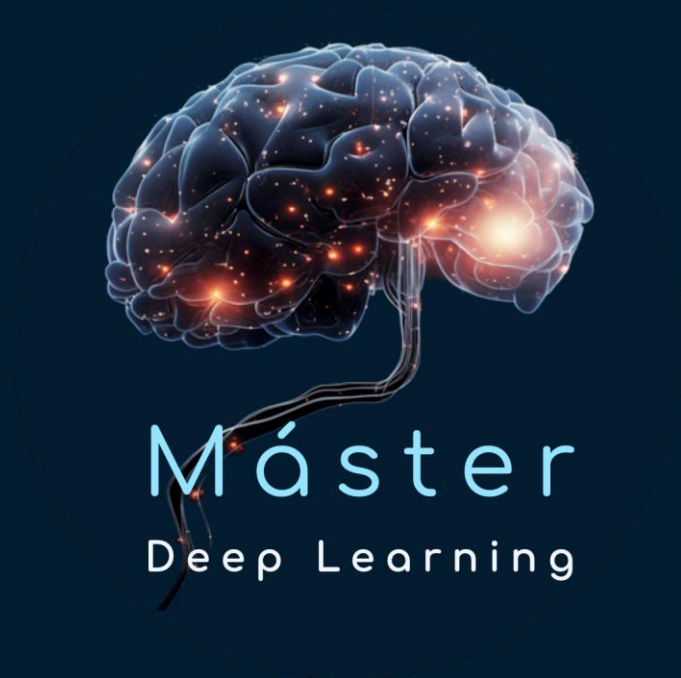


MLOps

Tema 2.

**Entorno de desarrollo y
control de versiones con Git**



POLITÉCNICA

UNIVERSIDAD
POLITÉCNICA
DE MADRID



Máster
Deep Learning

Desarrollo de ML

“Si solo tienes tiempo para configurar adecuadamente una parte de la infraestructura, que sea el entorno de desarrollo para los científicos de datos. Esto se debe a que el entorno de desarrollo es donde trabajan los ingenieros, y cualquier mejora en ese entorno se traduce directamente en un aumento de la productividad de ingeniería”

Ville Tuulos, autor del libro “Effective Data Science Infrastructure”

Estructura de proyecto ML

► Estimación de precio de una casa

```
upm-mdl-mlops
├── artifacts
│   ├── encoders.pkl      # Encoders de variables categóricas
│   ├── model.pkl         # Modelo entrenado (LinearRegression)
│   └── scaler.pkl        # Scaler para variables numéricas
├── data
│   └── Housing.csv       # Dataset de ejemplo
├── notebooks
│   └── modeling.ipynb    # Notebook con análisis y experimentos
├── src
│   ├── inference_api.py  # Script FastAPI para servir inferencias
│   └── train.py          # Script para entrenar y guardar el modelo
└── test
    └── test_inference_api.py # Tests de la API de inferencia
```



Entorno de desarrollo ML

- ▶ **JupyterLab, Google Colab, etc...**
- ▶ **Entornos de desarrollo integrados (IDEs)**
 - ▶ PyCharm, Spyder, Atom, Sublime Text...
 - ▶ **Visual Code Studio** ([download](#))
 - ◆ Altamente configurable
 - ◆ Programación en diferentes lenguajes
 - ◆ Extensiones (Jupyter, Git, Docker, Remote Dev, etc.)



Visual Code Studio

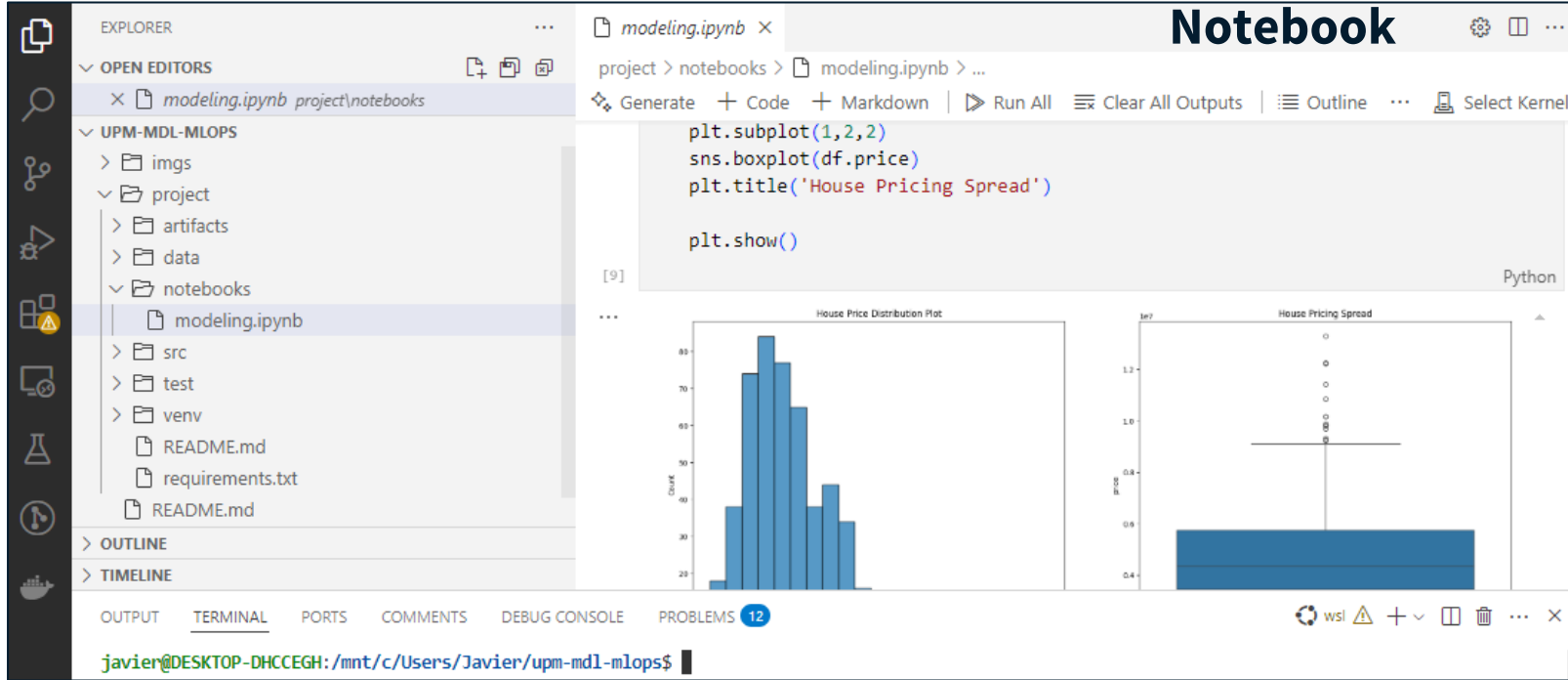
Project structure

Notebook

Git >

RemoteDev >

Docker >



The screenshot displays the Visual Studio Code interface with the following components:

- EXPLORER (Project structure):**
 - OPEN EDITORS: modeling.ipynb project\notebooks
 - UPM-MDL-MLOPS
 - imgs
 - project
 - artifacts
 - data
 - notebooks
 - modeling.ipynb (selected)
 - src
 - test
 - venv
 - README.md
 - requirements.txt
 - README.md
 - OUTLINE
 - TIMELINE
- Notebook (modeling.ipynb):**
 - Code cells:

```
plt.subplot(1,2,2)
sns.boxplot(df.price)
plt.title('House Pricing Spread')

plt.show()
```
 - Output: Two plots are shown. The first is a histogram titled "House Price Distribution Plot" with "Count" on the y-axis (0 to 80) and price on the x-axis. The second is a boxplot titled "House Pricing Spread" with "price" on the y-axis (0.4 to 1.2).
- Terminal:**
 - Output: javier@DESKTOP-DHCCEGH:/mnt/c/Users/Javier/upm-mdl-mlops\$

Terminal

Estandarización del entorno



Antes de trabajar...

- ▶ Si los programadores trabajan en sus equipos
 - ▶ Deben trabajar con la misma configuración
 - ▶ Especificar versiones exactas de paquetes y Python
 - ▶ Evitar problemas al integrar cambios
- ▶ Alternativa en un entorno cloud:
 - ▶ Facilita el soporte IT
 - ▶ Permite trabajo remoto (SSH)
 - ▶ Seguridad centralizada
 - ▶ Reduce brecha entre desarrollo y producción

Estandarización del entorno

- Estandarización del entorno de desarrollo
 - ▶ Entorno virtual en Python y requirements.txt
 - ◆ Conda, virtualenv, etc...

```
pandas==2.2.0
scikit-learn==1.3.2
wandb==0.15.12
matplotlib==3.7.2
jupyter==1.0.0
fastapi==0.115.11
uvicorn==0.34.0
joblib==1.3.2
seaborn==0.13.2
pyarrow==19.0.1
```

requirements.txt

```
$ sudo apt-update
$ sudo apt install python3.10-venv
$ python3 -m venv venv
$ source venv/bin/activate
$ (venv) pip install -r requirements.txt
```

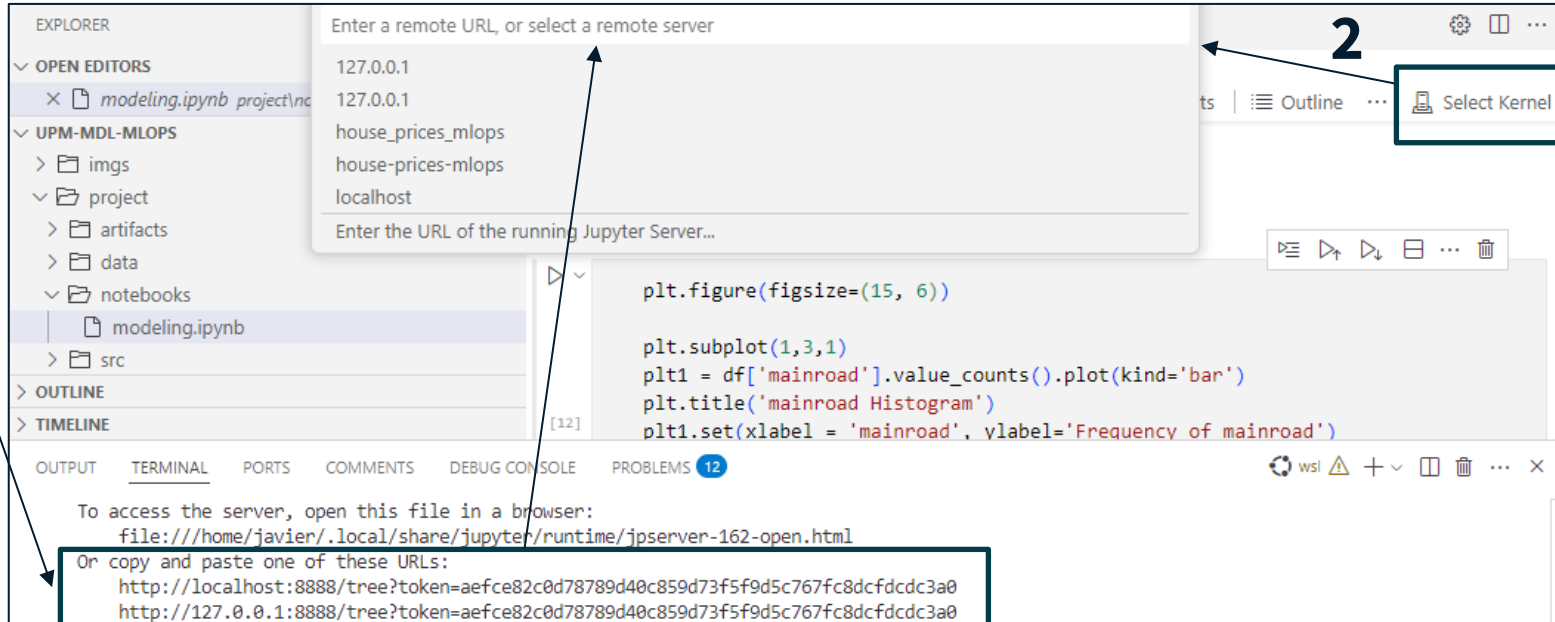


CREAR VENV E INSTALAR DEPENDENCIAS

Exploración y experimentación

```
$ (venv) jupyter-notebook  
// copiar URL servidor en VSCode
```

1



EXPLORER

- OPEN EDITORS
 - modeling.ipynb project\nc
- UPM-MDL-MLOPS
 - imgs
 - project
 - artifacts
 - data
 - notebooks
 - modeling.ipynb
 - src
- OUTLINE
- TIMELINE

Enter a remote URL, or select a remote server

- 127.0.0.1
- 127.0.0.1
- house_prices_mlops
- house-prices-mlops
- localhost

Enter the URL of the running Jupyter Server...

