**2023/12/22 00:00|Лекция. Обзор сторонних библиотек. Часть 2.**

**Обзор сторонних библиотек. Часть 2**

На этом уроке продолжим тему, посвящённую сторонним библиотекам, точнее, основам сторонних библиотек.

На прошлом уроке мы познакомились с консольной версией программы pip, то есть заходили в консоль, печатали здесь необходимые нам программы: pip install, pip uninstall и так далее, скачивали, обновляли, удаляли необходимые для нас пакеты. **PyCharm же пошёл чуть дальше, у него внутри есть окошко.** Оно работает как pip, но с нормальным интерфейсом.

**Где оно находится?** Нажимаем на настройки нашего проекта settings(рис.1) и потом мы должны будем перейти в меню под названием project и название нашего проекта, который сейчас у нас есть: “Project: Модуль сторонней библиотеки”. Нажимаем сюда, раскрываем точнее это меню и нажимаем на “Python Interpreter”(рис.2), то есть мы выбираем этот python интерпретатор. Здесь у нас уже перечислены все установленные здесь библиотеки. Они установлены именно в этом виртуальном окружении. Написан также, кстати, путь виртуального окружения, в нашем случае это Python(рис.3). Он по умолчанию скачан, то есть вот это оригинальный Python.

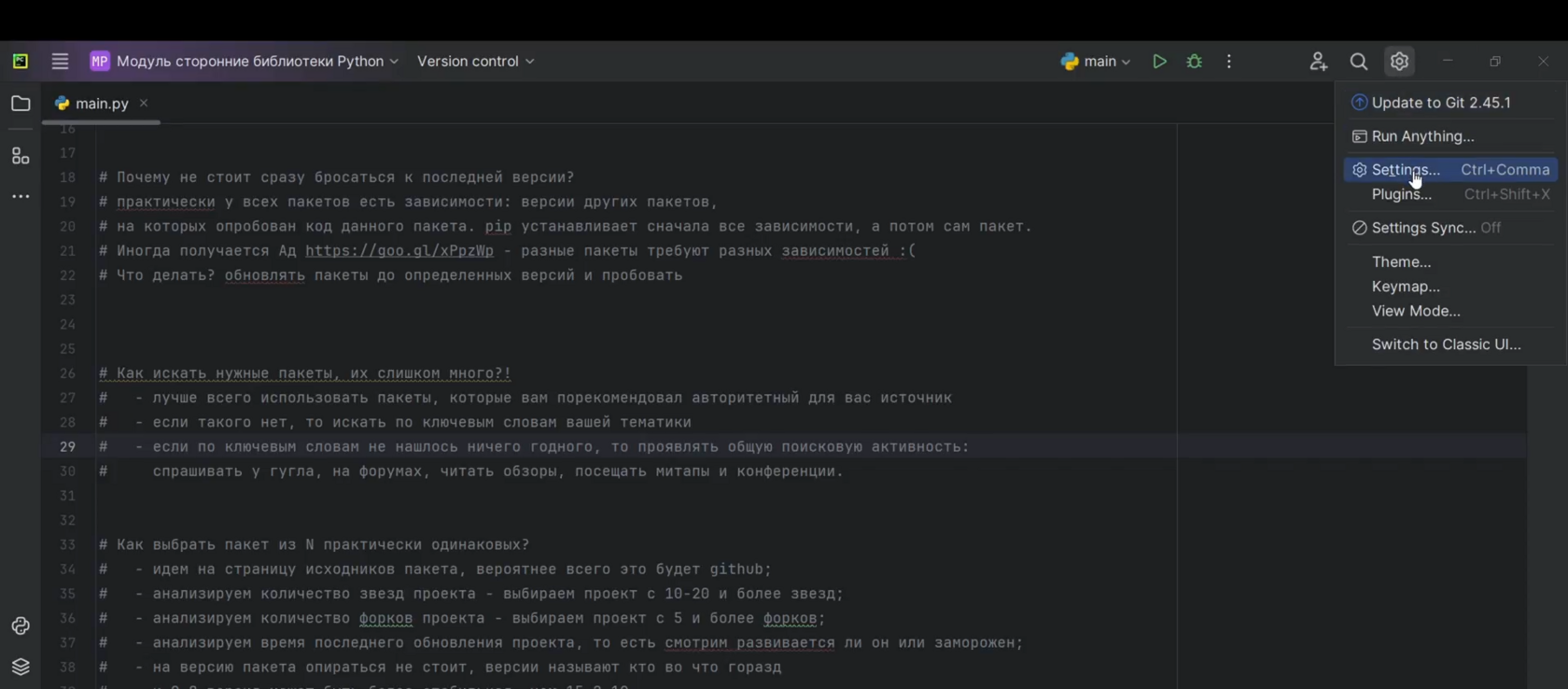


Рис.1

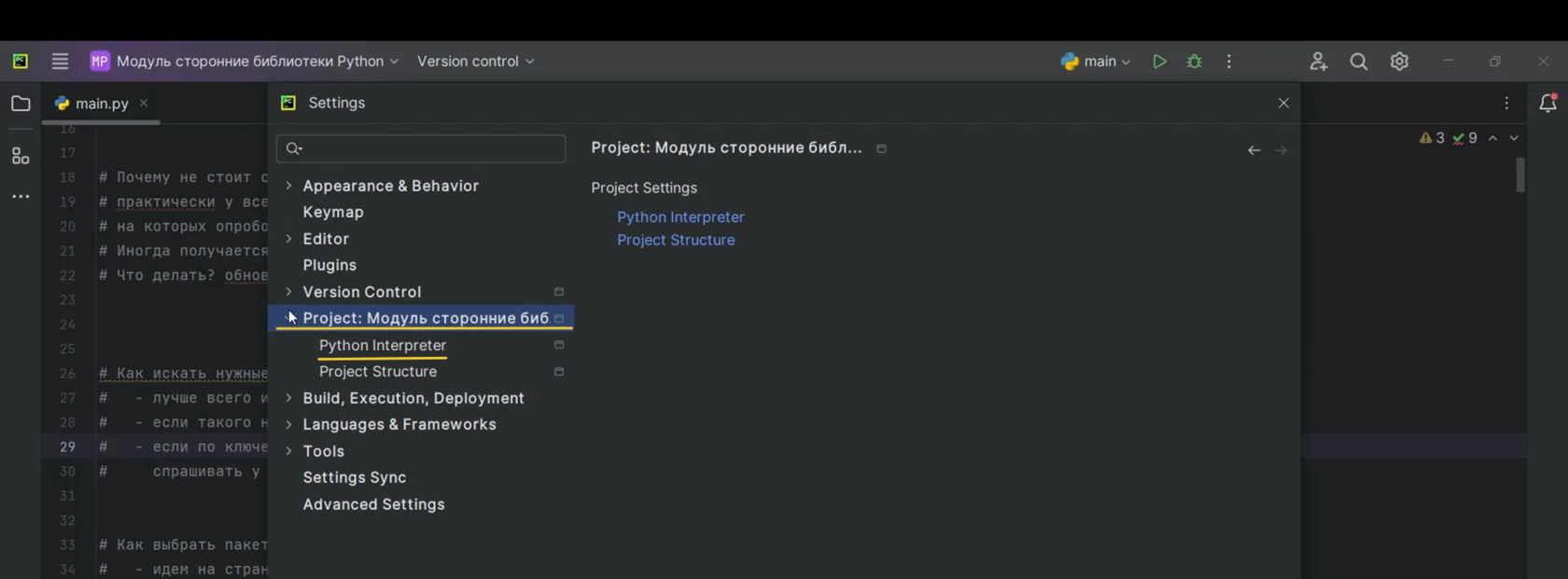


Рис.2

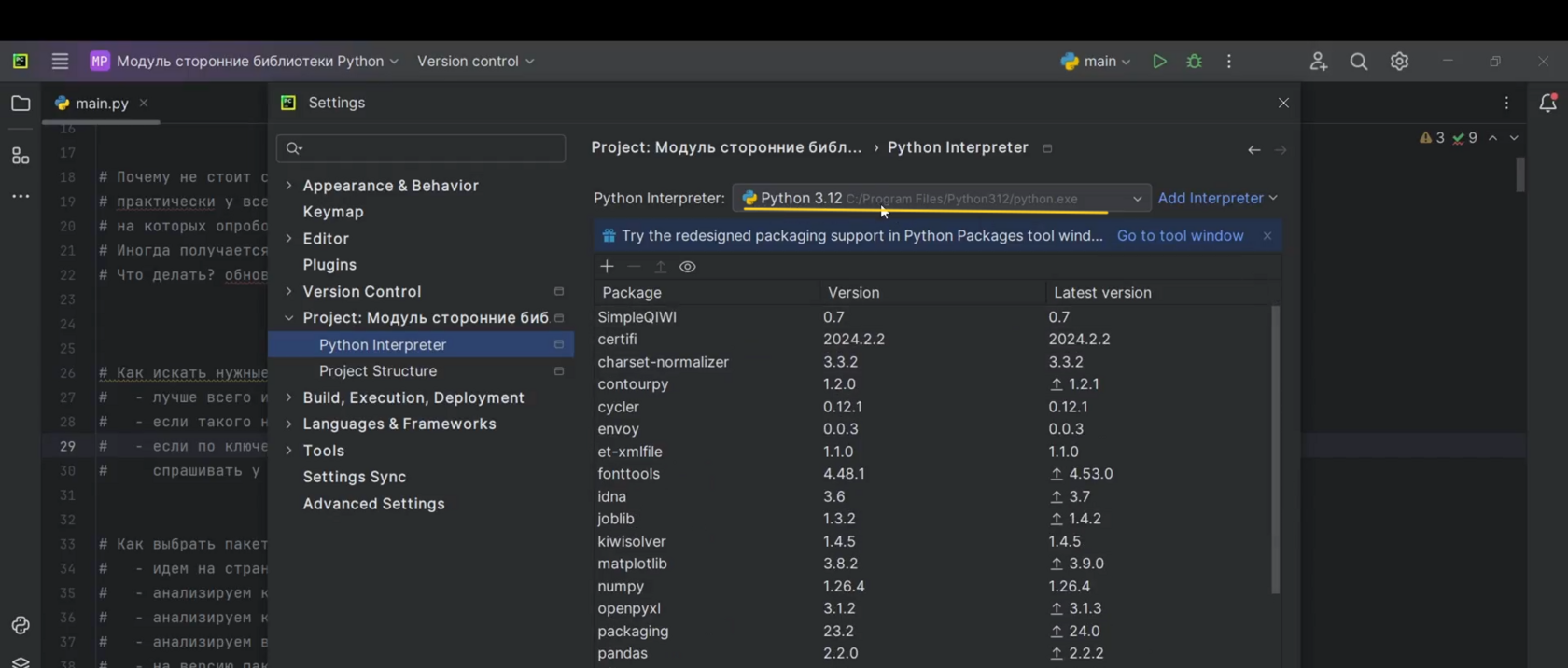


Рис.3

Дальше перечислены все библиотеки. Вы можете видеть их список. Перечислены нынешние версии этих библиотек, а также перечислены следующие, точнее, последние версии этих библиотек. Причём можем заметить, что там, где у нас есть стрелочка вверх(рис.4), это значит, что у нас наша нынешняя версия устарела по сравнению с последней вышедшей, просто PyCharm берет и подгружает сервера PyPi.org.

