Django Meetup Documentation

Release 0.0.1

DjangoMeetup.com

Jan 31, 2019

CONTENTS

Django Meetup is an open source project designed to provide a working model for Meetup group members and other Django enthusiasts. This guide takes you through the process of setting up the public website from the Django Meetup github repository.

Note that the information provided in the official Django documentation is far more comprehensive. However, this provides some additional practical understanding of what is going on when setting up this project.

The understanding is important because due to various permuations, such as different operating systems or different version numbers, you may encounter some errors and issues along the way (it happens often!). Understanding the why to what you're trying to do can help solve the problem, sometimes.

At worst, if you encounter a problem you're unable to solve, you can simply delete all the folders created below, or one of either the environment or public website folders, and then start again.

While a large component of Django users use Linux/Mac, many new users come to Django via Windows. This guide tries to document instructions for all those OS variants.

CHAPTER

ONE

CONTENTS:

1.1 Quick Installation

For those already comfortable with the process, this section provides a skeleton of the steps involved.

Note, you would of course need to insert the name of your git repository when you see **<yourrepo>**. Also, these steps use github's HTTPS cloning method (as opposed to SSH).

If you prefer more detail, you can follow the full instructions in the subsequent sections.

1.1.1 On Linux/Mac:

1.1.1.1 Download Project

Create your project directory: `mkdir DjangoMeetup`

Change to project directory: `cd DjangoMeetup`

Fork the project from: `https://github.com/DjangoMeetup/public-website`

Clone the project: `git clone https://github.com/<yourrepo>/public-website.git`

Add parent repo: `git remote add upstream https://github.com/DjangoMeetup/public-website.git`

1.1.1.2 Virtual Environment

In the same directory, create a virtual environment: `python -m venv env`

Activate your environment: `source env/bin/activate`

1.1.1.3 Load Packages

Install packages: `pip install requirements/dev.txt`

1.1.1.4 Migrate Database

Run migrations: `python manage.py makemigrations`

Run migrate: `python manage.py migrate`

1.1.1.5 Environment Variables

Create the environent file **.env** and add:

```
DEBUG=True
DOMAIN=localhost
SECRET_KEY=-#^op)191*7@$4qthsxjjs7dl1*-@1$111^je_@@&3h9&ipe#w
GOOGLE_RECAPTCHA_SECRET_KEY= 6LexSoNNCCCCCCEt_8VpyiaubPwb48LLq21wmp4Mr
EMAIL_HOST=smtp.gmail.com
EMAIL_HOST_USER=admin@djangomeetup.com
EMAIL_HOST_PASSWORD=safeandsecure
```

Change SECRET_KEY and GOOGLE_RECAPTCHA_SECRET_KEY to your own keys

1.1.1.6 Run Server

Start the server: `python manage.py runserver`

1.1.2 On Windows

Windows commands are largely the same as for Linux and Macs. In fact, the differences become nearly negligible if you download and use the excellent command line tool Git Bash If you do use Git Bash, you'll have to include the source command when you activate the virtual environment, ie. `source env/Sripts/activate`.

However, if using the Windows command line tool, then users should bear these in mind:

1. you only need type `py` instead of the full `python`.
2. you must use back slashes for command line file paths.
3. virtual environment activate folder is kept in a Scripts folder (vs bin for LinuxMac)
4. virtual environment activation does not require the source command, ie. it will suffice to use `env/Sripts/activate`
5. file paths are not case sensitive.

Here are the steps restated for Windows users:

1.1.2.1 Download Project

Create your project directory: `mkdir DjangoMeetup`

Change to project directory: `cd DjangoMeetup`

Fork the project from: `https://github.com/DjangoMeetup/public-website`

Clone the project: `git clone https://github.com/<yourrepo>/public-website.git`

Add parent repo: `git remote add upstream https://github.com/DjangoMeetup/public-website.git`

1.1.2.2 Virtual Environment

In the same directory, create a virtual environment: `py -m venv env`

Activate your environment: `env\scripts\activate`

1.1.2.3 Load Packages

Install packages: `pip install requirements\dev.txt`

1.1.2.4 Migrate Database

Run migrations: `py manage.py makemigrations`

Run migrate: `py manage.py migrate`

1.1.2.5 Environment Variables

Create the environent file **.env** and add:

```
DEBUG=True
DOMAIN=localhost
SECRET_KEY=-#^op)l91*7@$4qthsxjjs7dl1*-@1$l11^je_@@&3h9&ipe#w
GOOGLE_RECAPTCHA_SECRET_KEY= 6LexSoNNCCCCCEt_8VpyiaubPwb48LLq21wmp4Mr
EMAIL_HOST=smtp.gmail.com
EMAIL_HOST_USER=admin@djangomeetup.com
EMAIL_HOST_PASSWORD=safeandsecure
```

