Módulo 8
Fundamentos de integración continua

Sistemas de Control de Versiones (SCV)

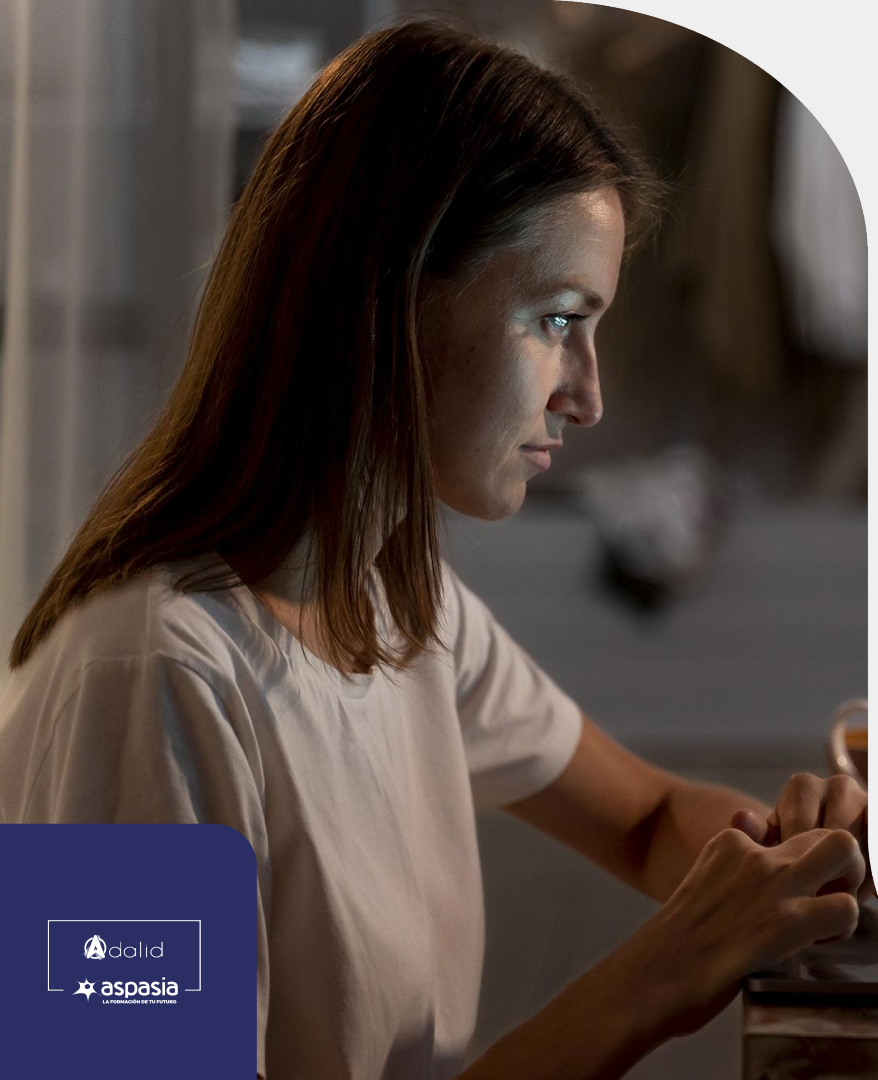
Módulo 8

AE 3.1

OBJETIVOS

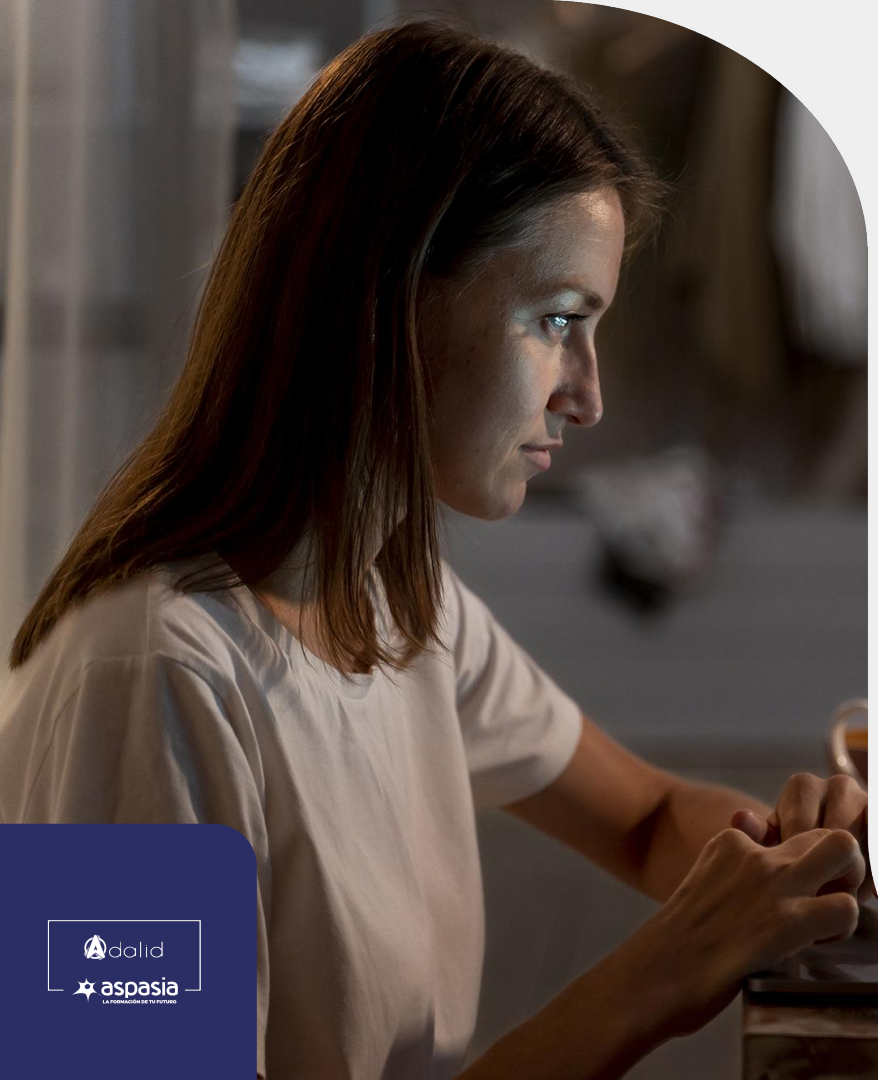
Entender qué es un SCV y cómo resuelve problemas en la gestión del código fuente. Conocer los conceptos clave de Git y su uso en proyectos de desarrollo. Aprender a gestionar repositorios locales y remotos con GitHub, GitLab y Bitbucket. Explorar flujos de trabajo eficientes con ramas, merge, stash y rebase.

