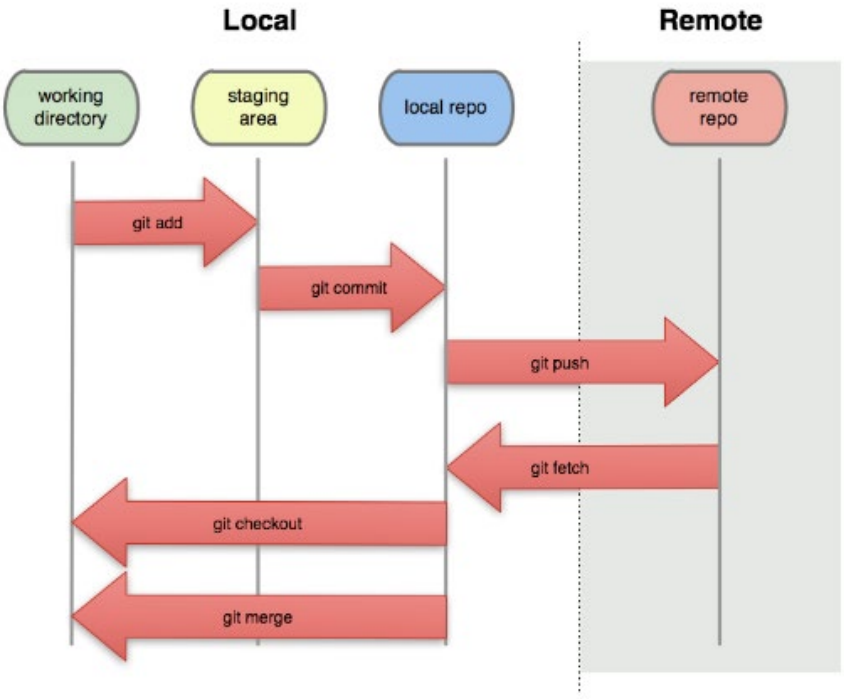
Git/Github

¿Qué es GIT?

Es un software de control de versiones, su propósito es llevar registro de los cambios en archivos de computadora y coordinar el trabajo que varias personas realizan sobre archivos compartidos (También puedes trabajar solo no hay problema. Existe la posibilidad de trabajar de forma remota y una opción es GitHub.



Instalación de GIT

Podemos descargarlo de su web https://git-scm.com/download/win

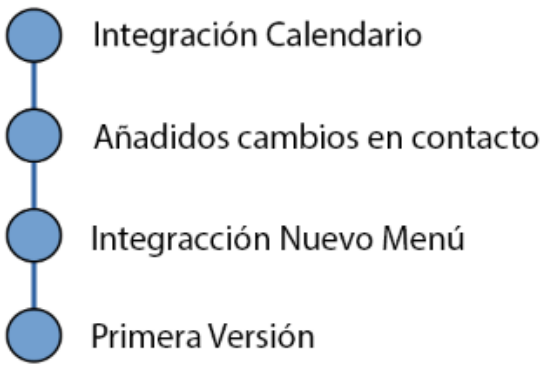
El proceso de instalación es tan sencillo como el de cualquier otro programa en Windows.

Filosofía de trabajo en Git

En Git, al igual que otros **sistemas de control de versiones**, los proyectos se almacenan en repositorios. **Un repositorio** es el lugar donde se almacenan los archivos del proyecto junto a todo el historial de cambios que **Git** gestiona.

A efectos prácticos no es más que la carpeta en la que vas a almacenar los archivos que quieres gestionar.

Git **almacena instantáneas** del proyecto **cada vez que confirmas algún cambio**. Es como si, cada vez que confirmes un cambio, **Git** tome una fotografía de todos los archivos del proyecto y lo almacene tal cual.

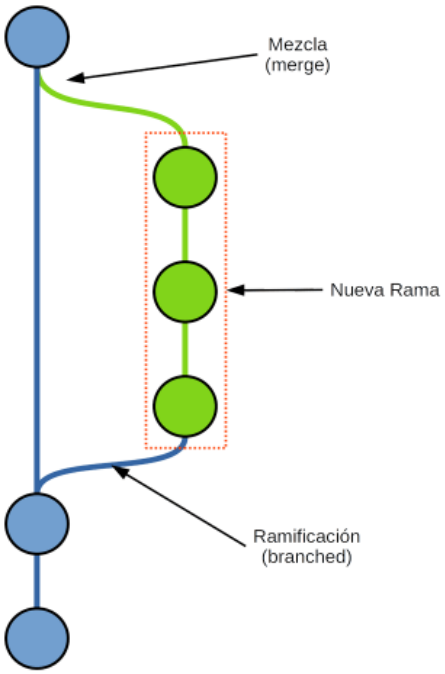


Cada uno de los puntos representa un cambio confirmado (Commit)

A una sucesión de *commits* como esta se le conoce como rama (*branch*). En este caso tienes un proyecto de una sola rama, a esa rama se le denomina ***master.***

Pero, existe un método más práctico y es el de **ramificar el proyecto**.

