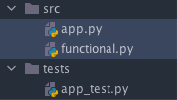
**Введение**

В этой работе можно будет познакомиться с созданием ci пайплайнов, а также с автоматическим запуском тестов, которые мы написали ранее, применением линтеров и тайпчекеров.

Если ваши результаты при некоторых действиях будут отличаться от скриншотов, это нормально, т.к. используемые нами веб фреймворки достаточно сильно отличаются.

**Реструктуризация проекта**

Для более удобной работы изменим структуру проекта:



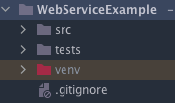
файлы:

1. *src/app.py* - web сервис
2. *src/functional.py* - файл с функцией, решающей поставленную задачу (калькулятор, форматирование текста или числа Фибоначчи)
3. *tests/app\_test.py* - написанные ранее тесты

Тем самым мы отделили файлы нашего проекта от тестов. На самом деле такое разделение крайне условно и очень сильно зависит от практик языка, на котором мы пишем, договоренностей в команде и т.д.

**Важный момент**

Обратите внимание на то, что после тестов, у вас скорее всего появилась папка с названием *.pytest\_cache*. Как следует из названия - это папка, созданная pytest, с кэшированными данными для повышения скорости тестирования. В нашем репозитории на gitlab это явно лишнее (большое количество файлов, изменения в которых нас совсем не интересуют). Это же относится и к виртуальному окружению python (обычно venv, [почитать](https://dvmn.org/encyclopedia/qna/12/chto-takoe-virtualnoe-okruzhenie-zachem-ono-kakie-byvajut-analogi/)). Добавим .gitignore файл. Таким образом наш проект принимает вид:



Для тех, кто еще не сталкивался с данным файлом ([ссылка](https://mbaev.com/posts/chto-takoe-gitignore.html), [ссылка](https://tyapk.ru/blog/post/gitignore#:~:text=%D0%BD%D0%B0%20%D1%87%D1%82%D0%B5%D0%BD%D0%B8%D0%B5-,.,glob%20%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%20%D0%B4%D0%BB%D1%8F%20%D0%B2%D1%8B%D0%B1%D0%BE%D1%80%D0%BA%D0%B8%20%D1%84%D0%B0%D0%B9%D0%BB%D0%BE%D0%B2.))

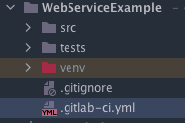
**Чем заполнить данный файл?**

Тут на помощь приходит гугл и сообщество. В [данном репозитории](https://github.com/github/gitignore) есть примеры для огромного количество популярных и не очень языков программирования. В случае python - [.gitignore](https://github.com/github/gitignore/blob/main/Python.gitignore), копируем его содержимое и вставляем в ранее созданный файл. Помимо особенностей языка, различные файлы может генерировать операционная система, например, MacOS создает скрытый .DS\_Store файл, который нам тоже не нужен в нашем репозитории. Добавим в инор файл в зависимости от ОС: [linux](https://github.com/github/gitignore/blob/main/Global/Linux.gitignore), [windows](https://github.com/github/gitignore/blob/main/Global/Windows.gitignore), [macos](https://github.com/github/gitignore/blob/main/Global/macOS.gitignore). Помимо этого можно добавить к игнорированию папки, создаваемые IDE (.idea для продукции Jetbrains, .vscode - Visual Studio Code и т.д)

