Control de versiones con GIT

Contenido

¿Qué es el control de versiones?

Conceptos básicos

La necesidad

La idea de un control de versiones surge de un concepto muy básico. Hazte la siguiente pregunta, ¿Cuántas veces has creado archivos estilo *"entrega.doc", "entrega_corregida.doc", "entrega_final_correccion.doc"* …? Todo para no romper o perder un cambio que hayas hecho, asegurarte que es el fichero correcto. Que podrías haber modificado el mismo archivo, pero piensas ¿y si este cambio me va a hacer falta?, mejor no lo borro.

¿Y si a todo eso le añades, además, que es un trabajo en grupo? Terminan apareciendo demasiados archivos "chatarra" que luego hay que acordarse de borrar.

De estas necesidades nacen los **sistemas de control de versiones**, herramientas como **GIT**, Mercurial, CVS o Subversion (entre otras muchas), que vienen a solucionar esta necesidad.

La herramienta.

En este curso vamos a usar únicamente GIT, ya que es el más popular y comúnmente utilizado, pero todas las terminologías usadas serán similares, ya que la base e idea son la misma.

¿Qué nos permite esta herramienta? Con GIT dispondremos de nuestros archivos en un **repositorio** y nos permitirá controlar y ver todos los cambios realizados, tanto por nosotros como por algún colaborador. Estos repositorios pueden ser **locales** o **remotos**, quiere decir

esto que dispondremos del guardado de los archivos y el control de las versiones en nuestro sistema local o en un equipo externo. Lo normal es que se trabaje con sistemas remotos siempre, por razones más que obvias, si se rompe tu sistema pierdes los archivos, si se rompe el servidor del repositorio, mantienes una copia de los archivos en local siempre (esto se explicará más adelante).

Otra de las ventajas de usar esta herramienta son las llamadas **Branch,** o ramas en español, con las que puedes trabajar en múltiples cambios distintos sin que afecten el código **main** o principal. De esta forma puedes hacer cambios experimentales y mantenerlos en el repositorio sin que afecte de ninguna forma a la rama principal. Esto es muy útil en casos de que seáis varios trabajando en lo mismo y tus cambios puedan afectar directamente a otro módulo del código en el que un compañero esté trabajando. Así entonces tu harás tus cambios y los mantendrás en el repositorio, harás tus pruebas y, cuando estés seguro de su correcto funcionamiento, realizar un **merge** o integrar a la rama principal.

Terminología

Antes de continuar, será mejor que se aclaren las terminologías. Es importante aclarar que siempre vamos a usar el nombre en inglés ya que es el original y lo que se usa comúnmente, a excepción de repositorio.

- **Repositorio:** Es donde se almacenan los datos actualizados e histórico de cambios.
- **Remote o Remoto:** Repositorio al que se suben las modificaciones.
- **Local:** Copia del repositorio que se mantiene en la máquina local hasta sincronizar con el remoto.
- **Main o Main Branch o rama principal:** Es la base de los archivos donde se almacenan los cambios principales y revisados.
- **Branch o rama:** Bifurcación de los datos que parten de un punto común a otra rama o del propio main**.** Es la forma más segura de trabajar, para que la rama principal solo contenga los cambios revisados. Si se trabaja solo, es raro realmente usar ramas. Con las ramas aíslas modificaciones que se mantienen sincronizadas.
- **Commit o publicar:** Guardar un conjunto de cambios junto con una descripción aproximada de lo que se ha realizado. Los commits se mantienen en local hasta que se sincroniza.
- **Stash o preparar:** Cuando un cambio está en stash significa que está listo para ser commiteado.
- **Add o añadir:** Añadir un cambio a lo preparado o stasheado (palabra inventada, pero la usamos los españoles, realmente se debería decir *stashed*)
- **Push o subir:** Sincroniza tus cambios en local subiéndolos al repositorio remoto.
- **Pull o bajar:** Descarga los cambios que se encuentren en el remoto distintos de los almacenados en local.
- **Merge o fusionar:** Integrar los cambios de una rama en otra o la principal.
- **Conflict o conflicto:** Cuando los cambios en local chocan con los del repositorio o cuando un merge de dos ramas presenta choques. Resolver un conflicto se le dice **merge conflic**.
- **Status o estado:** Información del estado de tu repositorio local respecto al remoto.

