Introducción

Fabric es una librería para Python y una herramienta basada en linea de comandos para realizar despliegues o tareas para la administración de sistemas a través de SSH.

Fabric provee de una suite básica para ejecutar operaciones en local o en consolas remotas, subir o descargar archivos, así como funcionalidades auxiliares como mostrar la salida de las operaciones ejecutadas en remoto.

El uso típico de Fabric involucra el desarrollo de un módulo Python (fabfile) conteniendo una o más funcionalidades. Estas funcionalidades se podrán ejecutar a través de la linea de comandos fab.

Hello, fab

Un ejemplo muy sencillo y que se ejecuta en local es:

```
def hello():
    print('Hello fab!')
```

Si guardamos este pequeño trozo de código en archivo llamado `fabfile.py` en la raiz de nuestro proyecto podremos realizar lo siguiente:

```
$ fab hello
Hello fab!

Done.
```

Podemos complicarlo un poco más y pasar argumentos a nuestras tareas.

```
def hello(name):
    print('Hello %s!' % name)
```

Para poder pasarle los argumentos a nuestro `fabfile`, escribiremos lo siguiente en cualquier terminal:

```
$ fab hello:Quique
Hello Quique!

Done.
```

Como podemos ver, no es nada complicado emplear argumentos a nuestras tareas. No hay límites a la hora de pasar argumentos y, como en python, también podemos emplear kwargs en Fabric.

```
def hello(name, surname, treat='Sr'):
    print('Hello %s. %s %s!' % (treat, name, surname))
```

Y podemos ejecutarlo como:

```
$ fab hello:Quique,Piña,treat=Lord
Hello Lord Quique Piña!


Done.
```

Comandos locales

Imaginémonos que queremos automatizar el proceso de testear nuestra aplicación y realizar un commit en git solo cuando el test haya pasado correctamente.

```
from fabric.api import local

def prepare_deploy():
    local('python ./main_test.py')
    local('git add -p && git commit')
    local('git push')
```

Si ejecutamos este código, el resultado por pantalla será el siguiente:

```
$ fab prepare_deploy
[localhost] run: python ./main_test.py
...
----------------------------------------
Ran 3 test in 0.000s

Ok

[localhost] run: git add -p && git commit
<interactive Git add / git commit edit message session>
[localhost] run: git push
<git push session, possibly merging conflicts interactively>

Done.
```

Podemos observar que como el test a pasado automaticamente, Fabric a ejecutado las siguientes lineas y se ha realizado un add/commit y push de manera interactiva por el usuario.

En el supuesto caso que nuestro test no hubiera pasado, el resultado sería el siguiente:

```
$ fab prepare_deploy
[localhost] run: python ./main_test.py
...
======================================================
ERROR: test_stupidity (__main__.Test_People_Stupidity)
------------------------------------------------------
Traceback (most recent call last):
    File "main_test.py", line 25, in test_stupidity
        raise Exception("Much people. Wow stupidity")
    Exception: Much people. Wow stupidity
------------------------------------------------------
Ran 3 test in 0.000s

FAILED (errors=1)

Fatal error: local() encountered an error while executing 'python ./main_test.py'

Aborting.
```

En este caso, Fabric ha sabido detectar el error y no ha continuado con los siguientes comandos que tenía pendiente a ejecutar.

¡Bien! Acabamos de automatizar que para realizar un push en git se deban pasar los test correspondientes.

Comandos remotos

Ahora vamos a proceder a realizar cosas un poco más complicadas y que tienen mucho más que ver con realizar despligues.

Para realizar despligues másivos en diferentes máquinas, debemos especificar a que máquinas se van realizar dichos despligues. Para ello, debemos instanciar una lista con los nombres o IP de las diferentes máquinas en `env.hosts`.

Un ejemplo muy básico sería:

```
from fabric.api import env, run
```

```
env.hosts = ["airkiller.cloud.lan"]
env.user  = "mc_ripper"

def do_some_cool_stuff():
    run('sh /home/mc_ripper/start_show_scrapper.sh')
```

```
$ fab do_some_cool_stuff
[airkiller.cloud.lan] run: sh /home/mc_ripper/start_show_scrapper.sh
Starting show scrapper
No tv shows on config file

Done.
```

En el ejemplo anterior Fabric se conectará a la máquina **airkiller.cloud.lan** con el usuario **mc__ripper** y ejecutará el comando `sh /home/mc_ripper/start_show_scrapper.sh`.

Subida de archivos

Fabric tiene una herramienta muy poderosa para poder realizar subidas a las máquinas. Con el comando **put** podemos decirle a Fabric que archivo de la máquina local queremos subir y dónde.

```
from fabric.api import env, run
from fabric.decorators import with_settings
from fabric.operations import put


env.hosts = ['airkiller.cloud.lan']
env.user  = 'mc_ripper'

@with_settings(warn_only=True)
def rm_project_folder():
    project_folder = '/home/%s/cool_project' % env.user
    run('rm -rf %s/*' % project_folder)

def cp_project_folders(folders):
    project_folder = '/home/%s/cool_project' % env.user
    for fld in folders:
        put(fld, project_folder)

def cp_project_files(files):
    project_folder = '/home/%s/cool_project' % env.user
    for f in files:
        put(f, project_folder)
```

```
def virtualenv(ve):
    project_folder = '/home/%s/cool_project' % env.user
    run('virtualenv %s/%s' % (project_folder, ve))

def install_requirements(ve):
    project_folder    = '/home/%s/cool_project' % env.user
    env_folder        = '%s/%s' % (project_folder, ve)
    requirements_file = '%s/requirements.txt' % project_folder
    run('source %s && pip install -r %s' % (env_folder, requirements_file))

def deploy_project():
    folders_to_deploy = ['doc', 'src', 'lib']
    files_to_deploy   = ['requirements.txt', 'main.py']

    rm_project_folder()
    cp_project_folders(folders_to_deploy)
    cp_project_files(files_to_deploy)
    virtualenv('cool_project_ve')
    install_requirements('cool_project_ve')

def run_project():
    project_folder = '/home/%s/cool_project' % env.user
    env_folder     = '%s/cool_project_ve' % project_folder
    main_file      = '%s/main.py'
    run('source %s && python %s start' % (env_folder, main_file))
```