Chile
dalid
aspasia
LA FORMACIÓN DE TU FUTURO



¿QUÉ VAMOS A VER?

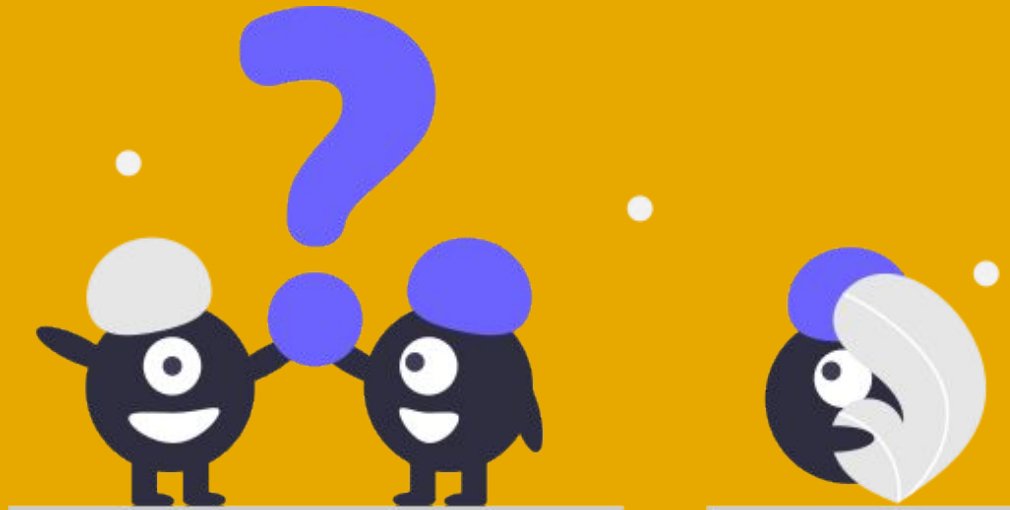
- Sistemas de Control de Versiones (SCV).
- Qué es un SCV.
- Problema que resuelve un SCV.
- Principales conceptos de un SCV. (Repositorio, Diff, Commit, Branch, Merge,
- Clone, Fork)
- Tipos de SCV y alternativas.
- Centralizados (SVN, CVS).
- Distribuidos (Git, Mercurial).
- Git como sistema de control de versiones.
- Instalación, configuración y comandos básicos.

