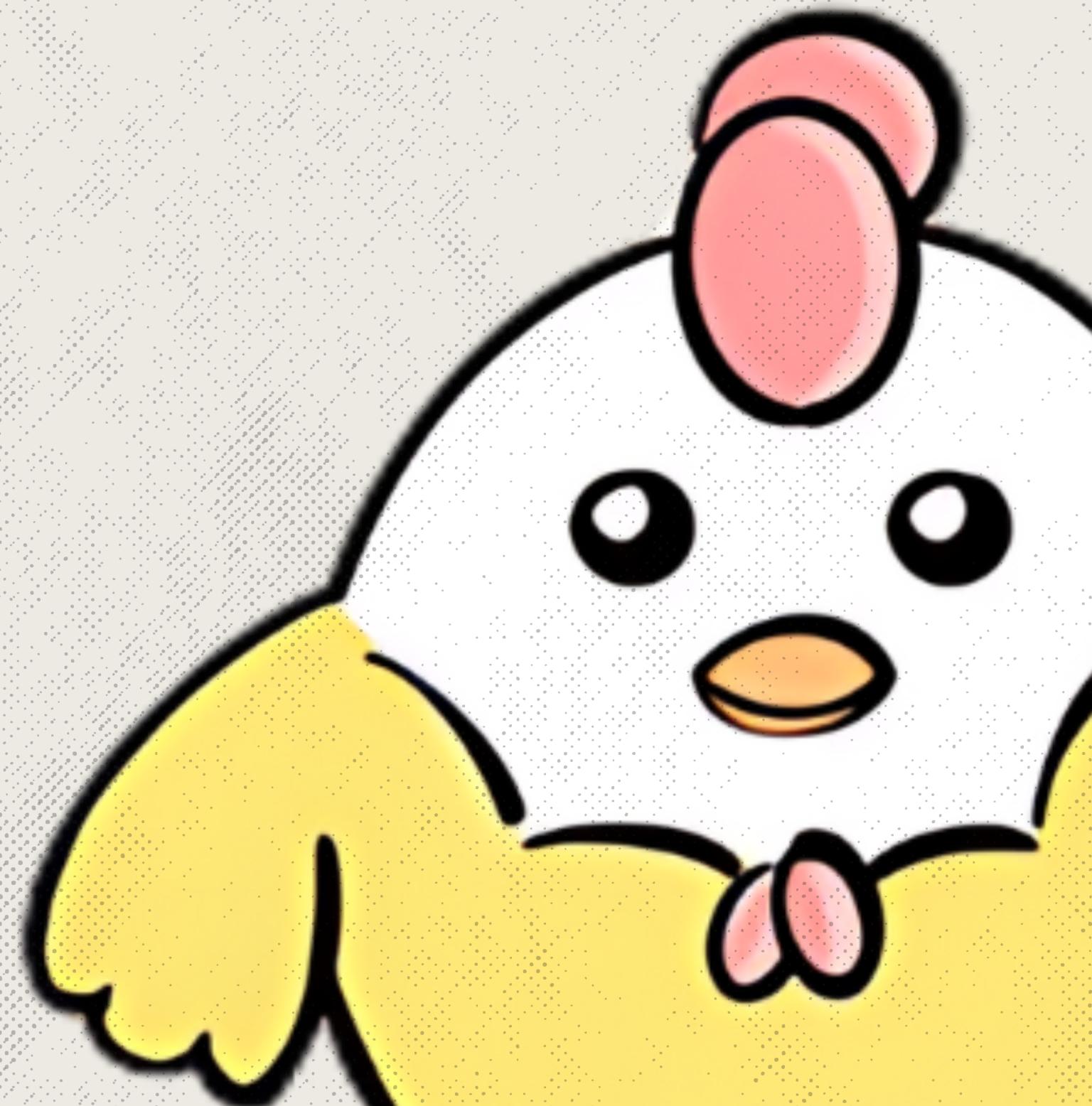


# Git y GitHub para Principiantes: Controla Tu Código Como un Profesional

**Camila Valenzuela Fierro**  
**Desarrolladora Web**



# OBJETIVOS

Aprenderemos sobre:

- Control de versiones.
- Diferencias entre Git y GitHub.
- Configuración básica y comandos esenciales.
- Colaboración en equipo con GitHub.