Change SECRET_KEY and GOOGLE_RECAPTCHA_SECRET_KEY to your own keys

1.1.2.6 Run Server

Start the server: `py manage.py runserver`

1.2 Download Project

1.2.1 Create A Project Folder

Create a DjangoMeetup folder, preferably near the top level user folder on your computer. Here's a couple of ideas:

```
On Linux/Mac:
~/projects/djangomeetup

On Windows:
C:\Users\Owner\Projects\DjangoMeetup
```

As a heads up to where we're going, in this folder we are going to have:

(i) the project files downloaded from the Git repository (called **public-website**).

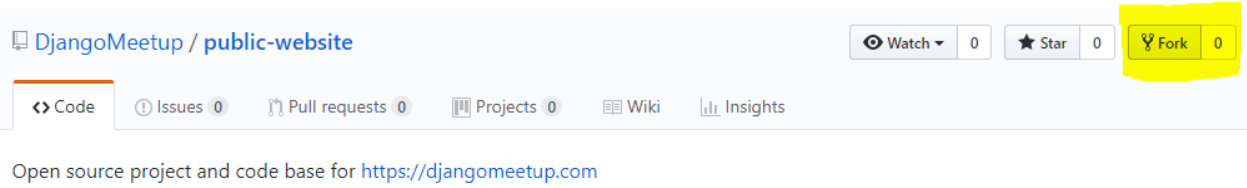
This folder will be referred to as the project repository folder, because it contains the .git folder.

(ii) a virtual environment, to allow future environment replication (in this guide it will be called **env**)

1.2.2 Get a Copy of The Project

1.2.2.1 Fork the Project

Go to your browser and log in to your own GitHub account. Then head to the DjangoMeetup repo and fork it.

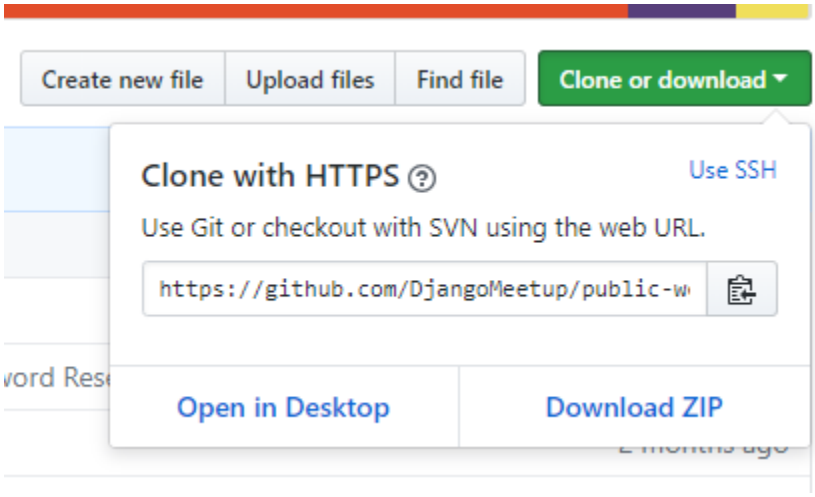


After you've done that, head back to your own GitHub account where you'll see that it's been forked underneath the project name public-website.

We now need to clone that repo to the project folder on our computer. We can do this either using HTTPS (simple), or using SSH way (slightly more involved). Note, you only have to choose one of these approaches.

1.2.2.2 Clone to Your Computer – HTTPS

If you chose the HTTPS method, then you need to clone the repo address using HTTPS, which you can get from your repo, as per image below (note that your github name will appear instead of DjangoMeetup).