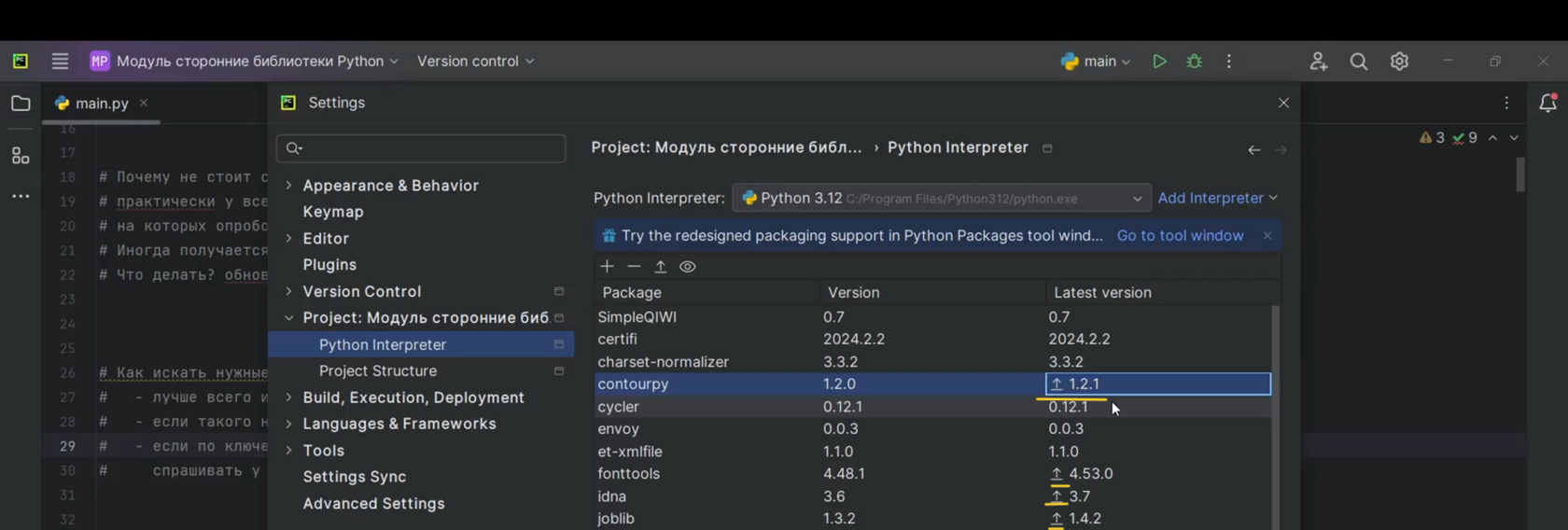


Рис.4

**Что можно делать в этом меню?**Для начала можем просто брать и скачивать необходимые для нас пакеты, нажав на кнопку плюсик(рис.5). То есть здесь уже ищем необходимое нам название пакета. Допустим, это будет 2mp3, который качали на прошлом уроке. Вот здесь можем выбрать это окошко и просто нажать Install Package(рис.6). Когда установится наш пакет, высветится надпись “Package ‘2mp3’ installed successfully”(рис.7), а именно то, что пакет успешно установлен. Когда закрываем, высвечивается 2mp3 библиотека, причём с последней версией.