La idea es tomar el estado actual de la **rama *master*** y crear una nueva rama a partir de ese estado. De esta forma puedes hacer cambios en esta nueva rama y luego, si quieres, puedes combinarla con la rama *master*.



Esto ocurre mucho en proyectos grandes. Donde lo usual es que cada colaborador cree una rama cuando va a realizar alguna modificación.

Luego, cuando sus cambios son comprobados (es decir, cuando sean validados mediante pruebas) se mezclan los resultados.

Estados de archivos en Git

En Git un archivo puede estar en uno de tres estados:

Modificado (*modified*): indica que has modificado el archivo, pero todavía no lo has confirmado a tu base de datos.

Preparado (*staged*): significa que has marcado un archivo modificado en su versión actual para que vaya en tu próxima confirmación.

Confirmado (*commited*): indica que los datos están almacenados de manera segura en tu base de datos local.

Comandos básicos

Versión de git

```
git version
```

```
// Ayuda sobre los comandos
git help
```

Registrar nuevo usuario asociado a git

```
git config --global user.name "Fran"
```

```
git config --global user.email "francisco.morales@escuelarartegranada.com"
```

Creando Nuestro Primer Reposiorio

Una vez localizados en nuestro directorio podremos iniciar nuestro repositorio git. Una vez realizado git init podremos empezar a funcionar.

```
git init
```

Ver que archivos no han sido registrados

```
git init
```

Agregar todos los archivos para que esté pendiente de los cambios

```
git add .
```

Crear commit (instantánea del proyecto). Es importante siempre poner un comentario descriptivo.

```
git commit -m "primer commit"
```

Muestra la lista de commit del más reciente al más antiguo.

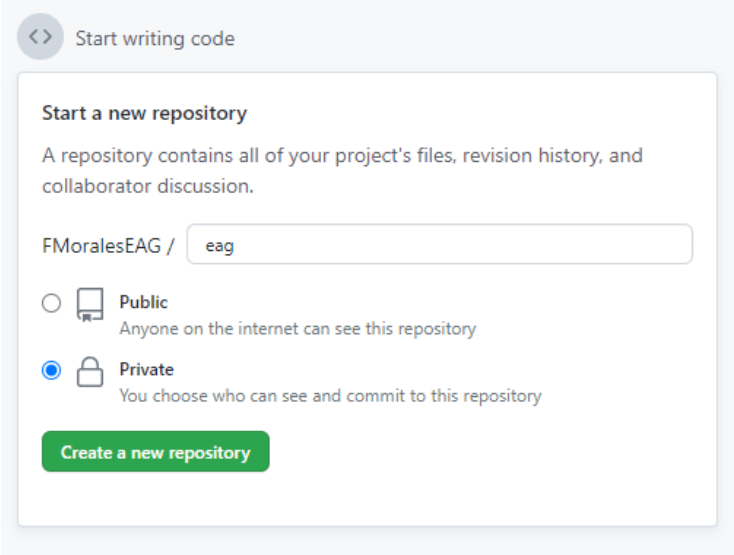
```
git log
```

GITHUB

Github es la plataforma online para trabajar con proyectos en git, en simples palabras es utilizar git de forma remota. Pero son cosas totalmente distintas.

https://github.com/

Crear un nuevo repositorio



Una vez realizados estos pasos podremos subir a remoto los cambios realizados cada vez que deseemos.

Para ello haremos uso de Git push

Ejercicios Clase

Ejercicio 1

Configurar Git definiendo el nombre del usuario, el correo electrónico. Mostrar la configuración actual.

Ejercicio 2

Crear un repositorio nuevo con el nombre Clase y mostrar su contenido.

Ejercicio 3

Comprobar el estado del repositorio.

Crear un fichero indice.txt con el siguiente contenido:

> 1: Introducción a Git
>
> 2: Flujo de trabajo básico
>
> 3: Repositorios remotos

Comprobar de nuevo el estado del repositorio.

Añadir el fichero a la zona de intercambio temporal.

Ejercicio 4

Realizar un commit de los últimos cambios con el mensaje “Indice” .

Ejercicio 5

Realiza un cambio y haz un nuevo commit.

Mostrar los cambios de la última versión del repositorio con respecto a la anterior.

Cambiar el mensaje del último commit por “Añadido contenido”

Volver a mostrar los últimos cambios del repositorio.

