



## > Конспект > 1 урок > Airflow

> [Планировщики задач](#)

> [Cron](#)

> [Airflow](#)

> [Граф и DAG](#)

### > Планировщики задач

Так или иначе, нам часто приходится выполнять наши задачи по некоторому расписанию, например:

1. Отправка менеджеру "свежего" отчёта по основным метрикам
2. Обновление базы данных
3. Даже банальные git pull и git commit можно делать по расписанию

Очевидно, что каждая из этих задач может звучать тривиально, но на деле это может быть большое количество команд, которые необходимо выполнять последовательно раз в день, а может и раз в 5 минут.

Да, вполне можно написать скрипты, исполнение которых приведет к желаемому результату, но даже простое исполнение скрипта один раз в определенный интервал может стать проблемой, например, в меру нашей забывчивости.

**Планировщики задач** способны избавить нас от выполнения "рутинных" и однообразных задач по расписанию.

Сам по себе, **планировщик задач** - это некоторая утилита, чаще всего с довольно тривиальным синтаксисом, способная раз в определенное время (или, например, по наступлению определенного состояния) выполнять установленную команду. Наиболее популярный сейчас планировщик - **Cron**.

## > Cron

**Cron (Command Run ON)** - это системная утилита в операционных системах семейства Unix, которая используется для выполнения определенных действий по расписанию.

В файл **crontab** помещаются инструкции, написанные на специальном синтаксисе, понятном **cron**.

Для описания инструкции Вам достаточно указать расписание, по которому следует выполнять команду, а затем саму команду.

Расписание указывается в виде следующего выражения:

минута час день(месяца) месяц день(недели)

Каждое значение может быть описано двумя цифрами или заданным шаблоном. День недели также можно описывать английскими сокращениями (MON, TUE, WED, THU, FRI, SAT, SUN).

Шаблоны, которые могут быть использованы в выражении:

- \* - любое значение
- , - несколько значений
- диапазон значений
- / - оператор который позволяет указать значения, которые будут повторяться в течение определенного интервала между ними. К примеру, если в поле минута указано `*/3`, задача будет исполняться каждую третью минуту

Синтаксис указания расписания, по началу, может казаться довольно громоздким и непонятным, но на деле - это один из самых удобных способов. Быстро и удобно составить необходимое расписание может помочь [сайт](#).

Далее, указывается исполняемая команда, тут уже всё ограничивается исключительно Вашим полетом фантазии, это может быть почти любая команда.

Пример описываемой инструкции:

`/3 * * 5 * echo 'Three minutes passed in May'` - каждые три минуты в 5 месяце (мае) будет выводиться сообщение о том, что прошло три минуты.

Это также может быть выполнение python-скрипта.

С помощью cron мы можем выполнять даже последовательность инструкций. Например, каждый день, в 12 часов, мы хотим подготавливать отчет, с помощью скрипта **prepare.py** через 5 минут (12:05), мы хотим отправлять наш отчет и для этого у нас есть специальный скрипт **send.py**, для этого можем подготовить инструкцию:

`00 12 * * * python3 prepare.py` - каждый день в 12 часов запускается скрипт, который подготавливает наш отчет

`05 12 * * * python3 send.py` - каждый день в 12:05 отчет будет отправляться менеджеру

Основными командами, позволяющими Вам работать с crontab, будут:

`crontab -l` - просмотр всех записанных инструкций

`crontab -e` - редакция инструкций

Со всеми доступными командами Вы можете ознакомиться, выполнив `man crontab`

Несмотря на все достоинства Cron, также остается существенное количество **недостатков**:

- **Отсутствие какой-либо визуализации.** Когда задач становится сильно много, очень сложно понимать что, когда и как выполняется, исключительно по `crontab -l` Вам довольно неудобно фильтровать Ваши задачи, строить последовательности их выполнения
- **Отсутствие какой-либо интерактивности.** Чтобы посмотреть исполняемые файлы, Вам придется довольно долго бегать от `crontab -l` до выполняемых

