

DAW

Despliegue de aplicaciones web

1. Introducción

Juan Jesús Tortajada Cordero

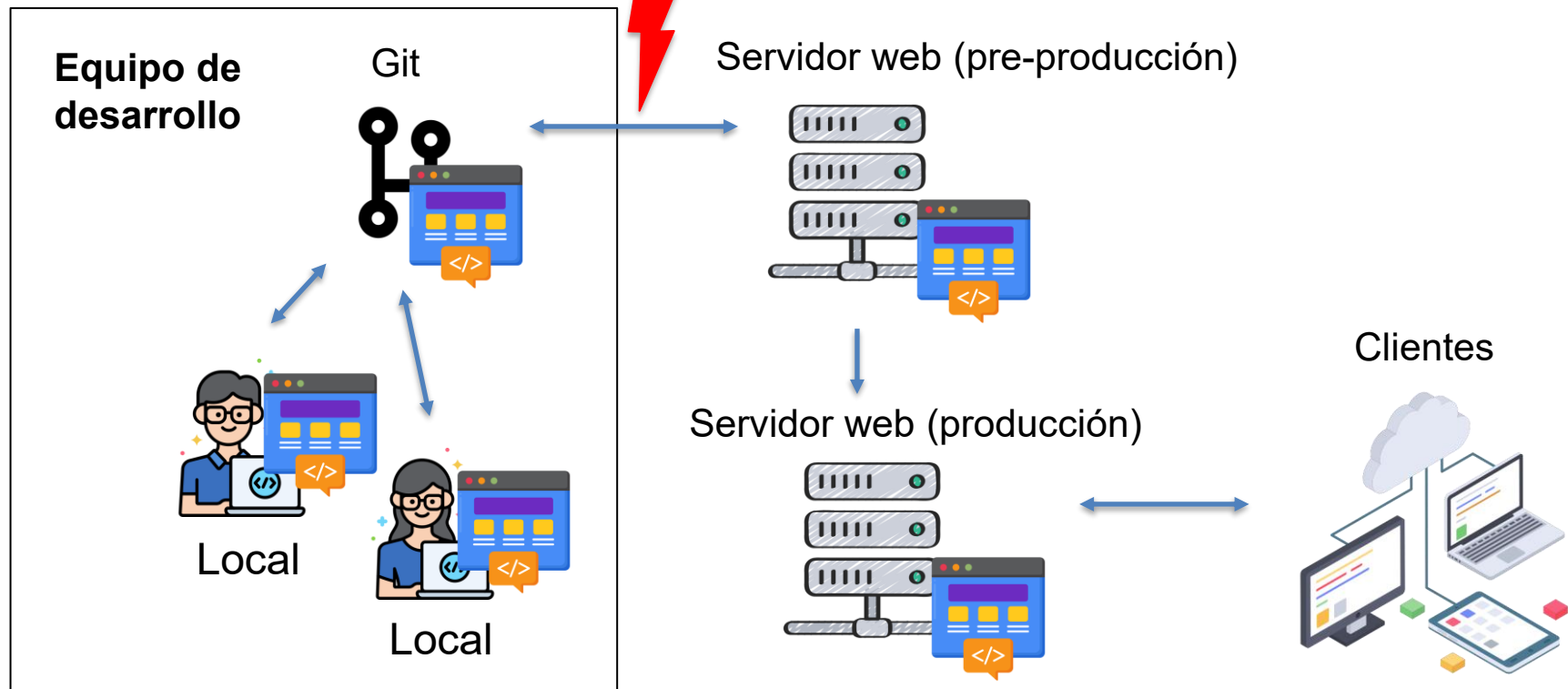
jtortajada@florida-uni.es

Índice

- 1. Despliegue web**
- 2. Aplicaciones web**
- 3. Virtualización: máquinas virtuales (MVs) y contenedores**
- 4. Control de versiones (Git)**
- 5. Documentación**
- 6. Conclusiones**
- 7. Ejemplos prácticos**

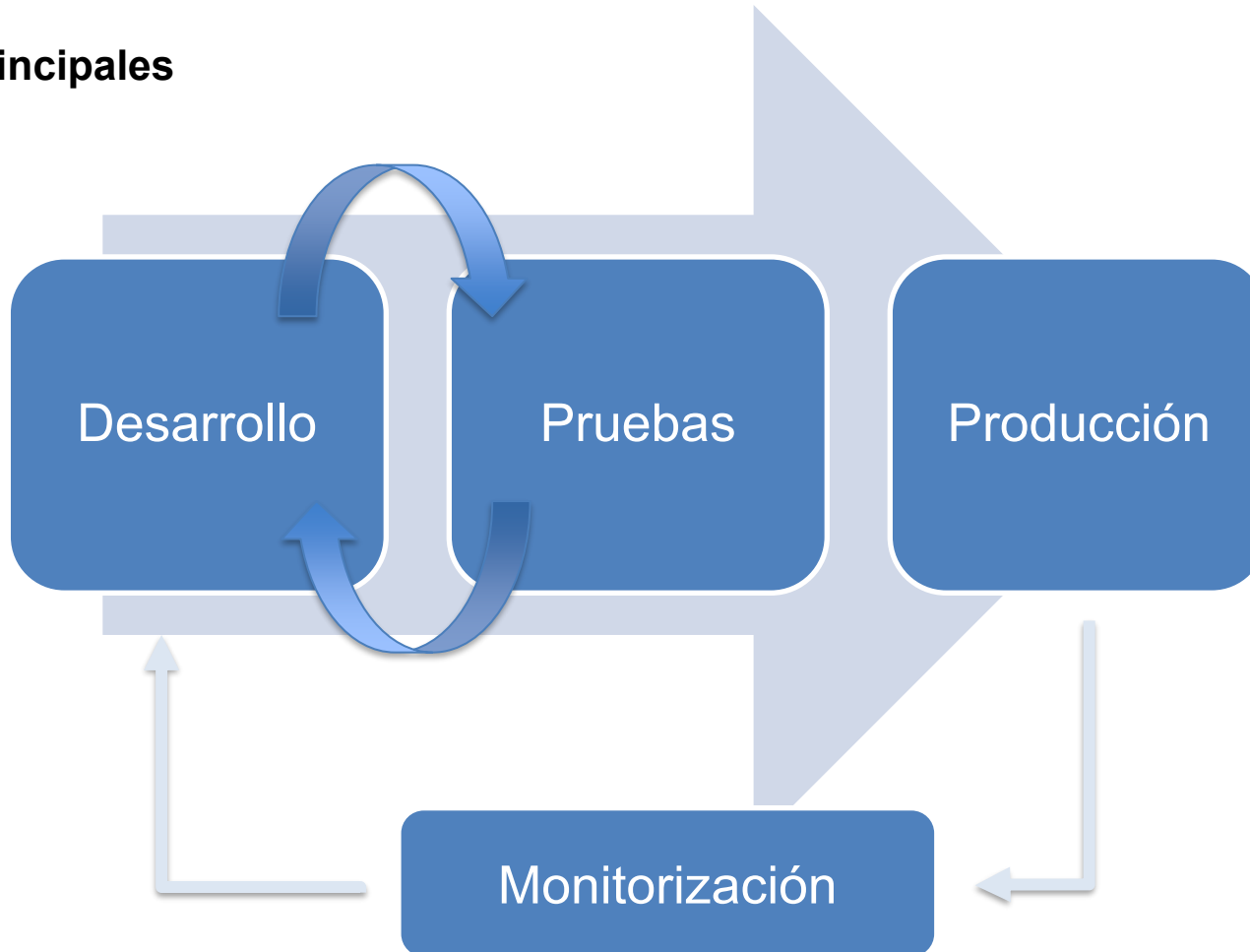
1. Despliegue web

Desplegar: instalar una aplicación en un servidor web con el objetivo de que esté disponible para otros usuarios (incluidos otros desarrolladores)



1. Despliegue web

Fases principales



1. Despliegue web

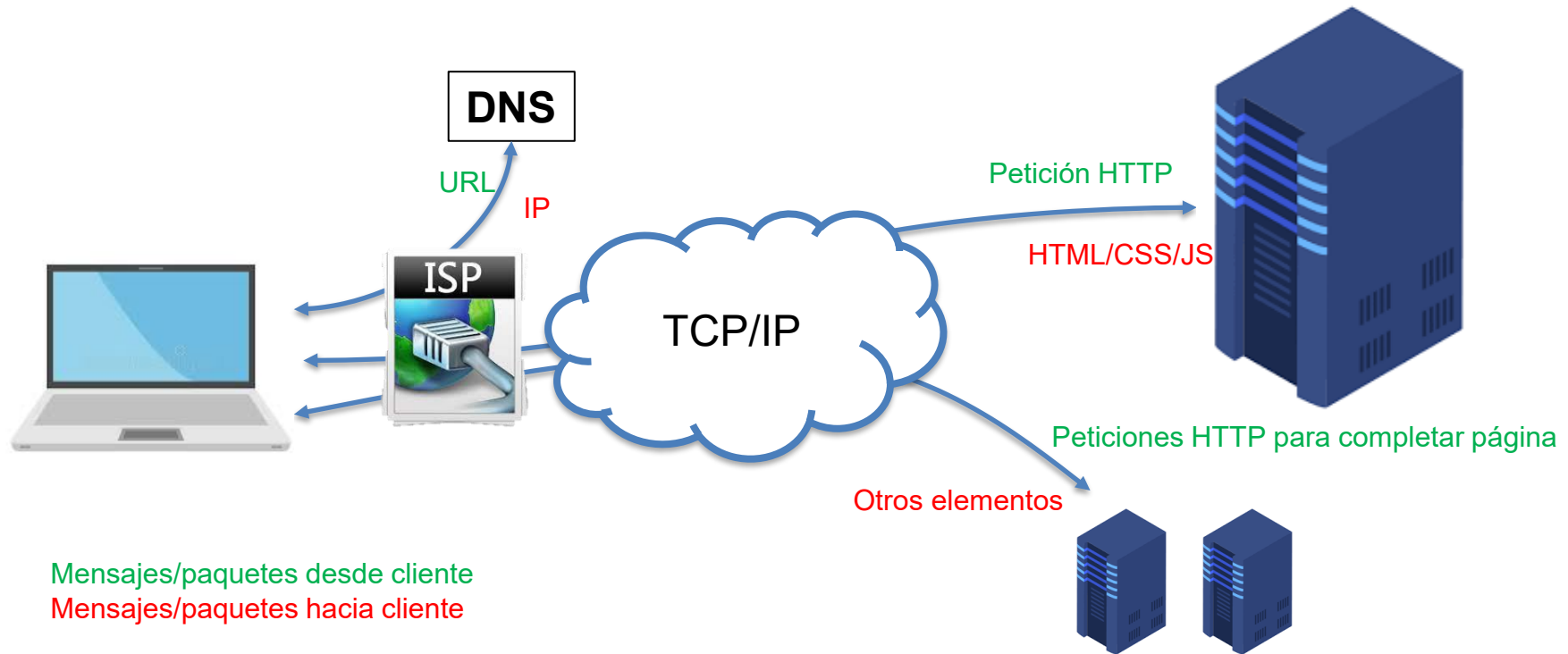
Buenas prácticas

- ✓ Elaborar un plan de despliegue: procedimientos, acciones y horarios.
- ✓ Respetar los entornos de desarrollo locales, colaborativos y de pre-producción.
- ✓ Evaluar las diferencias entre el entorno de desarrollo y el entorno real.
- ✓ Testear la aplicación en cada entorno.
- ✓ Utilizar sistemas de control de versiones (Git).
- ✓ Utilizar ramas para cada funcionalidad o grupo de funcionalidades.
- ✓ Utilizar el desarrollo en local en la medida de lo posible.
- ✓ Implementar mecanismos de monitorización y respuesta rápida.
- ✓ Prepararse para imprevistos (las cosas suelen romperse).



2. Aplicaciones web

Cuando se introduce una dirección en el navegador...



2. Aplicaciones web

Cuando se introduce una dirección en el navegador...

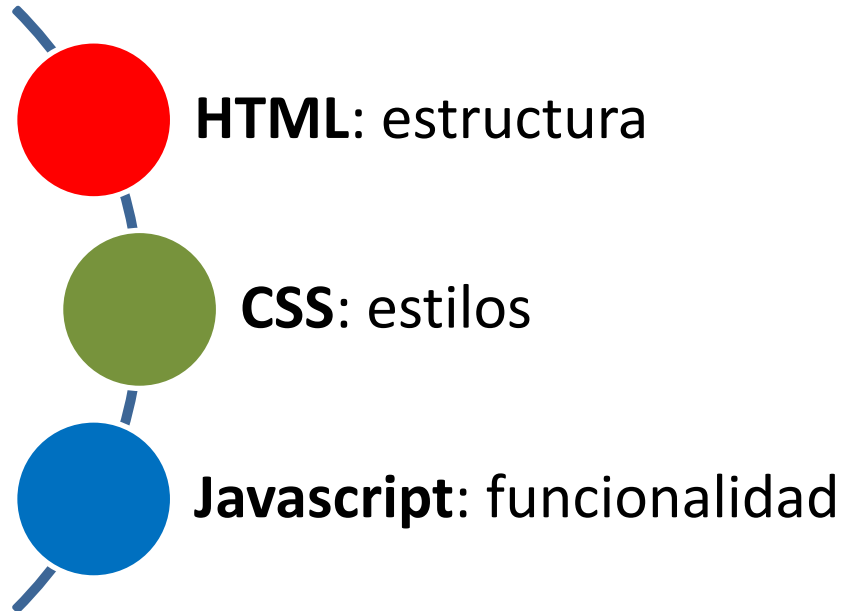
1. Ordenador conectado por módem a un ISP (Telefónica, Vodafone,...).
2. Escribimos una URL (dirección) en el navegador.
3. El ISP recibe la URL y la traduce a una dirección IP (protocolo DNS).
4. El navegador solicita una petición HTTP al servidor que “escucha” en la dirección IP para que le envíe el recurso (por ejemplo, una página web).
5. El servidor acepta (mensaje “200 OK”) la petición y envía el recurso como paquetes de datos (protocolo TCP). Puede dar otros mensajes.
6. El navegador recibe la página y la recorre en busca de elementos que necesite para completarla (por ejemplo, imágenes).
7. El navegador realiza nuevas peticiones al servidor (al mismo o a otros) para obtener cada elemento de la página.
8. El navegador muestra la página completa.