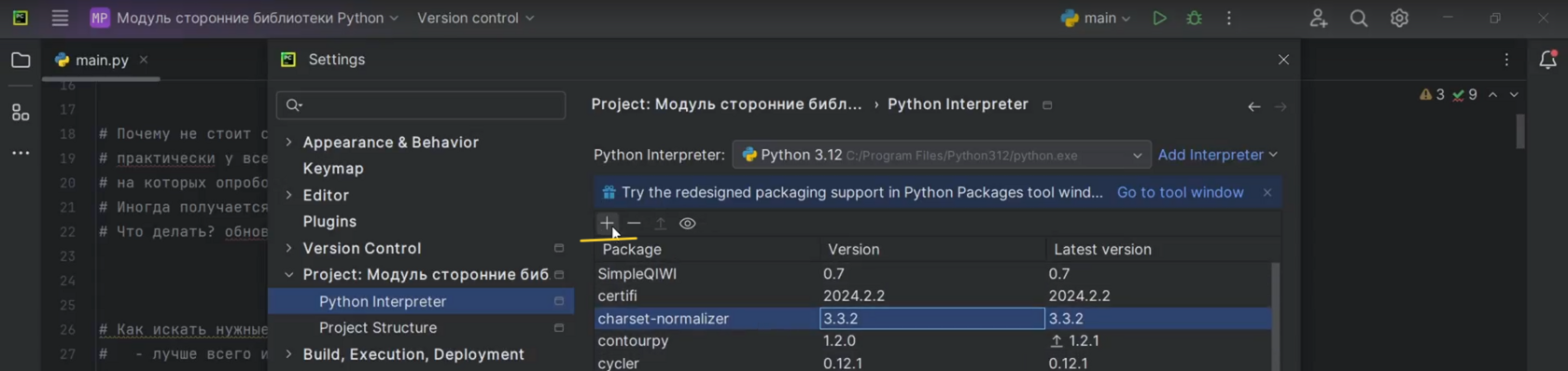


Рис.5

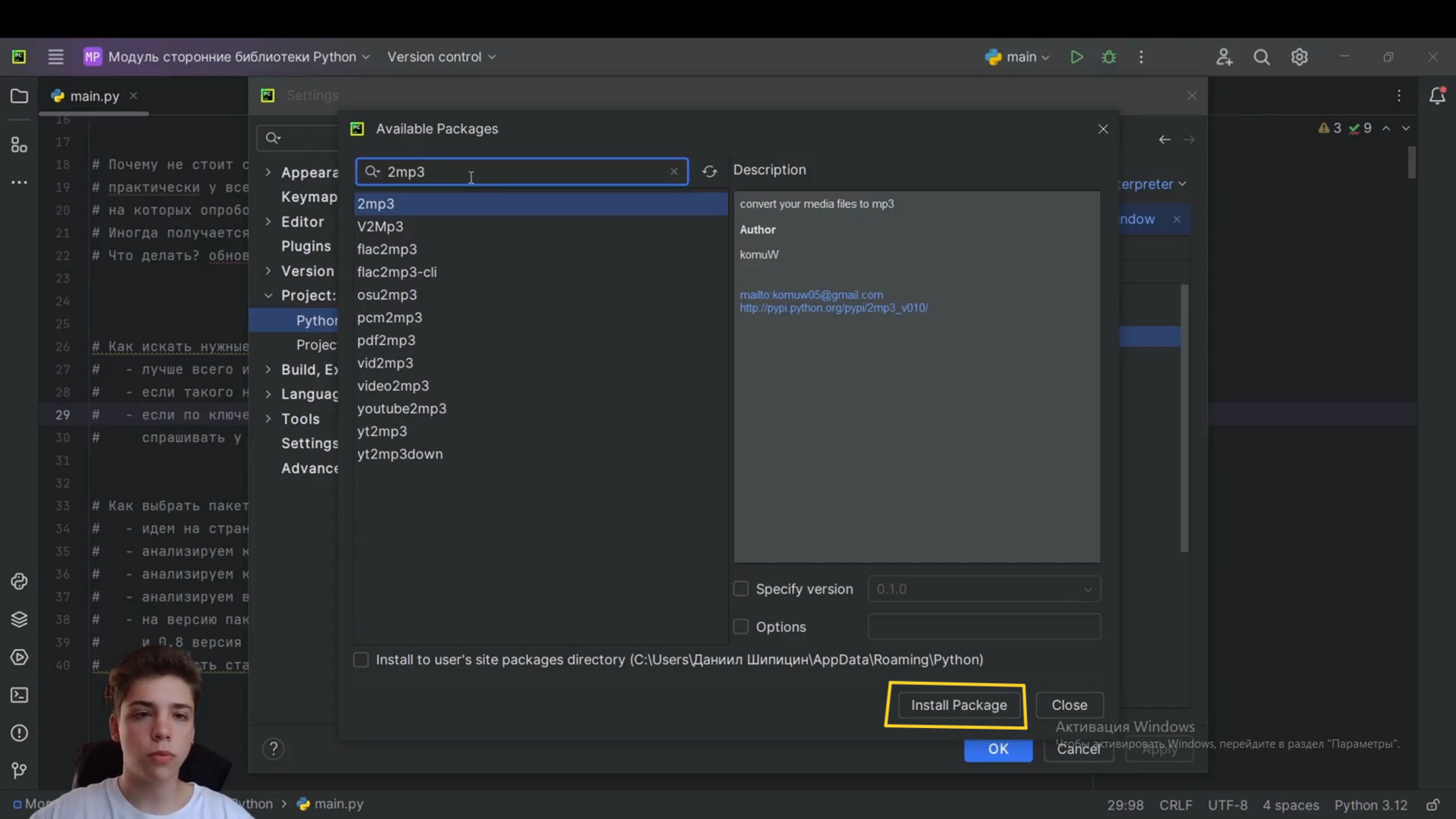


Рис.6



Рис.7

Если же хотим обновить какой-либо пакет, то можем его выбрать и нажать на соответствующую кнопку “upgrade”(рис.8). Cразу же автоматически подгружается новая версия этого пакета. Помимо этого высвечивается, что наше окружение не учитывает вот это обновление, поэтому нам нужно загрузить, то есть обновить, по сути, наше виртуальное окружение. Выбираем наш Python интерпретатор, либо можем просто проигнорировать это уведомление(рис.9).