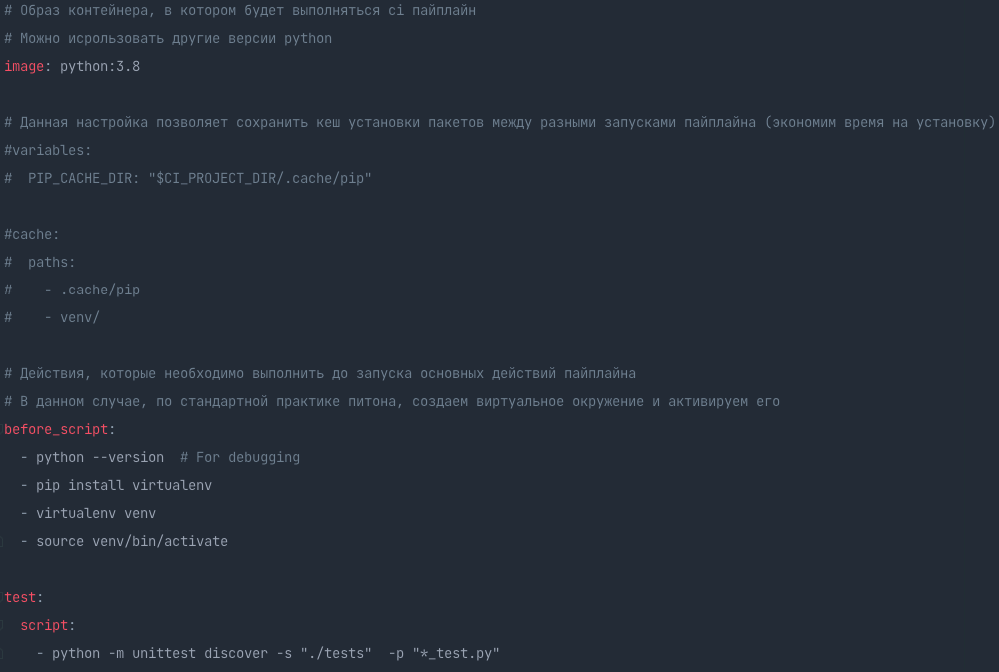
*Делаем коммит и выполняем пуш в свой репозиторий!*

**Создаем файл с пайплайном**

Создаем в корне нашего проекта файл *gitlab-ci.yml*



Содержимое файла



Используемая команда для запуска тестов применима для tornado. В случае использования других фреймворков используйте команду, которую вы выполняли во 2 л.р.

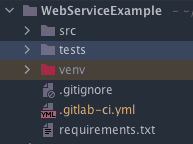
Однако, если мы сейчас запустим пайплайн, все практически мгновенно упадет, т.к. мы не установили зависимости (FastAPI, Tornado и т.д.). Самым простым способом будет захардкодить эти зависимости, как это сделано с модулем virtualenv в ci файле, однако есть более удобный способ - requirements файл.

Выполним в терминале команду:

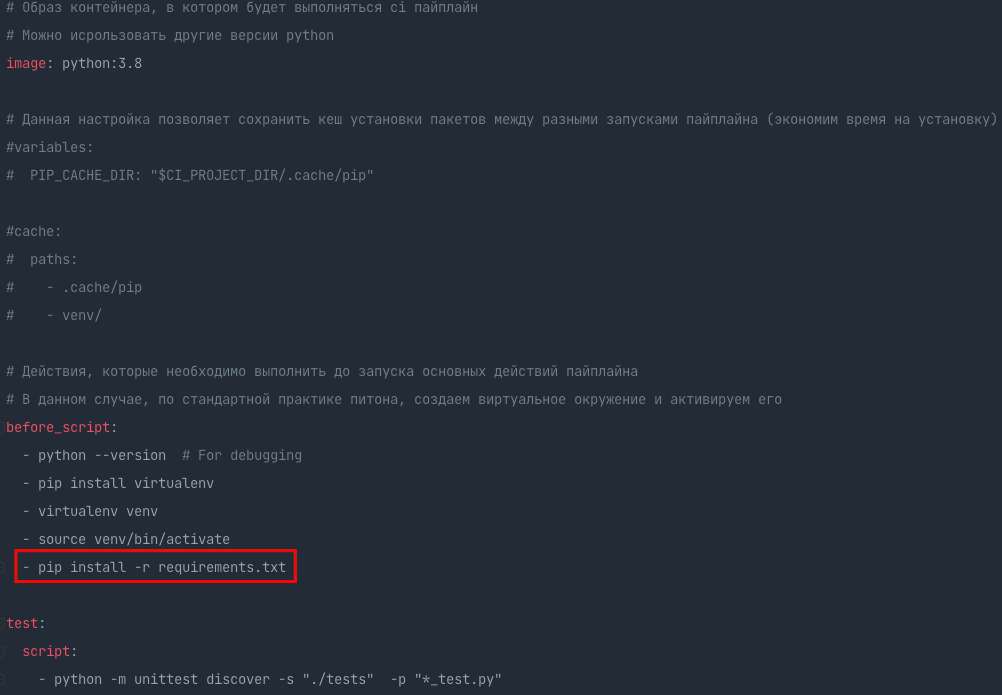
*pip freeze > requirements.txt*

После чего в корне нашего проекта создается одноименный файл, где прописаны наши зависимости с их версиями.

Текущая структура проекта:

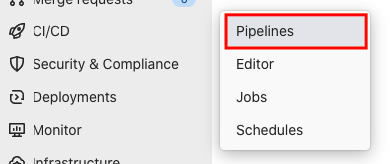


Используем сгенерированный файл в CI:

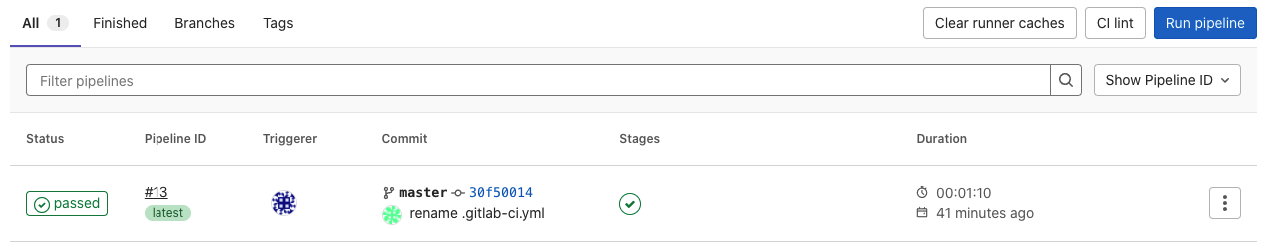


*Делаем коммит и выполняем пуш в свой репозиторий!*

Далее переходим в раздел Pipelines в CI/CD



Если все было сделано верно, вы увидим результат об успешном завершении пайплайна.



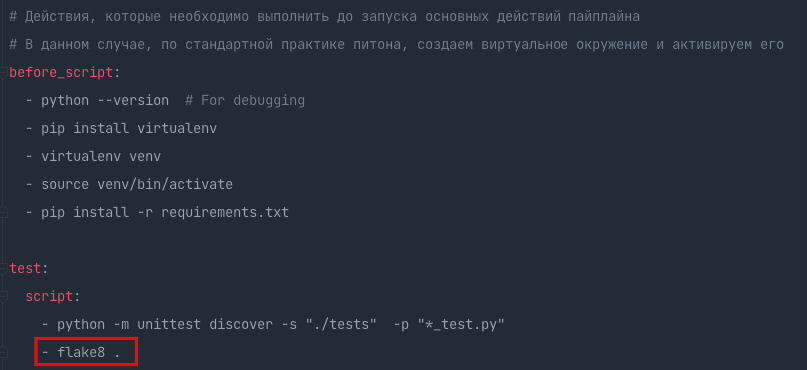
**Код-чекеры**

Кроме использования написанных тестов, часто применяются код-чекеры они же [линтеры](https://thecode.media/linter/). Одним из самых популярных в python является flake8. Мы воспользуемся его базовым функционалом (на самом деле есть много субмодулей, которые позволяют жестко контролировать стиль написания кода).