Then use your command line tool to run the clone command (replace **USERNAME** with your github name):

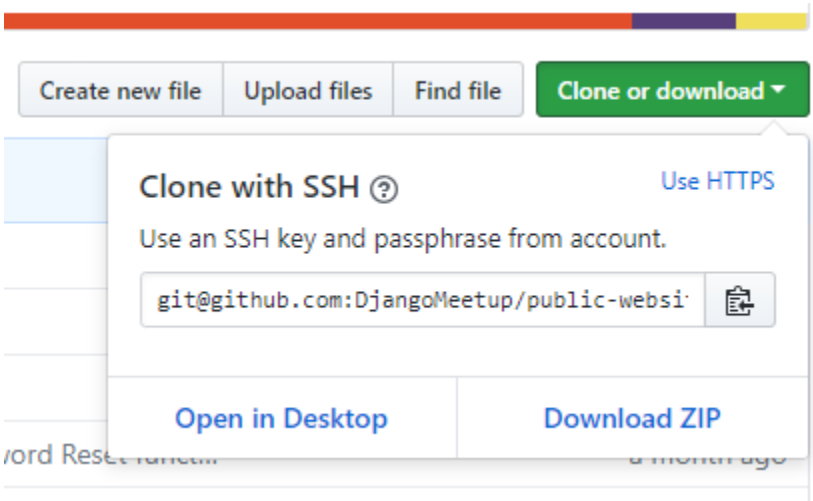
```
git clone https://github.com/**USERNAME**/public-website.git
```

If successful, this should have created a public-website folder within your DjangoMeetup folder.

1.2.2.3 Clone to Your Computer – SSH

If you chose the HTTPS way above, you can exclude this step. Otherwise, if you prefer to use the security of SSH, then use this approach.

Note that it essentially replicates the HTTPS way, except that we need to add an SSH key.



Also, the remote links are different between HTTPS and SSH, as shown here (replace **USERNAME** with your github name):

```
HTTPS:
https://github.com/USERNAME/REPOSITORY.git

SSH:
git@github.com:USERNAME/REPOSITORY.git
```

1.2.2.3.1 Steps to Create SSH Key

The SSH key allows you to connect to GitHub without supplying your username or password at each visit. These keys are kept in your ~/.ssh folder (or the .ssh folder in your home directory on Windows).

If you don't already have SSH keys in your.ssh folder, GitHub provides an overview of creating SSH keys here: https://help.github.com/articles/connecting-to-github-with-ssh/

As a general guide, here are the required steps.

(i) Generate the key pair (with a secure pass phrase).

If you use the RSA protocol, these typically default to two files called:

a. id_rsa (the private key, with a secure password)

b. id_rsa.pub (the public key)

(ii) Start the ssh-agent on your PC, and add the private key to your .ssh folder

(iii) Add the public key to your GitHub account

You do this by clicking on your profile pic, choosing SSH and GPS keys, New SSH, and then pasting in the contents of the public key file.

1.2.2.3.2 Clone via SSH

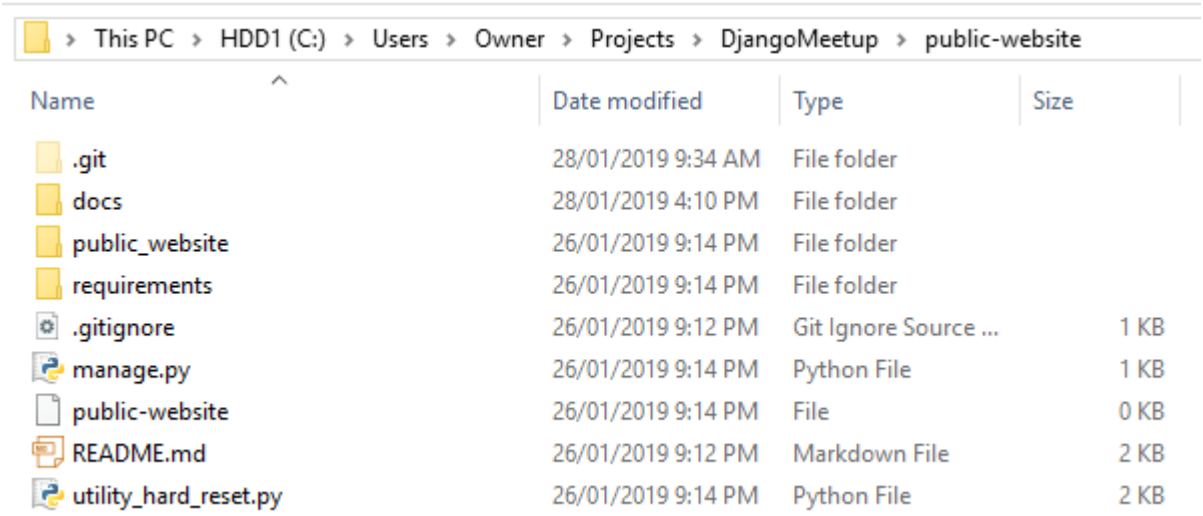
First we need to clone the repo address using SSH, which you can get from the repo, as per image below (note that your github name will appear instead of DjangoMeetup).