2. Aplicaciones web

Contenido (código) de una página web

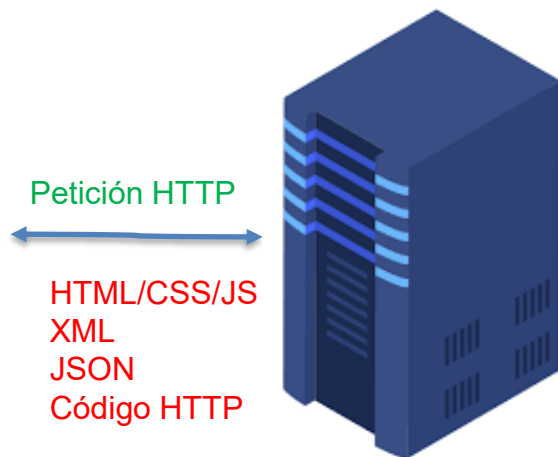


catholic.net



2. Aplicaciones web

La aplicación web desde el lado del servidor



Petición:

- Servidor escucha en puerto (p.ej. 80 -Apache-)
- Comprobaciones de autenticación (si procede)
- Ejecución de scripts de servidor (p.ej. PHP)
- Consulta (si procede) a base de datos (DBMS, p.ej. MySQL)
- Construcción (dinámica) del contenido
- Preparación de contenido HTML, CSS y JS

Respuesta:

- Petición aceptada (código HTTP 2XX, devuelve contenido)
- Petición redirigida (código HTTP 3XX)
- Petición denegada (código HTTP 4XX)
- Error de servidor (código HTTP 5XX)

<https://developer.mozilla.org/es/docs/Web/HTTP/Status>

2. Aplicaciones web

La aplicación web: estructura en capas

Servidor web

- Comunicación TCP (p.ej. puerto 80, 443)
- Respuesta estática (fichero HTML, CSS, imágenes, etc.)
- Ejecución de un programa (script)
- Ejemplos: Apache, Nginx, Tomcat, NodeJS,...

Capa de aplicaciones

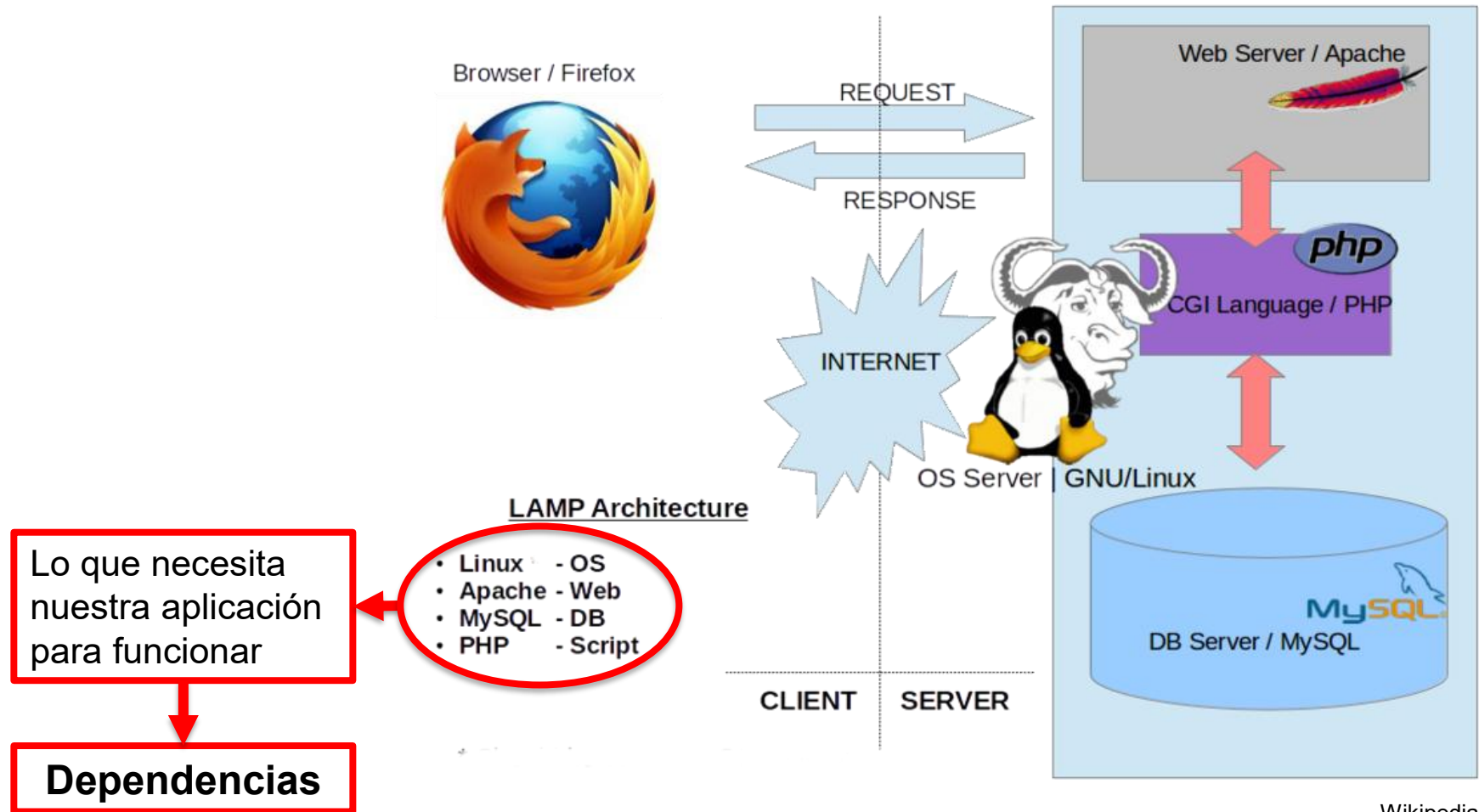
- Web dinámica: ejecución de un programa para generar el contenido solicitado por el navegador web
- Lenguajes compilados: Java, C, C++, C#,...
- Lenguajes interpretados (scripts): PHP, Python, Perl,...

SGBD (DBMS) Sistema gestor de BDD

- Persistencia de la información (más allá de los ficheros)
- Estructuración y escalabilidad
- SGBD relacional (SQL): MySQL, MariaDB, PostgreSQL,...
- SGBD no relacional: MongoDB,...

2. Aplicaciones web

La aplicación web: estructura en capas → Ejemplo: pila LAMP



Wikipedia

DAW - Despliegue de aplicaciones web

1. Introducción

2. Aplicaciones web

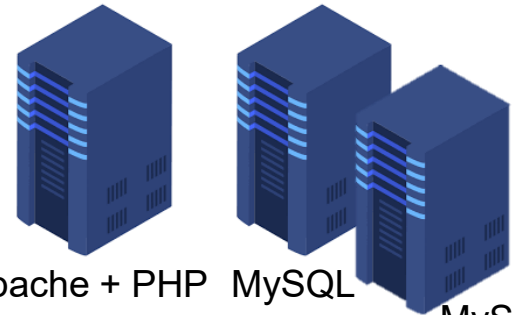
Un servidor o muchos servidores



Apache +
PHP +
MySQL



Apache + PHP MySQL



Apache + PHP MySQL MySQL
(backup)



Apache



Microservicios



MySQL



Balanceador
de carga



Apache 1
+ PHP



Apache 2
+ PHP



MySQL

3. Virtualización



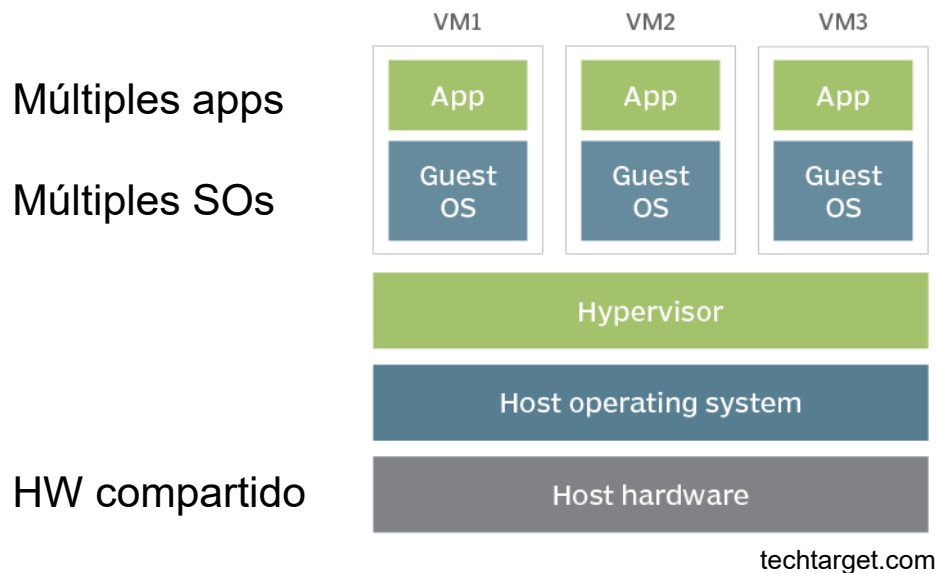
DAW - Despliegue de aplicaciones web

1. Introducción

3. Virtualización

Tecnología para crear una o varias representaciones virtuales de un servidor (**máquina virtual**) dentro de una máquina física.

A nivel de comportamiento, es como tener una réplica de un equipo físico.



ufsexplorer.com

3. Virtualización

Máquinas virtuales

Ventajas:

- ☺ Permiten desplegar aplicaciones web en local o en la nube.
- ☺ Disponer de varios sistemas operativos y versiones (actuales/anteriores).
- ☺ Facilitar el desarrollo y las pruebas de software.
- ☺ Crear copias de seguridad.
- ☺ Fáciles de manejar y mantener.

Desventajas:

- ☹ Consumo de recursos (CPU, disco duro, RAM).
- ☹ Problemas de eficiencia.

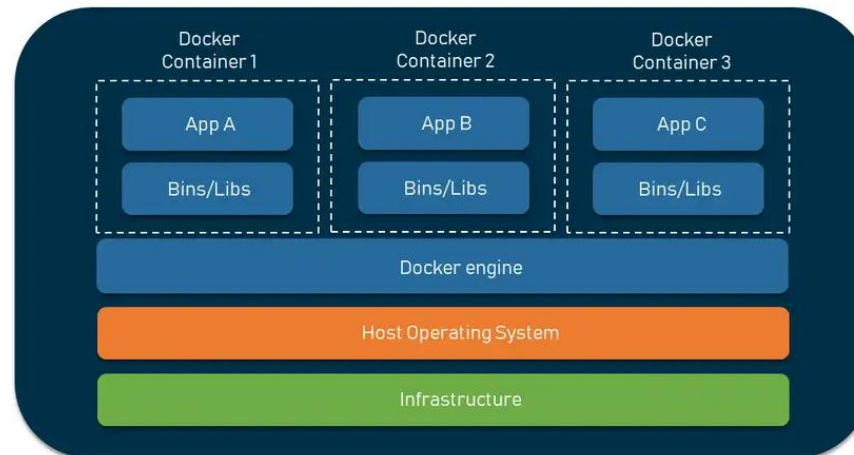


vmware®

3. Virtualización

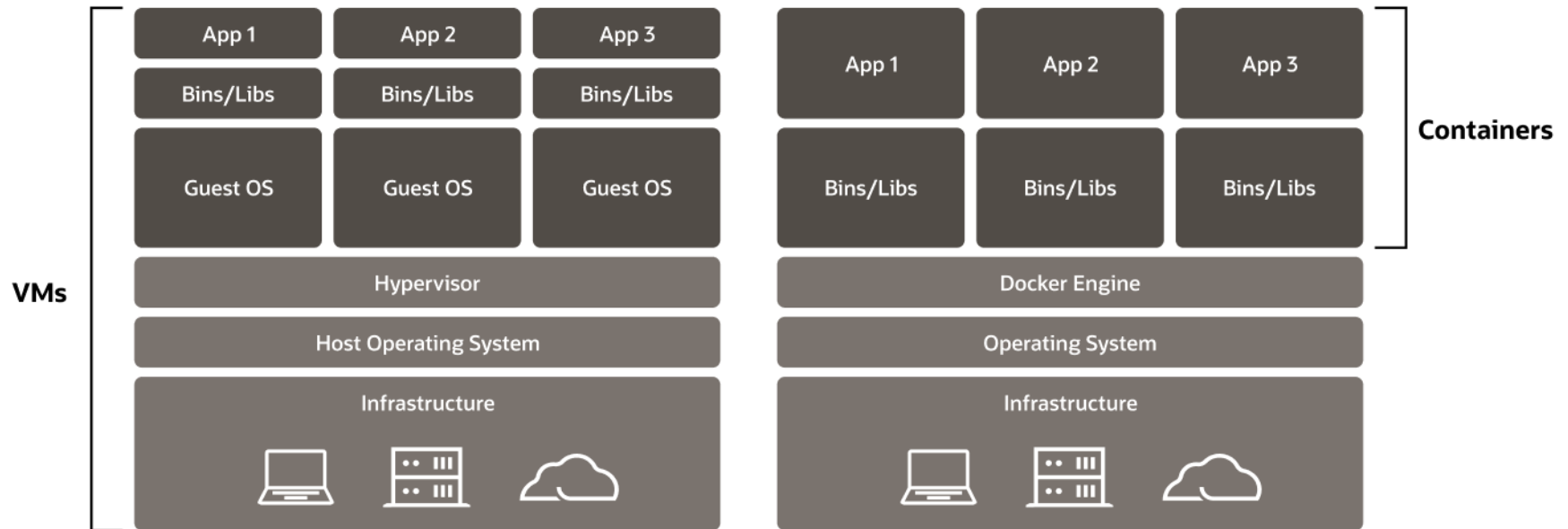
¿Y si solo nos interesa virtualizar el SW, obviando el HW? → **Contenedores**

