Logotipo

Descripción generada automáticamente con confianza media

**UNIVERSIDAD AUTONOMA DEL BENI JOSE BALLIVIAN**

**FACULTAD INGENERIA Y TECNOLOGIA**

**CARRERA ING. DE SISTEMAS**

**Logotipo

Descripción generada automáticamente**

**ESTUDIANTE: UNIV. Juan Jose Garcia Caceres**

**UNIV. Dussan Claure Raduan**

**ASIGNATURA: Tecnologías emergentes**

**DOCENTE: Ing. Hermes Rodriguez**

**SEMESTRE: 7mo**

**TRINIDAD - BENI –BOLIVIA**

Índice

[Introducción 2](#_Toc196300004)

[Objetivos 2](#_Toc196300005)

[Relevancia del Tema 2](#_Toc196300006)

[Fundamentos Teóricos y Definiciones 2](#_Toc196300007)

[Sistema de Control de Versiones (VCS) 2](#_Toc196300008)

[Git 2](#_Toc196300009)

[Arquitectura de Git 3](#_Toc196300010)

[Modelo de Datos 3](#_Toc196300011)

[Grafo de Commits 3](#_Toc196300012)

[Protocolos de Transferencia 3](#_Toc196300013)

[Flujos de Trabajo (Workflows) 3](#_Toc196300014)

[Git Flow 3](#_Toc196300015)

[GitHub Flow 3](#_Toc196300016)

[Ejemplos 4](#_Toc196300017)

# Introducción

Los Sistemas de Control de Versiones (VCS) representan un pilar fundamental en el desarrollo de software moderno, proporcionando mecanismos robustos para gestionar cambios, facilitar la colaboración y garantizar la trazabilidad del código. Entre estas herramientas, Git ha emergido como estándar de facto, revolucionando la industria con su modelo distribuido, eficiencia en operaciones locales y flexibilidad para adaptarse a diversos flujos de trabajo.

# Objetivos

El presente informe tiene como objetivo analizar los Sistemas de Control de Versiones (VCS), con especial énfasis en Git, para:

* Comprender su origen y evolución, desde los primeros métodos manuales hasta las herramientas distribuidas modernas.
* Identificar los problemas clave que resolvieron estos sistemas en el desarrollo de software, como la pérdida de código, conflictos en equipo y falta de trazabilidad.
* Examinar los flujos de trabajo estandarizados (Git Flow, GitHub Flow, entre otros) y su aplicación en proyectos reales.
* Destacar la relevancia actual de los VCS en metodologías ágiles, DevOps y desarrollo colaborativo.

# **Relevancia del Tema**

En la era del desarrollo de software colaborativo y la entrega continua, los Sistemas de Control de Versiones se han convertido en una pieza fundamental por las siguientes razones:

* Eficiencia en equipos: Permiten que múltiples desarrolladores trabajen simultáneamente sin sobrescribir cambios, facilitando la escalabilidad de proyectos.
* Seguridad e integridad: Garantizan un historial completo de modificaciones, permitiendo revertir errores o rastrear bugs.
* Adaptabilidad a metodologías modernas: Son la base de prácticas como CI/CD (Integración/Despliegue Continuo) y repositorios en la nube (GitHub, GitLab).
* Estándar en la industria: El 90% de las empresas tech utilizan Git, según surveys recientes, lo que lo convierte en una habilidad esencial para desarrolladores.

# Fundamentos Teóricos y Definiciones

## **Sistema de Control de Versiones (VCS)**

Herramienta que registra cambios en archivos a lo largo del tiempo, permitiendo recuperar versiones específicas.

**Clasificación:**

* Centralizados (CVCS): Ej: Subversion (SVN). Requieren un servidor central.
* Distribuidos (DVCS): Ej: Git, Mercurial. Cada usuario tiene una copia completa del repositorio.

## Git

DVCS creado por Linus Torvalds (2005) para gestionar el kernel de Linux.

**Objetivos clave:**

* Velocidad en operaciones locales.
* Integridad mediante hashes SHA-1.
* Soporte para trabajo offline y colaboración no lineal.

# Arquitectura de Git

## Modelo de Datos

Git almacena información como objetos inmutables en .git/objects:

* Blobs: Contenido crudo de archivos (sin metadatos).
* Trees: Directorios que referencian blobs u otros trees.
* Commits: Instantáneas con autor, fecha, mensaje y referencia a un tree padre.
* Tags: Etiquetas estáticas para marcar commits importantes (ej: v1.0).

## Grafo de Commits

* Los commits forman un DAG (Directed Acyclic Graph) donde cada nodo es un commit y las aristas son relaciones padre-hijo.
* HEAD: Puntero al commit actual.

## Protocolos de Transferencia

* Local: Operaciones directas en el mismo sistema.
* HTTP/S: Acceso mediante APIs RESTful.
* Git: Protocolo nativo (puerto 9418).
* SSH: Para autenticación segura.