Then use your command line tool to run the full clone command (replace **USERNAME** with your github name):

```
git clone git@github.com:USERNAME/public-website.git
```

1.2.2.4 If Clone was Successful

Within the public-website folder, you now should see a structure similar to the folder in the image (this is taken from Windows, but should be the same structure on Linux/Mac).



Note that on Windows I have enabled View, Hidden Items. This allows me to see the files with a dot prefix.

By way of clarifying terminology, what we see are the contents of the project repository root folder (ie the top **public-website** folder). This contains the project root folder (ie. the bottom **public-website** folder).

Within that folder are the:

- project root folder, which as mentioned above is the bottom **public-website** folder
- docs folder, which contains the source code for this guide
- manage.py file, for running django commands
- requirements folder, which contains package requirements
- .git repository
- .gitignore file, which specifies which files to keep out of git
- README.md file for the project
- utility_hard_reset.py tool for resetting the database

1.2.2.5 Add Parent Repo Address To Git

Ok, now lets account for modifications made to the project parent, which we will want to bring to our local copy. In order to do that, we need to add the parent repository (albeit read-only).

Note that we'll want to use the DjangoMeetup repo this time. So run the following command to add the parent (or upstream) remote location:

```
With HTTPS:
git remote add upstream https://github.com/DjangoMeetup/public-website.git

With SSH:
git remote add upstream git@github.com:DjangoMeetup/public-website.git
```

This means we now have two git remotes for our local repository. We can check that by running:

```
git remote
```

And if you want to see the remotes and the addresses they point to, run:

```
git remote -v
```

This should return the remote shortcut names of origin and parent (and addresses if you used the latter command).

1.2.2.6 Update from Parent

If at a later stage the upstream repository is changed and we want to bring those changes into our local repository, we can then update our local repo by running:

```
git pull upstream master
```

1.3 Virtual Environment

1.3.1 Create the Virtual Environment

Each time we start a new project, we want a specific environment to be associated with that project. That way, the project's dependencies can easily be modified, and we can quickly switch to other projects that have different dependencies. This includes the python version, as well as the various django and 3rd party packages.

You can store the virtual environment wherever you like. But it should be easily identifiable, and neatly isolated from all other projects.

For example, you could have a folder solely dedicated for virtual environments, and another folder dedicated for your projects.

For simplicity, what we'll do is store the virtual environment within the project itself. That way, the environment + all project files will be kept in the same folder.

So let's create a new virtual environment inside the DjangoMeetup folder, at the same level as the repository folder (ie. the same level as the top **public-website** folder).

Versions of Python before 3.3 used pip or pyvenv + third party tools to create virtual environments. However, Python 3.3 and versions since have an in-built venv module, which is recommended. So we'll assume you're using venv.

You can use any name you choose for the virtual environment, but typical names include the likes of venv , env, or venv_projectname. Here we'll just use **env** to help distinguish from the venv command.

Make sure your command line is in the correct folder, then run the following command to create the virtual environment env:

```
On Linux/Mac:
python -m venv env

On Windows:
py -m venv env
```

Because the virtual environment folder **env** is above the git repository in the public-website folder, its kept out of the git source control. That's because the environment is for your own setup, and including it in source control would just be bloat.

Within the DjangoMeetup folder, we should now just have the two folders:

1. the env folder
2. the public-website folder (ie. the git repository folder)

1.3.2 Activate & Deactivate Virtual Environment

We then activate the virtual environment. In Linux and Mac, you use the source command to run the activate file in the bin folder. In Windows, you simply run the activate.bat file in the Scripts folder.

Note that the venv command you ran earlier creates a bin folder in Linux/Mac and a Scripts folder in Windows, so your path to the activate file will obviously differ depending on your OS.

For PowerShell users, see the note below.