- ✓ Un contenedor es una unidad de SW que empaqueta el código de una aplicación y todas sus dependencias (**Docker**).
- ✓ Se ejecuta en el sistema operativo huésped (host), donde debe haberse instalado previamente una aplicación específica (**Docker Engine**).
- ✓ No son persistentes, se ejecutan a partir de una **imagen**.



altexsoft.com

3. Virtualización



Virtual Machines

- Each virtual machine (VM) includes the app, the necessary binaries and libraries and an **entire guest operating system**

Containers

- Containers include the app and all of its dependencies, but **share the kernel** with other containers.
- Run as an isolated process in userspace on the host operating system.
- **Not** tied to any specific infrastructure - containers run on any computer, on any infrastructure and in any cloud.

oracle.com

4. Control de versiones (Git)



Git es un sistema de control de versiones distribuido que permite gestionar y hacer un seguimiento de cambios en el código.

- ✓ **Desarrollo colaborativo:** facilita la colaboración de múltiples desarrolladores, que pueden trabajar de manera simultánea sin sobrescribir el trabajo de los demás. Cada desarrollador trabaja en una característica (rama) o revisión y posteriormente se integra de manera ordenada.
- ✓ **Manejo de versiones:** historial de cambios, comparación de versiones, reversión a una versión anterior y mantenimiento de diferentes versiones (desarrollo/pruebas/producción).
- ✓ **Automatización de despliegues:** facilita la integración y entrega continua (CI/CD), haciendo que cada cambio en el código desencadene automáticamente procesos de prueba, integración y despliegue en producción.

¿**Git o GitHub?** Git es la tecnología de control de versiones, mientras que GitHub es básicamente un sistema de almacenamiento en la nube para gestionar repositorios Git. Otro servicio similar a GitHub es **GitLab**.

4. Control de versiones (Git)



¿Cómo funciona Git? Hay cuatro áreas diferenciadas...

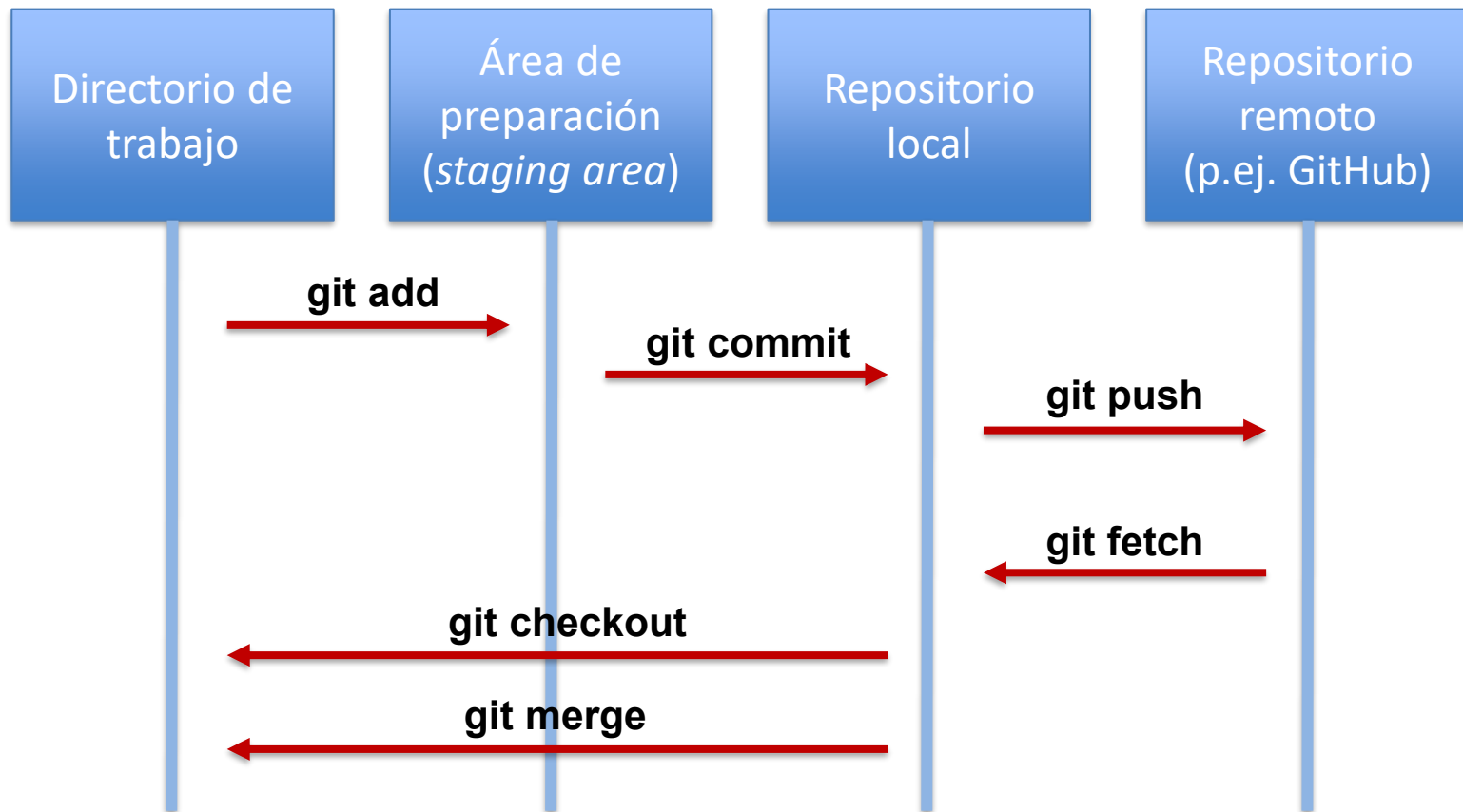


1. Trabajamos en nuestro código en el **directorio de trabajo** del IDE (Visual Studio Code, Sublime, Eclipse, PHPStorm, etc.).
2. Añadimos a la **staging area** los ficheros cuyas versiones queramos controlar (no tienen por qué ser todos los del directorio).
3. Proporcionamos una descripción de los cambios que se han hecho en esos ficheros (p.ej. “fichero nuevo”, “cambios en fichero X”,...) y se mandan (*commit*) al **repositorio local**.
4. El repositorio local y el **repositorio remoto** se sincronizan para que esté siempre disponible el control de versiones (última versión y versiones anteriores) para nosotros y nuestros colaboradores.

4. Control de versiones (Git)



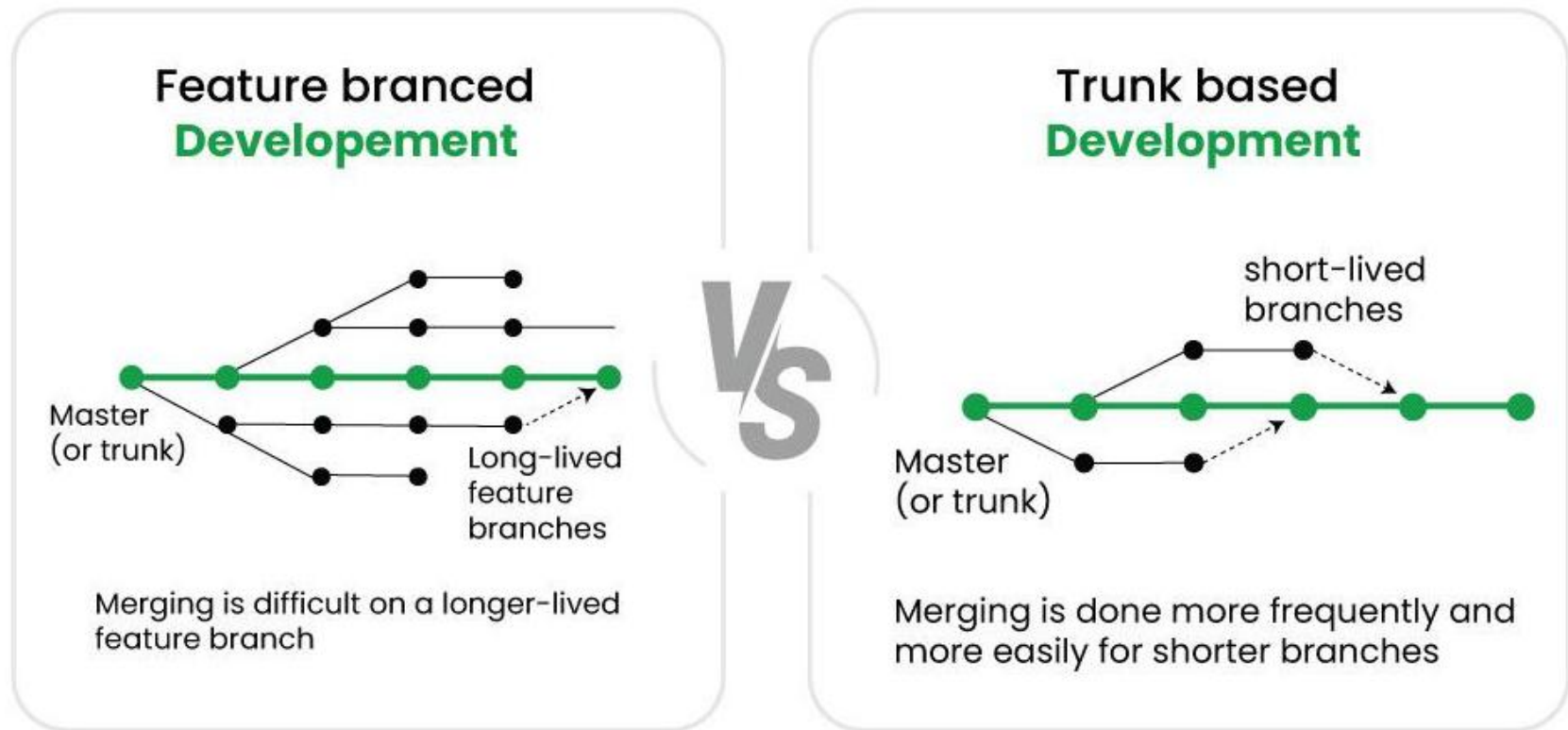
El paso de código de un área a otra se realiza con comandos (flujo de trabajo):



4. Control de versiones (Git)



Estrategias de trabajo en ramas (Git Flow vs Trunk Based)



<https://www.geeksforgeeks.org/trunk-based-development-in-software-development/>

4. Control de versiones (Git)



Documentación y tutoriales

GitHub Cheat Sheet: https://training.github.com/downloads/es_ES/github-git-cheat-sheet/

Visual Git Cheat Sheet: <https://ndpsoftware.com/git-cheatsheet.html#loc=index>;

Documentación oficial: <https://git-scm.com/doc>

Material de Florida Oberta.

5. Documentación

Las herramientas de documentación son necesarias para que los desarrolladores puedan entender, utilizar y mantener el código desarrollado (propio o de otros). Todos los lenguajes de programación tienen herramientas que permiten generar documentación (habitualmente como páginas webs) a partir de anotaciones realizadas en el propio código.

PHP → [phpDocumentor](#)

JavaScript → [JSDoc](#)

Java → [Javadoc](#)

Para documentar APIs se puede utilizar [OpenAPI/Swagger](#), una especificación de formato para describir los *endpoints*, métodos, parámetros, respuestas y cualquier otra información relevante de una API. Se puede integrar en cualquier lenguaje de programación utilizando las bibliotecas correspondientes.

6. Conclusiones

- ✓ Desplegar es básicamente alojar una aplicación en un servidor web para que los clientes la puedan utilizar.
- ✓ Una aplicación web es accesible a través de un navegador web (cliente), al que un servidor proporciona vistas (páginas web con HTML+CSS+JS) de manera dinámica en función de las solicitudes que envía el cliente.
- ✓ El servidor crea las vistas ejecutando cierto código (interpretado -PHP, Python, Perl- o compilado -Java-).
- ✓ Desarrollar una aplicación web requiere componentes y herramientas para su correcto funcionamiento, uso y mantenimiento (servidor, librerías, *frameworks*, bases de datos, dependencias, control de versiones, documentación, etc.).
- ✓ Para facilitar el despliegue de la aplicación se emplean técnicas de virtualización, que reproducen el entorno de desarrollo y pruebas en máquinas dedicadas a producción (máquinas virtuales o contenedores).

