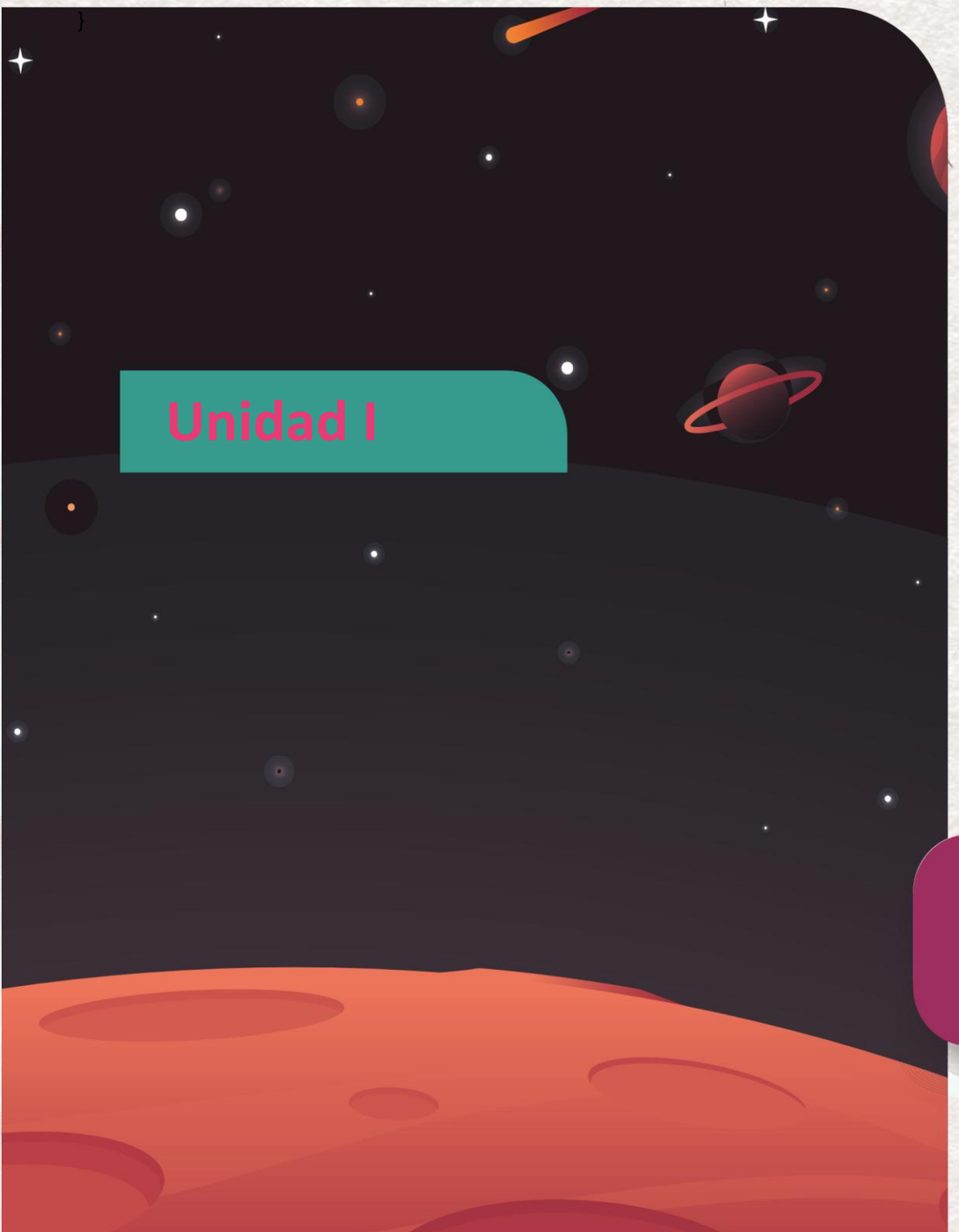
CESDE

Santiago Yosa González

Ingeniero Sistemas de Información



Fundamentación Sistemas de control de versiones

-Introducción a Git: conceptos básicos, terminología y beneficios para el control de versiones.