In Windows, you can also use Git Bash, which mimics the Linux and Mac commands. In which case, you would also use the source command.

```
On Linx & Mac:
source env/bin/activate

On Windows:
env\Scripts\activate

On Windows Git Bash
source env/Scripts/activate
```

After you've run activate you'll be reminded that you're in the virtual environment, because at the beginning of your prompt you will now see the environment name in parentheses. For example, the **(env)** in the following:

```
On Linux/Mac:
(env) ~/DjangoMeetup/ $

On Windows:
(env) C:\Owner\Projects\DjangoMeetup >
```

This means the Python you now execute will be from the virtual environment, as are all packages (and package installations will go to that virtual environment, rather than the system) To deactivate the environment, you simply use the command:

```
deactivate
```

Note that if you are using Windows PowerShell, you might run into restrictions trying to utilize the activation script and will have to first run the command below as an administrator. This allows the activation script to be run since it was signed locally.

```
Set-ExecutionPolicy RemoteSigned
```

1.4 Load Packages

Now that we have the environment basics set up, we can add the packages required for the project.

Packages are a collection of libraries, with each library containing pre-written code typically for some specific use. The Python Package Manager bundles these libraries together in a package, which makes it easier to download the entire package, rather than each library seperately.

We could add the packages individually, one-by-one, using the following pip command:

```
pip install package-name
```

However, its smarter to use the dependency management.

1.4.1 Dependency Management

The repo includes requirements files in the requirements folders, namely the **base.txt**, **dev.txt** and **prod.txt**. If you inspect each file you'll see the the packages and their version numbers required. Note that the dev and prod file already include a link to the base file.

We're only concerned with development at this stage. So install the required dev & base packages via the following command:

```
    On Linux & Mac:
pip install requirements/dev.txt

    On Windows:
pip install requirements\dev.txt
```

Note we referred to the nested folder where the requirements are contained. You could have changed directories into the requirements folder and achieved the same result via the following command:

```
pip install dev.txt
```

1.5 Migrate Database

After cloning the project, we need to create the equivalent tables in our database. Note that the database is deliberately exclude from the GitHub source control, via an entry of *.sqlite3 in the .gitignore file. That way, each developer builds the database from scratch and and then works with a clean set of tables for their development purposes.

First though, a good practice is to first run the makemigrations command before you make changes to the database. This doesn't touch the database, but instead creates a trail of python code that replicates the instructions required to be made. In turn, this can be used later to roll back or recreate model changes, and investigate what took place.

After having run the migrations command, we then run the migrate command. This does affect the database, and the first time we run it, it migrates the complete data structure by replicating the models as tables in the database. Thereafter the migrate command updates the database with the latest changes.

So, lets make migrations and migrate the models via the following:

```
On Linux/Mac:
python manage.py makemigrations
python manage.py migrate

On Windows:
py manage.py makemigrations
py manage.py migrate
```

1.6 Environment Variables

The settings file uses python-decouple to read in environment variables from a .env file. However, this file contains sensitive password data, so it is deliberately excluded from the public git repo (via the .gitignore file). Not also, that because the file name is preceded by a dot ('.') it becomes a hidden file.

So we need to create our own .env file. For initial purposes of getting the web project up and running, we can use dummy variables to provide you with an idea of the type of variables required.

In the public-website folder, we create that .env file, and then save the following into it.

```
DEBUG=True
DOMAIN=localhost
SECRET_KEY=-#^op)191*7@$4qthsxjjs7dl1*-@1$111^je_@@&3h9&ipe#w
GOOGLE_RECAPTCHA_SECRET_KEY= 6LexSoNNCCCCCCEt_8VpyiaubPwb48LLq21wmp4Mr
EMAIL_HOST=smtp.gmail.com
EMAIL_HOST_USER=admin@djangomeetup.com
EMAIL_HOST_PASSWORD=safeandsecure
```

This would allow the basic front page of the web site to get up and running. However, you'll still need to tailor these variables for your own usage. In particular, you'll want to use your own **SECRET_KEY**. And in the case of the **GOOGLE_RECAPTCHA_SECRET_KEY**, you'll have to get one from Google otherwise your captcha won't work.

So lets modify these variables to provide your own. Here's a rundown on how to do that.

1.6.1 Secret Key

Your secret key is variously used for hashing tokens for user sessions, password resets, and other types of encryption. Its important to keep your's secure, and out of version control. For development purposes, generate your own secret key. You can create one either by django's inbuilt function, or by using a helpful website.