Установим его с помощью команды:

*pip install flake8*

Далее добавим его в тестирование:

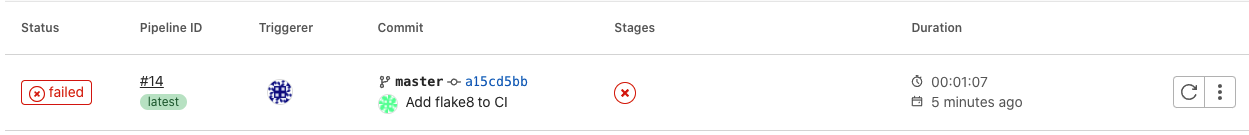


**ВАЖНО!**

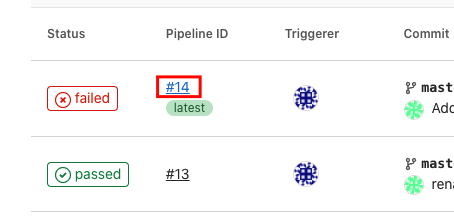
Наш ***список зависимостей изменился,*** **не забудьте опять выполнить команду pip freeze**. Чтобы упростить эту процедуру можно воспользоваться [poetry](https://python-poetry.org/), который делает это автоматически.

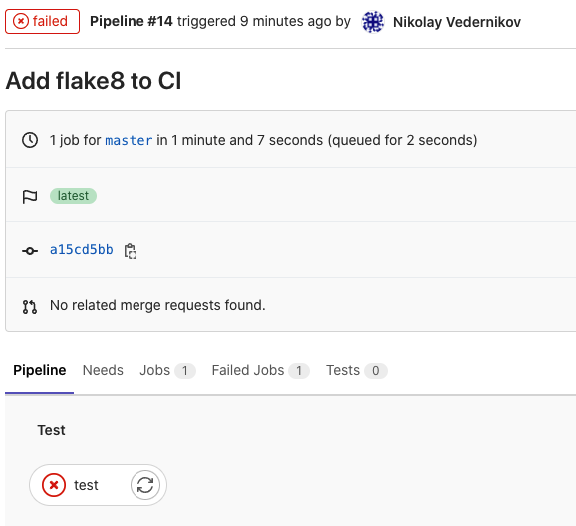
*Делаем коммит и выполняем пуш в свой репозиторий!*

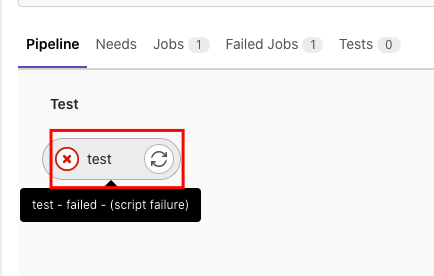
В этот раз пайплайн завершился с ошибкой.



