EGC

Git

Configurar el ssh

1. Configurar usuario

```
git config --gobal user.name "Yato03"
git config --global user.email "miguelhs3523@gmail.com"
```

1. Configurar la clave (dar todo a enter)

```
 ssh-keygen -t rsa -b 4096
```

1. Visualizar la clave para copiarla y meterla en GitHub

```
 cd ~/.ssh
 cat id_rsa.pub
```

Cherry picking

1. Te cambias a la rama a la que quieres hacer el merge:

```
git checkout <rama>
```

1. Encuentra el hash del commit que quieres traerte

```
git log <la otra rama>
```

1. Realizar el cherry picking

```
git cherry-pick <hash>
```

Puede ser que al realizar *git log* no veas el commit aún. Si realizas *git status* podrás ver los archivos modificados. Para seguir debes de hacer lo siguiente:

```
# Miramos si el cambio no se ha aplicado
git status

You are currently cherry-picking commit f440bd1.
  (fix conflicts and run "git cherry-pick --continue")
  (use "git cherry-pick --skip" to skip this patch)
  (use "git cherry-pick --abort" to cancel the cherry-pick
  operation)

Unmerged paths:
  (use "git add/rm <file>..." as appropriate to mark
  resolution)
        deleted by us:   archivo # <-- Aquí está el archivo

                      # que queremos fusionar

# Añadimos el archivo
git add *
```

```
# Continuamos con el proceso
git cherry-pick --continue

# Comprobamos que ha funcionado
git log

commit 19f6ebf644daae4d2d01927868de1df315b5b7ad (HEAD ->
egc_test)
Author: yato03 <miguelhs3523@gmail.com>
Date:   Sun Jan 5 07:26:21 2025 -0500

    feat: Cambio 2 # <-- Este debe de ser el commit
                                                                    # que queríamos
```

Rebase interactivo

1. Iniciamos el rebase interactivo a partir de un commit, en el ejemplo usaremos el 4º commit desde la posición actual:

```
git rebase -i HEAD~4
```

Se abrirá un editor de texto con estos últimos 4 commits.

1. Cambiamos el tipo de los commits para fusionarlos

```
# Cambiamos los que queramos fusionar de "pick" a "squash"
pick b2a1b0b b
pick c3b2a1c c
pick d4c3b2a d
pick e5f6e78 e

# De esta manera el c y d se fusionarán con b, porque es
# el que tienen justo arriba

pick b2a1b0b b
squash c3b2a1c c
squash d4c3b2a d
pick e5f6e78 e
```

1. A continuación cambiamos la descripción del nuevo commit que contendrá el cambio de los que hemos fusionado

2. Continuamos con el proceso (esto ocurre si cambiamos la descripción)

```
git rebase --continue
```

Reset

```
git reset [modo] [commit]

# Soft -> te traes los cambios a stagging area (añadidos pa
# hacer commit)
git reset --soft HEAD~1

# Mixed -> te traes los cambios a working tree (sin estar
# añadidos)
```

```
git reset --mixed HEAD~1

# Hard -> pierdes los cambios
git reset --hard HEAD~1
```

- Deshacer un reset

```
git reset --hard f1254c5 # -> la lias con un reset

# Miras los cambios recientes
git reflog

f1254c5 (HEAD -> egc_test) HEAD@{0}: reset: moving to HEAD~1
a31d20f HEAD@{1}: reset: moving to a31d20f
f1254c5 (HEAD -> egc_test) HEAD@{2}: reset: moving to HEAD~1
a31d20f HEAD@{3}: reset: moving to a31d20f
5480540 HEAD@{4}: reset: moving to HEAD~1
f1254c5 (HEAD -> egc_test) HEAD@{5}: reset: moving to HEAD~1
a31d20f HEAD@{6}: reset: moving to HEAD
a31d20f HEAD@{7}: reset: moving to HEAD
a31d20f HEAD@{8}: reset: moving to HEAD
a31d20f HEAD@{9}: commit: feat: c
f1254c5 (HEAD -> egc_test) HEAD@{10}: commit: feat: b

# Eliges el commit que quieres y te vas a él con el reset
git reset --hard a31d20f
```

Parches

Crear el parche

```
# Parche con los cambios SOLO no añadidos
git diff > parche.patch

# Parche con los cambios SOLO añadidos
git diff --cached > parche.patch

# Parche con todos los cambios
git diff HEAD > parche.patch
```

Añadir parche

```
git apply parche.patch
```

Github Actions

Matrix en Github Actions

```
jobs:
  build:
    runs-on: ubuntu-latest
    strategy:
      fail-fast: false
      matrix:
        postgres-version: ['14.9', '15']

    services:
```

```
      postgres:
        image: postgres:14.9
        image: postgres:${{matrix.postgres-version}}
        env:
          POSTGRES_USER: decide
          POSTGRES_PASSWORD: decide
```

Automatización de releases

```
- name: Crear Release en GitHub
  id: crear_release
  uses: ncipollo/release-action@v1
  with:
    tag: ${{ github.ref_name }}
    name: "Release ${{ github.ref_name }}"
    body: |
      ## Cambios en esta versión
      - Detalle de los cambios realizados.
    draft: false
    prerelease: false
```

Docker

Pasos para que docker furule

```
# 1. Copiamos el .env
cp .env.docker.example .env

# 2. Creamos el .moduleignore (por si acaso)
echo 'webhook' > .moduleignore

# 3. Ejecutamos el docker
sudo docker compose -f docker/docker-compose.dev.yml up

# 4. Disfrutar
```

