Chapter 2: Getting Started

Installing Django is a multi-step process, due to the multiple moving parts in modern Web development environments. In this chapter, we'll walk you through how to install the framework and its few dependencies.

Because Django is "just" Python code, it runs anywhere Python does – including on some cell phones! But this chapter just covers the common scenarios for Django installations. We'll assume you're installing it either on a desktop/laptop machine or a server.

Later, in Chapter 12, we'll cover how to deploy Django to a production site.

Installing Python

Django itself is written purely in Python, so the first step in installing the framework is to make sure you have Python installed.

Python Versions

The core Django framework (version 1.4+) works with any Python version from 2.5 to 2.7, inclusive. Django's optional GIS (Geographic Information Systems) support requires Python 2.5 to 2.7.

If you're not sure which version of Python to install and you have complete freedom over the decision, pick the latest one in the 2.x series: version 2.7. Although Django works equally well with any version from 2.5 to 2.7, the later versions of Python have performance improvements and additional language features you might like to use in your applications. Plus, certain third-party Django add-ons that you might want to use might require a version newer than Python 2.5, so using a later version of Python keeps your options open.

> **Django and Python 3.x**
>
> At the time of writing, Python 3.3 has been released, but Django only supports it experimentally. This is because the Python 3.x series introduces a substantial number of backwards-incompatible changes to the language itself, and, as a result, many major Python libraries and frameworks, including Django (as of version 1.4), have not yet caught up.
>
> Django 1.5 is slated to support Python 2.6, 2.7, and 3.2. However, support for Python 3.2 is considered a "preview", which means the Django developers are not yet confident enough to promise stability in production. For that, they suggest you wait until Django 1.6.

Installation

If you're on Linux or Mac OS X, you probably have Python already installed. Type `python` at a command prompt (or in Applications/Utilities/Terminal, in OS X). If you see something like this, then Python is installed:

```
Python 2.7.3rc2 (default, Apr 22 2012, 22:30:17)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Otherwise, you'll need to download and install Python. It's fast and easy, and detailed instructions are available at http://www.python.org/download/

Installing Django

At any given time, two distinct versions of Django are available to you: the latest official release and the bleeding-edge development version. The version you decide to install depends on your priorities. Do you want a stable and tested version of Django, or do you want a version containing the latest features, perhaps so you can contribute to Django itself, at the expense of stability?

We'd recommend sticking with an official release, but it's important to know that the development version exists, because you'll find it mentioned in the documentation and by members of the community.

Installing an Official Release

Official releases have a version number, such as 1.4.2, 1.4.1 or 1.4, and the latest one is always available at http://www.djangoproject.com/download/.

If you're on a Linux distribution that includes a package of Django, it's a good idea to use the distributor's version. That way, you'll get security updates along with the rest of your system packages.

If you don't have access to a prepackaged version, you can download and install the framework manually. To do so, first download the tarball, which will be named something like `Django-1.4.2.tar.gz`. (It doesn't matter which local directory you download this file into; the installation process will put Django's files in the right place.) Then, unzip it and run `setup.py install`, as you do with most Python libraries.

Here's how that process looks on Unix systems:

1. `tar xzvf Django-1.4.2.tar.gz`
2. `cd Django-*`
3. `sudo python setup.py install`

On Windows, we recommend using 7-Zip (http://www.djangoproject.com/r/7zip/) to unzip `.tar.gz` files. Once you've unzipped the file, start up a DOS shell (the "Command Prompt") with administrator privileges and run the following command from within the directory whose name starts with `Django-`:

```
python setup.py install
```

In case you're curious: Django's files will be installed into your Python installation's `site-packages` directory – a directory where Python looks for third-party libraries. Usually it's in a place like `/usr/lib/python2.7/site-packages`.

Installing the "Development" Version

Django uses Git (http://git-scm.com) for its source control. The latest and greatest Django development version available from Django's official Git repository (https://github.com/django/django). You should consider installing this version if you want to work on the bleeding edge, or if you want to contribute code to Django itself.

Git is a free, open source distributed revision-control system, and the Django team uses it to manage changes to the Django codebase. You can download and install Git from http://git-scm.com/download but it is easier to install with your operating system's package manager. You can use Git to grab the very latest Django source code and, at any given time, you can update your local version of the Django code to get the latest changes and improvements made by Django developers.

When using the development version, keep in mind there's no guarantee things won't be broken at any given moment. With that said, though, some members of the Django team run production sites on the development version, so they have an incentive to keep it stable.

To grab the latest Django, follow these steps:

1. Make sure you have Git installed. You can get the software free from http://git-scm.com/, and you can find excellent documentation at http://git-scm.com/documentation.

2. Clone the repository using the command `git clone https://github.com/django/django djmaster`

3. Locate your Python installation's `site-packages` directory. Usually it's in a place like `/usr/lib/python2.7/site-packages`. If you have no idea, type this command from a command prompt:

   ```
   python -c 'import sys, pprint; pprint.pprint(sys.path)'
   ```

   The resulting output should include your `site-packages` directory.

4. Within the `site-packages` directory, create a file called `djmaster.pth` and edit it to contain the full path to your `djmaster` directory to it. For example, the file could just contain this line:

   ```
   /path/to/djmaster
   ```

5. Place `djmaster/django/bin` on your system PATH. This directory includes management utilities such as `django-admin.py`.

> **Tip:**
>
> If `.pth` files are new to you, you can learn more about them at http://www.djangoproject.com/r/python/site-module/.

After downloading from Git and following the preceding steps, there's no need to run `python setup.py install`–you've just done the work by hand!

Because the Django code changes often with bug fixes and feature additions, you'll probably want to update it every once in a while. To update the code, just run the command `git pull origin master` from within the `djmaster` directory. When you run that command, Git will contact https://github.com/django/django, determine whether any of Django's code has changed, and update your local version of the code with any changes that have been made since you last updated. It's quite slick.

Finally, if you use Django development version, you should know how to figure out which version of Django you're running. Knowing your version number is important if you ever need to reach out to the community for help, or if you submit improvements to the framework. In these cases, you should tell people the revision, also known as a "commit," that you're using. To find out your current commit, type "git log -1" from within the `django` directory, and look for the identifier after "commit". This number changes each time Django is changed, whether through a bug fix, feature addition, documentation improvement or anything else.

Testing the Django installation

For some post-installation positive feedback, take a moment to test whether the installation worked. In a command shell, change into another directory (e.g., *not* the directory that contains the `django` directory) and start the Python interactive interpreter by typing `python`. If the installation was successful, you should be able to import the module `django`:

```
>>> import django
>>> django.VERSION
(1, 4, 2, 'final', 0)
```

Interactive Interpreter Examples

The Python interactive interpreter is a command-line program that lets you write a Python program interactively. To start it, run the command `python` at the command line.

Throughout this book, we feature example Python interactive interpreter sessions. You can recognize these examples by the triple greater-than signs (`>>>`), which designate the interpreter's prompt. If you're copying examples from this book, don't copy those greater-than signs.

Multiline statements in the interactive interpreter are padded with three dots (`...`). For example:

```
>>> print """This is a
... string that spans
... three lines."""
This is a
string that spans
three lines.
>>> def my_function(value):
...     print value
>>> my_function('hello')
hello
```

Those three dots at the start of the additional lines are inserted by the Python shell – they're not part of our input. We include them here to be faithful to the actual output of the interpreter. If you copy our examples to follow along, don't copy those dots.

Setting Up a Database

At this point, you could very well begin writing a Web application with Django, because Django's only hard-and-fast prerequisite is a working Python installation. However, odds are you'll be developing a *database-driven* Web site, in which case you'll need to configure a database server.

If you just want to start playing with Django, skip ahead to the "Starting a Project" section – but keep in mind that all the examples in this book assume you have a working database set up.

Django supports four database engines:

- PostgreSQL (http://www.postgresql.org/)
- SQLite 3 (http://www.sqlite.org/)
- MySQL (http://www.mysql.com/)
- Oracle (http://www.oracle.com/)

For the most part, all the engines here work equally well with the core Django framework. (A notable exception is Django's optional GIS support, which is much more powerful with PostgreSQL than with other databases.) If you're not tied to any legacy system and have the freedom to choose a database backend, we recommend PostgreSQL, which achieves a fine balance between cost, features, speed and stability.

Setting up the database is a two-step process:

- First, you'll need to install and configure the database server itself. This process is beyond the scope of this book, but each of the four database backends has rich documentation on its Web site. (If you're on a shared hosting provider, odds are that they've set this up for you already.)
- Second, you'll need to install the Python library for your particular database backend. This is a third-party bit of code that allows Python to interface with the database. We outline the specific, per-database requirements

in the following sections.

If you're just playing around with Django and don't want to install a database server, consider using SQLite. SQLite is unique in the list of supported databases in that it doesn't require either of the above steps. It merely reads and writes its data to a single file on your filesystem, and Python versions 2.5 and higher include built-in support for it.

On Windows, obtaining database driver binaries can be frustrating. If you're eager to jump in, we recommend using Python 2.7 and its built-in support for SQLite.

Using Django with PostgreSQL

If you're using PostgreSQL, you'll need to install either the `psycopg` or `psycopg2` package from http://www.djangoproject.com/r/python-pgsql/. We recommend `psycopg2`, as it's newer, more actively developed and can be easier to install. Either way, take note of whether you're using version 1 or 2; you'll need this information later.

If you're using PostgreSQL on Windows, you can find precompiled binaries of `psycopg` at http://www.djangoproject.com/r/python-pgsql/windows/.

If you're on Linux, check whether your distribution's package-management system offers a package called "python-psycopg2", "psycopg2-python", "python-postgresql" or something similar.

Using Django with SQLite 3

You're in luck: no database-specific installation is required, because Python ships with SQLite support. Skip ahead to the next section.

Using Django with MySQL

Django requires MySQL 4.0 or above. The 3.x versions don't support nested subqueries and some other fairly standard SQL statements.

You'll also need to install the `MySQLdb` package from http://www.djangoproject.com/r/python-mysql/.

If you're on Linux, check whether your distribution's package-management system offers a package called "python-mysql", "python-mysqldb", "mysql-python" or something similar.

Using Django with Oracle

Django works with Oracle Database Server versions 9i and higher.

If you're using Oracle, you'll need to install the `cx_Oracle` library, available at http://cx-oracle.sourceforge.net/. Use version 4.3.1 or higher, but avoid version 5.0 due to a bug in that version of the driver. Version 5.0.1 resolved the bug, however, so you can choose a higher version as well.

Using Django Without a Database

As mentioned earlier, Django doesn't actually require a database. If you just want to use it to serve dynamic pages that don't hit a database, that's perfectly fine.

With that said, bear in mind that some of the extra tools bundled with Django *do* require a database, so if you choose not to use a database, you'll miss out on those features. (We highlight these features throughout this book.)

Starting a Project

Once you've installed Python, Django and (optionally) your database server/library, you can take the first step in developing a Django application by creating a *project*.

A project is a collection of settings for an instance of Django, including database configuration, Django-specific options and application-specific settings.

If this is your first time using Django, you'll have to take care of some initial setup. Create a new directory to start working in, perhaps something like `/home/username/djcode/`.

> **Where Should This Directory Live?**
>
> If your background is in PHP, 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 in doing so you risk the possibility that people will be able to view your raw source code over the Web. That's not good.
>
> Put your code in some directory **outside** of the document root.

Change into the directory you created, and run the command `django-admin.py startproject mysite`. This will create a `mysite` directory in your current directory.

> **Note**
>
> `django-admin.py` should be on your system path if you installed Django via its `setup.py` utility.
>
> If you're using the development version, you'll find `django-admin.py` in `djmaster/django/bin`. Because you'll be using `django-admin.py` often, consider adding it to your system path. On Unix, you can do so by symlinking from `/usr/local/bin`, using a command such as `sudo ln -s /path/to/django/bin/django-admin.py /usr/local/bin/django-admin.py`. On Windows, you'll need to update your `PATH` environment variable.
>
> If you installed Django from a packaged version for your Linux distribution, `django-admin.py` might be called `django-admin` instead.

If you see a "permission denied" message when running `django-admin.py startproject`, you'll need to change the file's permissions. To do this, navigate to the directory where `django-admin.py` is installed (e.g., `cd /usr/local/bin`) and run the command `chmod +x django-admin.py`.

The `startproject` command creates a directory containing five files:

```
mysite/
    manage.py
    mysite/
        __init__.py
        settings.py
        urls.py
        wsgi.py
```

> **Note**
>
> Doesn't match what you see?
>
> The default project layout recently changed. If you're seeing a "flat" layout (with no inner `mysite/` directory), you're probably using a version of Django that doesn't match this tutorial version. This book covers Django 1.4 and above, so if you're using an older version you probably want to consult Django's official documentation.

The documentation for Django 1.X version is available at https://docs.djangoproject.com/en/1.X/.

These files are as follows:

- `mysite/`: The outer `mysite/` 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. Type `python manage.py help` to get a feel for what it can do. You should never have to edit this file; it's created in this directory purely for convenience.
- `mysite/mysite/`: 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. `import mysite.settings`).
- `__init__.py`: A file required for Python to treat the `mysite` directory as a package (i.e., a group of Python modules). It's an empty file, and generally you won't add anything to it.
- `settings.py`: Settings/configuration for this Django project. Take a look at it to get an idea of the types of settings available, along with their default values.
- `urls.py`: The URLs for this Django project. Think of this as the "table of contents" of your Django-powered site.
- `wsgi.py`: An entry-point for WSGI-compatible webservers to serve your project. See How to deploy with WSGI (https://docs.djangoproject.com/en/1.4/howto/deployment/wsgi/) for more details.

Despite their small size, these files already constitute a working Django application.

Running the Development Server

For some more post-installation positive feedback, let's run the Django development server to see our barebones application in action.

The Django development server (also called the "runserver" after the command that launches it) is a built-in, lightweight Web server you can use while developing your site. It's included with Django so you can develop your site rapidly, without having to deal with configuring your production server (e.g., Apache) until you're ready for production. The development server watches your code and automatically reloads it, making it easy for you to change your code without needing to restart anything.

To start the server, change into your project container directory (`cd mysite`), if you haven't already, and run this command:

```
python manage.py runserver
```

You'll see something like this:

```
Validating models...
0 errors found.

Django version 1.4.2, using settings 'mysite.settings'
Development server is running at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

This launches the server locally, on port 8000, accessible only to connections from your own computer. Now that it's running, visit http://127.0.0.1:8000/ with your Web browser. You might see a different Django version depending on which version of Django you have installed. You'll see a "Welcome to Django" page shaded in a pleasant pastel blue. It worked!

One final, important note about the development server is worth mentioning before proceeding. Although this

server is convenient for development, resist the temptation to use it in anything resembling a production environment. The development server can handle only a single request at a time reliably, and it has not gone through a security audit of any sort. When the time comes to launch your site, see Chapter 12 for information on how to deploy Django.

Changing the Development Server's Host or Port

By default, the `runserver` command starts the development server on port 8000, listening only for local connections. If you want to change the server's port, pass it as a command-line argument:

```
python manage.py runserver 8080
```

By specifying an IP address, you can tell the server to allow non-local connections. This is especially helpful if you'd like to share a development site with other members of your team. The IP address `0.0.0.0` tells the server to listen on any network interface:

```
python manage.py runserver 0.0.0.0:8000
```

When you've done this, other computers on your local network will be able to view your Django site by visiting your IP address in their Web browsers, e.g., http://192.168.1.103:8000/ . (Note that you'll have to consult your network settings to determine your IP address on the local network. Unix users, try running "ifconfig" in a command prompt to get this information. Windows users, try "ipconfig".)

What's Next?

Now that you have everything installed and the development server running, you're ready to :doc: learn the basics Chapter 3, of serving Web pages with Django.

Copyright Adrian Holovaty, Jacob Kaplan-Moss, et al.
This work is licensed under the GNU Free Document License.

« previous ◇ table of contents ◇ next »