2025

Nivel básico

DOMINA GIT

CONTROL DE VERSIONES DE PRINCIPIO A FIN

Ariadna García
ariadnagrc

ÍNDICE

¿Qué aprenderás en esta guía?

Este material es un **tutorial básico** y paso a paso diseñado para ayudarte a subir archivos a un repositorio Git. El objetivo es enseñarte el flujo de trabajo más común en Git, ideal para proyectos personales o para quienes están empezando a aprender sobre el control de versiones.

En este tutorial, nos enfocaremos en acciones simples como crear un repositorio, agregar archivos, hacer commits y subir esos cambios a un repositorio remoto. No cubriremos conceptos avanzados como el uso de ramas (branches) o fusiones (merge), ya que este enfoque está pensado para quienes recién comienzan y prefieren entender primero lo básico.

A lo largo de este tutorial, aprenderás cómo:

- ***Inicializar un repositorio*** *Git local.*
- *Agregar y hacer* ***seguimiento de archivos****.*
- ***Realizar commits*** *con mensajes descriptivos.*
- ***Subir esos commits*** *a un repositorio remoto, como GitHub o GitLab.*
- ***Actualizar tu repositorio*** *local con los cambios más recientes del repositorio remoto utilizando git pull.*

Este flujo de trabajo te permitirá gestionar y versionar tu código de manera sencilla, sin necesidad de configuraciones complejas. Una vez que domines estos pasos, podrás pasar a conceptos más avanzados como las ramas y la colaboración en equipo.

Para que no te pierdas con los términos técnicos, en la siguiente página tienes una breve guía de vocabulario que te servirá de referencia a lo largo del material.

1 CONCEPTOS CLAVE

Repositorio

Un repositorio es un contenedor que almacena todos los archivos de un proyecto y su historial de cambios. Puede ser local (en tu máquina) o remoto (en un servidor como GitHub, GitLab, etc.).

Commit

Un commit es una instantánea de los cambios en tu repositorio. Representa un punto en el historial de tu proyecto con un mensaje descriptivo. Cada commit tiene un identificador único (hash).

Staging Area

El staging area es donde se colocan los archivos que vas a incluir en tu próximo commit. Usas git add para mover archivos al área de preparación antes de hacer el commit.

Origin

Origin es el nombre predeterminado para el repositorio remoto desde el que se clonó un repositorio. Se usa comúnmente para hacer git pull y git push hacia y desde el remoto.

Remote

Un remote es una versión del repositorio almacenada en un servidor. Los remotos se usan para sincronizar tu repositorio local con el repositorio en línea. Origin es el remoto predeterminado.

Clone

Es el comando que se usa para copiar un repositorio remoto a tu máquina local. Incluye todos los archivos y el historial de cambios del proyecto.

Pull

Es el comando utilizado para descargar y aplicar los cambios desde el repositorio remoto a tu repositorio local. Combina git fetch (para obtener los cambios) y git merge (para integrarlos en tu rama actual).

2 CONFIGURACIÓN INICIAL DE USUARIO

Configurar usuario y correo electrónico de forma global

Cuando utilizas Git por primera vez, es importante configurar tu nombre y correo electrónico. Esta información se asocia a cada commit que realices, lo que permite identificar quién hizo cada cambio en el proyecto.

```
>_
git config --global user.name "Tu Nombre"
git config --global user.email "tu-email@ejemplo.com"
```

Este comando actualizará el nombre y el email globalmente, lo que significa que se aplicará a todos los repositorios de Git en tu sistema.

Configurar usuario y correo electrónico para un repositorio específico

Si deseas tener configuraciones diferentes para distintos proyectos, puedes cambiar el nombre o correo solo para un repositorio particular.
Navega al directorio de tu repositorio y usa los mismos comandos, pero sin la opción --global:

```
>_
git config user.name "Tu Nombre"
git config user.email "tu-email@ejemplo.com"
```

Verificar los cambios

Para asegurarte de que los cambios se aplicaron correctamente, puedes consultar la configuración actual de Git con el siguiente comando:

```
>_
git config --global --list
```

Editar usuario y correo electrónico

Si necesitas cambiar estos valores, Git permite modificar tanto la configuración global (para todos los repositorios) como la configuración local (para un repositorio específico).

```
git config --global user.name "Nuevo nombre"
git config --global user.email "nuevo-correo@ejemplo.com"
```

o

```
git config user.name "Nuevo nombre"
git config user.email "nuevo-correo@ejemplo.com"
```

Restablecer la configuración de usuario y correo electrónico

Si alguna vez necesitas restablecer el nombre de usuario o el correo electrónico a sus valores predeterminados (o eliminarlos completamente), puedes hacerlo con los siguientes comandos:

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

Con estos pasos, podrás modificar de forma efectiva el nombre de usuario y correo electrónico asociados a tus commits en Git. Esto es especialmente útil cuando trabajas en varios proyectos o si necesitas actualizar tus datos de contacto en los registros de Git.

RECUERDA

Cuando uses --global en comandos como git config, estás aplicando esa configuración a todos los repositorios de tu usuario en el sistema.

3 EDITOR DE TEXTO

Git utiliza un editor de texto para redactar mensajes de commit y resolver conflictos. Por defecto, en muchas instalaciones, Git usa Vim. Si prefieres usar otro editor, puedes cambiarlo.

Cambiar a Visual Studio Code

```
>_
git config --global core.editor "code --wait"
```

Cambiar a Nano

```
>_
git config --global core.editor "nano"
```

Puedes configurar el editor que prefieras (como Sublime Text, Atom, etc.) usando el mismo comando con el nombre del editor.

