PROYECTO DE SISTEMAS INFORMÁTICOS

José A. Macías B-309
j.macias@uam.es

roberto.marabini@uam.es coordinador

Evaluación Continua: 85% horas de asistencia
45 h no presenciales
30 h presenciales

Temario

P1 (APTO/NO APTO) Lenguaje y herramientas: Python3, Git, Github, PyCharm, Flake8 (PEP8)

P2 (APTO/NO APTO) Entorno de desarrollo web: Django, TDD (test-driven development), Coverage, PostgreSQL, Heroku

P3 (0-10) Aplicación web a desarrollar (parte 1) 45%

P4 (0-10) Aplicación web a desarrollar (parte 2) 55%

Control 1 → Conocimientos python { 10 ~~preg~~ minutos, 15 preg. test, Para aprobar → 14 bien
(APTO/NO APTO)

Control 2 → Control final

NOTA FINAL: 0'8 * Proyecto + 0'2 * Control2

1. Normas

- El examen es individual.
- Se podrá hacer uso del código de las prácticas realizadas durante el curso.
- Se puntuará la corrección de los resultados, pero también la "calidad" del código (legibilidad, comentarios, seguimiento del patrón de diseño proporcionado por Django, etc.).
- Se entregará en Moodle como resultado del examen el fichero producido por el comando `git archive --format zip --output ../examenX.zip master` (donde X será la letra que identificará el examen (mirar la parte superior de la primera página del enunciado para identificarla))
- Se incluirá un fichero de texto (con el nombre memoria.txt) describiendo como se ha realizado cada uno de los apartados descritos a continuación.
- La hora límite para entrega de los ficheros será fijada por el profesor. A partir de dicha hora se penalizará con un punto cada minuto de retraso.

Enunciado

Para la implementación de una pagina web para cinéfilos se construye la siguiente base de datos que relaciona películas con actores:

pelicula(id, nombreP)
productora(id, nombreP)
produce(pid, pelicula(id)⇑, productora(id)⇑, coste)
donde ⇑ indica clave extranjera, y coste contiene la cantidad invertida en la película.

Recordad que Django añade automáticamente el atributo *id* que funciona como clave primaria.

Actividades a realizar

1. Cread un proyecto llamado `proyecto` en Django. Dentro del proyecto crear una aplicación llamada `aplicacion`. Los datos se persistirán en una base de datos llamada `examen` creada en postgres.

2. Cread el modelo de datos (fichero `models.py`) y configurad el proyecto para que los modelos puedan ser accedidos usando el sistema de administración de Django (`http://localhost:8000/admin/`).

3. Escribid un script (llamado `poblar.py`) que inserte los datos siguientes en la base de datos usando la API ofrecida por Django.

   pelicula(1001,'pelicula_a')
   pelicula(1002,'pelicula_b')
   pelicula(1003,'pelicula_c')

   productora(1001,'productora_a')
   productora(1002,'productora_b')
   productora(1003,'productora_c')
   productora(1004,'productora_d')

   peli productora coste

   produce(1001,1001,1001, 100)
   produce(1002,1002,1002, 200)
   produce(1003,1002,1003, 100)
   produce(1004,1001,1003, 150)

4. Despliega el proyecto en Heroku y puebla la base de datos usando el fichero `poblar.py`, así mismo activa el interfaz de administración usando como nombre de usuario y password `alumnodb`. Nota: incluir en el fichero `memoria.txt` el URL en el cual la aplicación está accesible en Heroku. IMPORTANTE: Cuando acabéis de implementar este punto, y siempre antes de la hora de finalización del examen, mostrárselo a vuestro profesor.

5. Usando el template `productora.html`, crea una página web accesible en la dirección $PROJECT_URL/aplicacion/productora/ que para la productora con id = 1001 devuelva un listado con las peliculas en las que dicha productora ha participado. Nota: productora.html no debe modificarse. Si la base de datos no contiene ninguna película en la que la susodicha productora haya participado se debe informar al usuario a través de la variable `error`.

