Git y Github

Errant Programmer

17/12/23

Índice

1. Instalación

Antes de empezar a explicar a como utilizar git, vamos primero a tener que instalarlo.

1.1. Instalación de git para Windows

Para poder instalar git, nos vamos a dirigir a la página web de git

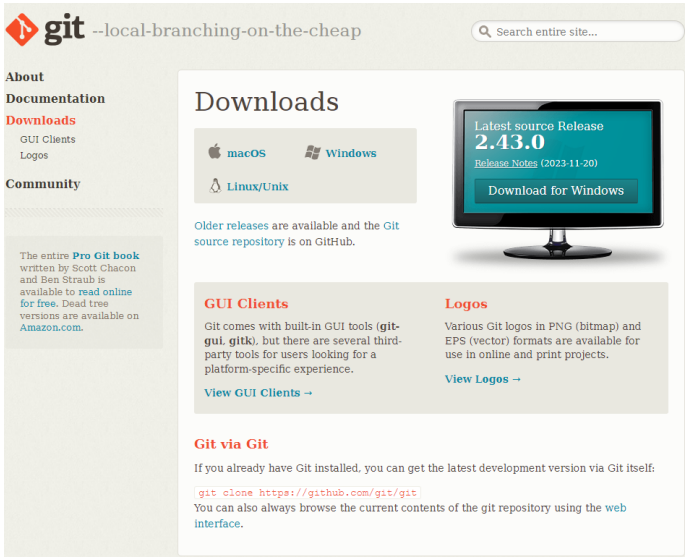

Figura 1: Pagina de descarga de git

Para nuestro caso vamos a descargar git según la versión de 32bits o 64 bits, según tenga nuestro computador.

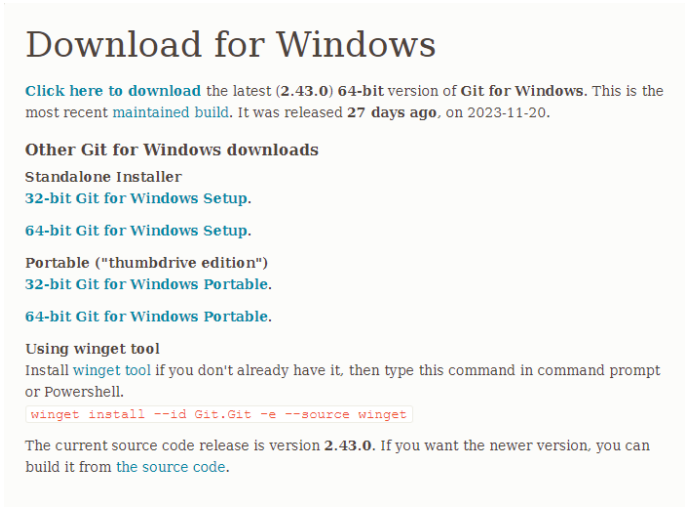

Figura 2: Git: Instalación Windows

Una vez termine al descargar, pasamos a ejecutarlo.

