Logotipo

El contenido generado por IA puede ser incorrecto.

Santiago Yosa González

**Ingeniero Sistemas de Información**



Logotipo

El contenido generado por IA puede ser incorrecto.}

**Unidad I**

**Fundamentación Sistemas de control de versiones**

-Creación y clonación de repositorios Git.

-Manejo de archivos: añadir, modificar, eliminar y confirmar cambios (commits).

-Uso de ramas en Git: creación, fusión y resolución de conflictos

**GIT**

**Situación**

Imaginen una película de viajes en el tiempo. El protagonista está en la **Línea de Tiempo Sagrada** (Rama main). De repente, viaja al pasado y mueve una silla. Esto crea una **Línea de Tiempo Ramificada** (Rama feature). En esa nueva línea, el protagonista puede construir una casa entera sin afectar la línea sagrada. Pueden existir 10 realidades paralelas al mismo tiempo.

Pero, ¿qué pasa al final de la película? Debemos **unificar** las líneas de tiempo para tener un final coherente.

* Si en la línea A el héroe se comió una pizza, y en la línea B no hizo nada, al unirlas, el héroe se comió la pizza (**Merge limpio**).
* Pero si en la línea A el héroe salvó a la ciudad, y en la línea B destruyó la ciudad... al intentar unirlas, el universo colapsa. No pueden pasar las dos cosas a la vez. El guionista (ustedes) debe decidir qué pasó realmente. Eso es un **Conflicto de Merge**.

**Clonación y Remotos (origin)**

git clone descarga un proyecto desde la nube. git push sube cambios. git pull baja cambios.

**¿Qué es origin?** No es una palabra mágica. Es un **alias** (un apodo) estándar para la URL del repositorio remoto. Podrías llamarlo github o servidor, pero por convención usamos origin.

**Git Clone vs. Git Init:** git clone automáticamente:

1. Hace git init.
2. Añade el remoto (git remote add origin URL).
3. Baja todos los datos (git fetch).
4. Hace check out en la rama principal.

**Referencias Remotas:** Cuando haces git fetch, bajas las ramas remotas (ej. origin/main). Estas son **referencias de solo lectura**. No puedes editar origin/main directamente; debes fusionarla en tu main local.

**Ramas**

Esta es la parte técnica más importante para desmitificar Git.

* **¿Qué es una rama?**
  + Muchos piensan que es una copia de los archivos. **FALSO**.
  + Una rama es simplemente un **puntero móvil** (un archivo de texto de 40 bytes) que apunta a un commit específico (Hash).
  + Crear una rama es instantáneo (O(1)) porque solo se crea un archivo nuevo con un hash. No se copian gigabytes de datos.
* **¿Qué es HEAD?**
  + Es un puntero que dice "¿Dónde estoy parado yo?". Usualmente HEAD apunta a una Rama, y la Rama apunta a un Commit.
  + **Detached HEAD (Cabeza separada):** Ocurre si haces checkout directo a un Commit (git checkout a1b2c3) en lugar de a una rama. Significa que no estás en ninguna rama. Si haces commit aquí, esos cambios no pertenecerán a ninguna rama y el "Garbage Collector" de Git los borrará eventualmente.

**Fusión (Merge) y Tipos de Merge**

Existen dos formas principales en que Git une historias:

1. **Fast-Forward (Avance Rápido):**
   * Si la rama main no ha avanzado desde que creaste feature, Git simplemente mueve el puntero de main hacia adelante hasta donde está feature. Es lineal. No crea un "commit de fusión".
2. **3-Way Merge (Merge Recursivo):**
   * Si main avanzó (alguien más subió cambios) Y feature también avanzó.
   * Git debe crear un **Nuevo Commit de Merge**. Este commit es especial: tiene **dos padres**.
   * Se llama "3-way" porque Git usa 3 puntos para comparar:
     1. La punta de main.
     2. La punta de feature.
     3. El **Ancestro Común** más cercano (la base desde donde se bifurcaron).