2

Select Kernel

```
plt.figure(figsize=(15, 6))  
  
plt.subplot(1,3,1)  
plt1 = df['mainroad'].value_counts().plot(kind='bar')  
plt.title('mainroad Histogram')  
plt1.set(xlabel = 'mainroad', ylabel='Frequency of mainroad')
```

OUTPUT TERMINAL PORTS COMMENTS DEBUG CONSOLE PROBLEMS 12

To access the server, open this file in a browser:
file:///home/javier/.local/share/jupyter/runtime/jpserver-162-open.html
Or copy and paste one of these URLs:
<http://localhost:8888/tree?token=aefce82c0d78789d40c859d73f5f9d5c767fc8dcfdcdc3a0>
<http://127.0.0.1:8888/tree?token=aefce82c0d78789d40c859d73f5f9d5c767fc8dcfdcdc3a0>

**LANZAR JUPYTER SERVER Y EJECUTAR NOTEBOOK**

Entrenamiento de modelo y artefactos



```
$ (venv) python3 src/train.py
```

- Carga el dataset de casas y sus precios (/data)
- Preprocesa el dataset, codifica y escala
- Entrena modelo linear de regresión con conjunto train
- Evalúa el modelo con conjunto test
- Guarda artefactos en /artifacts
 - Modelo entrenado, escaler y encoder



Inferencia con nuevos datos



```
$ (venv) python3 src/inference_api.py
```

- ▶ Lanza servidor **uvicorn** con el modelo previamente entrenado (artifacts/model.pkl) mediante **fastapi**
 - ▶ <http://localhost:8000/predict>
- ▶ Cliente puede lanzar HTTP Post con los datos de una casa y el servicio responde con la estimación del precio
 - ▶ La petición debe ser codificada y escalada antes de inferir

```
$ (venv) python3 test/test_inference_api.py
```

- ▶ Ejecuta tests unitarios para probar el servicio
- ▶ Utiliza **fastapi.testclient** de manera sencilla para ello



Ejemplo de llamada para inferencia



```
$ curl -X POST \  
  -H "Content-Type: application/json" \  
  -d '{  
    "area": 5000,  
    "bedrooms": 4,  
    "bathrooms": 3,  
    "stories": 2,  
    "mainroad": "yes",  
    "guestroom": "yes",  
    "basement": "yes",  
    "hotwaterheating": "yes",  
    "airconditioning": "no",  
    "parking": 2,  
    "prefarea": "no",  
    "furnishingstatus": "semi-furnished"  
  }' \  
  http://localhost:8000/predict
```



Control de versiones

Sistema de control de versiones

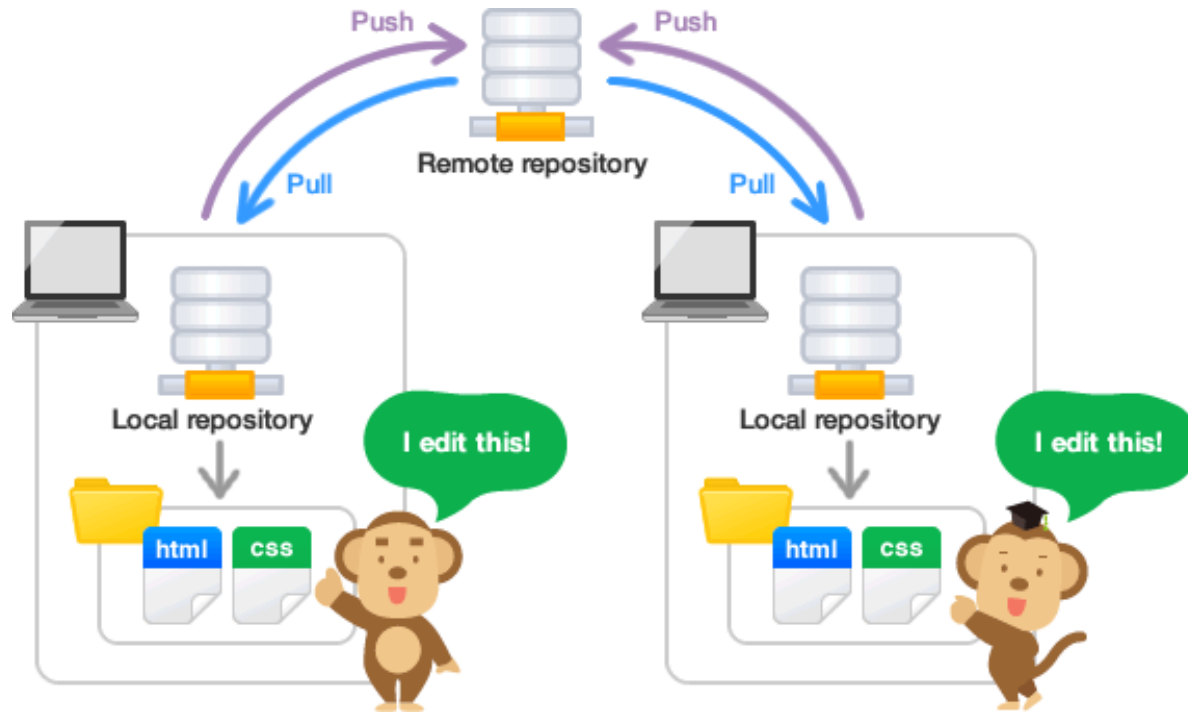


Funciones deseables para un proyecto de software y ML:

- Versionar los ficheros a lo largo del tiempo
- Compartir ficheros
- Crear copias de seguridad y restaurarlas
- Sincronizar desarrolladores a la última versión
- Deshacer cambios
- Realizar desarrollos alternativos (ramas)



- ▶ Git es el **protocolo** estándar para el control de versiones
 - ▶ Usado por muchas empresas y organizaciones
 - ▶ Mantiene el repositorio en tu máquina local
 - ▶ Sincroniza repositorio local con repositorio en servidor
- ▶ Existen plataformas para alojar repositorios Git
 - ▶ [GitHub](#) y [GitLab](#) son las más populares
 - ◆ Interfaz web
 - ◆ Pull/Merge requests
 - ◆ Issues y Wikis
 - ◆ Integraciones CI/CD...
 - ◆ Gestión de equipos y permisos...



- ▶ Si cambias de máquina, puedes descargar del servidor
- ▶ Si un compañero hace modificaciones, puedes sincronizar
- ▶ Cada versión (*commit*) queda etiquetada en un histórico

Instalación de Git



1. Básico (terminal)

- ▶ Descarga e instala Git desde git-scm.com

2. Entornos gráficos

- ▶ GitHub Desktop, Tortoise, GitKraken, ...

3. Integrado en IDEs

- ▶ Eclipse, Xcode, **Visual Code Studio...**



Configuración de Git

1. Crear una cuenta en [GitHub](#)
 - ▶ Píde tu licencia [GitHub PRO](#) como alumno UPM
2. Configura tu nombre y correo (se usarán en tus commits)

```
$ git config --global user.name "Tu Nombre"  
$ git config --global user.email tuemail@example.com  
$ git config -l
```



