Django FTP Deploy Documentation

Release 2.0

Lukasz Pakula

November 12, 2014

Contents

django-ftp-deploy allows you to automatically deploy GIT repositories to FTP servers. You don't need to install git on the server!

> **Warning:**
> Version **2.0** is not back compatibile due to new migration system implemented in Django 1.7.
>
> For older django versions (1.5 / 1.6) use version **1.x**

Support:

- Django 1.7
- Python 2.7 / 3.3 / 3.4

Features:

- Manage multiple services (a service is one repository-to-ftp configuration)
- Verification service configuration
- Repository hook management
- Restore failed deploys
- Email notifications
- Logs and Statistics

Supported GIT repositories:

- Bitbucket
- Github

Current tests coverage status:

76% coverage

CHAPTER 1

User Guide

1.1 Installation

Installation and requirements for django-ftp-deploy module

1.1.1 Requirements

1. celery
2. pycurl
3. certifi
4. django_braces
5. django_crispy_forms

Required third party libraries are **installed automatically** if you use pip to install django-ftp-deploy.

1.1.2 Installation

Note: FTP Deploy Server is optional and doesn't need to be installed for basic usage. It is however, highly recommended that you install FTP Deploy Server to gain full functionality.

1. The recommended way to install the Django FTP Deploy is via pip:

   ```
   pip install django-ftp-deploy
   ```

   If you aren't familiar with pip, download a copy of the `ftp_deploy` and add it to your Python path. You need to install all requirements manually as well.

2. Add `ftp_deploy` and `ftp_deploy.server` to your `INSTALLED_APPS` setting:

   ```
   #settings.py

   INSTALLED_APPS = (
     ...
     'ftp_deploy',
     'ftp_deploy.server',
     ...
   )
   ```

3. Make sure you have `django.core.context_processors.request` in your `TEMPLATE_CONTEXT_PROCESSORS` setting:

   ```
   #settings.py

   TEMPLATE_CONTEXT_PROCESSORS = (
     ...
     'django.core.context_processors.request',
     ...
   )
   ```

4. Add the `ftp_deploy` URLs to your project URLconf as follows:

   ```
   #projectname/urls.py

   urlpatterns = patterns('',
       ...
       url(r'^ftpdeploy/', include('ftp_deploy.urls')),
       url(r'^ftpdeploy/', include('ftp_deploy.server.urls')),
       ...
     )
   ```

5. Synchronize your database:

   ```
   python manage.py migrate ftp_deploy
   ```

6. Copy static files into your `STATIC_ROOT` folder

   ```
   python manage.py collectstatic
   ```

1.1.3 Configuration

- Add folder containing `settings.py` file to your Python path
- Add `DEPLOY_BITBUCKET_SETTINGS` and/or `DEPLOY_GITHUB_SETTINGS` configuration to your settings:

  ```
  #settings.py

  DEPLOY_BITBUCKET_SETTINGS = {
    'username'      : '',
    'password'      : '',
  }

  DEPLOY_GITHUB_SETTINGS = {
    'username'      : '',
    'password'      : '',
  }
  ```

- Set django_crispy_forms template pack to *bootstrap 3*

  ```
  #settings.py
  CRISPY_TEMPLATE_PACK = 'bootstrap3'
  ```

- Add celery configuration:

  ```
  #settings.py
  BROKER_URL = ''
  CELERY_RESULT_BACKEND=''
  #settings depends on message broker and result backend, see example below
  ```

- Go to your project root folder and run celery worker as follow:

  ```
  celery -A ftp_deploy worker --concurrency 1
  ```

  Note: Celery example above apply only for development enviroment. Celery worker in production should be run as a deamon. Read more in Celery documentation.

  Warning: Remember to include '*–concurrency 1*' option when running the worker. That avoid to perform more then one task at the same time.

Celery - RabbitMQ

If you are using Ubuntu or Debian install RabbitMQ by executing this command:

```
sudo apt-get install rabbitmq-server
```

- Update celery configuration as follows:

  ```
  #settings.py
  BROKER_URL = 'amqp://'
  CELERY_RESULT_BACKEND='amqp'
  ```

Celery - django

Note: Configuration presented below use django as a broker and result backend, however this is not recommended for production enviroment. Read more in Celery documentation.

