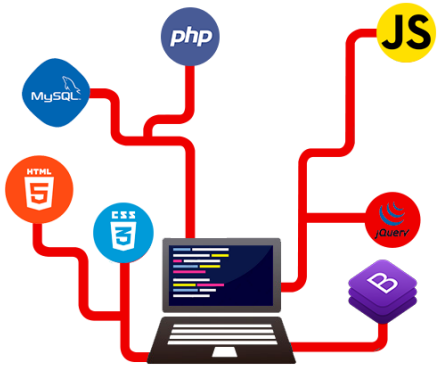


Instituto de Enseñanza Secundaria

Santiago Hernández

UD1.- Plataformas de programación web en entorno servidor

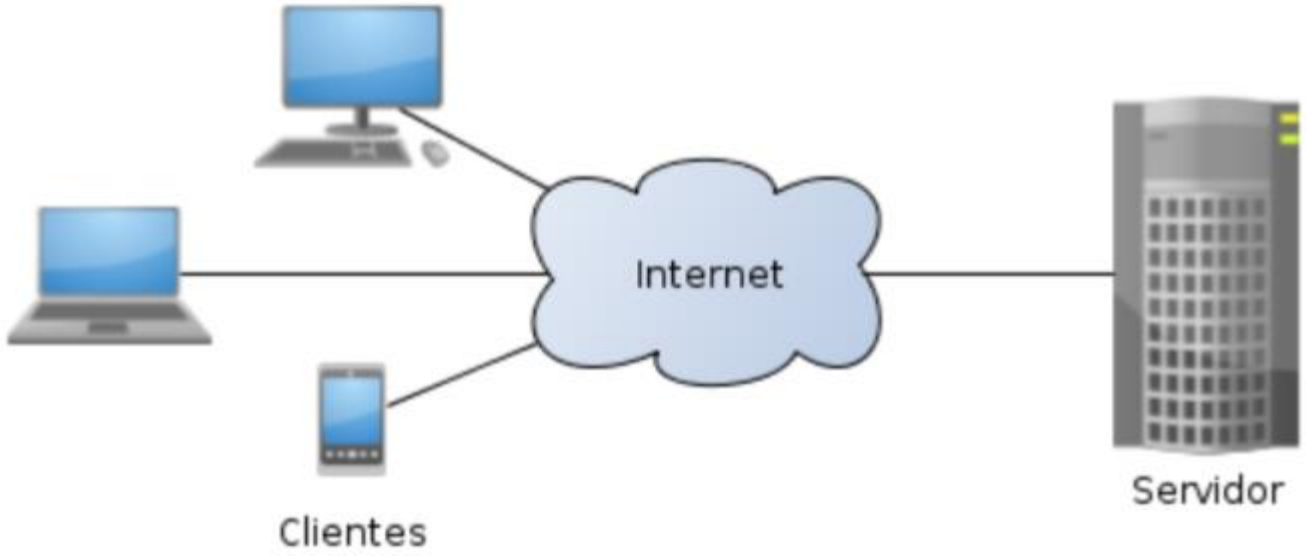
Curso 2021 – 2022

DESARROLLO WEB EN ENTORNO DE SERVIDOR

ÍNDICE

MODELOS DE EJECUCIÓN DE CÓDIGO EN SERVIDORES Y CLIENTES WEB

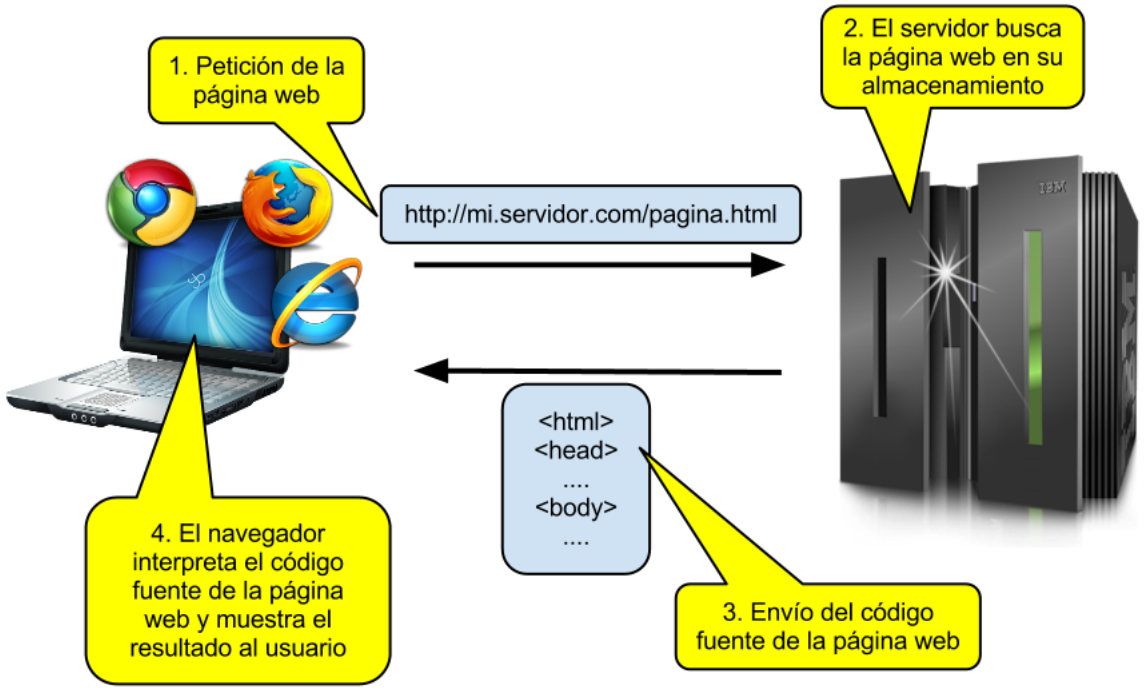
- El desarrollo de aplicaciones web se apoya en la **arquitectura cliente-servidor**.