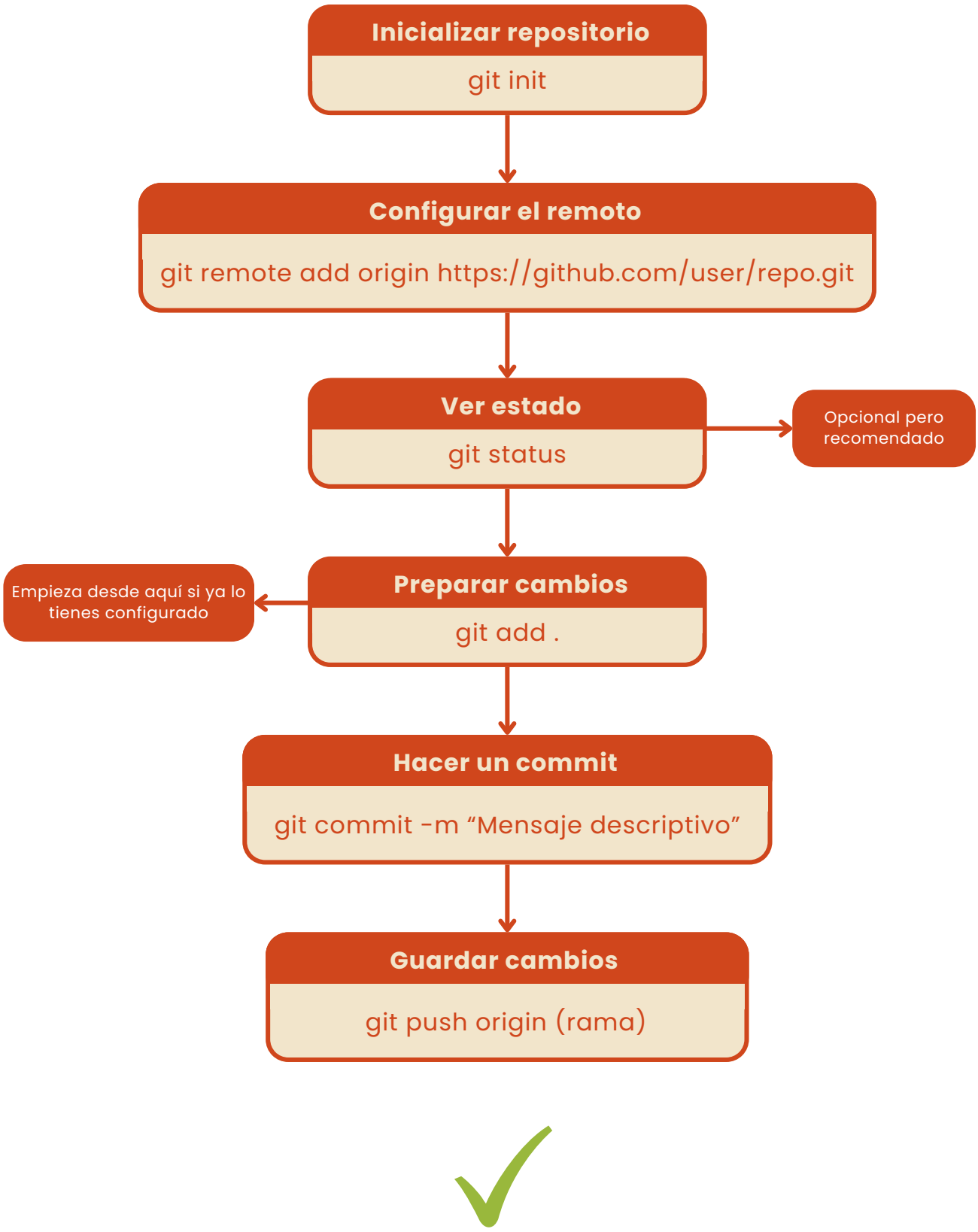
Configuración de la línea de comando

Git permite también personalizar el formato de la salida de los comandos, lo que puede hacer que la información sea más fácil de leer. Tiene una opción para habilitar colores en la salida de los comandos, lo que facilita la lectura.

Para habilitarlo, ejecuta:

```
>_
git config --global color.ui auto
```

4 PROCESO DE SUBIDA



Antes de poder utilizar Git para controlar las versiones de tu proyecto, primero necesitas inicializar un repositorio. Esto le indica a Git que comience a rastrear los cambios en ese directorio.

Inicializar un repositorio

El comando git init se utiliza para comenzar a gestionar un proyecto con Git desde cero. Este es el punto de partida cuando quieres usar control de versiones en un directorio local que todavía no ha sido inicializado como repositorio de Git. Esto se hace creando un subdirectorio oculto llamado .git, el cual contiene toda la información necesaria del repositorio (historial, ramas, configuración, etc.).

Para ello, desde la terminal navega hasta la carpeta dónde está tu proyecto.

```
>_
cd ruta/de/tu/proyecto
```

Una vez dentro de tu proyecto, ejecuta el siguiente comando:

```
>_
git init
```

Después de esto, tu proyecto estará listo para usar Git. Ya puedes empezar a realizar seguimientos de tus archivos y versiones.

Configurar un repositorio remoto

En Git, un repositorio remoto es una versión de tu proyecto que está alojada en internet o en una red. Usualmente, se encuentra en plataformas como GitHub, GitLab, Bitbucket, etc. Esto te permite colaborar con otras personas, sincronizar cambios, y tener una copia de respaldo en la nube.