6. Crea un test (a ejecutarse con la orden `python manage.py test aplicacion.tests`) que:

   - borre todas las películas, actores y participaciones

error
productora
peliculas

- cree la película (1001, 'pelicula1')
- cree la productora (1001, 'productora1')
- cree la productora (1002, 'productora2')
- cree el produce (1001,1001,1002,100)
- acceda a la vista relacionada con el URL $PROJECT_URL/aplicacion/productora/
- compruebe que las películas devueltas son las correctas.

2. Normas de calificación

Para obtener 5 puntos (aprobar) es necesario que tras ejecutar los comandos:

```
  dropdb -U alumnodb -h localhost examen
createdb -U alumnodb -h localhost examen
python manage.py makemigrations aplicacion
python manage.py migrate
python manage.py createsuperuser
python ./poblar.py
```

Se pueda acceder a la interfaz de administración `http://localhost:8000/admin/` y obtener un listado con los datos enumerados en el punto (3) del apartado anterior. Igualmente es necesario que los datos persistan en una base creada en postgres.

Para obtener 7 puntos se deben satisfacer todos los puntos del apartado anterior, y además debe ser posible acceder a la interfaz de administración de la aplicación desplegada en heroku usando el username/password `alumnodb`. Nota: el código desplegado en Heroku debe ser IDÉNTICO al subido a Moodle, en caso contrario la puntuación de este apartado será nula.

Para obtener 8 puntos se deben satisfacer todos los puntos del apartado anterior, y además la página $PROJECT_URL/aplicacion/productora/ debe funcionar correctamente. Se deben poder suministrar todas las variables usadas en el template productora.html.

Para obtener 10 puntos se deben satisfacer todos los puntos del apartado anterior, y además el test debe satisfacer todos los requerimientos solicitados.

A. Templates

```
<html>
 <head>
 </head>
<body>
```

```
{% if error %}
   {{ error }}
{% endif %}
Peliculas en las que la productora {{ productora.nombreP }} con
identificador {{ productora.id}} ha participado
<table>
{% for pelicula in peliculas %}
        <tr>
            <th>{{ pelicula.id }}</th>
            <td>{{ pelicula.nombreP }}</td>
        </tr>
{% endfor %}
</table>
</body>
</html>
```

* test_services_P4.py
* test_additional_P4.py
  ↓
  clase GameEndTests
* Peticiones AJAX → CSRF (Cross Site Request Forgy)
  - → setCookie
  - → csrfSafeMethod
  - → $.ajaxSetup
* JavaScript → move "genérico" → intercambio de obj. JSON
* Paginar Listado de juegos
* Manual de usuario → usuarioFinal

RECUERDO

1. Hacer Logout
2. Pasar Tests
   - ~~test_models~~. tests_models
   - ~~test~~ tests_function
   - test_services_P4
   - test_additional_P4 (CREAR POR PARTE NUESTRA)
     - ↓ por ejemplo: EndGameTests
     - ⇓ coverage.txt que aumente la cobertura
3. Objeto JSON → se pueden meter más atributos (por ej. winner) (opcional)
4. FILTROS
5. PROTECCIÓN URL's → CONTROL PROPIO ERROR 404

PARA 5 PTOS.

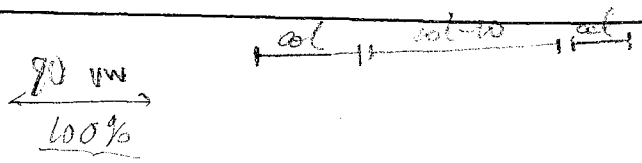
- ~~Todos~~ Entrega a tiempo
- Frontend amigable
- Heroku & local se ejecuta una partida.
- Desarrollado finalización de partida
- Las piezas se mueven con el ratón.

