BOOTCAMP

Generación T

streambe

¿Qué problema soluciona Git?

Antes de Git, los desarrolladores enfrentaban desafíos significativos:

- Pérdida de código por sobrescritura accidental
- Dificultad para identificar quién hizo qué cambio
- Conflictos constantes al trabajar en equipo
- Imposibilidad de experimentar sin comprometer el proyecto principal

Git resuelve estos problemas mediante un sistema que registra meticulosamente cada cambio, permitiendo volver a versiones anteriores cuando sea necesario y facilitando el trabajo colaborativo sin interferencias.

Versionado y trabajo en equipo

Historial transparente

Cada modificación queda documentada con información del autor, fecha y descripción del cambio, creando un registro completo del desarrollo del proyecto.

Trabajo paralelo

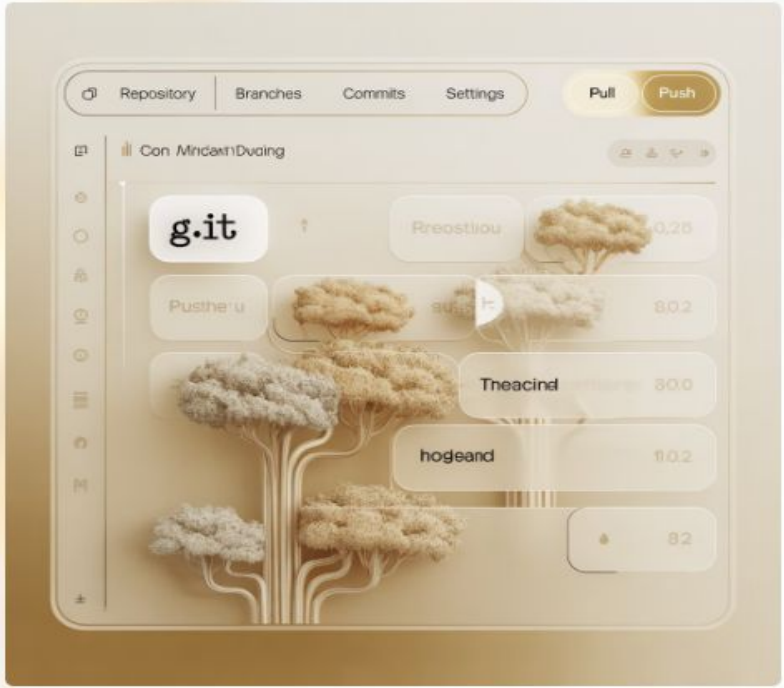
Múltiples desarrolladores pueden trabajar simultáneamente en distintas partes del código sin interferir entre sí, aumentando la productividad del equipo.

Experimentación segura

Las ramas (branches) permiten probar nuevas características o correcciones en un entorno aislado, sin afectar al código principal hasta que estén completamente probadas.

Esta combinación de características hace que Git sea indispensable para equipos modernos de desarrollo de software.

¿Qué es un repositorio Git?



Un repositorio Git contiene todos los archivos del proyecto junto con su historial completo de cambios, almacenado en una estructura especial denominada **.git**.

Repositorio local

Reside en tu computadora personal, permitiéndote trabajar sin conexión a internet y realizar cambios privados antes de compartirlos.

Repositorio remoto

Alojado en servidores como GitHub, GitLab o Bitbucket, facilita la colaboración y sirve como copia de seguridad centralizada accesible para todo el equipo.

Conceptos clave: commit, branch y más

Commit

Una instantánea permanente de los cambios realizados en el repositorio, incluyendo un mensaje descriptivo, identificador único (hash), autor y fecha. Es la unidad básica del historial de Git.

Branch (rama)

Una línea de desarrollo independiente que permite trabajar en características o correcciones sin afectar la rama principal. Facilita el desarrollo paralelo y la experimentación segura.

Merge y Pull Request

Procesos para integrar cambios entre ramas. El merge combina cambios localmente, mientras que un pull request es una solicitud formal para incorporar cambios en repositorios remotos, facilitando la revisión de código.

