Git

Sistema de control de versiones distribuido

¿Qué es Git?

Git nos permite tener control sobre los cambios en nuestros proyectos. Podemos ver el historial de cambios, crear ramas para trabajar en nuevas funcionalidades y fusionarlas con la rama principal.

Es la herramienta de control de versiones más utilizada en la actualidad.

¿Por qué usar Git?

Si has estado programando por un tiempo, probablemente hayas experimentado la frustración de perder código, de no poder volver a una versión estable del mismo o de no poder colaborar con otros desarrolladores de manera eficiente.

Git resuelve estos problemas y nos permite trabajar de forma más segura en nuestros proyectos.

¿Cómo funciona Git?

Git posee varias areas de trabajo:

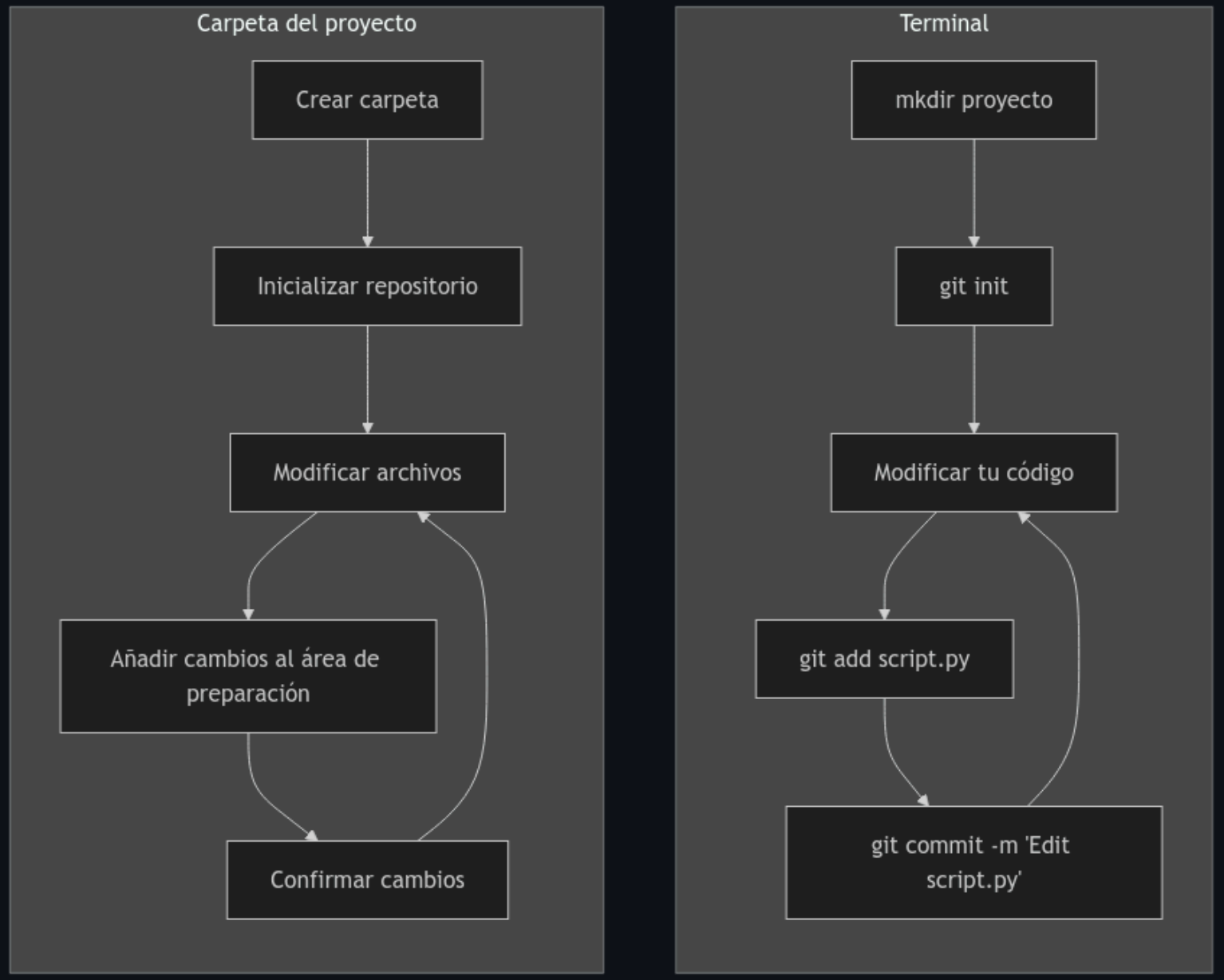
- **Directorio de trabajo**, donde estamos escribiendo el código
- **Área de preparación**, zona temporal donde se almacenan los cambios antes de confirmarlos
- **Repositorio**, donde almacenamos los cambios confirmados