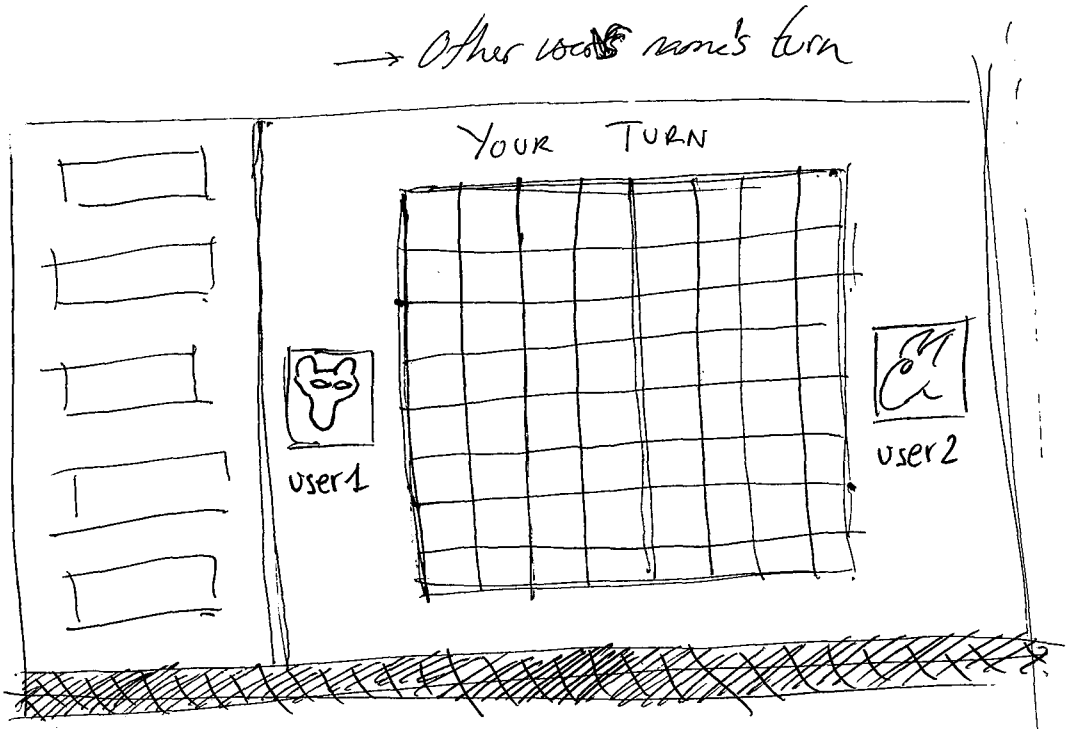
PARA 6/9 PTOS:

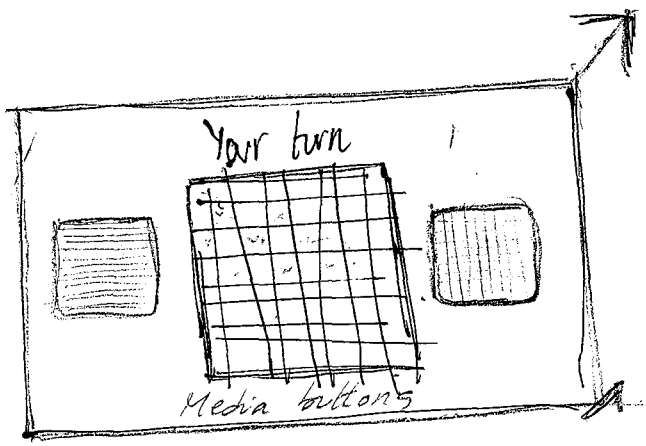
- Apartado anterior
- tests GameEndTests son correctos y completos.