-Configuración inicial de Git: configuración de usuario, iniciación de repositorios y conceptos básicos del flujo de trabajo de Git

GIT

Situación

Imagina que estás trabajando en un proyecto de equipo para la universidad. El trabajo final es un único archivo de código. Un compañero hace cambios, otro hace los suyos, y tú haces los tuyos. Para no perder el trabajo, se guardan copias con nombres como proyecto_final_v1.py, proyecto_final_v2_corregido.py, proyecto_final_definitivo.py.

Al final, nadie sabe qué versión es la más actual, quién hizo qué cambio, o cómo revertir un error que rompió el código hace una semana.

¿Cómo manejan actualmente los cambios o versiones de sus proyectos cuando trabajan en equipo o incluso individualmente?

Un **Sistema de Control de Versiones (VCS)** es un software que ayuda a un equipo a gestionar los cambios en un conjunto de archivos a lo largo del tiempo. Permite rastrear quién hizo qué, cuándo y por qué, y revertir a versiones anteriores si es necesario.

Git es un sistema de control de versiones distribuido que fue creado por Linus Torvalds. Su principal característica es que cada usuario tiene un repositorio local completo, lo que lo hace rápido, robusto y descentralizado.

Debemos entender que Git **no** guarda "diferencias" (deltas) por defecto, guarda **Snapshots** (instantáneas).

Directorio de Trabajo (Working Directory): Los archivos de tu proyecto tal como los ves en tu disco duro. Cualquier cambio que hagas aquí se considera un cambio sin rastrear.
Área de Staging (Staging Area / Index): Un área intermedia donde seleccionas los cambios específicos que quieres guardar en tu próximo *commit*. Es como una "lista de espera" para el repositorio.
Repositorio (Repository): La base de datos de Git, donde se guardan de forma permanente los *commits* (las instantáneas de tus cambios) y todo el historial del proyecto.

Terminología Clave de Git:

- **Repositorio (Repo):** Una carpeta que Git rastrea. Contiene todos los archivos del proyecto y la base de datos de Git (.git/).
- **Commit:** Una "instantánea" de los cambios en un momento determinado. Es la unidad básica del historial. Cada *commit* tiene un ID único y un mensaje que explica el cambio.
- **Branch (Rama):** Un puntero ligero y movible a un *commit*. Permite trabajar en nuevas funcionalidades sin afectar la rama principal (generalmente main o master).
- **main (o master):** La rama principal de un proyecto. Es la versión estable y funcional.
- **Pull:** El acto de descargar los cambios de un repositorio remoto a tu repositorio local.
- **Push:** El acto de enviar tus *commits* locales a un repositorio remoto.
- **Clone:** Crear una copia local completa de un repositorio remoto existente.

Antes de usar Git, debes configurarlo para que sepa quién eres. Esta configuración se guarda de forma global en tu máquina.

Verificar si hay "basura" de otros usuarios
git config --list --show-origin

git config --global --unset user.name
git config --global --unset user.email

Configurar tu nombre y correo, si es tu maquina personal:

- git config --global user.name "Tu Nombre"
- git config --global user.email tu.correo@ejemplo.com

Consideración para equipos públicos: Debemos considerar en equipos públicos la configuración como Local.
No contiene credenciales de acceso. Para evitar que persistan los datos de estudiante a otro, al final de la sesión se debe borrar estos datos de global si así lo hizo o de local si lo desea.

Cacheo del Personal Access Token

git config --global credential.helper wincred

 # Borra las credenciales guardadas en Windows para el dominio
github.com cmdkey /delete:git:https://github.com
cmdkey /delete:“legacyGeneric:target=git:https://github.com”
O usando git credential (más técnico)
echo "url=https://github.com" | git credential reject

Configurar un nuevo repositorio:

- git init: Inicializa un nuevo repositorio de Git en el directorio actual.
- git clone [URL]: Clona un repositorio ya existente

Flujo de trabajo básico de comandos:

- **git status:** Muestra el estado actual del directorio de trabajo y del *staging area*.
- **git add <nombre_del_archivo> o git add .**Mueve los cambios de un archivo específico al *staging area*.
- **git commit -m "Mensaje del commit":** Guarda los cambios que están en el *staging area* como un nuevo *commit* en el repositorio.
- **git log:** Muestra el historial de *commits* del repositorio.

