



El futuro digital  
es de todos

MinTIC



Vigilada Mineducación



## CICLO III: Desarrollo de Software

**Hechos**  
QUE CONECTAN 

# Sesión 03:

# Desarrollo de Software

Introducción al desarrollo Web y plataformas de  
desarrollo colaborativo

# Objetivos de la sesión

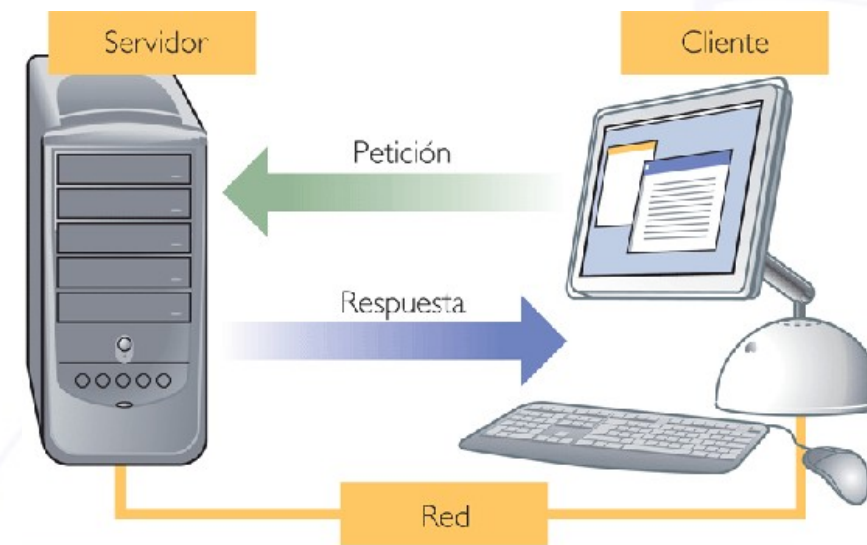
Al finalizar esta sesión estarás en capacidad de:

1. Modelo cliente-servidor en el desarrollo Web.
2. Rol del cliente, el navegador, en el desarrollo web.
3. Rol del servidor en el desarrollo Web.
4. Protocolo HTTP.
5. Presentar qué es una plataforma de desarrollo colaborativo.
6. Exponer la importancia del uso de plataformas para el desarrollo colaborativo.
7. Manejar de forma básica un proyecto con github.

# Introducción al desarrollo Web

# Modelo Cliente/Servidor - Definiciones

- Es una combinación de sistemas que pueden colaborar entre sí para dar a los usuarios toda la información que requiera sin necesidad de saber donde está ubicada.
- Es una arquitectura de procesamiento cooperativo donde uno de los elementos solicita servicios a otro.
- Es un procesamiento de datos de tipo colaborativo entre dos o más computadoras conectadas a una red.
- Se aplica a la arquitectura de software que describe el procesamiento entre dos o más programas: una aplicación y un servicio.





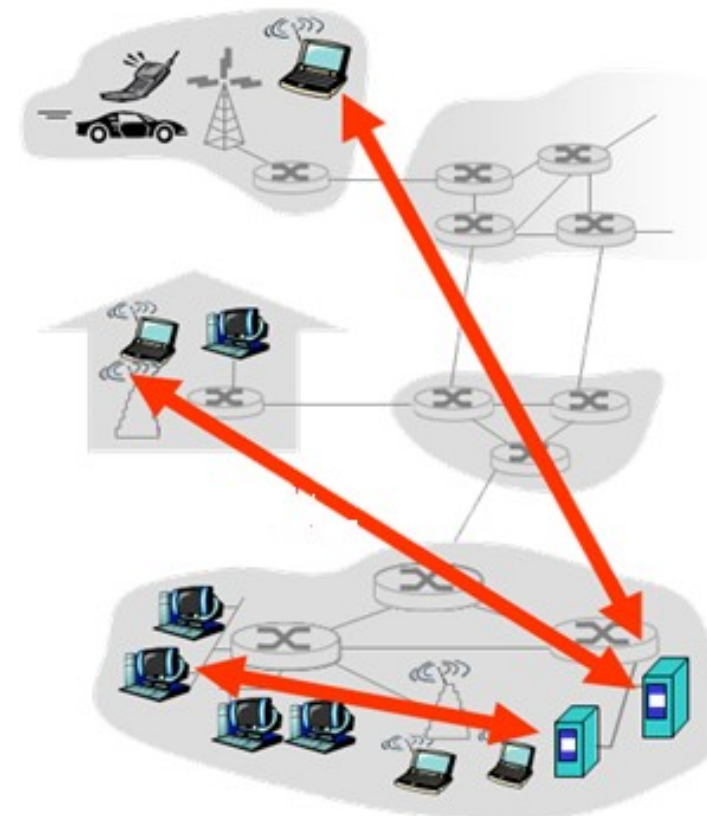
# Modelo Cliente/Servidor - Arquitectura