In order to use django as broker and backend, project need to have django-celery project installed:

- Install django-celery using pip:

  ```
  pip install django-celery
  ```

- Add *djcelery* to your `INSTALLED_APPS` setting

  ```
  #settings.py

  INSTALLED_APPS = (
    ...
    'kombu.transport.django',
    'djcelery',
    ...
  )
  ```

- Update celery configuration as follows:

  ```
  #settings.py
  BROKER_URL = 'django://'
  CELERY_RESULT_BACKEND='djcelery.backends.database:DatabaseBackend'
  ```

- Synchronize your database:

```
python manage.py migrate djcelery
python manage.py migrate kombu.transport.django
```

1.2 Usage

Note: For full functionality you need to have *FTP Deploy Server* installed in your project, otherwise you can manage your deploys in admin page. For further informations visit *installation* section.

Important: It's important to always use **non fast forward** git merge to your deploy branch! Otherwise POST Hook has no information about commits included in merge.

```
git merge --no-ff
```

1.2.1 Login

Login screen for FTP Deploy Server application. Page is available at:

`/ftpdeploy/`

1.2.2 Dashboard

The Main **Dashboard** page is available at:

`/ftpdeploy/dashboard/`

The dashboard allows you to manage all of your services from one page. All failing services are placed on the top of the list, to make easy navigate to failed deploys.
A screenshot is available in the *Other* section.

If service is deploying, status icon is changed to animated cog.

1.2.3 Service

Service is one repository-to-ftp configuration.

Add Service

To add new service click `Add Service` button on the *Service* dashboard page or visit:

`/ftpdeploy/service/add`

The `Add Service` form includes sections with fields listed below. Please read the following instructions in order to fill out the form correctly:

`FTP Settings`

Host: FTP server host

Username: FTP server username

Password: FTP server password

Path: path to the root directory of your project

`Repository`

Source: Source of repository (Bitbucket or Github)

Respository Name: Name of the repository to deploy. After you choose source, the list is populated dynamically from your repository account

Respository Slug: Slug of your Repository Name. This field is populated *dynamically* by using 'slugify' on the repository name

Branch: Branch name for deploy service

`Notification`

Notification: Notification set using by service.

`Security`

Security Key: 30 character alpha-numeric, unique, random generated string. Security Key is used to create repository hook.

During saving process, all values are validated. The following checks are performed:

- FTP hostname
- FTP login credentials
- FTP path
- Repository login credentials
- Repository slug name (if exists on repository list)
- Repository branch
- Repository hook

All services will fail their first validation, as no POST hook will exist yet. You will be prompted to add the service hook by the error message. All issues are listed under status section in `Manage Service` page.

Note: Repository POST hook is **required**. It provides information about pushed git commits, along with branch name, which is used in the deploy process. You can manage this manually on the repository page if you need to. The validation process respects all changes you have made directly on the repository site as well.

Manage Service

To manage a service click the `Manage` button next to the service name or visit:

`/ftpdeploy/service/{service_id}/manage`

The manage page contains sections such as:

- Statistics of service

  - POST Hook status (if not set up, `Add Hook` link is provided)
  - number of success deploys
  - number of fail deploys if exists

- number of skiped deploys if exists
- last deploy user
- last deploy date

- Notification

  Represent current notification settings. You can change notification by clicking `Cog icon`

- Status

  Icon representing current status. If validation passes, it displays date of the latest status check, otherwise list of issues. In order to refresh the status you need to click *status icon* (the same applies for services list on dashboard page) or edit and save service.

  Note: Status is not refreshed automatically because of expense of validation process. Usually takes up to 15 seconds to go through all validation points.

- Restore Deploys (if any of deploys has failed)

  List of failing deploys for service in chronological order. The list provides the following details:

  - Deploy date
  - Deploy user
  - Deploy commits (commit message, commit user, commit raw node)
  - Restorable flag
  - Status

  If the list contain deploys that you don't want to restore you can skip them by clicking the `Skip` button.