To use the Django Inbuilt function, run the following command:

```
On Linux/Mac:
python3 manage.py shell -c "from django.core.management.utils import get_random_
↪secret_key; print(get_random_secret_key())"

On Windows:
py manage.py shell -c "from django.core.management.utils import get_random_secret_key;
↪ print(get_random_secret_key())"
```

Alternatively you can generate a django secret key from other places, such as this website: https://www.miniwebtool.com/django-secret-key-generator/

1.6.2 Google Recaptcha Secret Key

The Django Meetup website uses Google's reCAPTCHA v2 version. You will need to sign up for an API key pair for the site, and register your localhost.

Head over to https://www.google.com/recaptcha/admin#list and register your localhost address after choosing a re-CAPTCHA version 2 tickbox type.

For the label, use localhost, and for the domain, use 127.0.0.1. Then add an email address of your choice.

The admin site then should provide you with your site key and your secret key.

The site key is added to the HTML in the widget file. The file address for that in mine was: CAN WE ADD A settings.RECAPTCHA_SITE_KEY AS A VIEWS CONTEXT VARIABLE?

```
/djangomeetup/public-website/apps/formality/templates/formality/widget_recaptcha.html

/env/bin/public-website/apps/formality/templates/formality/widget_recaptcha.html
```

The secret key is added to your .env file, as shown above, and should be kept away from git source control, and anyone else's eyes.

If you want a tutorial on the process, this provides some useful information: https://simpleisbetterthancomplex.com/tutorial/2017/02/21/how-to-add-recaptcha-to-django-site.html

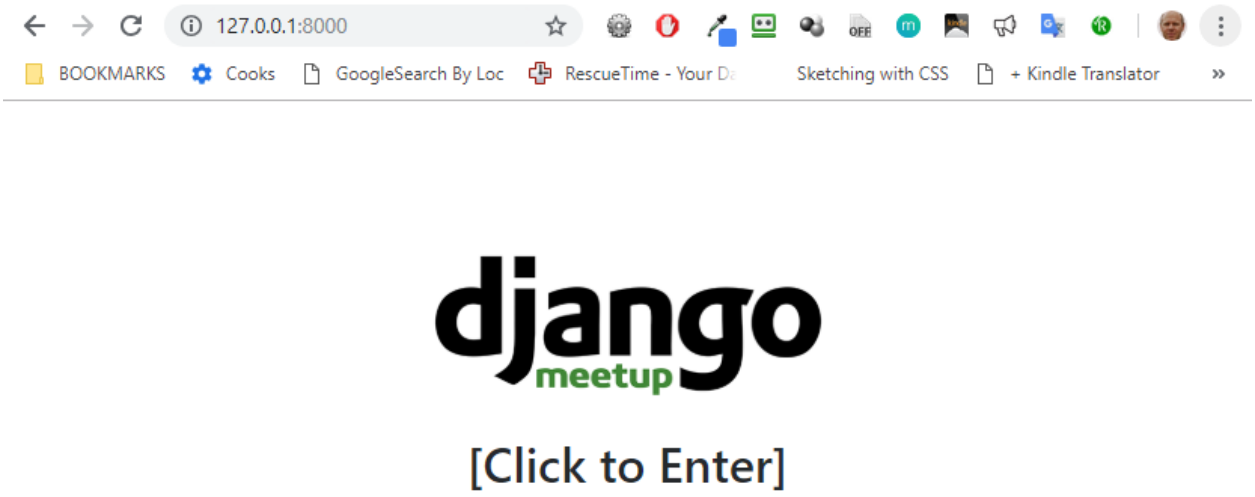
1.7 Run The Server

Now we're ready to serve the website by initiating the built-in python server via the following command:

```
On Linux/Mac:
python manage.py runserver

On Windows:
py manage.py runserver
```

When you go to the localhost page, ie. http://127.0.0.1:8000/, you should now see the public website for Django Meetup working on your local server.



1.8 Appendix 1: Python Installation

This guide assumes you have Python 3.X installed. If you are still using Python 2.X, you may encounter some issues. And while you can use the word python to run commands in both versions, if you have both versions installed you can avoid ambiguity by using python3 as the command.

If you've not got Python installed, head over to the Python release pages and download from there. At time of writing, Python 3.7 would have been the latest stable version to use. If you want to check your Python version, open your command line tool (eg. Bash, Git Bash for Windows, PowerShell, or just the plain old Windows command line), and run the following command:

```
On Linux/Mac:
python -V

On Windows:
py -V
```

Where py is the built-in Python launcher for Python3 on Windows.

