# django-cron Documentation

***Release 0.3.5***

**Tivix Inc.**

**Jun 06, 2018**

# Contents


**1 Introduction** **3**

**2 Installation** **5**

**3 Configuration** **7**

**4 Sample Cron Configurations** **9**
- 4.1 Retry after failure feature . . . 9
- 4.2 Run at times feature . . . 9
- 4.3 Allowing parallels runs . . . 10
- 4.4 FailedRunsNotificationCronJob . . . 10

**5 Locking Backend** **13**
- 5.1 Cache Lock . . . 13
- 5.2 File Lock . . . 13
- 5.3 Custom Lock . . . 13

**6 Changelog** **15**
- 6.1 0.5.1 . . . 15
- 6.2 0.5.0 . . . 15
- 6.3 0.4.6 . . . 15
- 6.4 0.4.5 . . . 15
- 6.5 0.4.4 . . . 15
- 6.6 0.4.3 . . . 16
- 6.7 0.4.2 . . . 16
- 6.8 0.4.1 . . . 16
- 6.9 0.4.0 . . . 16
- 6.10 0.3.6 . . . 16
- 6.11 0.3.5 . . . 16
- 6.12 0.3.4 . . . 16
- 6.13 0.3.3 . . . 17
- 6.14 0.3.2 . . . 17
- 6.15 0.3.1 . . . 17
- 6.16 0.3.0 . . . 17
- 6.17 0.2.9 . . . 17
- 6.18 0.2.8 . . . 17
- 6.19 0.2.7 . . . 17

6.20 0.2.6 . . . . . 17
6.21 0.2.5 . . . . . 17
6.22 0.2.4 . . . . . 18

**7 Indices and tables** **19**

Contents:

CHAPTER 1

# Introduction

Django-cron lets you run Django/Python code on a recurring basis proving basic plumbing to track and execute tasks. The two most common ways in which most people go about this is either writing custom python scripts or a management command per cron (leads to too many management commands!). Along with that some mechanism to track success, failure etc. is also usually necesary.

This app solves both issues to a reasonable extent. This is by no means a replacement for queues like Celery ( http://celeryproject.org/ ) etc.

This open-source app is brought to you by Tivix.

CHAPTER 2

# Installation

1. Install `django_cron` (ideally in your virtualenv!) using pip or simply getting a copy of the code and putting it in a directory in your codebase.
2. Add `django_cron` to your Django settings `INSTALLED_APPS`:

```
INSTALLED_APPS = [
    # ...
    "django_cron",
]
```

3. Run `python manage.py migrate django_cron`
4. Write a cron class somewhere in your code, that extends the `CronJobBase` class. This class will look something like this:

```
from django_cron import CronJobBase, Schedule

class MyCronJob(CronJobBase):
    RUN_EVERY_MINS = 120 # every 2 hours

    schedule = Schedule(run_every_mins=RUN_EVERY_MINS)
    code = 'my_app.my_cron_job'    # a unique code

    def do(self):
        pass    # do your thing here
```

5. Add a variable called `CRON_CLASSES` (similar to `MIDDLEWARE_CLASSES` etc.) thats a list of strings, each being a cron class. Eg.:

```
CRON_CLASSES = [
    "my_app.cron.MyCronJob",
    # ...
]
```

6. Now everytime you run the management command `python manage.py runcrons` all the crons will run if required. Depending on the application the management command can be called from the Unix crontab as often as required. Every 5 minutes usually works for most of my applications, for example:

```
> crontab -e
*/5 * * * * source /home/ubuntu/.bashrc && source /home/ubuntu/work/your-project/
↪bin/activate && python /home/ubuntu/work/your-project/src/manage.py runcrons > /
↪home/ubuntu/cronjob.log
```

Management Commands:

1. run a specific cron with `python manage.py runcrons cron_class ...`, for example:

```
# only run "my_app.cron.MyCronJob"
$ python manage.py runcrons "my_app.cron.MyCronJob"

# run "my_app.cron.MyCronJob" and "my_app.cron.AnotherCronJob"
$ python manage.py runcrons "my_app.cron.MyCronJob" "my_app.cron.
↪AnotherCronJob"
```

