Install Python**¶**

Get the latest version of Python at https://www.python.org/downloads/ or with your operating system’s package manager.

Django on Jython

Jython (a Python implementation for the Java platform) is not compatible with Python 3, so Django ≥ 2.0 cannot run on Jython.

Python on Windows

If you are just starting with Django and using Windows, you may find How to install Django on Windows useful.

Install Apache and mod\_wsgi**¶**

If you just want to experiment with Django, skip ahead to the next section; Django includes a lightweight web server you can use for testing, so you won’t need to set up Apache until you’re ready to deploy Django in production.

If you want to use Django on a production site, use Apache with mod\_wsgi. mod\_wsgi can operate in one of two modes: an embedded mode and a daemon mode. In embedded mode, mod\_wsgi is similar to mod\_perl – it embeds Python within Apache and loads Python code into memory when the server starts. Code stays in memory throughout the life of an Apache process, which leads to significant performance gains over other server arrangements. In daemon mode, mod\_wsgi spawns an independent daemon process that handles requests. The daemon process can run as a different user than the Web server, possibly leading to improved security, and the daemon process can be restarted without restarting the entire Apache Web server, possibly making refreshing your codebase more seamless. Consult the mod\_wsgi documentation to determine which mode is right for your setup. Make sure you have Apache installed, with the mod\_wsgi module activated. Django will work with any version of Apache that supports mod\_wsgi.

See How to use Django with mod\_wsgi for information on how to configure mod\_wsgi once you have it installed.

If you can’t use mod\_wsgi for some reason, fear not: Django supports many other deployment options. One is uWSGI; it works very well with nginx. Additionally, Django follows the WSGI spec (**PEP 3333**), which allows it to run on a variety of server platforms.

Get your database running**¶**

If you plan to use Django’s database API functionality, you’ll need to make sure a database server is running. Django supports many different database servers and is officially supported with PostgreSQL, **MySQL**, Oracle and **SQLite (default database)**.

If you are developing a simple project or something you don’t plan to deploy in a production environment, SQLite is generally the simplest option as it doesn’t require running a separate server. However, SQLite has many differences from other databases, so if you are working on something substantial, it’s recommended to develop with the same database as you plan on using in production.

In addition to the officially supported databases, there are backends provided by 3rd parties that allow you to use other databases with Django.

In addition to a database backend, you’ll need to make sure your Python database bindings are installed.

* If you’re using PostgreSQL, you’ll need the psycopg2 package. Refer to the PostgreSQL notes for further details.
* If you’re using MySQL, you’ll need a DB API driver like mysqlclient. See notes for the MySQL backend for details.
* If you’re using SQLite you might want to read the SQLite backend notes.
* If you’re using Oracle, you’ll need a copy of cx\_Oracle, but please read the notes for the Oracle backend for details regarding supported versions of both Oracle and cx\_Oracle.
* If you’re using an unofficial 3rd party backend, please consult the documentation provided for any additional requirements.

If you plan to use Django’s manage.py migrate command to automatically create database tables for your models (after first installing Django and creating a project), you’ll need to ensure that Django has permission to create and alter tables in the database you’re using; if you plan to manually create the tables, you can simply grant Django SELECT, INSERT, UPDATE and DELETE permissions. After creating a database user with these permissions, you’ll specify the details in your project’s settings file, see DATABASES for details.

If you’re using Django’s testing framework to test database queries, Django will need permission to create a test database.

Remove any old versions of Django**¶**

If you are upgrading your installation of Django from a previous version, you will need to uninstall the old Django version before installing the new version.

If you installed Django using pip or easy\_install previously, installing with pip or easy\_install again will automatically take care of the old version, so you don’t need to do it yourself.

If you previously installed Django using python setup.py install, uninstalling is as simple as deleting the django directory from your Python site-packages. To find the directory you need to remove, you can run the following at your shell prompt (not the interactive Python prompt):

$ python -c "import django; print(django.\_\_path\_\_)"

Install the Django code**¶**

Installation instructions are slightly different depending on whether you’re installing a distribution-specific package, downloading the latest official release, or fetching the latest development version.

It’s easy, no matter which way you choose.

**Installing an official release with pip¶**

This is the recommended way to install Django.

1. Install pip. The easiest is to use the standalone pip installer. If your distribution already has pip installed, you might need to update it if it’s outdated. If it’s outdated, you’ll know because installation won’t work.
2. Take a look at virtualenv and virtualenvwrapper. These tools provide isolated Python environments, which are more practical than installing packages systemwide. They also allow installing packages without administrator privileges. The contributing tutorial walks through how to create a virtualenv.
3. After you’ve created and activated a virtual environment, enter the command pip install Django at the shell prompt.

**Installing a distribution-specific package¶**

Check the distribution specific notes to see if your platform/distribution provides official Django packages/installers. Distribution-provided packages will typically allow for automatic installation of dependencies and easy upgrade paths; however, these packages will rarely contain the latest release of Django.