  Warning: Skipping deploys may cause inconsistent data between your repository and FTP files or may fail to restore deploys.
  Example: if you skip a deploy with commit that creates a new file, and next deployment include commit that attempts to remove this file, the entire restore process would fail because of trying remove a file that actually doesn't exist.

  Restoring deploys is described in the Restore Failed Deploys section.

- Recent Deploys

  List of recent deploys. List mirror Logs filtered by current service.

- If service is deploying or has deploy in the queue, progress bar is presented to display current status. Restore deploy is locket at this time as well.

Edit Service

To edit service click `Edit` button next to the service name or visit:

```
/ftpdeploy/service/{service_id}/edit
```

Edit page provides the same functionality as the Add Service page. If you need to load list of your repositories again, you need to reset *Source* drop down list, and choose option again.

After you press submit, service data goes through the validation process again, and redirect you to the Manage Service page

Restore Failed Deploys

If some deploy has failed, the service has an opportunity to restore it. It's possible by capturing payload data from POST Hook and storing the data before a deployment is performed. The restoring process works as follows:

`Restoring process`

- Find first failed and not skipped deploy
- Built the restore tree, since first fail deploy up to the most recent deploy (omit skip deploys)
- Build new payload data based on restore tree
- Build commits information and files diff from new payload
- After click restore send new payload to deploy (as it would be a normal POST Hook), remove old deploys included in restore, and store new payload.

In order to restore deploys you need to click `Restore Deploys` on Manage Service page. That will bring the popup window with information about the restore such as :

- Number of commits included in restore along with details (commit message, commit user, raw node)
- File diff (New Files, Modified Files, Removed Files)

To run restore process you need to press `Restore` button.

Note: If your restore keep failing you can manage this manually. As you never lose deploys and commits information you can rely on *File diff* even after failed restore. You can just transfer and remove all relevant files included in *File diff* and skip all failed deploys. That help you keep your data consistent.

1.2.4 Notifications

Configurable sets of notifications.

Add/Edit Notification

To add new notification click `Add Notification` button on the *Notification* dashboard page or visit:

```
/ftpdeploy/notification/add
```

You can add emails as many as you like and choose what kind of notification they going to receive. In addition there are two extra options as follow:

deploy_user: user who make an deploy

commit_user: email list of users who made a commit(s) included in deploy

In order to edit notification you can click `Edit` button next to notification name or visit:

```
/ftpdeploy/{notification_id}/edit
```

Edit Notification screen provide same functionality as *Add Notification* page.

Email Templates

The email notification system provides html and text templates that can be overriden if you wish. In order to do that you need to create your own templates for success and fail notification separately:

```
/ftp_deploy/email/email_success.html
/ftp_deploy/email/email_success.txt

/ftp_deploy/email/email_fail.html
/ftp_deploy/email/email_fail.txt
```

All templates are rendered with the following context information:

`Success Template`

- *{{service}}* object
- *{{host}}* of the current website (where the email came from)
- *{{commits_info}}* in format [['commit message','commit user','raw node'],[...]]
- *{{files_added}}* , *{{files_modified}}*, *{{files_removed}}* in format ['file_name_1', 'file_name_2', ...]

`Fail Template`

- *{{service}}* object
- *{{host}}* of the current website (where the email came from)
- *{{error}}* message of the exception

1.2.5 Logs

In order to see logs page you need to click `Log` button on the top of the page or visit:

```
/ftpdeploy/log
```

Logs provide information about all activity in the FTP Deploy application.

In addition log list contain information about number of commits included in deploy. If you need to see more details about included commits, you can click the `commits number` (commit message, commit user, raw node).