Hasta aquí guardamos nuestro trabajo en local.

- **git remote add origin <URL_del_repositorio_remoto>:** Este es un paso que se realiza una sola vez para decirle a tu repositorio local dónde se encuentra el repositorio remoto.
- **Push:** Transmites los *commits* que has realizado en tu repositorio local al repositorio remoto.

  git push -u origin main (la primera vez)
  Despues sería: git push

- **git pull:** Obtener los cambios de otros, que se han subido al repositorio remoto.

Crear un Repositorio en GitHub

- Iniciar sesión en GitHub
- Hacer clic en "New Repository"
- Asignar un nombre y, opcionalmente, una descripción
- Elegir "Public" o "Private"
- Seleccionar "Add a README file".
- Seleccionar “Add .gitignore”.
- Seleccionar “Add License”.
- Hacer clic en "Create Repository"

Diferencias entre un clone y un fork

Característica	Fork (Bifurcar)	Clone (Clonar)
Ubicación de la Copia	Crea una copia completa del repositorio en tu cuenta de GitHub (o de la plataforma). Es una copia remota.	Crea una copia completa del repositorio en tu computadora local. Es una copia local.
Propósito Principal	Hacer una copia personal para proponer cambios al proyecto original, desarrollar una versión independiente, o simplemente experimentar sin afectar a los demás.	Descargar el repositorio para trabajar directamente en él en tu máquina, ya sea para colaborar o simplemente para usarlo.
Permisos de Escritura	Tienes permisos completos de escritura (push) en tu fork porque es un repositorio en tu propia cuenta.	Generalmente, solo puedes hacer push al repositorio clonado si tienes permisos de colaborador en el repositorio original.

Categoría	Comando	Descripción
🔧 Configuración	git config --global user.name "Tu Nombre"	Configura tu nombre en Git.
	git config --global user.email "tuemail@example.com"	Configura tu correo en Git.
	git config --global -l	Muestra la configuración global.
📂 Crear o Clonar Repositorio	git init	Inicializa un repositorio Git en la carpeta actual.
	git clone URL_DEL_REPOSITORIO	Clona un repositorio remoto en tu máquina.
📌 Estado y Seguimiento	git status	Muestra el estado de los archivos.
	git add .	Agrega todos los archivos al área de preparación.
	git add nombre_archivo	Agrega un archivo específico al área de preparación.
	git reset nombre_archivo	Quita un archivo del área de preparación.
	git rm nombre_archivo	Elimina un archivo del repositorio.
✅ Confirmar Cambios (Commits)	git commit -m "Mensaje del commit"	Guarda los cambios en el historial con un mensaje.
	git commit --amend -m "Nuevo mensaje"	Modifica el último commit.
🔄 Trabajo con Ramas	git branch	Muestra las ramas disponibles.
	git branch nombre_rama	Crea una nueva rama.
	git checkout nombre_rama	Cambia a otra rama.
	git checkout -b nombre_rama	Crea y cambia a una nueva rama.
	git merge nombre_rama	Fusiona una rama con la actual.
	git branch -d nombre_rama	Elimina una rama local.
🚀 Subir y Descargar Cambios	git remote add origin URL_DEL_REPOSITORIO	Asocia el repositorio local con un remoto.
	git push origin nombre_rama	Envía los cambios al repositorio remoto.
	git pull origin nombre_rama	Descarga y fusiona los cambios del remoto.
	git fetch origin	Descarga cambios sin fusionarlos aún.
🔍 Revisar Historial	git log	Muestra el historial de commits.
	git log --oneline --graph --all	Muestra un historial simplificado con ramas.
	git show HASH_DEL_COMMIT	Muestra detalles de un commit específico.
⏪ Deshacer Cambios	git checkout -- nombre_archivo	Revierte un archivo a su última versión confirmada.
	git reset --soft HEAD~1	Revierte el último commit pero mantiene los cambios en staging.
	git reset --hard HEAD~1	Revierte el último commit y borra los cambios.
	git revert HASH_DEL_COMMIT	Revierte un commit creando un nuevo commit.
🛠️ Otros Comandos Útiles	git stash	Guarda temporalmente cambios no confirmados.
	git stash pop	Recupera los cambios guardados con git stash.
	git diff	Muestra las diferencias entre la versión actual y la última confirmada.