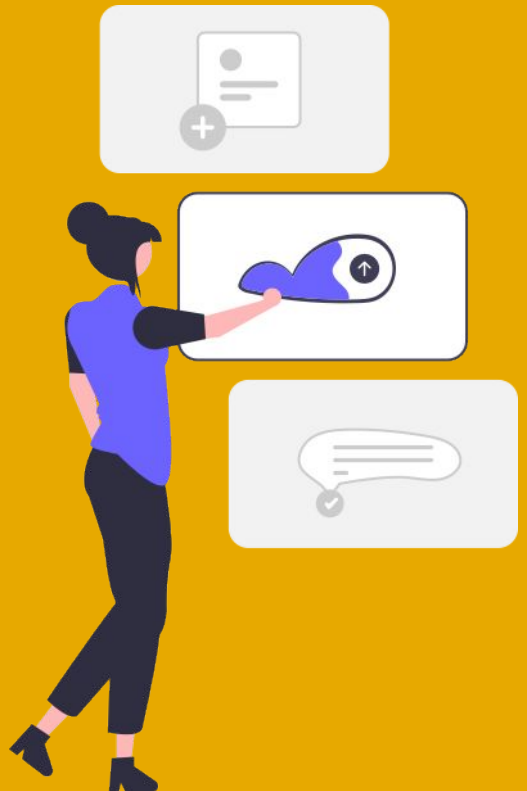


¿QUÉ VAMOS A VER?

- Utilización de Git en repositorios locales
- Commits y restauración de archivos.
- Ignorando archivos.
- Ramas,uniones, conflictos y tags.
- Stash y rebase.
- Centralización de repositorios.
- Servicios de centralización de repositorios (Gitlab, Github, Bitbucket)
- Repositorios remotos, push y pull.
- Fetch vs Pull.
- Clone y Fork de un repositorio.
- Flujos de trabajo.

¿Que es SCV?

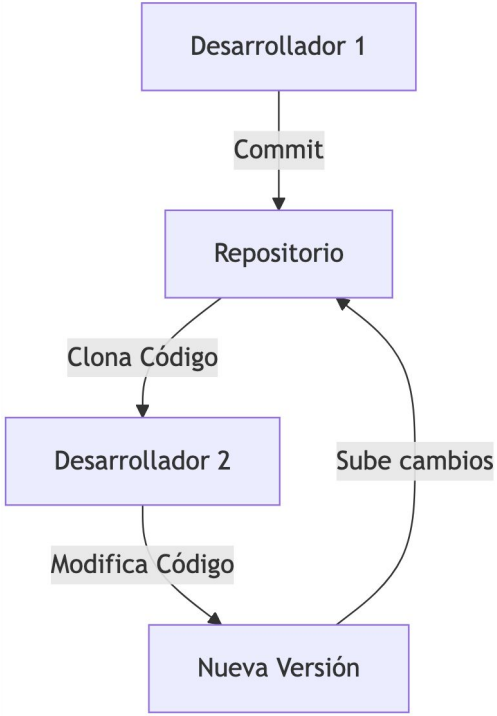




Introducción a los Sistemas de Control de Versiones (SCV)

¿Qué es un SCV?

