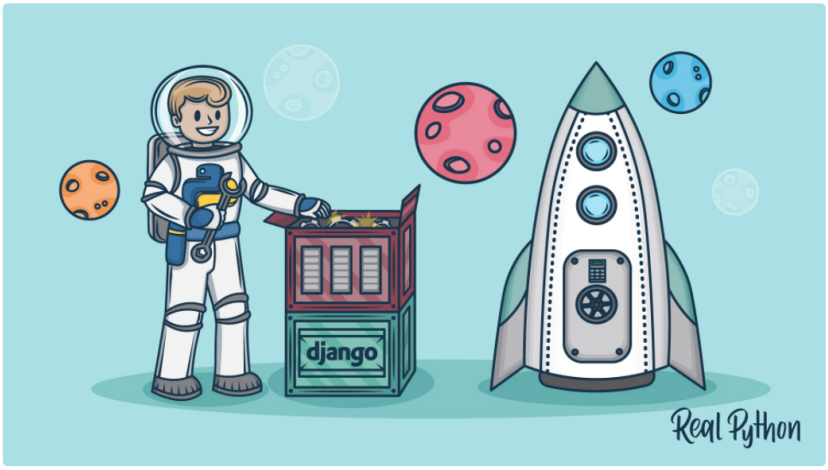

Your First Steps With Django: Set Up a Django Project

by Martin Breuss Jul 21, 2021 24 Comments basics best-practices django web-dev

Mark as Completed

Tweet Share Email

Table of Contents

Watch Now This tutorial has a related video course created by the Real Python team. Watch it together with the written tutorial to deepen your understanding: **How to Set Up a Django Project**

Before you can start to build the individual functionality of a new Django web application, you always need to complete a couple of setup steps. This tutorial gives you a reference for the necessary steps to set up a Django project.

The tutorial focuses on the initial steps you'll need to take to start a new web application. To complete it, you'll need to have Python installed and understand how to work with virtual environments and Python's package manager, `pip`. While you won't need much programming knowledge to complete this setup, you'll need to know Python to do anything interesting with the resulting project scaffolding.

By the end of this tutorial, you'll know how to:

- Set up a **virtual environment**
- **Install** Django
- Pin your project **dependencies**
- Set up a Django **project**
- Start a Django **app**

All Tutorial Topics

advanced api basics best-practices community databases data-science devops django docker flask front-end gamedev gui intermediate machine-learning projects python testing tools web-dev web-scraping

Table of Contents

- Prepare Your Environment
- Install Django and Pin Your Dependencies
- Set Up a Django Project
- Start a Django App
- Command Reference
- Conclusion

Mark as Completed

Tweet Share Email

Recommended Video Course
How to Set Up a Django Project

Use this tutorial as your go-to reference until you've built so many projects that the necessary commands become second nature. Until then, follow the steps outlined below. There are also a few exercises throughout the tutorial to help reinforce what you've learned.

Prepare Your Environment

When you're ready to start your new Django web application, create a new folder and navigate into it. In this folder, you'll set up a new virtual environment using your command line:

Shell

```
$ python3 -m venv env
```

This command sets up a new virtual environment named `env` in your current working directory. Once the process is complete, you also need to activate the virtual environment:

Shell

```
$ source env/bin/activate
```

If the activation was successful, then you'll see the name of your virtual environment, `(env)`, at the beginning of your command prompt. This means that your environment setup is complete.

You can learn more about how to work with virtual environments in Python, and how to perfect your Python development setup, but for your Django setup, you have all you need. You can continue with installing the `django` package.

Install Django and Pin Your Dependencies

Once you've created and activated your Python virtual environment, you can install Django into this dedicated development workspace:

Shell

```
(env) $ python -m pip install django
```

This command fetches the `django` package from the Python Package Index (PyPI) using `pip`. After the installation has completed, you can **pin** your dependencies to make sure that you're keeping track of which Django version you installed:

Shell

```
(env) $ python -m pip freeze > requirements.txt
```

This command writes the names and versions of all external Python packages that are currently in your virtual environment to a file called `requirements.txt`. This file will include the `django` package and all of its dependencies.