Flujo de trabajo básico en Git

Working Directory

Es tu directorio de trabajo donde modificas archivos. Git detecta estos cambios pero no los registra automáticamente hasta que tú lo indiques.

Staging Area

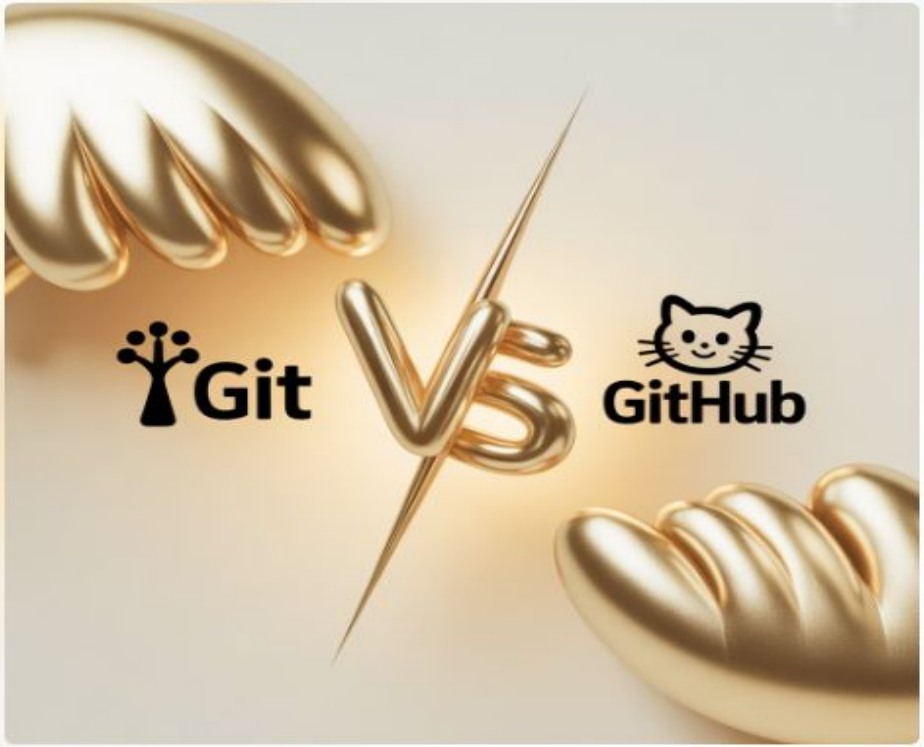
Área intermedia donde seleccionas específicamente qué cambios quieres incluir en tu próximo commit, permitiendo preparar cuidadosamente cada confirmación.

Repository

Una vez confirmados (commit), los cambios se almacenan permanentemente en el historial del repositorio, creando un punto de referencia al que siempre podrás volver.

Este flujo de trabajo de tres etapas es fundamental para entender cómo Git gestiona los cambios y proporciona control preciso sobre qué se registra en el historial.

Diferencias entre Git y GitHub



Git	GitHub
Software de control de versiones	Plataforma web para alojar repositorios Git
Funciona localmente (sin internet)	Requiere conexión a internet
Enfocado en gestión de versiones	Añade características sociales y colaborativas
Herramienta de línea de comandos	Interfaz web amigable
Creado por Linus Torvalds	Fundado por Tom Preston-Werner, adquirido por Microsoft

Alternativas a GitHub incluyen GitLab, Bitbucket y Azure DevOps, todas basadas en Git pero con características distintivas.

Conclusiones y ventajas de usar Git

Seguridad y confiabilidad

Cada commit crea un historial inmutable con checksums que garantizan la integridad del código. Nunca pierdes trabajo y siempre puedes retroceder a versiones anteriores.

Eficiencia en equipo

Facilita la colaboración entre desarrolladores en todo el mundo. Las ramas permiten trabajar en paralelo sin interferencias, aumentando la productividad general.

