним давайте сделаем тест на удаление созданной записи
для этого сделаем метод на удаление в нашем файле петстор

у нас тут есть парамтер который меняется внутри пути и это наш id, поэтому мы его передаем как аргумент

```
def delete_pet(id):
```

дальше нам нужно передать запрос с методом делит и адрес, который будет содержать наш id

```
def delete_pet(id):
    return requests.delete(configuration.BASE_URL + configuration.PET_API_PATH + "/" + str(id))
```

зачем добавился str(id) - это потому что id у нас число, а питон не js и строки с числами складывать не умеет, нам нужно привести все к одному формату - строке
переходим к самому тесту

```
def test_success_deleting_added_pet():
```

внутри нам обязательно нужно создать нового питомца, чтобы мы его могли удалить
но нам не нужен весь результат запроса создания, нам нужен только id для его переиспользвоания, поэтому только его мы в переменную и запишем

```
added_pet_id = petstore.create_new_pet().json()["id"]
```

дальше мы вызываем наш метод удаления и записываем результат вызова в переменную

```
deleted_pet = petstore.delete_pet(added_pet_id)
```

по нестареющей классике проверяем код ответа

```
assert deleted_pet.status_code == 200
```

и еще мы тут можем проверить что в параметре мессадж ответа удаления передается id удаленной записи

```
assert deleted_pet.json()["message"] == str(added_pet_id)
```

вуаля!