



# Sistema de Control de Versiones GIT



[www.sena.edu.co](http://www.sena.edu.co)

## GitHub como alojamiento de repositorios Git

# Hay dos caminos al iniciar

Crear el repositorio VACÍO  
en la nube y conectarlo al ya  
existente en local.

Crear el repositorio con al  
menos un archivo en la nube y  
CLONAR en local

# Clonar repositorios Git



Clonar un repositorio en Git significa copiar un repositorio existente de un servidor Git (como GitHub) a tu máquina local. Para hacer esto, sigue estos pasos:

- **Obtén la URL del Repositorio:**

En el sitio web del servicio donde se aloja el repositorio (por ejemplo, GitHub), encuentra la URL del repositorio. Puedes copiar la URL desde la interfaz del repositorio.

- **Abre la Terminal o el Símbolo del Sistema:**

Abre la terminal o el símbolo del sistema en tu máquina local.

- **Navega al Directorio en el que Quieres Clonar el Repositorio:**

Utiliza el comando `cd` para cambiar al directorio en el que deseas clonar el repositorio.

Por ejemplo:

**`cd ruta/del/directorio`**

# Clonar repositorios Git



Clonar un repositorio en Git significa copiar un repositorio existente de un servidor Git (como GitHub) a tu máquina local. Para hacer esto, sigue estos pasos:

- **Clona el Repositorio:**

```
git clone https://github.com/usuario/nombre-del-repositorio.git
```

.

- **Ingresa tus Credenciales (si es necesario):**

Si el repositorio está configurado para autenticación, se te pedirá que ingreses tus credenciales (nombre de usuario y contraseña para HTTPS o clave privada para SSH).

Ahora puedes trabajar en el repositorio, realizar cambios y contribuir a él. La información del repositorio remoto también estará configurada automáticamente como **origin**, por lo que podrás realizar operaciones como **git pull** y **git push** fácilmente.

# Para actualizar el repositorio que fue clonado y que actualizaron



Navegar al Directorio del Repositorio  
**cd nombre-del-repositorio**

Actualizar desde el Repositorio Remoto  
**git pull origin main**

**CREACIÓN DE RAMAS...**

# ¿Qué es una rama?

Una rama es una versión particular del código.  
Cada miembro del equipo creará su rama a partir de una rama común.

La rama común suele ser master. En la rama máster **debe estar el código completamente funcional.**

# Branch Locales (Ramas)

- Las ramas son utilizadas para desarrollar funcionalidades aisladas unas de otras. La rama master es la rama "por defecto" cuando creas un repositorio. Crea nuevas ramas durante el desarrollo y fusi3nalas a la rama principal cuando termines.





# Branch Locales (Ramas)

- Fáciles de crear y borrar
- No tienen por qué ser públicos
- Útiles para organizar el trabajo y los experimentos



# Características de las ramas

- Todo es local
- Operaciones más rápidas
- Puedes trabajar sin red
- Todos los repositorios de los desarrolladores son iguales
- En caso de emergencia puede servir de backup

# ¿Cómo se crea una rama?

git checkout **-b** *nombreRama*

→ *para crear la rama*

git push -u origin *nombreRama*

→ *para subir la rama!*

git checkout -b nombreRama1  
git push -u origin nombreRama1

git checkout -b nombreRama2  
git push -u origin nombreRama2



# Trabajo Colaborativo

# Configuración Inicial



1. El propietario del proyecto debe crear o tener un repositorio en la plataforma de Git
2. El propietario del repositorio añade los miembros del proyecto

The screenshot shows the GitHub repository settings page. The top navigation bar includes links for Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. The 'Settings' link is highlighted with a red box. On the left sidebar, the 'Collaborators' tab is selected and highlighted with a red box. The main content area is titled 'Who has access' and shows two sections: 'PUBLIC REPOSITORY' and 'DIRECT ACCESS'. The 'DIRECT ACCESS' section indicates that 1 collaborator has access. Below this, the 'Manage access' section is visible, featuring a search bar and a list of collaborators. The 'Add people' button is highlighted with a red box. The collaborator list shows a user named 'omargutierrez' with the role of 'Collaborator' and a 'Remove' button.