Estándar profesional

Dominar Git es una habilidad esencial para cualquier desarrollador moderno. Su adopción universal en la industria lo convierte en conocimiento imprescindible.

Aprender Git no solo mejora tu flujo de trabajo personal, sino que te prepara para integrarte efectivamente en equipos de desarrollo profesionales en cualquier empresa tecnológica.

Comandos Esenciales de Git

1 git init

Crea un nuevo repositorio Git en el directorio actual. Es el primer comando para iniciar un proyecto bajo control de versiones.

2 git status

Muestra el estado de tu directorio de trabajo y el "staging area". Indica qué archivos han sido modificados, cuáles están listos y cuáles no se están rastreando.

3 git add <archivo>

Prepara los cambios de un archivo específico (o de todos con `.`) para ser incluidos en el próximo commit. Mueve los cambios al "staging area".

4 git commit -m "mensaje"

Guarda permanentemente los cambios del "staging area" en el historial del repositorio. El mensaje es una descripción concisa de lo que se hizo.

Estos comandos forman la base de la interacción diaria con Git, permitiéndote rastrear y guardar el progreso de tu proyecto de forma efectiva.

Ramas y Trabajo Colaborativo en Git

1

¿Qué es una rama (branch)?

Una rama en Git es una línea independiente de desarrollo que diverge del flujo principal. Técnicamente, es un apuntador móvil que señala a un commit específico en el historial del repositorio.

2

Propósito de las ramas

Las ramas permiten:

- Desarrollar funcionalidades aisladas sin afectar el código principal
- Experimentar con cambios sin comprometer la estabilidad del proyecto
- Facilitar el trabajo en paralelo entre múltiples desarrolladores

3

Comandos principales

- **git branch**: Lista todas las ramas locales
- **git checkout -b nueva-rama**: Crea y cambia a una nueva rama
- **git merge**: Integra los cambios de una rama a otra

4

Conflictos de merge

Ocurren cuando Git no puede resolver automáticamente las diferencias entre dos ramas. Sucede cuando ambas ramas modifican la misma parte de un archivo de manera diferente. Requieren intervención manual para decidir qué cambios conservar.

Subir y Bajar Cambios con GitHub

1

Repositorio Remoto

Es una versión del proyecto hospedada en un servidor como GitHub, GitLab o Bitbucket, permitiendo la colaboración entre múltiples desarrolladores y sirviendo como respaldo centralizado.

2

Comandos Esenciales

- **git clone [URL]**: Crea una copia local completa del repositorio remoto
- **git remote add origin [URL]**: Vincula tu repositorio local con un repositorio remoto
- **git push**: Envía tus cambios locales al repositorio remoto
- **git pull**: Obtiene y fusiona cambios del repositorio remoto a tu copia local

3

Conflictos de Fusión

Cuando dos desarrolladores modifican la misma sección de código, Git no puede determinar automáticamente qué cambios conservar. En este caso, Git marca el conflicto en el archivo, requiriendo resolución manual para decidir qué cambios mantener.

La resolución de conflictos es una parte fundamental del trabajo colaborativo. Utiliza **git status** para identificar archivos conflictivos, edítalos manualmente para resolver las diferencias, y luego utiliza **git add** seguido de **git commit** para finalizar la resolución.

Manejo de errores y buenas prácticas en Git

Recuperación de archivos

Con **git checkout -- archivo.txt** puedes restaurar un archivo a su último estado confirmado, descartando cambios locales.

Mediante **git reset** puedes deshacer cambios usando diferentes modos:

- --soft: preserva cambios en área de preparación
- --mixed: preserva cambios en directorio de trabajo
- --hard: elimina todos los cambios no confirmados

El archivo .gitignore

Permite especificar archivos y directorios que Git debe ignorar:

- Archivos de configuración personal
- Archivos temporales y logs
- Información confidencial
- Directorios de dependencias (node_modules)

Historial con git log

Visualiza el historial completo de commits con **git log**, mostrando identificador SHA, autor, fecha y mensaje.