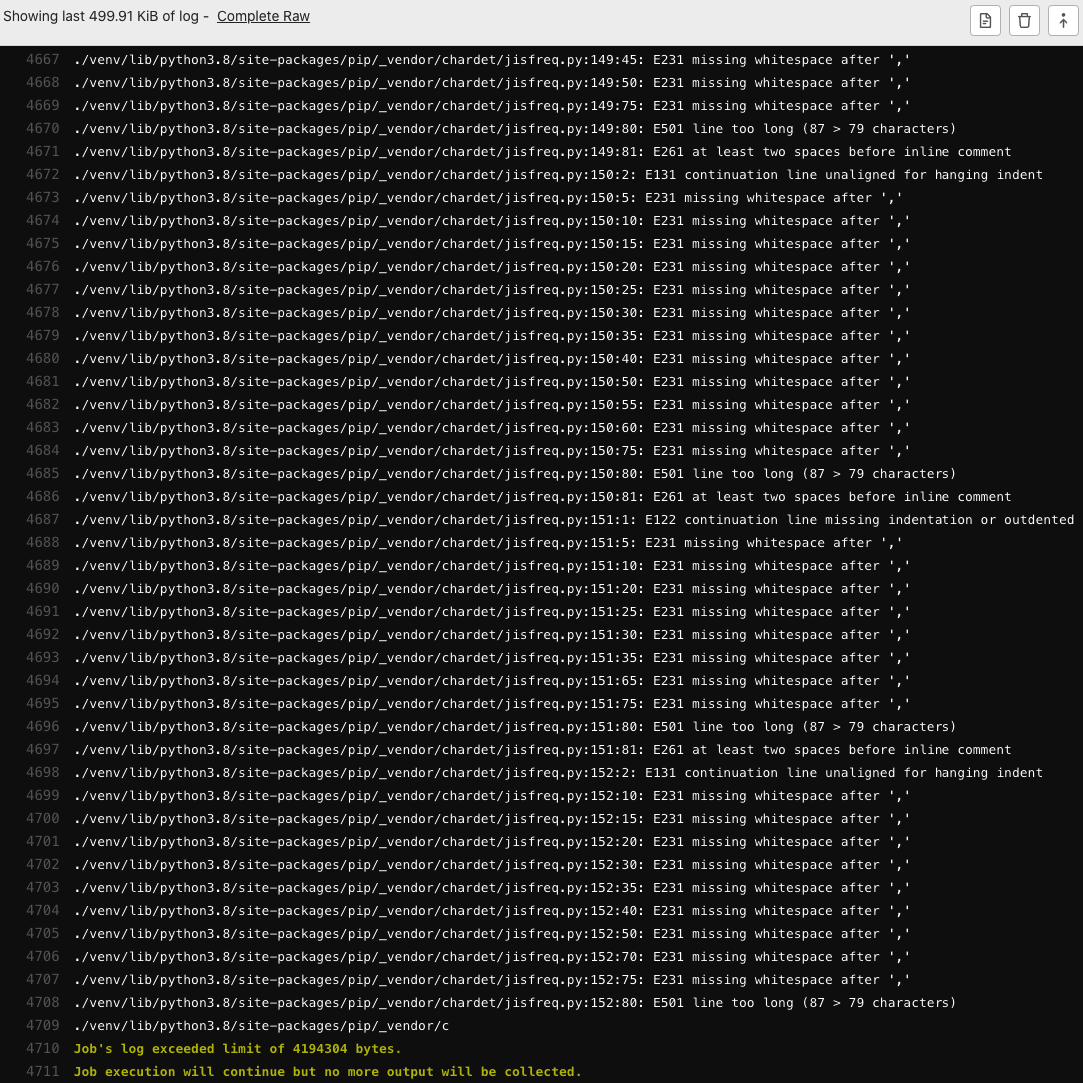
Разберемся с возникшей проблемой. Переходим в упавший пайплайн.



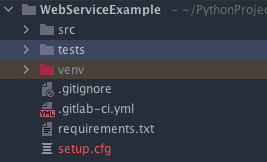




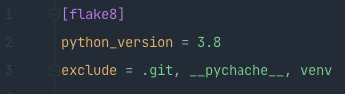
Далее нам открываются логи нашего пайплайна. Видим большое количество ошибок от flake8 в коде, который находится в библиотеках, которые мы использовали.



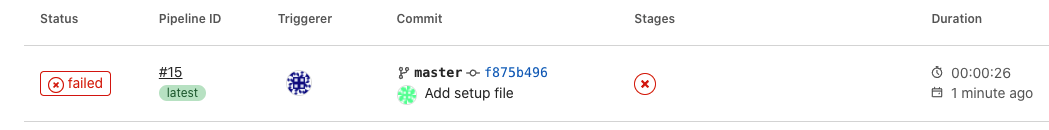
Чтобы исключить эти файлы создадим файл *setup.cfg.*



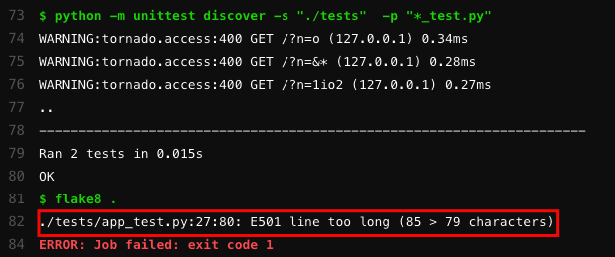
Укажем используемую версию python и папки, которые не надо проверять.