скриптов(хорошо, если они хотя бы в одной директории или структурированы), в попытке создать картину происходящего

- **Работа в терминале.** Редко, конечно, для аналитика работа в терминале становится серьёзной проблемой, но тем не менее, она требует гораздо больше ресурсов и усилий
- **Отсутствие сообщений об ошибках.** В случаях, когда что-либо не может выполниться и падает с ошибкой, cron не делает никакого уведомления об этом. Поэтому на отслеживание ошибки может уйти несколько рабочих часов (вспоминаем первые два пункта)

Решить большую часть проблем **Cron** может **Airflow**.

## > Airflow

**Airflow** - это библиотека(даже комплекс библиотек), позволяющая очень легко и удобно работать с расписанием и мониторингом выполняемых задач.

Интерфейс Airflow выглядит следующим образом

The screenshot displays the Apache Airflow web interface. At the top, there's a navigation bar with the Airflow logo and links for DAGs, Security, Browse, Admin, and Docs. The current time is 21:11 UTC. Below the navigation bar, the 'DAGs' section is active, showing a list of DAGs. The interface includes filters for 'All' (26), 'Active' (10), and 'Paused' (16) DAGs, along with a search bar and a 'Filter DAGs by tag' input. The DAGs are listed in a table with columns: DAG, Owner, Runs, Schedule, Last Run, Recent Tasks, Actions, and Links. The table lists several example DAGs, including 'example\_bash\_operator', 'example\_branch\_dop\_operator\_v3', 'example\_branch\_operator', 'example\_complex', 'example\_external\_task\_marker\_child', 'example\_external\_task\_marker\_parent', 'example\_kubernetes\_executor', 'example\_kubernetes\_executor\_config', 'example\_nested\_branch\_dag', and 'example\_passing\_params\_via\_test\_command'. Each row shows the DAG's status, owner, number of runs, schedule, last run time, and a visual representation of recent task runs.

DAG	Owner	Runs	Schedule	Last Run	Recent Tasks	Actions	Links
example_bash_operator	airflow	2	00 * * *	2020-10-26, 21:08:11	6	[Play] [Refresh] [Stop]	...
example_branch_dop_operator_v3	airflow	0	* / 1 * * *		0	[Play] [Refresh] [Stop]	...
example_branch_operator	airflow	1	@daily	2020-10-23, 14:09:17	11	[Play] [Refresh] [Stop]	...
example_complex	airflow	1	None	2020-10-26, 21:08:04	37	[Play] [Refresh] [Stop]	...
example_external_task_marker_child	airflow	1	None	2020-10-26, 21:07:33	2	[Play] [Refresh] [Stop]	...
example_external_task_marker_parent	airflow	1	None	2020-10-26, 21:08:34	1	[Play] [Refresh] [Stop]	...
example_kubernetes_executor	airflow	0	None		0	[Play] [Refresh] [Stop]	...
example_kubernetes_executor_config	airflow	1	None	2020-10-26, 21:07:40	5	[Play] [Refresh] [Stop]	...
example_nested_branch_dag	airflow	1	@daily	2020-10-26, 21:07:37	9	[Play] [Refresh] [Stop]	...
example_passing_params_via_test_command	airflow	0	* / 1 * * *		0	[Play] [Refresh] [Stop]	...

Здесь мы видим большую подпись **DAGs**:

**DAG** - это основной юнит работы с Airflow, мы обсудим его подробнее позже, для начала можем считать, что это некоторая глобальная задача, решаемая путем последовательного выполнения более мелких, редуцированных задач.

Чуть детальнее рассмотрим интерфейс:

На главной страничке у нас перечислены все доступные ДАГи, вкладочки **All**, **Active** и **Paused** позволяют фильтровать ДАГи в соответствии с состоянием их выполнения, у каждого ДАГа сначала стоит переключатель, отвечающий за то, активен ли DAG или нет, затем идет название, владелец, информация о запусках и их состояниях, расписание(в формате Cron), информация по последним более маленьким задачам, составляющих нашу большую и некоторые хот-кеи для работы с ДАГом: запуск мгновенно, перезагрузить и удалить.