Opciones útiles: **--oneline** (vista compacta), **-p** (diferencias), **--stat** (estadísticas), **--graph** (visualización de ramas).

Buenas prácticas

Mensajes de commit descriptivos que expliquen *qué* cambió y *por qué*.

Utiliza ramas (branches) cortas por cada tarea y reintégralas frecuentemente.

Realiza commits pequeños y frecuentes para reducir conflictos.

Implementa revisiones de código y mantén comunicación constante con el equipo.

Introducción Express a React con Vite

streambe

GENERACIÓN T

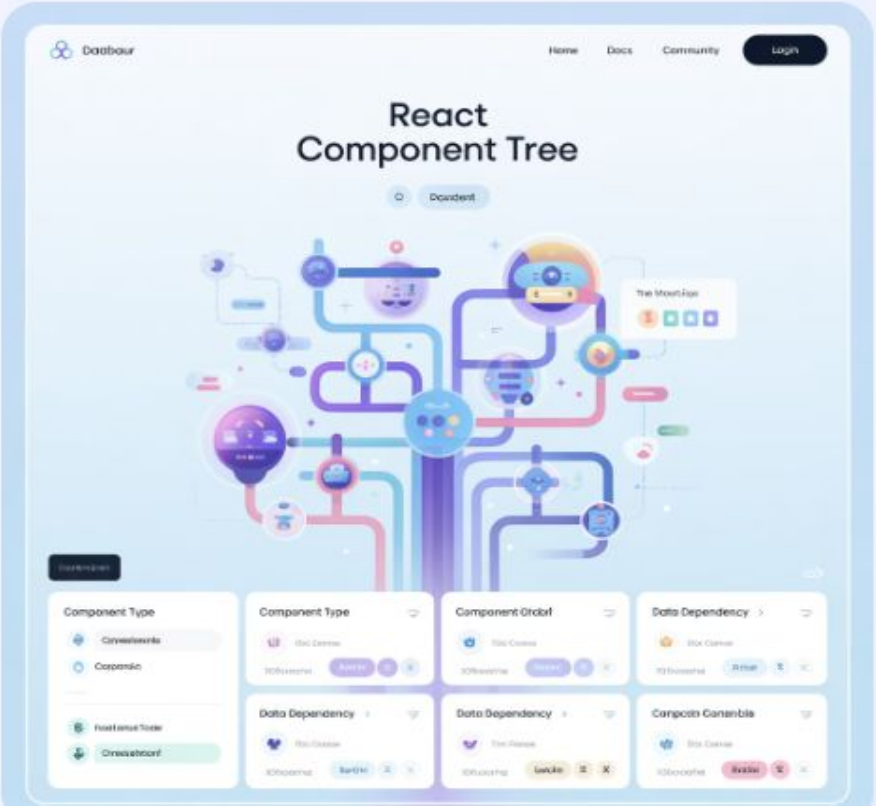
¿Qué es React?

React es una **biblioteca de JavaScript** desarrollada por Facebook que permite construir interfaces de usuario interactivas y reutilizables para aplicaciones web.

Sus principales características incluyen:

- Basada en componentes reutilizables
- Virtual DOM para renderizado eficiente
- Flujo de datos unidireccional
- Ecosistema extenso y comunidad activa

React no es un framework completo, sino una biblioteca centrada específicamente en la interfaz de usuario.



Componentes y Estado en React

Componentes

Son bloques de construcción independientes y reutilizables que encapsulan HTML, CSS y JavaScript.

Pueden ser funcionales (modernos) o de clase (tradicionales).

```
function Saludo(props) {
  return
```

Hola, {props.nombre}!

```
;}
```

Estado (State)

Datos privados controlados por el componente que pueden cambiar durante su ciclo de vida.