Comentarios generales, corrección P3

* Tolerancia a fallos y erróneas → URLs erróneas
* Registro → Login
* Logout → sólo para usuarios login
* Repetar la ~~navegabilidad~~ lógica del juego → opciones → → navegabilidad → baja usabilidad
* Cuidado con compartir código.







test5 Estados válidos con 1 jugador. // full_clean() ⊛ save()
¿Sólo es válido CREATED?

→ si es ACTIVE o FINISHED hace raise de ValidationError.

test6 Intenta crear un juego sin el jugador 1, lanza excepción ValidationError para cualquier estado de juego. ⊛ full_clean ← explota

test7 full_clean_save()
Valida celdas válidas.
REVISAR full_clean() save() → debe corregir las inválidas (?) → no espera ningún error (?)

test8 Piezas fuera del tablero, full_clean ⊛
espera por piezas en el rango [0,63], lanza ValidationError cuando falla.

test9 Relaciones inversas (?)

test10 Posiciones no válidas dentro del tablero
⊛ save() → raises ValidationError si posiciones no válidas.

test11 save() ⊛ To string
sin ratón: ~~to~~
for (id, status) in cat [X] cat_user_test (cat1, cat2, cat3, cat4)

con ratón: añadir --- Mouse [X] mouse_user_test(59)

d20 dice

TESTS DE GAMEMODEL

test 1

```
game = Game (cat_user = self.users[0])
game.full_clean()
game.save()
self.assertIsNone (mouse.user)
self.assertEqual( 0,2,4,6,59)
```

test1 | full_clean, save

crea un juego, lo inicializa, se asegura de que el ratón jugador no esté, verifica que las posiciones son correctas, que el turno es del gato y que el estado es CREATED.

test2 | save() | aquí transiciona a ACTIVE

crea un juego con ratón y gato, lo guarda, & verifica posiciones iniciales, que el turno es del gato y que el estado es ACTIVE.

test3 | save()

más o menos test2.

test4 | ~~save()~~ full_clean(), save()

comprueba que los estados ACTIVE y FINISHED son los válidos cuando hay dos jugadores.

TESTS DE MOVEMODEL

Test 1 "Movimientos válidos"

- game.moves.count()
- n_moves

CREA MOVIMIENTOS Y COMPRUEBA EL NÚMERO.

test2 "Movimientos en un juego no activo"

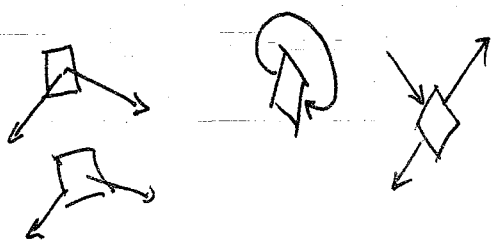
→ crea un juego sin ~~nada~~ (estado CREATED)

→ assertRaisesRegex(MSG.ERROR.MOVE).

Crear un movimiento ~~con~~ inválido (de 0 a 9 p. ejemplo)

explota en create().

~~test3~~



(1,1) → (2,2)

(1,3)

2

2%8 = 2

2//8 = 0

I hate myself

I wanna

Cause all of me

Cause all of you

D/F

2 alérgico

- check QuerySet API de Django
- check populate_rango.py

~~HH~~ 1

→ vali

11 %8 → 3

~~11%8 → 1~~

~~(2,2)~~

(2,4)

! (x//8 %2 XOR x%8 %2)

JUEGO ≡ MOUSE & CATS

	1	2	3	4	5	6	7	8
1	(1,1)	1	(1,3)	3	(1,5)	5	(1,7)	7
2	8	(2,2)	10	(2,4)	12	(2,6)	14	(2,8)
3	(3,1)	17	(3,3)	19	(3,5)	21	(3,7)	23
4	24	(4,2)	26	(4,4)	28	(4,6)	30	(4,8)
5	(5,1)	33	(5,3)	35	(5,5)	37	(5,7)	39
6	40	(6,2)	42	(6,4)	44	(6,6)	46	(6,8)
7	(7,1)	49	(7,3)	51	(7,5)	53	(7,7)	55
8	56	(8,2)	58	(8,4)	60	(8,6)	62	(8,8)