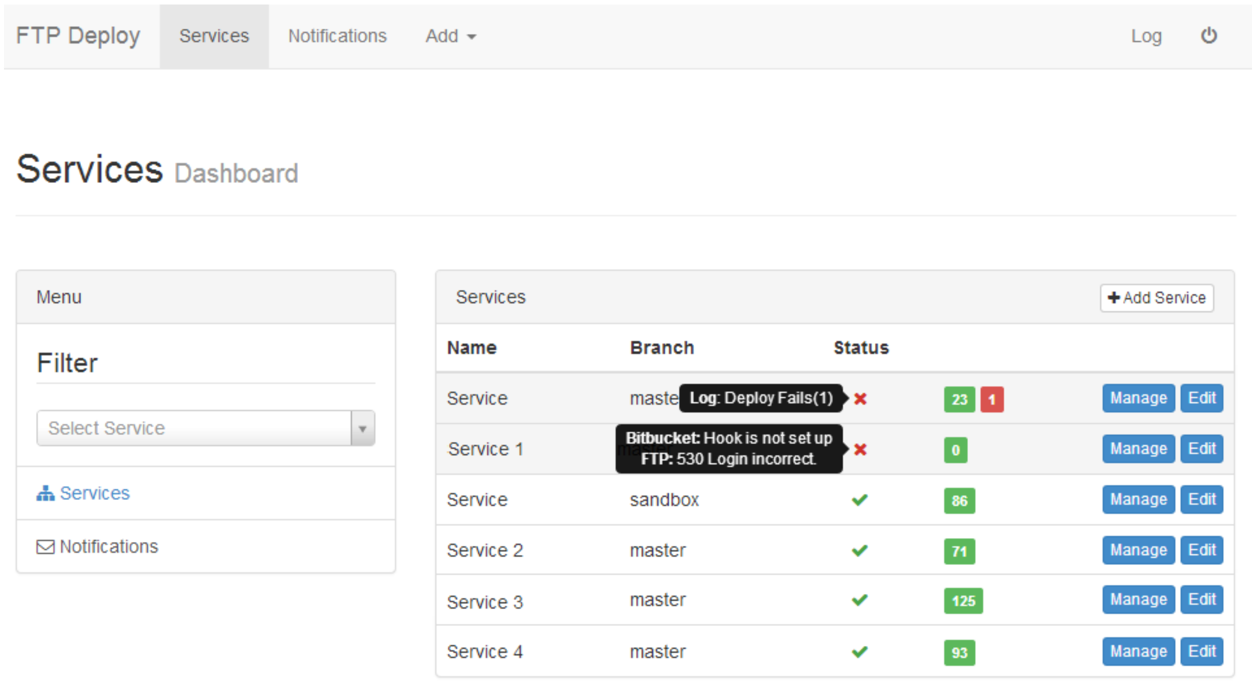
1.3 Other

Other informations about django-ftp-deploy module

1.3.1 Screens

All screens are only for preview purposes, and may be different then original application.