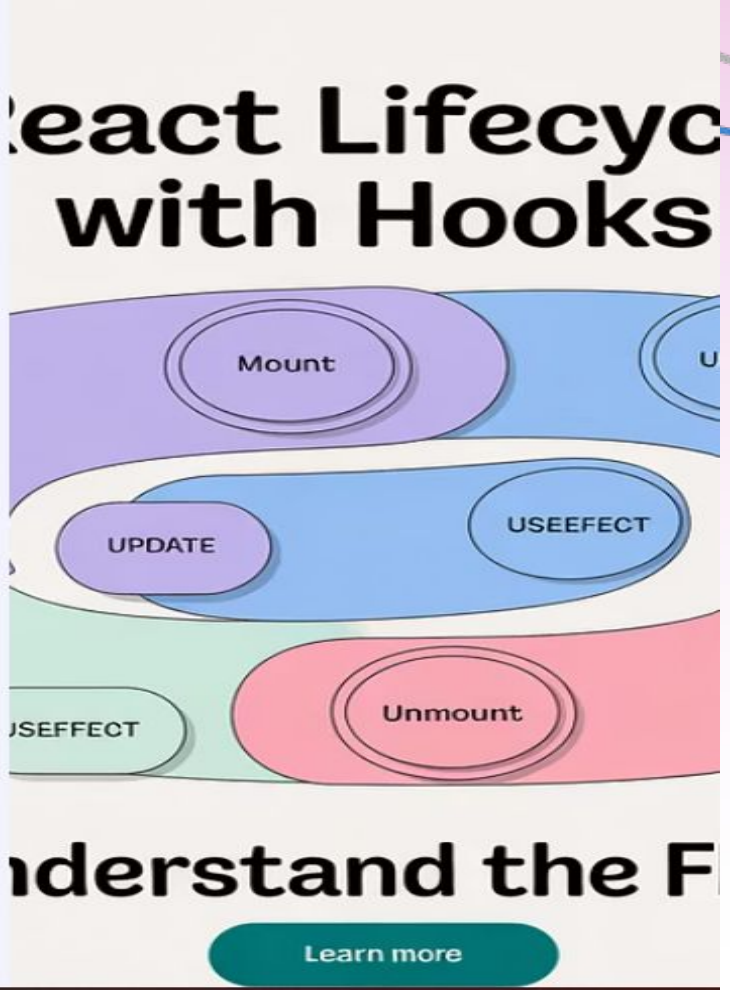
Cuando el estado cambia, React vuelve a renderizar el componente.

```
const [contador,
setContador] =
useState(0);
```

Props

Datos recibidos desde componentes padre que no pueden modificarse (inmutables).

Permiten la comunicación entre componentes.



¿Por qué Vite en lugar de Create React App?

Velocidad Superior

Arranca el servidor de desarrollo en milisegundos en vez de minutos.

Utiliza ESM nativo del navegador durante el desarrollo.

Ligereza

Menor tamaño de instalación y dependencias.

Configuración mínima y sensata por defecto.

HMR Optimizado

Actualización de módulos en caliente ultra rápida.

Conserva el estado durante las actualizaciones de código.

Create React App ha sido la herramienta estándar durante años, pero su lentitud y configuración rígida han llevado a los desarrolladores a buscar alternativas más modernas como Vite.