MODELOS DE EJECUCIÓN DE CÓDIGO EN SERVIDORES Y CLIENTES WEB

1. El **cliente** se conecta al **servidor** para **solicitar** algún servicio → **Petición**.
2. El **servidor** se encuentra en ejecución de forma **ininterrumpida** a la **espera** de que los diferentes clientes realicen una solicitud → **Respuesta**.
3. Se suele tratar de obtener el contenido de una **página web**. También podemos hablar de servicios web donde no se generan páginas web sino **contenido en XML o JSON** para ser consumido por una aplicación cliente.
4. La **solicitud** que hacen los clientes al servidor se le llama petición (***request***)
5. Lo que el servidor devuelve a dicho cliente le llamamos **respuesta** (*response*).
6. Estos términos son los usados por el protocolo **HTTP**.
7. La arquitectura cliente-servidor plantea la posibilidad de **numerosos clientes** atendidos por un mismo servidor. El **servidor** será un software **multitarea** capaz de atender peticiones simultáneas de numerosos clientes.
8. Cuando una aplicación o servicio web tiene muchas solicitudes por unidad de tiempo puede ser que un conjunto de servidores o **cluster** desempeñen este servicio en equipo.

MODELOS DE EJECUCIÓN DE CÓDIGO EN SERVIDORES Y CLIENTES WEB



PÁGINAS ESTÁTICAS Y DINÁMICAS

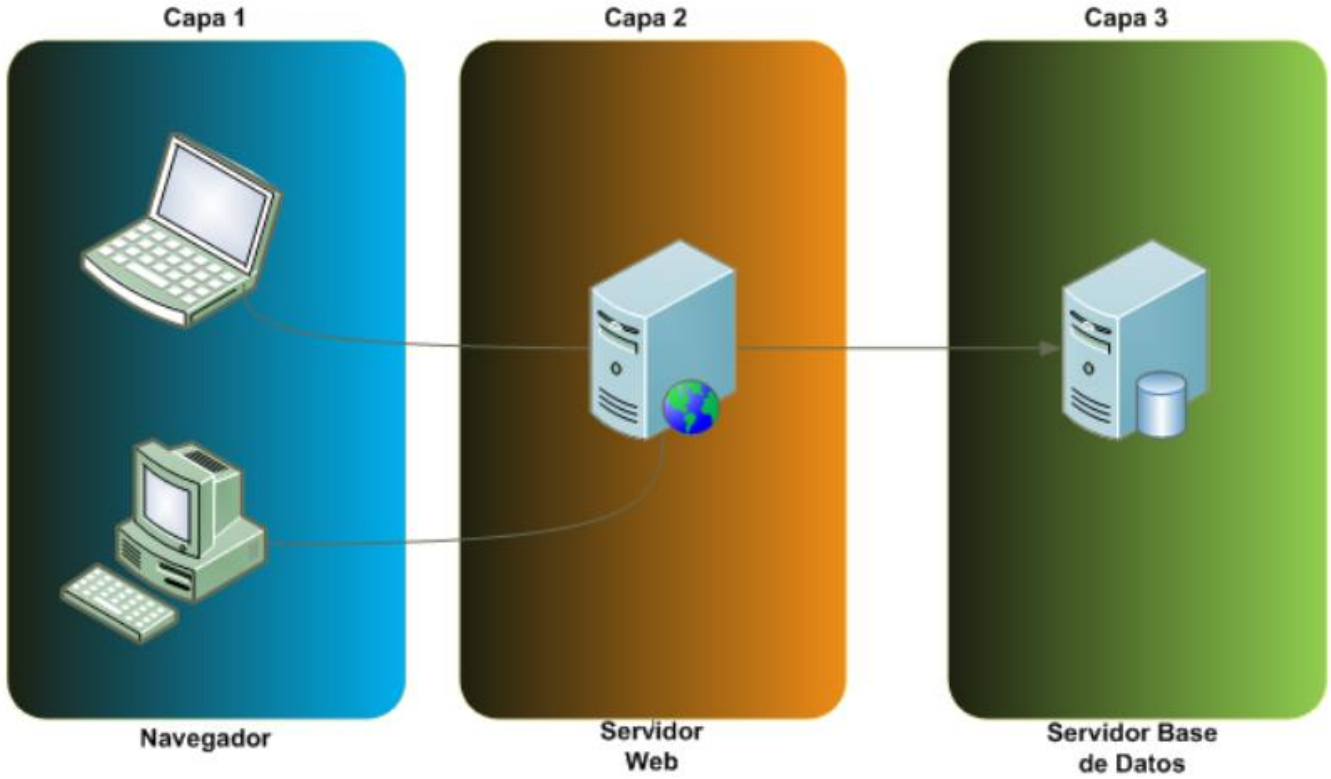
- **Páginas estáticas**
  - Están construida enteramente con código HTML y CSS, código escrito directamente en ficheros y que no cambia.
  - Para visualizarla sólo es preciso un navegador web. No hace falta ningún otro software.
- **Páginas dinámicas**
  - Una página es dinámica cuando el contenido y aspecto es cambiante.
  - Estos cambios pueden ser fruto de que el servidor modifica el HTML entre peticiones → **páginas dinámicas del lado del servidor**.
  - También pueden ocurrir si ejecutamos código en el navegador, javascript → **páginas dinámicas del lado del cliente**.
  - La tecnología AJAX realiza una combinación de ambos.
    - Permite actualizar el contenido de una página sin recargarla completamente.
    - Javascript solicita datos al servidor.
    - Los datos recibidos son usados para renovar el contenido de la página web.