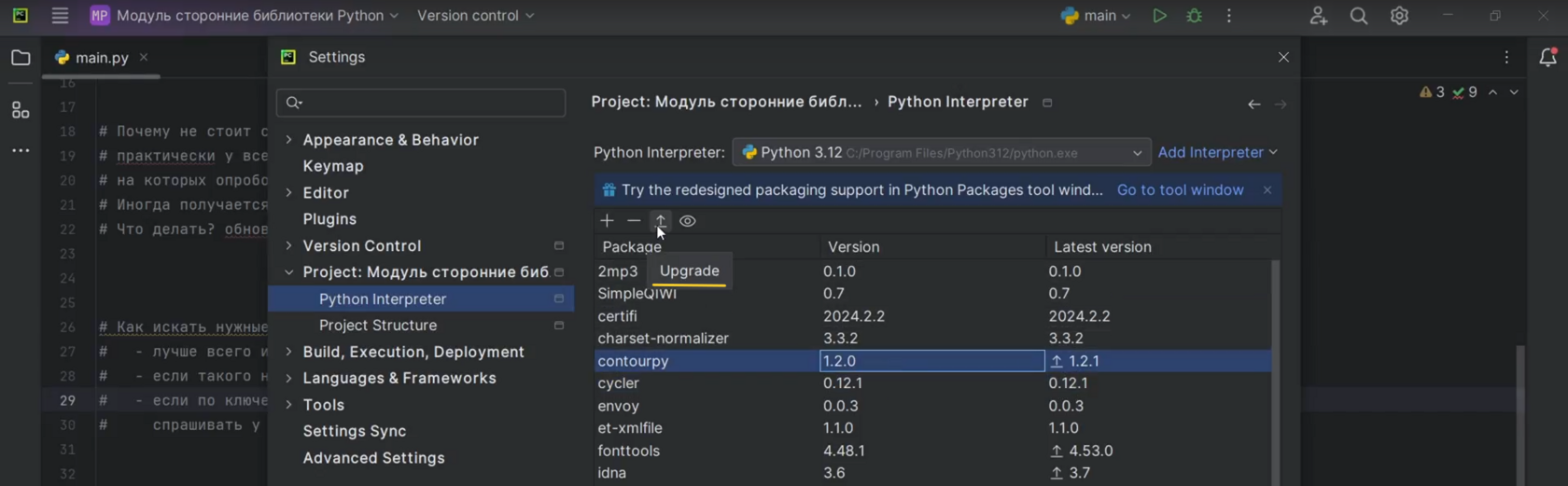


Рис.8

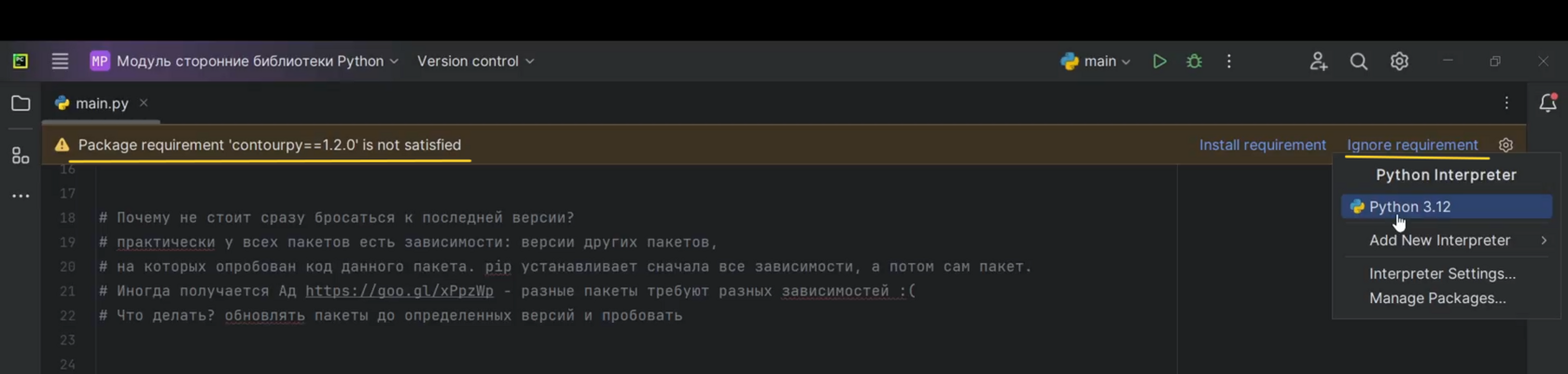


Рис.9

Но не торопитесь сразу обновлять все свои пакеты до последней версии. Зачастую есть мнение, что последняя версия, значит, самая рабочая, но это не всегда так. Последние версии зачастую даже бывают не такие стабильные, как их прошлые. Помимо этого, в последних версиях могут удалиться те функции, которые вы использовали до этого, особенно если это старый проект. Поэтому нужно следить за тем, какой у вас проект. Нужно понимать, на каких версиях вы работаете и не торопиться обновлять, но это не единственная причина.

**Почему же не стоит сразу бросаться к последней версии?**Потому что практически у всех пакетов есть зависимости, версии других пакетов. Мы уже говорили на прошлом уроке, что, когда качаем необходимый для нас пакет, он, может быть, был построен на других пакетах. Сначала pip подгружает наши первородные пакеты, а потом уже подгружает тот пакет, который хотели скачать, но из-за этого получается ад. Он называется ад зависимостей.

**Что такое ад зависимостей(рис.10)?**

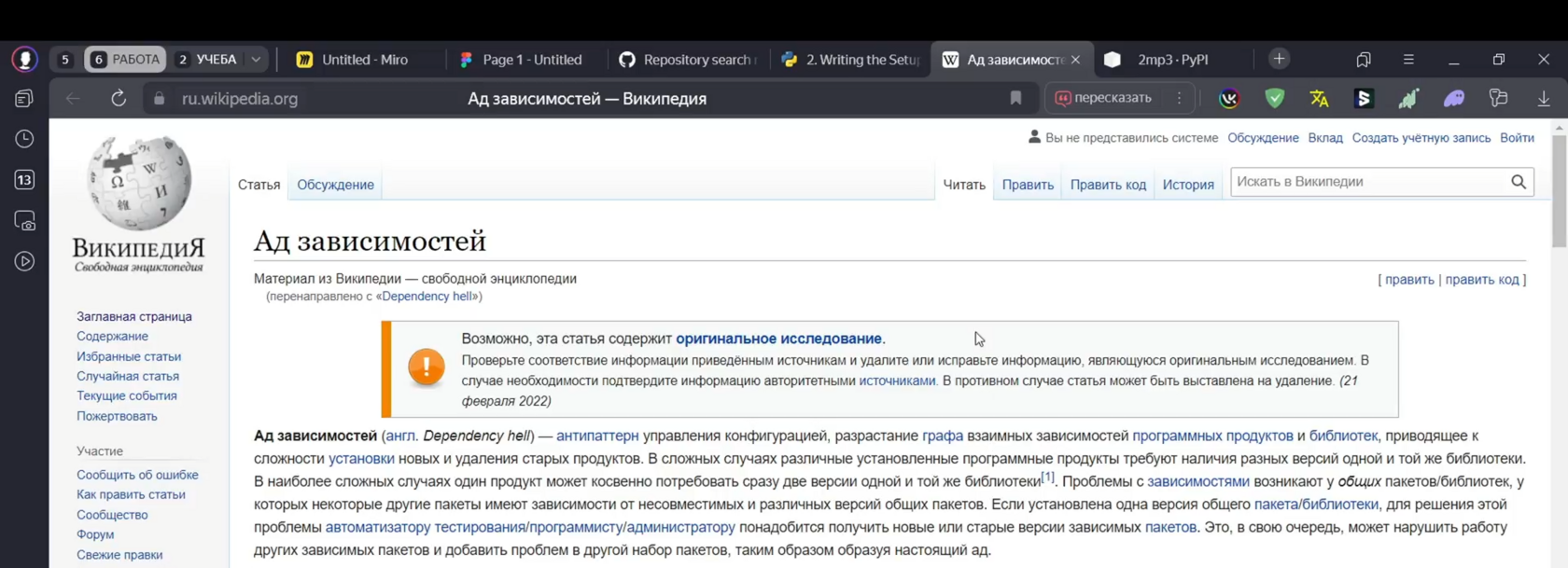


Рис.10

Если вкратце, то это просто в действительности ад, потому что есть, например, пакет. Этот пакет подгрузил зависимость одной библиотеки на такой-то версии, а есть пакет, который подгрузил точно такой же пакет, но с другой версией, то есть ему нужно работать с этим же пакетом, но с другой версией. В итоге у нас получается несовместимость версии. Пакет “а” работает с пакетом “b” на версии 1.0, пакет “с” работает с пакетом “b” на версии 2.0.

**Что с этим делать?** Какого-то прям автоматического решения нет. Мы либо просто берём и смотрим вдруг у нас, оказывается, что библиотека “а” работает уже с пакетом “b” на версии 2.0 и обновляем все. Это нужно самому наследовать.

Другое же решение — это создать своё виртуальное окружение. В нем будем с нуля подкачивать все необходимые нам пакеты. Это будет голое, свободное от библиотек окружение, куда подкачаем необходимые для нас пакеты и, скорее всего, они не будут конфликтовать между собой.

