git

parte 2: gestion de ramas

Repasamos

- git config: nombre, email
- git init: ir a una carpeta y comenzar con git
- git add: marcar los ficheros para "comitear"
- git status: me dice que ficheros hay para "comitear" y los que no
- git commit: guardo el estado
- git log: muestro el historial de commits

git commit -m "<msg>"

- Para modificar un mensaje de un commit:
  git commit --amend -m "<nuevo_msg>"

Comandos git que veremos hoy

- git branch
- git checkout
- git merge (gestion de conflictos)
- git reset
- git revert

```
git branch
```

Comando `git branch`

Permite gestionar las ramas dentro de un repositorio.

Las ramas son como líneas de tiempo o versiones paralelas de un proyecto, donde puedes trabajar de manera aislada sin afectar a la rama principal u otras ramas.

`git branch`

Entendiendo las ramas en git ...

> Supongamos que tenemos un documento de Word de 50 páginas, por ejemplo, nuestra memoria final del curso.
> Después de una revisión, nos hemos dado cuenta de que tenemos que cambiar lo siguiente:
> - De la sección 1, quitar la parte final y añadirla a la sección 2
> - Juntar la sección 3 y 4, en una sección nueva llama sección 7
> - Renombrar las secciones

¿Cómo gestionarías esto?

```
git branch
```

Una posible solución sería:

1. Guardo el documento actual como "**main**".
2. Creo un nuevo fichero a partir de "**main**" y lo llamo "feature/modifico_seccion_1"
3. Trabajo en ese documento y realizo las modificaciones necesarias.
4. Meto los cambios de "**feature/modifico_seccion_1**" en "**main**"

```
git branch
```

Ahora el documento **main ya tiene solucionado el primer problema. Continuamos con el segundo problema:**

1. Creo un nuevo fichero a partir de "**main**" y lo llamo "**feature/juntamos_secciones_3_4**"
2. Trabajo en ese documento y realizo las modificaciones necesarias.
3. Meto los cambios de "**feature/feature/juntamos_secciones_3_4**" en "**main**"

```
git branch
```

Ahora el documento **main ya tiene solucionado el primer y el segundo problema. Continuamos con el tercer problema:**

1. Creo un nuevo fichero a partir de "**main**" y lo llamo "**feature/rename_sections**"
2. Trabajo en ese documento y realizo las modificaciones necesarias.
3. Meto los cambios de "**feature/rename_sections**" en "**main**"

Mi fichero "main" ya tiene todas las correciones.

```
git branch
```

Traduccion a comandos git:

1. git status => estoy en la rama **main**.
2. git branch feature/modifico_seccion_1 => creo la rama feature/modifico_seccion_1
3. git checkout feature/modifico_seccion_1 => me voy a la rama feature/modifico_seccion_1
4. Trabajo en ese documento y realizo las modificaciones necesarias hasta que esa versión del documento sea correcta.
5. Vuelvo a la rama main (git checkout main)
6. Le digo que fusione main con feature/modifico_seccion_1 (git merge feature/login)

¿Cómo serían el resto de features?

```
git branch
```

Comando `git branch`

- Listar las ramas disponibles:

  `git branch`

- Crear una nueva rama:

  `git branch <nombre_de_rama>`

- Eliminar una rama

  `git branch -d <nombre_de_rama>`

- Renombrar una rama

  `git branch -m <nombre_de_rama>`

```
git checkout
```

Comando `git checkout`

El comando `git checkout` es uno de los más utilizados en Git, ya que permite cambiar entre ramas, restaurar archivos o incluso volver a versiones anteriores de tu proyecto.
Aunque Git ahora recomienda usar `git switch` para cambiar de rama y `git restore` para restaurar archivos, `git checkout` sigue siendo muy popular y es fundamental entender cómo funciona.

`git checkout`

Comando `git checkout`

- Cambiar de rama:
  `git checkout feature/login`
- Crear y cambiar de rama:
  `git checkout -b feature/login`
- Restaurar un fichero a su última versión confirmada:
  `git checkout index.html`
- Volver a un commit anterior:
  `git log`
  `git checkout abcd1234`
- Volver a la rama principal:
  `git checkout main`

`git checkout`

Comando `git checkout`

- Recuperar archivos desde otra rama:
  `git checkout feature/login -- index.html`
- Crear una rama desde un commit específico:
  `git checkout -b feature/login <hash>`
- Restaurar un fichero a su última versión confirmada:
  `git checkout index.html`

```
git merge
```

Comando `git merge`

El comando `git merge` permite combinar los cambios de dos ramas diferentes en un único historial.

`git merge`

Comando `git merge`

1. **Cambiamos a la rama a la que deseamos incorporar los cambios:**
   `git checkout main`

2. Le añadimos los cambios de feature/blabla:
   `git merge feature/blabla`

> Posibles salidas:
> - *Merge made by the 'recursive' strategy.*
>   - Todo ha ido bien => GG
>
> - Auto-merging index.html CONFLICT (content): Merge conflict in index.html Automatic merge failed; fix conflicts and then commit the result.
>   - Tienes que resolver el conflicto en index.html

`git merge`

Resolviendo el conflicto

- Abrimos el fichero que tiene conflicto (index.html):

```
1 <<<<<<< HEAD
2 <h1>Login Page</h1>
3 =======
4 < h1>Login Portal</h1
5 > >>>>>>> feature/login
```

- Traducción:
  - Línea 1: Te indica que en main tiene lo siguiente.
  - Línea 3: Final del contenido en main.
  - Linea 5: Te indica que en feature/login tiene lo anterior.

`git merge`

Resolviendo el conflicto

1. Decidimos con que parte queremos quedarnos:

```
<<<<<<< HEAD
<h1>Login Page</h1>
=======
< h1>Login Portal</h1
> >>>>>>> feature/login
```

2. Eliminamos el resto:

```
<h1>Login Page</h1>
```

3. Confirmamos el cambio:
    1. git add .
    2. git commit -m "fix conflict"

Comando `git reset`

Se usa para deshacer cambios en el historial de commits.

=> prohibido su uso en la asignatura (y en la vida)

Comando `git revert`

A diferencia de `git reset`, que modifica el historial de commits, `git revert` crea un nuevo commit que deshace los cambios de un commit anterior.

Es útil para **mantener el historial intacto** mientras se deshacen cambios.

Comando `git revert`

Ejemplo:

```
git revert abc1234
```

- **abc1234** es el hash del commit que deseas deshacer.
- **Salida esperada**:

> Git creará un nuevo commit que invierte los cambios realizados por el commit `abc1234`, pero sin eliminarlo del historial.