<> Code Issues Pull requests Actions Projects Wiki Security Insights **Settings**

General

Access

**Collaborators**

Moderation options

Code and automation

Branches

Tags

Rules

Actions

Webhooks

Environments

Codespaces

Pages

Security

### Who has access

**PUBLIC REPOSITORY**

This repository is public and visible to anyone.

[Manage](#)

**DIRECT ACCESS**

1 has access to this repository. [1 collaborator](#).

### Manage access

☐ Select all Type ▾


Find a collaborator...


☐ [omargutierrez](#)  
ingeomar2021 • Collaborator Remove

**Add people**

# Añadiendo Miembros al repositorio

3. Se agregan los colaboradores por username o por email

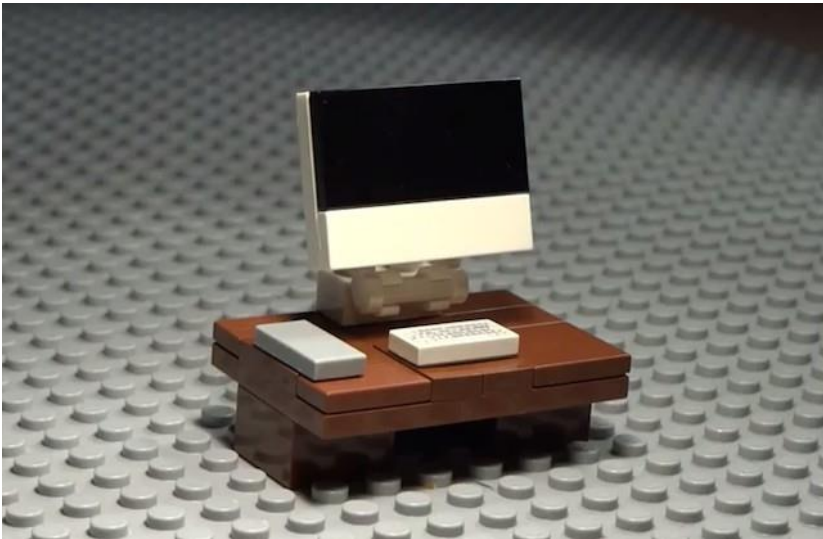




Add a collaborator to **clonado**

Select a collaborator above

# Una vez creado el repositorio, ¡el equipo se une!



Todos los miembros del equipo **ACEPTAN** la invitación que llega a su correo.

# Ahora todos hacemos GIT CLONE

Con la instrucción *git clone* cada miembro del equipo descarga el código **inicial**.

```
cd directorio_proyecto  
git clone url_repositorio
```



# Flujo de Trabajo Colaborativo:

## 1. Crea una Rama para tu Tarea:

```
bash
```

[Copy code](#)

```
git checkout -b nombre_de_tu_rama
```

## 2. Realiza tus Cambios y Confirma:

```
bash
```

[Copy code](#)

```
git add .  
git commit -m "Descripción del cambio"
```

## 3. Sincroniza con el Repositorio Remoto:

```
bash
```

[Copy code](#)

```
git pull origin main
```

## 4. Sube tus Cambios al Repositorio Remoto:

```
bash
```

[Copy code](#)

```
git push origin nombre_de_tu_rama
```

# Cada desarrollador empieza a modificar su código

RAMA Integrante1

```
//  
*  
* Author: Eric Marten Channel Controller  
* Blog: www.developerschannel.com  
* Email: eric@channel.com  
*  
#OverrideName = "LogonController",  
urlPatterns = {"/logon/*"}  
public class LogonController extends HttpServlet {  
  
    @Override  
    protected void service(HttpServletRequest request, HttpServletResponse response)  
    throws ServletException, IOException {  
        switch(request.getMethod()) {  
            case "POST":  
                login(request, response);  
                break;  
            case "GET":  
                logout(request, response);  
                break;  
        }  
    }  
}
```

RAMA Integrante 2

```
//  
*  
* Author: Eric Marten Channel Controller  
* Blog: www.developerschannel.com  
* Email: eric@channel.com  
*  
#OverrideName = "LogonController",  
urlPatterns = {"/logon/*"}  
public class LogonController extends HttpServlet {  
  
    @Override  
    protected void service(HttpServletRequest request, HttpServletResponse response)  
    throws ServletException, IOException {  
        switch(request.getMethod()) {  
            case "POST":  
                login(request, response);  
                break;  
            case "GET":  
                logout(request, response);  
                break;  
        }  
    }  
}
```