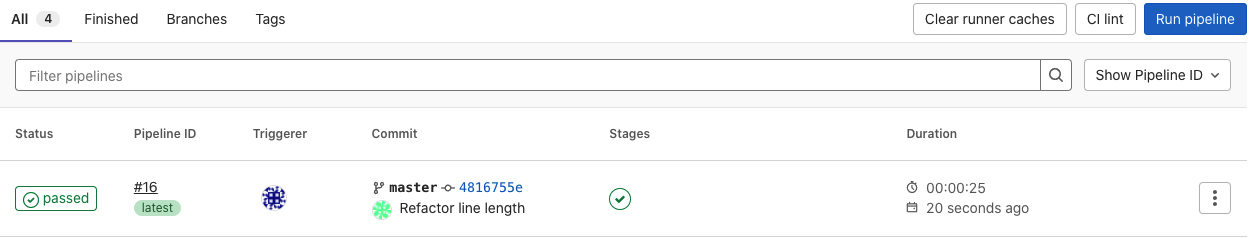
*Делаем коммит и выполняем пуш в свой репозиторий!*



Повторно смотрим логи и исправляем все несоответствия PEP правилам.



*Делаем коммит и выполняем пуш в свой репозиторий!*

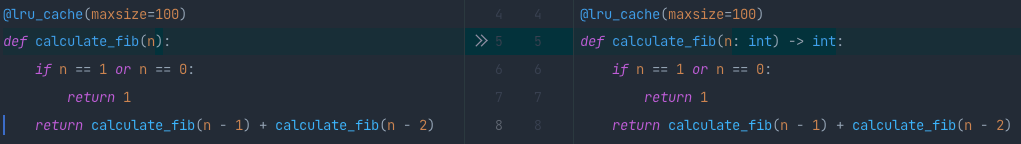


**Проверка типов данных**

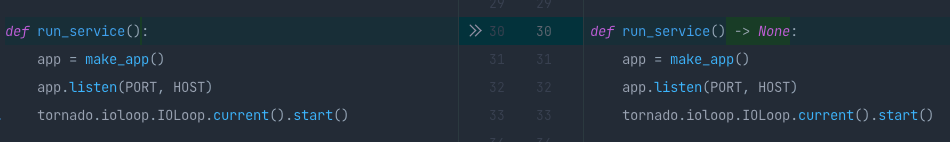
Для языков с динамической типизацией существует проблема, который лишини ЯП со статической типизацией, ошибки с типами данных. Например, у С++/Go/Rust и многих других статически типизированных языков, проверки типов производятся компилятором. В случае python мы лишены данной возможности “из коробки”, однако воспользуемся аннотацией типов и модулем mypy.

Установим mypy:

*pip install mypy*



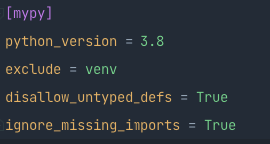
Если функция ничего не возвращает (аналог void в с-like языках), необходимо указать None.



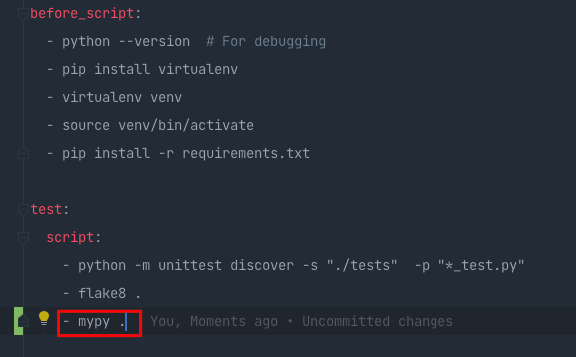
Питон все также остается динамически типизированным языком (интерпретатор полностью игнорирует аннотации), однако становится проще не ошибиться в большом проекте, помимо этого улучшается работа подсказок в IDE.

*Таким образом анатируем все функции и методы, которые есть у нас в коде*.

Также нам необходимо добавить базовые настройки в setup файл.



Добавим запуск mypy в нашем пайплайне.



*Делаем коммит и выполняем пуш в свой репозиторий!*

