importd Documentation

Release 0.4.0

Amit Upadhyay

April 14, 2016

Contents

1 hello world with importd 3

2 management commands 5

3 automatically configure django 7

4 manually configuring django 9

5 wsgi server 11

6 gunicorn server 13

7 auto-configution of templates 15

8 auto configuration of static folder 17

9 importd is relocatable 19

10 auto configuration of sqlite3 as database 21

11 @d decorator 23

12 auto imports 25

13 importd works well with smarturls 27

14 importd works well with fhurl 29

15 views can return non HttpResponse objects 31

16 importd with existing apps 33

17 importd and custom models 35

18 easy access to commonly used django methods and classes 37

19 url mount support 39

20 Blueprint 41

20.1 Usage 41

21 livereload support 43

22 a more detailed example 45

importd is the fastest way to django. No project creation, no apps, no settings.py, just an import.

Installation:

```
$ easy_install importd
```

Slides of a talk I gave about importd: http://amitu.com/importd/

importd is being developed on http://github.com/amitu/importd/.

See the Changelog: https://github.com/amitu/importd/blob/master/ChangeLog.rst.

CHAPTER 1

hello world with importd

With importd there is no need to create a django project or app. There is no settings.py or urls.py, nor is there a need of manage.py. A single file is sufficient, e.g. hello.py:

```
from importd import d

@d("/")
def index(request):
    return d.HttpResponse("hello world")

if __name__ == "__main__":
    d.main()
```

To run this hello.py:

```
$ python hello.py
Validating models...

0 errors found
Django version 1.4.1, using settings None
Development server is running at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

To see if it works:

```
$ curl "http://localhost:8000"
hello world
```

CHAPTER 2

management commands

d.main() acts as management command too:

```
$ python hello.py help shell
python hello.py help shell (02-18 21:14)
Usage: hello.py shell [options]

Runs a Python interactive interpreter. Tries to use IPython or bpython, if
one of them is available.

Options:
  -v VERBOSITY, --verbosity=VERBOSITY
                        Verbosity level; 0=minimal output, 1=normal output,
                        2=verbose output, 3=very verbose output
  --settings=SETTINGS   The Python path to a settings module, e.g.
                        "myproject.settings.main". If this isn't provided, the
                        DJANGO_SETTINGS_MODULE environment variable will be
                        used.
  --pythonpath=PYTHONPATH
                        A directory to add to the Python path, e.g.
                        "/home/djangoprojects/myproject".
  --traceback           Print traceback on exception
  --plain               Tells Django to use plain Python, not IPython or
                        bpython.
  -i INTERFACE, --interface=INTERFACE
                        Specify an interactive interpreter interface.
                        Available options: "ipython" and "bpython"
  --version             show program's version number and exit
  -h, --help            show this help message and exit
```

Further d.do() method can be used to call management command, eg d.do(“syncdb”) from python code.

CHAPTER 3

automatically configure django

importd sets DEBUG to true. This can be disabled by calling d(DEBUG=False) before any other importd functionality.

CHAPTER 4

manually configuring django

importd automatically configures django when needed. This can be disabled by calling d(dont_configure=True) before any other importd functionality.

CHAPTER 5

wsgi server

importd based hello.py is a **‘wsgi app‘_** without any more work.

... wsgi example http://www.tornadoweb.org/documentation/wsgi.html ...

gunicorn server

importd works with **gunicorn_** server, which is recommended for production setup instead of runserver command seen above, which is good only for debugging.

gunicorn is a dependency of importd, so if you have importd installed properly, gunicorn should be in your path.

Running hello.py with gunicorn:

```
$ gunicorn -w 2 hello:d
2013-02-18 21:20:06 [50844] [INFO] Starting gunicorn 0.17.2
2013-02-18 21:20:06 [50844] [INFO] Listening at: http://127.0.0.1:8000 (50844)
2013-02-18 21:20:06 [50844] [INFO] Using worker: sync
2013-02-18 21:20:06 [50847] [INFO] Booting worker with pid: 50847
2013-02-18 21:20:06 [50848] [INFO] Booting worker with pid: 50848
```

CHAPTER 7

auto-configution of templates

importd automatically includes templates folder in directory containing hello.py to TEMPLATE_DIRS settings.

CHAPTER 8

auto configuration of static folder

importd automatically maps /static/ path to folder named *static*, in the same directory as hello.py.

importd is relocatable

importd based script, like hello.py can be invoked from any folder, templates and static folders would be properly configured.

```
$ cd /any/folder
$ python /full/path/to/hello.py
Validating models...

