HINTS

MIGRACIONES

Django usa migraciones para propagar los cambios que realiza a sus modelos en su base de datos. La mayoría de las veces, éste puede generarlos por ti.

Para crear una migración, ejecute:

```
1 django-admin makemigrations <app_name>
```

Esto creará un archivo de migración en el submódulo de migration de app_name. La primera migración se llamará `0001_initial.py`, la otra comenzará con `0002_`, luego `0003`, y así sucesivamente...

Si omite `<app_name>`, esto creará migraciones para todas sus `INSTALLED_APPS`.

Para propagar migraciones a su base de datos, ejecute:

```
1 django-admin migrate <app_name>
```

Para mostrar todas sus migraciones, ejecute:

```
1 django-admin showmigrations app_name
2
3 app_name
4   [X] 0001_initial
5   [X] 0002_auto_20160115_1027
6   [X] 0003_somemodel
7   [ ] 0004_auto_20160323_1826
```

`[X]` significa que la migración se propagó a su base de datos. Y `[ ]` significa que la migración no se propagó a su base de datos. Usa django-admin migrate para propagarlo.

También puede llamar a revertir migraciones, pasando el nombre de la `migrate` command. Dada la lista anterior de migraciones (mostrada por `django-admin showmigrations`):

```
1 django-admin migrate app_name 0002  # Roll back to migration 0002
2 django-admin showmigrations app_name
3
4 app_name
5   [X] 0001_initial
6   [X] 0002_auto_20160115_1027
7   [ ] 0003_somemodel
8   [ ] 0004_auto_20160323_1826
```

MIGRACIONES MANUALES

Algunas veces, las migraciones generadas por Django no son suficientes. Esto es especialmente cierto cuando desea realizar migraciones de datos.

Por ejemplo, vamos a tener ese modelo:

```
class Article(models.Model):
    title = models.CharField(max_length=70)
```

Este modelo ya tiene datos existentes y ahora desea agregar un `SlugField`:

```
class Article(models.Model):
    title = models.CharField(max_length=70)
    slug = models.SlugField(max_length=70)
```

Se crean las migraciones para agregar el campo, pero ahora hay que asegurar que se migre con lo ya existente y no haya ninguna pérdida.

Se puede hacer algo como esto en la terminal:

```
$ django-admin shell
>>> from my_app.models import Article
>>> from django.utils.text import slugify
>>> for article in Article.objects.all():
...     article.slug = slugify(article.title)
...     article.save()
...
>>>
```

Pero se tendrá que hacer en todos sus entornos (es decir, en el escritorio de su oficina, en su computadora portátil, entre otros) para que sea definitivo, por lo que lo haremos en una migración. Lo primero será crear una migración vacía:

```
$ django-admin makemigrations --empty app_name
```

Esto creará un archivo de migración vacío. Ábrelo, contiene un esqueleto base. Digamos que su migración anterior se llamó `0023_article_slug`, y esta se llamará `0024_auto_20160719_1734`. Esto es lo que escribiremos en nuestro archivo de migración:

```python
from __future__ import unicode_literals
from django.db import migrations
from django.utils.text import slugify

def gen_slug(apps, schema_editor):
    Article = apps.get_model('app_name', 'Article')
    for row in Article.objects.all():
        row.slug = slugify(row.name)
        row.save()

class Migration(migrations.Migration):
    dependencies = [
        ('hosting', '0023_article_slug'),
    ]

    operations = [migrations.RunPython(gen_slug,
    reverse_code=migrations.RunPython.noop)]
```

CREANDO MIGRACIONES

1. LIMPIAR MIGRACIONES

Se debe realizar primero un backup de la base de datos, y un `commit`.

Si se han hecho muchos cambios en el modelo, sobre todo en las primeras etapas del proyecto, quizá sea bueno limpiar tanto las tablas `django_migrations`, como el directorio `app/migrations/`.

Eliminar la tabla `django_migrations`.

```
psql -U nombre_usuario nombre_db
DROP TABLE IF EXISTS django_migrations;
\q
```

Eliminar los directorios `app/migrations/`, ir al directorio raíz del proyecto, y ejecutar.

```
find ./ -type d -name "migrations" -exec rm -rf {} \;
```

Crear migraciones con makemigrations de las apps en el proyecto. Esta parte se debe hacer por cada app.

```
./manage.py makemigrations app1
./manage.py makemigrations app2
./manage.py makemigrations appX
```

Restablecer.

```
./manage.py migrate --fake-initial
```

2. MIGRACIONES FALSAS

Cuando se ejecuta una migración, Django almacena su nombre en una tabla `django_migrations`.

Crear y falsificar las migraciones iniciales para el esquema existente si su aplicación ya tiene modelos y tablas de base de datos, y no tiene migraciones.

Primero crea migraciones iniciales para tu aplicación.

```
./manage.py makemigrations your_app_label
```

Ahora, falsifica migraciones iniciales según lo aplicado.

```
./manage.py migrate --fake-initial
```

Falsifica todas las migraciones en todas las aplicaciones.

```
./manage.py migrate --fake
```

Falsifica migraciones de una sola aplicación.

```
./manage.py migrate --fake core
```

3. NOMBRES PERSONALIZADOS PARA ARCHIVOS DE MIGRACIÓN

Use `makemigrations --name <your_migration_name>` para permitir nombrar las migraciones, en lugar de usar un nombre generado.

```
./manage.py makemigrations --name <your_migration_name> <app_name>
```

4. CONFLICTOS MIGRATORIOS

Algunas veces, las migraciones entran en conflicto, lo que hace que ésta no tenga éxito. Puede suceder en muchos escenarios, sin embargo, ocurre de manera regular al desarrollar una aplicación con un equipo de desarrolladores.

Los conflictos de migración comunes ocurren mientras se usa el control de origen, especialmente cuando se usa el método de característica por rama. Para este escenario, usaremos un modelo llamado `Reporter`, con el `name` y la `address` como atributos.

En este punto, dos desarrolladores se encargarán de una característica, por lo que ambos obtienen esta copia inicial del modelo `Reporter`. El desarrollador A agrega una `age` que da como resultado el archivo `0002_reporter_age.py`. El desarrollador B agrega un campo `bank_account` que da como resultado `0002_reporter_bank_account`. Una vez que fusionan su código e intentan realizar las

migraciones, se produce un conflicto de migración; esto ocurre porque dichas migraciones alteran el mismo modelo `Reporter`. Además de eso, los nuevos archivos comienzan con `0002`.

5. FUSIONANDO MIGRACIONES

La solución más simple para esto es ejecutar el comando `makemigrations`, con una marca `--merge`.

```
./manage.py makemigrations --merge <my_app>
```

Esto creará una nueva migración para resolver el conflicto anterior.

Cuando este archivo adicional no es bienvenido en el entorno de desarrollo, otra opción es eliminar las migraciones en conflicto. Luego, se puede hacer una nueva migración usando el comando regular `makemigrations`.