- Si te da un error de que el puerto está cogido:

```
sudo service mariadb stop
```

> Recuerda que si quieres hacer otra vez la ejecución en local lo tienes que habilitar poniendo el mismo comando pero con *start*

Quitar el modo *debug* de la ejecución en docker

En *docker/entrypoints/development_entrypoint.sh:*

```
# Al final del archivo:
exec flask run --host=0.0.0.0 --port=5000 --reload --debug

# Quitarle el --debug
exec flask run --host=0.0.0.0 --port=5000 --reload
```

Añadir workers al *guicorn*:

En *docker/entrypoints/production_entrypoint.sh:*

```
# Ultima línea
gunicorn --bind 0.0.0.0:5000 app:app --log-level info
--timeout 3600

# Añadir --workers
gunicorn --bind 0.0.0.0:5000 app:app --log-level info
--timeout 3600 --workers 4
```

> Es to junto pero no me cabía 😔

Vagrant

Ejecutar en vagrant

1. Nos quitamos de posibles errores eliminando carpetas que no vamos a usar (si las tenemos claro)

```
rm -r uploads
rm -r rosemary.egg-info
rm app.log*
```

1. Copiamos el .env

```
cp .env.vagrant.example .env
```

1. Nos movemos a la carpeta *vagrant*:

```
cd vagrant
```

1. Ejecutamos el vagrant:

```
vagrant up
```

> Al ejecutarse ya debería de estar en https://localhost:5000 pero si no es así continua con los pasos

1. Entramos en la máquina virtual

```
vagrant ssh
```

1. Ejecutamos el proyecto con flask

```
flask run --host 0.0.0.0 --reload --debug
```

Reiniciar el vagrant

```
# Paramos la máquina
vagrant halt

# Destruimos la máquina
vagrant destroy

# La volvemos a levantar
vagrant up
```

Ansible: Creación de un nuevo usuario en la base de datos

> Se usará como ejemplo el usuario *uvlhub_user*

1. En *vagrant/02_install_mariadb.yml* nos encontramos con esto:

```
# Línea 36
- name: Create .my.cnf for root
      copy:
        dest: /root/.my.cnf
        content: |
          [client]
          user=root
          password={{ mariadb_root_password }}
        owner: root
        mode: '0600'
```

1. Queremos copiarlo y modificarlo para un nuevo usuario pero si te fijas hace referencia al usuario *root* del sistema y no tenemos un usuario *uvlhub_user* asi que lo tenemos que crear primero.

2. Para hacer un nuevo usuario a nivel de sistema operativo necesitamos crearlo:

```
- name: Configure .my.cnf for uvlhub_user
  hosts: all
  become: yes
  tasks:
    - name: Ensure uvlhub_user exists
      user:
        name: uvlhub_user
        state: present
        shell: /bin/bash

- name: Create home directory for uvlhub_user
      file:
        path: /home/uvlhub_user
        state: directory
        owner: uvlhub_user
        group: uvlhub_user
        mode: '0755'
```

1. Quedaría así:

```
- name: Configure .my.cnf for uvlhub_user
  hosts: all
  become: yes
  tasks:
    - name: Ensure uvlhub_user exists
      user:
        name: uvlhub_user
        state: present
        shell: /bin/bash

    - name: Create home directory for uvlhub_user
      file:
        path: /home/uvlhub_user
        state: directory
        owner: uvlhub_user
        group: uvlhub_user
        mode: '0755'
```

```
    - name: Create .my.cnf for uvlhub
      copy:
        dest: /home/uvlhub_user/.my.cnf
        content: |
          [client]
          user=uvlhub_user
          password={{ mariadb_uvlhub_password }}
        owner: uvlhub_user
        group: uvlhub_user
        mode: '0600'
```

Cambiar la versión de ubuntu:

1. En el *vagrant/Vagrantfile* nos encontraremos con esto:

```
Vagrant.configure("2") do |config|

  # Choose a base box
  config.vm.box = "ubuntu/jammy64"
```

1. Cambiar la versión por otra:

```
Vagrant.configure("2") do |config|

  # Choose a base box
  config.vm.box = "ubuntu/focal64"
```

Cambiar la memoria y cpu de la máquina virtual

1. En el *vagrant/VagrantFile* nos encontraremos con esto:

```
# Provider configuration
  config.vm.provider "virtualbox" do |vb|
    vb.memory = "2048"
    vb.cpus = 4
  end
```

1. Modificar a los valores que nos diga. Ejemplo:

```
# Provider configuration
  config.vm.provider "virtualbox" do |vb|
    vb.memory = "4096"
    vb.cpus = 2
  end
```

Cambiar configuración de ejecución

1. Todo lo relativo a la ejecución está en *vagrant/05_run_app.yml:*

```
# Al final del todo
- name: Run Flask application
  shell: |
    source {{ working_dir }}vagrant_venv/bin/activate
    cd {{ working_dir }}
    nohup flask run --host=0.0.0.0 --port=5000 --reload
    --debug > app.log 2>&1 &
  args:
```

```
  executable: /bin/bash
environment: "{{ common_environment }}"
async: 1
poll: 0
```

1. Cambiar lo pertinente en el *flask run*

Render

https://docs.uvlhub.io/deployment/render