PÁGINAS ESTÁTICAS Y DINÁMICAS

Páginas dinámicas

DESARROLLO EN CAPAS

- Es un modelo de desarrollo software en el que el objetivo primordial es la separación de las partes que componen un sistema:
  - **Acceso a Datos.**
  - **Lógica de negocio.**
  - **Capa de presentación.**
- Podemos separar funcionalidades.
- Seguimos trabajando con filosofía cliente-servidor.
- Separamos el servidor en dos máquinas.



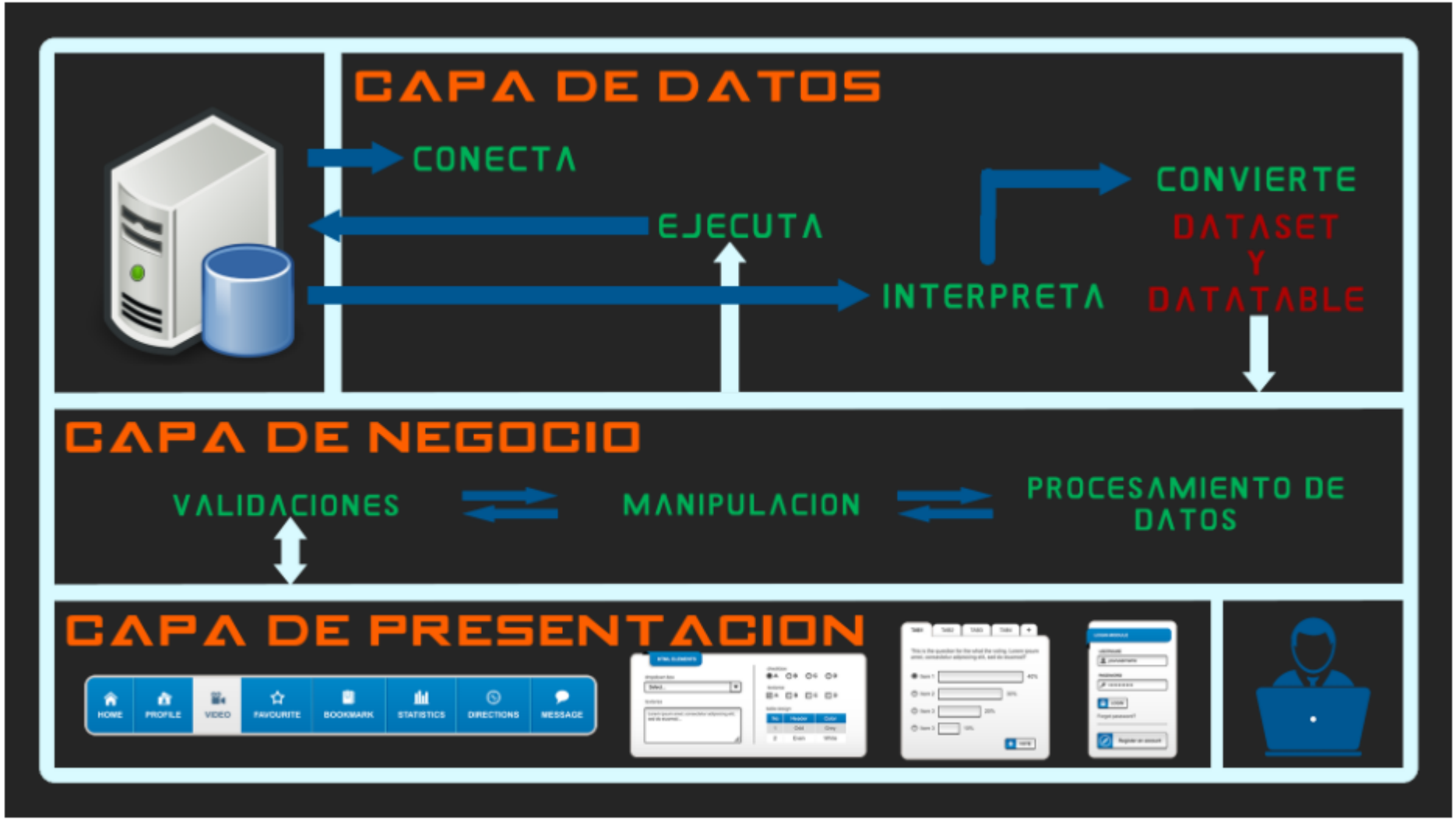
DESARROLLO EN CAPAS

- **Capa de presentación**
  - Capa donde la aplicación se **muestra** al usuario.
  - Básicamente es la **GUI** (Graphical User Interface, Interfaz Gráfica de Usuario).
  - En el caso de una aplicación web sería el código HTML que se carga directamente en el **navegador web**.
  - Se ejecuta directamente en el equipo del cliente.
- **Capa de negocio**
  - Capa intermedia donde se lleva a cabo toda la **lógica** de la aplicación.
  - Se ejecuta en el lado servidor.
  - Tras realizar los cálculos y/o operaciones sobre los datos, genera el código HTML que será presentado al usuario en la capa siguiente.