Metodología básica con Git

Metodología básica



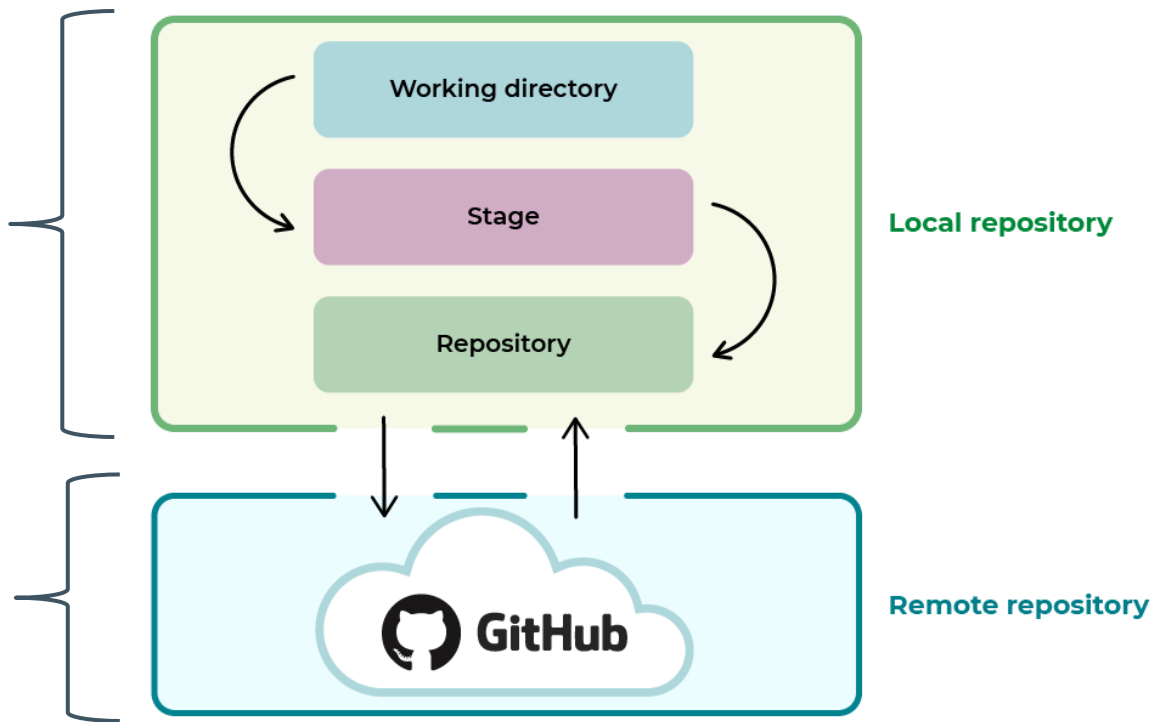
- ▶ Primera vez:
 1. Iniciar/clonar el repositorio Git del proyecto
 2. Añadir proyecto (archivos) al repositorio local
 3. Asociar repo local con el remoto
- ▶ Siguiendo ocasiones:
 4. Actualizar el repositorio local
 5. Modificar el repositorio local
 6. Envío de cambios al repositorio remoto

Repositorios locales y remoto



Cada desarrollador
trabaja en su
repositorio local

Se mantiene el
repositorio remoto
común



Creación del repositorio Git

- ▶ *Primera vez:*
 1. *Iniciar/clonar el repositorio Git del proyecto*
 2. *Añadir proyecto (archivos) al repositorio local*
 3. *Asociar repo local con el remoto*

1. Iniciar el repositorio Git



Sólo se realiza una vez en el ciclo de vida de un proyecto

1. Crea un repositorio Git vacío (inicio de proyecto desde cero)

1. Accede a tu GitHub y crea un repositorio remoto vacío
2. Copia la URL del repositorio remoto `https://github.com/tu_usuario/repo.git`
3. Navega al proyecto e inícialo como repositorio Git

```
$ cd /ruta/a/tu/proyecto  
$ git init
```

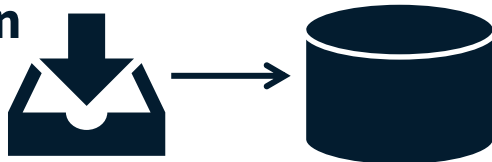
2. Obtener una copia local de un repositorio remoto existente

```
$ cd /ruta/donde/guardar  
$ git clone <URL-del-repositorio>
```



1. Iniciar el repositorio Git

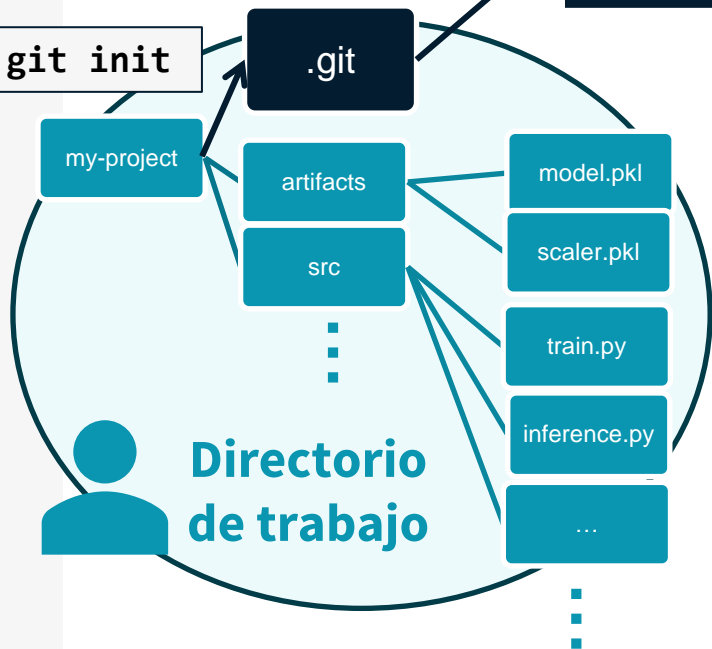
Area de preparación



Repositorio local Git

- ▶ Mini sistema de ficheros
- ▶ Guarda todos los detalles
- ▶ Complicado y optimizado
- ▶ El *cliente git* gestiona esto por nosotros
 - ▶ Terminal, pluggins, ...

```
$ git init
```



