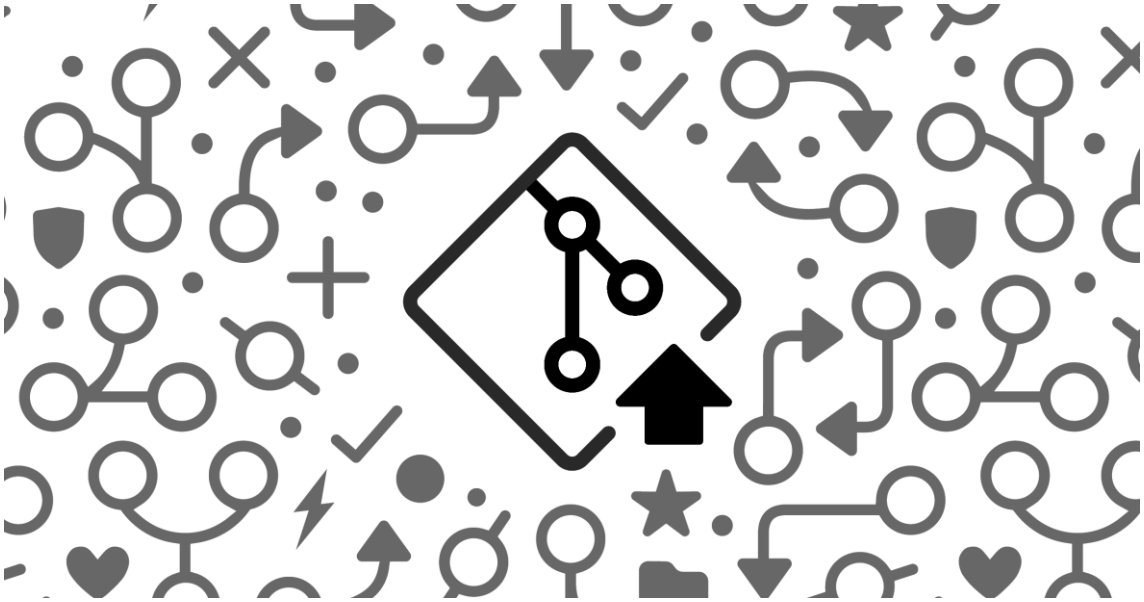
CONTROL DE VERSIONES

¿De qué sirven las emociones si no se pueden compartir?

(Anna Gavalda)

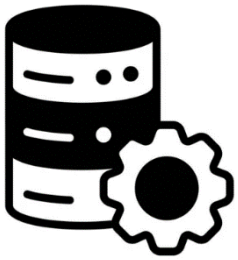
¿QUE ES UN CONTROL DE VERSIONES?

Un sistema de control de versiones (VCS por sus siglas en inglés) es un sistema que registra los cambios realizados sobre un archivo o conjunto de archivos a lo largo del tiempo, de modo que pueda recuperar en cualquier momento versiones específicas. Si, por ejemplo, usted es un diseñador gráfico o un desarrollador web, y quiere mantener cada versión de una imagen o de un archivo HTML de un sitio web, una de las decisiones más importantes para su proyecto podría ser la de tener un sistema de control de versiones ya que permitiría revertir proyectos o parte de ellos a un estado anterior, comparar cambios a lo largo del tiempo, y ver quién modificó por última vez algo que puede estar causando un problema, quién introdujo un error y cuándo, entre otras cosas más. Usar un control de versiones, si está en la nube, permite también tener una copia de los archivos en cualquier momento.



En un comienzo (y aun en la actualidad) el control de versiones se realizaba de forma local mediante la generación de directorios o carpetas renombradas con la fecha en la cual se realizaban cambios en los archivos involucrados en el desarrollo, método que generalmente lleva a la confusión y a la generación de errores, debido a que podemos tener una multitud de carpetas y archivos acumulados a lo largo del tiempo.
Todos estos inconvenientes a la hora de desarrollar software le dan vida a los sistemas de control de versiones actuales, primero a sistemas de control de versiones centralizados CVCS (Centralized Version Control Systems) como Subversion y Perforce en los que todos los archivos son almacenados en un servidor central desde donde los desarrolladores pueden descargar cada uno de los archivos a editar. El problema de estos sistemas es que sucede cuando falla el servidor central. No hay copias locales de esos proyectos.

Con el tiempo, comienzan a surgir sistemas de control de versiones distribuidos DVCS (Distributed Version Control Systems) como Git, Mercurial, Bazaar o Darcs donde los desarrolladores clonan todo el repositorio directamente en su equipo, convirtiéndose en un respaldo completo del proyecto original y de tener la posibilidad de restaurarlo en caso de fallos o inconvenientes con el servidor.



Nota

Podemos hacer una analogía cuando dentro de un video juego, cada cierto tiempo, vamos grabando partidas del mismo. Cada vez que grabamos, guardamos una versión de nuestro juego, para no iniciar nuevamente desde el principio, y poder donde nos quedamos. En algunos juegos podemos "gestionar" y regresar a cualquier punto o "versión" del pasado donde hayamos grabado nuestro juego. Eso es un control de versiones.

GIT

Git es uno de los temas que es necesario dominar en la vida profesional de un desarrollador, es muy probable que la mayoría de empresas y proyectos con los que se vea involucrado usen Git en su día a día. GIT, un sistema de control de versiones gratuito, de código abierto, desarrollado por Linus Torvalds, uno de los principales desarrolladores del kernel de Linux, para manejar todo tipo de proyectos con rapidez y eficiencia. Actualmente es el control de versiones usado para los siguientes proyectos: el kernel de Linux, el Sistema Operativo Android, Bitcoin, Twitter, Netflix, VLC y muchos más.
GIT se presenta como un sistema distribuido, en el que todos los nodos manejan la información en su totalidad y por lo tanto pueden actuar de cliente o servidor en cualquier momento, es decir, se elimina el concepto de "centralizado". Esto se logra gracias a que cada vez que se sincroniza los cambios con el repositorio remoto, GIT, guarda una copia entera de los datos con toda la estructura y los archivos necesarios.
Así ya no es necesario salir a Internet para consultar los cambios históricos sobre un archivo o para ver quién fue la última persona que lo editó, todo se hace directamente sobre la copia local y luego, cuando lo considere oportuno, puede enviar esos cambios hacia el repositorio remoto.
Otra gran diferencia con otros scvs que almacenan los archivos originales, conservando una lista de los cambios realizados a dichos archivos en cada versión, Git guarda una "foto" (snapshot) del estado de cada archivo en un momento concreto. Si uno de los archivos no ha cambiado no crea una nueva copia del mismo, simplemente crea una referencia al archivo original.
La segunda es la eficiencia. Dada su naturaleza distribuida, una vez que tengamos nuestro repositorio en forma local en nuestro working directory, podemos trabajar sin necesidad de conexión permitiendo que la velocidad de proceso dependa únicamente en los recursos locales. Todos los cambios se irán almacenando en forma local y cuando lo consideremos necesarios publicaremos los cambios sobre el repositorio en el que se sincronizan los cambios.

¿COMO TRABAJAR CON GIT?

Para descargarse la versión para Sistemas Operativos debe dirigirse al siguiente sitio web: https://git-scm.com/. Esta versión tiene tanto la versión de línea de comando como la interfaz gráfica de usuario estándar.

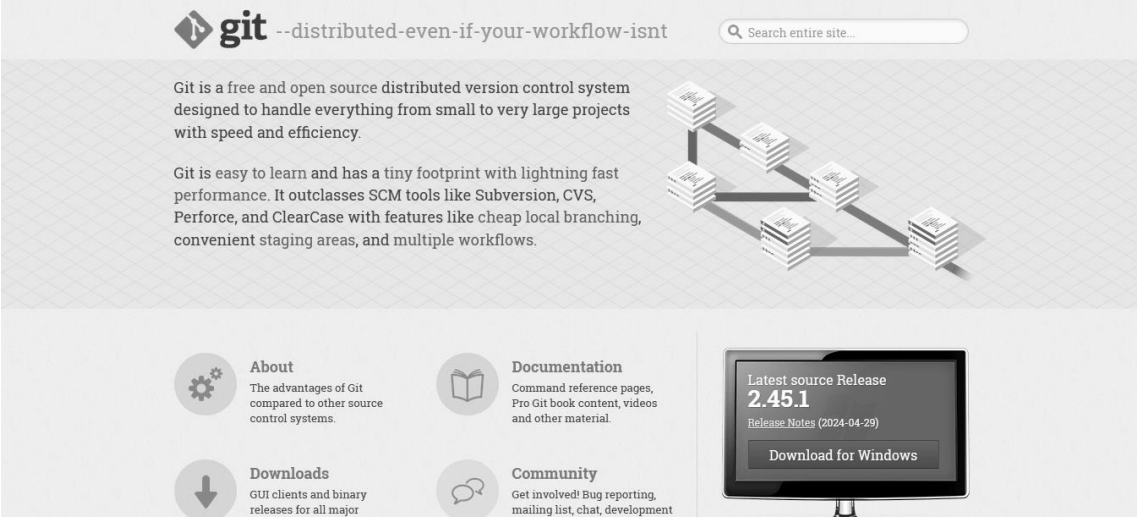


Figura 6.1. Sitio de descarga de GIT.