**Conflictos**

Un conflicto ocurre cuando Git no puede decidir automáticamente qué cambio conservar en una fusión de 3 vías.

**Anatomía de un conflicto:** Git modifica el archivo físico e inserta marcadores:

<<<<<<< HEAD (Current Change)

print("Hola Mundo")

=======

print("Hello World")

>>>>>>> feature/ingles (Incoming Change)

* <<<<<<< HEAD: Lo que tú tienes en tu rama actual.
* =======: El separador (la frontera).
* >>>>>>> rama: Lo que viene de la rama que intentas fusionar.

**La Regla de Oro:** Para solucionar un conflicto, debes **borrar los marcadores** y dejar el código exactamente como quieres que funcione. Git no "sabe" cuál es correcto, tú sí.

**Preguntas Frecuentes**

**¿Si borro una rama, borro mis commits?**

* **R:** Depende. Si la rama fue fusionada (merge) a main, los commits ya son parte de la historia de main, así que borrar la etiqueta de la rama es seguro. Si la rama *no* ha sido fusionada y la borras (forzando con -D), esos commits quedan "huérfanos" y se perderán. Las ramas son desechables, los commits son (casi) sagrados.

**Hice git pull y me salieron esos símbolos raros (<<<<). ¿Rompí el código?**

* **R:** No, el código está en un estado intermedio de fusión. Git te está protegiendo de sobrescribir el trabajo de tu compañero o el tuyo propio. Es normal. Solo necesitas editar el archivo, decidir qué líneas se quedan, guardar y hacer un commit para finalizar la fusión.

**Demo 1: La Rama Fantasma (Visualizando Punteros)**

1. En un repo limpio, hacer un commit.
2. Crear rama git branch prueba.
3. Hacer otro commit en main.
4. Mostrar con git log --graph --all --oneline cómo se bifurcan visualmente.

**Demo 2: El Conflicto Intencional (Hacer en vivo)**

1. Crear archivo historia.txt con la línea: "Había una vez un perro." -> Commit en main.
2. Crear rama git checkout -b gato.
3. Cambiar la línea a: "Había una vez un gato." -> Commit en rama gato.
4. Volver a main: git checkout main.
5. Cambiar la misma línea a: "Había una vez un dragón." -> Commit en main.
6. **El Choque:** Intentar git merge gato.
7. Mostrar el error en terminal: *CONFLICT (content): Merge conflict in historia.txt*.
8. Abrir VS Code. Mostrar los marcadores. Elegir "Dragón" (o ambos). Finalizar el merge.

Tabla

El contenido generado por IA puede ser incorrecto.

Antes de comenzar una nueva tarea o al inicio de una sesión

git checkout main # Cambia a la branch principal

git pull origin main # Descarga los últimos cambios del remoto

Si estás trabajando en una branch personal existente**, asegúrate de sincronizarla con la branch principal de vez en cuando:**

git merge origin main # Merge los cambios de main a tu branch

**Convenciones**

|  |  |  |  |
| --- | --- | --- | --- |
| Prefijo | Significado | ¿Qué es? | Ejemplo de Mensaje |
| feat | Feature (funcionalidad) | Se usa cuando añades una nueva funcionalidad o característica al proyecto. Es un cambio que agrega valor. | feat: Añadir botón de inicio de sesión |
| fix | Fix (arreglo) | Se usa cuando corriges un bug o un error en el código. Es un cambio que soluciona un problema. | fix: Corregir error de validación en formulario |
| docs | Docs (documentación) | Se usa para cambios que solo afectan a la documentación del proyecto, como el archivo README.md o los comentarios en el código. | docs: Actualizar instrucciones de instalación |
| style | Style (estilo) | Se usa para cambios que no afectan a la lógica del código, sino a su formato (espacios en blanco, formato de las líneas, punto y coma, etc.). | style: Ajustar indentación en archivos JS |
| refactor | Refactor (refactorización) | Se usa para cambios que reestructuran el código sin cambiar su comportamiento. Mejora la legibilidad o el rendimiento interno. | refactor: Simplificar la lógica de la función de cálculo |
| test | Test (prueba) | Se usa para añadir o modificar pruebas (tests) unitarias o de integración. | test: Añadir prueba para la función de suma |
| ci | Continuous Integration | Se usa para cambios relacionados con la configuración de la integración continua (CI) o el despliegue. | ci: Configurar despliegue automático |
| chore | Chore (tarea secundaria) | Se usa para cambios rutinarios o de mantenimiento que no afectan al código del proyecto, como actualizaciones de dependencias o build scripts. | chore: Actualizar dependencias de npm |