- **Funciones del Cliente:**

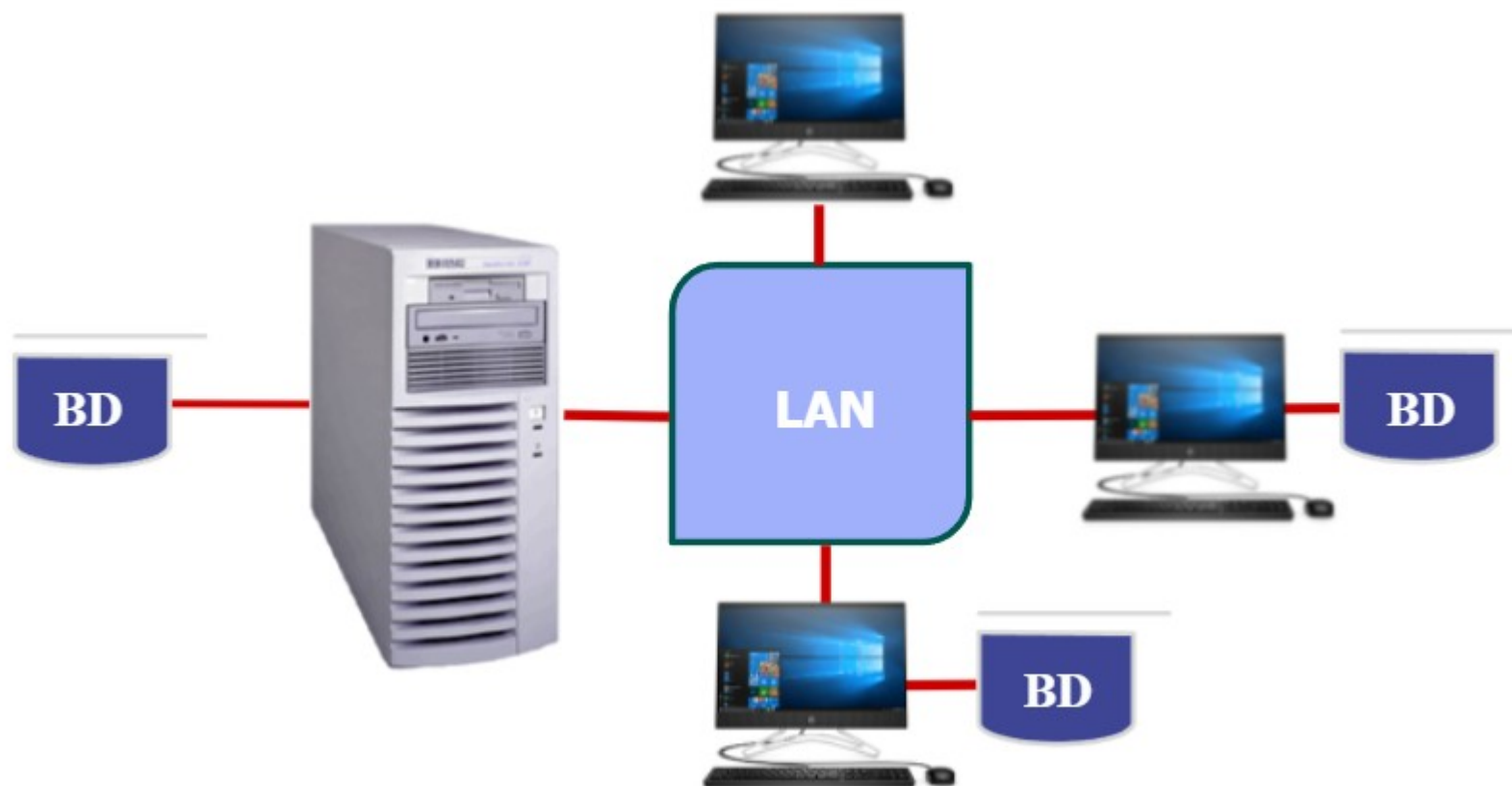
- Interactuar con usuario.
- Manejar procesos de conexión al cliente (Red).
- Enviar solicitudes al servidor.
- Procesar respuestas.
- Control de fallos.
- Control de seguridad.

- **Funciones del Servidor:**

- Recibe (escucha requerimientos).
- Activa procesos de seguridad
- Devuelve solicitudes del cliente.
- Bloqueos, resolver errores,
- Administración y manejo de red.



# Modelo Cliente/Servidor - Procesamiento



# Modelo Cliente/Servidor - Características

- Recursos compartidos.
- Transparencia de ubicación.
- Protocolos asimétricos.
- Encapsulamiento de servicios.
- Facilidad de escalabilidad.
- Integridad.



# Modelo Cliente/Servidor - Tipos de servidores

- Servidores de archivos.
- Servidores de bases de datos.
- Servidores de transacciones.
- Servidores de Groupware.
- Servidores de Objetos.
- Servidores Web.

# Modelo Cliente/Servidor - Elementos

- **Red:** Es un conjunto de elementos (clientes, servidores y base de datos) unidos de forma física o no física por medio de protocolos de transmisión de información establecidos.
- **Cliente:** Hace referencia a un solicitante de servicios, este cliente puede ser un ordenador o una aplicación la cual requiere información procedente de la red para funcionar.
- **Servidor:** Este hace referencia a un proveedor de servicios, este servidor a su vez puede ser un ordenador o una aplicación la cual envía información a los demás elementos de la red.