Luego deberá realizar un doble click sobre el archivo descargado para comenzar a instalarlo en su equipo. Luego apruebe la GNU General Public License haciendo un click sobre el botón Next.

Figura 6.2. GNU General Public License.

El instalador le solicitará una ubicación de instalación. Deje el predeterminado a menos que desee cambiarlo y haga clic en Next.

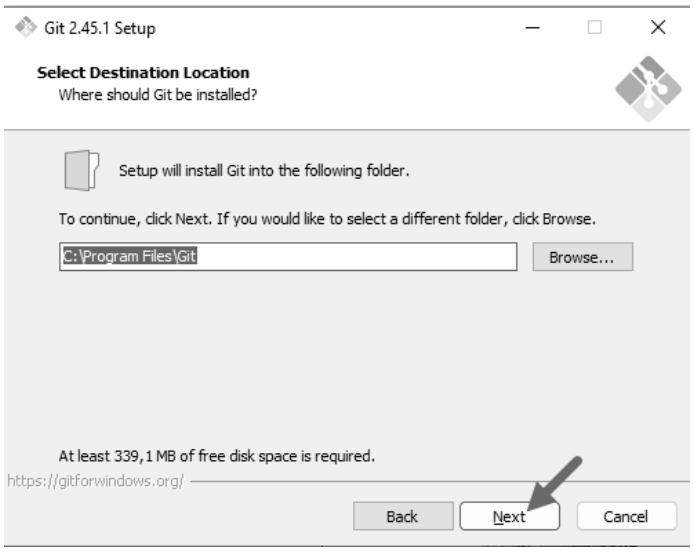


Figura 6.3. Path para instalar GIT.

En la pantalla de selección de componentes, deje los valores predeterminados a menos que necesite cambiarlos y haga clic en Next.

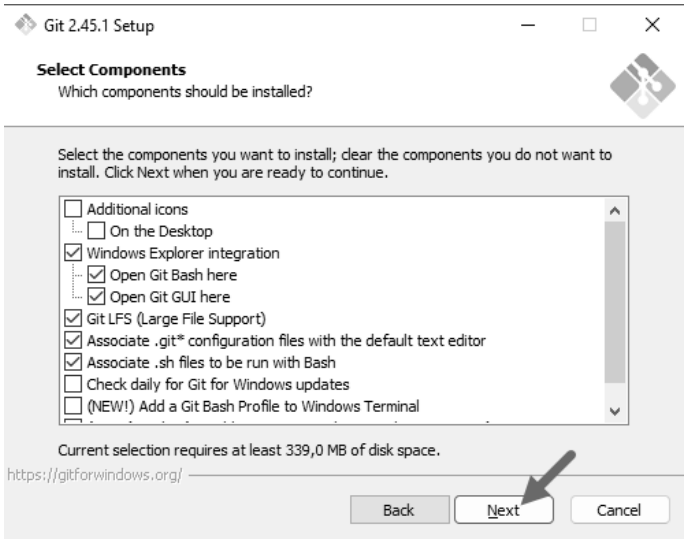


Figura 6.4. Instalación de componentes de GIT.

El instalador ofrece crear una carpeta en el menú de inicio. Haga click en Next para aceptar y continuar con el siguiente paso.

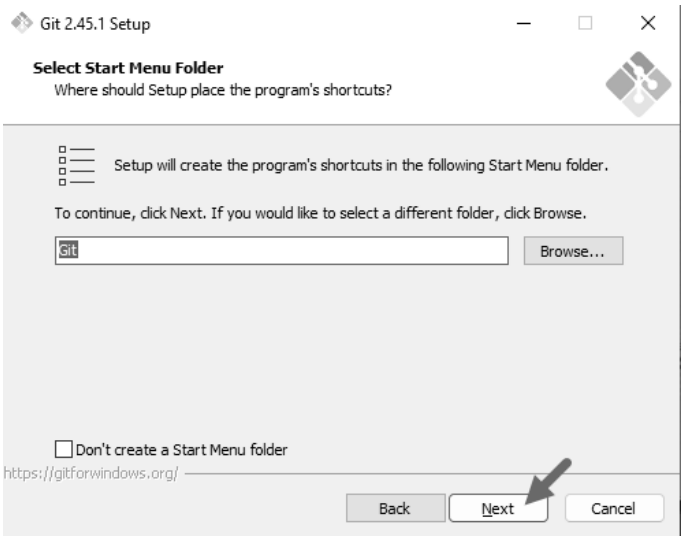


Figura 6.5. Agregar GIT en el menú de inicio.

Seleccione un editor de texto que desee usar con Git. Utilice el menú desplegable para seleccionar Visual Studio Code (o el editor de texto que prefiera) y haga clic en Next.

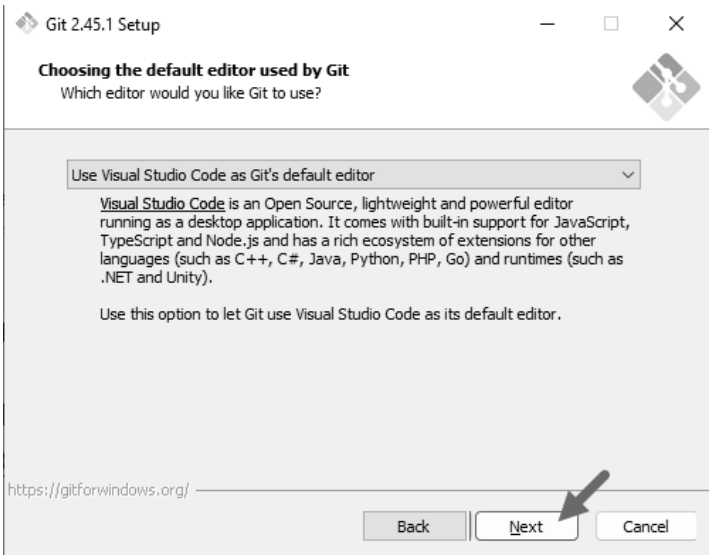


Figura 6.6. Determinar el Editor donde se usará GIT.

El siguiente paso le permite elegir un nombre diferente para su rama inicial. El valor predeterminado es main deje la opción predeterminada y haga clic en Next.

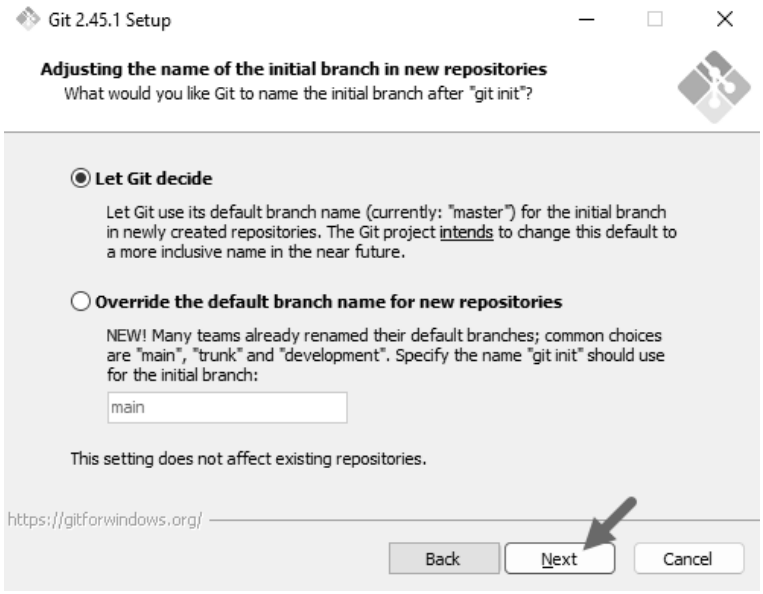


Figura 6.7. Selección de la rama inicial.

El siguiente paso le permite cambiar el conjunto predeterminado de directorios que se incluye cuando ejecuta un comando desde la línea de comando. Mantenga la selección recomendada y haga clic en Next.