## ¿Qué es Git?

- Git es un sistema de control de versiones distribuido creado por Linus Torvalds en 2005, diseñado para gestionar proyectos de software con una gran cantidad de cambios.
- Control de versiones: El control de versiones permite a los desarrolladores guardar el historial de cambios en sus proyectos, revertir errores y colaborar sin conflictos.
- Distribuido vs. Centralizado: A diferencia de otros sistemas centralizados, cada clon de un repositorio Git contiene toda la historia del proyecto, lo que permite trabajar sin conexión y tener múltiples copias de seguridad.

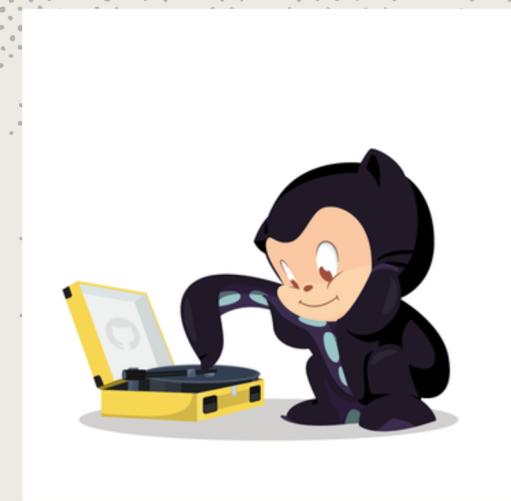
# GITHUB

## ¿Qué es GitHub?

- Es una plataforma en la nube para alojar proyectos Git. Facilita la colaboración entre equipos y el uso de herramientas de integración continua, manejo de issues, pull requests, etc.
- Comparte y revisa código mediante pull requests.
- Alternativas: GitLab, Bitbucket.

## Diferencia entre Git y GitHub:

- Git: Herramienta local para gestionar versiones de código.
- GitHub: Plataforma online que permite colaborar y compartir proyectos usando Git.
- Git sin GitHub: Un desarrollador puede tener su proyecto bajo control de versiones con Git en su computadora sin necesidad de subirlo a la nube.
- Git + GitHub: De esta manera, un equipo puede crear un repositorio en GitHub y trabajar colaborativamente, haciendo que GitHub sea un espacio centralizado donde todos pueden compartir y fusionar su trabajo.



# CONFIGURACIÓN DEL AMBIENTE DE TRABAJO

## macOS:

- Git generalmente viene preinstalado. Para verificar si está instalado, solo necesitas abrir la Terminal y escribir:

```
bash                               Copiar código  
git --version
```

Si no está instalado, puedes instalar Git rápidamente usando Homebrew (un gestor de paquetes para macOS):

```
bash                               Copiar código  
brew install git
```

## Windows

- En Windows, Git no viene preinstalado, por lo que se necesita instalar Git for Windows (incluye Git Bash, una terminal similar a la de Unix):
  - Descarga desde [git-scm.com](https://git-scm.com).
  - Durante la instalación, hay varias configuraciones importantes como seleccionar el editor por defecto y el estilo de línea de comandos (se recomienda usar Git Bash para tener una experiencia más similar a Unix).

# COMANDOS BÁSICOS DE GIT

## Configuración Inicial

- Al configurar el nombre y correo electrónico con **git config**, estás asociando tus cambios con tu identidad, algo importante para colaborar y atribuir trabajo correctamente.
- **git config --global user.name "Tu Nombre"**
- **git config --global user.email "tuemail@example.com"**
- Para revisar la configuración actual y verificar que todo esté correcto.
- **git config --list**

# INICIALIZAR UN REPOSITORIO

- Esto inicia un nuevo historial de cambios y crea una carpeta oculta **.git**
- Puedes hacer la prueba así:
- Crear un archivo (index.html, por ejemplo).
- Hacer cambios en ese archivo y ver cómo Git rastrea esos cambios con **git status**
- **git init**



# CONTROLANDO CAMBIOS

- **git status**

- Muestra qué archivos han cambiado, cuáles están preparados para ser "commiteados" (staged) y cuáles no. Este comando es fundamental para revisar el estado del repositorio antes de hacer commits.

- **git add**

- Mueve los archivos al área de preparación (staging area).
  - Ejemplos concretos:
- **git add archivo.txt** Añadir un solo archivo.
- **git add .** Añadir todos los archivos modificados.