4 "gatos" → jugador 1

1 "ratón" → jugador 2

Práctica 3

- → ORM (entrega intermedia)
- → interfaz de servicios

Práctica 4

Requisitos

- Identificación de usuarios { username, password
- Creación (registro) de usuarios
- Juego simultáneo con navegadores diferentes (online)
- Se puede jugar más de una partida simultáneamer
- No se pueden realizar movimientos ilegales.
- Cuando un jugador gana → finaliza la partida
- Se guarda en BD un registro de cada partida, incluyendo cada movimiento y de quién es.
- Se puede reproducir una partida guardada en BD.

Modelos DB

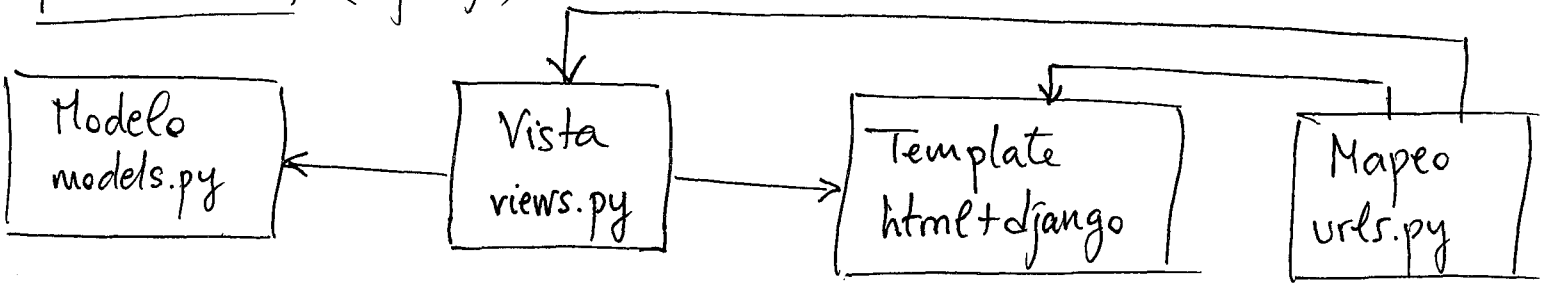
- **User**
  - username
  - password
- **Game**
  - cat_user
  - mouse_user
  - cat1
  - cat2
  - cat3
  - cat4
  - mouse
  - cat_turn
  - status
- **Move**
  - origin
  - target
  - game
  - player
  - date

Requisitos modelo de datos

- Usar clase User de Django
- Variables cat1, cat2, cat3, cat4 $\in [0,63] \cap \not\in \cap$ (casillas blanca
  Implementar restricción sobreescribiendo save()
- Status debe aceptar el conjunto predefinido de valores (variable GameStatus).
- Sobreescribir __str__ para todos los models.
- Conflicto con claves externas entre Game y User.
- Se puede ~~jugar~~ crear juego solo como "gato", pero no sin jugador "gato".
- Al realizar un movimiento, actualizar los valores del juego (usar save() de Move).

•Acceder a base de datos pc laboratorio:
psql psi alumnodb

M-V-T (Django)



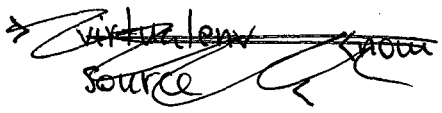
Si con los tests da un fallo de que no se ha podido crear la base de datos, necesitamos cambiar los permisos de usuario (del usuario de postgres)

#postgres# ALTER USER alumnodb CREATEDB;

CREAR CARPETA VIRTUALENV:

> virtualenv <nombre_carpeta>

ACTIVARLA:



> source <nombre_carpeta>/bin/activate

DESACTIVARLA:

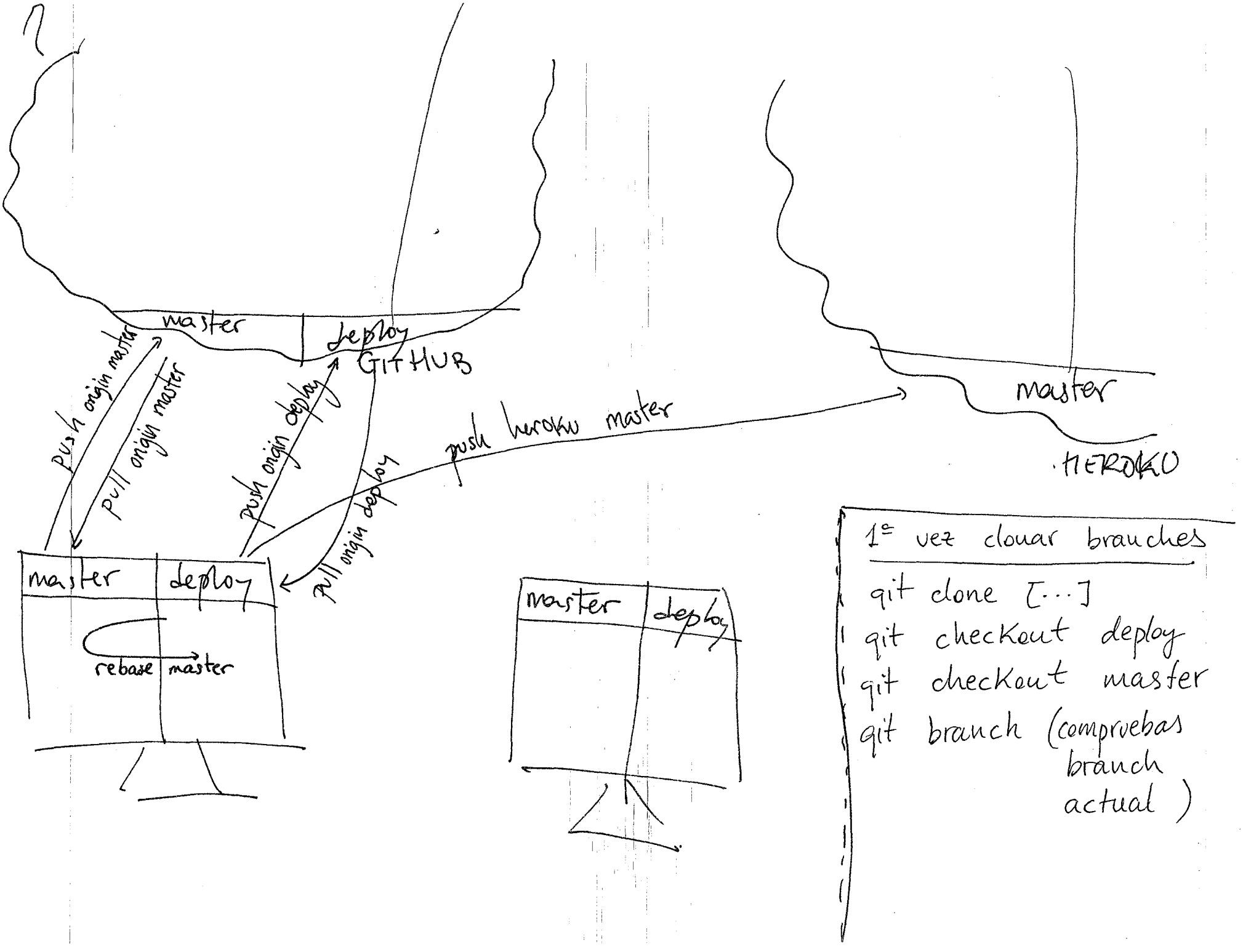
> deactivate

PASARLO A OTRO USUARIO

pip3 freeze → obtienes lo que tienes instalado

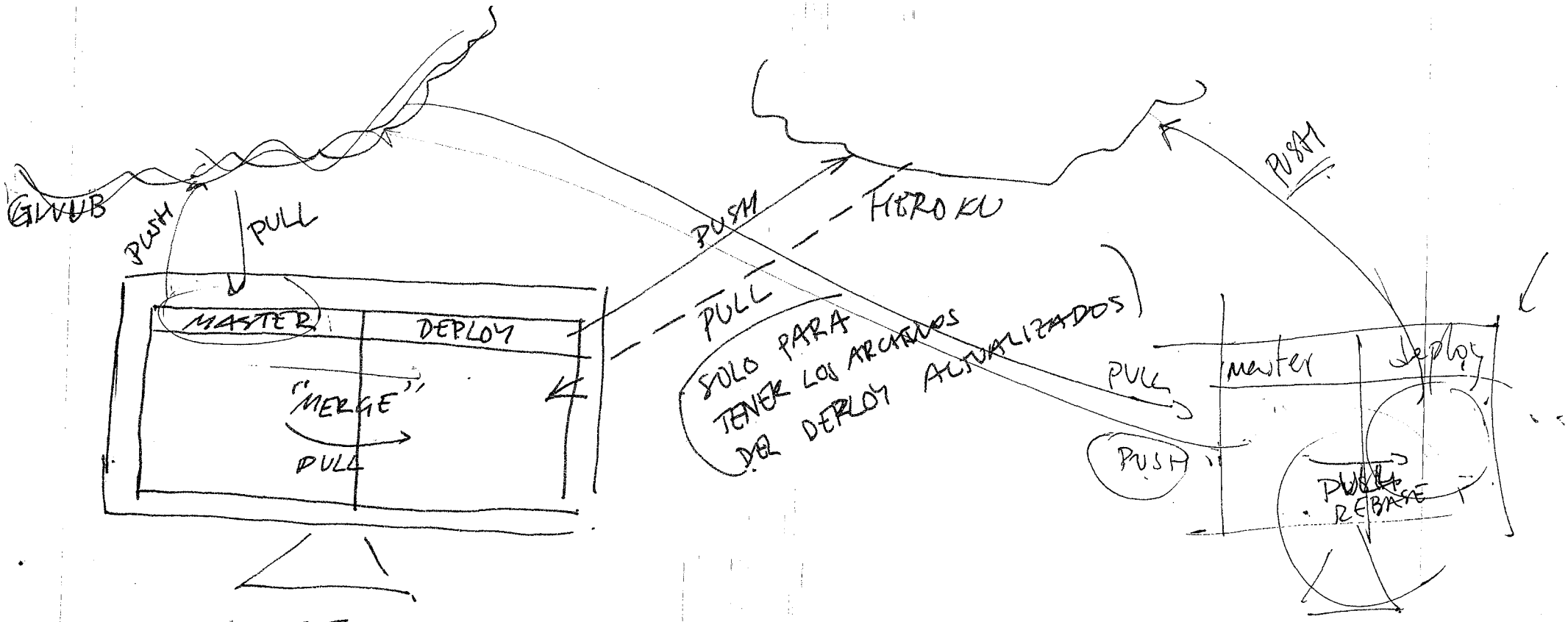
pip3 freeze > requirements.txt → se escribe en requirements.txt

pip3 install -r requirements.txt → instalarlo



1ª vez clonar branches

```
git clone [...]
git checkout deploy
git checkout master
git branch (comprobas branch actual)
```



NO ES NECESARIO TENER LA RAMA DEPLOY EN GITHUB (DE HECHO, SI NO ESTÁ MEJOR)

(master) → git push origin master

branch

(deploy) → git push heroku master

git checkout deploy

git pull origin master

git checkout -B deploy

git pull heroku master