Creando tu Primer Proyecto React con Vite

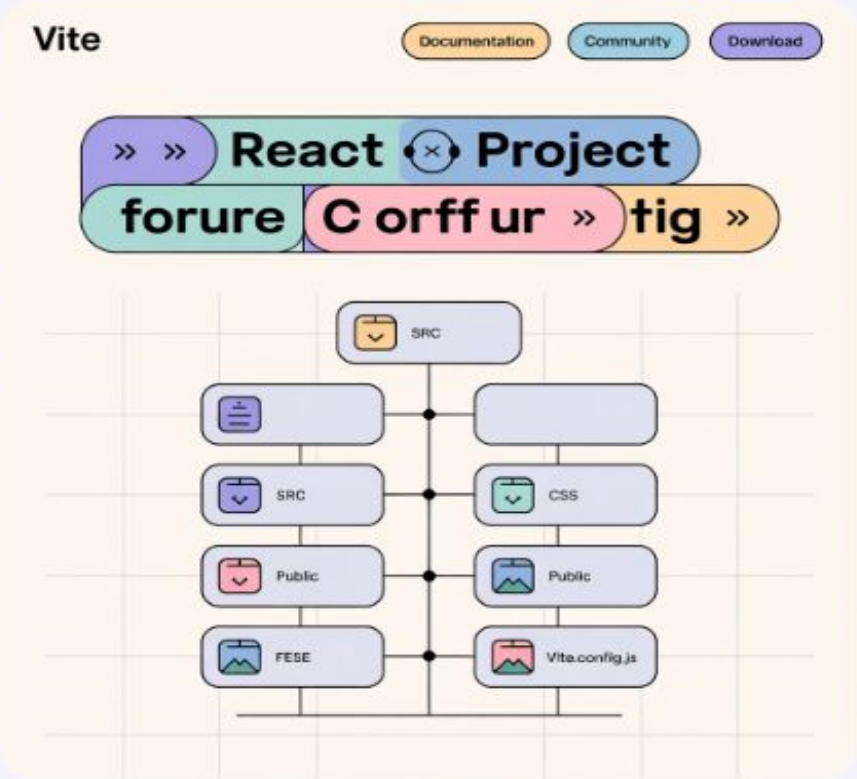
El proceso para crear un proyecto React con Vite es sorprendentemente sencillo y rápido, requiriendo solo unos pocos comandos en la terminal.

```
# Crear un nuevo proyecto
npm create vite@latest mi-app-react -- --template react

# Navegar al directorio del proyecto
cd mi-app-react

# Instalar dependencias
npm install

# Iniciar el servidor de desarrollo
npm run dev
```

Tras ejecutar estos comandos, tu aplicación estará disponible en **http://localhost:5173**. ¡Así de simple!

Vite configura automáticamente todo lo necesario para empezar a desarrollar con React, incluyendo JSX, ES6 y más.

Estructura de un Proyecto React con Vite



Archivos y Carpetas Principales

- **package.json**: Dependencias y scripts
- **vite.config.js**: Configuración de Vite
- **index.html**: Punto de entrada HTML
- **src/**: Código fuente de la aplicación
- **src/main.jsx**: Punto de entrada JavaScript
- **src/App.jsx**: Componente principal
- **public/**: Archivos estáticos

Esta estructura es limpia y permite organizar tu código de manera lógica desde el principio.

Creando tu Primer Componente React

1. Crear el archivo

Crea un nuevo archivo **MiComponente.jsx** dentro de la carpeta **src/components** (crea esta carpeta si no existe).

2. Escribir el código

```
import { useState } from
'react';
import './MiComponente.css';

function MiComponente() {
  const [contador,
setContador] = useState(0);

  return (
```

Mi Primer Componente

Has hecho clic {contador} veces

```
setContador(contador + 1)}>  ...

); }  export default MiComponente;
```

3. Importar y usar

Importa y usa tu componente en **App.jsx**:

```
import MiComponente from
'./components/MiComponente';

function App() {
  return (
```

Mi Aplicación React

```
); }
```

Próximos Pasos en tu Viaje con React

Dominar los Hooks

Aprende a usar useState, useEffect, useContext y otros hooks para manejar el estado y efectos secundarios.

Gestión de Estado

Explora soluciones como Context API, Redux o Zustand para manejar el estado global de tu aplicación.

Enrutamiento

Implementa React Router para crear aplicaciones de múltiples páginas con navegación fluida.

Integración con APIs

Aprende a conectar tu aplicación con servicios externos usando fetch, axios o React Query.

Recuerda que la mejor manera de aprender React es construyendo proyectos reales. ¡No tengas miedo de experimentar y cometer errores en el camino!

¿Alguna duda?

streambe GENERACIÓN T

Muchas gracias.

streambe GENERACIÓN T

Nuestras Redes

www.generaciont.org
www.streambe.com
www.instagram.com/generaciont_ar
www.tiktok.com/@generaciont
generaciont@generaciont.org
Cel: 11 61331747

GENERACIÓN T

streambe