# Modelo Cliente/Servidor - Elementos

- **Protocolo:** Conjunto de normas o reglas y pasos establecidos de forma clara y concreta sobre el flujo de información en una red estructurada.
- **Servicios:** Conjunto de información que busca responder las necesidades de un cliente, donde esta información pueden ser mail, música, mensajes simples entre software, videos, etc.
- **Almacenamiento de datos:** Son bancos de información estructurada y no estructurada que forman parte de la red.

# Modelo Cliente/Servidor - Tipos de arquitecturas

- **Arquitectura de dos capas:** En esta arquitectura el cliente solicita recursos y el servidor responde directamente a la solicitud.
- **Arquitectura de tres capas:** En la arquitectura de tres capas existe un nivel intermediario, eso significa que la arquitectura generalmente está compartida por un cliente que es el que solicita los recursos provistos con una interfaz de usuario o mediante un navegador web.
- **Arquitectura de N capas:** En este caso de la arquitectura de N capas, el modelo cliente servidor requiere la ayuda de otros servidores para poder ejecutar sus propias tareas.

# Modelo Cliente/Servidor - Ventajas

- Facilita la integración entre diferentes sistemas y comparte información permitiendo que los dispositivos de la red puedan ser utilizados mediante una interfaz más amigable para el usuario.
- Gracias al uso de la interfaz de gráficas interactivas, las aplicaciones realizadas bajo este esquema tienen una mayor interacción con el usuario.
- La estructura modular facilita la integración de nuevas tecnologías y el crecimiento de la infraestructura computacional favoreciendo así la estabilidad de las soluciones.
- El modelo cliente servidor permite a las diferentes áreas de una empresa generar un orden de trabajo en donde cada sector puede trabajar en su área accediendo al mismo servidor e información que los demás, sin generar conflictos.

# Modelo Cliente/Servidor - Desventajas

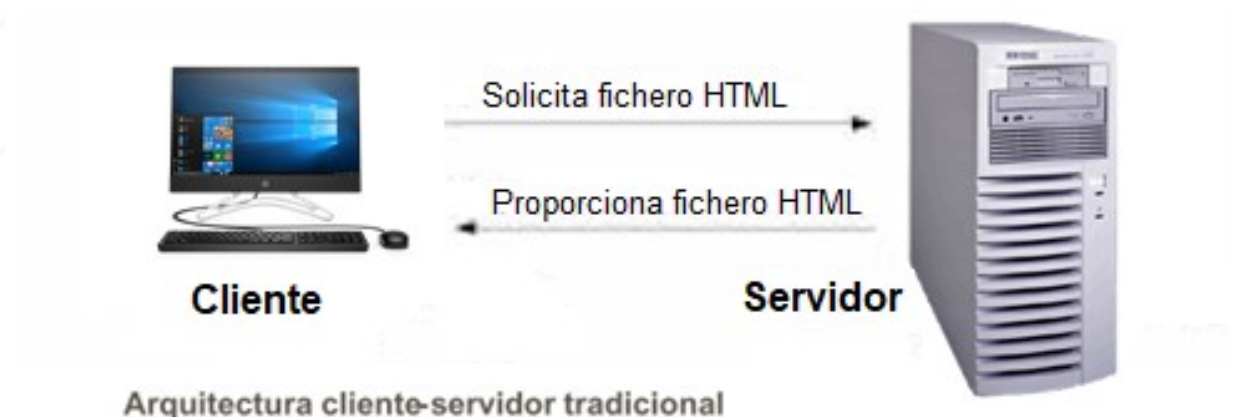
- Se necesita de expertos para que un servidor sea reparado.
- Al compartir canales de información entre servidores y clientes, se requiere que estas pasen por procesos de validación, es decir protocolos de seguridad que pueden tener algún tipo de puerta abierta permitiendo que se generen daños físicos, amenazas o ataques de malware.
- Hay una gran limitación en cuanto a los costos económicos ya que generalmente los servidores son computadoras de alto nivel con un hardware y software específicos para poder dar un correcto funcionamiento a las aplicaciones, las cuales son de alto valor. Además, también tiene un costo elevado reemplazar componentes que estén dañados.



# Modelo Cliente/Servidor - WEB

- Es una arquitectura software que involucra uno o más clientes solicitando servicios a uno o más servidores.
- El cliente puede ser un proceso en ejecución en un dispositivo de cómputo como una PDA o un teléfono móvil.
- El servidor puede ser un proceso en ejecución en un dispositivo de cómputo (normalmente de altas prestaciones).
- En la arquitectura Web actual aparecen además elementos que se sitúan en medio (proxies, cachés).
- **Beneficios:**
  - Usabilidad
  - Flexibilidad
  - Interoperabilidad
  - Escalabilidad.