DESARROLLO EN CAPAS

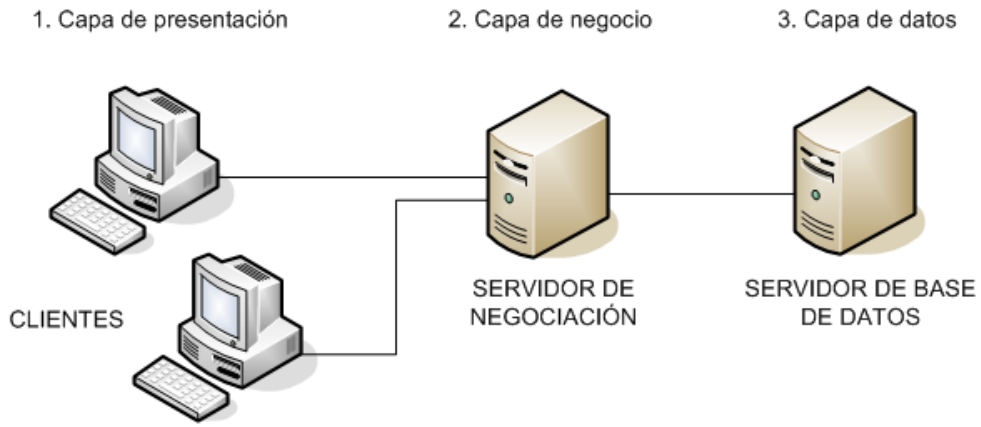
- **Capa de datos**
  - Capa que **almacena los datos**.
  - Hace referencia al **SGBD** que es el encargado de almacenar los datos.
  - Dependiendo de la arquitectura de la aplicación, esta capa y la de negocio se pueden encontrar físicamente en el mismo equipo, aunque también es posible que se tengan que separar por cuestiones de rendimiento.
  - La capa de datos sirve toda la información necesaria a la capa de negocio para llevar a cabo sus operaciones.

DESARROLLO EN CAPAS



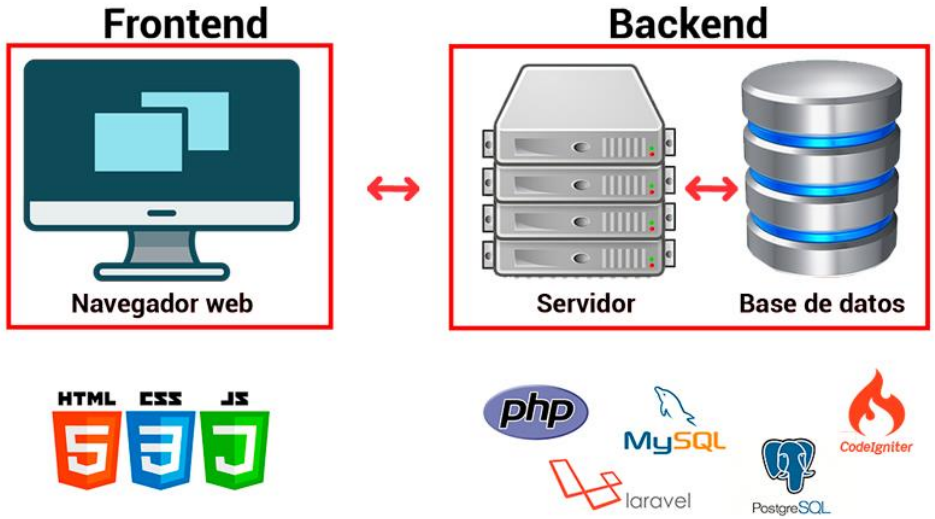
DESARROLLO EN CAPAS

- En una **aplicación web** nos encontramos con:
  - **Navegador web**
    - Mozilla Firefox, Internet Explorer o Google Chrome...
  - **Servidor web**
    - Acompañado de un **lenguaje de programación** web permite desarrollar la parte que ocupa la capa de negocio.
    - Ejemplo: Apache + PHP.
  - **Base de datos**
    - Cualquier SGBD relacional.
      - MySQL o PostgreSQL.
    - Sistemas no relacionales.
      - MongoDB.



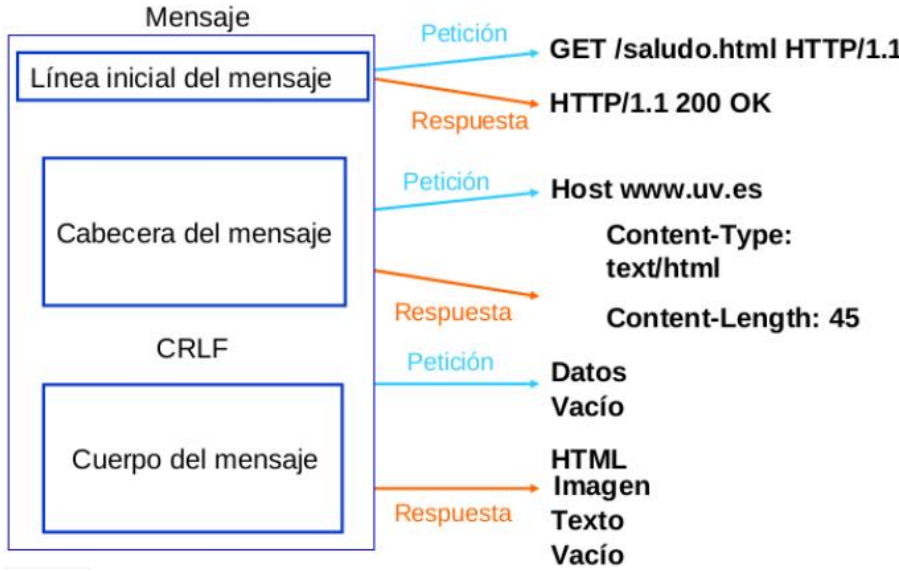
TECNOLOGÍAS DE DESARROLLO SERVIDOR

- **Java EE**
  - Servidor de aplicaciones con JSP y Servlest.
- **PHP en conjunción con un servidor web**
  - Es el más extendido.
- **Node.js**
  - Utiliza Javascript.
- Otras:
  - .Net, CGI, Rubi, Python ...



HTTP

- **HTTP** es el protocolo usado para la transferencia de recursos en la web.
- El **cliente** o **navegador** se denomina *User Agent*.
- Se basa en transacciones según el esquema **petición-respuesta**.
- Los recursos se identifican por un localizador único: **URL**.
- Es un protocolo **sin estado**.
  - No guarda información entre conexiones.
  - Esto lo hace flexible y escalable pero es un problema para crear aplicaciones web.
  - Http requiere de un extra para *recordarnos* cuando abrimos **sesión** en una aplicación.