1. Dashboard



Services Dashboard

Menu

Filter

Select Service

Services

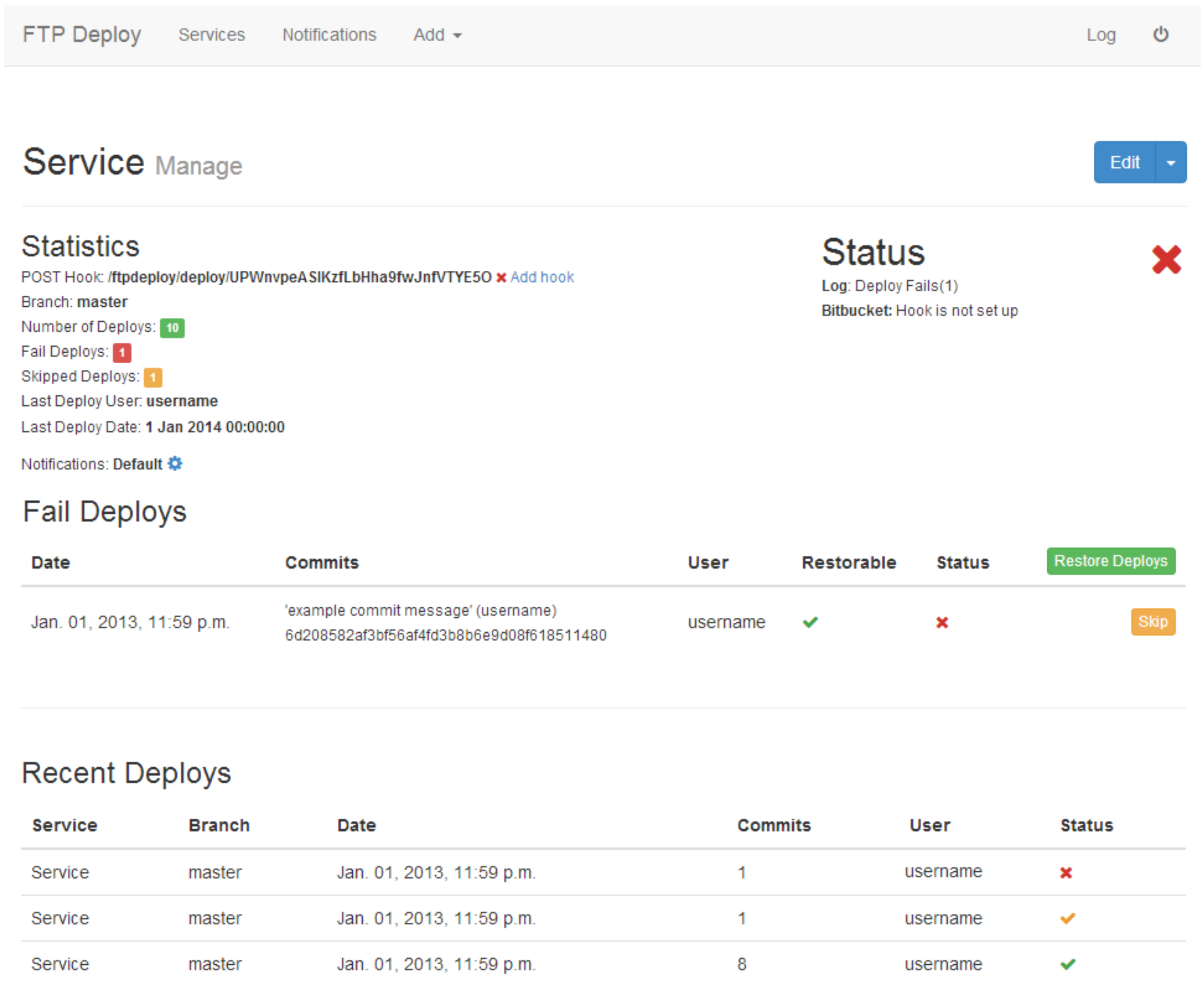
Notifications

Services

+ Add Service

Name	Branch	Status		
Service	maste Log: Deploy Fails(1)	✗	23 1	Manage Edit
Service 1	Bitbucket: Hook is not set up FTP: 530 Login incorrect.	✗	0	Manage Edit
Service	sandbox	✓	86	Manage Edit
Service 2	master	✓	71	Manage Edit
Service 3	master	✓	125	Manage Edit
Service 4	master	✓	93	Manage Edit

2. Manage



Service Manage

Statistics

POST Hook: /ftpdeploy/deploy/UPWnvpeASlKzfLbHha9fwJnfVTYE5O ✖ Add hook
Branch: master
Number of Deploys: 10
Fail Deploys: 1
Skipped Deploys: 1
Last Deploy User: username
Last Deploy Date: 1 Jan 2014 00:00:00