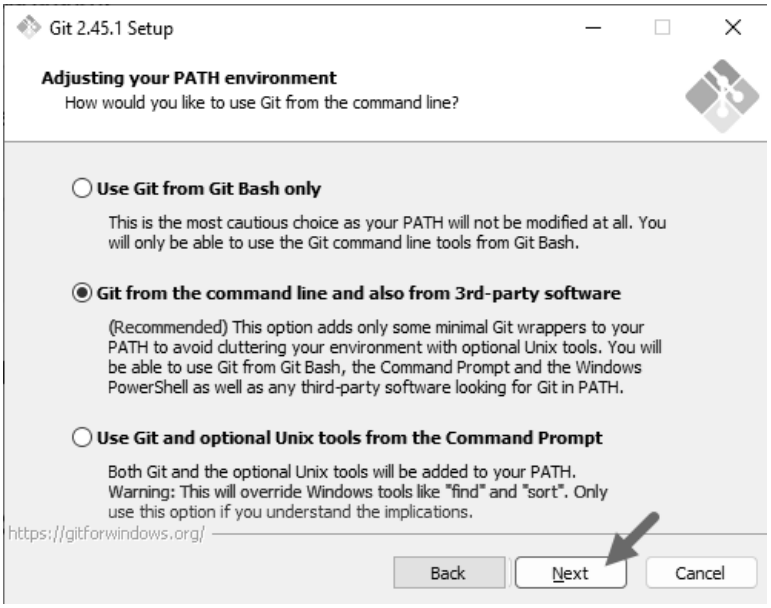


Figura 6.8. Directorios de donde se ejecutará los comandos.

El instalador le solicita que seleccione el cliente SSH que utilizará Git. Git ya viene con su propio cliente SSH, por lo que, si no necesita uno específico, deje la opción predeterminada y haga clic en Next.

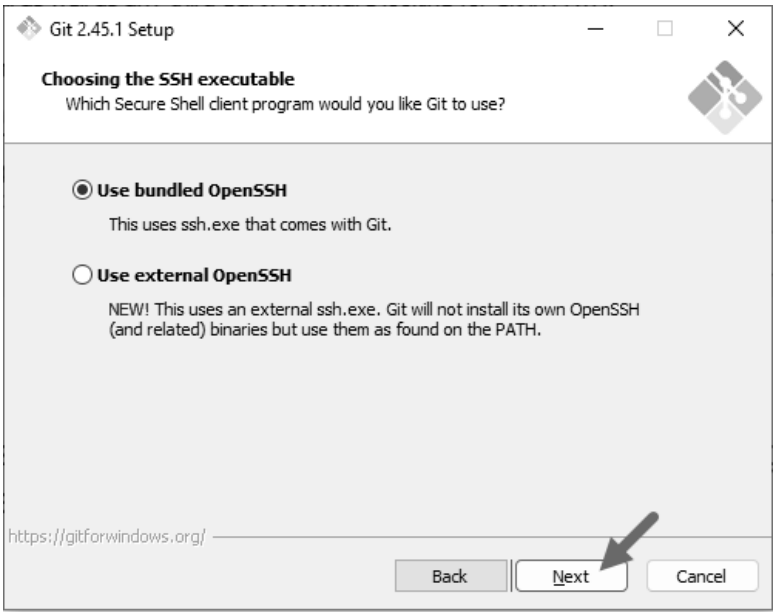


Figura 6.9. Seleccionar cliente SSH.

La siguiente opción se relaciona con los certificados de servidor. La opción predeterminada se recomienda para la mayoría de los usuarios. Haga clic en Next.

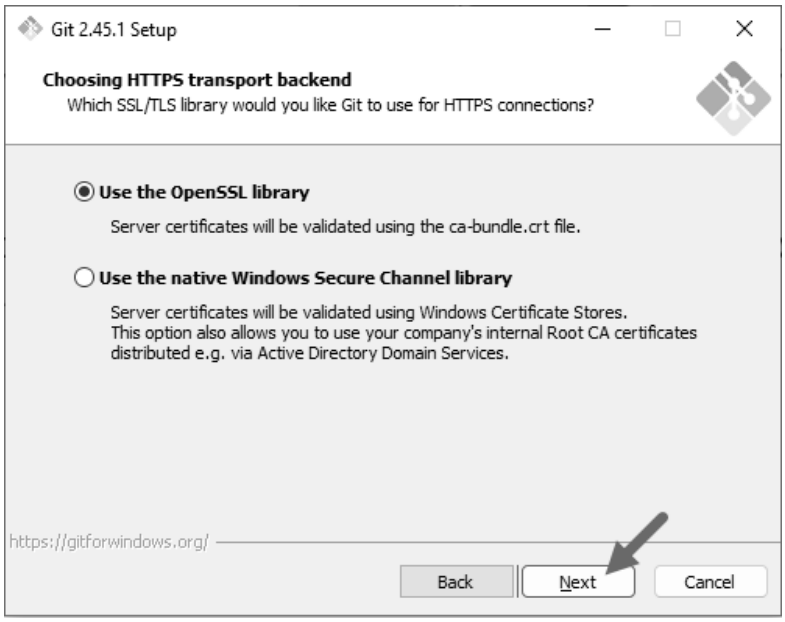


Figura 6.10. Certificados del servidor.

La siguiente selección configura la conversión de los finales de línea en archivos de texto. Puede ser de dos tipos: LF para sistemas UNIX y CRLF para Windows. Se recomienda la selección predeterminada para Windows. Haga clic en Next para continuar.

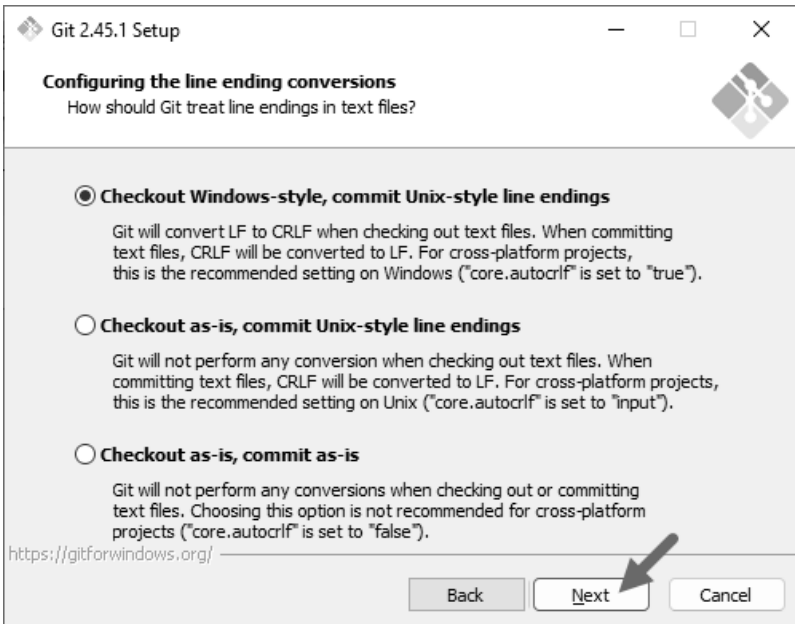


Figura 6.11. Conversión de líneas dentro de GIT.

A continuación, seleccione el emulador de terminal para Git Bash. La mejor opción es MinTTY , que se ofrece por defecto. Selecciónela y continúe con el siguiente paso.

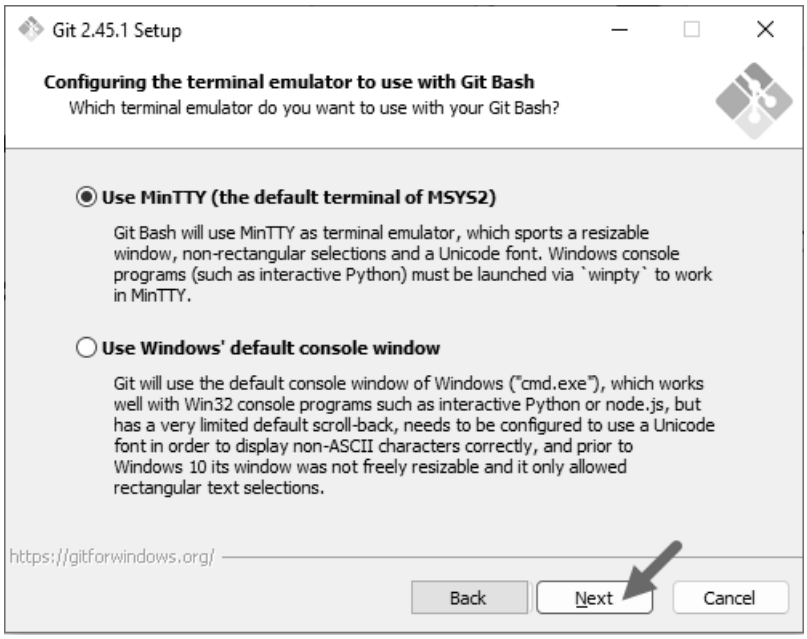


Figura 6.12. Emulador de terminal.