**Installing the development version¶**

Tracking Django development

If you decide to use the latest development version of Django, you’ll want to pay close attention to the development timeline, and you’ll want to keep an eye on the release notes for the upcoming release. This will help you stay on top of any new features you might want to use, as well as any changes you’ll need to make to your code when updating your copy of Django. (For stable releases, any necessary changes are documented in the release notes.)

If you’d like to be able to update your Django code occasionally with the latest bug fixes and improvements, follow these instructions:

1. Make sure that you have Git installed and that you can run its commands from a shell. (Enter git help at a shell prompt to test this.)
2. Check out Django’s main development branch like so:
3. $ git clone https://github.com/django/django.git

This will create a directory django in your current directory.

1. Make sure that the Python interpreter can load Django’s code. The most convenient way to do this is to use virtualenv, virtualenvwrapper, and pip. The contributing tutorial walks through how to create a virtualenv.
2. After setting up and activating the virtualenv, run the following command:
3. $ pip install -e django/

This will make Django’s code importable, and will also make the django-admin utility command available. In other words, you’re all set!

When you want to update your copy of the Django source code, just run the command git pull from within the django directory. When you do this, Git will automatically download any changes.

**Part-1**

Throughout this tutorial, we’ll walk you through the creation of a basic poll application.

It’ll consist of two parts:

* A public site that lets people view polls and vote in them.
* An admin site that lets you add, change, and delete polls.

We’ll assume you have Django installed already. You can tell Django is installed and which version by running the following command in a shell prompt (indicated by the $ prefix):

**$** python -m django --version

If Django is installed, you should see the version of your installation. If it isn’t, you’ll get an error telling “No module named django”.

This tutorial is written for Django 2.0, which supports Python 3.4 and later. If the Django version doesn’t match, you can refer to the tutorial for your version of Django by using the version switcher at the bottom right corner of this page, or update Django to the newest version. If you’re using an older version of Python, check What Python version can I use with Django? to find a compatible version of Django.

See How to install Django for advice on how to remove older versions of Django and install a newer one.

Where to get help:

If you’re having trouble going through this tutorial, please post a message to django-users or drop by #django on irc.freenode.net to chat with other Django users who might be able to help.

Creating a project**¶**

If this is your first time using Django, you’ll have to take care of some initial setup. Namely, you’ll need to auto-generate some code that establishes a Django project – a collection of settings for an instance of Django, including database configuration, Django-specific options and application-specific settings.

From the command line, cd into a directory where you’d like to store your code, then run the following command:

**$** django-admin startproject mysite

This will create a mysite directory in your current directory. If it didn’t work, see Problems running django-admin.

Note

You’ll need to avoid naming projects after built-in Python or Django components. In particular, this means you should avoid using names like django (which will conflict with Django itself) or test (which conflicts with a built-in Python package).

Where should this code live?

If your background is in plain old PHP (with no use of modern frameworks), you’re probably used to putting code under the Web server’s document root (in a place such as /var/www). With Django, you don’t do that. It’s not a good idea to put any of this Python code within your Web server’s document root, because it risks the possibility that people may be able to view your code over the Web. That’s not good for security.

Put your code in some directory **outside** of the document root, such as /home/mycode.

Let’s look at what startproject created:

mysite/

**manage.py** : integration of model, create, update, rebuild

mysite/

\_\_init\_\_.py :

**settings.py** : for configuration of database configuration, image, css, javascript, cookies etc.

**urls.py :** link/address of all applications(admin, other wesite)

wsgi.py

These files are:

* The outer mysite/ root directory is just a container for your project. Its name doesn’t matter to Django; you can rename it to anything you like.
* manage.py: A command-line utility that lets you interact with this Django project in various ways. You can read all the details about manage.py in django-admin and manage.py.
* The inner mysite/ directory is the actual Python package for your project. Its name is the Python package name you’ll need to use to import anything inside it (e.g. mysite.urls).
* mysite/\_\_init\_\_.py: An empty file that tells Python that this directory should be considered a Python package. If you’re a Python beginner, read more about packages in the official Python docs.
* mysite/settings.py: Settings/configuration for this Django project. Django settings will tell you all about how settings work.
* mysite/urls.py: The URL declarations for this Django project; a “table of contents” of your Django-powered site. You can read more about URLs in URL dispatcher.
* mysite/wsgi.py: An entry-point for WSGI-compatible web servers to serve your project. See How to deploy with WSGI for more details.

The development server**¶**

Let’s verify your Django project works. Change into the outer mysite directory, if you haven’t already, and run the following commands:

**$** python manage.py runserver

You’ll see the following output on the command line:

Performing system checks...

System check identified no issues (0 silenced).

You have unapplied migrations; your app may not work properly until they are applied.

Run 'python manage.py migrate' to apply them.

February 15, 2018 - 15:50:53

Django version 2.0, using settings 'mysite.settings'

Starting development server at http://127.0.0.1:8000/

Quit the server with CONTROL-C.

Note

Ignore the warning about unapplied database migrations for now; we’ll deal with the database shortly.

You’ve started the Django development server, a lightweight Web server written purely in Python. We’ve included this with Django so you can develop things rapidly, without having to deal with configuring a production server – such as Apache – until you’re ready for production.