Notifications: Default

Status ✖

Log: Deploy Fails(1)
Bitbucket: Hook is not set up

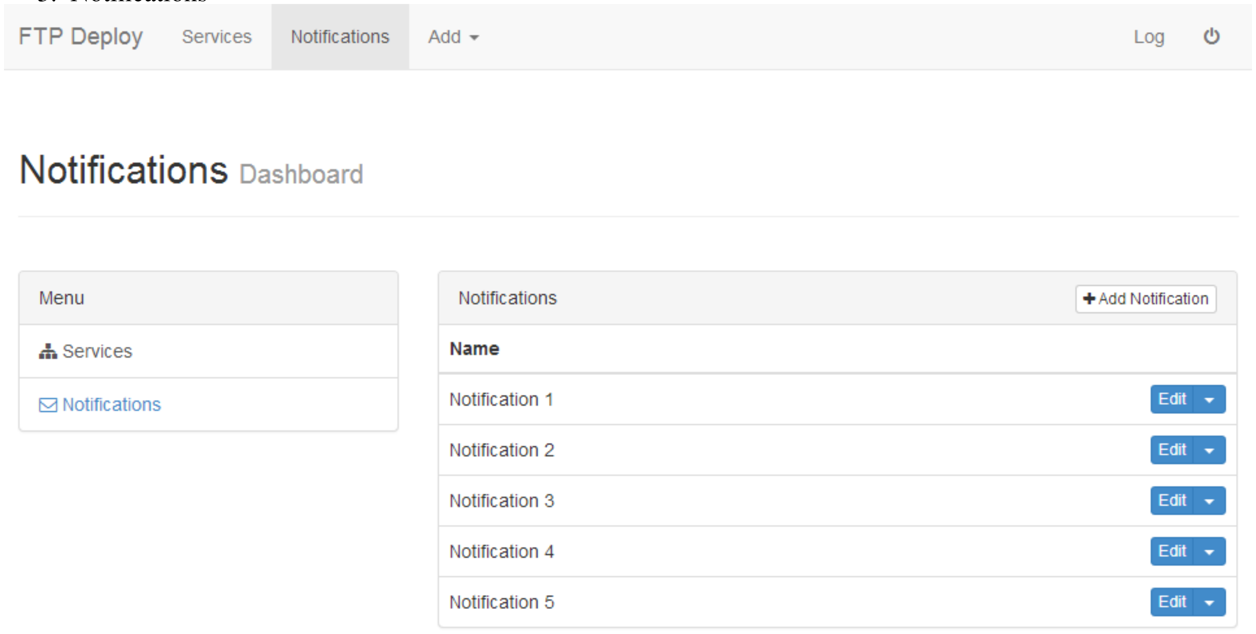
Fail Deploys

Date	Commits	User	Restorable	Status	Restore Deploys
Jan. 01, 2013, 11:59 p.m.	'example commit message' (username) 6d208582af3bf56af4fd3b8b6e9d08f618511480	username	✓	✖	Skip

Recent Deploys

Service	Branch	Date	Commits	User	Status
Service	master	Jan. 01, 2013, 11:59 p.m.	1	username	✖
Service	master	Jan. 01, 2013, 11:59 p.m.	1	username	✓
Service	master	Jan. 01, 2013, 11:59 p.m.	8	username	✓