Cada confirmación es un punto en la historia del proyecto.

Flujo de trabajo

Vamos a analizar el flujo de trabajo en Git:

1. Creamos una carpeta para el proyecto.
2. Inicializamos un repositorio de Git con `git init`.
3. Hacemos cambios en el repositorio, crear archivos, modificarlos, eliminarlos, etc.
4. Añadimos los cambios al área de preparación con `git add <file>`.
5. Confirmamos los cambios con `git commit -m <mensaje>`.


Carpeta del proyecto
Crear carpeta
Inicializar repositorio
Modificar archivos
Añadir cambios al área de preparación
Confirmar cambios
Terminal
mkdir proyecto
git init
Modificar tu código
git add script.py
git commit -m 'Edit script.py'

Estructura de cambios en el repositorio

Los cambios se representan como una rama en la historia del proyecto:

◇ .gitignore

En ocasiones, no queremos que Git rastree ciertos archivos o carpetas. Por ejemplo, archivos temporales o de configuración como `.env`.

Para evitar que Git rastree estos archivos, podemos crear un archivo `.gitignore` en la raíz del proyecto y añadir el nombre de los archivos o carpetas que queremos que git ignore.

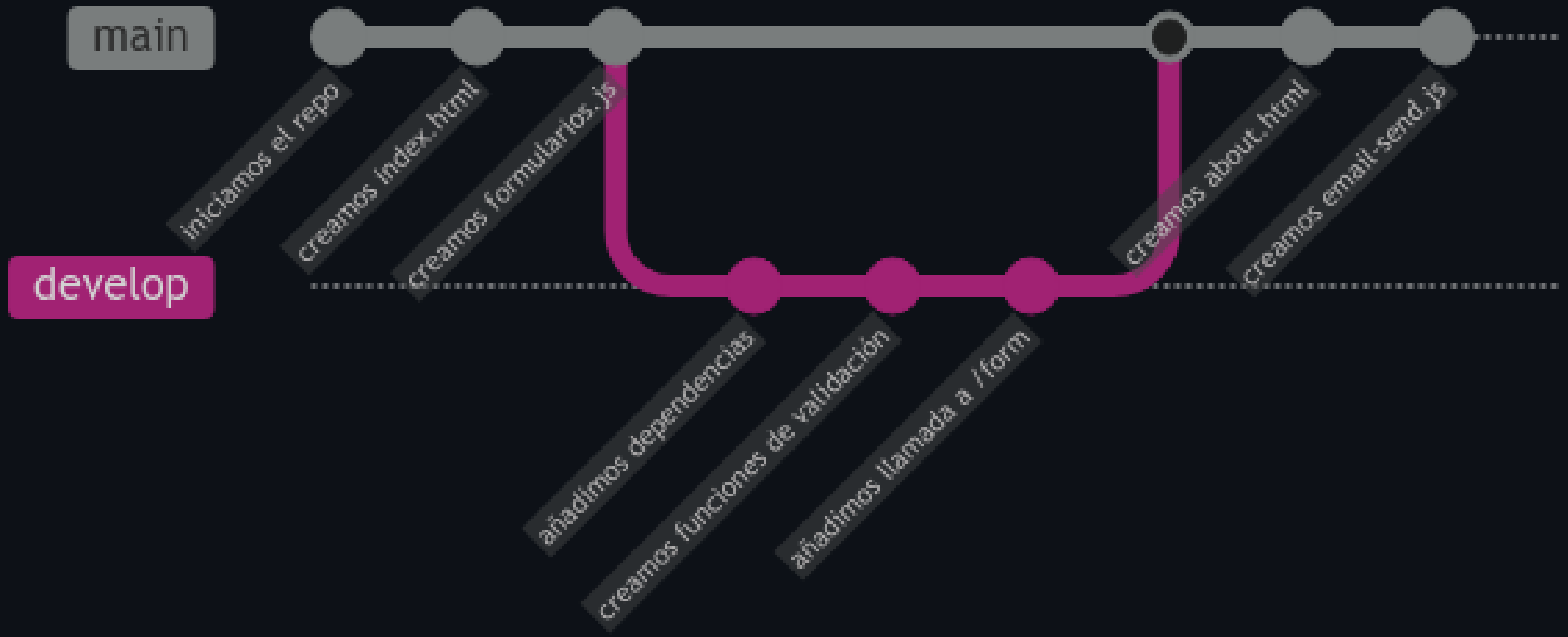
En `.gitignore` deberíamos añadir:

- Archivos de configuración, como `.env` en proyectos de Node.js o Python.
- Dependencias, como `node_modules` en proyectos de Node.js, `venv` en proyectos de Python o `vendor` en proyectos de Go.
- Archivos temporales, como `.DS_Store` en macOS o `Thumbs.db` en Windows.
- Archivos de logs, como `*.log` para ignorar todos los archivos con extensión `.log`.

Ramas en Git

Las ramas son un aspecto fundamental a la hora de trabajar con repositorios. Nos permiten separar el flujo de trabajo en diferentes líneas de tiempo.

Por defecto, Git crea una rama principal llamada `main`. Sin embargo, podemos crear nuevas ramas para trabajar en nuevas funcionalidades sin afectar el flujo de trabajo principal.


main
develop
iniciamos el repo
creamos index.html
creamos formularios.js
añadimos dependencias
creamos funciones de validación
añadimos llamada a /form
creamos about.html
creamos email-send.js

◇

Imagina que estas trabajando en una página web, debes crear un formulario en la pagina principal y una sección de noticias en una nueva página.

Tras crear tu proyecto, comienzas a trabajar en la rama `main` directamente. Sin embargo, tu jefe te llama y te pide con urgencia la sección de noticias y una sección de contacto que no tenías prevista.

Si no usas ramas, en este momento tendrás en tu proyecto mezclado código de la página principal y de la sección de noticias.

Si en lugar de una página , la situación se complica, ya que la lógica implementada a medias puede afectar a otras partes del código.

Las ramas te salvarán de estos problemas.

Merge

Una vez que hemos terminado de trabajar en una rama, podemos fusionarla con la rama principal. Esto se conoce como un `merge`.

Por ejemplo, si hemos trabajado en la rama `feature` y queremos fusionarla con la rama `main`, debemos hacer lo siguiente:

1. Cambiarnos a la rama `main` con `git checkout main`.
2. Fusionar la rama `feature` con `git merge feature`.

Resolución de conflictos

Cuando trabajamos en un proyecto con varias personas, es posible que dos o más personas modifiquen el mismo archivo al mismo tiempo. En este caso, Git no puede decidir por sí solo qué cambios son correctos. Deberemos ponernos de acuerdo y decidir qué cambios se mantienen y cuáles se descartan.

Como resolver conflictos

La primera vez que git nos indica un conflicto entre dos ramas puede ser confuso. Pero no te preocupes, es algo normal y normalmente no tenemos muchas mas opciones aparte de:

- Aceptar los cambios de la rama actual.
- Aceptar los cambios de la rama a fusionar.
- Modificar los cambios manualmente porque te interesa una mezcla de ambos o ninguno.

Comandos git

Configuración inicial

- `git config --global user.name "Tu nombre"`
- `git config --global user.email "tuemail@gmail.com"`
- `git config --global core.editor "nano"`
- `git config --global --list`

Flujo de trabajo

- `git init` para crear un repositorio en la carpeta actual.
- `git add <file>` para añadir un archivo al área de preparación.
- `git add .` para añadir todos los archivos al área de preparación. Cuidado con esto, intenta agrupar los cambios en commits coherentes.
- `git commit -m "Mensaje del commit"` para confirmar los cambios. El mensaje debería ser descriptivo.
- `git status` para ver el estado del repositorio.
- `git log` para ver el historial de cambios.

Ramas

- `git branch` para ver las ramas del repositorio.
- `git branch <nombre>` para crear una nueva rama.
- `git checkout <rama>` para cambiar a una rama.
- `git merge <rama>` para fusionar una rama con la rama actual.
- `git branch -d <rama>` para eliminar una rama.
- `git branch -D <rama>` para forzar la eliminación de una rama.
- `git branch -m <nombre>` para renombrar la rama actual.
- `git branch -a` para ver todas las ramas, incluidas las remotas.

◇ Logs y otros

- `git log` para ver el historial de cambios.
- `git log --oneline` para ver el historial de cambios de forma resumida.
- `git log --graph` para ver el historial de cambios en forma de grafo.
- `git log --graph --oneline` para ver el historial de cambios en forma de grafo y de forma resumida.
- `git diff` para ver los cambios realizados en el repositorio.
- `git diff <rama>` para ver los cambios realizados en una rama.
- `git diff <commit> <commit>` para ver los cambios realizados entre dos commits.
- `git reset --hard <commit>` para volver a un commit anterior. **CUIDADO** al usar esto, no es reversible, no se puede recuperar nada, solo es recomendable si en el commit que vas a borrar hay algo que **no** deberia estar en el repositorio, como contraseñas, claves privadas, etc.

◇

Repositorios remotos

Podemos trabajar con repositorios remotos para colaborar con otros desarrolladores o para tener una copia de seguridad de nuestro proyecto.

Esto es especialmente útil, nos permite no tener todos nuestros proyectos en un solo lugar y poder trabajar en ellos desde cualquier lugar de forma más segura y eficiente.

Algunos servicios de repositorios remotos son:

- GitHub
- GitLab
- Bitbucket

Self-hosted

Tambien disponemos de varias alternativas para alojar nuestros propios gestores de repositorios, como:

- Gitea
- GitLab
- Gogs

Github

Github es el servicio de repositorios remotos más utilizado en la actualidad. Nos permite colaborar con otros desarrolladores, tener un control de versiones de nuestros proyectos y tener una copia de seguridad de los mismos.

Al ser tan útil para desarrollo y ser tan ampliamente utilizado, es una herramienta imprescindible para cualquier desarrollador.

Tambien puede ser un vector de ataque, por lo que es importante tener en cuenta la seguridad de nuestros repositorios.

Empezando en Github

Si quieres trabajar con Github, necesitarás una cuenta. Puedes crear una en Github.

Además, la forma más cómoda de trabajar con Github, es usar su herramienta CLI oficial, esto no es obligatorio, pero nos facilitará mucho el trabajo y la cantidad de comandos que necesitamos para trabajar con Github.

Flujo completo de trabajo con github y gh

1. Iniciar sesión en Github con `gh auth login`, selecciona `http login` y sigue las instrucciones.
2. Crea una carpeta para tu proyecto y entra en ella.
3. Inicializa un repositorio con `git init`.
4. Añade los archivos al área de preparación con `git add .`.
5. Confirma los cambios con `git commit -m "Mensaje del commit"`.
6. Crea un repositorio en Github con `gh repo create`.
7. Selecciona `Push existing local repository` y sigue las instrucciones.