Un **Sistema de Control de Versiones (SCV)** es una herramienta que permite **rastrear cambios en archivos** a lo largo del tiempo, facilitando la colaboración en proyectos de software.



Problema que Resuelve un SCV

Antes de usar SCV:

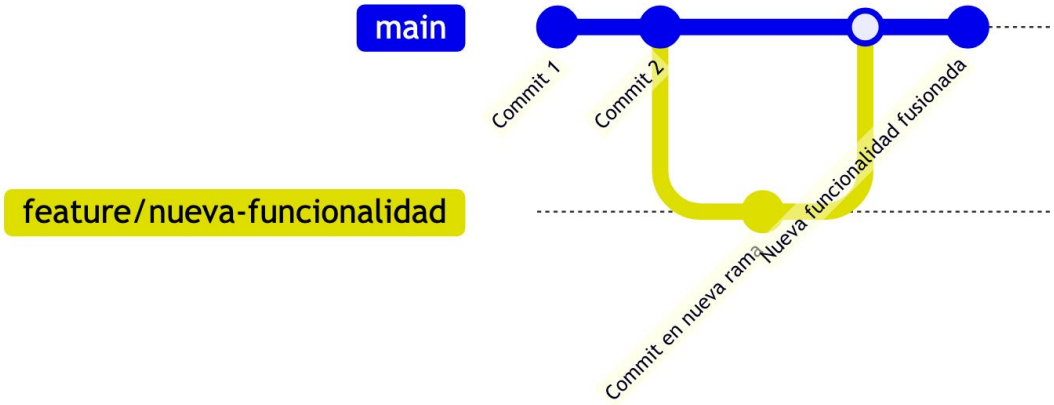
- Múltiples copias del código en carpetas diferentes.
- Dificultad para rastrear quién hizo qué cambio.
- Confusión en la fusión de cambios entre equipos.

Con SCV:

- Un historial organizado de cambios en el código.
- Permite revertir a versiones anteriores sin perder datos.
- Facilita la colaboración en equipos de desarrollo.

Conceptos Claves de un SCV

Ejemplo Visual de un Flujo con Branching:

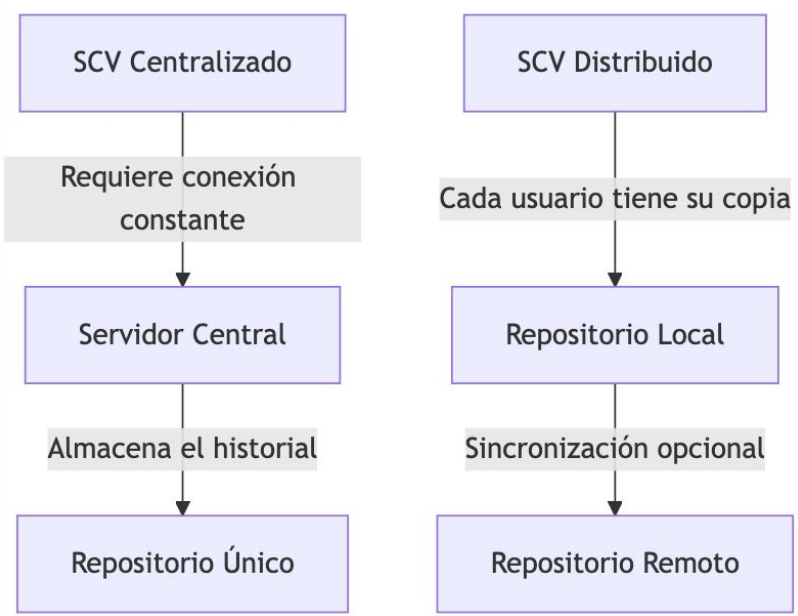


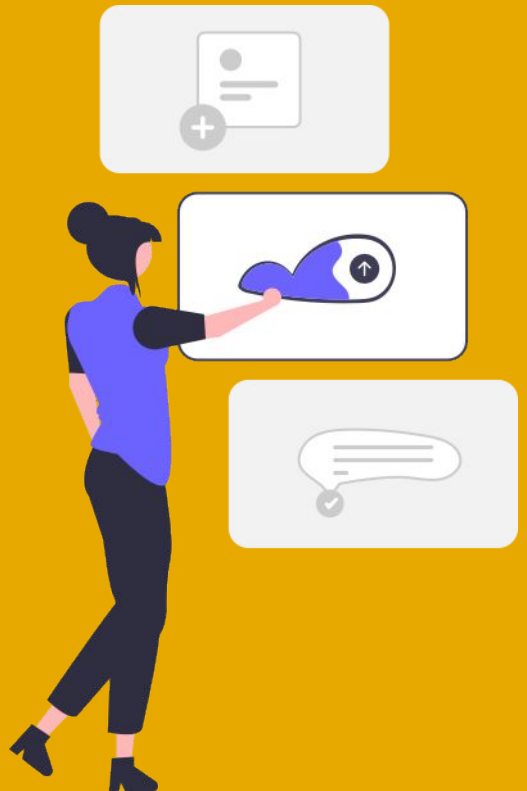
Terminología básica:

Concepto	Descripción
Repositorio	Lugar donde se almacenan los archivos y su historial de cambios.
Commit	Guarda cambios en el repositorio con un mensaje descriptivo.
Branch	Una rama de trabajo separada para desarrollar nuevas funcionalidades sin afectar la versión principal.
Merge	Fusiona una rama con otra para unir cambios.
Clone	Crea una copia exacta de un repositorio en otro equipo.
Fork	Crea una copia independiente de un repositorio en otra cuenta (usado en proyectos open-source).
Diff	Comparar cambios entre dos versiones de un archivo.

Tipos de SCV

Existen dos tipos principales de sistemas de control de versiones:





Git como Sistema de Control de Versiones

Adalid
aspasia
LA FORMACIÓN DE TU FUTURO

Introducción a Git (SCV más usado)

¿Por qué Git es el estándar hoy en día?

- **Rápido y eficiente** en proyectos grandes.
- **Soporta trabajo offline** con repositorios locales.
- **Facilita la colaboración** con herramientas como GitHub y GitLab.

¿Qué es Git y por qué es importante?

Git es un **sistema de control de versiones distribuido** que permite a los desarrolladores rastrear cambios en el código y colaborar en proyectos de manera eficiente.

Beneficios de usar Git:

- Rastreo de cambios en el código.
- Colaboración eficiente en equipos de desarrollo.
- Facilidad para revertir errores y restaurar versiones previas.
- Soporte para trabajo remoto con repositorios en GitHub, GitLab o Bitbucket.

Instalación de Git

Instalar Git en diferentes sistemas operativos:

- **Windows:**
  - https://git-scm.com/downloads
- **MacOS:**
  - brew install git
- **Linux (Debian/Ubuntu):**
  - sudo apt update
  - sudo apt install git

Configuración Inicial de Git

Configurar Git con el usuario correcto permite rastrear contribuciones adecuadamente.

Configurar el usuario y el correo electrónico:

```
git config --global user.name "Tu Nombre"
git config --global user.email "tuemail@example.com"
```

Verificar la configuración:

```
git config --list
```

Comandos básicos en Git

Comando	Función
`git init`	Inicializa un repositorio Git en una carpeta.
`git clone <URL>`	Clona un repositorio remoto en la máquina local.
`git add .`	Agrega todos los archivos al área de preparación.
`git commit -m "Mensaje"`	Guarda cambios en el historial con un mensaje.
`git push origin main`	Sube los cambios al repositorio remoto.
`git pull origin main`	Descarga los cambios más recientes del repositorio remoto.
`git branch <nombre>`	Crea una nueva rama de desarrollo.
`git checkout <rama>`	Cambia a otra rama.
`git merge <rama>`	Fusiona una rama con la principal.

Ejemplo de un Flujo de Trabajo con Git

Git optimiza el trabajo en equipo y la gestión de cambios en proyectos de software.

```
# Inicializar un repositorio
git init

# Agregar y confirmar cambios
git add .
git commit -m "Primer commit"

# Crear una nueva rama
git branch feature-nueva
git checkout feature-nueva

# Hacer cambios y fusionarlos con la rama principal
git add .
git commit -m "Implementando nueva funcionalidad"
git checkout main
git merge feature-nueva
```

Crear y Usar un Repositorio Local

Git permite registrar cambios de manera organizada.