El siguiente paso le permite definir el comportamiento del comando git pull. Se recomienda la opción predeterminada a menos que necesite cambiar su comportamiento específicamente. Haga clic en Next para continuar con la instalación.

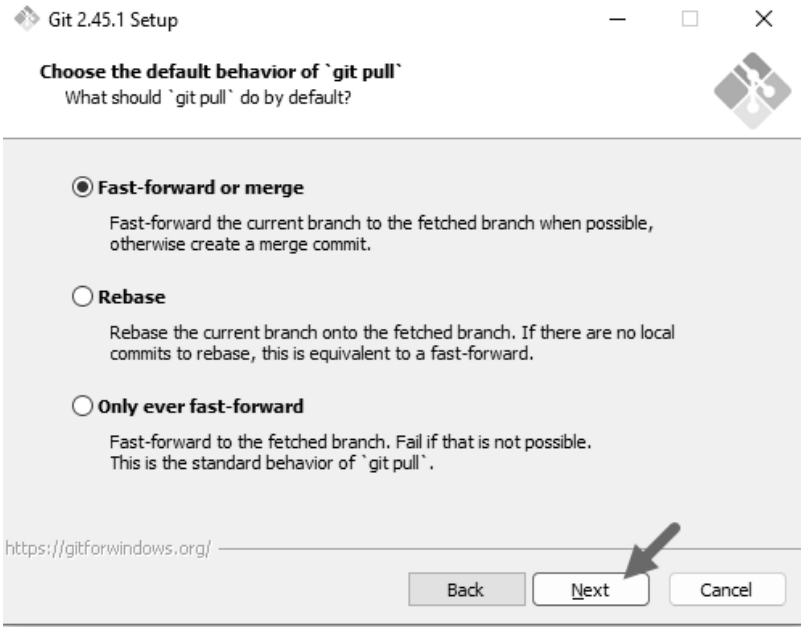


Figura 6.13. Comportamiento del comando GIT PULL.

En este paso seleccione su administrador de credenciales preferido y haga clic en Next.

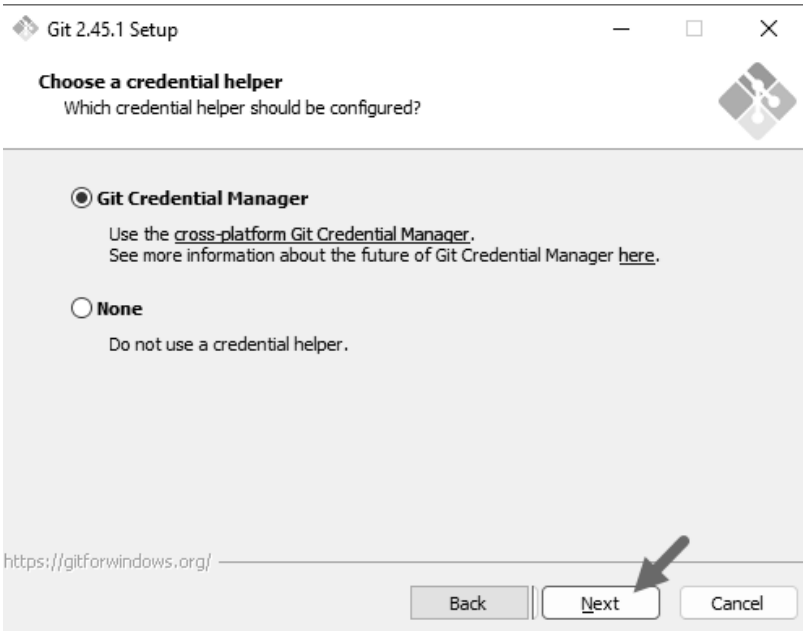


Figura 6.14. Seleccione el administrador de credenciales.

El siguiente paso le permite decidir qué opciones adicionales habilitar.

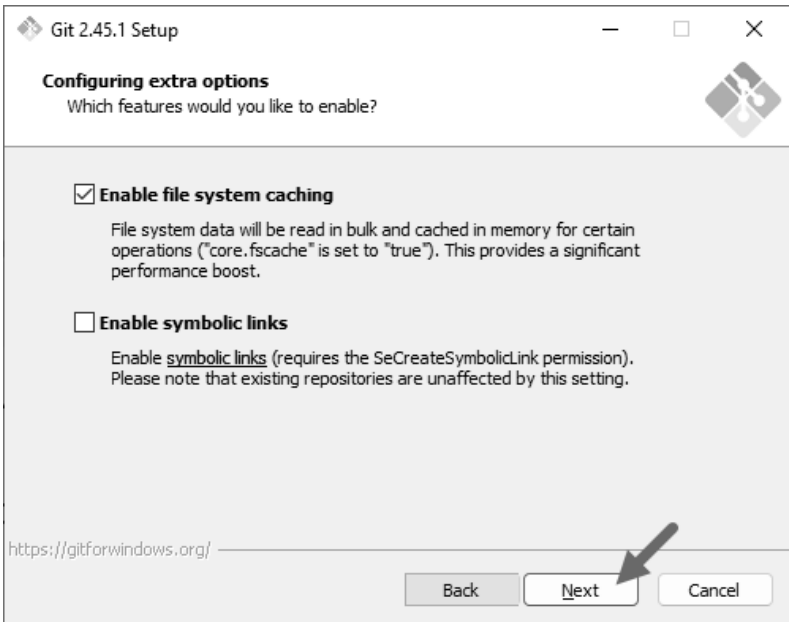


Figura 6.15. Configurando Opciones Extras.

Puede instalar funciones experimentales o de prueba. Lo recomendable es no seleccionar ninguna y haga click en install.

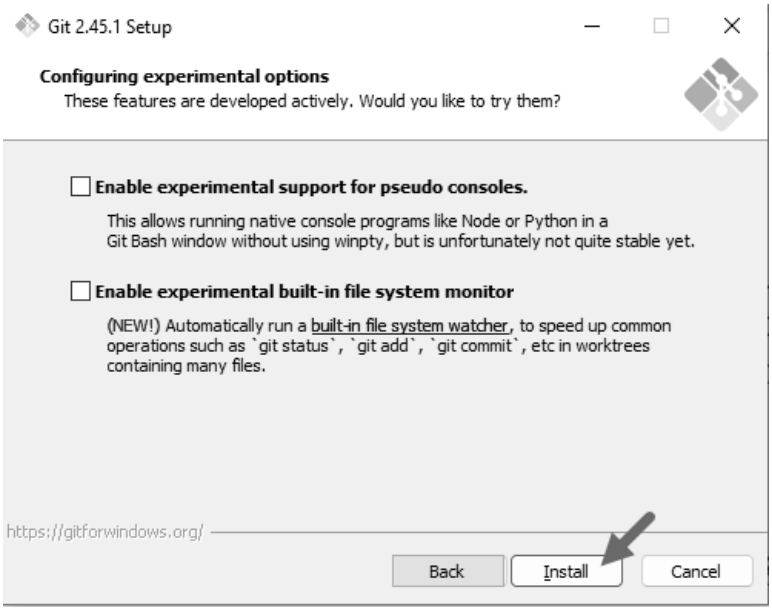
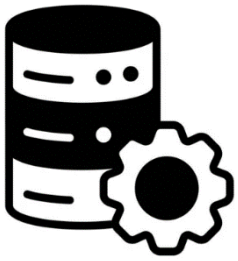


Figura 6.16. Configurando Opciones Experimentales.

Una vez que se complete la instalación haga clic en Finish. Después de la instalación, abra una ventana con la terminal de Windows y ejecute estos dos comandos.

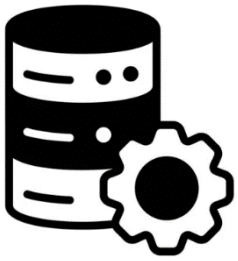
```
git config --global user.name "<your-name>"
git config --global user.email "<your-email-address>"
```

Estos dos comandos configuran las credenciales (usuario y correo) para trabajar con los repositorios, es necesario ejecutarlo la primera vez que trabaja con GIT y son necesarias para etiquetar los commits con sus datos.