- **git commit**

- Importancia de los mensajes descriptivos y cómo cada commit debería ser un "punto de restauración" lógico y significativo.
  - Ejemplo: **git commit -m "Añadir la estructura HTML básica"**

# VISUALIZACIÓN DE CAMBIOS

- Visualización de Historial (git log):
  - **git log** muestra todos los commits en orden cronológico
  - **git log --oneline** para visualizar un resumen breve.
  - **commit hashes:** identifican de forma única cada cambio.

```
$ git log --format=medium
commit 5bf53efc38e1d780b32daf7c1d9acb54a7e93936
Author: Will Anderson <will@itsananderson.com>
Date:   Mon Jul 21 08:27:50 2014 -0700

    Normalize indents to 4 spaces for all source files

commit bb82a7ab75f14cce5d8f8a60e97bfa8a2ceabbe
Author: Will Anderson <will@itsananderson.com>
Date:   Mon Jul 21 01:11:38 2014 -0700

    Bump version to 0.2.4

commit d9bb2ff54751797208c0e551004cab1c1adb01cc
Author: Will Anderson <will@itsananderson.com>
Date:   Mon Jul 21 01:11:18 2014 -0700

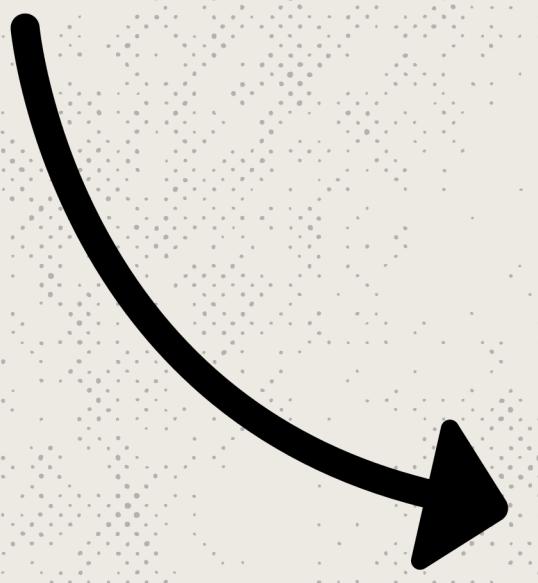
    Update .npmignore with new dev files

commit c75067e0f76c0e0adad58bb1ac663834f572f696
Author: Will Anderson <will@itsananderson.com>
Date:   Sun Jul 20 23:00:44 2014 -0700

    Add full coverage for application mixin

commit 98ee9cbbb4454912d352b8bdb6a3dcbcd64a38f9
Author: Will Anderson <will@itsananderson.com>
Date:   Sun Jul 20 14:34:18 2014 -0700

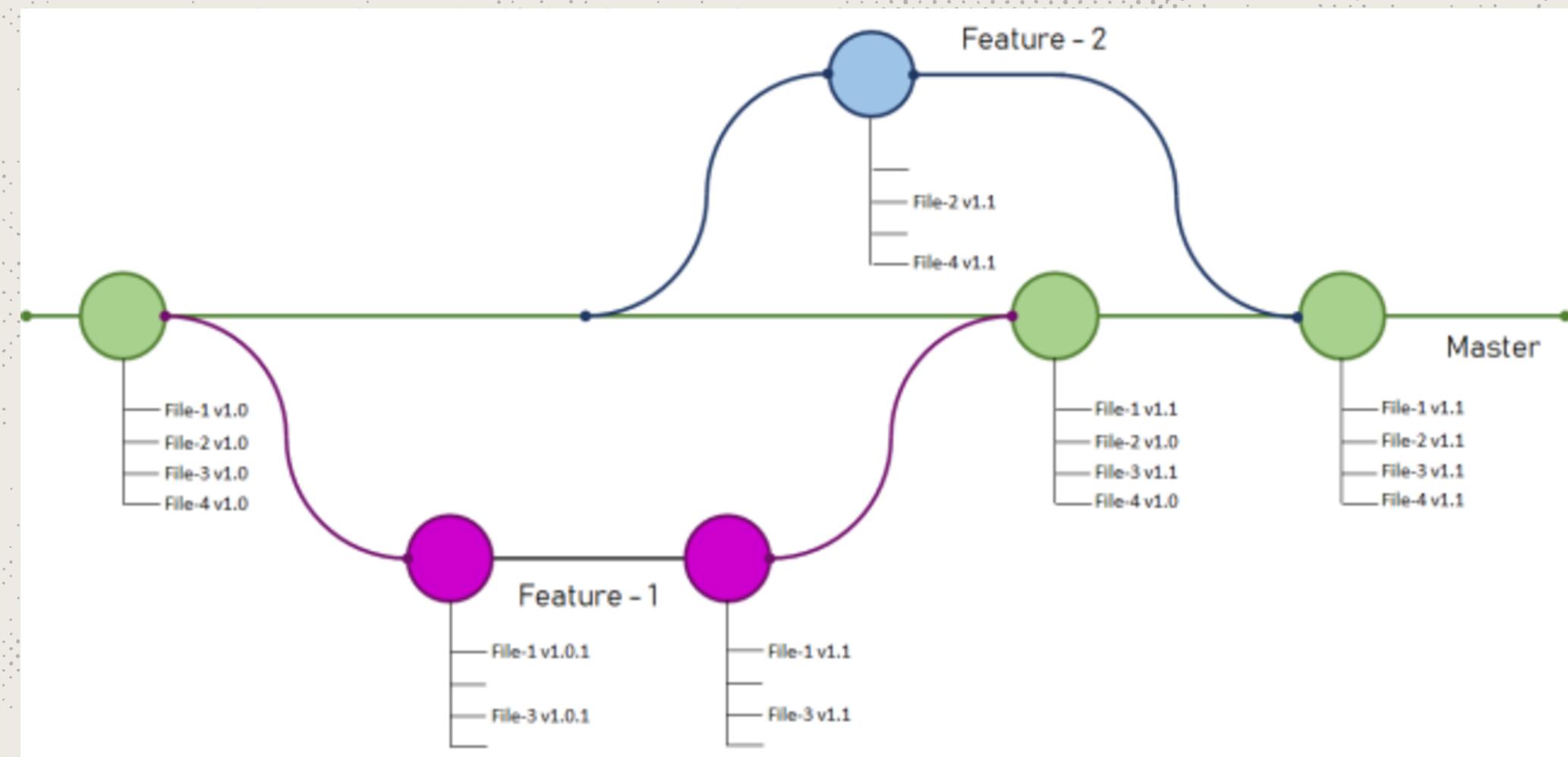
    Add full coverage for routeParams utility
```



```
[Divya1@Divya:myProj [feature1] $ git log --pretty=oneline
006b9ce5a89e4bb4e6c5817d5ee67ed50b71e360 (HEAD -> feature1) DevOps structure defined
b3378b219c865e25a5b00428d48c289d3e50fc1a (origin/feature1) small edit in lib welcome.sh
f405de04fdc73715a18ab208315fcfafdbdae0408 modified the last edit date
578de10c8aa0c94426c54764e0d7a0e0a26e7b13 Editing the common lib file
```