Note also that the commands are to be pasted in after the terminal prompt sign. In Linux, the default prompt is $. In Windows the default is >.

1.8.1 Flags

A quick word about the command flags generally used with python commands. For example, when you type python -m venv, the -m indicates that a module name follows, so there's no need to use the .py suffix.

You can see all the flags by typing

```
python --help
```

Alternatively you can look at the docs here: https://docs.python.org/3.7/using/cmdline.html

1.9 Appendix 2: Debugging Tricks

There are several approaches you can use to help you in debugging. Here are some of the more standard tricks.

1.9.1 Use The Debug Toolbar

The Django Meetup model comes with the django debug toolbar in dev mode. So use that.

1.9.2 Evaluate Variables In Settings File

If for some reason the settings file doesn't seem to be picking up the variables, you can use this approach to help determine what's going on.

Simply introduce a print() statement into the settings file, with the name of the environment variable you want to inspect.

Of course, you'll have to introduce it after the environment variable has been called.

For example, in the settings file, a print statement could appear as following:

```
GOOGLE_RECAPTCHA_SECRET_KEY = config('GOOGLE_RECAPTCHA_SECRET_KEY')
print('RECAPTCHA KEY: ', GOOGLE_RECAPTCHA_SECRET_KEY)
```

Then when you execute the runserver command, you will see this variable printed out among the debugging feedback.

1.10 Appendix 4: Sphinx Help

1.10.1 Using Sphinx

1.10.2 PDF Creation

1.10.2.1 Install Mitex

Download Mitex from mitex.org

Run installer and auto install missing packages.

This will create an app called TeXworks.

Open up TeXworks.

1.10.2.2 Create the PDF

Back in terminal, go to folder with Sphinx makefile and run:

./make.bat latexpdf

This creates a folder called latex in your build folder

Within it, Then in the documents/latext there is a file called DjangoMeetup.tex This is the raw latex file.

Open this file in TeXworks. Click run. It should produce the finished pdf report in the latext folder.

1.11 Appendix 3: reStructuredText Help

the-reference

This section provides an overview of the style and reStructuredText (rST) used in this document. More comprehensive guides exist for rST, such as the rST Specifications or Sphinx docs. But here we provide just a quick overview and reference for the typical usage found in this guide.

rST is somewhat similar to Markdown, but it has more features. However, rST can be more complex. In particular, the header structure is somewhat more confusing.

You can compare the guide's source code with the rendered output to help get a better idea. Otherwise online live-rendering tools are useful for practicing. For example, you could copy the source for this page and take it over to rst.ninjs.org.

Tip: Alternatively, you can examine the source code for other projects to see their use of rST, such as:

Pylons

ReadTheDocs - Pylons

Source code on GitHub - Pylons

Datasette

ReadTheDocs - Datasette

Source code on GitHub - Datasette

1.11.1 Headers

In rST, there is no set pattern for creating headers. Instead, the header patterns are essentially derived from the order. So the first style encountered will be an outermost title (like HTML H1), the second style will be a subtitle, the third will be a subsubtitle, and so on.

This automatic creation of header styles can make it a bit confusing to understand what header you're actually typing. And if you don't understand that, you can get weird results.

So this guide tends to use this approach to the headers:

- a Title (or H1) pattern for the top of the page
- a Section (or H2) pattern to mark out sections
- a Subsection (or H3) pattern to mark out subsections
- a Sub-subsection (or H4) pattern to mark out sub-subsections
- etc

Now that we know the order is important, we can realise that the symbols used are less important.

That said, we use the following convention to help standardise this guide.

Note: The symbols used in this convention are not important for the output. Instead, they provide a quick understanding for collaborative guide writers as to what was intended.

The length of the underline must be at least as long as the title itself. If an under and overline are used, their length must be identical (eg. for the H1).

To standardise the source code in this guide, we use a length of **50 chars** for the underlines. A header exceeding that length should be rare.

```
==================================================
Heading 1 - for Page Titles (ie. "=" above & below)
==================================================`

Heading 2 - for Section Marking
==================================================

Heading 3 - for
--------------------------------------------------

This is a heading 4
..................................................

This is a heading 5
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

1.11.2 Paragraphs

Paragraphs are simply written as plain text. If you write on the next line, it is automatically added to that paragraph.

To create a new paragraph just add a line space between the sentences.

1.11.3 Indentation

Indentation can be used to indicate block quotes, definitions (in definition list items), and local nested content:

Here is some text

> **Here is some indented text.** Then we indent it some more

Then we remove it all

1.11.4 Inline markup

Inline markup features used most often are italics, bold, and code.