El programador edita sobre su directorio de trabajo

2. Añadir proyecto al repo local



Los archivos están en nuestro directorio pero no en el repositorio local

3. Agregar archivos al área de preparación (*staging area*)

```
$ git add nombre_del_archivo
```

Fichero particular

```
$ git add .
```

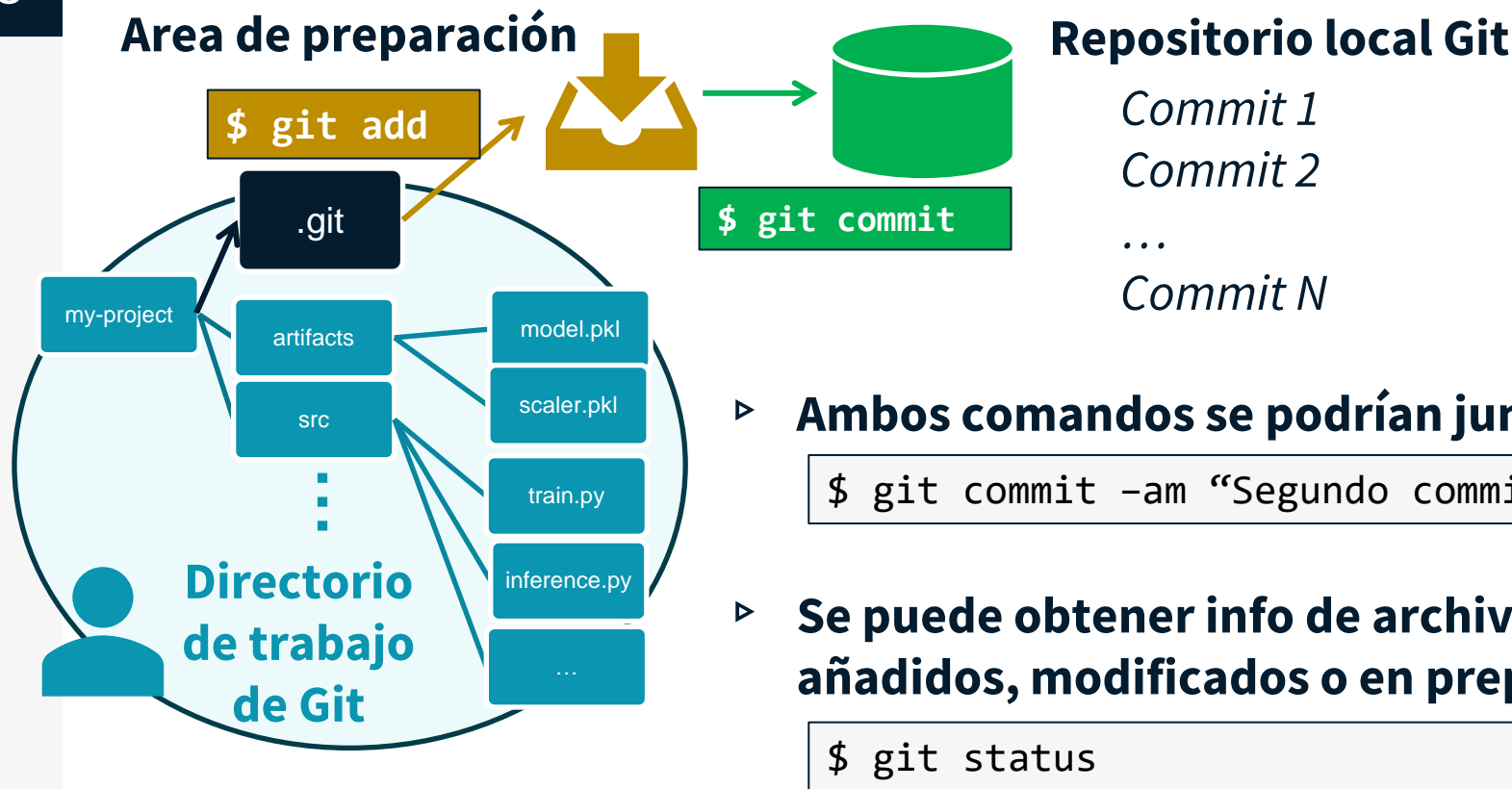
Todos los ficheros

4. Confirmar los archivos en preparación al repositorio local

```
$ git commit -m "Comentario del primer commit"
```

5. Si añades/cambias ficheros, se vuelve a realizar ambos pasos

2. Añadir proyecto al repo local



- ▶ **Ambos comandos se podrían juntar**

```
$ git commit -am "Segundo commit"
```

- ▶ **Se puede obtener info de archivos no añadidos, modificados o en preparación**

```
$ git status
```

2. Añadir proyecto al repo local



- ▶ Quizás no queramos añadir determinados ficheros:
 - ▶ La carpeta del entorno virtual **venv**
 - ▶ Archivos de configuración o compilados
 - ▶ Ficheros de log, caché o checkpoints
- ▶ Hay que añadir el fichero **.gitignore**

```
__pycache__/  
*.py[cod]  
*$py.class  
venv/  
env/  
.venv/  
.ipynb_checkpoints/  
...
```



3. Asociar repo local con remoto



¡Hasta ahora hemos trabajado siempre en local!

6. Asociar el repositorio local con el remoto

```
$ git remote add origin https://github.com/tu_usuario/repo.git
```

7. Crear token (classic) en GitHub con permisos “repo” y “workflow”

- ▶ <https://github.com/settings/tokens/new>

8. Subir el proyecto al repositorio remoto (rama “master”)

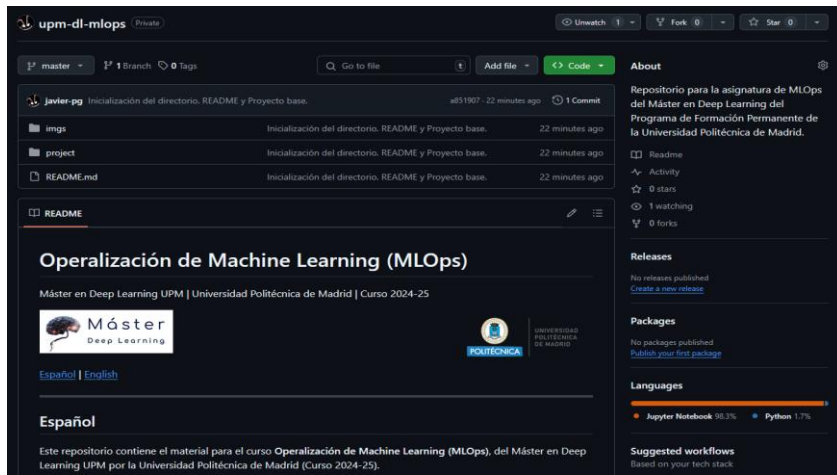
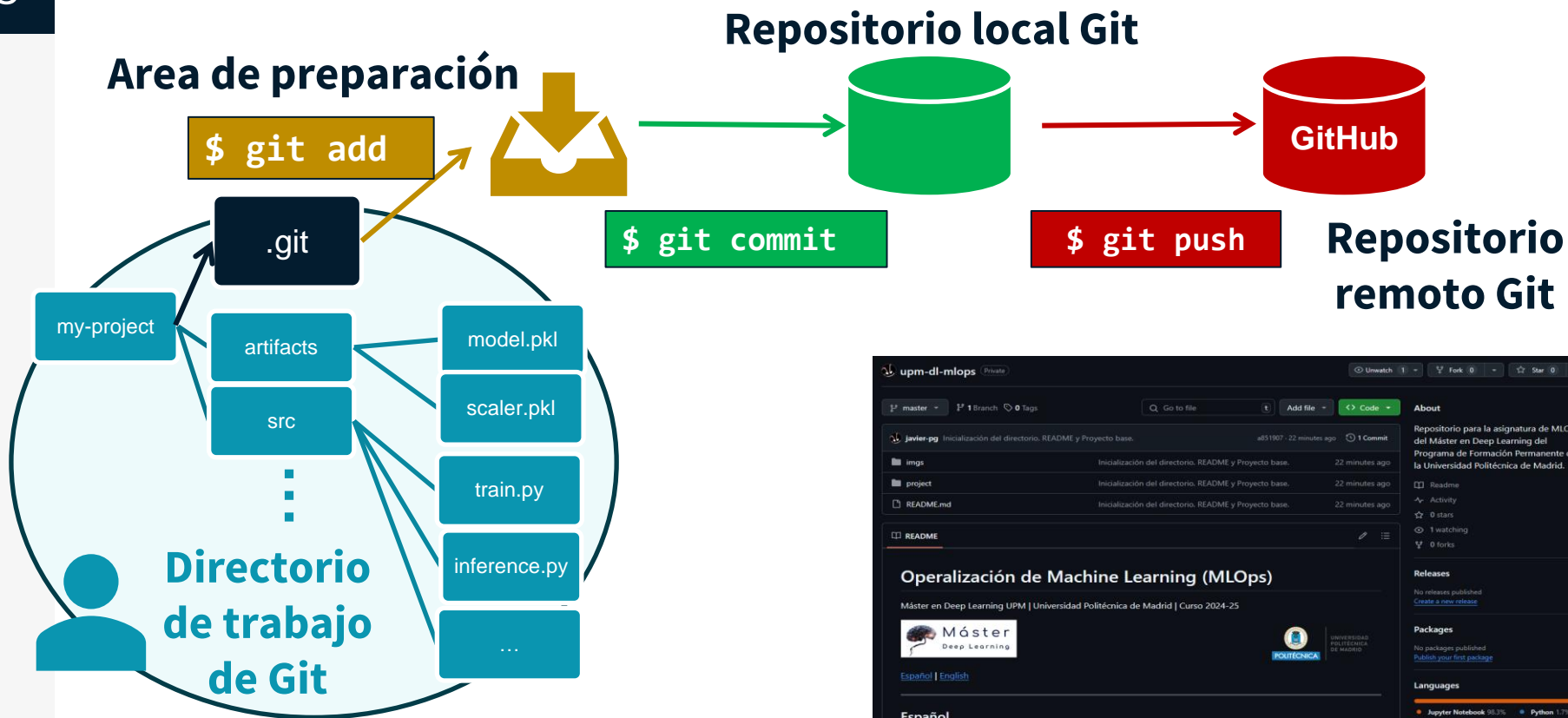
```
$ git push -u origin master
```