Now’s a good time to note: **don’t** use this server in anything resembling a production environment. It’s intended only for use while developing. (We’re in the business of making Web frameworks, not Web servers.)

Now that the server’s running, visit http://127.0.0.1:8000/ with your Web browser. You’ll see a “Congratulations!” page, with a rocket taking off. It worked!

Changing the port

By default, the runserver command starts the development server on the internal IP at port 8000.

If you want to change the server’s port, pass it as a command-line argument. For instance, this command starts the server on port 8080:

**$** python manage.py runserver 8080

If you want to change the server’s IP, pass it along with the port. For example, to listen on all available public IPs (which is useful if you are running Vagrant or want to show off your work on other computers on the network), use:

**$** python manage.py runserver 0:8000

**0** is a shortcut for **0.0.0.0**. Full docs for the development server can be found in the runserver reference.

Automatic reloading of runserver

The development server automatically reloads Python code for each request as needed. You don’t need to restart the server for code changes to take effect. However, some actions like adding files don’t trigger a restart, so you’ll have to restart the server in these cases.

Creating the Polls app**¶**

Now that your environment – a “project” – is set up, you’re set to start doing work.

Each application you write in Django consists of a Python package that follows a certain convention. Django comes with a utility that automatically generates the basic directory structure of an app, so you can focus on writing code rather than creating directories.

Projects vs. apps

What’s the difference between a project and an app? An app is a Web application that does something – e.g., a Weblog system, a database of public records or a simple poll app. A project is a collection of configuration and apps for a particular website. A project can contain multiple apps. An app can be in multiple projects.

Your apps can live anywhere on your Python path. In this tutorial, we’ll create our poll app right next to your manage.py file so that it can be imported as its own top-level module, rather than a submodule of mysite.

To create your app, make sure you’re in the same directory as manage.py and type this command:

**$** python manage.py startapp polls

That’ll create a directory polls, which is laid out like this:

polls/

\_\_init\_\_.py

admin.py

apps.py

migrations/

\_\_init\_\_.py

models.py

tests.py

views.py

This directory structure will house the poll application.

Write your first view**¶**

Let’s write the first view. Open the file polls/views.py and put the following Python code in it:

polls/views.py

**from** **django.http** **import** HttpResponse

**def** index(request):

**return** HttpResponse("Hello, world. You're at the polls index.")

This is the simplest view possible in Django. To call the view, we need to map it to a URL - and for this we need a URLconf.

To create a URLconf in the polls directory, create a file called urls.py. Your app directory should now look like:

polls/

\_\_init\_\_.py

admin.py

apps.py

migrations/

\_\_init\_\_.py

models.py

tests.py

urls.py

views.py

In the polls/urls.py file include the following code:

polls/urls.py

**from** **django.urls** **import** path

**from** **.** **import** views

urlpatterns = [

path(''^”, views.index, name='index'),

]

The next step is to point the root URLconf at the polls.urls module. In mysite/urls.py, add an import for django.urls.include and insert an include() in the urlpatterns list, so you have:

mysite/urls.py

**from** **django.urls** **import** include, path

**from** **django.contrib** **import** admin

urlpatterns = [

path('polls/', include('polls.urls')),

path('admin/', admin.site.urls),

]

The include() function allows referencing other URLconfs. Whenever Django encounters include(), it chops off whatever part of the URL matched up to that point and sends the remaining string to the included URLconf for further processing.

The idea behind include() is to make it easy to plug-and-play URLs. Since polls are in their own URLconf (polls/urls.py), they can be placed under “/polls/”, or under “/fun\_polls/”, or under “/content/polls/”, or any other path root, and the app will still work.

When to use include()

You should always use include() when you include other URL patterns. admin.site.urls is the only exception to this.

You have now wired an index view into the URLconf. Lets verify it’s working, run the following command:

**$** python manage.py runserver

Go to http://localhost:8000/polls/ in your browser, and you should see the text “*Hello, world. You’re at the polls index.*”, which you defined in the index view.

The path() function is passed four arguments, two required: route and view, and two optional: kwargs, and name. At this point, it’s worth reviewing what these arguments are for.

**path() argument: route¶**

route is a string that contains a URL pattern. When processing a request, Django starts at the first pattern in urlpatterns and makes its way down the list, comparing the requested URL against each pattern until it finds one that matches.

Patterns don’t search GET and POST parameters, or the domain name. For example, in a request to https://www.example.com/myapp/, the URLconf will look for myapp/. In a request to https://www.example.com/myapp/?page=3, the URLconf will also look for myapp/.

**path() argument: view¶**

When Django finds a matching pattern, it calls the specified view function with an HttpRequest object as the first argument and any “captured” values from the route as keyword arguments. We’ll give an example of this in a bit.

**path() argument: kwargs¶**

Arbitrary keyword arguments can be passed in a dictionary to the target view. We aren’t going to use this feature of Django in the tutorial.

**path() argument: name¶**

Naming your URL lets you refer to it unambiguously from elsewhere in Django, especially from within templates. This powerful feature allows you to make global changes to the URL patterns of your project while only touching a single file.

When you’re comfortable with the basic request and response flow, read part 2 of this tutorial to start working with the database.