Inicializar un repositorio Git en una carpeta:

```
mkdir mi-proyecto && cd mi-proyecto
git init
```

Ver el estado del repositorio:

```
git status
```

Añadir archivos al área de preparación:

```
git add .
```

Confirmar cambios con un commit:

```
git commit -m "Primer commit"
```

Ver el historial de commits:

```
git log --oneline
```

Restauración de Archivos y Commits

Git permite corregir errores y recuperar cambios fácilmente.

Deshacer cambios antes de hacer commit:

```
git checkout -- archivo.txt
```

Restaurar un archivo eliminado:

```
git checkout HEAD -- archivo.txt
```

Revertir un commit anterior:

```
git revert HEAD
```

Ignorar Archivos con .gitignore

Crear un archivo .gitignore para excluir archivos que no deben ser versionados. Evitar versionar archivos innecesarios mejora la limpieza del repositorio.

Ejemplo de **.gitignore**:

```
node_modules/
.env
build/
```

Añadir **.gitignore** al repositorio:

```
git add .gitignore
git commit -m "Añadiendo .gitignore"
```

Creación y Gestión de Ramas (Branching en Git)

Crear una nueva rama:

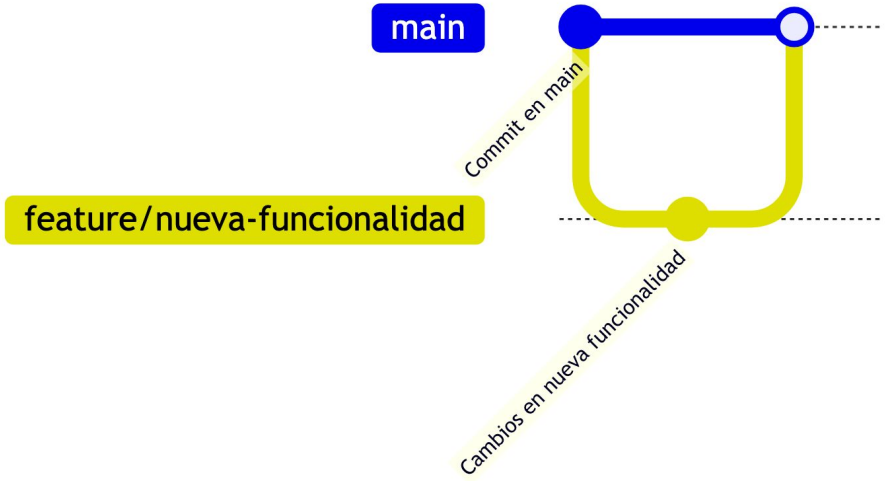
```
git branch nueva-rama
```

Cambiar a una rama existente:

```
git checkout nueva-rama
```

Crear y cambiar a una nueva rama directamente:

```
git checkout -b nueva-rama
```



Fusionar Ramas (Merge) y Resolver Conflictos

Git permite fusionar código de forma controlada y resolver conflictos fácilmente.

Fusionar una rama con main:

```
git checkout main
git merge nueva-rama
```

Si hay conflictos, Git mostrará algo como esto en el archivo afectado:

```
<<<<<<< HEAD
Código en la rama principal
=======
Código en la rama "nueva-rama"
>>>>>>> nueva-rama
```

Resolver el conflicto y confirmar cambios:

```
git add archivo-afectado
git commit -m "Resuelto conflicto en archivo"
```

Uso de Tags en Git

Los tags ayudan a marcar versiones importantes del proyecto.

Crear un tag en Git:

```
git tag -a v1.0 -m "Versión estable 1.0"
```

Subir un tag a un repositorio remoto:

```
git push origin v1.0
```

Listar todos los tags:

```
git tag
```

Uso de Stash en Git

git stash permite guardar cambios temporalmente y recuperarlos luego.