**Осталось последние 2 вопроса разобрать:**

1. Как искать нужные для нас пакеты? То есть подходящие для нашего проекта или вообще, которые в принципе, можем использовать, но мы о них не знаем.
2. Как выбрать пакет из множества практически одинаковых? На прошлом уроке мы заходили на страничку GitHub и у нас высвечивалось очень много различных библиотек. Там было больше 50 страниц на разных языках, но с одним и тем же домыслом: это был пакет, связанный с mp3 конвертацией.

**Вернёмся к 1 вопросу: как искать нужные пакеты?**

Пока что вы начинающий разработчик, вы будете потихонечку только все тыкать. Вам, скорее всего, просто будут говорить, либо прям вести под ручку, по необходимым пакетам. Пока вы на начальном уровне, вы будете использовать те пакеты, которые знают действительно все, там даже подсказывать нечего. Но все же, если вы не знаете, что использовать, спросите у начальника, спросите у друга или ещё у кого-то, то есть спросите у человека, являющийся для вас авторитетом или на каком-то достоверном сайте. Что вы можете использовать вам, скорее всего, там уже подскажут.

Если же такого нет, то лезем в интернет и ищем по ключевым словам. Если не можем найти по ключевым словам, то проявляем общую поисковую активность, то есть заходим в “Google”, “Яндекс”, ищем на сторонних ресурсах, спрашиваем на разных форумах. Можно просто бесконечно таких вопросов найти и прочитать, что вообще там нам рассказывают. Можно ещё точно также посещать митапы (MeetApp - неформальная встреча специалистов одной профессии или сферы, на которой обсуждают заранее заявленную тему) и конференции.

С какой-то необходимой для нас библиотекой, пакетом определились. Как же выбрать именно ту, которая действительно будет работать? Очень много проектов загружается на сам GitHub. Мы можем просто перейти на страничку GitHub и посмотреть, сколько звёзд или сколько форков стоит у какого-то пакета.

Опять же по поводу mp3 библиотеки, то есть не будем от неё отставать. Заходим сперва на сайт PyPi.org, потом ищем какую-то любую интересующую нас библиотеку. Допустим, вот эта(рис.11). Очень маленькое описание, непонятно что тут на китайском написано. Давайте перейдём на ссылку на проект, то есть на homepage(рис.12). Тут можем обратить внимание на то, сколько звёзд стоит у данного проекта, правый верхний угол, а также сколько стоит форков у этого проекта(рис.13).