Cuando creas un nuevo proyecto y lo inicializas con Git usando git init, ese repositorio solo existe en tu máquina local. Para subirlo a la nube (por ejemplo, a GitHub), necesitas decirle a Git dónde está el repositorio remoto.
Aquí es donde entra:

```
>_
git remote add origin <URL-del-repositorio>
```

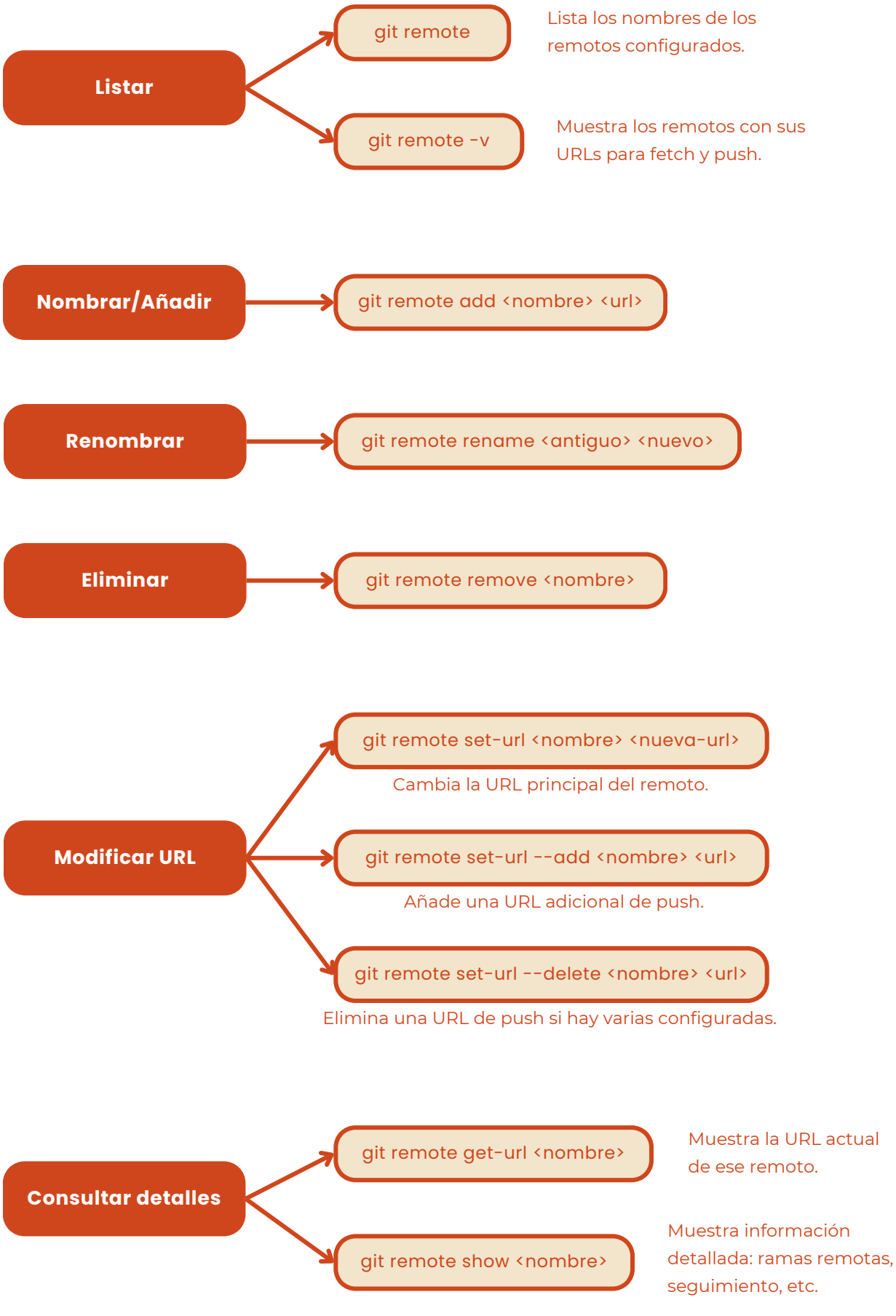
Este comando añade un repositorio remoto a tu proyecto local, y lo nombra como origin.

- **git remote**: indica que vas a trabajar con repositorios remotos.
- **add**: significa que vas a agregar uno nuevo.
- **origin**: es el nombre que le das al remoto. Es una convención universal en Git para referirse al repositorio principal, aunque puedes usar otro nombre si lo deseas.
- **<URL-del-repositorio>**: es el enlace HTTPS o SSH al repositorio remoto (por ejemplo, el que te da GitHub al crear un nuevo repositorio).

Ejemplo

```
>_
git init
git add .
git commit -m "Primer commit"
git remote add origin https://github.com/usuario/mi-proyecto.git
```

Con esto, ya tienes una conexión establecida entre tu proyecto local y el repositorio remoto en GitHub.


Listar
git remote
Lista los nombres de los remotos configurados.
git remote -v
Muestra los remotos con sus URLs para fetch y push.
Nombrar/Añadir
git remote add <nombre> <url>
Renombrar
git remote rename <antiguo> <nuevo>
Eliminar
git remote remove <nombre>
Modificar URL
git remote set-url <nombre> <nueva-url>
Cambia la URL principal del remoto.
git remote set-url --add <nombre> <url>
Añade una URL adicional de push.
git remote set-url --delete <nombre> <url>
Elimina una URL de push si hay varias configuradas.
Consultar detalles
git remote get-url <nombre>
Muestra la URL actual de ese remoto.
git remote show <nombre>
Muestra información detallada: ramas remotas, seguimiento, etc.