# TRABAJO CON BRANCHES (RAMAS)

- Trabajando con Ramas:
  - **git branch** y **git checkout -b** Versiones paralelas del código donde puedes desarrollar nuevas características sin afectar la rama principal (main o master).



# DESHACIENDO CAMBIOS

- Deshaciendo Cambios:
  - git checkout, git reset y git revert: Cada uno se usa para diferentes tipos de deshacer:
    - **git checkout**
      - Recuperar un archivo o moverse a otra rama.
    - **git reset**
      - Retroceder cambios locales.
    - **git revert**
      - Crear un nuevo commit que deshace los cambios de uno anterior, ideal para deshacer errores de forma segura.

# WORKFLOW EN GITHUB

## Objetivos Detallados

Conectar un repositorio local con GitHub.

Entender el proceso de enviar y recibir cambios.

Familiarizarse con el flujo de trabajo colaborativo en GitHub.

- Crear un Repositorio en GitHub:
  - Paso a paso, desde la creación del repositorio en GitHub hasta cómo copiar el URL para conectarlo con el repositorio local.
- Conectar el Repositorio Local con GitHub (git remote):
  - `git remote add origin https://github.com/usuario/repo.git`: Este comando establece una conexión entre el repositorio local y el remoto.
  - Verificar remotos: `git remote -v` para asegurarse de que la conexión esté correctamente configurada.
- Subir Cambios a GitHub (git push):
  - `git push -u origin main` sube los cambios a GitHub. `-u` establece la rama por defecto, simplificando futuros pushes.
- Recibir Cambios de GitHub (git pull):
  - Cómo sincronizar el repositorio local con cambios remotos usando `git pull`.
  - Resolución de Conflictos: Cómo manejar conflictos de fusión y la importancia de comunicarse en equipo.

# RECURSOS

- Documentación Git: <https://git-scm.com/>
- Documentación Github: <https://docs.github.com/es>
- Tutorial Atlassian: <https://www.atlassian.com/git/tutorials>

# EMPOLIAMOS TUS IDEAS