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

Estrategia Gitflow

Main (Master): Es "Tierra Sagrada". Lo que hay aquí *funciona*. Es producción. Nadie hace cambios directos aquí.
Develop (Desarrollo): Donde se integra todo el trabajo del equipo.
Feature Branches (Características): Cada integrante crea su propia "rama" para trabajar en una tarea específica (ej. feature/login, feature/boton-pago).

- *Regla:* Nunca trabajes en main. Crea una rama, trabaja, y luego fusiónala.

Hice un commit pero me equivoqué de nombre/correo, ¿tengo que borrar todo?"
No. Si es el *último* commit y no lo has subido a GitHub, puedes usar git commit --amend --author="Nuevo Nombre <correo@ejemplo.com>" para reescribir la historia de ese último paso. (Ojo: esto cambia el "hash" del commit).

2. Git me pide usuario y contraseña en la terminal, pero yo entro con Google/GitHub, ¿qué pongo?
Desde 2021, GitHub ya no acepta la contraseña de tu cuenta en la terminal. Debes usar un "Personal Access Token" O, mejor aún, dejar el campo vacío y esperar a que el "Git Credential Manager" abra la ventana emergente del navegador para autenticarte visualmente. Si no sale la ventana, tu configuración de credential.helper está mal.

Ejemplos demostrativos

Demo 1: El Ciclo de Vida y el .git

1. Crear carpeta demo_git.
2. git init.
3. **Visualización:** Abrir la carpeta y activar "Mostrar elementos ocultos". Entrar a .git. Mostrar que ahí "vive" la base de datos. Borrar esa carpeta es borrar la historia.
4. Crear archivo.txt.
5. git status (Está en rojo/Untracked).
6. git add archivo.txt.
7. git status (Está en verde/Staged).
8. git commit -m "Primer guardado".
9. git log (Ver el hash único).

Demo 2: Trabajemos en un repositorio común

```
# frase_final.py
# El programa solo funcionará una vez que todas las variables estén completas y fusionadas.

# El líder del equipo añade su parte (la primera palabra de la frase)
parte_1 = "El gato"


# La frase completa se genera concatenando las variables
frase_final = f"{parte_1} {parte_2} {parte_3} {parte_4} {parte_5} {parte_6}."
print(frase_final)
```

¿Qué creen que sucede si dos personas modifican la misma línea de código y luego intentan unir sus cambios?

Preguntas motivadoras y de afianzamiento

Si borro mi archivo trabajo.py de la carpeta y vacío la papelera de reciclaje, pero ya había hecho un commit antes. ¿Perdí el archivo?

No. El archivo vive seguro dentro de la base de datos .git. Puedes recuperarlo con un comando de checkout/restore. El Working Directory es efímero, el repositorio es permanente.

Estamos trabajando con Gitflow. ¿Por qué no puedo hacer mis cambios directamente en la rama main si soy el único trabajando en el proyecto hoy?"

Porque estás creando un hábito. En un equipo real, main puede estar desplegando automáticamente a producción. Si cometes un error en main, tumbas el servidor. Usar ramas (feature/lo-que-sea) te da un entorno seguro (sandbox) para romper cosas sin afectar el producto final.

Práctica 1: Inicialización Segura y Flujo Local

Configurar un entorno limpio en PC público y realizar el primer ciclo de commit.