2. force run your crons with `python manage.py runcrons --force`, for example:

```
# run all crons, immediately, regardless of run time
$ python manage.py runcrons --force
```

3. run without any messages to the console `python manage.py runcrons --silent`, for example:

```
# run crons, if required, without message to console
$ python manage.py runcrons --silent
```

# CHAPTER 3

# Configuration

**CRON_CLASSES** - list of cron classes

**DJANGO_CRON_LOCK_BACKEND** - path to lock class, default: `"django_cron.backends.lock.cache.CacheLock"`

**DJANGO_CRON_LOCKFILE_PATH** - path where to store files for FileLock, default: `"/tmp"`

**DJANGO_CRON_LOCK_TIME** - timeout value for CacheLock backend, default: `24 * 60 * 60 # 24 hours`

**DJANGO_CRON_CACHE** - cache name used in CacheLock backend, default: `"default"`

**DJANGO_CRON_DELETE_LOGS_OLDER_THAN** - integer, number of days after which log entries will be clear (optional - if not set no entries will be deleted)

For more details, see *Sample Cron Configurations* and *Locking backend*

CHAPTER 4

# Sample Cron Configurations

## 4.1 Retry after failure feature

You can run cron by passing `RETRY_AFTER_FAILURE_MINS` param.

This will re-runs not next time runcrons is run, but at least `RETRY_AFTER_FAILURE_MINS` after last failure:

```
class MyCronJob(CronJobBase):
    RUN_EVERY_MINS = 60 # every hours
    RETRY_AFTER_FAILURE_MINS = 5

    schedule = Schedule(run_every_mins=RUN_EVERY_MINS, retry_after_failure_mins=RETRY_
↪AFTER_FAILURE_MINS)
```

## 4.2 Run at times feature

You can run cron by passing `RUN_EVERY_MINS` or `RUN_AT_TIMES` params.

This will run job every hour:

```
class MyCronJob(CronJobBase):
    RUN_EVERY_MINS = 60 # every hours

    schedule = Schedule(run_every_mins=RUN_EVERY_MINS)
```

This will run job at given hours:

```
class MyCronJob(CronJobBase):
    RUN_AT_TIMES = ['11:30', '14:00', '23:15']

    schedule = Schedule(run_at_times=RUN_AT_TIMES)
```

Hour format is HH:MM (24h clock). django-cron will interpret these times in the local timezone of your site, as specified by the TIME_ZONE setting.

You can also mix up both of these methods:

```
class MyCronJob(CronJobBase):
    RUN_EVERY_MINS = 120 # every 2 hours
    RUN_AT_TIMES = ['6:30']

    schedule = Schedule(run_every_mins=RUN_EVERY_MINS, run_at_times=RUN_AT_TIMES)
```

This will run job every 2h plus one run at 6:30.

## 4.3 Allowing parallels runs

By default parallels runs are not allowed (for security reasons). However if you want enable them just add:

```
ALLOW_PARALLEL_RUNS = True
```

in your CronJob class.

**Note:** Note this requires a caching framework to be installed, as per https://docs.djangoproject.com/en/dev/topics/cache/

If you wish to override which cache is used, put this in your settings file:

```
DJANGO_CRON_CACHE = 'cron_cache'
```

## 4.4 FailedRunsNotificationCronJob

This example cron check last cron jobs results. If they were unsuccessfull 10 times in row, it sends email to user.

Install required dependencies: Django>=1.7.0, django-common>=0.5.1.

Add django_cron.cron.FailedRunsNotificationCronJob to your CRON_CLASSES in settings file.

