django

Installation Tutorial
By James Denton
STEP I : PREPARATION

i. Make a home for Django
   SSH into SILO, from root ~/ type:

   ```
   mkdir -p ~/lib/python2.6/site-packages
   ```

ii. Edit / Pico your bash files, while still at root ~/ (pico .bashrc)
   To .bashrc add these lines at the bottom:
   ```
   module load python/2.6
   alias python=python2.6
   ```

   To .bash_profile add these lines:
   ```
   export PATH=$PATH:$HOME/local/lib/python2.X/site-packages/django/bin
   export PYTHONPATH=$PYTHONPATH:$HOME/local/lib/python2.X/site-packages
   ```

iii. Source the changes you made to the bash files.
   ```
   source ~/.bash_profile
   source ~/.bashrc
   ```

iv. Test the path changes, type `python` to start the python interpreter.
   Note the version it lists as it starts, it should be 2.6 or higher.
STEP II : INSTALL

- Move into the directory we created in Step I:
  
  ```
  cd ~/lib/python2.6/site-packages/
  ```

- Download the latest svn of Django, by typing:
  
  ```
  svn export http://code.djangoproject.com/svn/django/trunk/django django
  ```
  
  Note: you can add `--force` just after export in the above command to update Django in the future.

- Copy the `django-admin.py` from `django/bin` to root’s bin `~/bin`
  
  ```
  cp ~/lib/python2.6/site-packages/django/bin/django-admin.py ~/bin/
  ```

- Test your Install:
  
  - Type `python` to start the interpreter.
  - Type `import django`
  
  Note: This command will fail if your install was unsuccessful.
STEP III: Creating a Project

- Return to root ~/ and create your new project folder.
  ```
  mkdir ~/project
  cd ~/project
  ```

- Create your project, type: `django-admin.py startproject mysite`

- Create three more folders, for organization of file types:
  ```
  mkdir ~/project/media ~/project/scripts ~/project/templates
  ```

- Create a symbolic link to connect python files to our project.
  ```
  cd ~/lib/python2.6/site-packages
  ln -s ~/project/mysite
  ```

- To Test your Link, Return to Root ~/ and then type `python` to start up the interpreter. Then `import mysite` which will fail if the project was created incorrectly.
Step IV: The Development Server

- To run the development server, type:
  ```python
  python manage.py runserver silo.cs.indiana.edu:####
  ```
- Where #### is the extra port assigned to you, which is listed on the students and ports page, linked on the class website.
- To Test, in a browser, go to the address: silo.cs.indiana.edu:####
- You should see:

  ![It worked! Congratulations on your first Django-powered page.]

  Of course, you haven't actually done any work yet. Here's what to do next:
  
  - If you plan to use a database, edit the DATABASE_* settings in mysite/settings.py.
  - Start your first app by running python mysite/manage.py startapp [appname].

  You're seeing this message because you have DEBUG = True in your Django settings file and you haven't configured any URLs. Get to work!

- Note: The ‘server’ only exists while this process is running, but you can always open a new ssh and leave this in the background. Ctrl + C to exit the development server.
Step V: Database Setup

- Note: These steps assume you have set up a mysql database. Django will need a database of its own, so we must create one for its use. DO NOT start sql if it’s already running, you can confirm this by checking active processes:
  ```
  ps -ef | grep username
  ```

- Start up SQL, for me this is simply:  
  ```
  ~/bin/mysql_start
  ```
  (If you haven’t setup a start file, let us know so we can guide you through it.)

- Connect to SQL as root,  
  ```
  ~/bin/root
  ```

- Create the database for Django to use:  
  ```
  create database project;
  ```

- Don’t forget to grant access to yourself, accomplished by:  
  ```
  grant all on project.* to 'username'@'silos.cs.indiana.edu';
  ```

- Exit SQL:  
  ```
  quit;
  ```

- Now edit settings.py to specify your newly created database:  
  ```
  pico ~/project/mysite/settings.py
  ```
Step VI: Editing settings.py

pico ~/project/mysite/settings.py

• Make your database section in settings.py look like this:

```python
DATABASE_NAME = 'project'
DATABASE_USER = 'username' # Your SQL username.
DATABASE_PASSWORD = 'password' # Your SQL password.
DATABASE_HOST = 'siro.cs.indiana.edu'
DATABASE_PORT = 'xxxx' # Extra port, on studentst and ports page.
```