Guardar cambios temporalmente sin hacer commit:

```
git stash
```

Listar los cambios guardados:

```
git stash list
```

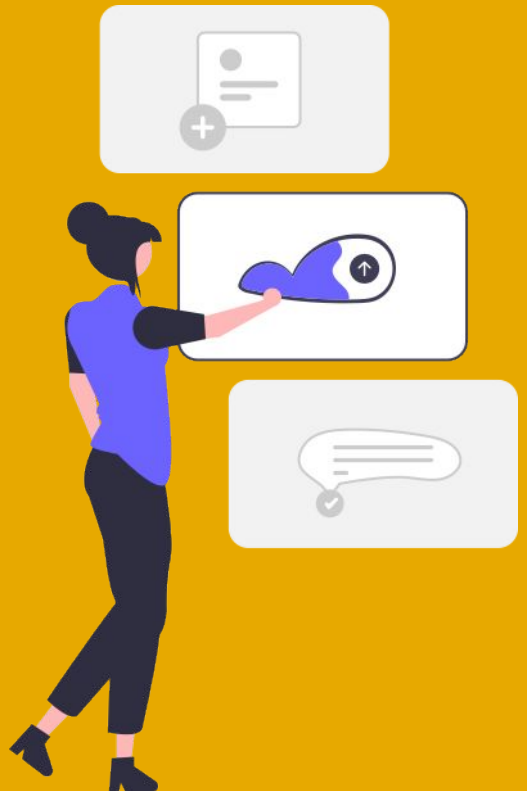
Restaurar los cambios más recientes:

```
git stash pop
```

Uso de Rebase en Git

git rebase es útil para limpiar el historial de commits antes de fusionar cambios.

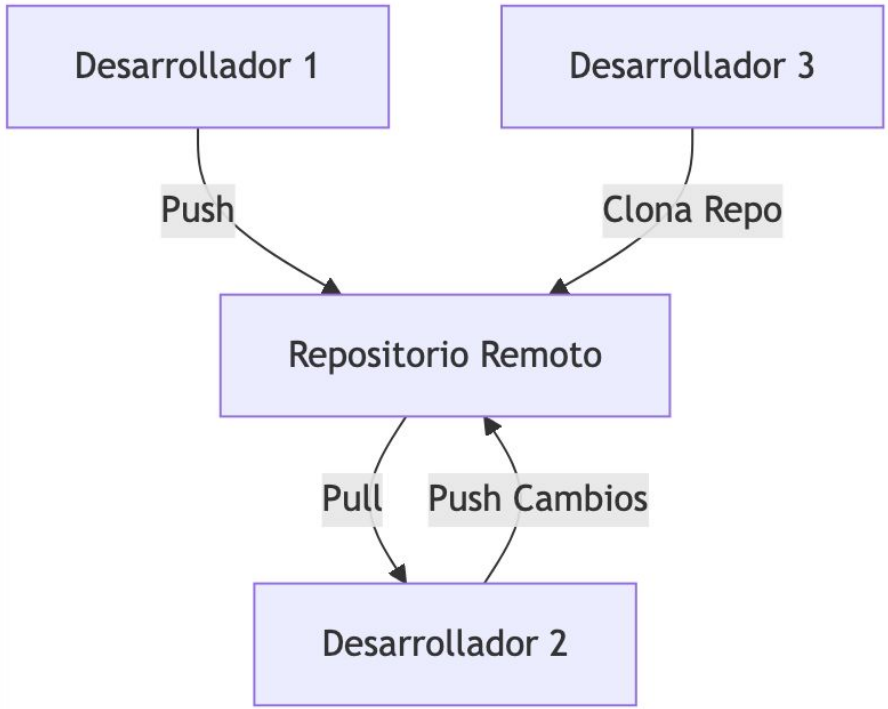
Comando	Ventaja	Desventaja
`git merge`	Mantiene el historial original	Puede generar commits innecesarios
`git rebase`	Mantiene un historial limpio	Puede ser riesgoso en repositorios compartidos



Centralización de Repositorios

¿Qué es la Centralización de Repositorios?

La **centralización de repositorios** es el uso de servicios en la nube para alojar código, permitiendo que múltiples desarrolladores trabajen en un mismo proyecto de manera colaborativa.



