Introducción a Git y GitHub

José Domingo Muñoz

IES Gonzalo Nazareno

Septiembre 2023

Git / GitHub

Git / GitHub

- **Git** es un software de **control de versiones** diseñado por Linus Torvalds.
  - Nos ayuda a llevar registro de los cambios de los ficheros de un repositorio (directorio).
- **GitHub**: Es un servicio que nos permite guardar **repositorios Git** en un servidor remoto.

Instalación de git

- Instalación de git

```
$ apt install git
```

- Configuración de git.

```
git config --global user.name "José Domingo Muñoz"
git config --global user.email josedom@correo.com
```

Uso básico de git

- Creamos un directorio

```
$ mkdir curso-de-git
$ cd curso-de-git
```

- Inicializamos el repositorio

```
$ git init
Inicializado repositorio Git vacío en ...
```

- Se ha creado el directorio `.git`. El fichero de configuración del repositorio en `.git/config`

Añadir ficheros

- Añadimos un nuevo fichero al repositorio:

```
echo "Esto es una prueba">ejemplo.txt
git add ejemplo.txt
```

- Y confirmamos los cambios:

```
git commit -m "He creado el fichero ejemplo.txt"
```

Modificar un fichero

- Al modificar un fichero, podemos confirmar el cambio pero tenemos que usar la opción **a**:

```
nano ejemplo.txt
git commit -am "He modificado el fichero ejemplo.txt"
```

Estado del repositorio

- Estado del repositorio

```
git status
```

- Listado de commits:

```
git log
```

Otras operaciones

- Cambiar nombre a fichero:

```
git mv ejemplo.txt ejemplo2.txt
git commit -am "He cambiado el nombre del fichero"
```

- Eliminar un fichero

```
git rm ejemplo2.txt
git commit -am "He borrado el fichero ejemplo2"
```

Volviendo a versiones anteriores

- Podemos volver al estado de un repositorio en un commit anterior.
- Si queremos volver a un commit anterior sin perder las modificaciones posteriores:

```
git checkout <id del commit>
```

- Para volver a la última versión:

```
git switch -
```

- Si quiero **eliminar** los últimos 3 commits:

```
git reset --hard HEAD~3
```

GitHub

GitHub

- **GitHub** es un servicio web que te permite tener repositorios git remotos en un servidor.
- Nos permite tener sincronizado nuestro repositorio local, con un repositorio remoto.
- Para conectarnos al servidor remoto de GitHub vamos a usar una conexión SSH.
- Necesitamos que en la conexión SSH no nos pidan la contraseña, por lo tanto hay que configurar en el servicio nuestra clave pública (apartado **“SSH keys”** de tu perfil en GitHub).

Creación de repositorios

- Lo más sencillo es crear un repositorio en GitHub y luego clonarlo en tu ordenador.
- Vamos a usar la **URL ssh** para realizar la clonación.
- Utilizaremos la **URL https** para clonar repositorios que no son nuestros. En estos repositorios no podemos hacer modificaciones.
- Para clonarlo:

```
git clone git@github.com:xxxxxxx/xxxxxxx.git
```

- Para comprobar que tienes configurado el repositorio usando la url SSH puedes ver el fichero de configuración en **.git/config**.

Sincronizando repositorio local y remoto

- Para subir los cambios desde nuestro repositorio local al remoto usamos:

```
git push
```

- Para descargar los cambios desde nuestro repositorio remoto al local usamos:

```
git pull
```

Añadiendo un repositorio remoto

- Vamos a crear un repositorio local, pero el nombre de la rama tiene que ser **main**:

```
git init -b main
```

- Si ya tenemos un repositorio, podemos cambiar el nombre de la rama:

```
git branch -M main
```

- Creamos un repositorio vacío (**Sin fichero README.md**) en GitHub.
- Añadimos el repositorio remoto y sincronizamos:

```
git remote add origin git@github.com:josedom24/prueba.git
git push -u origin main
```

Markdown

Markdown

- **Markdown** es un lenguaje de marcas que nos permite estructurar información.
- Se convierte de manera muy sencilla a HTML.
- La extensión de los ficheros markdown suele ser **md**.
- Los repositorios en GitHub suelen tener un fichero **README.md** donde escribimos la descripción del repositorio.
- Si accedemos al repositorio GitHub los ficheros markdown se visualizan como HTML.
- Markdown cheat sheet

Sintaxis Markdown

Título 1	`# Título 1`
Título 2	`## Título 2`
Título 3	`### Título 3`
Negrita:	`**bold text**`
Cursiva:	`*italicized text*`
Cita:	`> blockquote`