## Flujos de Trabajo (Workflows)

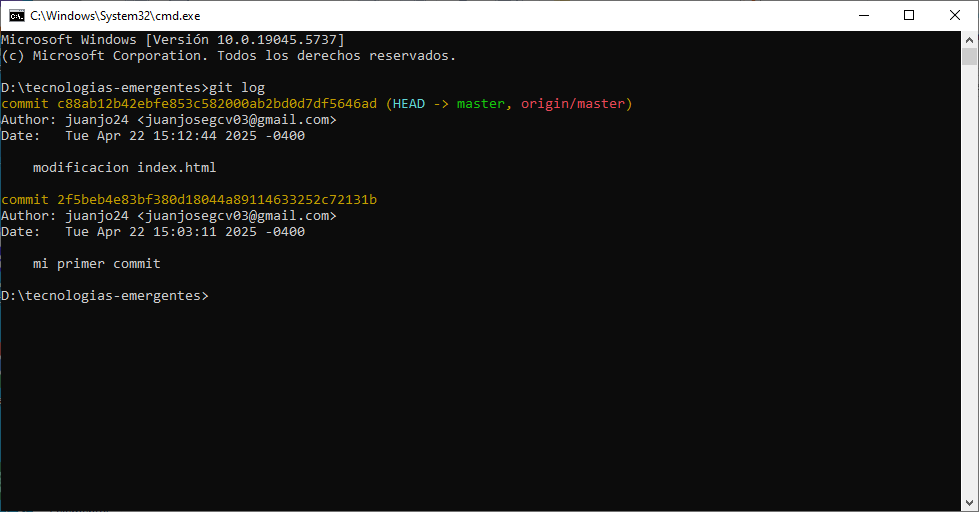
## Git Flow

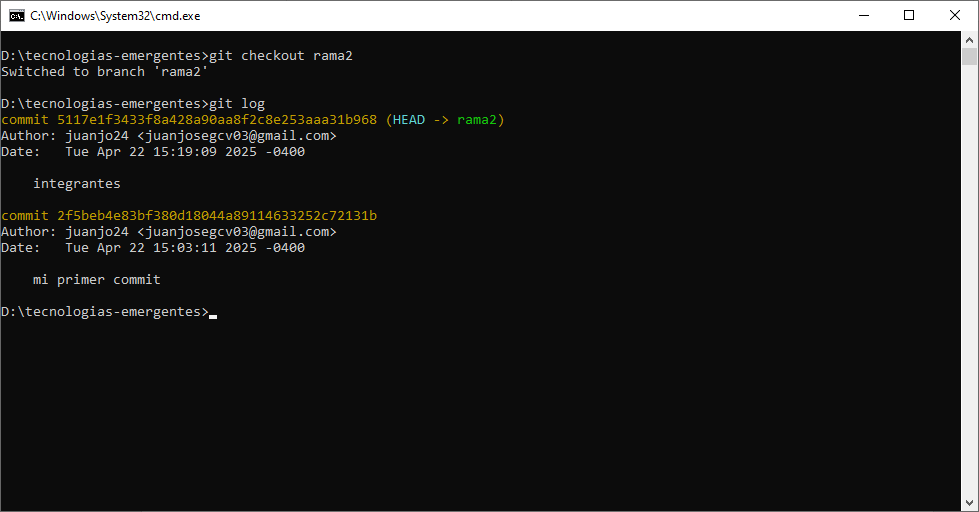
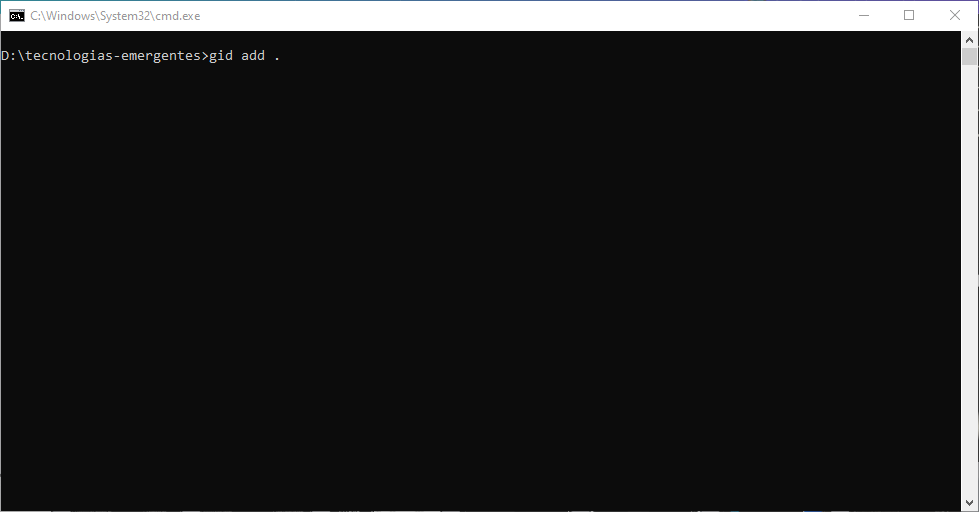
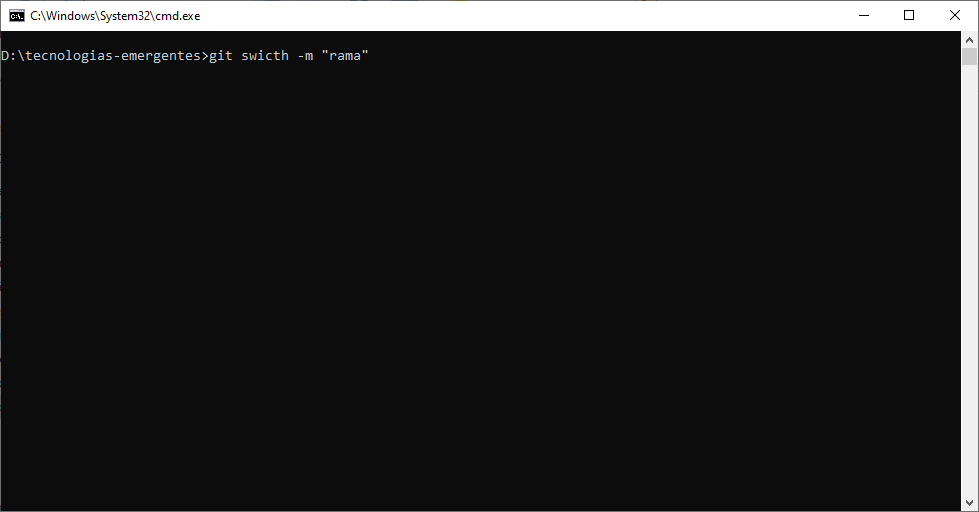
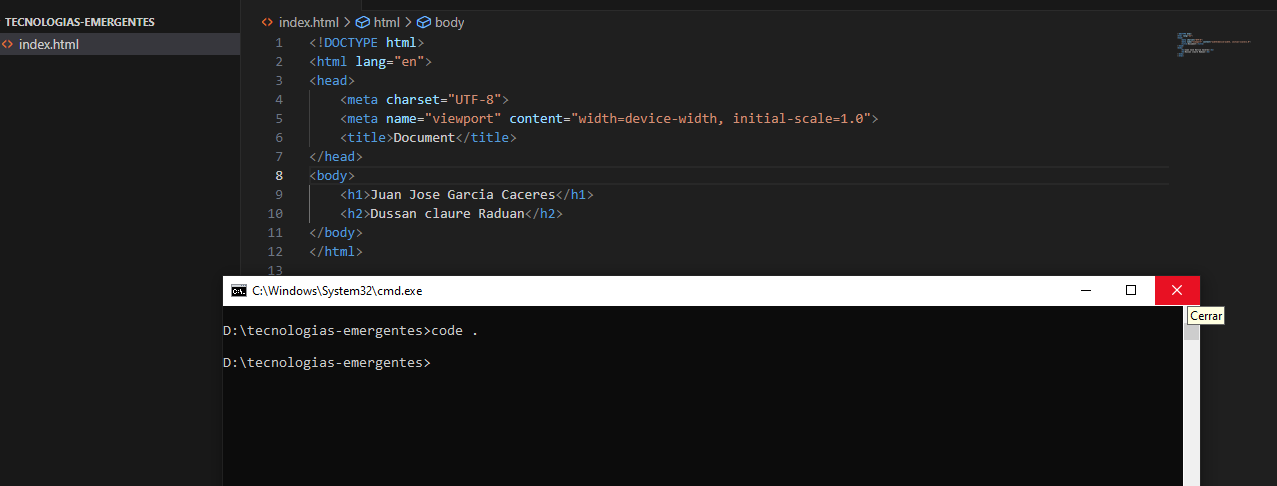
* Ramas principales:
  + main: Código estable (producción).
  + develop: Integración de features.
* Ramas de soporte:
  + feature/\*: Desarrollo de funcionalidades.
  + release/\*: Preparación de versiones.
  + hotfix/\*: Correcciones urgentes.
* Uso ideal: Proyectos con releases planificados.

## GitHub Flow

* Ramas:
  + main: Siempre desplegable.
  + feature/\*: Ramas temporales por cambio.
* Proceso:
  + Crear rama desde main.
  + Abrir Pull Request (PR) para revisión.
  + Merge a main tras aprobación.
* Uso ideal: Proyectos ágiles con CI/CD.

# Ejemplos :





**Conclusiones:**

Los Sistemas de Control de Versiones (VCS) han revolucionado el desarrollo de software al proporcionar un marco estructurado para gestionar cambios, facilitar la colaboración y garantizar la integridad del código.

Los VCS, liderados por Git, han transformado la ingeniería de software de una práctica artesanal a un proceso industrializado, donde la colaboración global, la iteración rápida y la calidad son prioritarias. Su adopción ya no es opcional, sino un requisito fundamental para cualquier equipo de desarrollo que busque eficiencia, transparencia y control en la evolución de sus proyectos.