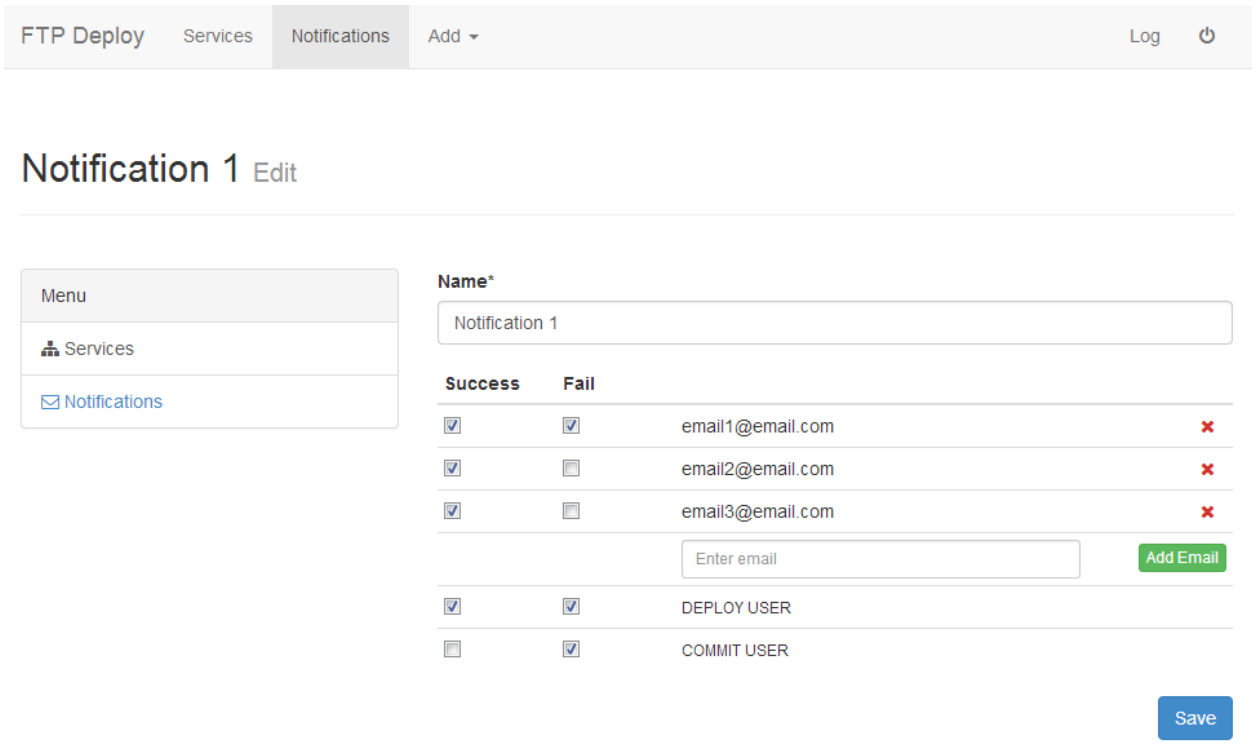
3. Notifications



Notifications Dashboard

Notifications

Name	
Notification 1	Edit
Notification 2	Edit
Notification 3	Edit
Notification 4	Edit
Notification 5	Edit