7. Ejemplos prácticos

1. Git (y GitHub)
2. Servidor web en local
3. Servidor web mediante virtualización de Ubuntu en VirtualBox
4. Servidor web mediante Docker

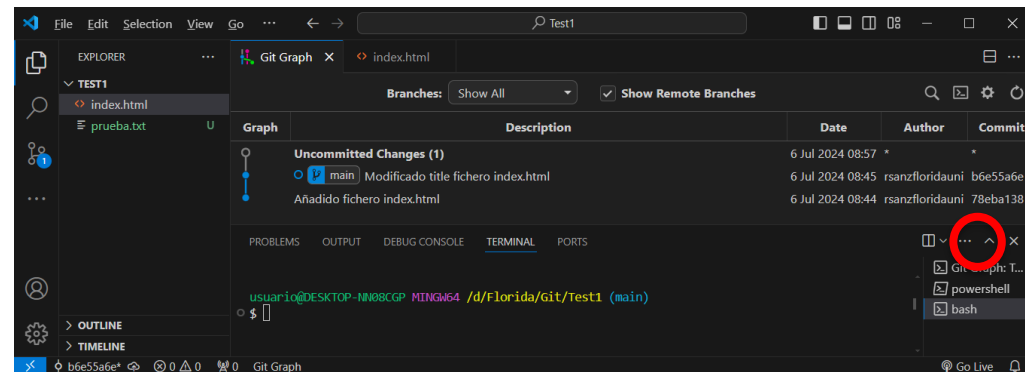
7. Ejemplos prácticos (1 - Git)

Configuración del entorno de trabajo

Descarga e instala **Git** desde <https://git-scm.com/downloads>

Descarga e instala **Visual Studio Code** desde <https://code.visualstudio.com/> y la extensión **Git Graph**

En VSCode abre un terminal (la selección por defecto es PoweShell). Haz clic en el desplegable del terminal (*Launch profile...*) y elige **Git Bash**



Desde el terminal, configura un usuario y un correo:

```
git config --global user.name tuUsuario
```

```
git config --global user.email tuCorreo
```

Por seguridad, es posible que Git no reconozca automáticamente la propiedad de los directorios de trabajo, por ello es conveniente que al crear un directorio ejecutes:

```
git config --global --add safe.directory tuDirectorio
```

NOTA: La extensión Git Graph aparece en la barra de estado de VSCode, pero solo estará disponible cuando estés en una carpeta que tenga un repositorio Git. En el resto de carpetas no aparecerá.

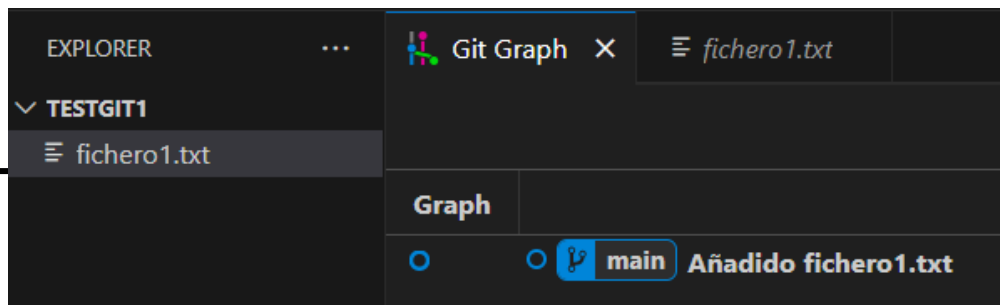
7. Ejemplos prácticos (1 - Git)

Primeros pasos (I)

1. Desde Git Bash, crea un directorio vacío: `mkdir TestGit1`
2. Accede al directorio: `cd TestGit1`
3. Inicializa el repositorio (se crea la subcarpeta oculta “.git”): `git init`
4. Crea un fichero en TestGit1: `touch fichero1.txt`
5. Introduce algo de contenido en el fichero y guarda cambios (puedes hacerlo desde el terminal con la instrucción `nano fichero1.txt` o desde VSCode).

La “U” que aparece junto al fichero en el explorador indica que está en estado *untracked* para Git.

6. Añade el fichero al índice de control de versiones (*staged*): `git add fichero1.txt`
La “A” que aparece ahora indica que el fichero está añadido al índice.
7. Realiza *commit* para enviarlo al repositorio: `git commit -m “Añadido fichero1.txt”`
8. Abre la extensión Git Graph y observa que se ha creado la rama *main* (según tu instalación, también puede llamarse *master*) con el *commit* que acabas de realizar.



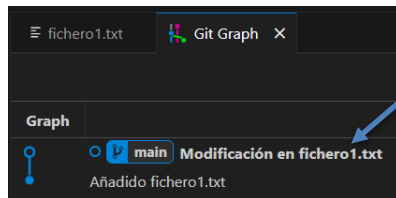
NOTA: con el comando `git status` puedes ver en cualquier momento el estado de los ficheros (*untracked*, *staged* -pendientes de *commit*-, en repositorio). El comando `git` muestra todos los comandos disponibles.

7. Ejemplos prácticos (1 - Git)

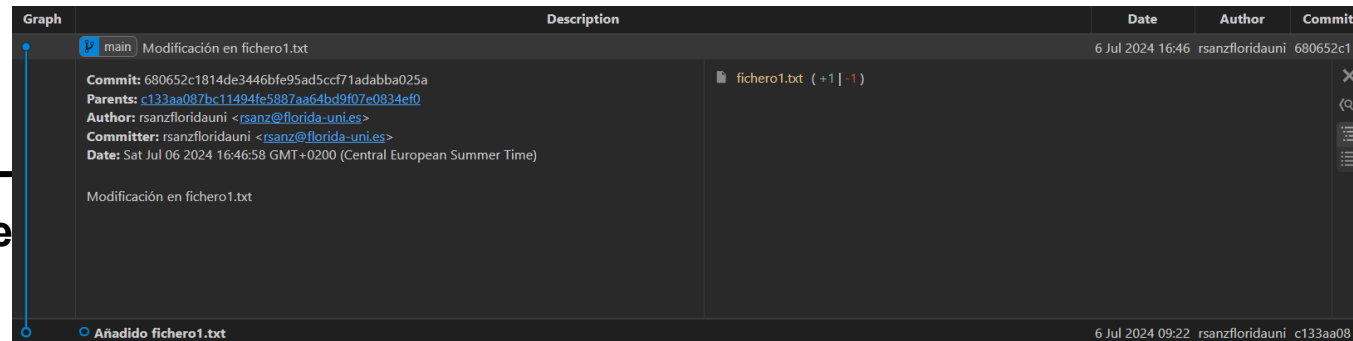
Primeros pasos (II)

1. Modifica el fichero1.txt (con `nano fichero1.txt` o desde VSCode).
2. Observa que el fichero aparece con una “M” al lado y si introduces `git status` en el terminal, aparece como *modified*.
3. Pasa el fichero de nuevo a *staged*: `git add fichero1.txt`
 - Para quitar el fichero de *staged*: `git restore --staged fichero1.txt`
4. Y “comitea”: `git commit -m “Modificación en fichero1.txt”`
 - Para revertir el *commit*: `git revert <id_commit>`

En Git Graph se puede ver la foto (*snapshot*) del repositorio, donde *main* incluye los cambios realizados.



Haciendo clic en cada *commit* se muestra el detalle:



DAW - Despliegue de

1. Introducción

7. Ejemplos prácticos (1 - Git)

Primeros pasos (III)

Comparar cambios

`git diff`: muestra las diferencias entre el área de trabajo y el área de preparación (staged)

`git diff --staged`: muestra las diferencias entre el área de preparación y el último commit

NOTA: al ejecutar esta instrucción en el terminal, se abre un paginador que va mostrando la información. Para salir del paginador y volver al terminal, hay que pulsar la tecla “q”.

Eliminar archivos

`rm <nombre_archivo>`

`git rm <nombre_archivo>`

`git commit -m “Eliminado el archivo <nombre_archivo>”`

7. Ejemplos prácticos (1 - Git)

Primeros pasos (IV)

Abre paginador ("q" para salir)

Para volver a una versión previa, el comando `git log` muestra el historial de versiones.

```
$ git log
commit 680652c1814de3446bfe95ad5ccf71adabba025a (HEAD -> main)
Author: rsanzfloridauni <rsanz@florida-uni.es>
Date: Sat Jul 6 16:46:58 2024 +0200

    Modificación en fichero1.txt

commit c133aa087bc11494fe5887aa64bd9f07e0834ef0
Author: rsanzfloridauni <rsanz@florida-uni.es>
Date: Sat Jul 6 09:22:18 2024 +0200

    Añadido fichero1.txt
```

Versión más reciente: *HEAD* apunta a la rama *main*, identificada por el id 6806...

Versión anterior a la que queremos volver, identificada por el id c133aa...

Hay que realizar un *checkout* indicando el id de *commit* deseado (no es necesario que esté completo, suelen utilizarse los 7 primeros caracteres): `git checkout c133aa0`

Ejecuta otra vez `git log`. Ahora *HEAD* apunta a este *commit*, no a la rama *main*. Si observas el contenido de *fichero1.txt*, ya no aparece la modificación que habías hecho.

Para cambiar entre versiones sería con el comando `git checkout <id_commit>`, o también puedes utilizar el botón derecho en Git Graph.

De momento, vuelve al *commit* asociado a la rama *main*: `git checkout main`

Aviso *HEAD detached*

HEAD es un puntero especial que señala a la referencia del *commit* actual en el que te encuentras trabajando. Si está desanclado (*detached*) significa que no estás apuntando a una rama y cualquier *commit* adicional no se asociará a ninguna rama si no lo haces manualmente.

7. Ejemplos prácticos (1 - Git)

Primeros pasos (V) - .gitignore

En la mayoría de los casos no necesitarás que todos los ficheros de tu proyecto estén en el repositorio, sobre todo si solo los necesitas para desarrollo o si contienen información sensible (p.ej. contraseñas).

Git explora todo el directorio raíz del proyecto y puede ser molesto que siempre te esté indicando esos ficheros como *untracked*. Para que Git no los tenga en cuenta, basta crear un fichero en la raíz del proyecto que se llame *.gitignore*, cuyo contenido sea la lista de ficheros y carpetas que no quieres incluir en el repositorio.

7. Ejemplos prácticos (1 - Git)

Primeros pasos (VI) - Ramas (*branch*)

Las ramas son líneas independientes de desarrollo para trabajar en diferentes características, corrección de errores, pruebas, etc. No interfieren con la rama principal (*main* o *master*) y lo habitual es que, una vez se ha realizado el trabajo necesario, se fusionen (*merge*) con la rama principal. Se suelen definir tres tipos de rama:

- *Feature branch*: trabajar con una nueva característica o corregir errores.
- *Release branch*: preparar una nueva versión del proyecto para producción.
- *Hotfix branch*: corregir rápidamente un error crítico de producción.

Flujo de trabajo habitual:

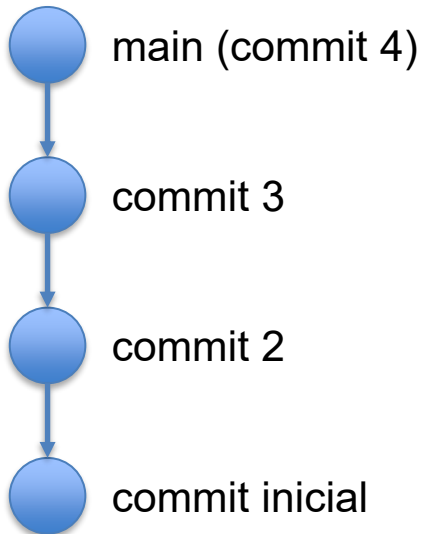
1. Crear una nueva rama: `git branch nuevaRama`
2. Cambiar de rama: `git checkout nuevaRama` (*HEAD* apunta a nuevaRama)
3. Trabajo en la rama: *adds*, *commits*, etc.
4. Fusionar la rama con otra rama (p.ej. *main*):
Cambiar a la rama destino: `git checkout main`
Indicar la rama a fusionar: `git merge nuevaRama`