RAMA Integrante N

```
//  
*  
* Author: Eric Marten Channel Controller  
* Blog: www.developerschannel.com  
* Email: eric@channel.com  
*  
#OverrideName = "LogonController",  
urlPatterns = {"/logon/*"}  
public class LogonController extends HttpServlet {  
  
    @Override  
    protected void service(HttpServletRequest request, HttpServletResponse response)  
    throws ServletException, IOException {  
        switch(request.getMethod()) {  
            case "POST":  
                login(request, response);  
                break;  
            case "GET":  
                logout(request, response);  
                break;  
        }  
    }  
}
```

# Integrar el código

`git checkout main`  
`git pull origin main`  
`git merge mi_rama`  
`git push`

Sitúo mi código en la rama destino

- Actualizo mi local
- Unifico código

```
/**
 * Author: Xiao Santos Daniel Castillo
 * Email: xiaodanielcastillo@gmail.com
 */
package com.example.myapplication;

import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

    // TODO: Replace this with your own logic
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button btn = findViewById(R.id.button);
        btn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Toast.makeText(MainActivity.this, "Hello World!", Toast.LENGTH_SHORT).show();
            }
        });
    }
}
```



# Resolvemos conflictos si los hay.

Puede que dos personas hayan modificado la misma parte y para GIT no sea fácil identificar cuál es la versión indicada

```
/**
 *
 * @author Eric Santos Daniel Castillo
 * @url http://ericcastillo.com
 * @email eric@ericcastillo.com
 */
@ExceptionHandler({
    HttpServerException.class,
    HttpServerException.class
})
public class HttpServerExceptionHandler implements Handler {

    @Override
    public void handle(HttpServerException request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setStatus(HttpServletResponse.SC_INTERNAL_SERVER_ERROR);
        response.setContentType("text/html");
        response.setCharacterEncoding("UTF-8");
        response.getWriter().write("Error interno del servidor.");
        response.getWriter().flush();
        response.getWriter().close();
    }
}
```

# Resolvemos conflictos si los hay.

- [illegible]

```

/**
 * Author: Eric Marlow Daniel Castillo
 * Email: ericcastillo@protonmail.com
 * Email: gomezmd@gmail.com
 */

@WebServlet(name = "loginController", urlPatterns = {"/login.html"})
@WebServlet(urlPatterns = {"/loginController"}, httpServletMappings = {response})

@Override
protected void service(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    login(request,response);
}

private void login(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    // ...
}

```

# Resolvemos conflictos si los hay.

Ajusto el archivo con el bloque  
correcto, o la mezcla de los dos.

```
/**
 * Author: Eric Santos Daniel Castillo
 * Email: eric.santos@univalle.edu.co
 */
public class LogonController extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        // TODO: Implementar la lógica de negocio
        // ...
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        // TODO: Implementar la lógica de negocio
        // ...
    }
}
```

# Resuelto el conflicto, guardamos cambios y los subimos

```
git add -all
git commit -m "Cuento qué hice"
git push
```

- Se agregan cambios
- Se confirman los cambios
- Se sube a la nube

[illegible]

# Al finalizar el proceso

Al finalizar el proceso, **todo el equipo** hará  
**git pull**  
Y tendremos el código actualizado



# Ignorando archivos en git

## Ignorando archivos en git

- El archivo `.gitignore` es un archivo de configuración en Git que especifica patrones de archivos y directorios que Git debe ignorar al realizar operaciones como `git add` y `git commit`. Estos patrones se utilizan para excluir archivos o directorios específicos del seguimiento de versiones de Git.
- Cuando trabajas en un proyecto, es común que haya archivos o directorios que no quieres que se rastreen en el repositorio, ya sea porque son generados automáticamente, son archivos temporales, contienen información sensible o por otras razones. Es en este escenario donde el archivo `.gitignore` resulta útil.



**G R A C I A S**

Línea de atención al ciudadano: 01 8000 910270  
Línea de atención al empresario: 01 8000 910682



[www.sena.edu.co](http://www.sena.edu.co)