• Now move to the bottom of settings.py file, and add on to the list of INSTALLED_APPS = (

```python
'...',
'django.contrib.admin',
'main',
'XXXX', (Note: XXXX is 'mysite.appname' Ex.
'mysite.polls'
)
```

Save and exit settings.py, then synchronize your SQL database with the necessary tables that your project requires:

```
python manage.py syncdb
```

Create a superuser, don’t forget your username/password for later admin.
Step VII: Models

- An example application, the polls app. First move in to our project/mysite folder:
  ```
  cd ~/project/mysite
  ```

- Use manage.py to create a python package within a directory structure for the polls app.
  ```
  python manage.py startapp polls
  ```

- Move into the polls directory, and edit models.py
  ```
  cd ~/project/mysite/polls/
pico models.py
  ```
Step VIII: Editing models.py

Your ~/project/mysite/polls/models.py should look like:

```python
from django.db import models
import datetime

class Poll(models.Model):
    question = models.CharField(max_length=200)
    pub_date = models.DateTimeField('date published')
    def __unicode__(self):
        return self.question
    def was_published_today(self):
        return self.pub_date.date() == datetime.date.today()

class Choice(models.Model):
    poll = models.ForeignKey(Poll)
    choice = models.CharField(max_length=200)
    votes = models.IntegerField()
    def __unicode__(self):
        return self.choice
```

Note: make sure your settings.py includes the mysite.polls in the list of INSTALLED_APPS.

• Once again, sync your database:
  python manage.py syncdb
• If you wish to see what is being sent to the SQL server:
  python manage.py sql polls
Step IX: Interactive Shell

- The Interactive API shell allows you to report, update, create, delete, and save entries for your application. To run the shell:

  ```python
  python manage.py shell
  ```

- Enter valid python code to get a feel for what the shell can do, first the yellow row and then the blue row:

```python
from mysite.polls.models import Poll, Choice
Poll.objects.all()  # Poll.objects.all()
import datetime
p = Poll(question='What’s up', pub_date=datetime.datetime.now())
```
```
p.save()  # Poll.objects.filter(question__startswith='What')
p.id
p.question
p.pub_date
p.pub_date = datetime.datetime(2007, 4, 1, 0, 0)
p.save()  # Poll.objects.filter(id=1)
Poll.objects.all()  # Poll.objects.get(id=1)
Poll.objects.filter(id=1)
```
Step X: The Admin Site

- **Edit urls.py** (pico ~/project/mysite/urls.py) and uncomment these three lines:

```python
from django.contrib import admin
admin.autodiscover()
(r'^admin/', include(admin.site.urls)),
```

- Again, start your development server:

  ```
  Python manage.py runserver silo.cs.indiana.edu:XXXX
  
  Where xxxx is your extra port, now browse to the address:
  
  http://silo.cs.indiana.edu:xxxx/admin/
  
  The page should look like:
  ```

  ![Django administration](image)

  - Use your admin username and password you created in Step 6, you should now see:
Step XI: Admin the Polls App

- Add our polls app to the admin site
  `pico ~/project/mysite/polls/admin.py`

- This file should look like this:
  ```python
  from mysite.polls.models import Poll
  from django.contrib import admin
  admin.site.register(Poll)
  ```

- Restart the development server in order to see the changes, and then browse again to: `http://silo.cs.indiana.edu:xxxx/admin/`

- From here you can edit your polls as necessary, but you may also desire to better organize your functions by editing admin.py further, by adding a class, such as:
  ```python
  class PollAdmin(admin.ModelAdmin):
      fields = ['pub_date', 'question']
  admin.site.register(Poll, PollAdmin)
  ```

- Refresh the Admin Site and you will see the changes you have made to the order of available polls.
The End

- For further apps, try:
  - http://djangoproject.com
- For sites that use Django:
  - http://www.tabblo.com/
  - http://www.lawrence.com/
  - http://projects.washingtonpost.com/congress/
  - http://www.ljworld.com/