7. Ejemplos prácticos (1 - Git)

Primeros pasos (VII) - Trabajo en remoto

Local

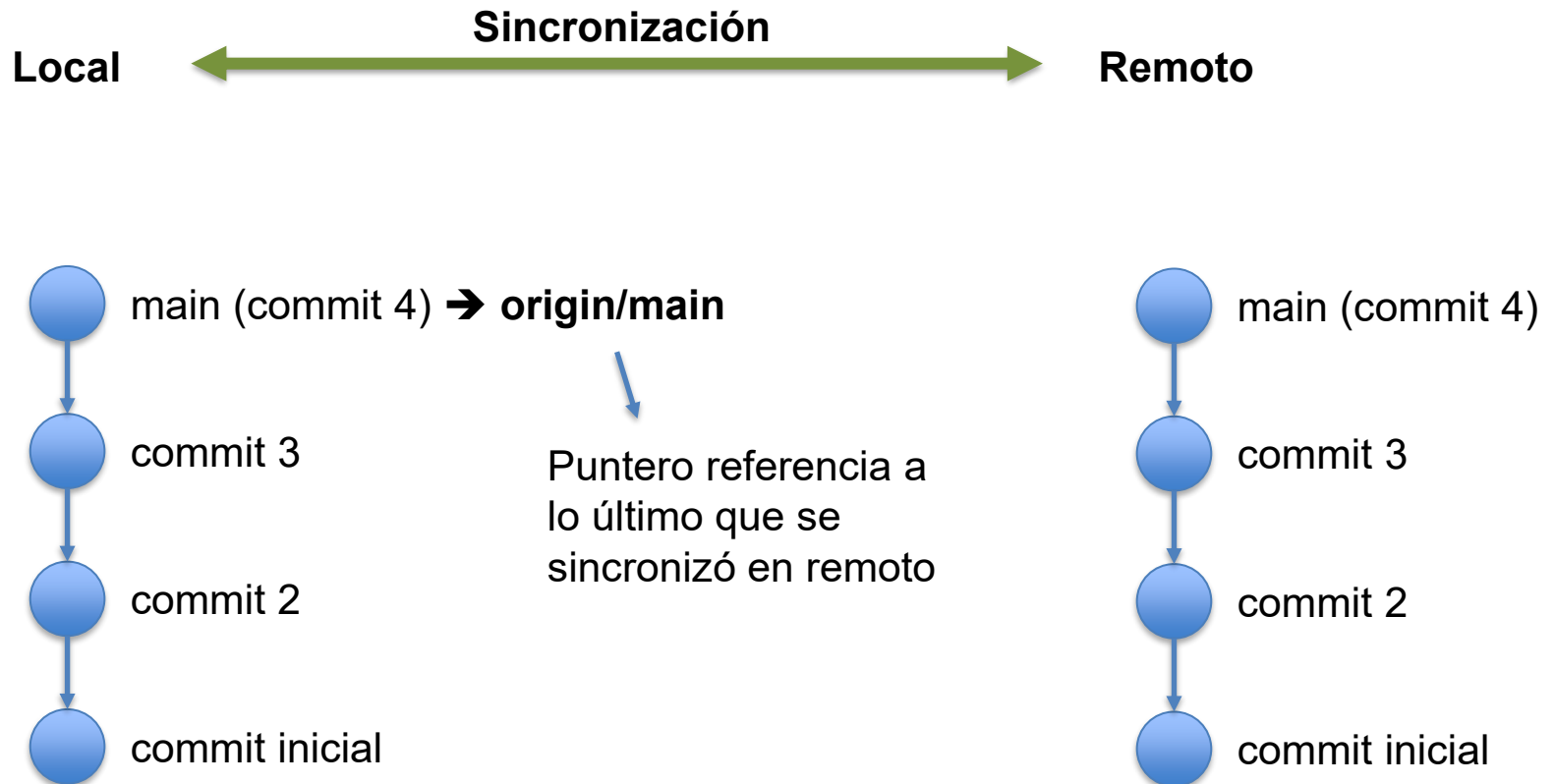
Remoto



Situación de partida: el repositorio local no está sincronizado con el repositorio remoto, que en este caso además está vacío. Hay una rama *main* de la que cuelgan 3 *commits*.

7. Ejemplos prácticos (1 - Git)

Primeros pasos (VII) - Trabajo en remoto

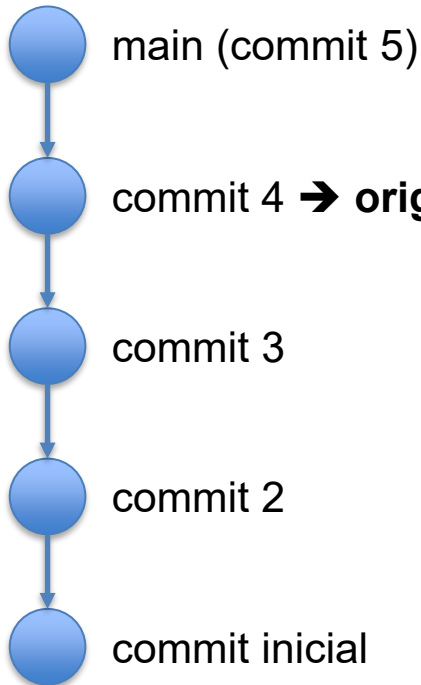


Local y remoto se sincronizan. El repositorio remoto se “trae” la rama *main* y todos los *commits* que cuelgan de ella. En local se crea un puntero nuevo llamado *origin* que apunta a lo último que hay en remoto. Al finalizar la sincronización, local y remoto deben tener lo mismo.

7. Ejemplos prácticos (1 - Git)

Primeros pasos (VII) - Trabajo en remoto

Local

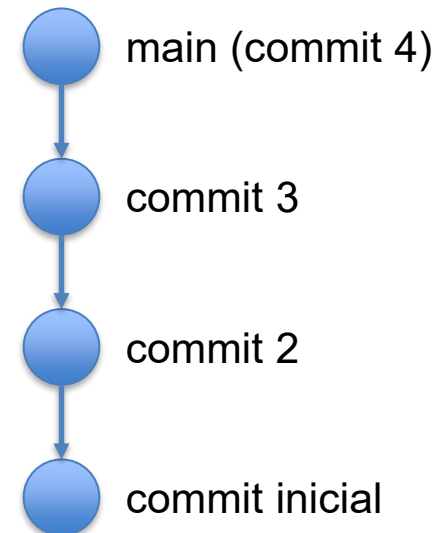


commit 4 → **origin/main**



Puntero referencia a
lo último que se
sincronizó en remoto

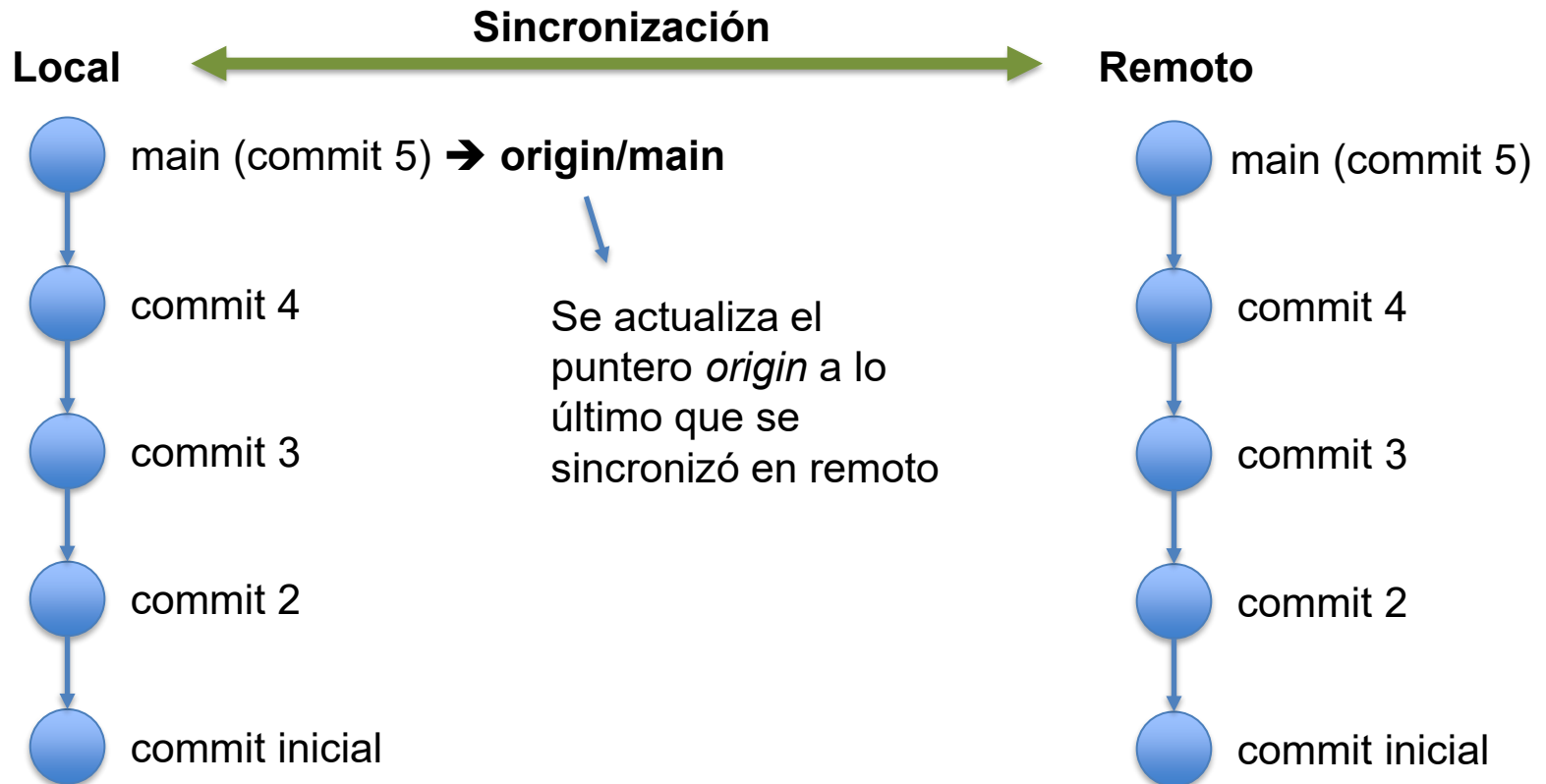
Remoto



Continuamos trabajando en local y realizamos otro *commit* (o varios), de forma que la rama *main* avanza. Ahora ya no coinciden *main* y la referencia *origin* al remoto.

7. Ejemplos prácticos (1 - Git)

Primeros pasos (VII) - Trabajo en remoto

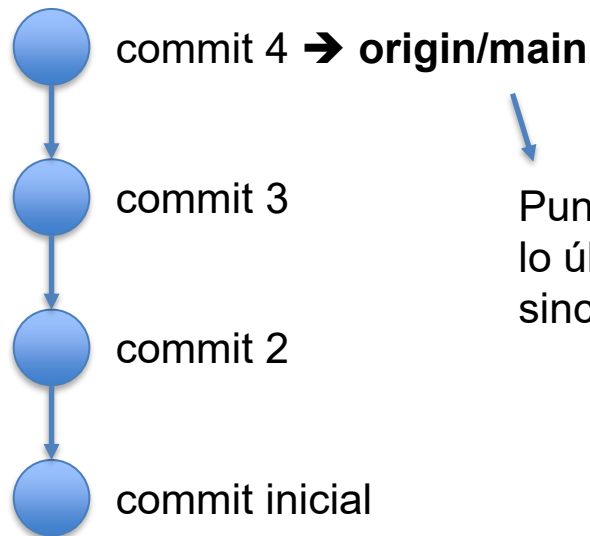


Nueva sincronización. Remoto se “trae” la referencia a la rama *main* (y si hubiera otros *commits* que no tuviera también se los traería) y el puntero *origin* se actualiza.

7. Ejemplos prácticos (1 - Git)

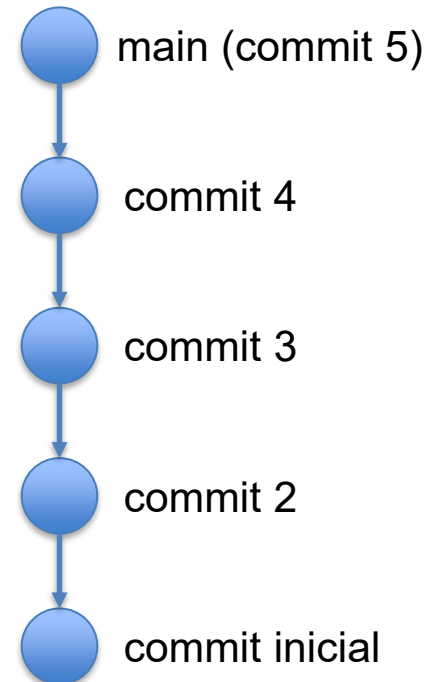
Primeros pasos (VII) - Trabajo en remoto

Local



Puntero referencia a
lo último que se
sincronizó en remoto

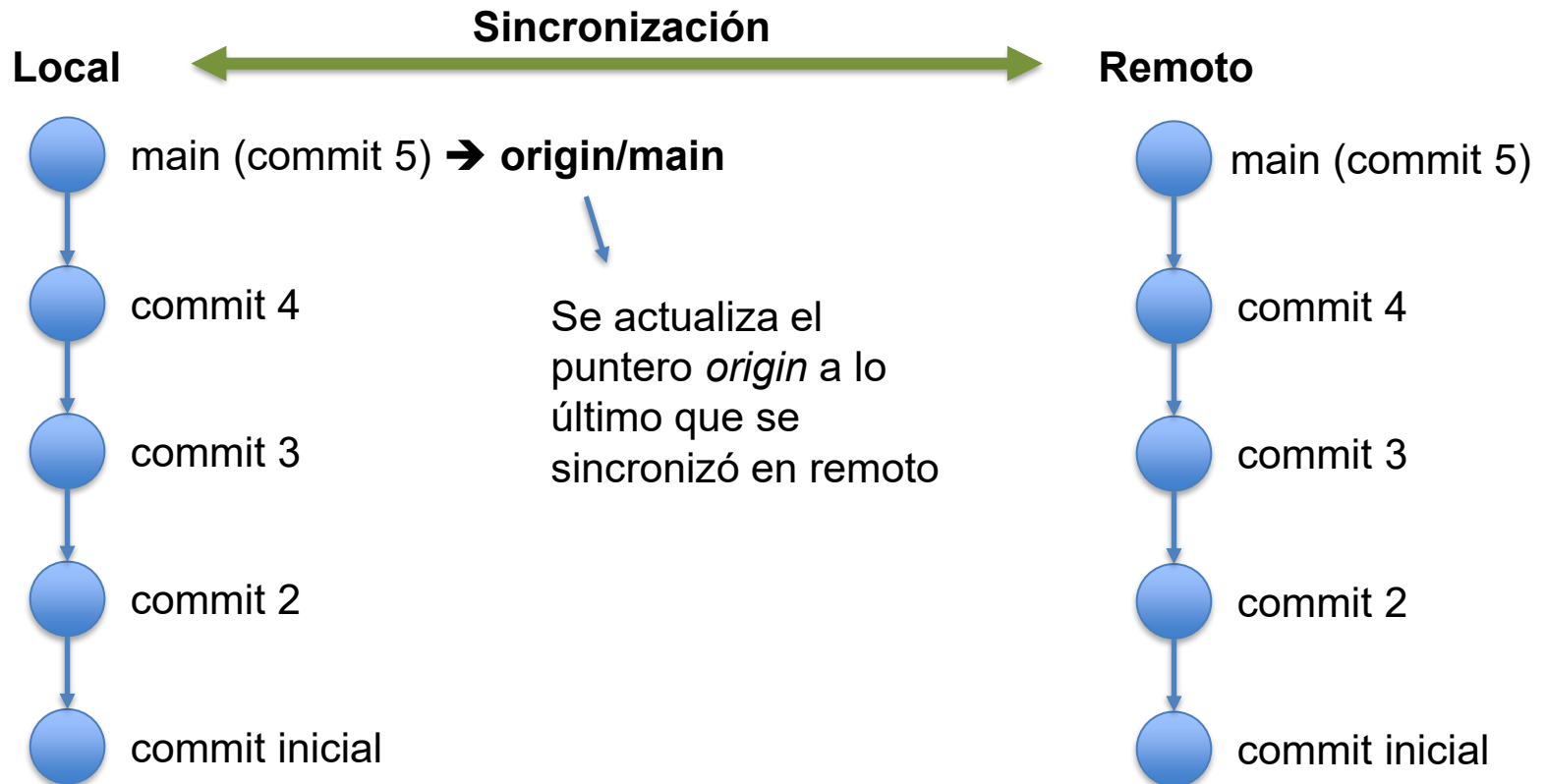
Remoto



Otra situación. Un compañero ha sincronizado su local con el remoto que compartimos. En este caso hay una versión más reciente en remoto que la que tenemos en local.