```
*Italic text* is surrounded by 1x asterisk.
**Bold text** is surrounded by 2x asterisks.
`Interpreted text` is surrounded by 1x backtick.
``inline code`` is surrounded by 2x backticks.
```

1.11.5 Separators

A horizontal line can be created by using 4 or more "-" in a row.

```
----
```

This renders like html's <hr>.

1.11.6 Literal Blocks

A paragraph consisting of two colons "::" signifies that the following text block(s) comprise a literal block. The literal block must either be indented or quoted. No markup processing is done within a literal block. It is left as-is, and is typically rendered in a monospaced typeface:

Here is an example of literal blocks:

```
::

  Two colons are added,then a blank line
  Then the text is indented, and it will print out in monospace.
  Each additional line must also be indented.
  When you want to stop the literal block, just un-indent a new line.
```

Literal blocks can be used to quickly render code:

```
::

  py -m venv env
```

1.11.7 Inline Code

Inline code snippets are written with double backticks before and after the text. For example: We then run the command: ``$ python -m venv env``

This renders as:

We then run the command: `$ python -m venv env`

1.11.8 Code-Blocks

Code-blocks can also be used to render different coding languages. To do that, use the format:

```
.. code-block:: languagename

For example:
.. code-block:: html
.. code-block:: sql
.. code-block:: python
```

And to add line numbers, you can use `:linenos:`:

```
.. code-block:: html
   :linenos:
```

Here's an a code-block formatted with html, with numbers

```
.. code-block:: html
   :linenos:

   <h1>Code Block with Numbers</h1>
   <p>some more code</p>
```

This would render as follows:

```
1  <h1>Code Block with Numbers</h1>
2  <p>some more code</p>
```

Here's a code-block formatted for sql, without numbers

```
.. code-block:: sql

   SELECT * FROM TABLE django_auth
```

Here's a code-block using specific emphasis

```
.. code-block:: python
   :emphasize-lines: 3,5

   def some_function():
       interesting = False
       print 'This line is highlighted.'
       print 'This one is not...'
       print '...but this one is.'
```

This renders as:

```
def some_function():
    interesting = False
    print 'This line is highlighted.'
    print 'This one is not...'
    print '...but this one is.'
```

1.11.9 Links

Links can be written simply in the normal form, as shown here:

```
http://djangomeetup.com
```

However if you want the link to be replaced by text, add some backticks, the text, angle brackets around the link, and an underscore at the end:

```
`Django Meetup <http://djangomeetup.com>`_
```

This renders as Django Meetup

We can thereafter use the reference text to repeatedly refer to that link. We do this by using that same text, surrounded by backticks and a final underscore.

```
`Django Meetup`_
```

This then renders as Django Meetup

1.11.10 Bullet Lists

Bullet lists can be written with a "*", "-", or "+". New lines can be added below each list point, but keep the indenting the same to keep it matched with that point.

```
Bullet List 1
---------------------------------------------
* A bulleted list item.
  You can add a new line, but make sure to indent it
* Second list item.

Bullet List 2
---------------------------------------------
- A bulleted list item.
- Second list item.
```

1.11.11 Enumerated Lists

Enumerated lists use the "#." format. Here are some examples:

```
Enumerated List 1
---------------------------------------------
#. This is a numbered list.
#. It has three items.
#. The last item.
```

You can nest lists, but you must leave a blank line above the first nested list point:

```
Enumerated List 2 - Nested
---------------------------------------------
#. This is a numbered list.
 * this is a nested sub-point
 * this is another sub-point
#. It has three items.

* this is a nested sub-point
#. The last item.

   * this is a nested sub-point
```

The numbered list renders as follows:

1. This is a numbered list
   - this is a nested sub-point
   - this is another sub-point
2. It has three items
   - this is a nested sub-point
3. The last item.
   - this is a nested sub-point

1.11.12 Images

Graphics can be in either the form images, or figures. For example, here's how an image may be placed:

```
.. image:: _static/images/logo.png
```

A figure (a graphic with a caption) may placed like this:

```
.. figure:: _static/images/directory-structure.png

   Directory Structure
```

1.11.13 Directives

This part is a reference.