Instrucciones:

1. **Limpieza Preventiva:** Abre la terminal (PowerShell/Git Bash) y ejecuta: git config --global --unset-all user.name git config --global --unset-all user.email Esto asegura que no uses el nombre de quien usó la PC antes.
2. **Crear Proyecto:** Crea una carpeta en el Escritorio llamada proyecto-inicial-[TU_NOMBRE].
3. **Inicializar:** Entra a la carpeta y ejecuta git init.
4. **Identidad Local (IMPORTANTE):** Configura tu firma SOLO para este proyecto: git config --local user.name "Tu Nombre Real" git config --local user.email "tu@email.com"
5. **Crear Contenido:** Crea un archivo hola.py y escribe print("Hola Git").
6. **Staging:** Ejecuta git add hola.py.
7. **Commit:** Ejecuta git commit -m "Creación del script de saludo".
8. **Verificación:** Ejecuta git log y verifica que aparezca TU nombre y tu correo en el autor.

Ramas y Simulación de Gitflow

Vas a simular la adición de una nueva funcionalidad usando ramas, protegiendo la rama principal.

Pasos a seguir:

1. Estando en tu proyecto anterior, verifica que estás en la rama principal (git branch). Debería decir main o master.
2. Crea una nueva rama para una "funcionalidad" de despedida. El comando es: git checkout -b feature/despedida (Esto crea la rama y te mueve a ella).
3. Crea un nuevo archivo adios.py con el código print("Adiós").
4. Realiza el add y el commit de este nuevo archivo.
5. Vuelve a la rama principal: git checkout main.
6. Observa tu carpeta. ¿Qué pasó con el archivo adios.py? (Debería desaparecer, esto es magia para los estudiantes).
7. Fusiona los cambios (Trae lo de la rama feature a main): git merge feature/despedida.
8. Verifica que el archivo reapareció.

Ejercicio independiente

Nombre: "Operación Limpieza Total: El Proyecto Secreto"
Contexto: Son agentes secretos. Deben clonar un repositorio, añadir información, subirla, y luego **eliminar todo rastro de su identidad** de la computadora pública para que el "enemigo" no pueda acceder a la cuenta.
Instrucciones:

1. Crea un repositorio nuevo en tu cuenta de GitHub (vía web) llamado agente-secreto.
2. Clona el repositorio en la PC (git clone URL).
3. Entra a la carpeta. Configura tu usuario de forma **LOCAL**.
4. Crea una rama feature/datos.
5. Crea un archivo mision.txt, haz add y commit.
6. Haz push de tu rama (git push origin feature/datos).
    - *Aquí saltará la ventana del navegador/wincred. Loguéate.*
7. Verifica en GitHub web que la rama existe.
8. **EL PASO FINAL (Evaluativo):** Ejecuta el protocolo de limpieza para borrar tus credenciales de Windows y eliminar la configuración local.
    - Borrar carpeta del proyecto.
    - Comando para borrar credenciales guardadas.

Criterios de logro:

- El código está en GitHub en la rama correcta.
- Al intentar hacer un nuevo git clone de un repo privado o un push después de la limpieza, la máquina **debe pedir credenciales de nuevo** (prueba de que se borraron correctamente).

¿Debería saber algo más?

El peligro del .git en servidores web:

- A veces los desarrolladores suben la carpeta .git a sus servidores de producción. Si alguien accede a tusitio.com/.git/config, puede ver las URLs de los remotos (que a veces incluyen tokens) o descargar todo el código fuente reconstruyendo los objetos.
- *Lección:* Siempre añadir .git al .gitignore (aunque es redundante) o configurar el servidor web para denegar acceso a esa carpeta.

Git no borra nada (Garbage Collection):

- Cuando borras una rama o haces un git reset --hard, los commits "perdidos" siguen en la base de datos como "dangling commits" (commits colgantes). Git solo los borra realmente cuando corre el git gc (Garbage Collector), que ocurre automáticamente cada cierto tiempo. Hasta entonces, un forense digital puede recuperarlos.

Credential Helpers Avanzados:

- Además de manager (Windows), existe store (guarda en texto plano en disco, **inseguro**) y cache (guarda en memoria RAM por X minutos).
- En un entorno público, git config --global credential.helper 'cache --timeout=3600' podría ser una alternativa interesante: guarda la clave en RAM por 1 hora y luego la olvida automáticamente, evitando el riesgo de que olviden borrarla de Windows Credential Manager.


CESDE
comfama