7. Ejemplos prácticos (1 - Git)

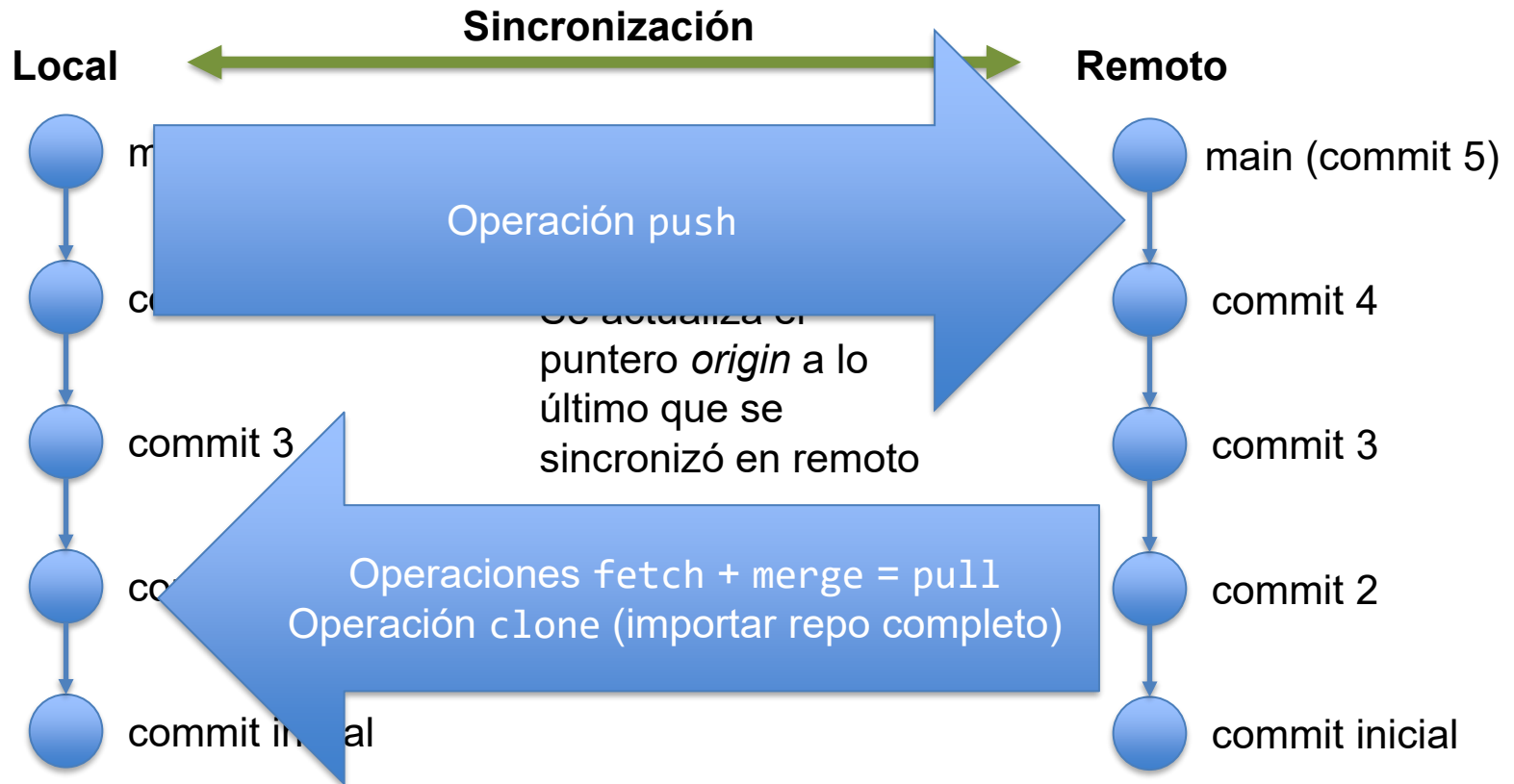
Primeros pasos (VII) - Trabajo en remoto



Al sincronizarse con remoto sucede lo mismo. Se comprobará la referencia más reciente correspondiente a la rama *main* y quien no la tenga (en este caso el local) se traerá la referencia y todos los *commits* que le falten. Finalmente actualizaría *origin*.

7. Ejemplos prácticos (1 - Git)

Primeros pasos (VII) - Trabajo en remoto



Al sincronizarse con remoto sucede lo mismo. Se comprobará la referencia más reciente correspondiente a la rama *main* y quien no la tenga (en este caso el local) se traerá la referencia y todos los *commits* que le falten. Finalmente actualizaría *origin*.

7. Ejemplos prácticos (1 - Git)

Primeros pasos (VII) - Trabajo en remoto

Sincronizar repositorio remoto en VSCode:

1. Crea en GitHub un repositorio privado vacío (sin README)
2. En el terminal de Git Bash añade el repositorio remoto que has creado:
`git remote add origin https://github.com/usuario/repositorio.git`
3. Asegúrate que no hay cambios pendientes: `git status`
4. Si los hubiera, utiliza las instrucciones *add* y *commit*
5. Haz el *push* a remoto: `git push -u origin main` (o master, según tu configuración)
6. Accede a tu repositorio de GitHub, donde debes ver ya el fichero1.txt
7. Si realizas cambios en el repositorio local deberás hacer un *push* para que se vean reflejados en remoto. En GitHub tendrás la información de todos los *commits*.

NOTA: la primera vez que realices un push desde VSCode pedirá autenticación. Puedes autenticarte utilizando el PAT (*Personal Access Token*) de GitHub. Si no tienes uno, puedes consultar cómo generarlo en <https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/managing-your-personal-access-tokens#creating-a-personal-access-token-classic>

7. Ejemplos prácticos (1 - Git)

Primeros pasos (VIII) - Trabajo en remoto con conflictos

Crea en GitHub (en la web directamente) un fichero README y realiza un *commit* para que se guarde en el repositorio (remoto en este caso).

Ahora modifica desde VSCode (local) el contenido de fichero1.txt, “omitea” los cambios y prueba hacer *push* a remoto.

```
$ git push -u origin main
To https://github.com/rsanzfloridauni/despliegueTest1.git
! [rejected]        main -> main (fetch first)
error: failed to push some refs to 'https://github.com/rsanzfloridauni/despliegueTest1.git'
hint: Updates were rejected because the remote contains work that you do not
hint: have locally. This is usually caused by another repository pushing to
hint: the same ref. If you want to integrate the remote changes, use
hint: 'git pull' before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

Pide realizar un *pull* para que los cambios hechos en remoto (fichero README) queden reflejados en local antes de mandar algo nuevo al remoto.

Hacemos `git pull origin main` y nos aparece un mensaje (MERGE_MSG) para que proporcionemos un *commit*. Podemos dejarlo así o añadir algo, guardar y cerrar. Repetir la instrucción `git push origin main` (ahora ya debería funcionar).

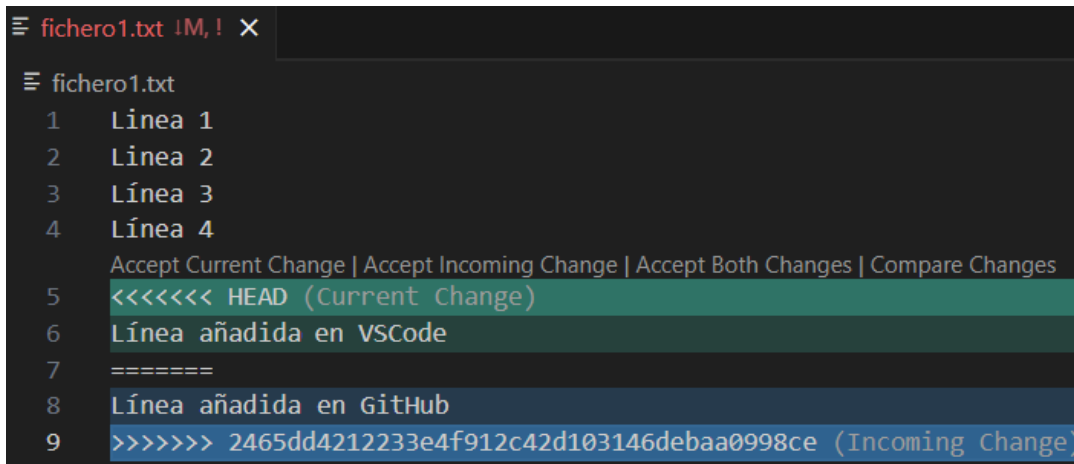
7. Ejemplos prácticos (1 - Git)

Primeros pasos (VIII) - Trabajo en remoto con conflictos

Ahora edita `fichero1.txt` en GitHub y también en VSCode, simulando dos desarrolladores que están trabajando a la vez sobre el mismo fichero. Recuerda “comitear” los cambios para que queden reflejados en cada repositorio (remoto y local).

De nuevo, al tratar de hacer *push* desde local a remoto aparece el mismo error. Hay que realizar un *pull* porque no hay lo mismo en local que en remoto.

A diferencia de antes, ahora no aparece un mensaje estándar para confirmar el *merge*, sino que nos muestra el fichero donde se está dando el conflicto.



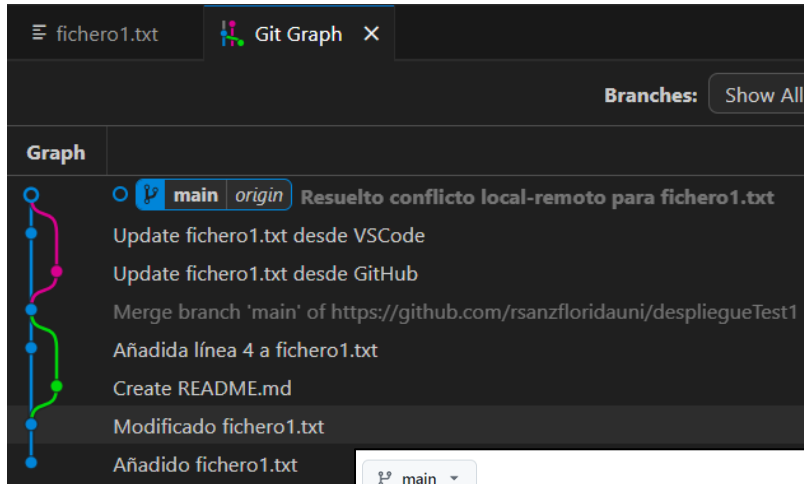
```
fichero1.txt !M,! X
fichero1.txt
1 Línea 1
2 Línea 2
3 Línea 3
4 Línea 4
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
5 <<<<<< HEAD (Current Change)
6 Línea añadida en VSCode
7 =====
8 Línea añadida en GitHub
9 >>>>>> 2465dd4212233e4f912c42d103146debaa0998ce (Incoming Change)
```

VSCode da la opción de quedarse con lo que hay en local (*Current change*), en remoto (*Incoming Change*) o ambos (*Both Changes*).

Acceptaremos ambos en este caso y repetiremos los pasos de *add*, *commit* y *push*.

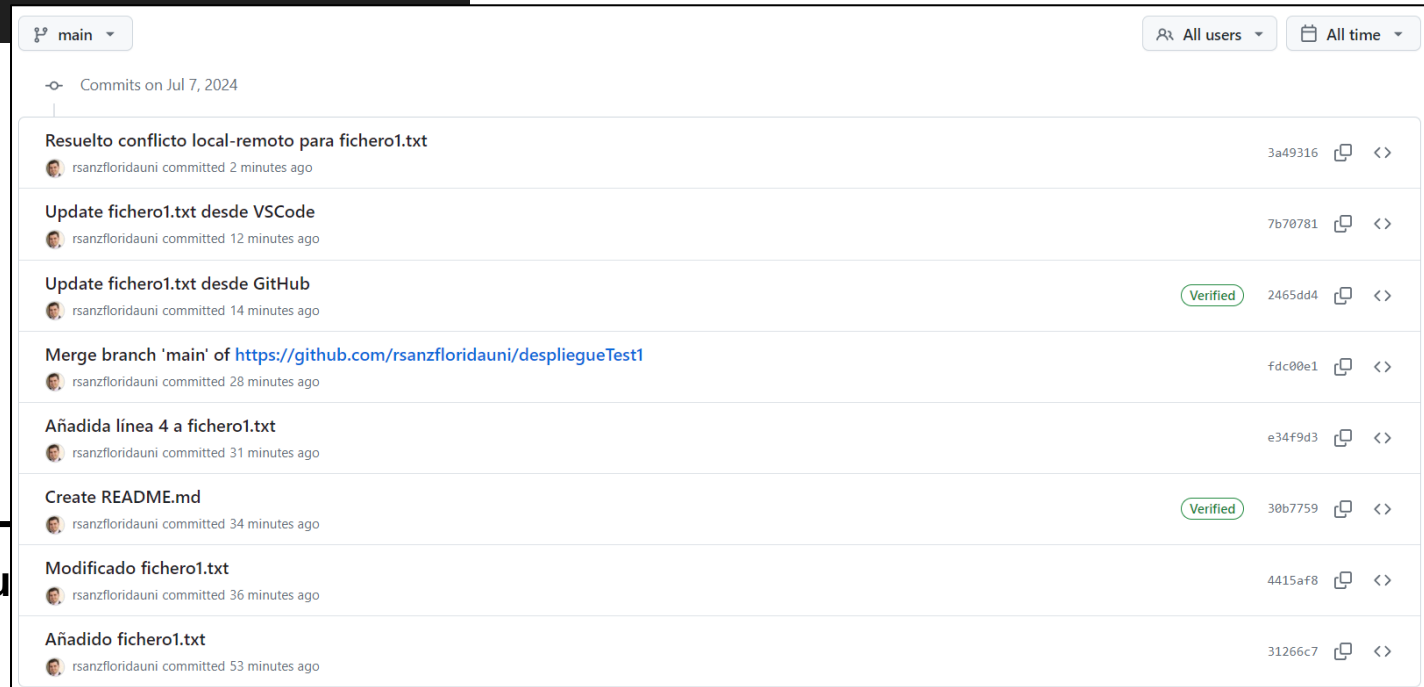
7. Ejemplos prácticos (1 - Git)

Primeros pasos (VIII) - Trabajo en remoto con conflictos



Local

Remoto



DAW - Despliegue

1. Introducción

7. Ejemplos prácticos (2 - Servidor local)

Descargar e instalar XAMPP: <https://www.apachefriends.org/es/index.html>

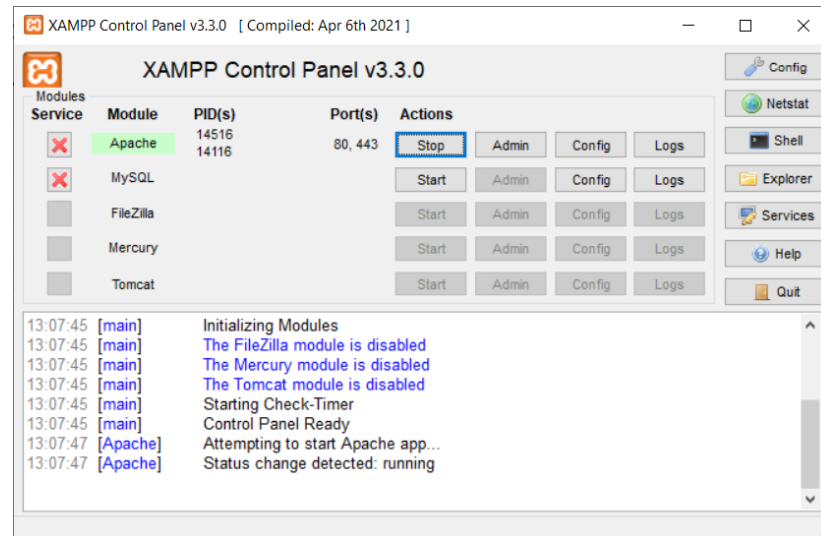
Crear una página web `index.html` básica con cualquier editor de texto:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Mi primera web</title>
  </head>
  <body>
    <p>Hola mundo</p>
  </body>
</html>
```