Note: There are many different versions of Django. While it's usually best to work with the most recent version when starting a new project, you might have to work with a specific version for one particular project. You can install any version of Django by adding the version number to the installation command:

Shell

```
(env) $ python -m pip install django==2.2.11
```

This command installs the version `2.2.11` of Django to your environment instead of fetching the most recent version. Replace the number after the double equals sign (`==`) with the specific Django version you need to install.

You should always include a record of the versions of all packages you used in your project code, such as in a `requirements.txt` file. The `requirements.txt` file allows you and other programmers to reproduce the exact conditions of your project build.

Exercise: Explore Further Show/Hide

Suppose you're working on an existing project with its dependencies already pinned in a `requirements.txt` file. In that case, you can install the right Django version as well as all the other necessary packages in a single command:

Shell

```
(env) $ python -m pip install -r requirements.txt
```

The command reads all names and versions of the pinned packages from your `requirements.txt` file and installs the specified version of each package in your virtual environment.

Keeping a separate virtual environment for every project allows you to work with different versions of Django for different web application projects. Pinning the dependencies with `pip freeze` enables you to reproduce the environment that you need for the project to work as expected.

Set Up a Django Project

After you've successfully installed Django, you're ready to create the scaffolding for your new web application. The Django framework distinguishes between **projects** and **apps**:

- **A Django project** is a high-level unit of organization that contains logic that governs your whole web application. Each project can contain multiple apps.
- **A Django app** is a lower-level unit of your web application. You can have zero to many apps in a project, and you'll usually have at least one app. You'll learn more about apps in the next section.

With your virtual environment set up and activated and Django installed, you can now create a project:

Shell

```
(env) $ django-admin startproject <project-name>
```

This tutorial uses `setup` as an example for the project name:

Shell

```
(env) $ django-admin startproject setup
```

Running this command creates a default folder structure, which includes some Python files and your management app that has the same name as your project:

```
setup/
│
├── setup/
│   ├── __init__.py
│   ├── asgi.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
│
└── manage.py
```

In the code block above, you can see the folder structure that the `startproject` command created for you:

- **`setup/`** is your top-level project folder.
- **`setup/setup/`** is your lower-level folder that represents your management app.
- **`manage.py`** is a Python file that serves as the command center of your project. It does the same as the `django-admin` command-line utility.

The nested `setup/setup/` folder contains a couple more files that you'll edit when you work on your web application.

> **Note:** If you want to avoid creating the additional top-level project folder, you can add a dot (`.`) at the end of the `django-admin startproject` command:
>
> Shell
>
> ```
> (env) $ django-admin startproject <projectname> .
> ```
>
> The dot skips the top-level project folder and creates your management app and the `manage.py` file right inside your current working directory. You might encounter this syntax in some online Django tutorials. All it does is create the project scaffolding without an extra top-level project folder.

Take a moment to explore the default project scaffolding that the `django-admin` command-line utility created for you. Every project that you'll make using the `startproject` command will have the same structure.

When you're ready, you can move on to create a **Django app** as a lower-level unit of your new web application.

Start a Django App

Every project you build with Django can contain multiple Django apps. When you ran the `startproject` command in the previous section, you created a management app that you'll need for every default project that you'll build. Now, you'll create a Django app that'll contain the specific functionality of your web application.

You don't need to use the `django-admin` command-line utility anymore, and you can execute the `startapp` command through the `manage.py` file instead:

Shell

```
(env) $ python manage.py startapp <appname>
```

The `startapp` command generates a default folder structure for a Django app. This tutorial uses `example` as the name for the app:

Shell

```
(env) $ python manage.py startapp example
```

Remember to replace `example` with your app name when you create the Django app for your personal web application.