Comprobar el estado

El comando git status es uno de los más importantes y más usados en Git. Su función principal es informarte del estado actual del repositorio, permitiéndote ver qué archivos han cambiado, cuáles están preparados para el próximo commit, y cuáles no.

Es el panel de control que te dice en qué punto estás del flujo de trabajo.

```
>_
git status
```

Git analiza el estado actual del directorio de trabajo y del área de preparación (staging area), y muestra:

1. En qué rama estás.
2. Si el directorio está limpio o hay cambios.
3. Qué archivos han sido modificados.
4. Qué archivos están preparados para hacer commit.
5. Qué archivos no están siendo rastreados aún (untracked).

Estado	Significado
Untracked files	Archivos que están en tu carpeta, pero Git no los está siguiendo aún. No se incluirán en el commit hasta que los agregues con git add.
Changes not staged for commit	Archivos modificados que sí están siendo seguidos por Git, pero no se han preparado para commit.
Changes to be committed	Archivos que ya fueron preparados con git add y están listos para ser confirmados con git commit.
Nothing to commit, working tree clean	Todo está sincronizado. No hay cambios pendientes.

Preparación de archivos

El comando git add se utiliza para preparar archivos para ser confirmados (committed). En términos técnicos, lo que hace es añadir cambios al área de staging (zona de preparación), donde Git espera los archivos que serán incluidos en el próximo commit.

Su sintaxis básica para añadir un archivo o varios específicos es:

```
>_
git add <archivo>
```

También puedes añadir todos los archivos de esta forma:

```
>_
git add .
```

CUIDADO!

Git add . también incluirá archivos nuevos (untracked), así que revisa antes con git status.

Si lo prefieres, añadir solo los archivos modificados o eliminados, pero no nuevos:

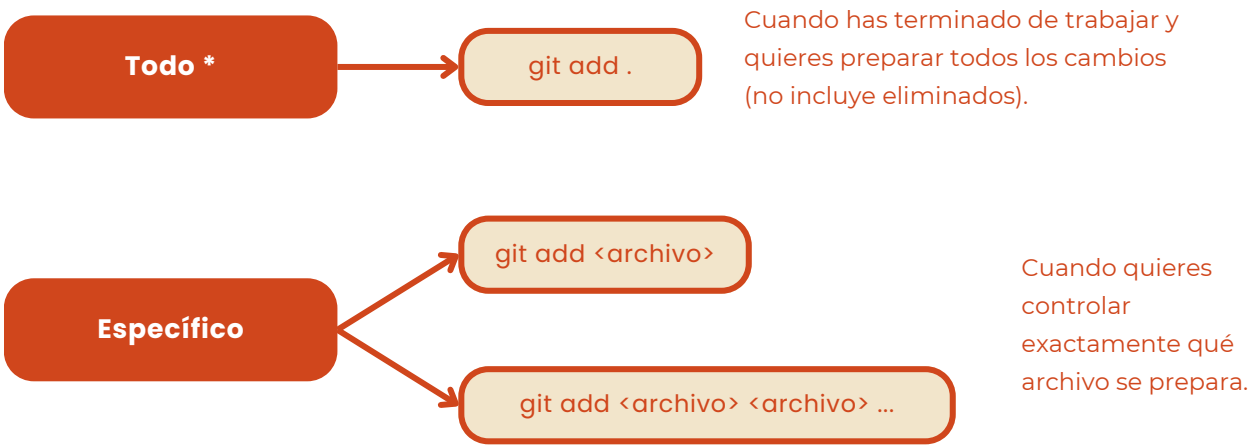
```
>_
git add -u
```

O por el contrario añadir todos los archivos, incluyendo los eliminados:

```
>_
git add -A
```

Por último, tenemos la opción de añadir por partes de forma interactiva:

```
>_
git add -p
```



RECUERDA

git add no guarda los cambios definitivamente: solo los prepara.
Para completar el proceso, debes hacer un commit con git commit.

Guardar estado del proyecto (commit)

Después de preparar los archivos con git add, el siguiente paso en el flujo de trabajo de Git es crear un commit, es decir, guardar un “snapshot” o fotografía del estado actual del proyecto. Este paso registra oficialmente los cambios en el historial del repositorio local.

Incluye:

- *Los archivos preparados.*
- *Un mensaje descriptivo del cambio.*
- *Un identificador único (hash SHA-1).*
- *Información del autor y la fecha.*

Para hacer un commit, de forma general se utiliza:

```
>_
git commit -m "Mensaje del commit"
```

-m: indica que vas a incluir un mensaje de una sola línea justo después.

Si necesitas escribir un mensaje más extenso y detallado puedes usar el siguiente comando, que abrirá el editor de texto preterminado:

```
>_
git commit
```

Puedes omitir git add, añadiendo automáticamente todos los archivos modificados o eliminados que ya están siendo rastreados (pero no nuevos archivos):

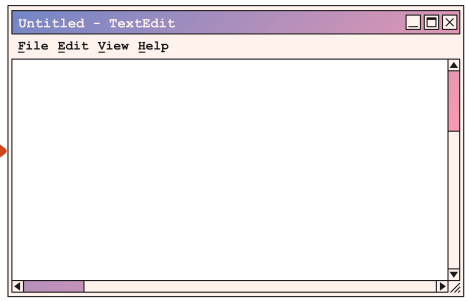
```
>_
git commit -a -m "Mensaje"
```

También puedes cancelar un commit anterior si este ha sido reciente:

```
>_
git reset --soft HEAD~1
```

Mensaje corto → `git commit -m “Mensaje”`

Abrir editor → `git commit` →



Añadir archivos + mensaje → `git commit -a -m "Mensaje"`

Revertir último commit → `git reset --soft HEAD~1`

Deja los cambios en staging.