- ▶ *Y autenticarse con correo electrónico y token creado (password)*
- ▶ *Si no se quiere introducir constantemente, usar [GCM \(o osxkeychain\)](#)*

```
$ git config --global credential.helper 'cache --timeout=900'
```



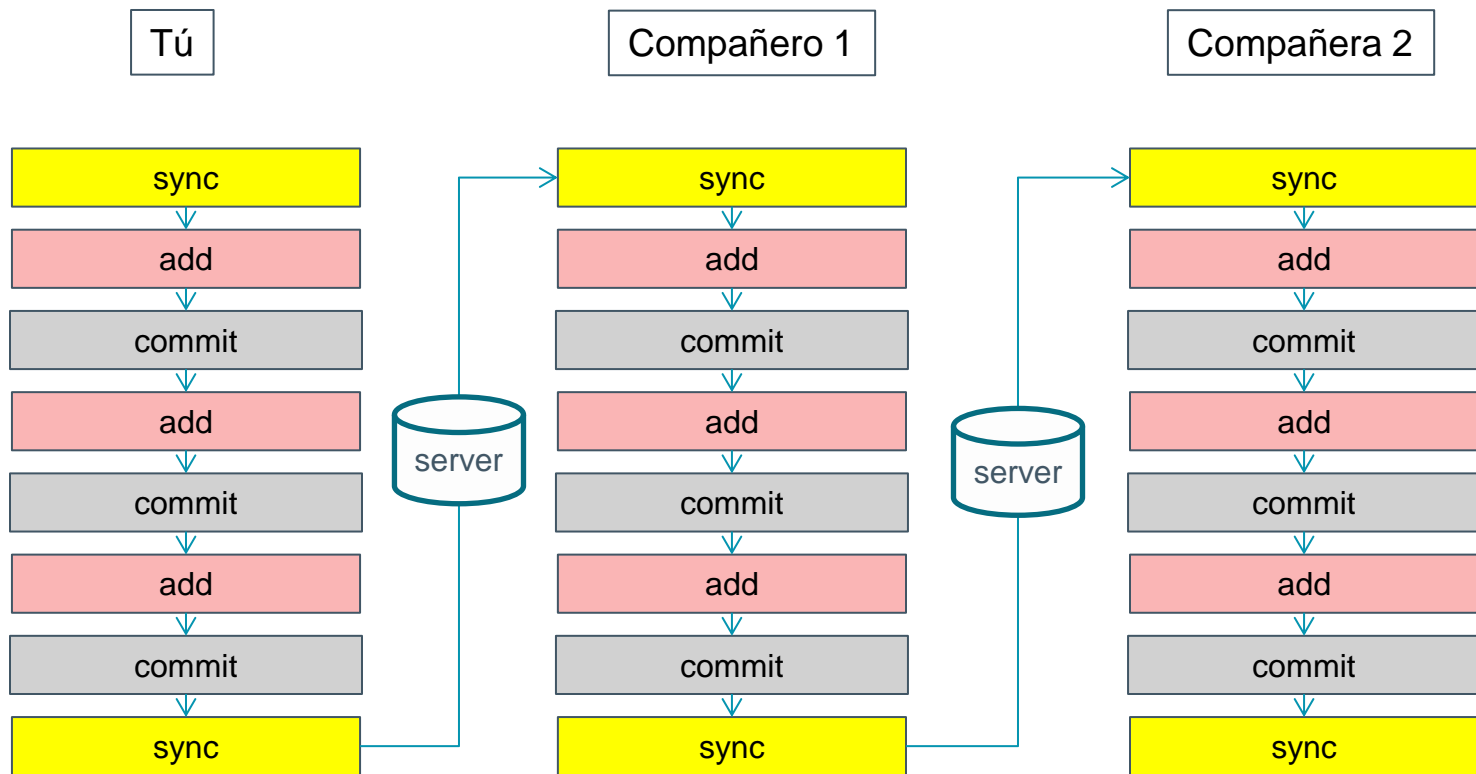
3. Asociar repo local con remoto



Trabajo básico con el repositorio Git

- ▶ *Una vez existe el repositorio remoto Git:*
 1. *Actualizar el repositorio local*
 2. *Trabajar en el repositorio local*
 3. *Envío de cambios al repositorio remoto*

Trabajo colaborativo



1. Actualizar el repositorio local



Te toca continuar programando en un proyecto (colaborativo)