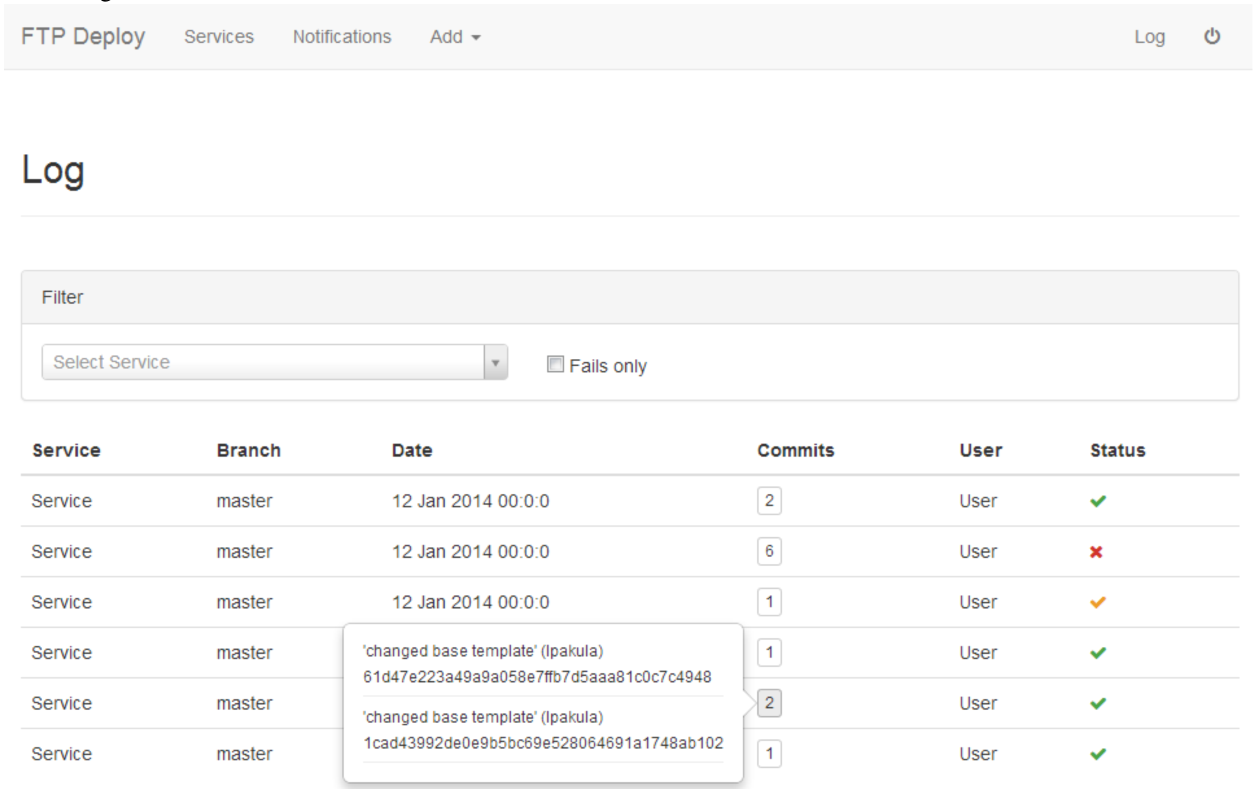


Name*

Notification 1

Success	Fail		
✓	✓	email1@email.com	✖
✓		email2@email.com	✖
✓		email3@email.com	✖
		Enter email	Add Email
✓	✓	DEPLOY USER	
	✓	COMMIT USER	

4. Log



Service	Branch	Date	Commits	User	Status
Service	master	12 Jan 2014 00:0:0	2	User	✓
Service	master	12 Jan 2014 00:0:0	6	User	✖
Service	master	12 Jan 2014 00:0:0	1	User	✓
Service	master		1	User	✓
Service	master		2	User	✓
Service	master		1	User	✓

'changed base template' (lpakula)
61d47e223a49a9a058e7ffb7d5aaa81c0c7c4948

'changed base template' (lpakula)
1cad43992de0e9b5bc69e528064691a1748ab102

1.4 Changelog

Changelog for django-ftp-deploy module

1.4.1 v2.0

- support for Django 1.7
- support for Python 3

1.4.2 v1.4.1

- travis integration
- documentation updates
- PEP8 updates

1.4.3 v1.4

- github support
- code updates
- documentation updates

1.4.4 v1.3

- queueing deploys

1.4.5 v1.2.1

- unit and integration tests development
- templates and style updates
- 'skip deploy' bug fix
- notification in service admin(django) page
- code updates
- PEP8 updates
- documentation updates

1.4.6 v1.2

- deploying process indicator for services
- lock restoring option whilst deploy is running
- catch deploys sended during process another deploy, to be able to restore them (need queuing in addition)
- login screen for server application

- add notification per service configuration
- develop Unit Tests
- new design

1.4.7 v1.1

- New rewrite version (doesn't compatibile with version 1.0)

CHAPTER 2

Get Involved!

Get involved and help make this app better!

2.1 Introduction

1. Clone `django-ftp-deploy` app:

   ```
   git clone https://bitbucket.org/lpakula/django-ftp-deploy.git
   ```

2. Add `django-ftp-deploy` folder to your Python path
3. Install application as described in *installation* section.
4. Install all requirements for dev environment. Go to `django-ftp-deploy` directory and use pip:

   ```
   pip install -r requirements/dev.txt
   ```

5. Add `FTP_TEST_SETTINGS` configuration to your settings:

   ```
   #settings.py

   FTP_TEST_SETTINGS = {
    'host'      : '',
    'username'  : '',
    'password'  : '',
    'path'      : '',
   }
   ```

6. Make sure `DEPLOY_BITBUCKET_SETTINGS` and `DEPLOY_GITHUB_SETTINGS` both have been added to settings file.
7. Install PhantomJS for intergration tests.
8. **Start Developing!**

2.2 Testing

Application use Nose as test runner and Fabric library to automate testing process.

In order to run tests go into *tests* directory and:

- *all* tests:

```
fab test
```

- *all* tests with *coverage*:

  ```
  fab testc
  ```

- *Unit Tests* only:

  ```
  fab testu
  ```

- *Integration Tests* only:

  ```
  fab testi
  ```

Unit Tests and *Integration Tests* accepts **module** attibute to specify module to test:

```
fab testu:module_name
```

CHAPTER 3

Roadmap

- Cron validation
- FTP password encryption
- Advanced statistics
- Support multi queues

CHAPTER 4

Indices and tables

- *genindex*
- *modindex*
- *search*