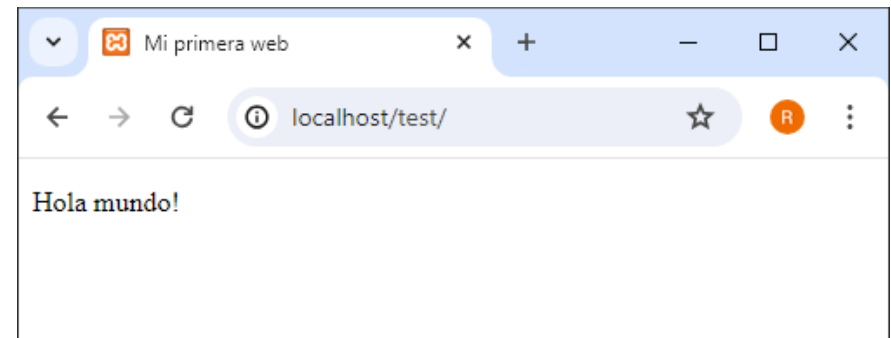
Acceder a la carpeta **htdocs** del directorio donde se ha instalado XAMPP, crear una carpeta (p.ej. **test**) y guardar ahí el fichero **index.html**.

7. Ejemplos prácticos (2 - Servidor local)

Asegurarse que el servidor Apache está activo.



Acceder a la URL: <http://localhost/test>



7. Ejemplos prácticos (3 - Máquina virtual)

Descargar e instalar VirtualBox: <https://www.virtualbox.org/>

Descargar imagen (ISO) de Ubuntu: <https://releases.ubuntu.com/>

➔ Versión 24.04 (Noble Numbat): 6 GB

➔ Versión 18.04 (Bionic Beaver): 2.3 GB ➔ **SUFICIENTE** ⬅

En VirtualBox, crear una máquina virtual nueva a partir de la imagen ISO descargada.

- Es importante marcar el checkbox “Skip unattended installation” (“Omitir instalación desatendida”) para que el terminal de Ubuntu funcione correctamente.
- Seleccionar la opción de que no se actualice el software automáticamente.
- Puedes reservar los recursos (RAM, procesador y disco duro) que se indican por defecto (siempre puedes modificarlo después o instalar otra máquina virtual).
- Arrancar la máquina virtual que se acaba de crear y seguir los pasos de la instalación.

Cuando se inicie Ubuntu, preguntará si quieres actualizar la versión: indica que no.

Finalmente, abrir un terminal.

7. Ejemplos prácticos (3 - Máquina virtual)

1. Actualizar la lista de paquetes: `sudo apt update`
2. Instalar Apache: `sudo apt install apache2`
3. Arrancar Apache: `sudo systemctl start apache2`
4. Activar Apache: `sudo systemctl enable apache2`
5. (Opcional) Instalar MySQL (MariaDB): `sudo apt install mariadb-server mariadb-client -y`
6. (Opcional) Instalar PHP: `sudo apt install php libapache2-mod-php php-mysql -y`
7. (Opcional) Instalar phpMyAdmin: `sudo apt install phpmyadmin`
8. (Opcional) Configurar Apache para utilizar phpMyAdmin: `sudo ln -s /etc/phpmyadmin/apache.conf /etc/apache2/sites-enabled/phpmyadmin.conf`
9. (Opcional) Reiniciar Apache: `sudo systemctl restart apache2`
10. Abrir navegador (en Ubuntu) e ingresar en <http://localhost> para cargar la página de Apache.
11. Si has instalado phpMyAdmin, ingresar <http://localhost/phpmyadmin> para cargarlo.

7. Ejemplos prácticos (3 - Máquina virtual)

A diferencia de Windows, el equivalente a la carpeta **htdocs** está en **/var/www/html**.

Accede a la carpeta: `cd /var/www/html`

Crear carpeta en el directorio **html** (p.ej. **test**): `sudo mkdir test`

Accede a la subcarpeta: `cd test`

Crea un fichero **index.html**: `sudo touch index.html`

Edita su contenido creando un “Hola mundo” similar al que hemos visto anteriormente para el servidor local con XAMPP: `sudo nano index.html`

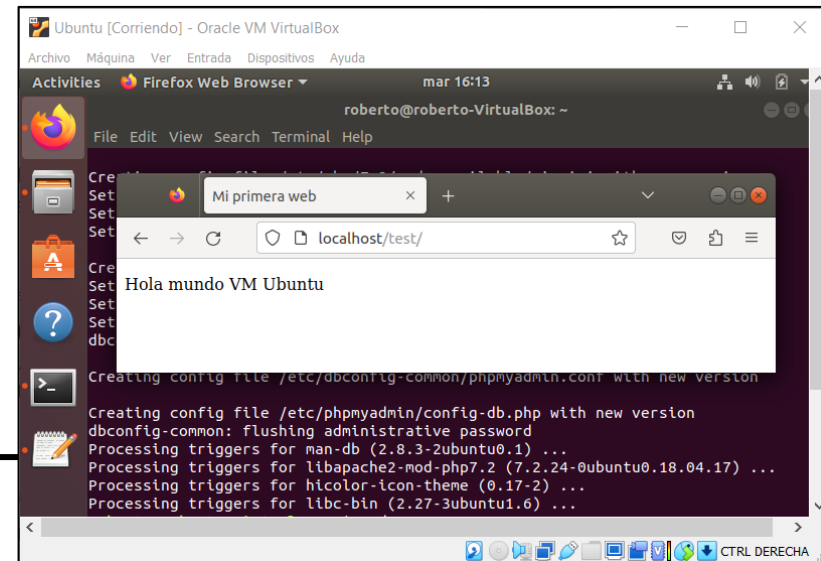
En Ubuntu, introducir la URL <http://localhost/test>

Esto es el equivalente a lo que hemos conseguido anteriormente, un servidor local en una máquina Linux en vez de Windows

A continuación, vamos a acceder desde fuera...

DAW - Despliegue de aplicaciones web

1. Introducción

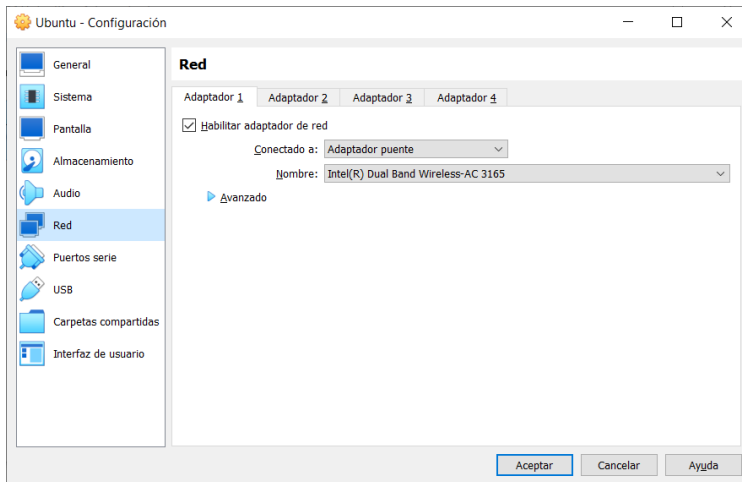


7. Ejemplos prácticos (3 - Máquina virtual)

En este caso vamos a acceder desde el sistema operativo host (Windows), aunque a efectos prácticos es como si accediéramos desde un ordenador remoto.

Apaga la MV (cierra Ubuntu o directamente desde la consola de VirtualBox).

En la **Configuración** de la MV → **Red** → Habilitar el adaptador de red para que esté conectado como **Adaptador puente** (*Bridged adapter*)

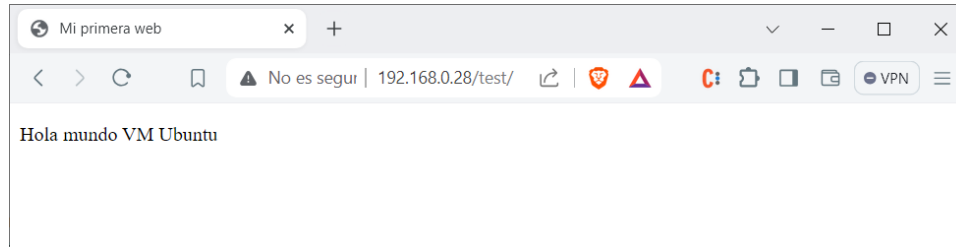


Iniciar la MV y en un Terminal, introducir **ip addr**

```
roberto@roberto-VirtualBox:~$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state
UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
fq_codel state UP group default qlen 1000
    link/ether 08:00:27:f7:5a:4c brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.28/24 brd 192.168.0.255 scope global dyn
amic noprefixroute enp0s3
```

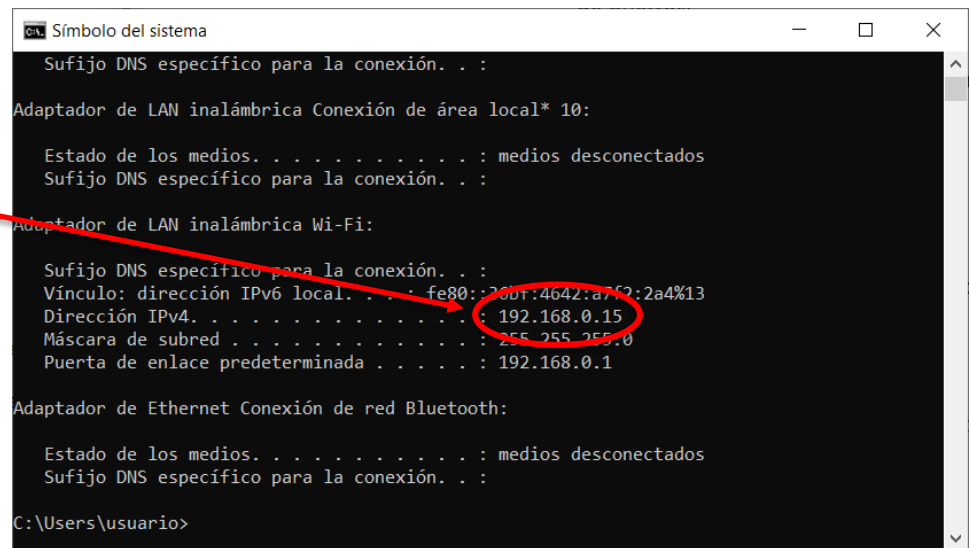
7. Ejemplos prácticos (3 - Máquina virtual)

Desde Windows, introducir la URL en un navegador: <http://192.168.0.28/test>



Podemos comprobar que esta IP es efectivamente distinta de la que tenemos asignada a nuestra máquina huésped (Windows en este caso).