> **Note:** If you created your project without the dot shortcut mentioned further up, you'll need to change your working directory into your top-level project folder before running the command shown above.

Once the `startapp` command has finished execution, you'll see that Django has added another folder to your folder structure:

```
setup/
│
├── example/
│   │
│   ├── migrations/
│   │   └── __init__.py
│   │
│   ├── __init__.py
│   ├── admin.py
│   ├── apps.py
│   ├── models.py
│   ├── tests.py
│   └── views.py
│
├── setup/
│   ├── __init__.py
│   ├── asgi.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
│
└── manage.py
```

The new folder has the name you gave it when running the command. In the case of this tutorial, that's `example/`. You can see that the folder contains a couple of Python files.

This Django app folder is where you'll spend most of your time when creating your web application. You'll also need to make some changes in the management app, `setup/`, but you'll build most of your functionality inside the Django app, `example/`.

Exercise: Explore Further Show/Hide

You'll get to know the generated Python files in more detail when working through a tutorial or building your own project. Here are three notable files that were created in the app folder:

1. **`__init__.py`**: Python uses this file to declare a folder as a package, which allows Django to use code from different apps to compose the overall functionality of your web application. You probably won't have to touch this file.
2. **`models.py`**: You'll declare your app's models in this file, which allows Django to interface with the database of your web application.
3. **`views.py`**: You'll write most of the code logic of your app in this file.

At this point, you've finished setting up the scaffolding for your Django web application, and you can start implementing your ideas. From here on out, it's up to you what you want to build to create your own unique project.

Command Reference

The table below provides you with a quick reference of the necessary commands to start your Django development process. The steps in the reference table link back to the sections of this tutorial where you can find more detailed explanations:

Step	Description	Command
1a	Set up a virtual environment	`python -m venv env`
1b	Activate the virtual environment	`source env/bin/activate`
2a	Install Django	`python -m pip install django`
2b	Pin your dependencies	`python -m pip freeze > requirements.txt`
3	Set up a Django project	`django-admin startproject <projectname>`
4	Start a Django app	`python manage.py startapp <appname>`

Use this table as a quick reference for starting a new web application with Django inside of a Python virtual environment.

Conclusion

In this tutorial, you went over all the steps necessary to set up the foundations for a new Django web application. You got familiar with the most common terminal commands that you'll repeat over and over again when using Django for web development.

You also read about *why* you want to use each command and what they produce, and you learned some tips and tricks associated with getting set up with Django.

In this tutorial, you learned how to:

- Set up a **virtual environment**
- **Install** Django
- Pin your project **dependencies**
- Set up a Django **project**
- Start a Django **app**

After completing the steps outlined in this tutorial, you're ready to start building your custom web application using Django. For example, you could create a portfolio app to showcase your coding projects. For a structured way to keep growing your Django skills, you can continue working through the resources provided in the Django learning path.

Keep setting up the fundamental scaffolding for Django web applications until these steps become second nature. If you need a refresher, then you can always use this tutorial as a quick reference.

About **Martin Breuss**

Martin likes automation, goofy jokes, and snakes, all of which fit into the Python community. He enjoys learning and exploring

and is up for talking about it, too. He writes and records content for Real Python and CodingNomads.

» More about Martin

Each tutorial at Real Python is created by a team of developers so that it meets our high quality standards. The team members who worked on this tutorial are:

 Adriana

 Aldren

 Bartosz

 Joanna

 Jacob

 Michael

What Do You Think?

Real Python Comment Policy: The most useful comments are those written with the goal of learning from or helping out other readers—after reading the whole article and all the earlier comments. Complaints and insults generally won't make the cut here.

What's your #1 takeaway or favorite thing you learned? How are you going to put your newfound skills to use? Leave a comment below and let us know.

Keep Learning

Keep Learning

Related Tutorial Categories: basics best-practices django web-dev

Recommended Video Course: How to Set Up a Django Project