Рис.11

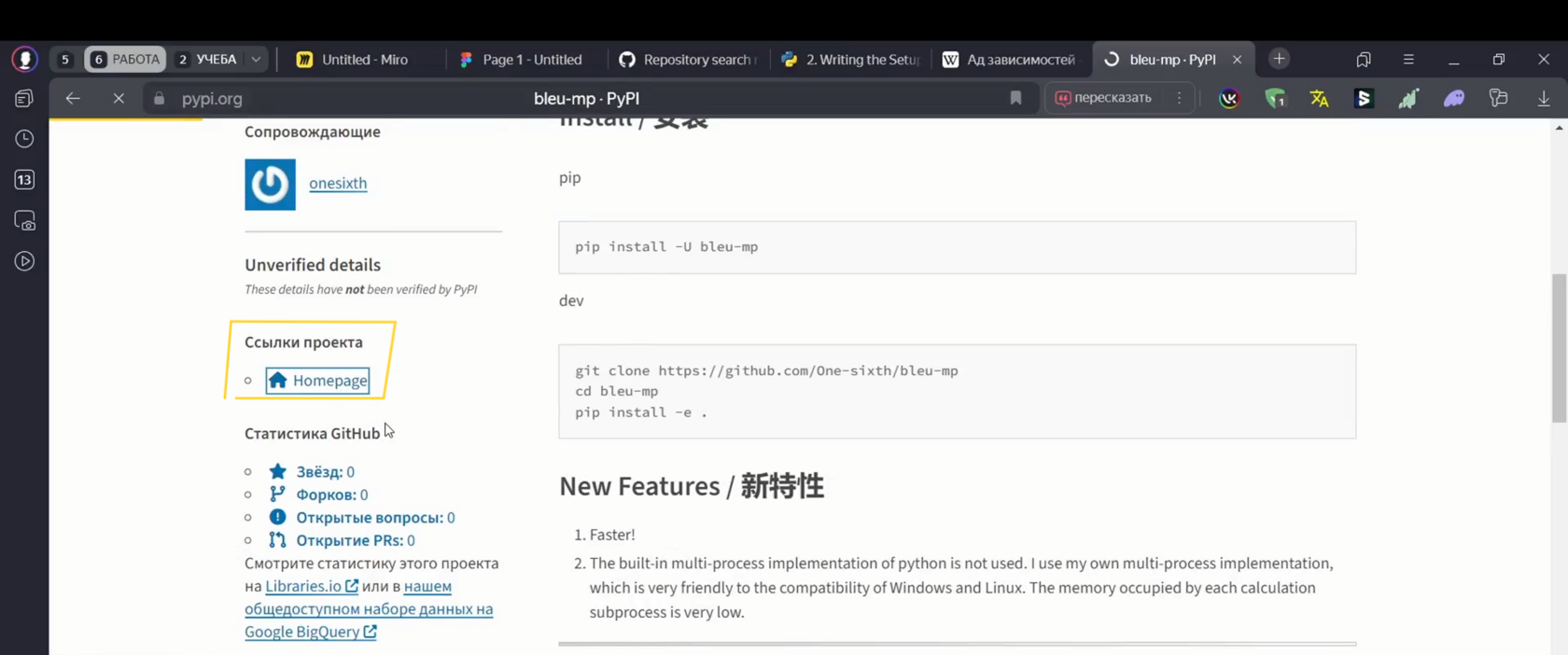


Рис.12

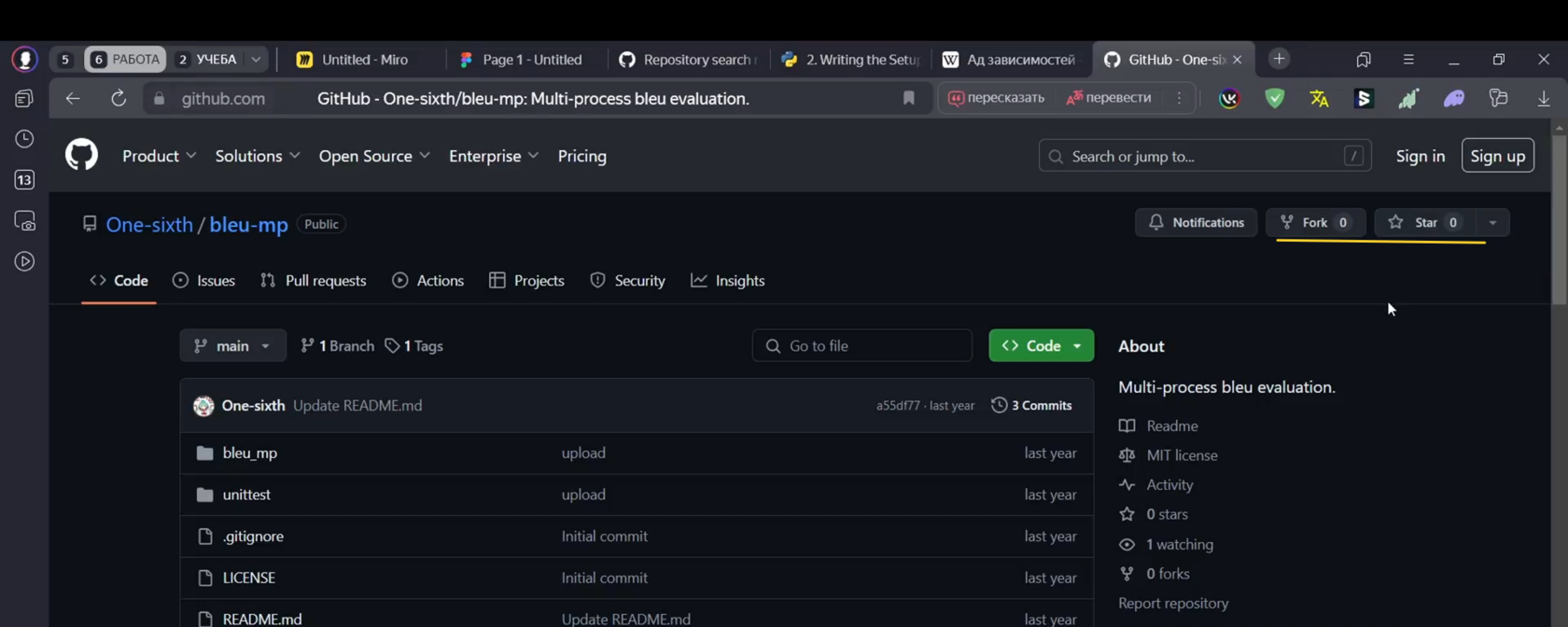


Рис.13

**Что по отдельности каждый из них значит?**

Звезду ставят только те программисты или пользователи, которым понравился этот проект, то есть, если никому этот проект не понравился, если нет ни одной звёздочки, то лучше не скачивать именно эту библиотеку. Желательно качайте, когда есть минимум 20 звёздочек.

**Что такое форки? Форк (fork)**— это когда программисту понравилась данная библиотека, данный пакет, но он её скачал и решил ещё видоизменить. То есть он скопировал себе эту библиотеку в репозиторий, начал как-то видоизменять код, подстраивать под себя. В общем, это тоже показатель качества. Их тоже смотрите от 20 штук. Скорее всего, это тогда будет достаточно хороший проект.

Для примера давайте посмотрим на проект Django(рис.14). Он у нас немножечко описан. Возьмём ссылку на GitHub проекта Django и посмотрим, сколько у него звёздочек. У него звёздочек 77000, это один из самых популярных фреймворков на Python(рис.15). Мы это все прекрасно знаем. У него очень много форков — это показатель реально качества. Если уже у какого-то пакета от 1000 звёздочек, 100%, берите его, если он вам действительно нужен, и не пожалеете об этом.