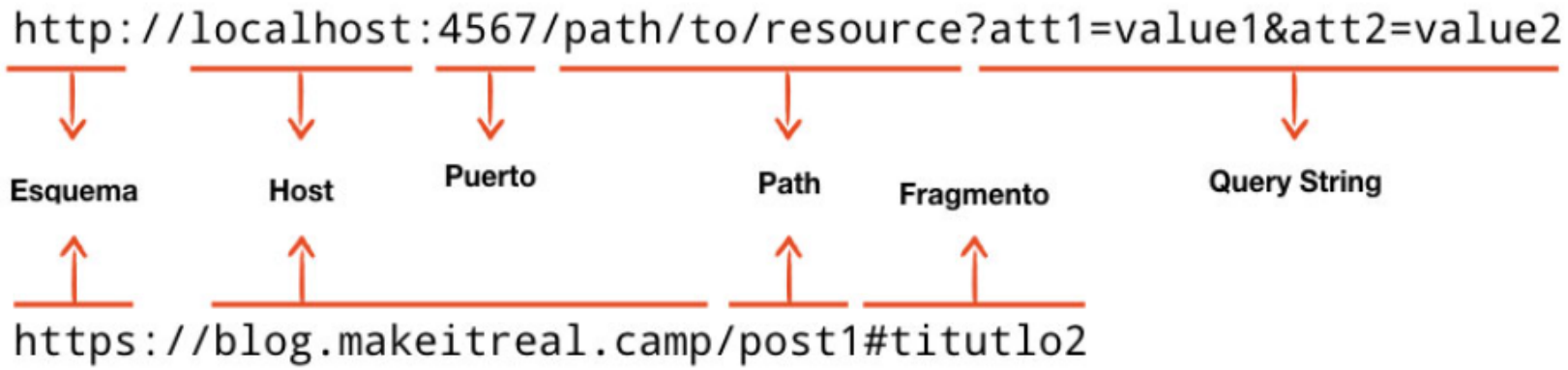


LAS URL

- Esquema de una URL es

  `protocolo://maquina:puerto/camino/fichero`

  - Protocolo: http o https.
  - Máquina: una IP o un nombre (DNS).
  - El puerto suele omitirse por usarse los de defecto: 80 y 443 para http y https respectivamente.
  - Camino o ruta relativa al directorio raíz del sitio.
  - Fichero es el nombre del recurso.



PETICIONES HTTP

- Un mensaje HTTP se compone de:
  - **La primera línea**.
    - Es diferente para la petición y la respuest.
  - Los **encabezados**.
  - El **cuerpo** (opcional)
- Estructura de las peticiones:

```
Método SP URL SP Versión Http retorno_de_carro
(nombre-cabecera: valor-cabecera (, valor-cabecera)*CRLF)*
Cuerpo del mensaje

nota:
CRLF es retorno de carro
SP es espacio
El cuerpo suele ir vacío pero no cuando enviamos formularios, subimos ficheros
```

PETICIONES HTTP

- Ejemplo de una petición:

```
GET /index.html HTTP/1.1
Host: www.sitioweb.com:8080
User-Agent: Mozilla/5.0 (Windows;en-GB; rv:1.8.0.11) Gecko/20070312
[Línea en blanco]
```

  - Encabezados comunes:
    - **Content-Type**: el tipo de contenido que se está enviando en el cuerpo de un mensaje de petición, por ejemplo text/html.
    - **Accept**: el tipo de contenido que el cliente está esperando.
    - **User-Agent**: el tipo de navegador que está haciendo la petición

- Solicitud del recurso /index.html de wikipedia.org.
  - Primera línea:
    - El **verbo** (en este caso GET)
    - El **recurso** (en este caso /index)
    - La **versión de HTTP** (en este caso HTTP/1.1)

```
GET /index.html HTTP/1.1
Host: wikipedia.org
Accept: text/html
```

PETICIONES HTTP

- Ejemplo con el método POST:

```
POST /en/html/dummy HTTP/1.1
Host: www.explainth.at
User-Agent: Mozilla/5.0 (Windows;en-GB; rv:1.8.0.11) Gecko/20070312 Firefox/1.5
Accept: text/xml,text/html;q=0.9,text/plain;q=0.8, image/png,*/*;q=0.5
Accept-Language: en-gb,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://www.explainth.at/en/misc/httpreq.shtml
Content-Type: application/x-www-form-urlencoded
Content-Length: 39

name=MyName&married=not+single&male=yes
```

PETICIONES HTTP

- En la cabecera se pueden enviar informaciones como:
  - El nombre del navegador.
  - El tipo de contenido solicitado y el aceptado (p. ej. página html, o un pdf, ...).
  - El juego de caracteres.
  - El idioma preferido.
  - Si se admite contenido comprimido ...
- El **cuerpo** del mensaje en las peticiones GET está vacío.
- La lista de métodos es bastante amplia pero de momento sólo nos preocupan los métodos GET y POST.
  - **GET** pide un recurso.
  - **POST** envía datos al servidor para ser procesados. Los datos se incluyen en el cuerpo del mensaje.