1. Aceptamos la licencia GNU-GPL que tiene git.

Figura 3: Git: GNU General Public License

2. Escogemos la carpeta donde se desea que se instale Git, recomiendo dejarlo en su valor por defecto.

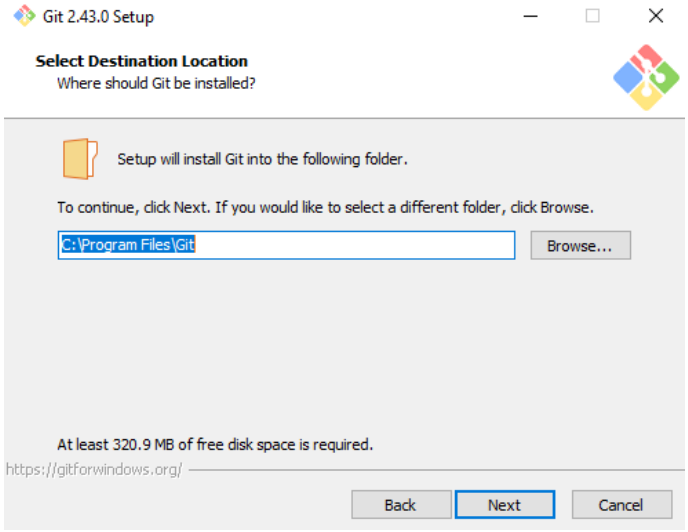

Figura 4: Git: Carpeta de instalación

3. Podemos cambiar la carpeta para los accesos directos del programa, pero de igual forma recomiendo dejarlo por defecto.

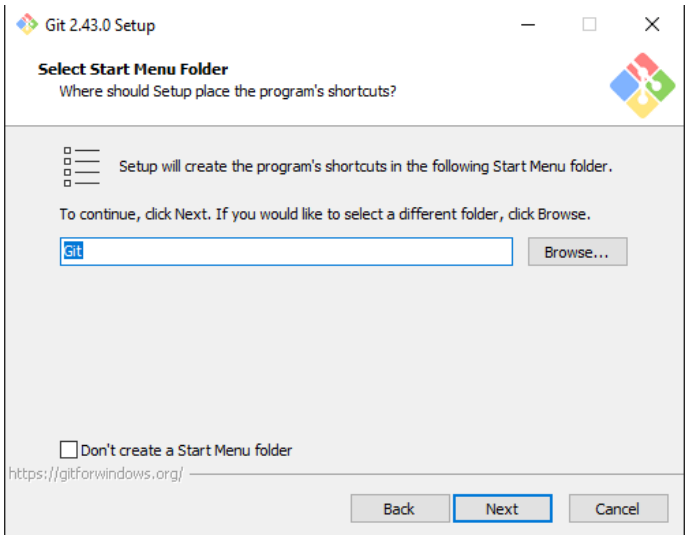

Figura 5: Git: Carpeta de acceso directo

4. Ahora en git, cuando vayamos a realizar un *commit* o alguna configuración desde la terminal, vamos a necesitar de un editor de texto, por defecto este viene con *Vim*. Recomiendo escoger el que le sea más comodo, en caso no este familiarizado con las keybindings de *Vim*.

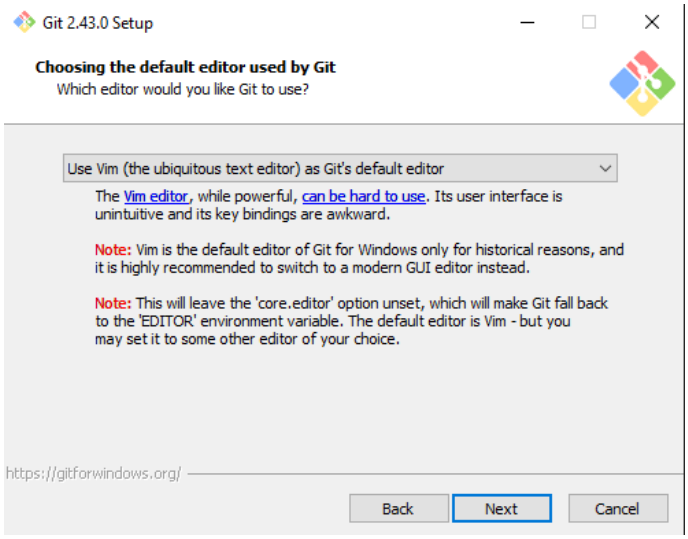

Figura 6: Git: Editor por defecto

5. Por defecto se va a estar marcada la opción de **Let Git decide**. Esto porque historicamente cuando creemos un nuevo repositorio con git(esto se vera más adelante) la rama principal se suele crear con el nombre de *Master*, pero como nosotros vamos a estar trabajando adicionalmente con *Github*, este recomienda que el nombre de la rama principal se denomine como *main*. De igual forma, más adelante veremos como podemos cambiarle el nombre a la rama principal.

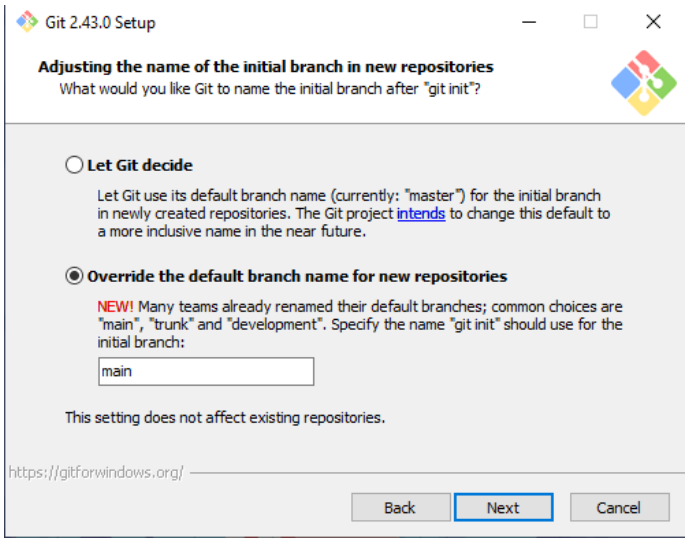

Figura 7: Git: Rama principal

6. Recomiendo dejar los ajuste del PATH por defecto, pero de igual manera los puede modificar según sus necesidades

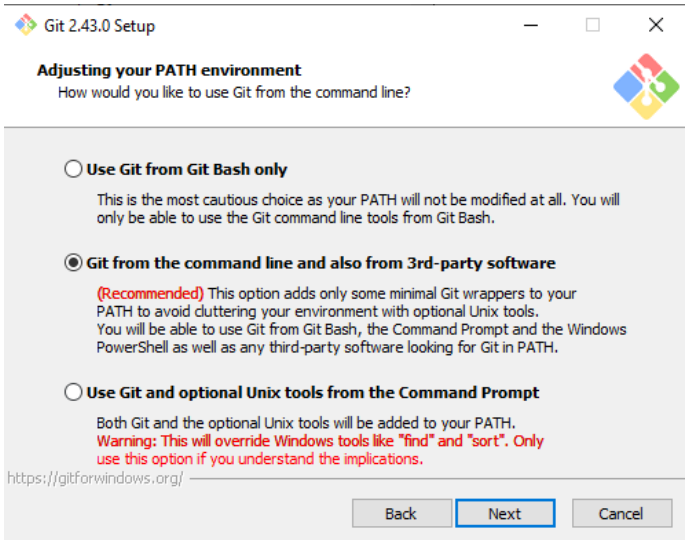

Figura 8: Git: Ajuste del PATH

7. Todas las demás opciones que siguen, recomiendo dejarlo por defecto.

8. Con esto ya terminamos de instalar git, y si buscamos dentro de nuestro programas vamos a poder verificar su instalación.

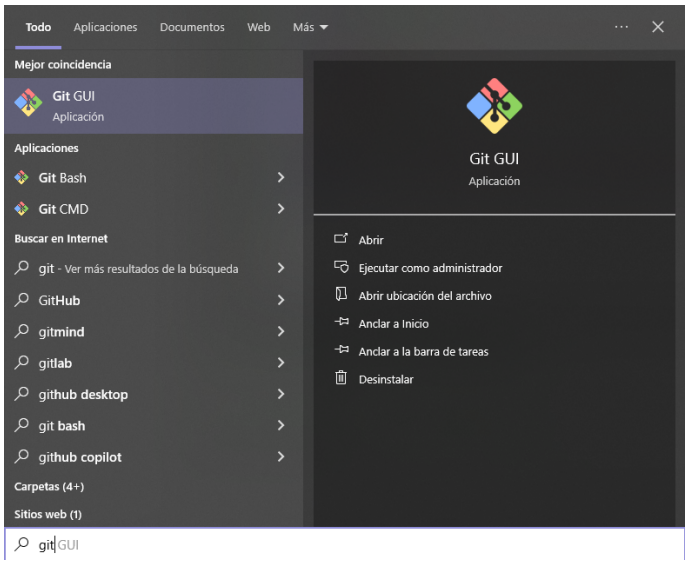

Figura 9: Git: Verificación de instalación

1.2. Instalación de git para Linux

La instalación de git para Linux, va a depender de la distro con la que usted cuente. Aquí ejemplos para multiples distro de Linux:

- Debian/Ubuntu

```
sudo apt-get install git
```

- Fedora 22 y posteriores

```
sudo dnf install git
```

- Gentoo

```
sudo emerge -ask -verbose dev-vcs/git
```

- Arch Linux

```
sudo pacman -S git
```

- Nix/NixOS

```
sudo urpmi git
```

2. ¿Qué es Git?

EN PROCESO DE REALIZACIÓN

3. Configuraciones Básicas de Git

Para realizar las configuraciones de git, vamos a poder realizar a 3 niveles de configuración, cada nivel de 2 formas distintas.

- A nivel de sistema.
  1. Modificando el archivo de configuración general que se encuentra en la siguiente ruta:

```
/etc/gitconfig
```

  2. Agregando la bandera *–system* después del git config

- A nivel de nuestro usuario.
  1. Modificando el archivo de configuración de usuario que se encuentra en una de las siguientes rutas:

```
~/.gitconfig

~/.config/git/config
```

  2. Agregando la bandera *–global* después del git config

- A nivel de proyecto.

1. Modificando el archivo de configuración del proyecto se encuentra en dentro de la carpeta *.git* de nuestro proyecto.

```
ruta_del_proyecto/.git/config
```

2. Sin agregar ninguna bandera a git config.

Importante

1. El trabajar con git suele ser normalmente desde la terminal, por lo que no se modificará directamente ninguno de los archivos mencionados anteriormente.
   - Si usted trabaja desde windows, se recomienda usar la terminal que se descargó al momento de instalar git(git bash).
2. Si va a trabajar con varias cuentas de github, ya sea que desee una cuenta personal y una cuenta para trabajo, esto se explicará en el CAPITULO CON MULTIPLES CUENTAS DE GITHUB.
   - Por tal motivo, todas las configuraciones mostradas solo será a nivel de repositorio.

3.1. Configuración de usuario

Lo primero que vamos a establecer es nuestro nombre de usuario y correo electrónico. Esto es indispensable porque Git lo va a utilizar para poder identificar a la persona que realizo algún cambio, es específico con los *commits*.

```
git config user.name "nombre de usuario"
git config user.email "correo_eletronico@ejemplo.com"
```

3.2. Configuración de editor

La segunda cosa importante que tenemos que configurar es el editor con el que va a trabajar git, cuando este nos solicite ingresar algún tipo de texto.
Si instalamos git desde windows, pudimos escoger con que editor va a trabajar git; o en caso necesitemos trabajar con algún otro tipo de editor.

```
git config core.editor "editor-de-codigo"
```

Aquí se presentan unos pocos, pero si gusta trabajar con otro editor puede ver la siguiente lista.

- Atom:

```
git config core.editor "atom --wait"
```

- Emacs:

```
git config core.editor emacs
```

- Nano:

```
git config core.editor "nano -w"
```

- Visual Studio Code:

```
git config core.editor "code --wait"
```

3.3. Configuración de salto de línea

Algo a tener en cuenta que Windows y Linux, no manejan el salto de fin de línea de cual forma, por lo que si vamos a trabajar en un proyecto donde tengamos que trabajar en múltiples sistemas operativos, esto es muy recomendable.

```
git config core.autocrlf opciones
```

Para esto vamos a tener 3 opciones:

true: Convierte los finales de salto de línea al formato CRLF. Este es el recomendable si usted está trabajando desde Windows.

false: No se realiza ninguna conversión automática de saltos de línea; por lo tanto, se van a quedar con el salto de línea definido por el sistema operativo donde fueron creados. Recomendado si usted trabaja desde Linux.

input: Git convertirá los saltos de línea al formato del sistema operativo con el que estamos trabajando. Sin embargo, al momento de realizar un commit estos se almacenarán en formato de tipo UNIX(LF). Recomendado si tiene que trabajar en múltiples sistemas operativos.

```
git config core.autocrlf input
```

4. Ciclo de Vida de los archivos

Antes de ver los comandos más básicos de git, debemos de conocer el ciclo de vida de los archivos que estos pueden tener cuando se trabajan con *git*. Estas de dividen en 4 etapas:

1. **Untracked:** Esto significa que Git no está siguiendo el archivo. Es decir, Git no sabe acerca de los cambios en este archivo y no lo incluirá en tu próximo commit a menos que lo agregues explícitamente con git add. Los archivos suelen estar en este estado cuando son nuevos en el repositorio.

2. **Unmodified:** Un archivo se encuentra en este estado cuando está en el repositorio y no ha sido modificado desde la última confirmación (commit). Git está al tanto de este archivo y no se realizaron cambios en él desde la última confirmación.

3. **Modified:** Después de modificar un archivo que Git está rastreando, el estado de ese archivo cambia a "Modificado". Esto significa que han habido cambios en el archivo desde la última confirmación y Git los ha detectado, pero aún no se han preparado para una nueva confirmación.

4. **Staged:** Cuando decides que quieres incluir los cambios de un archivo modificado en tu próxima confirmación, lo añades al área de preparación (staging area) con git add. Una vez que un archivo está en el área de preparación, se considera "preparadoz está listo para ser confirmado en el repositorio.

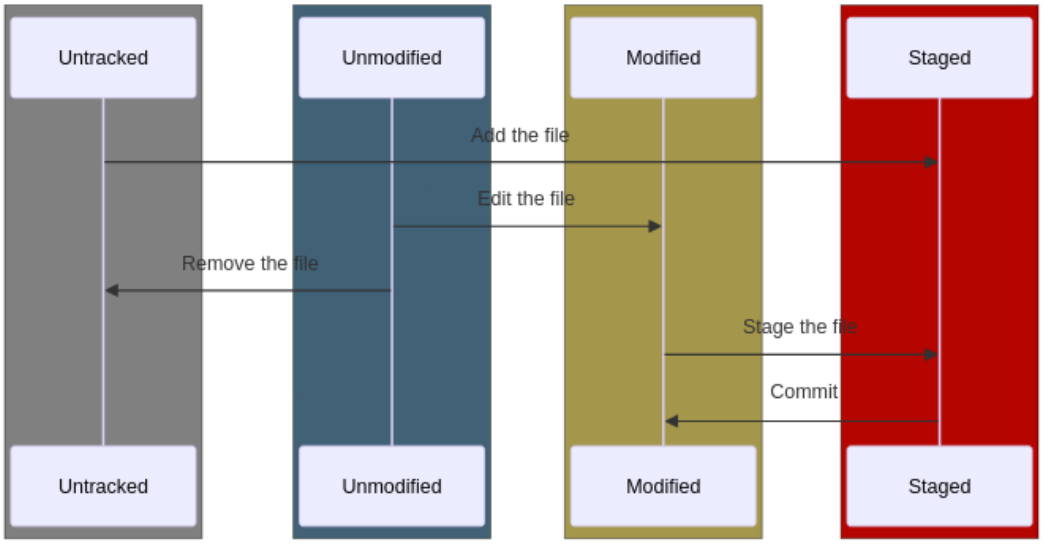

Figura 10: Ciclo de vida los archivo en Git

5. Fundamentos De Git

Durante el siguiente capítulo, vamos a conocer los comandos básicos de Git, con los cuales vamos a poder:

- Inicializar un repositorios
- Verificar el seguimiento de archivos
- Commit
- Ignorar archivos no deseados
- Errores comunes
- Navegar por el historial de nuestro proyecto

Importante

Para conocer más información sobre los diversos comandos que existen en git, y puede revisarla con algunos de los siguiente comandos:

```
git help <comando> # Recomendado si trabaja desde Windows

git <comando> --help

man git <comando>  # Recomendado si trabaja desde Linux
```

Ejemplo de uso:

```
git help config

git config --help

man git config
```

5.1. Inicializar un respositorio de Git

Para iniciar un nuevo proyecto, nos vamos a dirigir al directorio de nuestro y vamos a abrir nuestra terminal.

```
git init
```

Con este comando se nos va a crear un carpeta ***.git*** dentro de nuestro proyecto, el cuál va a contener los archivo necesarios para el repositorio.

5.2. Estado de los archivos

La herramienta para poder conocer el estados de cualquier archivo, es con el siguiente comando:

```
git status # Para ver la informacion de todos los archivos
git status nombre-del-archivo # Para ver informacion de un único archivo
```

Ejemplo con la creación de un archivo, por ejemplo si vamos a agregar un *README.md* al proyecto.

```
#=====================================================
# Codigo
#=====================================================
echo 'Nombre-del-proyecto' >> README.md
git status
#=====================================================
#Output
#=====================================================
On branch main

No commits yet

Untracked files:
(use "git add <file>..." to include in what will be committed)
README.md

nothing added to commit but untracked files present (use "git add" to track)
```

Como nos podemos percatar, no solo nos da información sobre los archivos, sino también sobre la rama en la que estamos trabajando; el manejo de las ramificaciones en *git* se realizará más adelante. Así mismo, si solo queremos conocer la información de los archivos, de una manera más corta, se puede hacer uso del siguiente comando:

```
#================================================
# Codigo
#================================================
git status -s # -s es abreviatura de --short
#================================================
# Output
#================================================
?? README.md
#================================================
# Forma general
#================================================
XY nombre-del-archivo
```

Los valores de XY, va a depender de los cambios que haigamos realizados, estos pueden tener 8 posibles valores:

1. **‘ ’:** sin modificar
2. **M :** modificado
3. **T :** Tipo de archivo modificado
4. **A :** añadido
5. **D :** borrado
6. **R :** renombrado
7. **C :** copiado
8. **U :** actualizado pero no fusionado

Para ver una lista de cada uno de las posibles combinaciones y su significado, puede revisar la documentación o revisar la documentación de git status como se explico anteriormente.

5.3. Rastreo de los archivos

Los archivos rastreados son aquellos que ya pertenecen al proyecto, estos pueden ser modificados, sin modificar o preparados. Los archivos sin rastrear son todos los demás.
Para empezar a rastrear un archivo, vamos a ejecutar el siguiente comando:

```
git add . # Todos los archivos, incluso los que ya estaban rastreados
git add NombreDelArchivo # Solo el archivo indicado
```

Importante

Si vamos a agregar multiples archivos, directorios o demás, ir escribiendo 1 por 1 puede ser muy tardado, por lo que se recomienda usar expresiones regulares.

5.4. Ignorar archivos

Esta opción puede parecer algo raro, ya que normalmente pensamos que todos los archivos que tengamos en nuestro proyecto son esenciales, pero no siempre es así. Ejemplo:

- La carpeta *.vscode*, la cual contiene información de como nosotros personalizamos nuestro entorno de trabajo en **vscode**, la cual no es relevante para el proyecto.
- Archivos generados al momento de compilar un proyecto, esos archivos siempre se generan cuando se compila; por lo tanto, no es necesario incluirlo.