Lista ordenada:	`1. First item`
	`2. Second item`
	`3. Third item`

Lista no ordenada:	`* First item`
	`* Second item`
	`* Third item`

Código:	`'code'`
Enlace:	`[title](https://www.example.com)`
Imagen:	`![alt text](image.jpg)`

Trabajando con ramas y uniones

Ramas

- Una **rama** representa una línea independiente de desarrollo, es como crear un nuevo área de trabajo que tendrá su historial propio de commits.
- Para listar las ramas locales ejecuta:

```
git branch
* main
```

- La rama en la que estás trabajando actualmente se señala con un asterisco *.
- GitHub llama a la rama principal **main**. En repositorios antiguos se llama **master**.
- En la rama principal es donde suele encontrarse la **última versión** del repositorio.

Ramificar

- Creamos una nueva rama, para realizar cambios al repositorio (sin tocar el estado de la rama principal).
- Si los cambios son correctos, tenemos que **fusionar** la rama creada con la rama principal.
- Para crear una nueva rama:

```
git branch [rama]
```

- Para pasar a la nueva rama utiliza el comando:

```
git checkout [rama]
```

Ramificar

- La nueva rama tiene los mismos ficheros que en la rama principal.
- Todos los cambios que realices en los ficheros en esta rama no se verán en la rama principal.
- Truco: con el comando **git checkout -b [rama]** se crea una nueva rama y te posicionas en ella.
- Con el comando **git branch -v** se ve el último commit de cada rama. Comprueba que coinciden el último commit en las dos ramas.
- Truco: Puedes usar una extensión de tu shell (bash, zsh,...) para que te muestre en el prompt la rama en la que estás trabajando.

Sincronizar ramas con el repositorio remoto

- Las ramas no se crean automáticamente en GitHub, hay que realizar un push para crearlas en remoto.
- **Nota: origin es el nombre del repositorio remoto.**

```
git push origin [rama]
```

Eliminar una rama

- Es bastante frecuente crear una rama, hacer los cambios que sean necesarios, unirla a una rama principal y después eliminar la rama que se había creado.

```
git branch -d [rama]
```

Unión / fusión de ramas

- Una vez que has trabajado en una rama, lo normal es querer incorporar los cambios a la rama principal. Para **unir/fusionar** una rama a la principal:

```
git checkout main
git merge [rama]
```

Conflictos

- Cuando sólo se han **añadido o eliminado** archivos en una rama, es fácil unirla a la principal.
- Cuando se hacen **modificaciones** en archivos, incluyendo **cambios en los nombres** de los archivos, git detecta esos cambios y los adapta automáticamente, pero a veces surgen **conflictos**.
- Los conflictos aparecen cuando **se ha modificado la misma parte del código en dos ramas diferentes**.

```
cat prueba.txt

<<<<<<< HEAD
Hola cómo estás
=======
hola que tal
>>>>>>> nuevo
```

Pull Request

¿Qué es un Pull Request?

- Un **Pull Request** es una petición que se hace al propietario de un repositorio original para que este último incorpore los cambios que se sugieren.
- Los **Pull Requests** se han convertido en la forma más habitual de colaborar en proyectos de software libre.
- ¿Cómo colaborar en un proyecto de software libre? ¿Qué es un Pull Request?

Pasos para realizar un PR

1. El usuario tiene que hacer un **fork** del repositorio al que quiere contribuir.
2. **Clonar** el repositorio.
3. **Crear una nueva rama**, donde realizaremos los cambios que posteriormente propondremos como cambios.
4. Subimos los cambios a nuestro repositorio, creando una rama en el repositorio remoto.
5. Crear el Pull Requests desde la página de GitHub.
6. El dueño del repositorio al que hemos realizado el PR tendrá que validar el cambio.
7. Y decide si lo **acepta**, haciendo una unión con la rama principal, o lo **rechaza**.

Sincronización de nuestro fork con el repositorio original

- Al hacer el fork del repositorio original hacemos una copia completa en un determinado momento, pero ese repositorio puede ir cambiando.
- Esos cambios no se verán reflejados en nuestro fork de forma automática.
- Antes de proponer un cambio (hacer un Pull Request), tener nuestro fork lo más actualizado posible para que no tengamos problemas que produzcan conflictos.

1. Nos posicionamos en la rama principal del fork.
2. Añadimos un repositorio remoto apuntando al repositorio original (**upstream**).
3. Buscamos los posibles cambios que ha tenido el repositorio original:

```
git fetch upstream
```

4. fusionamos los cambios del repositorio original que están en la rama principal.

```
git merge upstream/main
```