



> АВТОМАТИЗАЦИЯ ОТЧЕТНОСТИ

> Создаем бота в Telegram

Создайте своего телеграм-бота с помощью [@BotFather](#)

Чтобы получить chat_id, воспользуйтесь

ссылкой https://api.telegram.org/bot<токен_вашего_бота>/getUpdates

ВАЖНО: не забудьте выдать боту разрешение перед тем, как начать отправлять сообщения от имени бота в чаты/личные сообщения. Начните с ботом диалог, чтобы разрешить ему писать в личные сообщения. Для этого необходимо перейти по ссылке [t.me/<никнейм_бота>](#) и запустить бота. Чтобы разрешить боту писать в чат/канал, предварительно добавьте туда бота и в настройках чата/канала выдайте ему права на отправку сообщений.

> Автоматизация с помощью CRON

Cron — планировщик задач. Последовательность и периодичность выполнения задач описывается в файле crontab. Для редактирования crontab файла используется специальная одноименная утилита. Она позволяет внести правки в файл расписания через терминал, не прерывая процесс cron на время редактирования.

Основные команды для работы с планировщиком:

- Создать или отредактировать файл расписания — **crontab -e**
- Вывести содержимое файла расписания — **crontab -l**
- Удалить файл расписания — **crontab -r**
- Загрузить таблицу задач из файла — **crontab path/to/file.crontab**

Структура строки в crontab файле:

расписание	задание
* * * * *	/usr/bin/python path/to/script >> path/to/logging 2>&1
- - - - -	день недели (0–7) (воскресенье = 0 или 7)
- - - - -	месяц (1–12)
- - - - -	день (1–31)
- - - - -	час (0–23)
- - - - -	минута (0–59)

Параметры расписания задаются целыми числами. Можно использовать специальные символы:

- все допустимые значения параметра времени *
- задать диапазон допустимых значений -
- задать множество допустимых значений ,
- задать шаг допустимых значений параметра /

Потренироваться задавать расписание можно [тут](#).

Задача — это действие, которое мы хотим автоматизировать и выполнять согласно указанному расписанию. Для автоматизации отчета в задаче необходимо указать следующую информацию:

- **/usr/bin/python** — путь к интерпретатору (используйте команды *whereis python* или *which python* в терминале, чтобы узнать путь к вашему интерпретатору);
- **path/to/script** — путь к скрипту, который нужно автоматизировать;
- **path/to/logging** — путь к файлу, в который будем записывать логи выполнения скрипта (необязательно).

> Автоматизация с помощью GitLab CD/CI

Давайте рассмотрим еще одну систему автоматизации — **GitLab CI/CD**. При совершении какого-либо действия в репозитории или в соответствии с заданным расписанием, будет запускаться цепочка автоматизированных задач. CI/CD система предоставляет вычислительные единицы, на которых будут выполняться задачи.

Основные составляющие, необходимые для успешной автоматизации:

- код в репозитории, который хотим автоматизировать
- вычислительные единицы — runners
- задачи, автоматизирующие CI/CD

Настройки для CI/CD — задачи для runners хранятся в специальном файле `.gitlab-ci.yml` в корне репозитория. При желании имя файла и его расположение в репозитории можно изменить в Settings -> CI/CD -> General Pipelines.

Gitlab предлагает готовые шаблоны для `.gitlab-ci.yml`.

Если GitLab видит, что в ветке есть такой файл, он запускает сборку по алгоритму, описанному в этом файле.

Основное понятие Gitlab CI/CD — stages (стейджи). Стейджи определяют этапы выполнения процесса (сборки) и их порядок. Стейджи могут быть абсолютно разные, можно задать любое количество. По дефолту используется следующие стейджи:

```
stages:  
  - build  
  - test  
  - deploy
```

Такой выбор дефолтных стейджей обусловлен тем, что у разработчиков и правда *в целом* три этапа работы с кодом: собрать, протестировать(убедиться, что это можно деплоить), задеплоить. И если один из них упал, остальные запускать нет необходимости.

Действия, которые необходимо автоматизировать, описываются в **джобах**. В самом простом случае джоба имеет следующий вид:

```
job_name:
  stage: test
  script:
    - echo 'test run script'
```

Внутри джобы в секции **script** указываются команды, которые мы хотим автоматизировать, в секции **stage** указывается стейдж-этап, в котором джоба должна выполняться.

Можно воспользоваться секциями **before_script** и **after_script**, чтобы указать действия, которые должны быть выполнены соответственно до и после основной секции **script**.

В один стейдж можно поместить несколько джоб. **Джобы** изолированы друг от друга. При необходимости передавать файлы и папки между джобами можно при помощи секции **cache**.

Внутри одного стейджа - этапа все джобы выполняются без определенного порядка. По умолчанию джобы следующего этапа не стратуют, пока все джобы из предыдущего не завершились успешно.

Ознакомиться с другими полезными секциями можно [тут](#).

В GitLab существует несколько типов runner'ов. Один из них это docker. Docker (докер) - платформа для удобной разработки, тестирования и эксплуатации приложения.

Докер включает в себя следующие интересующие нас компоненты:

- image (образ)
- контейнер

Image - это компонент сборки докера - некоторый шаблон, который используется для создания контейнера. Образ может, например, содержать в себе ОС Ubuntu с предустановленными пакетами и приложениями.

Контейнер создается из докер-образа и представляет из себя ОС с пользовательскими файлами. Когда создается контейнер, докер-образ говорит докеру что должно находиться в контейнере и какие процессы должны быть запущены в нем. Именно внутри контейнера будет выполнено все, что указано в

секции `script` в джобе. Контейнеры изолированы между собой, поэтому процессы, запущенные в разных контейнерах не могут взаимодействовать.

В начале `.gitlab-ci.yml` файла в секции `image` можно указать докер-образ, на основе которого и будут создаваться контейнеры для запуска джоб.

```
image: cr.yandex/crp742p3qacifd2hcon2/practice-da:latest

stages:
  - build
  - test
  - deploy

job_name:
  stage: test
  script:
    - echo 'test run script'
```

> Как сохранить токен бота в секрете

В коде могут встречаться переменные, значения которых нужно хранить в секрете. Это могут быть, например, пароли для подключения к базе данных или токен для управления ботом. GitLab предоставляет возможность запустить код с такими переменными, при этом сохранив их в секрете.

Чтобы спрятать токен бота нужно действовать согласно следующему алгоритму:

- В GitLab'e перейти в раздел Settings -> CI/CD -> Variables
- Добавить секретную переменную, указав ее имя и значение

Auto DevOps

Add variable

Key

REPORT_BOT_TOKEN

Value

1987162789:AAFgHNqBv-v5VXP0cS0bt0xtXECUvw8akMs

Type

Variable

Environment scope

All (default)

Flags

☐ Protect variable ?
Export variable to pipelines running on protected branches and tags only.

☐ Mask variable ?
Variable will be masked in job logs. Requires values to meet regular expression requirements. [More information](#)

Cancel Add variable

- В коде импортировать библиотеку **os** и заменить значение секретной переменной на строку:

```
os.environ.get("REPORT_BOT_TOKEN")
```

Например, объявление бота в коде отчета теперь будет выглядеть следующим образом:

```
bot = telegram.Bot(token=os.environ.get("REPORT_BOT_TOKEN"))
```