RESPUESTAS HTTP

- Estructura de una respuesta:

```
Versión-http SP código-estado SP frase-explicación retorno_de_carro
(nombre-cabecera: valor-cabecera ("," valor-cabecera)* CRLF)*
Cuerpo del mensaje
```

- Ejemplo:

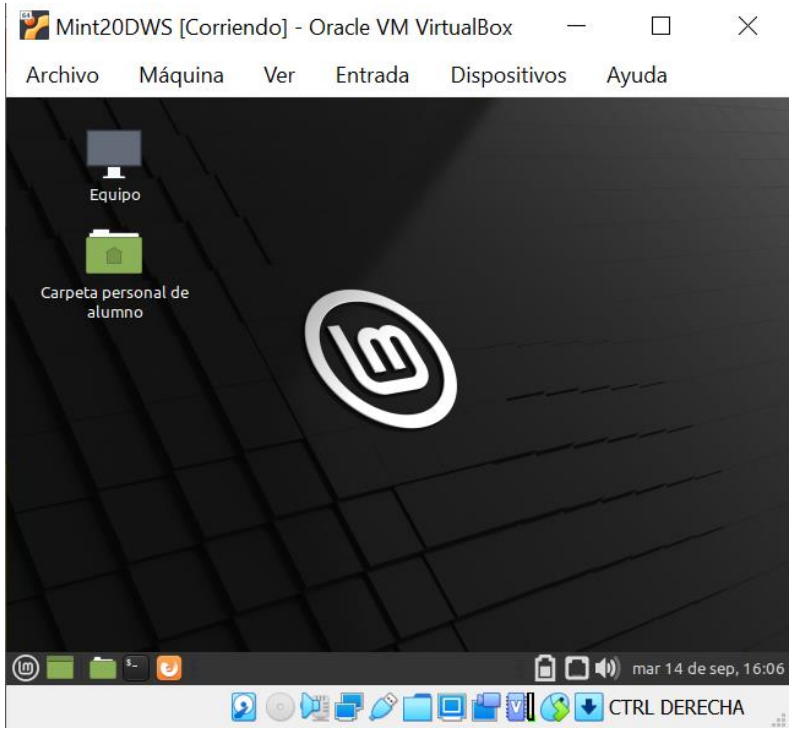
```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: 673
<html>
<head> <title> Título de nuestra web </title> </head>
<body>
¡Hola mundo!
</body>
</html>
```

CÓDIGOS DE ESTADO

- Un elemento importante de la respuesta es el código de estado.
- Resumen de estados:
  - Códigos 1xx : Mensajes
    - 100-111 Conexión rechazada
  - Códigos 2xx: Operación realizada con éxito
    - **200 OK**
  - Códigos 3xx: Redirección
  - Códigos 4xx: Error por parte del cliente
    - **403 Prohibido**
    - **404 No encontrado**
  - 500 Error interno
    - 501 No implementado
    - 502 505 Versión de HTTP no soportada

ENTORNO DE TRABAJO

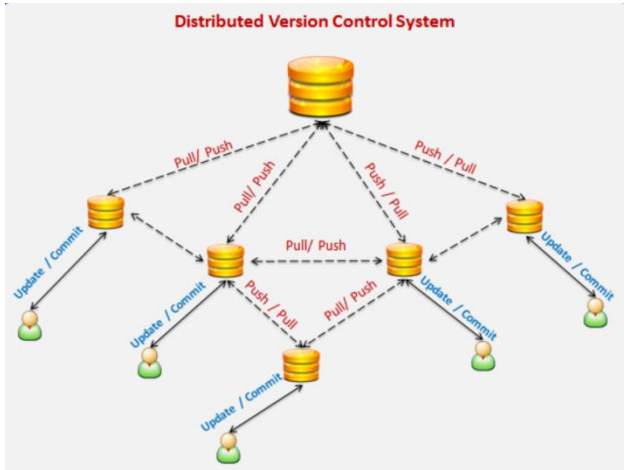
- Máquina virtual con **Linux Mint**:
  - VSC o **Microsoft Visual Studio Code** como editor de código.
  - **Git** como Sistema de Control de Versiones.
  - Servidor Web **Apache**.
  - Intérprete **PHP** y ***composer***.
  - Docker como entorno de virtualización.
  - Node (y npm)



CONTROL DE VERSIONES GIT

- Un **control de versiones** es un sistema que registra los cambios realizados en un archivo o conjunto de archivos a lo largo del tiempo, de modo que puedas **recuperar** versiones específicas más adelante.
- **SCV centralizados**
  - Enfoque tradicional. Ejemplo: CVS O Subversion.
  - Es fácil de administrar pero obliga a que el servidor siempre esté disponible.

- **SCV distribuidos**
  - **Git**, como Mercurial, Bazaar.
  - Los clientes mantienen una copia completa de todo el repositorio.
  - Permite hacer cambios sin estar conectados.
  - En algún momento deben sincronizarse los clientes y el servidor o servidores.



CONTROL DE VERSIONES GIT

Git

- Linus Torvaldas, creador de Linux, desarrolló Git para dar soporte al desarrollo de este SO.
- Características:
  - Es distribuido.
  - Sencillo y rápido.
  - Permite grandes proyectos con multitud de ramas.
  - Usa copias completas. Hace copia de los ficheros modificados y enlaces a los no modificados.
  - Asegura la integridad. Realiza un hash SHA-1 a cada fichero. Si un fichero es alterado git lo detecta.
  - La mayoría de cambios se hacen localmente.

GitHub (o bitbucket)

- Git puede usarse **localmente** y sin conectarse a ningún otro equipo.
- El uso de un servidor **remoto**:
  - Facilita la compartición de código y el trabajo en equipo
  - Aumenta la seguridad de nuestro código.

ESQUEMA DE ALMACENAMIENTO EN GIT

- Git crea una ***instantánea*** de cada archivo que es guardado.
  - Cómo es el archivo en un determinado momento.
  - Permite hacer un seguimiento del archivo.
- Cada instantánea se identifica por un hash SHA1 (código alfanumérico).
- Es posible **restaurar** una instantánea anterior.
- Es posible **crear ramas** sobre la misma aplicación.
- Y **fusionar** las ramas en un único archivo.
  - Detecta conflictos.
    - ¿qué hacer cuando hay cambios de distintos programadores sobre un mismo código?