To set up minimal number of failed runs set up MIN_NUM_FAILURES in your cron class (default = 10). For example:

```
class MyCronJob(CronJobBase):
    RUN_EVERY_MINS = 10
    MIN_NUM_FAILURES = 3

    schedule = Schedule(run_every_mins=RUN_EVERY_MINS)
    code = 'app.MyCronJob'

    def do(self):
        ... some action here ...
```

Emails are imported from ADMINS in settings file

To set up email prefix, you must add FAILED_RUNS_CRONJOB_EMAIL_PREFIX in your settings file (default is empty). For example:

```
FAILED_RUNS_CRONJOB_EMAIL_PREFIX = "[Server check]: "
```

`FailedRunsNotificationCronJob` checks every cron from `CRON_CLASSES`

CHAPTER 5

# Locking Backend

You can use one of two built-in locking backends by setting `DJANGO_CRON_LOCK_BACKEND` with one of:

- `django_cron.backends.lock.cache.CacheLock` (default)
- `django_cron.backends.lock.file.FileLock`

## 5.1 Cache Lock

This backend sets a cache variable to mark current job as "already running", and delete it when lock is released.

## 5.2 File Lock

This backend creates a file to mark current job as "already running", and delete it when lock is released.

## 5.3 Custom Lock

You can also write your custom backend as a subclass of `django_cron.backends.lock.base.DjangoCronJobLock` and defining `lock()` and `release()` methods.

CHAPTER 6

# Changelog

## 6.1 0.5.1

- Fixed error in file locking backend with Python 3
- Fixed *'NoneType' object has no attribute 'utcoffset'* error
- Updated unit tests and demo for Django 2.0 compatibility

## 6.2 0.5.0

- Added support for Django 1.10
- Minimum Django version required is 1.8
- Use parser.add_argument() instead of optparse.make_option() in runcrons command

## 6.3 0.4.6

- Model import error fix for Django 1.9.X

## 6.4 0.4.5

- Added ability to check how many time left until next run.

## 6.5 0.4.4

- Remove max_length from CronJobLog.message field.

## 6.6 0.4.3

- Added DJANGO_CRON_DELETE_LOGS_OLDER_THAN setting to allow automated log clearing.

## 6.7 0.4.2

- Fix for #57 (ignoring Django timezone settings)

## 6.8 0.4.1

- Added get_prev_success_cron method to Schedule (Issue #26)
- Improvements to Admin interface (PR #42)

## 6.9 0.4.0

- Added support for Django 1.8
- Minimum Django version required is 1.7
- Dropped South in favor of Django migrations
- WARNING! When upgrading you might need to remove existing South migrations, read more: https://docs.djangoproject.com/en/1.7/topics/migrations/#upgrading-from-south

## 6.10 0.3.6

- Added Django 1.7 support
- Added python3 support

## 6.11 0.3.5

- Added locking backends
- Added tests

## 6.12 0.3.4

- Added CRON_CACHE settings parameter for cache select
- Handle database connection errors
- Upping requirement to Django 1.5+

## 6.13 0.3.3

- Python 3 compatibility.

## 6.14 0.3.2

- Added database connection close.
- Added better exceptions handler.

## 6.15 0.3.1

- Added index_together entries for faster queries on large cron log db tables.
- Upgraded requirement hence to Django 1.5 and South 0.8.1 since `index_together` is new to Django 1.5

## 6.16 0.3.0

- Added Django 1.4+ support. Updated requirements.

## 6.17 0.2.9

- Changed log level to debug() in CronJobManager.run() function.

## 6.18 0.2.8

- Bug fix
- Optimized queries. Used latest() instead of order_by()

## 6.19 0.2.7

- Bug fix.

## 6.20 0.2.6

- Added *end_time* to list_display in CronJobLog admin

## 6.21 0.2.5

- Added a helper function ( run_cron_with_cache_check ) in runcrons.py

## 6.22 0.2.4

- Capability to run specific crons using the runcrons management command. Useful when in the list of crons there are few slow onces and you might want to run some quicker ones via a separate crontab entry to make sure they are not blocked / slowed down.
- pep8 cleanup and reading from settings more carefully (getattr).

CHAPTER 7

# Indices and tables

- genindex
- modindex
- search