RECUERDA

No se puede hacer git commit sin haber usado antes git add (salvo con -a). Cada commit es una unidad lógica de cambio. Es mejor hacer varios commits pequeños y bien descritos que uno grande y genérico.

Guardar y añadir cambios

Una vez que tus cambios están confirmados localmente con git commit, el siguiente paso habitual es compartirlos con los demás o guardarlos en un servidor remoto. Para eso existe git push. Este comando sube tus commits desde tu repositorio local al repositorio remoto (como GitHub, GitLab, Bitbucket, etc.).

Cuando haces el primer push de una rama nueva, necesitas indicarle a Git que quieres "rastrear" esa rama con la rama remota. Para eso se usa el modificador -u (o --set-upstream):

```
>_
git push -u origin main
```

Esto le dice a Git: “a partir de ahora, esta rama local main debe estar vinculada con la rama main del repositorio remoto origin”.

Cuando tu rama local está configurada para rastrear una rama remota, puedes usar simplemente:

```
>_
git push
```

Si no es así o estás trabajando con una nueva rama, debes especificar el remoto y la rama, como en git push origin nombre-de-la-rama. Esto se aplica cuando empujas cambios a una rama remota diferente o sin configuración previa.

```
>_
git push <remoto> <rama>
```

- **<remoto>**: normalmente origin (nombre por defecto del repositorio remoto).
- **<rama>**: como main o master, según la rama donde estés trabajando.

Ejemplo

```
>_
git push origin main
```

Esto sube tus commits en la rama main al repositorio remoto llamado origin.

Primer push	`git push -u origin <rama>`	
Push vinculado	`git push`	Sube a la rama vinculada.
Push específico	`git push origin <rama>`	Sube la <rama> al remoto origin
Todas las ramas	`git push --all origin`	Sube todas las ramas locales.
Etiquetas	`git push origin --tags`	Sube todas las etiquetas.
Eliminar	`git push origin --delete rama`	Elimina una rama del remoto.
 **Forzar**	`git push --force`	Fuerza el push, sobrescribiendo historial. Usar esto puede sobrescribir cambios de otros. Úsalo solo si sabes lo que haces.
Forzar seguro	`git push --force-with-lease`	Fuerza el push, pero verifica que nadie haya subido nada antes
Sin forzar	`git push --atomic`	Sube varias ramas/refs como una sola operación segura

5 CLONACIÓN Y ACTUALIZACIÓN

Una vez que tienes tu repositorio local configurado, es importante mantenerlo actualizado con los últimos cambios realizados.

Aunque estés trabajando de manera individual, es posible que desees mantener tu copia local al día con cambios previos, como correcciones hechas en otro dispositivo, o simplemente para asegurarte de que tu repositorio esté limpio y sin errores. Sin embargo, el uso más común ocurre en entornos colaborativos. Si varias personas están trabajando en el mismo proyecto, cada uno debe asegurarse de traer los cambios más recientes antes de añadir los suyos. De esta manera, se evitan conflictos y se trabaja siempre con la versión más actual del proyecto.

Git clone

Si es la primera vez que vas a trabajar en un proyecto, necesitarás usar git clone para copiar el repositorio remoto a tu máquina local. Esto descargará todos los archivos y el historial de cambios, permitiéndote comenzar a trabajar en el proyecto.