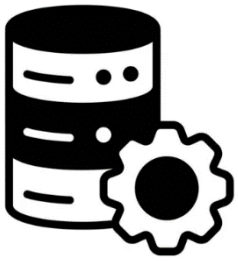
Nota

Si utiliza el parámetro --global, estos ajustes se aplicarán a todos los repositorios de su máquina. Si lo omite, sólo se establecerá la identidad del repositorio actual.



Nota

Para visualizar nuestra configuración actual, podemos usar el comando: git config --global --list



Nota

El comando git –version permite saber que versión de git tenemos instalado en nuestro equipo.

CONCEPTOS CLAVES

Para usar Git de manera efectiva, es importante comprender sus conceptos claves:

Repositorio (Repo): Un repositorio es un directorio que contiene todos los archivos y el historial de cambios de un proyecto. Puede ser local (en tu computadora) o remoto (alojado en una plataforma como GitHub, GitLab o Bitbucket).
Commit: Un commit es una instantánea de su proyecto en un momento específico. Incluye los cambios que ha realizado desde el último commit, junto con un mensaje que describe esos cambios.
Branch (Rama): Una rama es una versión paralela de su proyecto. Le permite trabajar en nuevas funciones o correcciones de errores sin afectar la base de código principal. La rama principal a menudo se llama main o master.
Staging Area (Index): Antes de hacer un commit, debe agregar los cambios al área de preparación. Esto le permite elegir selectivamente qué cambios quiere incluir en su próximo commit.
Directorio de Trabajo: Este es el directorio donde está trabajando activamente en su proyecto. Contiene los archivos reales que está editando.
Remoto: Un remoto es un puntero a un repositorio alojado en un servidor. Esto le permite colaborar con otros y hacer copias de seguridad de su trabajo.

REPOSITORIOS

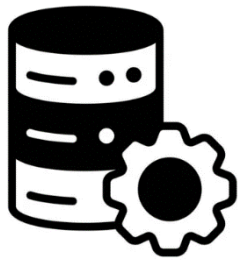
Un repositorio es un espacio físico donde se almacenan los archivos y carpetas de un proyecto con un historial de todos los cambios que han ido sufriendo a lo largo del tiempo. Existen dos repositorios principales en un proyecto Git, el repositorio local y el remoto. El repositorio local es el que cada usuario tiene en su equipo Toda su información se guarda en el directorio .git. El repositorio remoto permite colaborar con otros usuarios que van sincronizando y mezclando (merge) con el contenido de sus repositorios locales. La comunicación entre el repositorio local y los remotos se realiza mediante los comandos push (enviar) y pull (descargar).
El repositorio local cuenta con tres zonas, el primero es el directorio de trabajo (working directory) que contiene solo los archivos y carpetas del proyecto (los que estamos editando), el segundo es el INDEX (staging area o área de preparación), un área intermedia que va almacenando los archivos y carpetas que serán controlados en sus distintas versiones y finalmente con un commit lo enviamos al directorio de git o HEAD.
Por lo tanto, el flujo de trabajo básico en Git es así:

1. Modifique sus archivos en el directorio de trabajo.
2. Prepare los cambios que quiera incluir en la próxima confirmación.
3. Confirme sus cambios. (La confirmación tomará los archivos del índice y los almacenará como una instantánea en el repositorio).

Por otro lado, debido a lo que se comentó anteriormente Git tiene tres estados principales en los que se pueden encontrar los archivos: confirmado (committed), modificado (modified), y preparado (staged). Cuando creamos un archivo en VsCode en el Panel de Control de Código podemos observar que al costado del archivo aparece la letra 'U' esto significa que el archivo esta sin seguimiento, es decir que el control de versiones no está enterado de su existencia ya que no está en el repositorio.
Al hacer un click sobre el icono + a la izquierda del archivo se agrega para ser rastreado por el control de versiones. Ahora podemos ver que el costado del archivo ha cambiado por una letra 'A' que representa un nuevo archivo que se ha añadido al repositorio.
Si se confirma el archivo al realizar un commit debería notar (como es lógico) que no hay cambios pendientes, junto a ello se suele escribir un comentario indicando los cambios que se han realizado en el código. Una vez realizado un commit Git creara una nueva versión y generara una snapshot, de esta forma si en un futuro queremos revertir un cambio podremos volver a la versión anterior entre los commits realizados. Ahora si modificaremos el archivo debería aparecer la letra 'M' que significa que el archivo ha sido modificado, pero no se ha confirmado en el repositorio local.
Pero además de todo lo que vimos se pueden generar "ramas" (branches) que son líneas alternas que se generan a la par de la línea principal (master o main). Con esta característica podemos generar un branch para agregar una nueva funcionalidad a nuestro proyecto sin afectar el contenido principal, por ejemplo, se podría tener dos ramas adicionales más la principal es la rama master/main (producción), la segunda rama para desarrollo y una tercera para pre-producción.

CLONAR UN REPOSITORIO REMOTO

¿Y qué es GitHub? Un hosting online para los repositorios que utiliza Git para el mantenimiento y versionado del código fuente, añadiendo una serie de servicios extras para la gestión del proyecto y el código fuente. La versión gratuita de este hosting permite alojar nuestro código en repositorios públicos.



Nota

Tras la compra de GitHub por Microsoft en el año 2018. La empresa anuncio que se puede alojar en forma gratuita repositorios privados, pero con un número máximo de tres colaboradores por proyecto. En el caso de quiera tener más colaboradores deberá monetizar el servicio.

Clonar un repositorio extrae una copia integral de todo el proyecto alojado en GitHub que tiene en ese momento, incluyendo un registro de todo su historial para cada archivo y carpeta del proyecto. Para realizar esto debemos tener la dirección del repositorio que queremos clonar (generalmente es del tipo https://github.com/<username>/ <repo-name> .git):

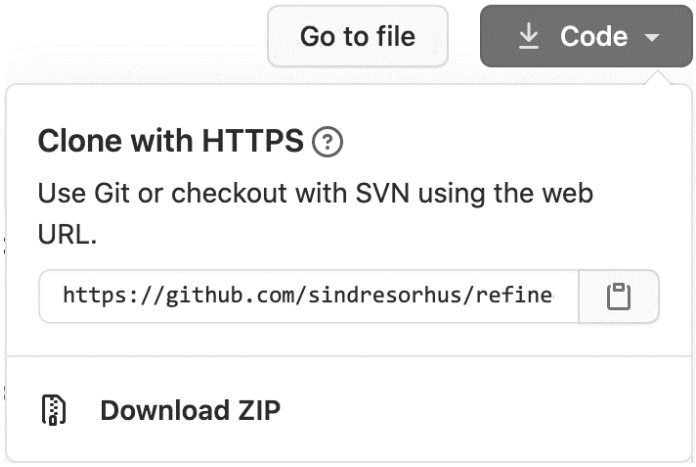


Figura 6.17. Clonar un Repositorio Remoto.

El proceso de clonación desde la terminal se realiza tipeando el siguiente comando:

```
git clone https://github.com/<username>/<repo-name>.git
```

¿QUE ES UN FORK?

Un fork (Bifurcación) es una copia separada de un repositorio que puede gestionar y modificar sin afectar al proyecto original. Esta es la diferencia con Clonar un repositorio, la clonación es el proceso de realizar una copia local de los archivos que se encuentran en un repositorio remoto. Para hacer un fork hay que tener nuestra cuenta de Github abierta y

visitar el repositorio en GitHub, para finalizar hay que realizar un click en el botón Fork situado en la parte superior derecha de la página.
En ese momento ya tendrá un fork del repositorio original que sólo estará disponible en su cuenta de GitHub. Es exactamente el mismo repositorio, hasta que empiece a hacer cambios.

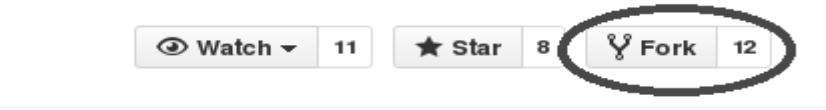
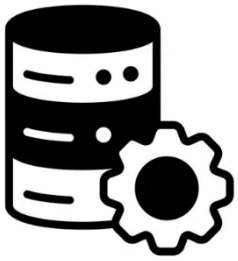


Figura 6.18. Fork de un Repositorio Remoto.