Ejercicio 6

Mostrar el historial de cambios del repositorio.

Crear la carpeta capitulos y crear dentro de ella el fichero capitulo1.txt con el siguiente texto.

> Git es un sistema de control de versiones ideado por Linus Torvalds.

Añadir los cambios a la zona de intercambio temporal.

Hacer un commit de los cambios con el mensaje “Añadido capítulo 1.”

Volver a mostrar el historial de cambios del repositorio.

Ejercicio 7

Crear el fichero capitulo2.txt en la carpeta capitulos con el siguiente texto.

> El flujo de trabajo básico con Git consiste en: 1- Hacer cambios en el repositorio. 2- Añadir los cambios a la zona de intercambio temporal. 3- Hacer un commit de los cambios.

Añadir los cambios a la zona de intercambio temporal.

Hacer un commit de los cambios con el mensaje “Añadido capítulo 2.”

Mostrar las diferencias entre la última versión y dos versiones anteriores.

Ejercicio 8

Crear el fichero capitulo3.txt en la carpeta capitulos con el siguiente texto.

> Git permite la creación de ramas lo que permite tener distintas versiones del mismo proyecto y trabajar de manera simultanea en ellas.

Añadir los cambios a la zona de intercambio temporal.

Hacer un commit de los cambios con el mensaje “Añadido capítulo 3.”

Mostrar las diferencias entre la primera y la última versión del repositorio.

Ejercicio 9

Mostrar quién ha hecho cambios sobre el fichero indice.txt.

Comandos útiles

Git checkout -- .

Con el podemos volver directamente al último commit realizado.

Es una forma rápida de revertir los cambios.

Git branch

Mostrará un listado con las ramas del repositorio.

Git branch –m master main

Con este comando podemos cambiar el nombre de una rama en concreto en este caso la rama master la llamaremos main.

Git config --global init.defaultBranch main

Con este comando podremos poner como usuario predefinido main en vez de master de forma que cada vez que creemos un repositorio main será la rama principal.

Git reset

Se trata de un comando que nos permite sacar todos o el archivo que deseamos del staging area (es decir después de haber realizado un git add)

Esto nos permite corregir esta acción en caso de error y no desear realizar un commit de esos archivos.

Git commit --am “Mensaje del commit”

Permite saltarnos el staging área y ahorrarnos el uso del git add. De forma que realizamos el commit directamente.

Solo funciona si el archivo ya ha sido trackeado anteriormente (No sirve para archivos nuevos)

Git add *html

Este comando nos permite mandar al staging área todos los archivos html con cambios.

.gitkeep

Git no es capaz de gestionar carpetas vacías. Por tanto este archivo nos será muy útil cuando tenemos carpetas que necesitaremos pero de momento están vacías. Si queremos que esta quede en nuestro repositorio necesitaremos crear .gitkeep

Este archivo estará vacío tan solo debe existir dentro del directorio.

Git alias

Ya sabemos que algunos comandos pueden ser largos y tediosos, pues bien git alias viene a simplificarnos esto.

Con git alias podremos guardar comandos más largos en otros personalizados.

Importante: Un alias debe ser único **No** podremos tener un alias llamado status por ejemplo.

Ejemplo:

Git config --global alias.s “status --short”

git config --global alias.lg "log --graph --abbrev-commit --decorate --format=format:'%C(bold blue)%h%C(reset) - %C(bold green)(%ar)%C(reset) %C(white)%s%C(reset) %C(dim white)-%an%C(reset)%C(bold yellow)%d%C(reset)' --all"

Git config --global –e

Con este comando accederemos a la configuración y podremos editarla si lo necesitamos.

Para editar con la tecla a

En caso de editarla al salir deberemos poner :wq

Si no hemos editado :q

Conocimientos Básicos

Pregunta 1:

Cuando utilizamos Git, ¿Todos los miembros del equipo tienen una copia del repositorio central?

- ◯ **Verdadero**
- ◯ **Falso**

Pregunta 2:

¿Un commit en Git, es como una fotografía del proyecto en ese momento?

- ◯ **Verdadero**
- ◯ **Falso**

Pregunta 3:

¿Para qué sirve el comando `git init` ?

- ◯ **Para configurar nuestro usuario y contraseña**
- ◯ **Para inicializar el repositorio**
- ◯ **Para iniciar el seguimiento de los archivos**

Pregunta 4:

Localmente, ¿Dónde se encuentra la carpeta que permite controlar todo nuestro repositorio de git?

- ◯ **En la carpeta git**
- ◯ **En la carpeta git-config**
- ◯ **En la carpeta .git**

Pregunta 5:

Si queremos eliminar git de nuestro repositorio, ¿qué debemos hacer?

- ◯ **Borrar la carpeta .git**
- ◯ **Ejecutar el comando git destroy**
- ◯ **Copiar el folder a otro directorio**

Pregunta 6:

¿Qué es el stage o el escenario?

- ◯ **Un lugar donde se muestra toda la información del commit**
- ◯ **El lugar donde podemos ver todos los archivos de nuestro repositorio**
- ◯ **Es un lugar donde podemos confirmar los archivos y carpetas que conformarán el commit**

Pregunta 7:

El comando `git s` , ¿Es propio de Git?

- ◯ **Verdadero**
- ◯ **Falso**

Git diff

Es el comando que utilizaremos para ver los cambios con respecto al commit anterior.

Es importante recordar que solo será posible antes de pasar al staging area.

Git diff --staged

Con este comando podremos realizar un git diff aunque el archivo se encuentre en staging área.

Git commit -- amend –m “mensaje”

Con el podremos modificar el mensaje del commit anterior.

Git reset

Te permite RESTABLECER tu estado actual a un estado específico. Puedes restablecer el estado de archivos específicos, así como el de toda una rama.

--soft

No restablece el fichero índice o el árbol de trabajo, pero restablece HEAD para commit.

--mixed

Restablece el índice, pero no el árbol de trabajo e informa de lo que no se ha actualizado.

--hard

Restablece el índice y el árbol de trabajo. Cualquier cambio en los archivos rastreados en el árbol de trabajo desde el commit son descartados.

Git reflog

Existe un registro de todas las referencias tomadas por el puntero HEAD en Git que se llama reflog. Es un registro que va a almacenar todos los commits por donde pase el puntero. El reflog, por ejemplo, será capaz de guardar los cambios de rama o la adición de un commit.

Ejercicio de Clase

1. Crea un directorio (superhéroes) con los archivos héroes, villanos y ciudades. Crea un commit por cada archivo.

2. Inserta 3 heroes. Realiza un commit. Inserta sus ciudades. Inserta sus villanos.

3. Crea el archivo poderes. Introduce si tienen algún poder. Realiza su commit.

4. Modifica el mensaje del último commit por “añadido poderes de los héroes”

5. Vuelve a antes de añadir el archivo poderes. (Se debe perder el archivo)

6. Nos hemos arrepentido vuelve al momento donde tenías el archivo poderes.

7. Vuelve al momento inicial del repositorio. (Solo debe quedar el archivo Heroes)

Ramas y modificación de archivos

Git mv “archivo.original” “archivo.modificado”

Aunque su funcionalidad real no es la de modificar el nombre de un archivo podemos utilizarlo para esto de la forma que hemos visto.

Git rm “archivo”

Con este comando eliminaremos el archivo que deseemos desde la propia consola. (Realmente es algo que también podríamos hacer de forma manual)

.gitignore

En el gitignore se especificarán todas las rutas y archivos que no se requieren y con ello, el proceso de control de versiones simplemente ignorará esos archivos.

Git Branch

El comando git branch es el que usaremos principalmente para trabajar con la creación de ramas, borrado de ramas y demás.

Git branch “nueva-rama”

Crearemos una nueva rama sobre la que trabajar y continuar una nueva línea de trabajo sin afectar a la rama principal.

Git checkout “rama”

Con este comando podremos movernos a la rama que deseemos.

Git checkout –b “rama”

Es la forma abreviada de crear y movernos a una rama. Es la forma más común.

Git merge

Será el comando utilizado para combinar ramas normalmente la master/main con otra rama que estuvimos trabajando.

Merge Fast-forward

Fast Forward en Git Merge sirve para realizar fusiones de avance rápido en las situaciones donde exista un proceso lineal desde el extremo de la rama actual y que se extienda hasta la rama de destino.

En ese caso, en vez de fusionar como tal las ramas, el sistema de Git se encargaría de mover el extremo de la rama actual al extremo a la rama

destino, sin necesidad de crear una confirmación o commit de fusión adicional.

Eliminar una rama

Git branch -d “rama”

Con este comando podremos borrar la rama que deseemos siempre que hiciéramos un merge previo.

Importante debemos situarnos en la rama master/main

Git branch -d “rama” –f

En este caso podremos eliminar la rama sin necesidad de merge. Este –f lo que provoca es forzar que se elimine.

Tags – Etiquetas

Los tags o etiquetas no son más que referencias a un commit específico. Suelen utilizarse para marcar las versiones.

Git tag v1.0.0.0

Para añadir una etiqueta al último commit tan solo deberemos usar git tag “etiqueta”.

Git tag

Nos muestra todas las etiquetas creadas.

Git tag –d

Para eliminar una etiqueta.

Git show “tag”

Podremos ver información sobre una etiqueta y su commit asociado.