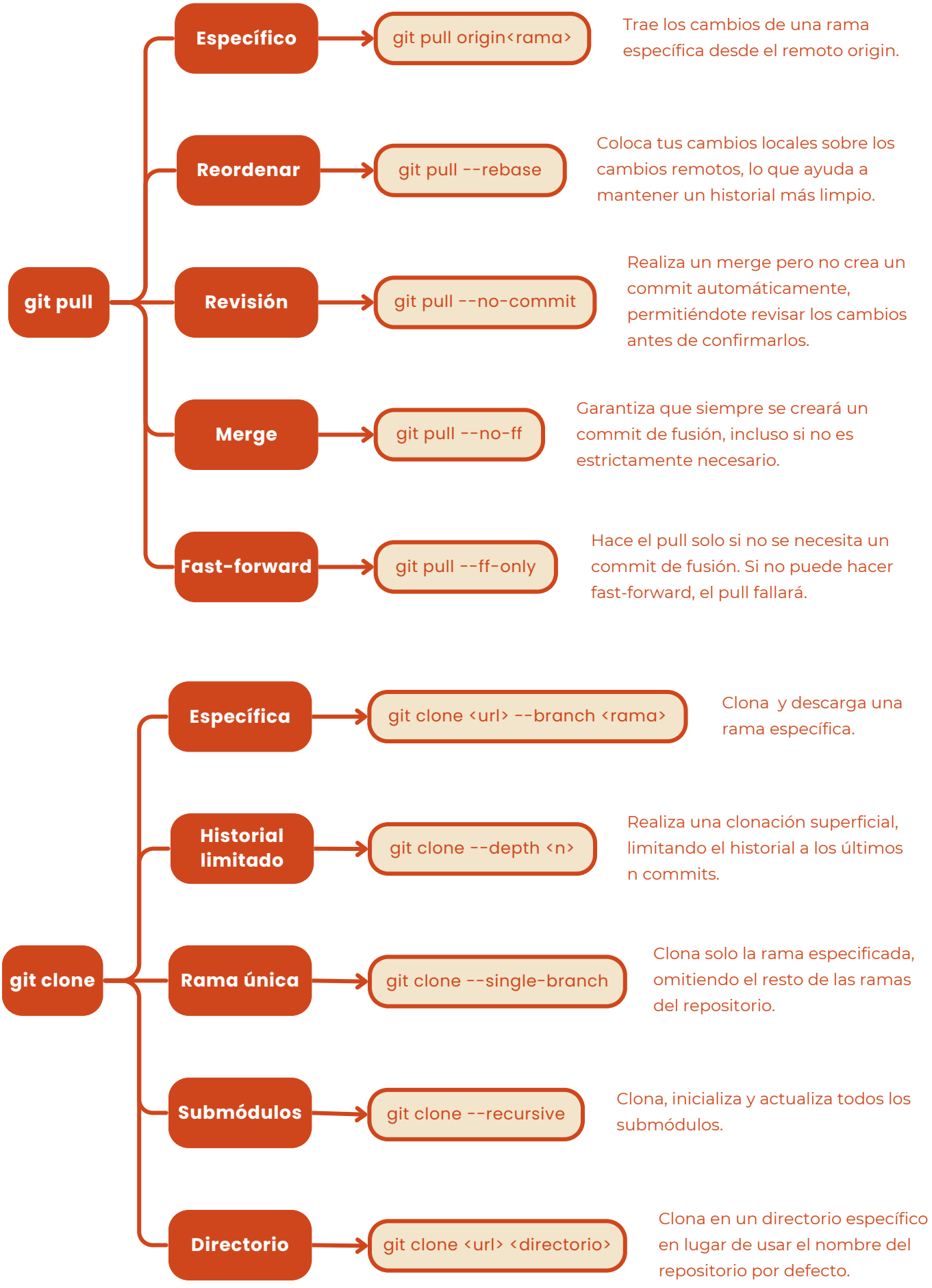
```
>_
git clone https://github.com/usuario/proyecto.git
```

Este paso solo es necesario cuando comienzas a trabajar en un proyecto y no tienes una copia local.

Git pull

Se usa después de clonar el repositorio o cuando ya tienes una copia local. Este comando se utiliza para traer los cambios más recientes desde el repositorio remoto hacia tu repositorio local, asegurando que trabajes con la versión más actualizada del proyecto.

```
>_
git pull origin main
```


git pull
Específico
git pull origin<rama>
Trae los cambios de una rama específica desde el remoto origin.
Reordenar
git pull --rebase
Coloca tus cambios locales sobre los cambios remotos, lo que ayuda a mantener un historial más limpio.
Revisión
git pull --no-commit
Realiza un merge pero no crea un commit automáticamente, permitiéndote revisar los cambios antes de confirmarlos.
Merge
git pull --no-ff
Garantiza que siempre se creará un commit de fusión, incluso si no es estrictamente necesario.
Fast-forward
git pull --ff-only
Hace el pull solo si no se necesita un commit de fusión. Si no puede hacer fast-forward, el pull fallará.
git clone
Específica
git clone <url> --branch <rama>
Clona y descarga una rama específica.
Historial limitado
git clone --depth <n>
Realiza una clonación superficial, limitando el historial a los últimos n commits.
Rama única
git clone --single-branch
Clona solo la rama especificada, omitiendo el resto de las ramas del repositorio.
Submódulos
git clone --recursive
Clona, inicializa y actualiza todos los submódulos.
Directorio
git clone <url> <directorio>
Clona en un directorio específico en lugar de usar el nombre del repositorio por defecto.

6 HISTORIAL DE CAMBIOS

El comando **git log** se usa para ver el historial de commits en tu repositorio.
Es como una lista cronológica que muestra cada versión guardada del proyecto.

```
>_
git log
```