Nota

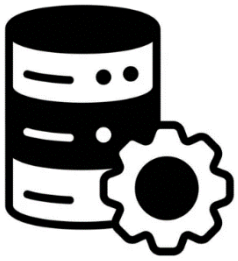
Como puede ver bifurcar un repositorio publico solo puede llevar unos segundos es la forma habitual con que se trabajan con proyectos Open Source. En el caso de que el repositorio sea privado la organización tendrá que incluirlo como colaborador antes de intentar bifurcarlo.

REPOSITORIOS LOCALES

Use el comando git init para crear un nuevo repositorio a partir de una carpeta existente en el equipo. En la línea de comandos, vaya a la carpeta raíz que contiene el código y ejecute:

```
git init
```

A partir de estos momentos VSCode creará una carpeta con el nombre. git dentro de su área de trabajo (no podrá ver esto de su Explorador de archivos de VSCode, ya que es un directorio oculto, pero puede encontrarlo en su administrador de archivos en la carpeta raíz de su proyecto).



Nota

Otra forma de tener un repositorio local (como vimos anteriormente) es clonando un repositorio ya existente de una plataforma como GitHub.

Una vez que ya tenemos un repositorio es importante saber en qué estado se encuentran los archivos que tenemos en nuestro proyecto, el comando git status nos devuelve un resumen del estado del repositorio, diciéndonos en que rama estamos, cuáles son los archivos que tienen cambios, y cuáles son los archivos que no están siendo considerados para el próximo commit. Para esto usamos el comando git status:

```
git status
```

Esta será una posible salida:

```
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        index.html
        main.js
        styles.css


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

Como vemos nos informa la rama en que nos encontramos trabajando, si se ha hecho un commit y los archivos que no se encuentra siguiendo. Con el comando git add, puede incorporar las modificaciones de un archivo en el directorio de trabajo al área de preparación. Básicamente, notifica a Git de cualquier cambio que deba incluirse en el siguiente commit:

```
git add <filename>
```

Donde filename puede ser todos los archivos, colocando un punto (.), o una lista con los nombres de los archivos junto a su extensión separados por espacios en blanco.
A pesar de esto, no ocurrirá ninguna alteración importante hasta que ejecute el comando git commit; de ahí que git add sea simplemente un paso intermedio para agregar posibles actualizaciones a consideración. Luego para realizar un commit desde la línea de comando debemos tipear lo siguiente:

```
git commit -m "Mensaje de Confirmación"
```

El comando git commit es el que realmente registra los cambios que ha realizado. Este comando se utiliza para agregar todos los cambios del área de preparación al repositorio Git local. Con este comando, también agregará una descripción de lo que se hizo con la adición de la opción -m seguido de un mensaje. Esto permite que otros colaboradores sepan qué tipo de modificaciones se incluyeron en este commit. Una vez publicado el commit, se limpiará el área de staging, y el repositorio pasará a incluir la versión más actualizada de esos archivos. Si tipeamos el siguiente comando:

```
git status
```

Esta sería una posible salida:

```
On branch master
nothing to commit, working tree clean
```

Esto indica que sobre la rama master no hay nada para comitear o sea que la rama está limpia. Para mostrar una lista de los últimos commits realizados debemos tipear lo siguiente:

```
git log --pretty=oneline
```

o también puede emplear el siguiente comando:

```
git log --oneline
```

Este comando permite ver los commits realizados agrupando cada commit en una sola línea.

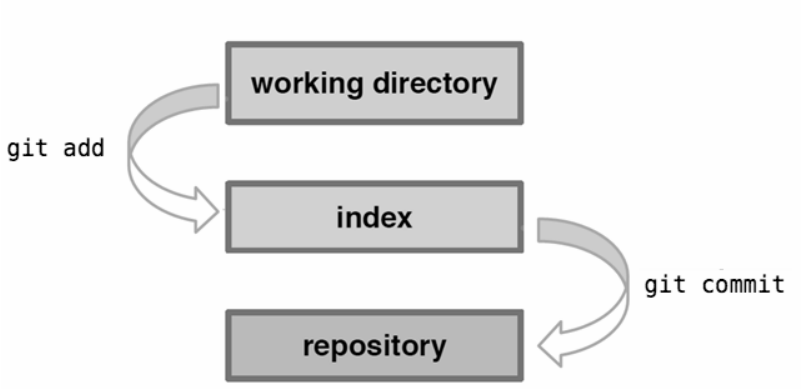
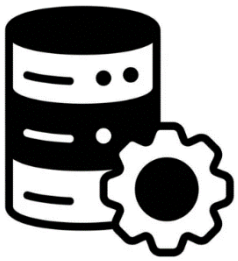
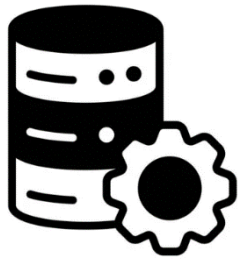


Figura 6.19. Un resumen de lo realizado con los comandos de git.



Nota

Es importante tener en cuenta que incluso cuando hace commit de los cambios, esos cambios se rastrean localmente y nada se altera en el servidor remoto hasta que se ejecuta git push



Nota

¿Cuándo conviene hacer commit? Es una pregunta que no tiene una única respuesta correcta. Cada desarrollador tiene sus criterios. En general se realiza cuando el proyecto responde a nueva funcionalidad o un gran cambio.

OMITIR ARCHIVOS EN GIT

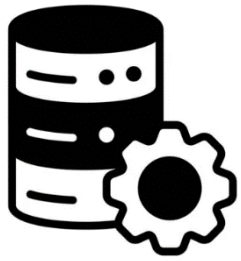
Puede indicar a Git que no realice un seguimiento de determinados archivos del proyecto agregando y configurando un archivo .gitignore. Por ejemplo, los archivos temporales del entorno de desarrollo, las salidas de prueba son ejemplos de archivos que probablemente no necesiten realizar un seguimiento. Este archivo se agrega en la raíz de nuestro proyecto y en cada línea se indica que archivo o carpeta ignorar. Por ejemplo:

```
#Archivos a Ignorar
.env
logs/
```

La primera línea (que comienza con un #) indica que es un comentario.

TRABAJANDO CON RAMAS (BRANCHES)

El branching es lo que hace que la gestión de la versión del código sea más poderosa. En lugar de un enfoque lineal, como el que hemos estado discutiendo hasta ahora, las ramas de Git permiten a los desarrolladores crear nuevas “versiones” de su base de código y realizar un seguimiento de los cambios de forma independiente. Por ejemplo, supongamos que está trabajando en una rama de característica llamada new-feature. Esta rama se utilizará para realizar un seguimiento de todos los cambios relacionados con esa característica en particular. De esta manera, puede mantener tu base de código principal (generalmente denominada main o master) separada de lo que se está haciendo en la rama de características.



Nota

Si encuentra un bug (error de código) en la rama master (que afecta al proyecto en producción), tendrá que crear una nueva rama (que usualmente se llaman bug fixing o hot fix) para hacer los arreglos necesarios. Cuando los cambios estén listos, los tendrá que fusionar con la rama main para que los cambios sean aplicados.

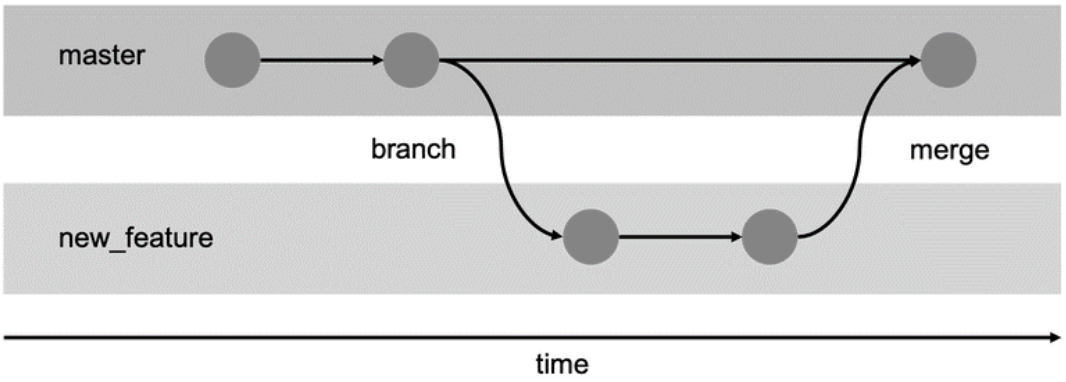


Figura 6.20. Branchs en Git.

Para saber en qué rama estamos trabajando debemos tipear lo siguiente:

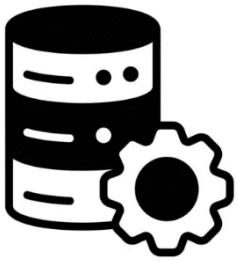
```
git branch
```

Crear una nueva rama es fácil, solo escriba este comando:

```
git branch <branch_name>
```

Si desea listar las ramas (Local o Remotas) debe tipear el siguiente comando:

```
git branch -a
```



Nota

Siempre puedes usar el comando git branch para saber en qué rama estamos. El nombre de la rama con asterisco al lado indica a qué rama está apuntado en un momento dado.

Para cambiar a la nueva rama, utilice:

```
git checkout <branch_name>
```

Vamos a hacer un ejemplo haga uno modificación sobre cualquier archivo y escriba el comando:

```
git switch -c test
```

Este comando crea la rama test y se cambia a esa rama automáticamente. El comando devolvería el siguiente mensaje:

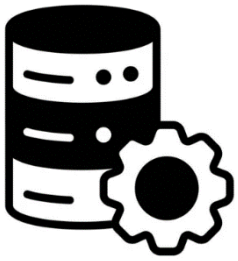
```
Switched to a new branch 'test'
```

Cuando lancemos el siguiente comando:

```
git branch
```

Devolverá el siguiente resultado:

```
 main
* test
```



Nota

Si desea cambiarse a la rama main nuevamente debe escribir el siguiente comando: git switch main

Con el comando git status veremos que aún no sean commiteado los cambios. Por lo tanto, lanzaremos el comando git add:

```
git add .
```

Y luego hacemos un commit:

```
git commit -m "Prueba nueva Rama"
```

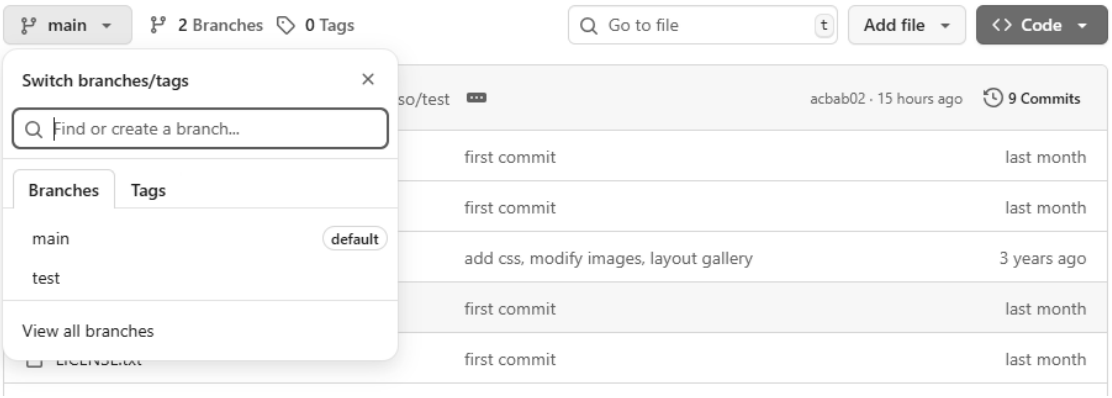
Escribimos nuevamente el comando git status y veremos la siguiente salida:

```
On branch test
nothing to commit, working tree clean
```

Finalmente hacemos un push para enviar los cambios al repositorio remoto :

```
git push -u origin4 test
```

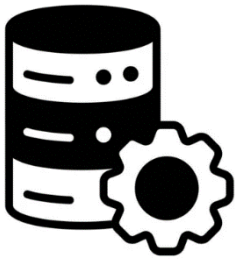
Este ultimo comando crea la rama test en el repositorio remoto y empuja los cambios a dicha rama. Ahora podemos ir a Github y ver todas las ramas incluida la rama test:



Podemos abrir el archivo modificado en ambas ramas y ver sus diferencias. Ahora incorporaremos los cambios a la rama master. Para ello haremos un click sobre el botón Compare & Pull request. Al enviar un Pull Request (PR), un desarrollador solicita que sus cambios sean revisados e integrados en el proyecto principal. Los Pull Requests permiten:

- Revisar cambios antes de integrarlos en la rama principal.
- Dejar comentarios y sugerencias sobre el código.
- Solicitar aprobaciones de otros colaboradores antes de hacer un merge.

Este proceso fomenta la calidad del código y asegura que todos los cambios sean revisados y aprobados antes de su incorporación., para ello otros usuarios revisan el código, hacen comentarios y sugieren mejoras.
Al pulsar el botón "Create Pull Request" se muestra otro formulario donde se puede escribir una descripción detallada del Pull Request. Escribiremos un pequeño comentario, por ejemplo: Test de Merge. Y Finalmente creamos el Pull Request



Nota

En la opción Reviewers de la misma ventana podemos indicar quienes son los revisores que pueden aprobar los cambios propuestos.

Una vez que se aprobó el Pull Request ya estamos habilitados para realizar el Merge usando el botón Merge Pull Request. A partir de ahora los cambios se verán afectados en la rama principal. Si una característica se descarta o ya no necesita esa rama, puede eliminarla ejecutando:

```
git branch -D <branch_name>
```

NUESTRO PROYECTO EN REMOTO

Todos los procesos de Git ocurren en forma local, sobre nuestra computadora. No obstante, podemos usar servicios como GitHub, GitLab o Bitbucket para alojar una copia de nuestros proyectos en remoto. Esto es una buena idea para tener una copia de seguridad en remoto y es imprescindible para trabajar en equipo. Cada persona del equipo puede subir sus commits al mismo repositorio en remoto. Colaborar con otros implica gestionar estos repositorios remotos, y mandar (push) y recibir (pull) datos de ellos cuando necesite compartir cosas. Una vez que ha hecho commit de sus cambios en un repositorio local, puedes enviarlo a un repositorio remoto. Esto es lo que permite a los colaboradores trabajar en el mismo proyecto y mantenerse actualizados sobre el progreso de cada uno.

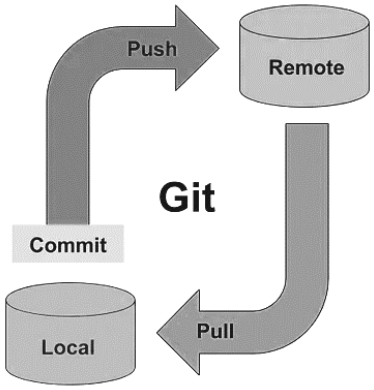


Figura 6.21. push y pull en git.

TRABAJAR CON REPOSITORIOS REMOTOS

Para crear un repositorio remoto debemos ir a Github (https://github.com/), y crearemos una nueva cuenta. Una vez creada la cuenta, debemos de terminar el proceso de registración yendo a nuestro correo electrónico y realizando los que nos piden. Luego iniciamos sesión y pedimos crear un nuevo repositorio yendo a la esquina superior derecha de la página, y luego realizando un click sobre la opción New Repository.

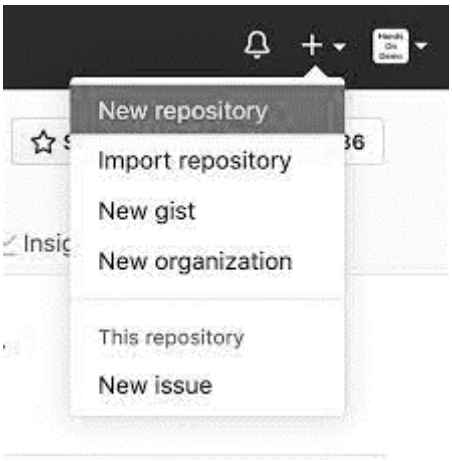
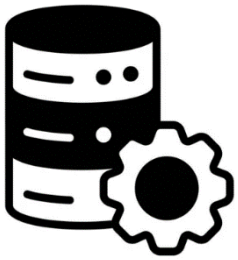


Figura 6.22. Creando un nuevo repositorio en GitHub.



Nota

A partir del año 2020 GitHub ha decidido cambiar el nombre de la rama por defecto o principal de master a main. Todos los repositorios nuevos que se creen empezarán a mostrar main como rama principal.

Luego colocamos un nuevo nombre a nuestro proyecto y una descripción y entonces hacemos "click" en "Create repository". Otra opción interesante en este momento es crear un README.md, esto es un archivo de texto que mostrara a otras personas de que se trata específicamente el proyecto. Al hacer clic en Create repository, se creará un nuevo repositorio con una URL única.
Para ver los remotos que tenemos configurados, debemos ejecutar:

```
git remote -v
```

Si no figura ningún repositorio debe agregarlo para ello debe escribir desde la terminal el siguiente comando:

```
git remote add origin (o alias_repositorio) url_del_repositorio
```

Esta línea sólo la debe ejecutar una vez por repositorio remoto. En la mayoría de los casos el nombre del remote se define por defecto en la creación del repositorio con el nombre de origin (este es un alias para no recordar la dirección completa, pero puede ser cualquier otro nombre).

Ahora si escribimos el siguiente comando:

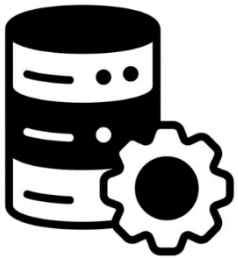
```
git remote -v
```

El resultado será:

```
origin  https://github.com/usuario/nombre_repositorio.git (fetch)
origin  https://github.com/usuario/nombre_repositorio.git (push)
```

Luego debemos modificar el nombre de la rama actual a main ya que en GitHub a partir del año 2020 la rama predeterminada de un repositorio se denomina main y pasar todo el historial que tengamos (en caso de que los haya) de master a la nueva rama:

```
git branch -M main
```



Nota

Este cambio de critério de GitHub es algo que no afecta en absoluto a Git (de hecho, en GitHub también puede establecer que el nombre de la rama principal siga siendo master). Aun así, si quiere que Git utilice «main» como nombre por defecto de la rama principal, puedes establecerlo a nivel global con este comando: git config --global init.defaultBranch main

Finalmente subimos todo a nuestro repositorio remoto dentro de la rama main:

```
git push -u origin main
```

En la primera subida, debe añadir un nombre de usuario y una contraseña. Después, Git guardará sus credenciales para que no tenga que volver a escribir la contraseña. Ahora vaya a su navegador y visite la página de su repositorio, en este momento deberá ver todos los archivos subidos. La próxima vez que se encuentre en el proyecto debe solo ejecutar:

```
git push
```

Y funcionará correctamente.

AÑADIENDO COLABORADORES AL PROYECTO

Para colaborar con un proyecto es necesario estar registrado para ese proyecto como colaborador en GitHub, para ello es necesario primero poseer una cuenta en GitHub. Luego el jefe de proyecto desde su cuenta dará acceso de escritura al repositorio a los miembros del proyecto que podrán modificar el código, de esta forma podrán funcionar los push. Desde su cuenta de GitHub deberá ir a la opción Settings y luego seleccionar su repositorio y agregar los colaboradores del proyecto.

COMO HACER UN PULL

Una vez hecho un push, cualquier integrante del equipo podría descargarse nuestros cambios escribiendo desde dentro de su copia local el siguiente comando:

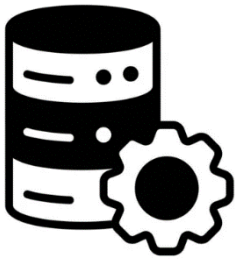
```
git pull origin
```

Si hay conflictos durante la fusión, la operación de Git Pull no será automática, y será necesario que corrija los cambios en los archivos afectados, luego deberá publicar un commit con dichas correcciones en el repositorio local, y, seguidamente, subirlos en el repositorio remoto mediante un Git Push.
Si este desarrollador quisiera revisar las modificaciones que se han realizado en el repositorio podría ejecutar:

```
git log
```

Si quisiéramos ver los cambios de un commit concreto podríamos ejecutar:

```
git show id_del_commit
```



Nota

Puede realizar un PULL en cualquier momento, pero generalmente es mejor commitear su código antes de traer los cambios del repositorio remoto.

¿COMO SUBIR UN PROYECTO CLONADO A UN REPOSITORIO DIFERENTE?

Vamos a suponer que queremos clonar un repositorio y colocar el contenido del mismo en otro repositorio distinto. En este caso primero debemos clonar el repositorio como lo hicimos anteriormente. Ahora al listar los repositorios remotos veremos lo siguiente:

```
origin  https://github.com/usuario/nombre_repositorio.git (fetch)
origin  https://github.com/usuario/nombre_repositorio.git (push)
```

En este listado podemos ver que todavía se apunta al servidor clonado, para ello debemos eliminar los repositorios de esta manera:

```
git remote rm origin
```

El próximo paso será agregar el nuevo repositorio remoto:

```
git remote add origin https://github.com/usuario/nombre_nuevo_repositorio.git
```

Ahora estaremos apuntando al nuevo repositorio, realizado esto podemos enviar el contenido al nuevo repositorio con un comando PUSH como lo hicimos también anteriormente.

VIAJE EN EL TIEMPO

Una de las características más importantes de Git es la capacidad de volver en el tiempo a estados anteriores de nuestro repositorio. Cuando creamos un commit para guardar nuestro trabajo, Git crea una ID única (también conocida hash) que nos permite mantener un registro de los cambios específicos confirmados junto con quién los realizó y cuándo. El concepto de HEAD es muy simple: se refiere al commit en el que está su repositorio posicionado en cada momento. Por regla general el HEAD suele coincidir con el último commit de la rama en la que este, ya que habitualmente estás trabajando en lo último. Pero si se mueve hacia cualquier otro commit anterior entonces el HEAD estará más atrás.

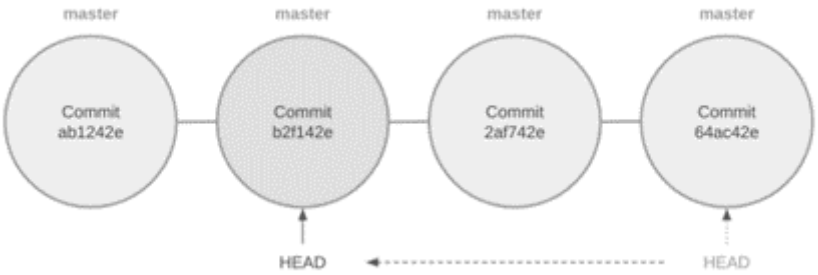


Figura 6.23. El Head de Git.

Ahora, supongamos que queremos restablecer nuestro repositorio al commit anterior. Una forma de hacerlo es cambiar temporalmente al commit anterior usando el comando git checkout.

```
git checkout <commit_id>
```

Ahora puede realizar todo lo que desea sobre ese commit sin preocuparse de perder el estado actual del proyecto. Puede volver al estado actual del proyecto escribiendo el siguiente comando:

```
git checkout <branch_name>
```

Donde branch_name es el nombre de la rama. Si la rama se llama main deberá escribir lo siguiente:

```
git checkout main
```

En cambio, si queremos descartar los cambios del commit anterior, usaríamos el comando git reset. La sintaxis del comando git reset para restablecer el repositorio a un commit anterior es:

```
git reset --hard <commit-sha-id>.
```

DESHACER UN COMMIT YA PUBLICADO

Ahora vamos a suponer que modificamos uno o más archivos y por error realizamos un commit y los subimos al servidor haciendo un push, pero nos dimos cuenta que esos cambios que hemos hecho no permite que la aplicación funcione correctamente. Podemos hacer un revert de los cambios. Un revert es una operación que toma un commit específico y crea un nuevo commit con el contenido del commit especificado de esta forma no pisamos nuestro historial.

```
git revert <commit-id>
```

Con git revert se crea un nuevo commit que revierte los cambios realizados en el último commit, pero no elimina dicho commit.
El comando anterior va a revertir el último commit que hemos realizado, esto será solo en nuestra computadora, para subir este cambio al servidor debemos correr también el comando git push.

¿QUE ES ARCHIVO README.MD?

El README es el punto de entrada a un proyecto, su función principal es explicar al público en general de que trata, como funciona y que se necesita para usarlo. Para agregar un README debe crearse un archivo README.md en la raíz de su proyecto. Este archivo es creado con el lenguaje Markdown. Markdown es un lenguaje de marcas como HTML o XML creado por John Gruber que nos permite crear un README de manera sencilla. Markdown es simple, pero puede hacerlo visualmente atractivo. A continuación, mostraremos una estructura simple de cómo debería verse un archivo README.md:

```
# Descripción del proyecto

 Comience con una breve descripción del proyecto, incluyendo su propósito y
características clave.

## Instalación

 Proporcione instrucciones claras sobre cómo instalar su proyecto, incluidos
los requisitos previos.

## Uso

 Explique cómo utilizar su proyecto, incluidos ejemplos de código, si
corresponde.

## Características

 Enumere las principales características o funcionalidades de su proyecto.

## Contribuir

 Invita a otros a contribuir y explicar cómo pueden hacerlo.

## Licencia

 Especifique la licencia del proyecto para aclarar cómo otros pueden usar su
código.

## Registro de cambios

 Mantenga un registro de cambios en su archivo README para realizar un
seguimiento del historial de versiones y las actualizaciones.
```