# EdX Management Commands Documentation

*Release 0.1*

**edX**

April 04, 2016

Reusable management commands for automated deployment of Django Applications.

CHAPTER 1

# Getting Started

If you have not already done so, create/activate a virtualenv. Unless otherwise stated, assume all terminal code below is executed within the virtualenv.

## 1.1 Install dependencies

Dependencies can be installed via the command below.

```
$ make requirements
```

## 1.2 Run migrations

Local installations use SQLite by default. If you choose to use another database backend, make sure you have updated your settings and created the database (if necessary). Migrations can be run with Django's migrate command.

```
$ python manage.py migrate
```

## 1.3 Run the server

The server can be run with Django's runserver command. If you opt to run on a different port, make sure you update OIDC client via LMS admin.

```
$ python manage.py runserver 8003
```

CHAPTER 2

# Management Commands

## 2.1 Use Case

These management commands are designed for use when there is a need to programmatically create or update Django users, groups, and permissions, without needing to hardcode these in migrations or application code.

The commands are designed to work idempotently - that is, running them multiple times with the same inputs should always result in the same outcomes, and without additional side-effects if they are run more than once.

## 2.2 manage_group

The `manage_group` command is used to ensure that a specific `django.contrib.auth.models.Group` exists, and that it has a specific set of permissions.

The following example will create a group named `mygroup` if it does not exist (and will do nothing if it does exist):

```
$ python manage.py manage_group mygroup
```

The following example will create or update the group named `privileged_group` and assign to it the permissions for adding and deleting instances of model “Thing” in “myapp”:

```
$ python manage.py manage_group privileged_group -p myapp:Thing:add_thing myapp:Thing:delete_thing
```

In the above example, if the group already possessed another permission such as “update_thing”, that permission would be automatically removed.

An option to remove, instead of create, is also available:

```
$ python manage.py manage_group mygroup --remove
```

In the above example, the matching group would be deleted if it existed, and the command would do nothing otherwise.

## 2.3 manage_user

The `manage_user` command is used to ensure that a specific user exists, and that it has a specific set of groups. It can also be used to toggle the `is_staff` and `is_superuser` bits on that user account.

The following example will create a user named `groucho` if it does not exist:

```
$ python manage.py manage_user groucho groucho@example.com
```

In the above example, if `groucho` already exists, the command will continue without action / error, as long as the supplied email address matches the one already stored. If this is not the case, the command will fail with an error. This is intended as a control measure against potential scripting errors that might cause an existing user to lose control of an existing account.

To set the staff or superuser bits, use the associated options:

```
$ python manage.py manage_user groucho groucho@example.com --staff --superuser
```

In the above example, both bits will be set to true. In order to set the bits to false, omit the options.

To assign a user to specific groups, use the *-g* option:

```
$ python manage.py manage_user groucho groucho@example.com -g mygroup privileged_group
```

The above will ensure that `groucho` will be assigned to `mygroup` and `privileged_group`, and removed from any others to which they may already have been assigned.

Note that if any of the groups specified with `-g` do not exist, the command will output a warning, but will continue without creating the group. This measure is intended to ensure that, in an automated deployment setting, a prior misconfiguration involving some particular group will not cause a cascading failure in setting (or removing) other group memberships.

An option to remove, instead of create, is also available:

```
$ python manage.py manage_user groucho groucho@example.com --remove
```

In the above example, the matching user would be deleted if it existed, and the command would do nothing otherwise.

CHAPTER 3

# Testing

The command below runs the Python tests and code quality validation—Pylint and PEP8.

```
$ make validate
```

Code quality validation can be run independently with:

```
$ make quality
```

CHAPTER 4

# Internationalization

All user-facing text content should be marked for translation. Even if this application is only run in English, our open source users may choose to use another language. Marking content for translation ensures our users have this choice.

Follow the internationalization coding guidelines in the edX Developer's Guide when developing new features.

## 4.1 Updating Translations

This project uses Transifex to translate content. After new features are developed the translation source files should be pushed to Transifex. Our translation community will translate the content, after which we can retrieve the translations.

Pushing source translation files to Transifex requires access to the edx-platform. Request access from the Open Source Team if you will be pushing translation files. You should also configure the Transifex client if you have not done so already.

The *make* targets listed below can be used to push or pull translations.

| Target | Description |
|---|---|
| pull_translations | Pull translations from Transifex |
| push_translations | Push source translation files to Transifex |

## 4.2 Fake Translations

As you develop features it may be helpful to know which strings have been marked for translation, and which are not. Use the *fake_translations* make target for this purpose. This target will extract all strings marked for translation, generate fake translations in the Esperanto (eo) language directory, and compile the translations.

You can trigger the display of the translations by setting your browser's language to Esperanto (eo), and navigating to a page on the site. Instead of plain English strings, you should see specially-accented English strings that look like this:

> Thé Fütüré øf Ønlïné Édüçätïøn $\sigma$ $\iota$# Før änýøné, änýwhéré, änýtïmé $\sigma$ #