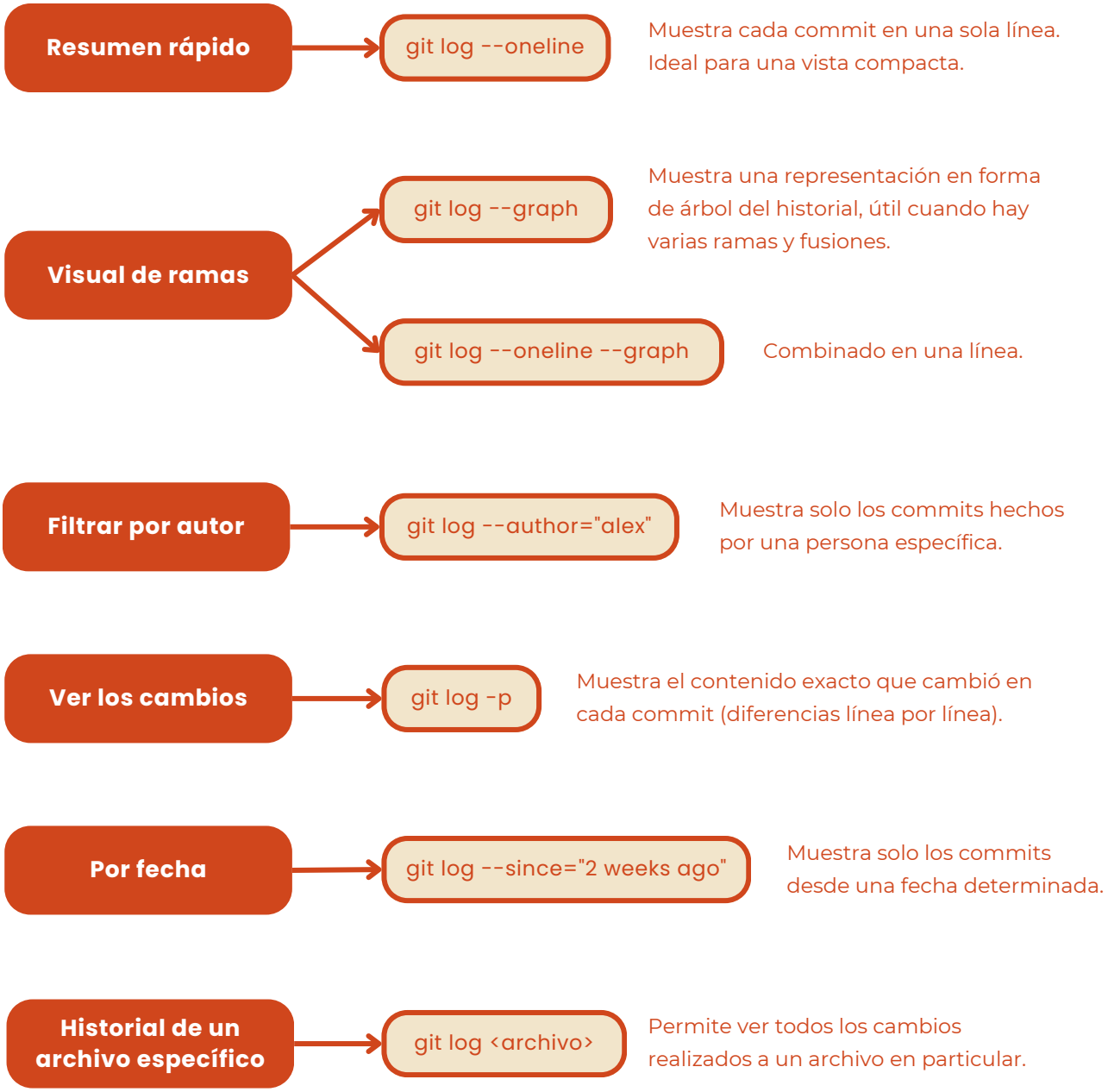
Cuando lo ejecutas, verás información como:

- *El hash del commit (una especie de ID único).*
- *El nombre y correo del autor.*
- *La fecha del commit.*
- *El mensaje que se escribió al hacer ese commit.*

Esto puede ayudarte a:

- *Ver quién hizo qué cambio y cuándo.*
- *Volver a un estado anterior si algo salió mal.*
- *Revisar el progreso del proyecto con claridad.*

```
>_
commit a1b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6q7r8s9t
Author: Ariadna García <ariadna@example.com>
Date:   Thu Apr 25 10:32:11 2025 +0200

    Corrige bug en el formulario de contacto

commit b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6q7r8s9t0u
Author: Alejandro Aránguez <alejandro@example.com>
Date:   Wed Apr 24 18:11:50 2025 +0200

    Añade estilos responsive al menú de navegación

commit c3d4e5f6g7h8i9j0k1l2m3n4o5p6q7r8s9t0u1v
Author: Enara Bonilla <enara@example.com>
Date:   Tue Apr 23 09:45:03 2025 +0200

    Inicializa proyecto con estructura base y archivos esenciales
```


Resumen rápido
git log --oneline
Muestra cada commit en una sola línea. Ideal para una vista compacta.
Visual de ramas
git log --graph
Muestra una representación en forma de árbol del historial, útil cuando hay varias ramas y fusiones.
git log --oneline --graph
Combinado en una línea.
Filtrar por autor
git log --author="alex"
Muestra solo los commits hechos por una persona específica.
Ver los cambios
git log -p
Muestra el contenido exacto que cambió en cada commit (diferencias línea por línea).
Por fecha
git log --since="2 weeks ago"
Muestra solo los commits desde una fecha determinada.
Historial de un archivo específico
git log <archivo>
Permite ver todos los cambios realizados a un archivo en particular.

7 ERRORES FRECUENTES

⚠ fatal: The current branch 'main' has no upstream branch.

Tu rama local no está conectada a una rama remota, por lo que Git no sabe dónde subir tus cambios. Vincula tu rama local a la rama remota ejecutando:

```
>_
git push -u origin main
```

⚠ error: failed to push some refs to '...'

Tu repositorio remoto tiene cambios que no tienes en local, y Git no puede sincronizar los historiales. Para solucionarlo, primero descarga los cambios remotos antes de hacer push:

```
>_
git pull origin main --rebase
git push
```

En Git, **rebase** es un comando que te permite reaplicar cambios de una rama encima de otra, reescribiendo el historial de commits. Su uso principal es mantener un historial más lineal y limpio, especialmente cuando estás trabajando con ramas.

Si no quieres usar rebase, puedes hacer un merge:

```
>_
git pull origin main
git push
```

⚠ Some refs were not updated o rejected

El push fue rechazado porque el historial de tu rama local es diferente al del remoto (hay commits en el servidor que no tienes). Sincroniza tu rama local con la remota usando:

```
>_
git fetch origin
git merge origin/main
# o con rebase:
git rebase origin/main
git push
```

⚠ remote: Repository not found.

La URL del repositorio remoto es incorrecta, el repositorio no existe, o no tienes acceso. Verifica que el repositorio exista y tu usuario tenga permisos. Revisala con:

```
>_
git remote -v
```

Si es incorrecta, corrígela con:

```
>_
git remote set-url origin <URL-correcta>
```

⚠ nothing to commit, working tree clean

No has hecho ningún cambio desde el último commit, o los cambios aún no están preparados con git add. Si hiciste cambios pero no los añadiste:

```
>_
git add .
git commit -m "mensaje"
```

Si no hiciste cambios, no necesitas hacer commit.