# Modelo Cliente/Servidor - WEB



# WEB - Servidores

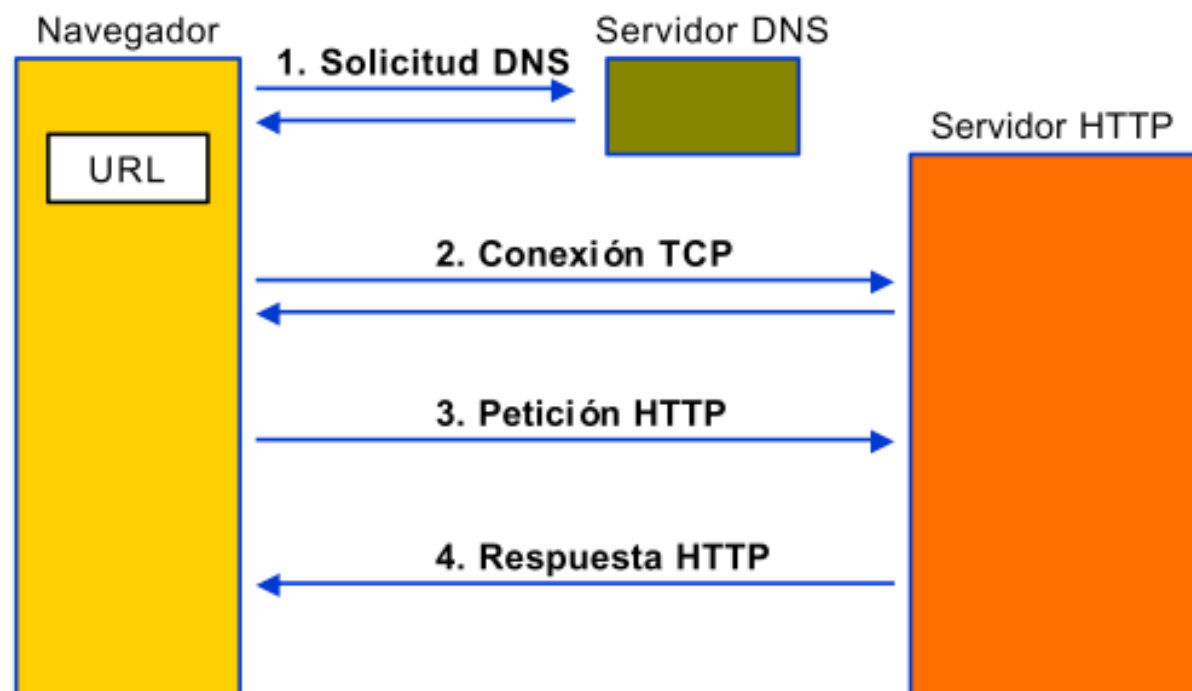
- Programa que contesta y genera la respuesta HTTP a las peticiones de recursos web por parte del cliente.
- Se conecta con el cliente.
- Recibe el mensaje HTTP de la petición.
- Procesa el mensaje HTTP.
- Localiza y envía el resultado (en forma de mensaje HTTP).

# WEB - Cliente

- Originan el tráfico Web.
  - Envían las peticiones y reciben las respuestas.
- Dos clases de clientes web: navegadores y robots.
- Los navegadores (Chrome, IE, etc).
  - Las peticiones están dirigidas por el usuario.
  - Repiten peticiones al mismo objeto cuando navegan por un sitio.
  - Utilizan caches de memoria y disco.
- Robots (spiders, y agentes inteligentes).
  - La velocidad y carga está limitada por la velocidad de proceso, y por la velocidad de la red.
  - Las peticiones son automatizadas.

# WEB - Navegadores

- Programa que ejecuta las solicitudes, a petición de un usuario, y recibe, analiza y presenta las respuestas.
- Pasos:



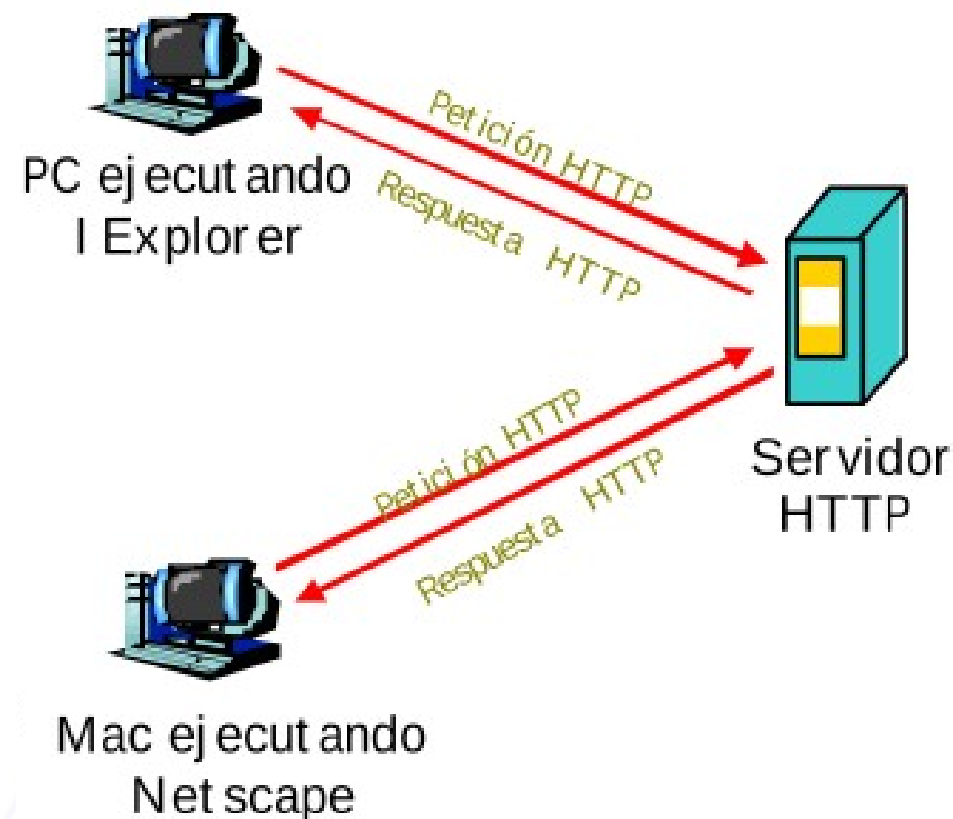
# WEB - Funciones de los Navegadores

- Elaboran y envían la petición HTTP.
- Reciben, interpretan y presentan la respuesta.
- Proporcionan la interfaz para conectarse y utilizar otros servicios: mail, news, FTP, etc. El protocolo por defecto es HTTP.
- Caché local.
- Manejo de las Cookies.



# Protocolo HTTP

- Los elementos software de la arquitectura web (clientes, servidores, proxies) utilizan el protocolo HTTP para comunicarse.
- HTTP define la sintaxis y la semántica que utilizan estos elementos para comunicarse.
- Es un protocolo basado en el esquema petición/respuesta.
- El cliente envía un mensaje de petición y el servidor contesta con un mensaje de respuesta.



# Protocolo HTTP

- El protocolo HTTP está basado en mensajes de texto plano.
- **Ventajas:**
  - Legible.
  - Fácil de depurar.
- **Desventajas:**
  - El mensaje es más largo.
  - Es un protocolo sin manejo de estados:
    - Hay ausencia de estado tras cada par petición-respuesta.
    - Tras la respuesta, el servidor cierra inmediatamente la conexión.
    - No existe el concepto de sesión.

# Tipos de peticiones y respuestas HTTP

## Tipos de peticiones:

- **GET:** Solicita una representación de un recurso específico. Sólo deben recuperar datos.
- **POST:** Envía una entidad a un recurso en específico, causando a menudo un cambio en el estado o efectos secundarios en el servidor.
- **PUT:** Reemplaza todas las representaciones actuales del recurso de destino con la carga útil de la petición.
- **DELETE:** Borra un recurso en específico.
- **Otros tipos de peticiones:**
  - PATCH
  - HEAD
  - OPTIONS
  - CONNECT
  - TRACE

## Tipos de respuestas:

- Informativas (100 - 199).
- Exitosas (200 - 299).
- Redireccionamiento (300 - 399).
- Error del Cliente (400 - 499).
- Error del Servidor (500 - 599).

## HTTP Status Codes



# Protocolo HTTPS

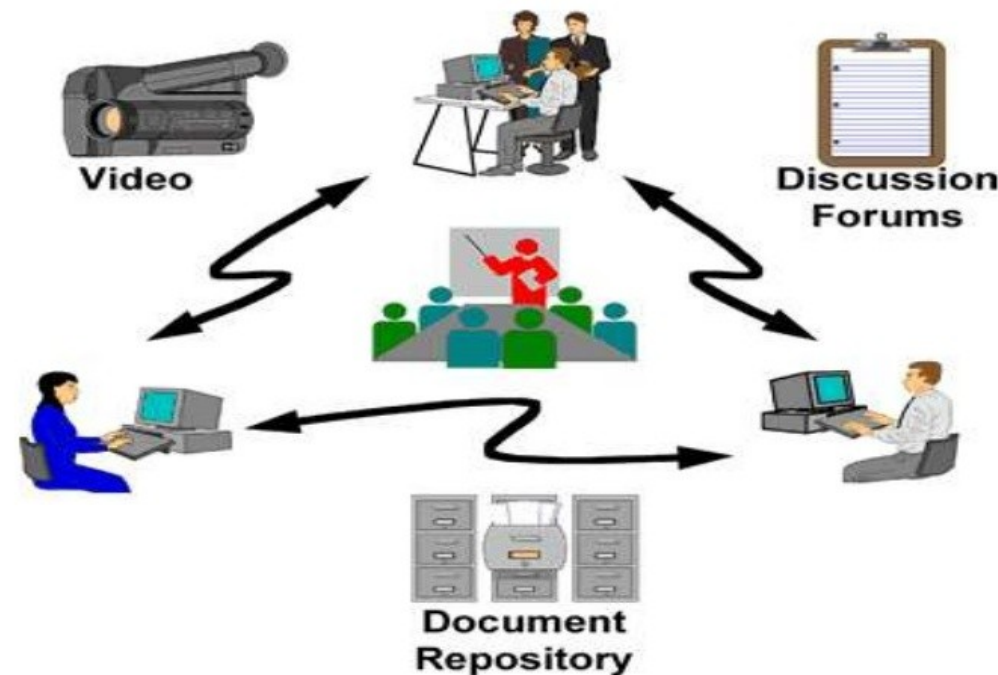
- HTTPS son las siglas de Hypertext Transfer Protocol Secure (protocolo seguro de transferencia de hipertexto).
- HTTPS es una versión del protocolo de transferencia que utiliza un cifrado seguro para la comunicación.
- En la siguiente tabla se detallan las diferencias entre HTTP y HTTPS:

	HTTP	HTTPS
Transmisión	Sin cifrar	Cifrada
Certificado	No	Sí
Número de puerto	80	443
Direccionamiento en el URL	http://	https://

# Plataformas de desarrollo colaborativo

# Desarrollo o Software colaborativo

- Software colaborativo o Group Ware se denomina al conjunto de programas o aplicaciones informáticas que integran el trabajo en un sólo proyecto con muchos usuarios presentes que se encuentran en diversas estaciones de trabajo, conectadas a través de una red (Internet o intranet).





# Características

- Desarrollo descentralizado y distribuido.
- Uso de diversas herramientas de comunicación asíncrona.
- Asignación de roles definidos.
- Múltiples colaboradores con competencias diversas.
- Generalmente el desarrollo es voluntario, no remunerado.
- Liberación rápida y frecuente.
- Aplicación del Modelo del Bazar.

# Categorías

- El software colaborativo se puede dividir en tres categorías:
  - **Herramientas de colaboración/comunicación:** Colaboración asíncrona como por ejemplo:
    - Correo electrónico.
    - Correo de voz.
    - Publicación en Web.

# Categorías

- **Herramientas de conferencia:** Colaboración sincrónica, ejemplo:
  - Conferencia de datos: Computadoras en red que comparten un espacio de presentación donde cada usuario puede modificar.
  - Conferencias de voz: Teléfonos o dispositivos que permiten interactuar a los participantes.
  - Conferencias de video (o audio conferencia): Dispositivos en red que comparten señales de audio o video.
  - Salas de Chat o mensajería instantánea: Una plataforma de discusión que facilita el intercambio inmediato de mensajes.
  - Sistemas para facilitar reuniones: Un sistema de conferencias integrado en una sala.

# Categorías

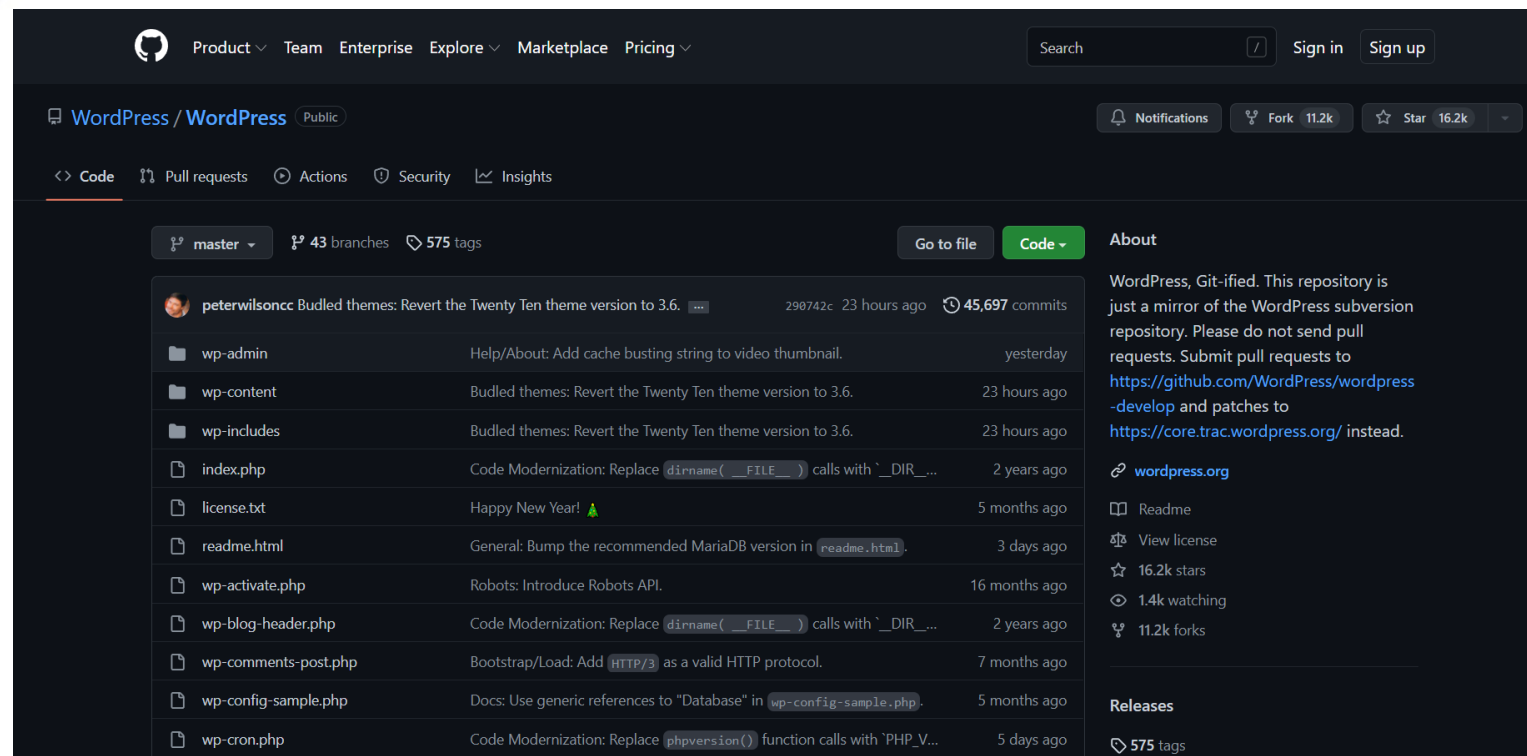
- **Herramientas de gestión colaborativa:** Facilitan las actividades del grupo, como por ejemplo:
  - Calendarios electrónicos: Para convenir fechas de eventos automáticamente y enviar notificaciones y recordatorios a los participantes.
  - Sistemas de gestión de proyectos: Para organizar y hacer seguimiento de las acciones y tareas en un proyecto desde que inicia hasta que se finaliza.
  - Sistemas de control de flujo de actividad: Para gestionar tareas y documentos en un proceso organizado de forma estructurada.
  - Sistemas de gestión del conocimiento: Para recoger, organizar, gestionar y compartir varios tipos de información.
  - Sistemas de soporte a redes sociales: Para organizar las relaciones de colectivos.