GIT – FLUJO DE TRABAJO

Crear el repositorio (proyecto)

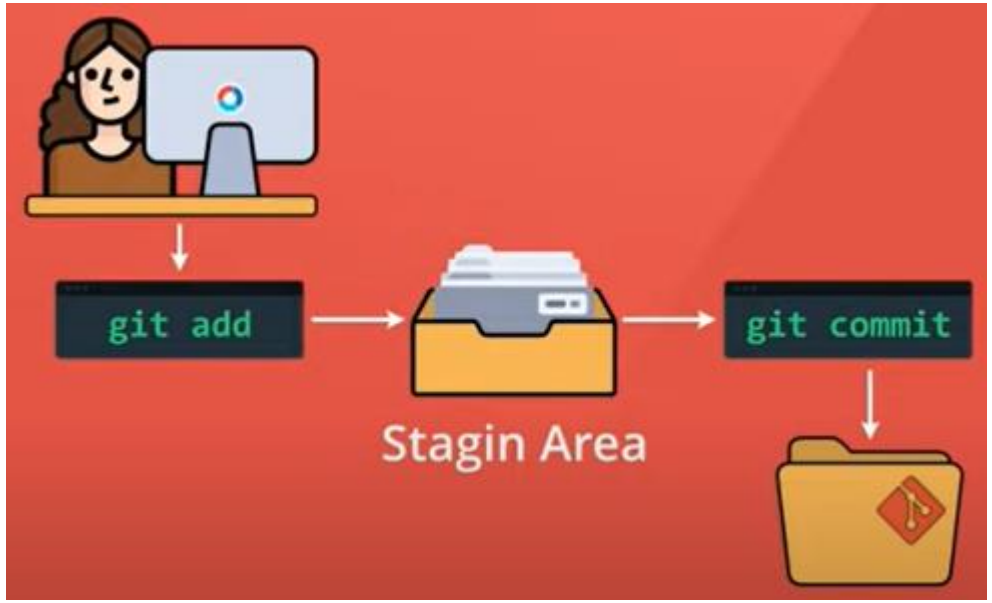
- `git init`
- `git clone`

Almacenar los cambios en el área de preparación

- `git add`

Comprometer los cambios en el repositorio

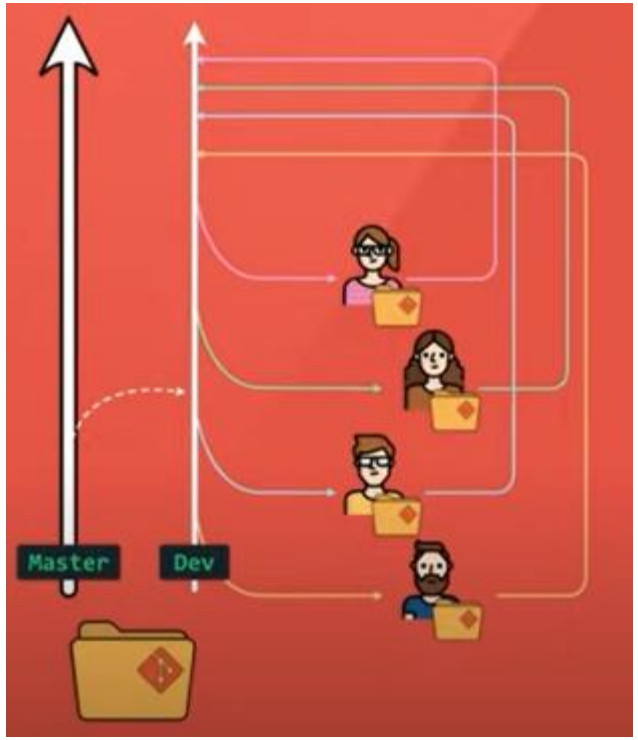
- `git commit`



GIT

Ramas

- Varias ramas del mismo proyecto:
  - Master o rama principal
  - Dev o rama de desarrollo
    - Cada desarrollador crea su propia rama.
    - Cuando el trabajo se completa la rama debe integrarse (**merge**) con dev.



Herramientas

- Línea de comandos
- Clientes gráficos → *para repositorios en **local***
  - *GitHub desktop*, *GitKraken*
- Repositorios en la nube → *para repositorios **remotos***
  - *Github, Bitbucket, Gitlab*
- Integrado en editores de código
  - *VS code → integra muy bien con Github*