¿Qué es la Centralización de Repositorios?

¿Por qué centralizar repositorios?

- **Acceso remoto y seguro** al código fuente.
- **Colaboración eficiente** entre equipos.
- **Historial de cambios accesible** en cualquier momento.
- **Integración con herramientas de CI/CD** para automatización.

Servicios de Centralización de Repositorios

Servicio	Descripción	Beneficios
GitHub	Plataforma de repositorios en la nube, ideal para proyectos open-source y privados.	Integración con CI/CD, Issues, Actions.
GitLab	Solución completa con herramientas de CI/CD integradas.	Control total, autoalojable.
Bitbucket	Alternativa enfocada en equipos empresariales con integración a Jira.	Soporte para repositorios privados gratuitos.

¿Qué es un Repositorio Remoto?

Un **repositorio remoto** es una versión de un repositorio alojada en un servidor (GitHub, GitLab, Bitbucket) que se puede sincronizar con repositorios locales.

Comando	Función
`git remote add origin <URL>`	Conectar repositorio local con un remoto.
`git push origin main`	Subir cambios al repositorio remoto.
`git pull origin main`	Descargar cambios del remoto al local.
`git fetch`	Obtener información del remoto sin fusionar cambios.

Diferencias entre git fetch y git pull

Comando	Función	Cuándo Usarlo
`git fetch`	Descarga los cambios pero NO los aplica.	Si quieres revisar cambios antes de aplicarlos.
`git pull`	Descarga y fusiona cambios automáticamente.	Si confías en los cambios remotos.

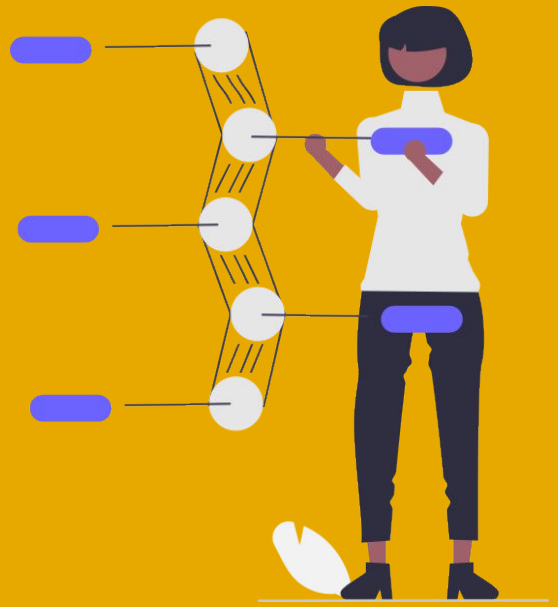
¿Cuál es la Diferencia entre Clone y Fork?

Comando	Función	Uso Común
`git clone`	Copia un repositorio remoto en local.	Usado para trabajar en un repositorio propio.
Fork	Crea una copia de un repositorio en una cuenta personal (GitHub, GitLab).	Ideal para contribuir a proyectos open-source.

Tipos de Flujos de Trabajo en Git

Flujo	Descripción	Ejemplo de Uso
Centralizado	Todos trabajan en la misma rama principal.	Proyectos pequeños.
Git Flow	Uso de ramas `develop`, `feature`, `release` y `hotfix`.	Equipos medianos y grandes.
GitHub Flow	Trabajo con `main` y ramas de `feature`.	Desarrollo ágil y CI/CD.

Resumen de lo aprendido 🧑‍🎓



Adalid aspasia

Resumen de lo aprendido

- Los SCV permiten rastrear cambios en el código, mejorar la colaboración y evitar conflictos.
- Git es un sistema distribuido que facilita la gestión de versiones mediante commits y ramas.
- Plataformas como GitHub centralizan repositorios y optimizan el trabajo en equipo.
- Flujos de trabajo eficientes con Git incluyen merge, rebase, stash y uso estratégico de branches.

GRACIAS POR TU ATENCIÓN

Nos vemos en la próxima clase

Chile
Adalid
aspasia
LA FORMACIÓN DE TU FUTURO