# GitHub

- GitHub es una de las plataformas de colaboración y control de versiones basada en web para desarrolladores de software más populares.
- GitHub utiliza Git para almacenar el código fuente de un proyecto y rastrear el historial completo de todos los cambios realizados en el código.
- Se puede pensar en GitHub como un sitio de redes sociales serio para desarrolladores de software. Los miembros pueden seguirse unos a otros, calificar el trabajo de los demás, recibir actualizaciones para proyectos específicos y comunicarse de forma pública o privada.

# GitHub

- Github permite que los desarrolladores coloquen proyectos creando repositorios de forma gratuita.
- Los repositorios pueden tener varios colaboradores y pueden ser públicos o privados.





# Gestión de repositorios en GitHub

1. Crear cuenta de usuario: Proporciona una dirección de correo electrónico, una contraseña y un usuario.
2. Responde la encuesta.

```
Welcome to GitHub!
Let's begin the adventure

Enter your email
✓ prueba@correo.com

Create a password
✓ .....

Enter a username
✓ prueba12343

Would you like to receive product updates and announcements via
email?
Type "y" for yes or "n" for no
→ no
```

Continue

# Gestión de repositorios en GitHub



3. Verifica tu cuenta.
4. Una vez verificada te debe llegar un correo con un código de verificación. Ingresa y accede.

Verify your account


Verificación

Solucione este rompecabezas para que sepamos que es una persona real

Verificar

You're almost done!

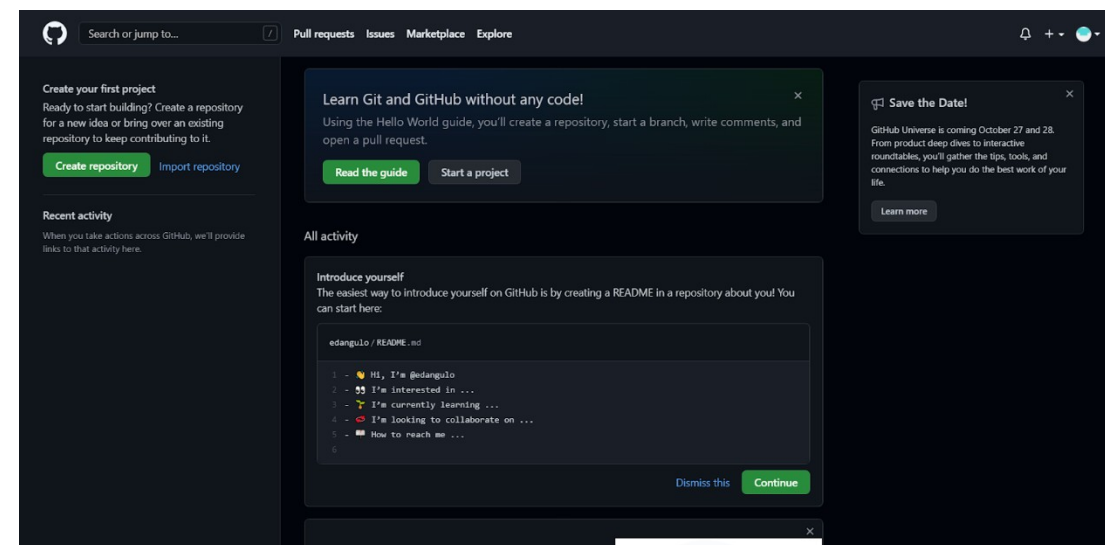
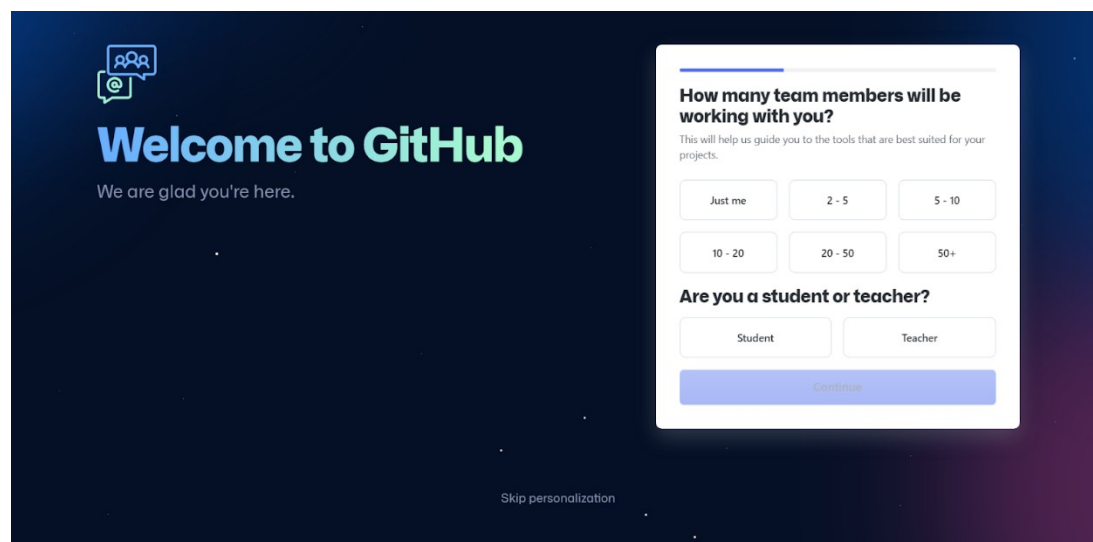
We sent a launch code to @uninorte.edu.co