**Preguntas afianzamiento**

**Estoy en la rama login. Hago cambios en un archivo pero NO hago commit. Luego hago git checkout main. ¿Qué pasa con mis cambios?**

Si los cambios no entran en conflicto con archivos de main, **los cambios viajan contigo**. Git se lleva el "trabajo sucio" a la otra rama. Si hay conflicto, Git no te dejará cambiar de rama hasta que hagas commit o guardes los cambios (stash).

**¿Es mala práctica tener 20 ramas abiertas en mi repositorio local?**

No, en absoluto. Recuerda que las ramas son solo punteros de 40 bytes. Puedes tener mil ramas. Lo que es mala práctica es tener ramas "zombies" que ya se fusionaron hace meses y siguen ahí generando ruido visual.

**Práctica 1: Ciclo Completo de Ramas (Aprendizaje)**

Crear una funcionalidad en una rama separada y fusionarla exitosamente.

**Paso a paso (Estudiante):**

1. Crea un nuevo repo: mkdir practica-ramas, cd, git init.
2. Crea index.html con estructura básica HTML5. Commit: "Inicio proyecto".
3. **Crea la rama:** git checkout -b feature/contacto. (Verifica con git branch que estás ahí).
4. Modifica index.html: Agrega un formulario <form>...</form>.
5. Commit en la rama: git commit -am "Agregado formulario de contacto".
6. **Vuelve al inicio:** git checkout main (Observa que el formulario desaparece).
7. **La Fusión:** git merge feature/contacto.
8. **Limpieza:** Borra la rama ya que terminaste: git branch -d feature/contacto.

**¿Debería saber algo más?**

1. **git stash (El Portapapeles Temporal):**
   * Situación: Estás trabajando en la rama feature con todo el código "roto" (a medias) y tu jefe te pide un arreglo urgente en main. No puedes cambiar de rama porque Git te lo prohíbe (archivos sucios).
   * Solución: No hagas un "commit basura". Usa git stash.
   * git stash: Guarda los cambios en un limbo temporal y limpia tu directorio.
   * Cambias a main, arreglas, vuelves.
   * git stash pop: Devuelve tus cambios sucios para seguir trabajando.
2. **git rebase vs. git merge (La guerra santa):**
   * merge: Preserva la historia tal cual ocurrió. Es veraz, pero sucio (muchos cruces de líneas).
   * rebase: Reescribe la historia. Toma tus cambios de la rama y los pone *al frente* de main como si hubieran pasado hoy.
   * **Peligro:** NUNCA hagas rebase en ramas públicas/compartidas. Cambia los hashes de los commits. Si tus estudiantes hacen rebase en una rama que otro está usando, romperán el repo del otro.
   * *Recomendación docente:* Enseña merge primero. rebase es para cuando dominen el flujo.
3. **.gitignore Global:**
   * Puedes configurar un gitignore para *toda* tu computadora (ej. para ignorar siempre archivos .DS\_Store de Mac o Thumbs.db de Windows).
   * git config --global core.excludesfile ~/.gitignore\_global.

Logotipo

El contenido generado por IA puede ser incorrecto.