1. Actualizar el repo local con los posibles cambios del repositorio remoto

```
$ git pull
```

a) Se integran directamente los cambios en nuestro directorio y repositorio ✓

b) Surgen conflictos entre nuestra versión y la remota que descargamos ✗

- ◆ Git identifica cada archivo conflictivo y lo edita con “<<<<<< ,====, >>>>>>”

```
<<<<<< HEAD  
código de tu repo local  
=====  
código del repo remoto  
>>>>>> nombre_de_la_rama_o_commit
```

- ◆ Editar cada archivo eligiendo/fusionando cambios
- ◆ Añade y confirma los archivos en el repo local

```
$ git add nombre_del_archivo  
$ git commit -m “Resolver conflictos”
```

1. Actualizar el repositorio local



2. Trabajar en el repo local

Estamos sincronizados con el repositorio remoto

2. Realiza cambios en tus archivos dentro del directorio

- ▶ Trabajamos con nuestro IDE o herramientas favoritas
- ▶ Añadir gradualmente cambios completos o relacionados

```
$ git add nombre_del_archivo
```
- ▶ Realizar commits frecuentes y pequeños con mensajes claros y concisos

```
$ git commit -m "Comentario descriptivo"
```
- ▶ Verificar el estado antes de hacer commit

```
$ git status
```
- ▶ El repositorio local irá registrando todos los commits



Haz cambios, añade o elimina ficheros

3. Envío de cambios al repo remoto

Cuando hayamos terminado una pequeña funcionalidad

3. Enviar los cambios al repositorio remoto

```
$ git push
```

a) Se integran directamente los cambios en el repo remoto ✓

b) Git detecta conflictos en una misma parte del código porque alguien ha modificado al mismo tiempo ✗

- ◆ Fusionamos manualmente los cambios y se confirman (como en **1. b)**)
- ◆ Se intenta nuevamente el push

```
$ git push
```



Otras funcionalidades

Ramas (branches)

Líneas de desarrollo independientes para trabajar en nuevas funcionalidades, correcciones o experimentos sin afectar la rama principal

- ▶ Listar ramas locales

```
$ git branch
```

- ▶ Crear y cambiar a una nueva rama

```
$ git checkout -b nombre_rama
```

- ▶ Cambiar de rama

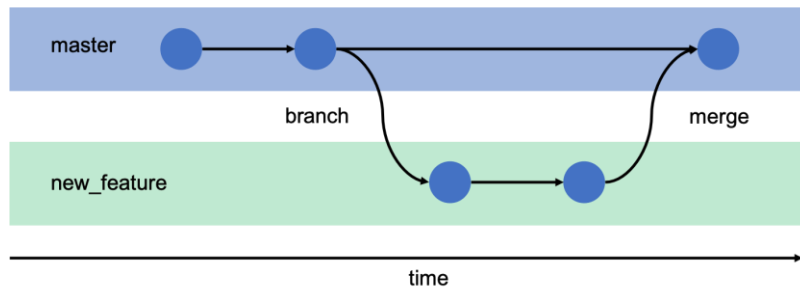
```
$ git checkout nombre_rama
```

- ▶ Fusionar una rama en la actual

```
$ git merge nombre_rama
```

- ▶ Eliminar una rama

```
$ git branch -d nombre_rama
```



Tags / Releases

Marcadores inmutables que señalan puntos específicos (como versiones estables) en la historia de un proyecto

- ▶ Crear un tag anotado (por ejemplo, v1.0) asociado al último commit

```
$ git tag -a v1.0 -m "Lanzamiento versión 1.0"
```

- ▶ Listar todos los tags

```
$ git tag
```

- ▶ Enviar un tag al repositorio remoto (que estará asociado a un commit)

```
$ git push origin v1.0
```

origin = <URL del repositorio remoto>

- ▶ Crear un release

- ▶ En plataformas como GitHub o GitLab, se puede usar un tag para crear un release, añadiendo notas, binarios u otra información relevante. Esto formaliza el lanzamiento basado en el tag. Se realiza desde la interfaz web del repositorio.

Cheat Sheet



Create a Repository

From scratch -- Create a new local repository

```
$ git init [project name]
```

Download from an existing repository

```
$ git clone my_url
```

Observe your Repository

List new or modified files not yet committed

```
$ git status
```

Show the changes to files not yet staged

```
$ git diff
```

Show the changes to staged files

```
$ git diff --cached
```

Show all staged and unstaged file changes

```
$ git diff HEAD
```

Show the changes between two commit ids

```
$ git diff commit1 commit2
```

List the change dates and authors for a file

```
$ git blame [file]
```

Show the file changes for a commit id and/or file

```
$ git show [commit]:[file]
```

Show full change history

```
$ git log
```

Show change history for file/directory including diffs

```
$ git log -p [file/directory]
```

Working with Branches

List all local branches

```
$ git branch
```

List all branches, local and remote

```
$ git branch -av
```

Switch to a branch, my_branch, and update working directory

```
$ git checkout my_branch
```

Create a new branch called new_branch

```
$ git branch new_branch
```

Delete the branch called my_branch

```
$ git branch -d my_branch
```

Merge branch_a into branch_b

```
$ git checkout branch_b
```

```
$ git merge branch_a
```

Tag the current commit

```
$ git tag my_tag
```

Make a change

Stages the file, ready for commit

```
$ git add [file]
```

Stage all changed files, ready for commit

```
$ git add .
```

Commit all staged files to versioned history

```
$ git commit -m "commit message"
```

Commit all your tracked files to versioned history

```
$ git commit -am "commit message"
```

Unstages file, keeping the file changes

```
$ git reset [file]
```

Revert everything to the last commit

```
$ git reset --hard
```

Synchronize

Get the latest changes from origin (no merge)

```
$ git fetch
```

Fetch the latest changes from origin and merge

```
$ git pull
```

Fetch the latest changes from origin and rebase

```
$ git pull --rebase
```

Push local changes to the origin

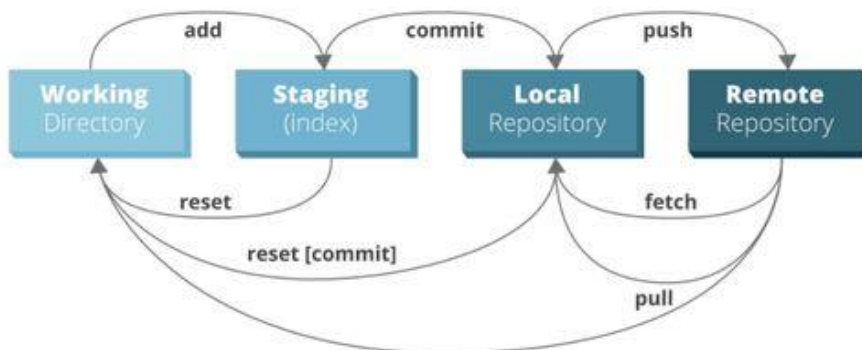
```
$ git push
```

Finally!