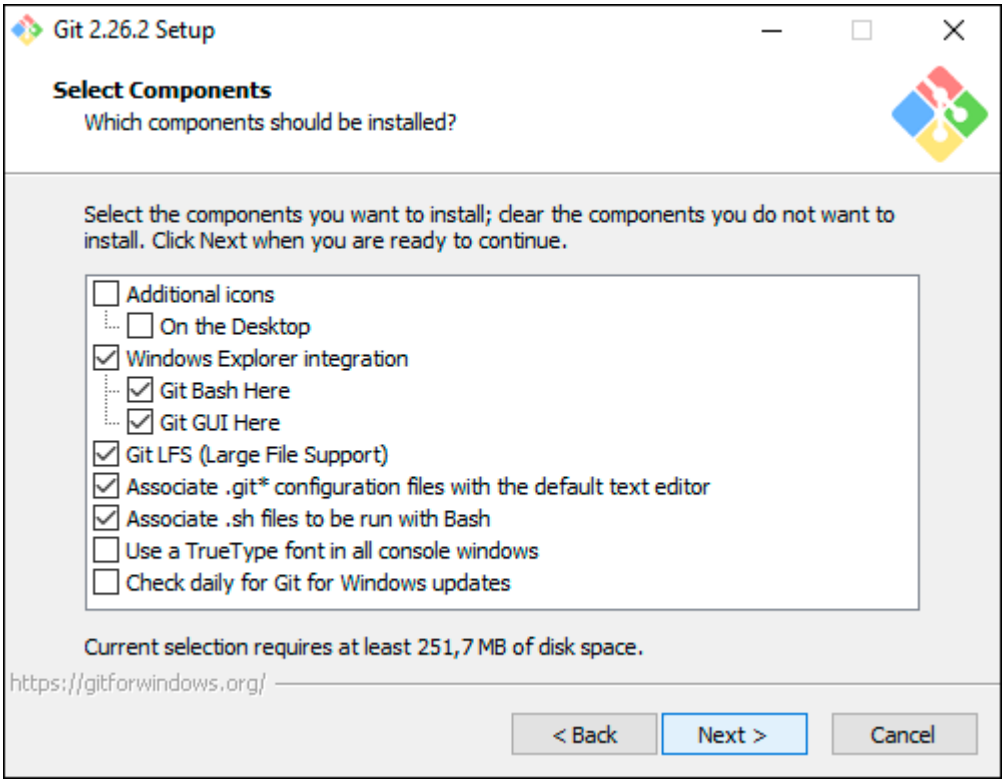
Aparecerá más terminología en adelante, pero se irá explicando según se avance.

Instalando y configurando GIT

Está "guía" de instalación es para Windows, en Linux viene GIT instalado por defecto. La configuración es igual en ambos casos.

Descargar e instalar

1. Debemos instalar una aplicación que descargaremos aquí .
2. Abrimos el instalador y le damos a **next** aceptando el acuerdo de licencia.
3. Escogemos el destino de la instalación, dejaremos la por defecto y le daremos a **next**.
4. Seleccionamos los componentes en la siguiente ventana. Yo recomiendo los siguientes:



5. A continuación, debemos escoger que editor de texto usará GIT por defecto. Mi recomendación es escoger o VS Code (Visual Studio Code) o Notepad++.
6. En la siguiente ventana, pedirá que ajustemos el nombre de la rama inicial en nuevos repositorios, dejamos la opción "*Let Git decide".*
7. Debemos ahora ajustar nuestro PATH, para que todo funcione de forma correcta a lo largo del curo (y por mayor comodidad) escogeremos la opción recomendada *"Git from the command line and also from 3rd-party software"*. De esta manera podremos utilizar las extensiones de VS code para GIT.
8. Esta siguiente pestaña es importante: Debemos seleccionar Checkout Windows-style. Esto es realmente importante, luego se explicará el porqué.
9. Las dos siguientes escogemos la por defecto.
10. En la que nos pide *"Choose a credential helper"* Escogeremos la primera opción *"Git Credential Manager Core".*
11. En la siguiente escogemos las dos opciones que nos dan, ambos son *"enable"*.
12. En la última de configuración, nos preguntará si queremos las opciones experimentales, no escogeremos ninguna. Le damos a *"Install"*.

13. Tras la instalación, le damos a **next** para que se cierre el instalador.

Configurando GIT

Lo primero, obvio y más importante, será crearnos nuestro usuario en GitHub, si es que no lo tenemos.

Vamos a la página de GitHub, le damos a Sign Up y seguimos el creador de usuario interactivo, son majísimos por crear algo así.

Escoge un buen nombre de usuario, es el que vas a usar por el resto de tu vida y querrás que sea fácil y cómodo de usar, tanto para ti como tus compañeros de equipo.

Ahora nos vamos a la terminal de Windows y escribimos lo siguiente:

- GIT CONFIG --GLOBAL USER.NAME "JOHN DOE"
- GIT CONFIG --GLOBAL USER.EMAIL johndoe@example.com

Modifica las credenciales por tu nombre de usuario y tu correo usado.

Con esto ya estaría configurado, no hace falta hacer más de momento, más adelante configuraremos la extensión de GIT en VS Code.

Usando GIT

Todo esto va a ser explicado usando un repositorio como ejemplo. Si no te he añadido como colaborador pídemelo para que puedas participar sin ser un "externo".

Este será nuestro repositorio, entra y comprueba que eres colaborador.

Ahora, todo lo que deberás aprender está dentro del repositorio.

Ejercicio

Vamos a plantear un ejercicio muy simple, todo ello guiado por mí.

1. Deberás clonar el repositorio.
2. Entra en la carpeta **saludos**.
3. Crear dentro de la carpeta un fichero en formato **md** similar al que ya hay con el nombre **tu_nombre.md** y saluda al grupo e indica que te gusta. Puedes usar los demás ficheros que ya hay para aprender como se usa el formato md o buscarlo en internet.
4. Añade tu archivo en un commit y pushealo a la rama **main**.
5. Crea tu propia rama y haz un checkout a ella.
6. Pushea tu rama al repositorio remoto.
7. En tu rama, coge el archivo **chuleta.md** y añade algo que consideres que falte.
8. Commitealo y súbelo a tu rama.

Ahora un ejercicio para casa:

1. Dentro de tu rama, añade al archivo **chuleta.md** la sección 3 que indique como se usa el formato **md**.
2. Crea una tabla, siguiendo el formato que ya hay, que explique como se usan cada uno de los caracteres especiales que hay dentro del formato **md**.
3. Crea un **pull request** de tu rama a la rama **main**, yo valoraré cual es el que mejor está y lo aprobaré.
4. Si tu rama es la aprobada, genial, aquí te quedas.
5. Si no es la escogida, deberás hacer un **pull desde otra rama a la tuya**.