0 errors found
February 18, 2013 - 21:23:11
Django version 1.5c1, using settings None
Development server is running at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

If in your program you need to refer to local path, call d.dotslash(path) method to translate relative paths to absolute paths properly, so your program continues to be relocatable.

CHAPTER 10

auto configuration of sqlite3 as database

For testing many a times sqlite is sufficient, and for those times importd automatically configures django with sqlite3 as database, with sqlite file stored in *db.sqlite* in the same folder as hello.py.

This can be disabled by passing actual database settings DATABASES to d().

CHAPTER 11

@d decorator

importd has a decorator that can be applied to any view to add it to URLS. By default the @d decorator takes the name of the view method, and constructs the url /method-name/.:

```
from importd import d

@d
def hello(request):
    return d.HttpResponse("hey there!")

if __name__ == "__main__":
    d.main()
```

In this case, importd will map hello() method to /hello/ url. This can be overridden by passing the URL where the view must be mapped to @d:

```
from importd import d

@d("/")
def hello(request):
    return d.HttpResponse("hey there!")

if __name__ == "__main__":
    d.main()
```

In this case hello method is mapped to /.

@d decorator also supports named urls via name keyword argument, eg:

```
from importd import d

@d("^home/$", name="home")  # named urls
def home(request):
    return "home.html"

if __name__ == "__main__":
    d.main()
```

CHAPTER 12

auto imports

Since most views.py methods will be defined in views.py of respective application, importd automatically imports views module of all apps configured to make sure all such decorators get called when django is configured.

For convenience importd also imports forms modules and signals modules of each app configured.

In some case this is not desirable, and can be disabled by passing *autoimport=False* as keyword arguments to *d()*.

CHAPTER 13

importd works well with smarturls

Since importd uses smarturls underneath this:

```
from importd import d

@d("^$")
def hello(request):
    return d.HttpResponse("hey there!")

if __name__ == "__main__":
    d.main()
```

is equivalent to:

```
from importd import d

@d("/")
def hello(request):
    return d.HttpResponse("hey there!")

if __name__ == "__main__":
    d.main()
```

Notice the simpler URL passed to @d("/") instead of d("^$"). Either form can be used.

Take a look at smarturls documentation to see how can simplify url construction for you.

CHAPTER 14

importd works well with fhurl

fhurl is a generic view for forms and ajax. importd integrates well with fhurl.:

```
from importd import d

@d("^fhurl/$")
class MyForm(d.RequestForm):
    x = d.forms.IntegerField(help_text="x in hrs")
    y = d.forms.IntegerField(help_text="y in dollars per hr")

    def save(self):
        return self.cleaned_data["x"] * self.cleaned_data["y"]

if __name__ == "__main__":
    d.main()
```

Running this:

```
$ python h2.py
Validating models...

0 errors found
February 20, 2013 - 09:40:56
Django version 1.5c1, using settings None
Development server is running at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Usage:

```
$ curl http://localhost:8000/fhurl/
{"errors": {"y": ["This field is required."], "x": ["This field is required."]}, "success": false}
$ curl "http://localhost:8000/fhurl/?x=10"
{"errors": {"y": ["This field is required."]}, "success": false}
$ curl "http://localhost:8000/fhurl/?x=asd"
{"errors": {"y": ["This field is required."], "x": ["Enter a whole number."]}, "success": false}
$ curl "http://localhost:8000/fhurl/?x=10&y=10"
{"response": 100, "success": true}
$ curl "http://localhost:8000/fhurl/?x=10&y=10&validate_only=true"
{"valid": true, "errors": {}}
$ curl "http://localhost:8000/fhurl/?x=10&y=asd&validate_only=true"
{"errors": {"y": ["Enter a whole number."]}, "valid": false}
$ curl "http://localhost:8000/fhurl/?json=true"
{"y": {"help_text": "y in dollars per hr", "required": true}, "x": {"help_text": "x in hrs", "require
```

fhurl can do a lot more, works with templates, renders the form and displays the form with errors, all with just one or two lines, check it out in fhurl docs.

fhurl with template:

```
from importd import d

@d("^fhurl/$", template="form.html")
class MyForm(d.RequestForm):
    x = d.forms.IntegerField(help_text="x in hrs")
    y = d.forms.IntegerField(help_text="y in dollars per hr")

    def get_json(self, _):
        # _ contains the data returned by .save() method
        return self.cleaned_data["x"] * self.cleaned_data["y"]

    def save(self):
        # .save() is always called, both in html mode or json mode
        # if json=true is passed, then get_json() is also called and
        # its result is returned.
        # else .save() is supposed to return a string that is redirected
        # fhurl assumes you always want to redirect to a new page after
        # saving a form, so that user does not accidentally resubmit the
        # form by hitting ctrl-R or on browser restart etc

        p = self.cleaned_data["x"] * self.cleaned_data["y"]
        return d.HttpResponseRedirect("/form-saved") # redirect to this url

if __name__ == "__main__":
    d.main()
```

form.html:

```
{% csrf_token %}
<form>{{ form }}</form>
```

Usage:

```
$ curl "http://localhost:8000/fhurl/"
<input type='hidden' name='csrfmiddlewaretoken' value='e1hIW2A0HWJMB27epijcc3XKD7JVB0nQ' />
<form><tr><th><label for="id_x">X:</label></th><td><input id="id_x" name="x" type="text" /><br /><spa
<tr><th><label for="id_y">Y:</label></th><td><input id="id_y" name="y" type="text" /><br /><span clas
$ curl -b "csrftoken=cnoaUDrr08haTTAMjpGWaPPBgt5rG1ZW" -d "csrfmiddlewaretoken=cnoaUDrr08haTTAMjpGWaP
<input type='hidden' name='csrfmiddlewaretoken' value='cnoaUDrr08haTTAMjpGWaPPBgt5rG1ZW' />
<form><tr><th><label for="id_x">X:</label></th><td><ul class="errorlist"><li>Enter a whole number.</l
<tr><th><label for="id_y">Y:</label></th><td><ul class="errorlist"><li>This field is required.</li><
$ curl -i -b "csrftoken=cnoaUDrr08haTTAMjpGWaPPBgt5rG1ZW" -d "csrfmiddlewaretoken=cnoaUDrr08haTTAMjp
HTTP/1.0 302 FOUND
Date: Wed, 20 Feb 2013 15:41:06 GMT
Server: WSGIServer/0.1 Python/2.7.1
Content-Type: text/html; charset=utf-8
Location: http://localhost:8000/asd

$ curl -b "csrftoken=cnoaUDrr08haTTAMjpGWaPPBgt5rG1ZW" -d "csrfmiddlewaretoken=cnoaUDrr08haTTAMjpGWaP
{"response": 10, "success": true}
```

views can return non HttpResponse objects

Django views are expected to only return HttpResponse based objects. importd allows you to do more than this.

A view can return a string, which is treated as name of template, which is rendered with RequestContext and returned. A view can also return a tuple of (str, dict), in this case the str is treated as name of template, and dict as the context:

```python
from importd import d
import time

@d # /index/, url derived from name of view
def index(request):
    return "index.html", {"msg": time.time()}

if __name__ == "__main__":
    d.main()
```

Further a view can also return arbitrary data structures not mentioned above, in such cases importd will convert that to JSON and return it to client:

```python
from importd import d

@d  # served at /json/, converts object to json string, with proper mimetype
def json(request):
    return {
        "sum": (
            int(request.GET.get("x", 0)) + int(request.GET.get("y", 0))
        )
    }

if __name__ == "__main__":
    d.main()
```

importd comes with convenience JSONResponse class to return arbitrary json object that may be a string, or a (string, dict) tuple.

importd with existing apps

Nothing special has to be done to work with existing apps, django specific INSTALLED_APPS must contain the name of apps as usual, and it can be passed to d() method:

```
from importd import d

d(INSTALLED_APPS=["django.contrib.auth", "django.contrib.contenttypes"])

from django.contrib.auth.models import User

@d("/<int:userid>/")
def hello(request, userid):
    user = User.objects.get(userid)
    return d.HttpResponse("hey there %" % user)

if __name__ == "__main__":
    d.main()
```

CHAPTER 17

importd and custom models

For custom models please create an app and add it to INSTALLED_APPS during configuration.

CHAPTER 18

easy access to commonly used django methods and classes

importd contains aliases for django methods and classes:

```
from importd import d

@d
def hello(request):
    return d.render_to_response("hello.html", d.RequestContext(request))

# d.render_to_response == django.shortcuts.render_to_response
# d.get_object_or_404 == django.shortcuts.get_object_or_404
# d.HttpResponse == django.http.HttpResponse
# d.patterns == django.conf.urls.defaults.patterns
# d.RequestContext == django.template.RequestContext
# d.forms == django.forms

if __name__ == "__main__":
    d.main()
```

CHAPTER 19

url mount support

Given a views.py in any app, we can use @d decorators to map it to urls. While this is convenient, it can make things difficult for apps to be redistributed where the final URL has to be controlled by end user.

importd has two mechanism for handling it. One option is to disable the @d view decorator across projects, or individual apps, or we can specify a "mount point" for each redistributable app.

eg:

An awesome app named "app" defines "/dashboard/" as a url by havint the following in app/views.py:

```
from importd import d

@d("/dashboard/")
def dashboard(request):
    return d.HttpResponse("app is awesome")
```

Lets say our main app is main.py, and it includes "app" in INSTALLED_APPS:

```
from importd import d

d(DEBUG=True, INSTALLED_APPS=["app"])

if __name__ == "__main__":
    d.main()
```

As it stands, our view defined in app.views is accessible by the URL /dashboard/. But lets say we want to configure things such that its available on /extra/dashboard/:

```
from importd import d

d(DEBUG=True, INSTALLED_APPS=["app"], mounts={"app": "/extra/"})

if __name__ == "__main__":
    d.main()
```

CHAPTER 20

Blueprint

Blueprint is way to group urls. For example, we make a RESTful-API. API has version. Each version of API has many urls. We embed version to url.:

```
@d("/v1/hello/", name="v1-hello")
def v1_hello(request): ....

@d("/v1/echo/", name="v1-echo")
def v1_echo(request): ....

@d("/v2/hello/", name="v2-hello")
def v2_hello(request): ....

@d("/v2/echo/", name="v2-echo")
def v2_echo(request): ....
```

If we can make view function group by API version, it is comfortable to manage API.:

```
v1 = Blueprint()

@v1("/hello/", name="hello")
def v1_hello(request): ....

@v1("/echo/", name="echo")
def v1_echo(request): ....

v2 = Blueprint()

@v2("/hello/", name="hello")
def v2_hello(request): ....

@v2("/echo/", name="echo")
def v2_echo(request): ....
```

This is based on django URL namespace. https://docs.djangoproject.com/en/dev/topics/http/urls/#url-namespaces

20.1 Usage

First, create blueprint and register view function to blueprint. The usage is simililary with @d decorator.:

```
from importd import d, Blueprint
bp = Blueprint()
```

```
@bp("/index")
def index3(request):
    return d.HttpResponse("app3/index")
```

Second, register blueprint to importd. It looks like mounts.:

```
d(
    DEBUG=True,
    INSTALLED_APPS=["app3"],
    blueprints={"app3": "app3.views.bp"}
)
```

Key of blueprints is namespace. Value of blueprints is blueprint module info. If blueprint module info is string, url prefix is "{namespace}/". For example, we register index3 view function as "/index" and namespace of blueprint is "app3". Generated URL is "/app3/index".

If you want to use more advanced features, you pass a dictionary.

```
blueprints={"app3-clone": {"blueprint": "app3.views.bp", "url_prefix": "app3-clone/"}}
```

livereload support

ImportD comes with livereload support.

To activate it you have to pass keyword argument “lr”, which is a dict.

eg.

```
d(
    DEBUG=True,
    lr={
        "static/scss,static/js": "make build"
    }
)
```

Subsequently, you can run python app.py livereload, which will listen on port 8000 and watch the folders “static/scss” and “static/js” and run “make build” if any file is modified in either folder.

You can pass any number of patterns as key value, eg if you want different cmds to run for css files and different for js files etc.

In order to use this feature, please install livereload package first.

This feature injects a auto reload js in each page, and runs a websocket in background, which signals the JS to reload the page everytime one of the watched file changes and command is finished.

CHAPTER 22

a more detailed example

This example features a few more use cases:

```
from importd import d

d(DEBUG=True, INSTALLED_APPS=["django.contrib.auth"]) # configure django

def real_index2(request):
    return d.HttpResponse("real_index2")

# setup other urlpatterns
d(d.patterns("",
    ("^$", real_index2),
))

@d # /index/, url derived from name of view
def index(request):
    import time
    return "index.html", {"msg": time.time()}

@d("^home/$", name="home")  # named urls
def real_index(request):
    return "home.html"

@d  # served at /json/, converts object to json string, with proper mimetype
def json(request):
    return {
        "sum": (
            int(request.GET.get("x", 0)) + int(request.GET.get("y", 0))
        )
    }

@d("/edit/<int:id>/", name="edit_page") # translats to ^edit/(?P<id>\d+)/$
def edit(request, id):
    return {"id": id}

@d("^fhurl/$")
class MyForm(d.RequestForm):
    x = d.forms.IntegerField()
    y = d.forms.IntegerField()

    def save(self):
        return self.cleaned_data["x"] + self.cleaned_data["y"]
```

```
if __name__ == "__main__":
    d.main()
```