When in doubt, use git help

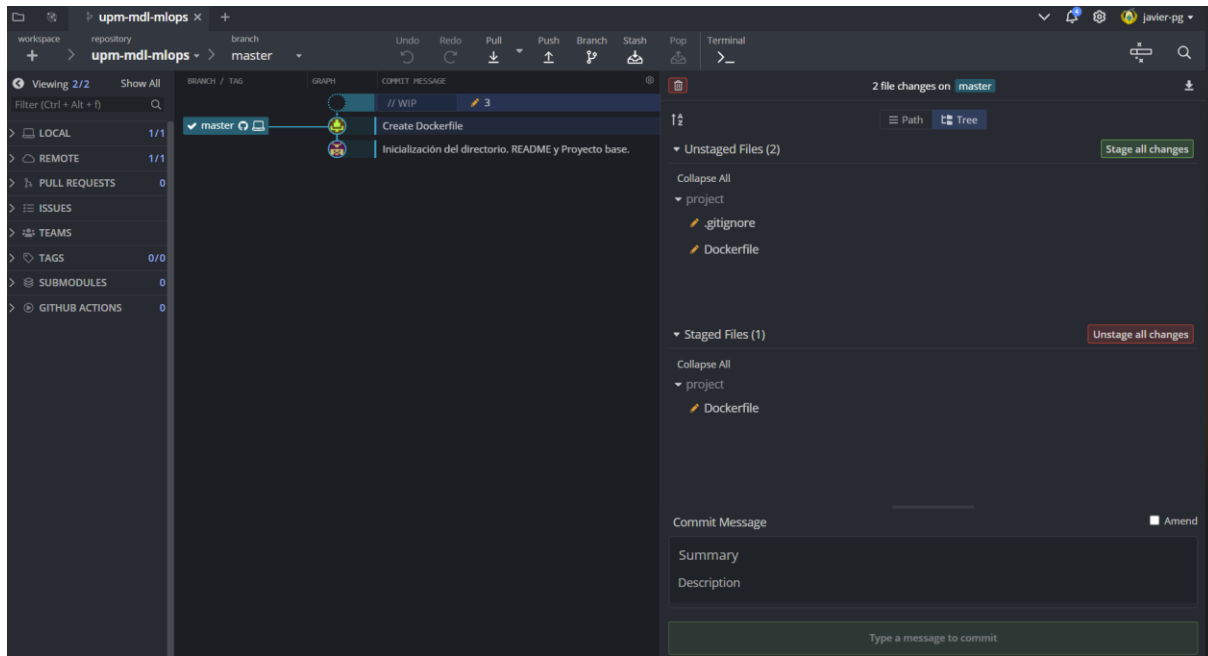
```
$ git command --help
```

Or visit <https://training.github.com/> for official GitHub training.



Herramientas cliente de Git

Existen herramientas e interfaces visuales para facilitar el uso y la gestión de Git, si bien estos se basan en los comandos previamente mencionados y la carpeta .git.

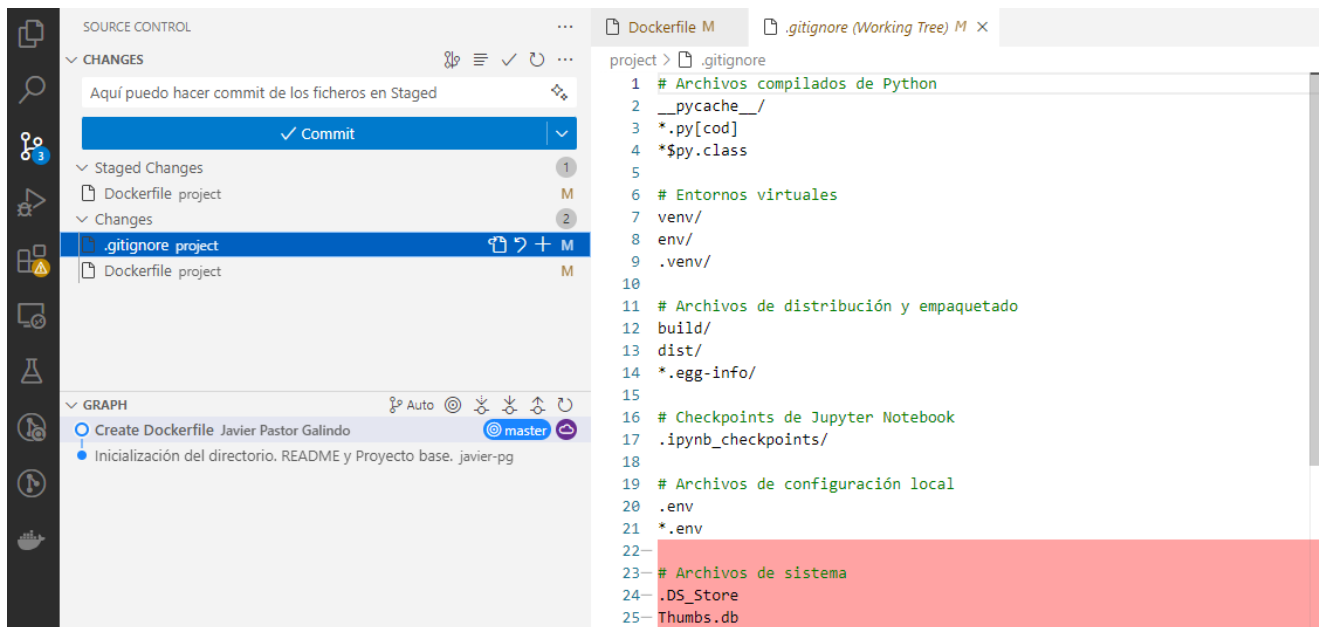


GitKraken

Plugins para Git



También existen integraciones con IDE para hacer más cómodo su uso en conjunción con el desarrollo.



Plugin para Visual Code Studio

Próximas clases

Desde el desarrollo a la producción



- ▶ Git facilita el **desarrollo** en local y colaboración remota.
- ▶ El código funciona en mi ordenador, ¿pero en el servidor o cloud también funcionaría?
 - ▶ Contenedores portables para reproducir entornos de ejecución → **Docker**
- ▶ Una vez aseguro que el proyecto funciona en producción, ¿actualizo, testeo y despliegue manualmente en cada cambio?
 - ▶ Pipelines automáticos de integración y despliegue automático → **GitHub Actions**

- ▶ Huyen, C. (2022). *Designing machine learning systems*. O'Reilly Media, Inc.
- ▶ Tuulos, V. (2022). *Effective Data Science Infrastructure. How to make data scientists productive*.
- ▶ Yasser, M (2022). Kaggle Datasets. *Housing Prices Prediction - Regression Problem*.
- ▶ Setu, S (2024). Kaggle Code. *House Pricing – Regression*.
- ▶ Martínez, Iván (2017). *Taller de Git y GitHub desde Cero. Facultad de Informática, Universidad Complutense de Madrid*.
- ▶ Wand, M (2012). *A Simple Introduction to Git: a distributed version-control system*. Northeastern University.