→ Enter code

Didn't get your email? [Resend the code](#) or [update your email address](#).

# Gestión de repositorios en GitHub

5. Aparecerá una encuesta opcional para conocer un poco más sobre el uso que le darás a la herramienta. Puedes escoger no hacerla.
6. Accede y comienza a añadir tus proyectos.





# Gestión de repositorios en GitHub

7. Crea un nuevo repositorio. Indicar el nombre y si será público o privado.


Create a new repository


A repository contains all project files, including the revision history. Already have a project repository elsewhere?  
[Import a repository.](#)

Owner \* / Repository name \*  
  / prueba1 

Great repository names are short and memorable. Need inspiration? How about [fantastic-couscous](#)?

Description (optional)

☒  **Public**  
Anyone on the internet can see this repository. You choose who can commit.

☐  **Private**  
You choose who can see and commit to this repository.


Initialize this repository with:  
Skip this step if you're importing an existing repository.

☒ **Add a README file**  
This is where you can write a long description for your project. [Learn more.](#)

☒ **Add .gitignore**  
Choose which files not to track from a list of templates. [Learn more.](#)

☒ **Choose a license**

Quick setup — if you've done this kind of thing before

 Set up in Desktop or ☐ HTTPS ☐ SSH [https://github.com/\[username\]/prueba1.git](https://github.com/[username]/prueba1.git)

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

...or create a new repository on the command line

```
echo "# prueba1" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/[username]/prueba1.git
git push -u origin main
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/[username]/prueba1.git
git branch -M main
git push -u origin main
```

...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

[Import code](#)

# Comandos de utilidad en Git



## Git 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

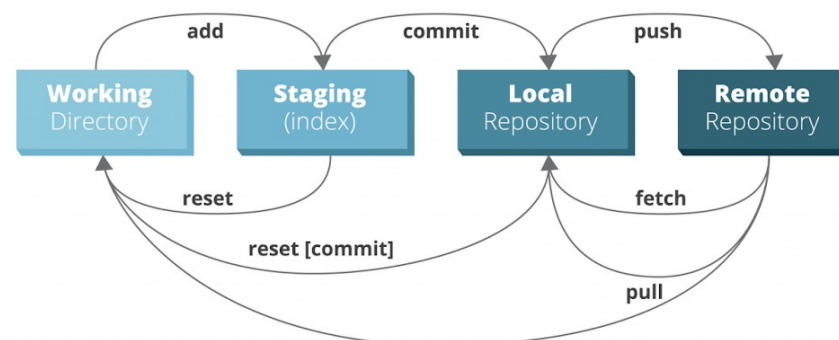
```
$ git push
```

### Finally!

When in doubt, use git help

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

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





# Otras alternativas a GitHub

•

Gitlab



GitLab



SOURCEFORGE

Bitbucket

•



Bitbucket

SourceForge

•

GitBucket





# Ejercicios de Práctica



El futuro digital  
es de todos

MinTIC

Misión  
TIC 2022

UN UNIVERSIDAD  
DEL NORTE

Vigilada Mineducación

**¡GRACIAS**

**POR SER PARTE DE  
ESTA EXPERIENCIA  
DE APRENDIZAJE!**

**Hechos**

QUE

**CONECTAN** ✓