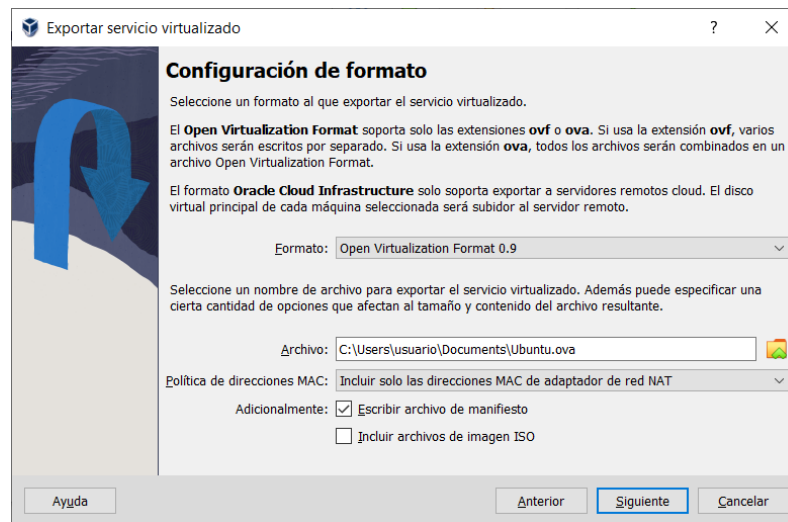
Esto quiere decir que a efectos prácticos son máquinas distintas y podrían estar en lugares físicos distintos.



7. Ejemplos prácticos (3 - Máquina virtual)

La (sencilla) aplicación web que hemos creado ya está incluida en una máquina virtual que se puede desplegar en cualquier servidor de hosting.

Para ello se puede exportar la máquina virtual Ubuntu desde la consola de VirtualBox

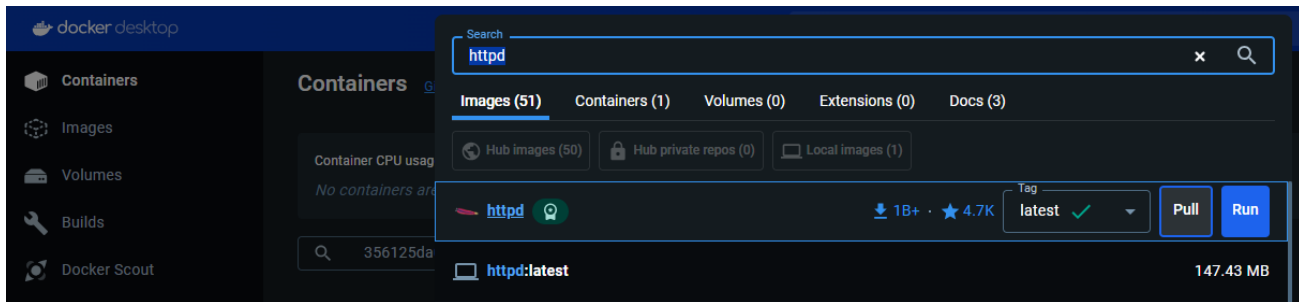


En el servidor de hosting solo será necesario disponer de un gestor de máquinas virtuales como VirtualBox para cargar la máquina virtual.

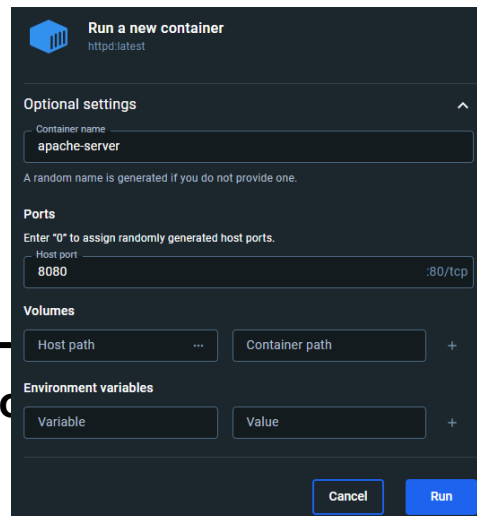
7. Ejemplos prácticos (4 - Docker)

Instalar Docker Desktop: <https://docs.docker.com/get-docker/>

Descargar la imagen de Apache (**httpd**) mediante la interfaz gráfica o mediante el comando **docker pull httpd**

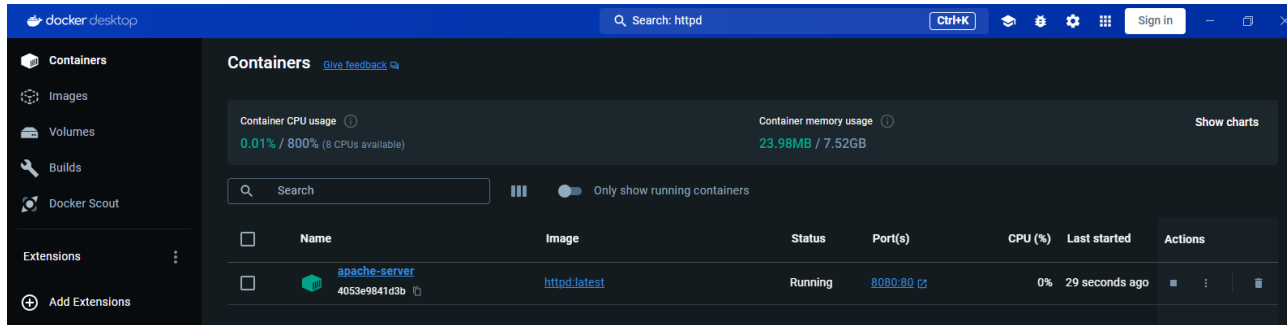


Ejecutar la imagen: **docker run -dit --name apache-server -p 8080:80 httpd**
O desde la interfaz, en Images:

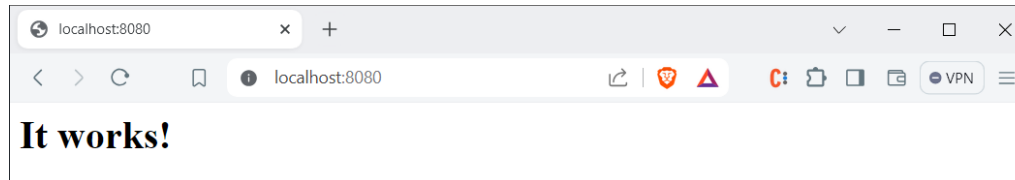


7. Ejemplos prácticos (4 - Docker)

Ahora ya aparece el contenedor activo, generado a partir de la imagen:



Para comprobarlo, accede a la URL: <http://localhost:8080>



7. Ejemplos prácticos (4 - Docker)

Hemos utilizado una imagen genérica con el servidor web Apache, ahora vamos a desarrollar de nuevo nuestra aplicación “Hola mundo” para incluirla en una imagen personalizada que nos permita realizar su despliegue en un contenedor.

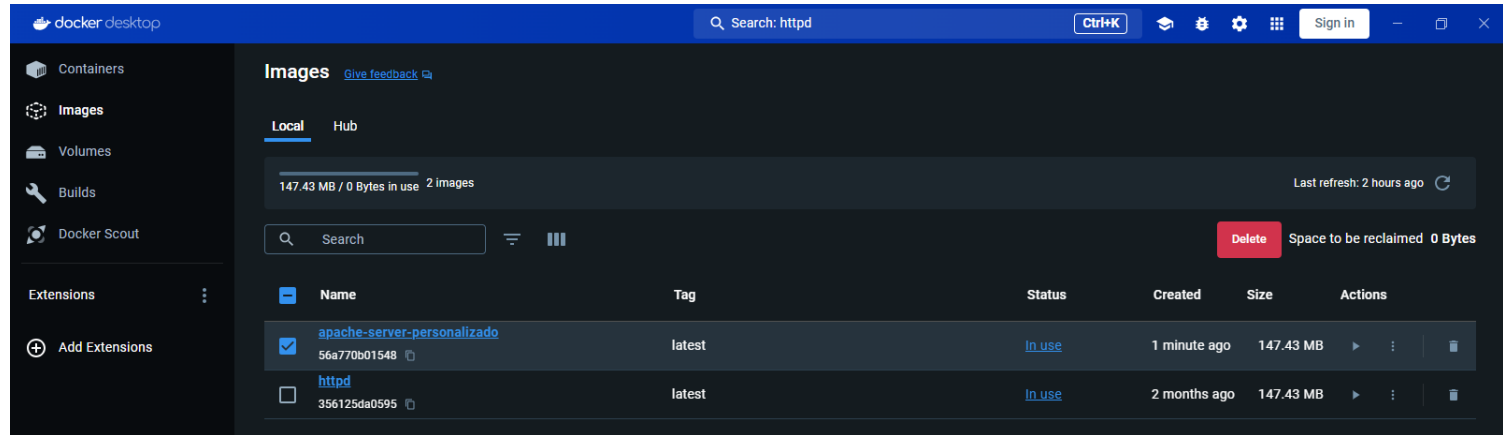
1. Crea una carpeta (no importa dónde) donde alojar la aplicación (p.ej. **test_docker**).
2. En la carpeta que has creado, crea un fichero llamado **Dockerfile** (sin extensión) con el siguiente contenido:

```
FROM httpd:2.4
COPY ./public-html/ /usr/local/apache2/htdocs/
```

3. En la carpeta que has creado, crea una subcarpeta llamada **public-html**, donde deberás crear también un fichero **index.html** “Hola mundo” con el mismo contenido que en ejemplos anteriores.
4. Desde CMD, Powershell o Terminal, accede a la carpeta **test_docker** y ejecuta la instrucción para crear la imagen (ATENCIÓN, el punto final “.” forma también parte de la instrucción): **docker build -t apache-server-personalizado .**

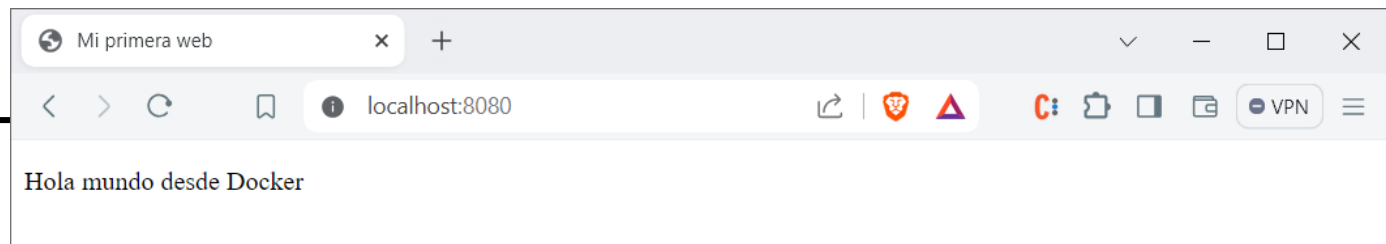
7. Ejemplos prácticos (4 - Docker)

En Docker Desktop puedes comprobar que ya aparece la imagen que acabas de crear, junto a la imagen de Apache (httpd) que habías descargado anteriormente.



Ejecuta, igual que antes, la imagen para que se genere el contenedor correspondiente, bien desde la interfaz o desde línea de comandos: `docker run -dit --name apache_personalizado -p 8080:80 apache-server-personalizado`

Para comprobar el funcionamiento, accede a la URL: <http://localhost:8080>



DAW - Despliegue

1. Introducción

7. Ejemplos prácticos (4 - Docker)

Ahora ya tenemos una imagen personalizada con nuestra aplicación correctamente desarrollada y testeada.

El siguiente paso es poder desplegar esta imagen como un contenedor en el servidor que hayamos preparado para alojar nuestra aplicación.

La forma más sencilla de hacerlo es subir la imagen a un registro de Docker, llamado Docker Hub (o a un registro privado), desde donde se podría descargar, tal y como hemos hecho al inicio de este ejemplo con el servidor Apache genérico. Esto se verá en temas sucesivos.



<https://hub.docker.com/>