⚠ Aborting commit due to empty commit message.

Intentaste hacer un commit sin indicar mensaje, y Git aborta el proceso. Incluye el mensaje directamente:

```
>_
git commit -m "mensaje descriptivo"
```

⚠ fatal: not a git repository (or any of the parent directories): .git

Estás intentando usar comandos de Git en un directorio que no está inicializado como repositorio. Si querías trabajar en un repositorio ya existente, navega a la carpeta correcta. Si quieres iniciar un nuevo repositorio, usa:

```
>_
git init
```

⚠ error: pathspec 'archivo' did not match any file(s) known to git

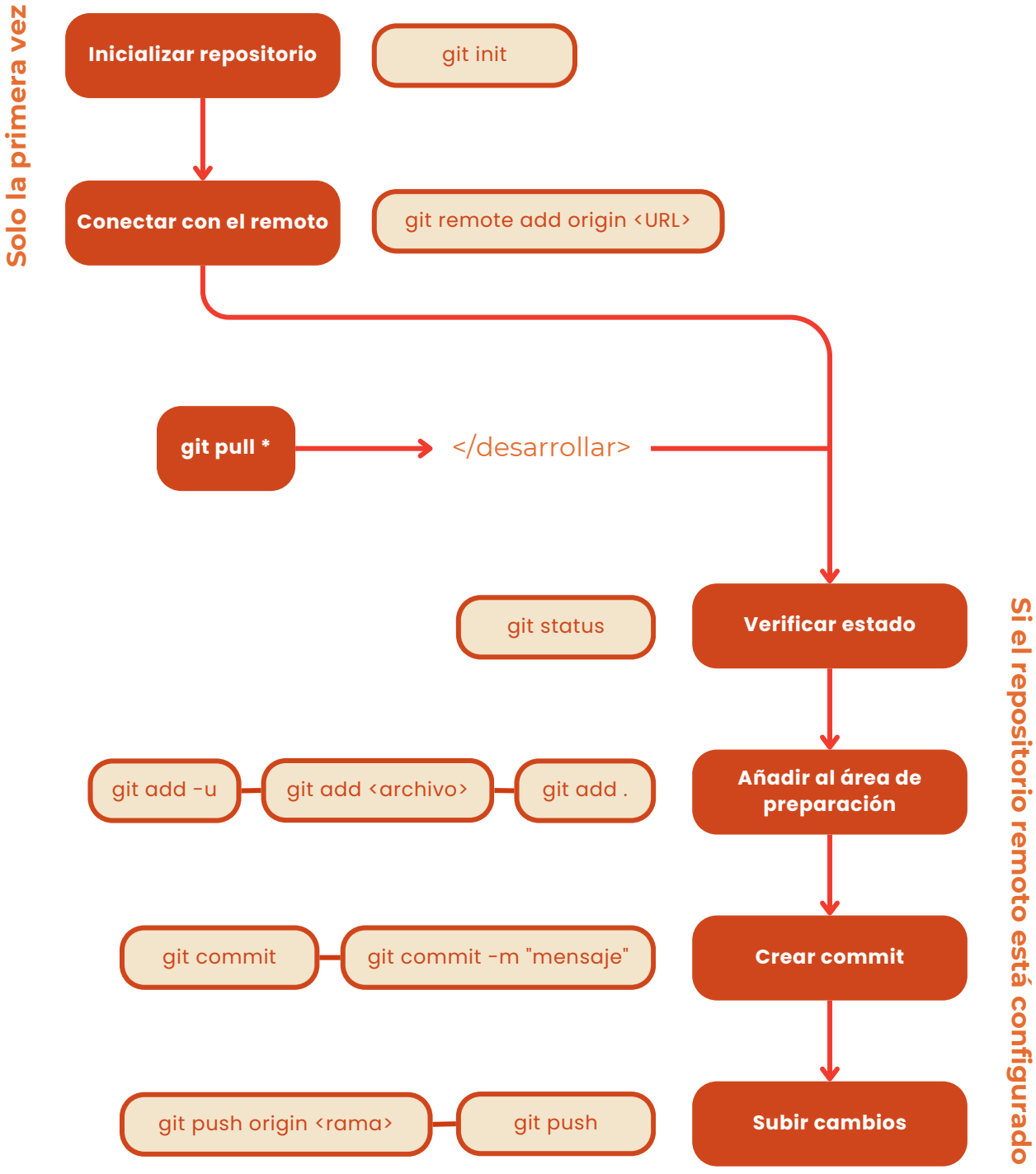
Estás intentando hacer add, checkout o alguna operación sobre un archivo que no existe o no ha sido creado aún. Asegúrate de que el archivo exista. Si lo acabas de crear, guarda los cambios y asegúrate de estar en el directorio correcto.

⚠ Changes not staged for commit

Has hecho cambios en archivos que Git detecta, pero que aún no has preparado con git add. Por lo tanto, si haces git commit, no se incluirán. Prepáralo correctamente:

```
>_
git add .
git commit -m "mensaje"
```

8 ESQUEMA FINAL DE REPASO



* git pull se usa cuando necesitas actualizar tu repositorio local con los cambios más recientes del remoto, ya sea antes de empezar a trabajar o para mantenerlo sincronizado en un entorno colaborativo. También se utiliza para fusionar los cambios descargados con git fetch.