DIAGRAMA DE GIT

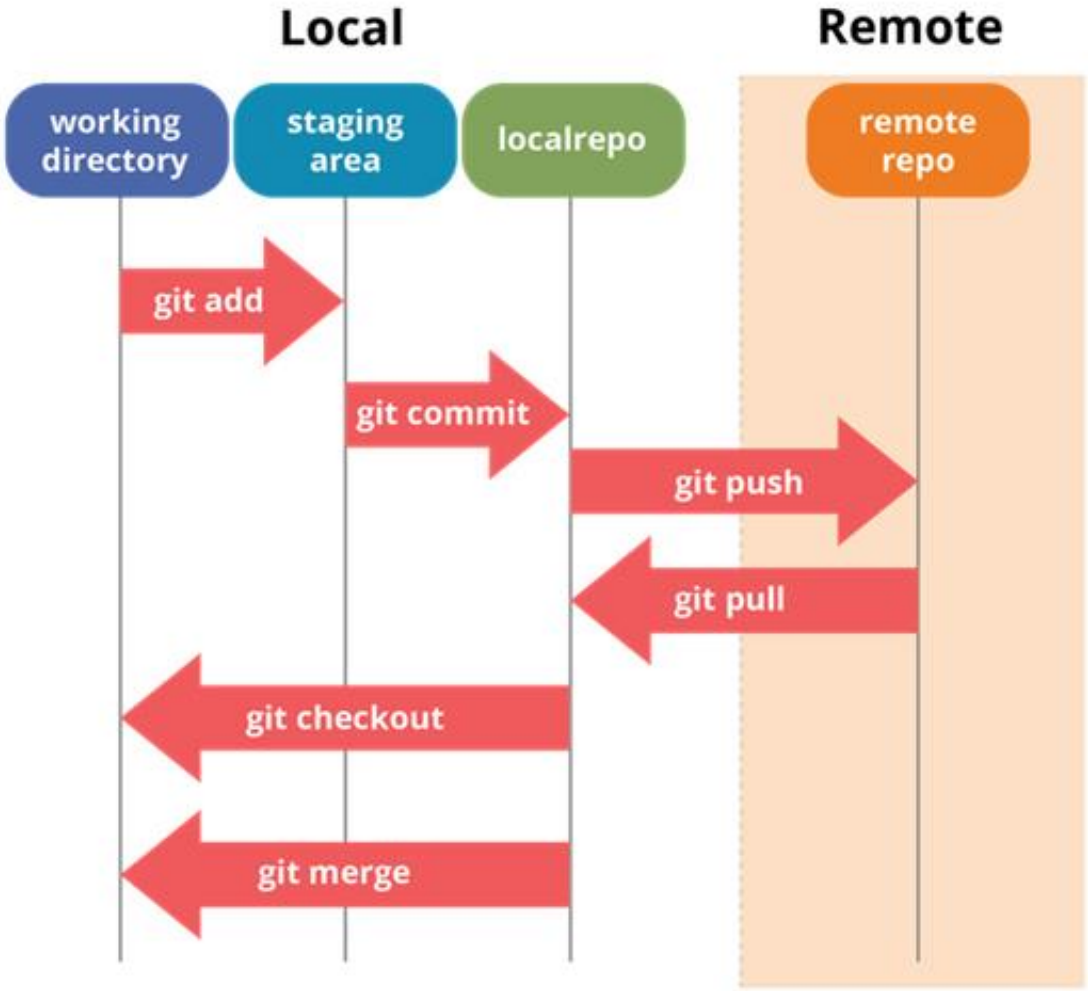


DIAGRAMA DE GIT

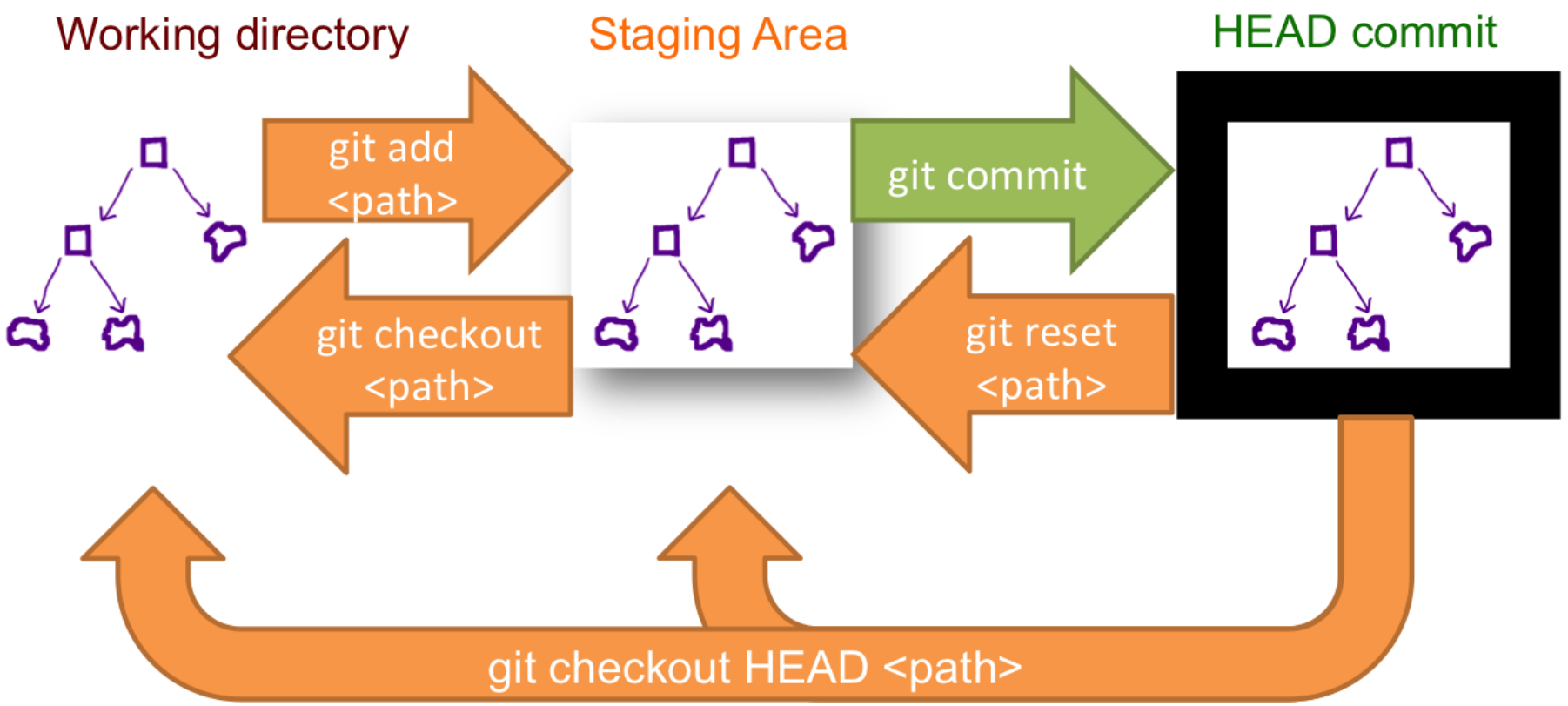


DIAGRAMA DE GIT