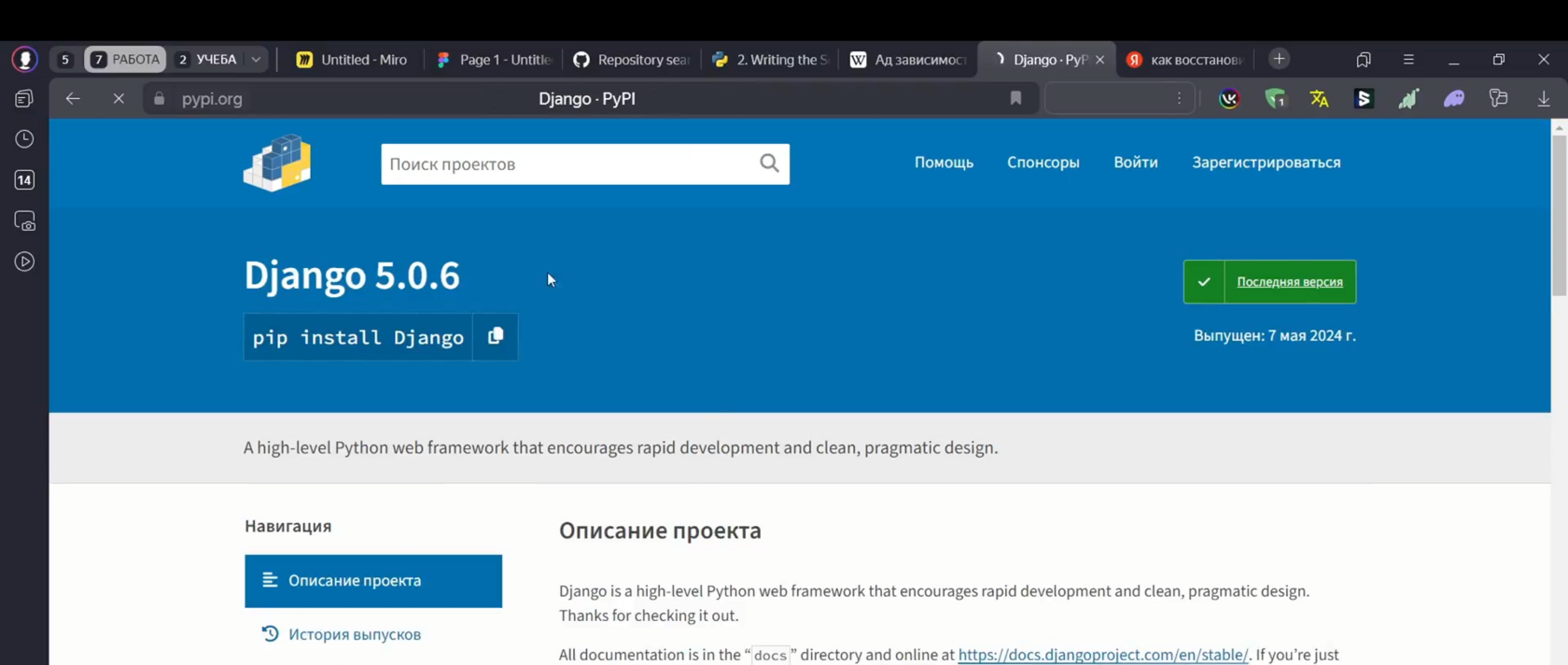


Рис.14

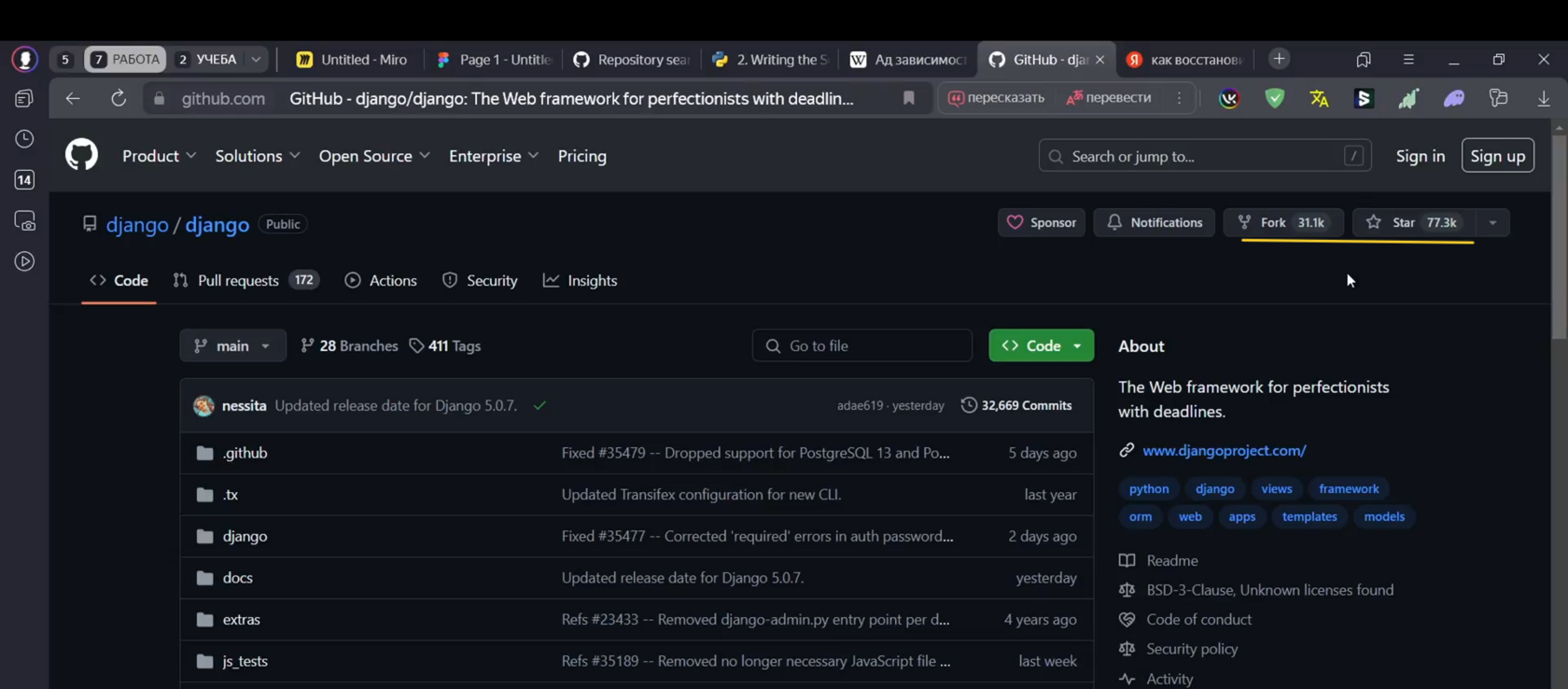


Рис.15

Последнее, что хотелось бы сказать, что не надо особенно смотреть на версию какого-либо проекта, потому что каждый программист на самом деле творческая личность, и эти вот версии он расставляет по-разному. Даже есть те программисты, которые никогда не поставят себе версию 1.0.0, потому что они считают, что идеала никогда не достичь, то есть будет всегда вот версия, там 0.9.9.9 и так далее. Есть некоторые программисты ставящие версии как хотят, то есть 15 версия, 16 версии и так далее. Это не обязательно показатель качества. Лучше смотреть на то, когда проект был обновлён в последний раз. То есть, если обновление было недавно, хотя бы до года, то тогда действительно проект нормальный. Если у него много звёздочек и fork на репозитории GitHub, то это хороший проект.

Если же обновление было недавно, но, допустим, это было какое-то ужасное обновление, тоже, скорее всего, не берите последнюю версию этого проекта. Скорее всего, эта версия ещё нестабильная. Хотя, если она выкачена на GitHub, её уже отправили в сервис, и, наверное, она там работает качественно, но лучше все-таки установить более старую версию. Версия 0.8, может быть более стабильная, чем 15.2.10. Это все зависит от программиста.

Конечно же, есть стандарт, и многие программисты придерживаются именно стандарта, но все равно это происходит не всегда. Поэтому, либо слушайте авторитетные источники, они подскажут конкретно с чем нужно работать. Пока вы на этапе обучения в принципе таких вопросов, скорее всего, у вас вообще возникать не будет, чем дальше вы будете обучаться, тем у вас будет больше профессиональных навыков для анализа какого-либо пакета.