Чаще всего, для того, чтобы добавить DAG необходимо просто загрузить его в специальную выделенную папку в git - репозитории. О подробном устройстве и создании самих ДАГов Вы узнаете в следующих лекциях.

## > Граф и DAG

Прежде всего, вспомним, что такое граф.

**Граф** - это некоторая математическая абстракция, которая задается множеством вершин и множеством дуг между ними.

Мы же не будем сильно погружаться в математическую теорию графов, и для нас граф - это описание маленьких задач, которые мы будем называть тасками и описание правил перехода между ними.

(спойлер: DAG - это тоже граф, но немного особенный)

В общем случае, граф выглядит примерно следующим образом



Такой граф называется обыкновенным, здесь мы видим 4 вершины(задачи) и две дуги(перехода) между задачами 0 и 1, и задачами 0 и 2. В случае обыкновенного графа наличие перехода между двумя задачами обозначает возможность перехода в обе стороны, то есть, если между 0 и 1 есть переход, это означает, что мы можем переходить из 0 в 1 и из 1 в 0.

С точки зрения математики всё, конечно, здорово, но на практике мы сталкиваемся со следующей проблемой: наличие двух переходов между двумя вершинами в обе стороны сразу же приводит нас к циклу. Циклом называется последовательность переходов в результате которой мы можем вернуться в начало (т. е. в изначальную задачу).

В нашем случае, цикл, например:

0 -> 1, 1 -> 0, мы вышли из задачи 0 и в неё же смогли вернуться.

### Почему это проблема?

У нас может не быть явного правила для завершения выполнения DAGa. Помним о том, что DAG - некоторая глобальная задача, а задачи - всего лишь более мелкие, необходимы для выполнения глобальной.

На практике:

Представьте, что у нас есть задача: выгрузить данные, подготовить их и выслать менеджеру.

- Задача 1 - выгрузка данных
- Задача 0 - подготовка и обработка
- Задача 2 - отправка данных

Начинаем выполнение нашей глобальной задачи из задачи 1:

Мы выгрузили данные, перешли в задачу 0, подготовили и обработали данные, но, вместо того, чтобы просто перейти в задачу 2, мы можем вернуться в задачу 1 и снова начать выгружать данные. И такое может повторяться сколько угодно раз.

Аналогичных примеров можно придумать довольно много.

Поэтому Airflow накладывает некоторые ограничения на загружаемые глобальные задачи:

1. Граф глобальной задачи должен быть ациклическим (не должен содержать ни одного цикла)
2. Граф глобальной задачи должен быть направленным (между задачами путь в каждую сторону должен быть указан напрямую)

Объединяем эти правила и переводим на английский язык, получаем

**DAG** - Directed acyclic graph, направленный, ациклический граф.

Приведем пример ДАГа, построенного в соответствии с нашей задачей:



Как видим, после выполнения задачи 1 мы однозначно попадем в задачу 0, а после задачи 0 в задачу 2, а после выполнения задачи 2 наш DAG будет считаться выполненным.

Кстати, присутствие отдельной задачи 3, которая никак не используется, не вносит никаких препятствий для выполнения нашего DAG. Таким образом, мы можем "отключать" недоработанные или на данный момент ненужные задачи.

### Отойдем от теоретической модели, что же на практике в Airflow?

Для каждой глобальной задачи составляется DAG, DAG состоит из подзадач, которые необходимо выполнить для завершения глобальной задачи.

В веб-интерфейсе мы можем наблюдать DAG целиком, а также состояние выполнения каждой конкретной задачи, можем устанавливать количество попыток выполнения задачи, а также, в случае чего, видеть, что и с какой ошибкой упало.

Пример довольно непростого ДАГа, который может встретиться при работе:

