Git

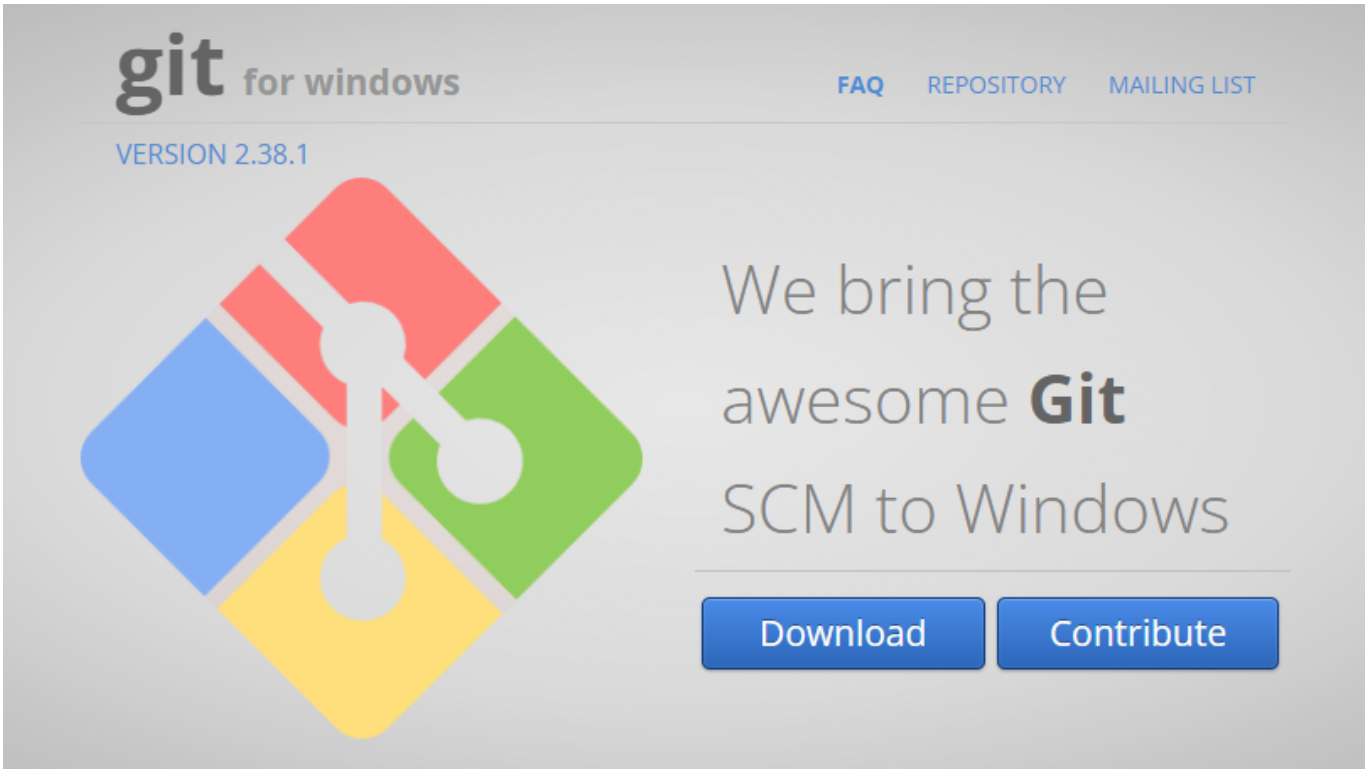
Git es un software de **control de versiones** que permite a los desarrolladores llevar un seguimiento de las diferentes **versiones** de un archivo o conjunto de archivos mientras se crea un nuevo software.

Esto es útil porque permite a los desarrolladores volver a una **versión anterior** de un archivo si hicieron un cambio que resultó en un error. También permite a los desarrolladores colaborar en el mismo proyecto al mismo tiempo.

Para usar git, primero debe instalarlo en su computadora. Luego, debe configurarlo para que sepa quién está haciendo cambios en qué archivos. Esto se hace mediante el uso de una cuenta de git. Una vez que haya configurado git, puede empezar a usarlo para controlar las versiones de sus archivos.

Paso 1: Descargar e instalar Git

Antes de nada, necesitas tener Git instalado en tu sistema Windows.



Paso 2: Configurar Git

Una vez que tengas Git instalado, necesitas configurarlo con tu nombre y dirección de correo electrónico. Esto se hace para que Git pueda asociar tus commits con tu identidad:

```
git config --global user.name " tu nombre "
git config --global user.email " tu@email.com "
```

Paso 3: Crear un repositorio

Para empezar a trabajar con Git, lo primero que necesitas hacer es crear un **repositorio**. Un repositorio de Git es un espacio en el que se almacenan los archivos de un proyecto y todo su historial de cambios. Puedes crear un repositorio de dos maneras: mediante la interfaz gráfica de Git o mediante la línea de comandos.

1. Abrir una terminal y navegar hasta la carpeta.
2. A continuación, debe escribir el comando `git init`. Este comando creará un repositorio git en su carpeta.

Añadir cambios

Ahora, cada vez que haga un cambio en uno de sus archivos, debe agregar el archivo al repositorio git. Esto se hace con el comando `git add`.

Confirmar cambios (commit)

Luego, debe confirmar los cambios con el comando `git commit`. Este comando le permitirá escribir un mensaje que describa los cambios que hizo en el archivo

Historial de cambios

Para ver un historial de los cambios que ha hecho en un archivo, puede usar el comando "git log". Este comando le mostrará todos los commits que ha hecho en el archivo. También puede ver qué líneas de código han cambiado en cada commit.

Aquí podéis ver 3 commits, el último de ellos del 16 de enero.

```
z:\apuntes>git log
commit 4094d00115cb9d888352a3a46a3ad528280b9a31 (HEAD -> master, origin/master,
origin/HEAD)
Author: Daniel Moreno <danimr@gmail.com>
Date:   Mon Jan 16 08:10:38 2023 +0100

    fdfd

commit 02033a82395a0957e47fd659a0fe429360f62fbd
Merge: 2d9dfa7b 3c064471
Author: Daniel Moreno <danimr@gmail.com>
Date:   Sun Jan 15 20:17:41 2023 +0100

    Merge branch 'master' of https://github.com/danimrprofe/apuntes into master

commit 2d9dfa7b939e6b47e9dd825724788586b9301a99
Author: Daniel  <danimr@gmail.com>
Date:   Sun Jan 15 20:17:26 2023 +0100

    fdfd
```

Paso 4: Clonar un repositorio

Si ya existe un repositorio de Git que deseas utilizar, puedes **clonarlo** en tu sistema local. Clonar un repositorio crea una copia local del repositorio remoto, lo que te permite tener acceso a todos los archivos del repositorio, así como su historial de cambios.

1. Abra la terminal.
2. Vaya a la carpeta en la que desea almacenar el repositorio clonado.
3. Escriba el siguiente comando y presione Enter:

```
git clone https://github.com/nombre-de-usuario/nombre-repositorio.git
```

4. El repositorio se ha clonado en su computadora.

Paso 5: Crear y confirmar commits

Cada vez que haces cambios en los archivos de tu proyecto, necesitas confirmarlos (hacer un "commit") para que queden registrados en el historial de cambios de Git. Esto se hace mediante el comando "git commit".

```
Z:\apuntes> git commit -m "cambios realizados"
[master 462d4c51] cambios realizados
 6 files changed, 47 insertions(+), 5 deletions(-)
 create mode 100644 docs/Imagen_digital/2023-01-13-10-27-18.png
 create mode 100644 docs/Imagen_digital/2023-01-13-10-27-41.png
 create mode 100644 docs/Imagen_digital/2023-01-13-10-28-14.png
 create mode 100644 docs/Seguridad/doxxeo/osint.md
```

En este commit se han incorporado 4 archivos nuevos al proyecto. En total, han sido modificados 6 archivos.

Paso 6: Pushear y pullar cambios

Todo este control de versiones que hemos hecho con el comando `git` se hacen en local, es decir, en el ordenador en el que estamos trabajando. Si queremos alojar nuestro proyecto, por ejemplo, en `github`, necesitaremos enviar los cambios realizados para que se guarden en el repositorio remoto. De esta forma otros usuarios puedan acceder a ellos.

A esta acción se le llama `pushear`. Pushear es el equivalente de hacer un commit en el repositorio remoto.

Ejemplo:

Vamos a enviar (sincronizar) los cambios realizados en el repositorio **local** al repositorio **remoto**.

```
E:\Docencia\apuntes>git push origin master
Enumerating objects: 36, done.
Counting objects: 100% (36/36), done.
Delta compression using up to 8 threads
Compressing objects: 100% (30/30), done.
Writing objects: 100% (30/30), 1.97 MiB | 3.53 MiB/s, done.
Total 30 (delta 4), reused 0 (delta 0), pack-reused 0
```

```
remote: Resolving deltas: 100% (4/4), completed with 4 local objects.
To https://github.com/danimrprofe/apuntes
   9118fa98..a8c5f455  master -> master
```

Ahora sí, nuestros cambios se han guardado correctamente en la nube de github, y podremos descargarlos desde cualquier dispositivo conectdo a Internet.

El comando se compone de las siguientes partes:

- **git push**: este es el comando para enviar cambios al repositorio remoto.
- **origin**: Este es el nombre del repositorio remoto. En teoría lo hemos indicado previamente. Si el repositorio fue clonado, ya aparece. Si fue creado desde nuestro pc, tendremos que agregarlo nosotros.
- **master**: este es el nombre de la rama que se enviará al repositorio remoto.

En este ejemplo, los cambios realizados en el repositorio local (E:\Docencia\apuntes) se envían al repositorio remoto (https://github.com/danimrprofe/apuntes) en la rama maestra.

El comando muestra el estado de la inserción, incluidos cuántos objetos se enumeraron, comprimieron y escribieron, y el número total de objetos. El comando también informa la cantidad de objetos locales que se usaron para completar la compresión delta. Finalmente, el comando informa qué rama se empujó y los detalles del empuje.

También puedes pullear cambios del repositorio remoto, lo que es equivalente a hacer un commit en tu repositorio local.

Paso 7: Ramas

Una rama de Git es un conjunto de commits que se encuentran en una línea separada de desarrollo. Las ramas se utilizan para desarrollar funcionalidades independientes, para hacer correcciones de errores o para experimentar con nuevas ideas.

En general, la rama por defecto se llamará `main` o `master`.

Para mostrar las ramas que tiene un repositorio:

```
z:\apuntes>git branch
* master
```

Crear una rama nueva

De manera predeterminada existe una rama master, que es la principal, si queremos modificar algo del código y queremos crear una rama distinta, llamada por ejemplo prueba, lo podremos hacer así:

```
git switch -c prueba
```

Con el argumento -c (crear) git creará la nueva rama y cambiará a ella

Cambiar a un otro commit

También puedes cambiar a un **commit** específico en vez de HEAD con git switch y la opción -d (detached)

```
git switch -d 67e01b9
```

Paso 8: Fusionar ramas

Una vez que hayas terminado de desarrollar una característica o de hacer una corrección de error en una rama, necesitarás fusionar esos cambios de vuelta a la rama principal. Esto se hace mediante el comando "git merge".

Paso 9: Resolver conflictos

A veces, cuando tratas de fusionar ramas, Git no puede hacerlo automáticamente. Esto se debe a que las ramas han divergido demasiado y Git no puede determinar qué cambios deben conservarse. En estos casos, tendrás que resolver los conflictos manualmente.

Paso 10: Etiquetas

Las etiquetas son marcadores que se pueden aplicar a commits específicos. Las etiquetas se utilizan para marcar versiones específicas de un proyecto. Por ejemplo, si estás desarrollando un software, podrías aplicar una etiqueta a cada versión que se libera.

Paso 11: Deshacer cambios

En Git, hay varias formas de deshacer cambios. La forma más sencilla de deshacer cambios es deshacer los últimos cambios confirmados mediante el comando "git reset". También puedes deshacer cambios que no hayan sido confirmados mediante el comando "git checkout".

Paso 12: Revertir commits

A veces, puede que quieras revertir un commit anterior, de forma que los cambios que se hicieron en ese commit se eliminen. Esto se hace mediante el comando "git revert".