



Hechos

QUE

CONECTAN 



El futuro digital
es de todos

MinTIC

CONTROL DE VERSIONES

PARTE 1

Universidad
Industrial de
Santander



‘Mision
TIC2022’

2.1. Introducción a Repositorios Software

2.1.1. Diferencia entre un servicio de alojamiento de repositorios y un sistema de control de versiones

Es importante reconocer que los servicios de alojamiento de repositorios y los sistemas de control de versiones son dos entidades independientes. Los sistemas de control de versiones, son las utilidades de líneas de comando de bajo nivel que se emplean para gestionar los cambios del ciclo de vida de desarrollo de software, todo en una colección de archivos de código fuente.

Los servicios de alojamiento de repositorios son aplicaciones web de terceros que encapsulan y mejoran un sistema de control de versiones. Es importante resaltar que, no se puede utilizar por completo un servicio de alojamiento de repositorios sin tener que usar un sistema de control de versiones subyacente.

En resumen, un repositorio de código es un servidor con un sistema de control de versiones, que permite guardar todo nuestro programa, toda nuestra aplicación en un lugar seguro, y lo hace a través de un control de versiones que permite mantener un historial de todos los cambios que se producen.

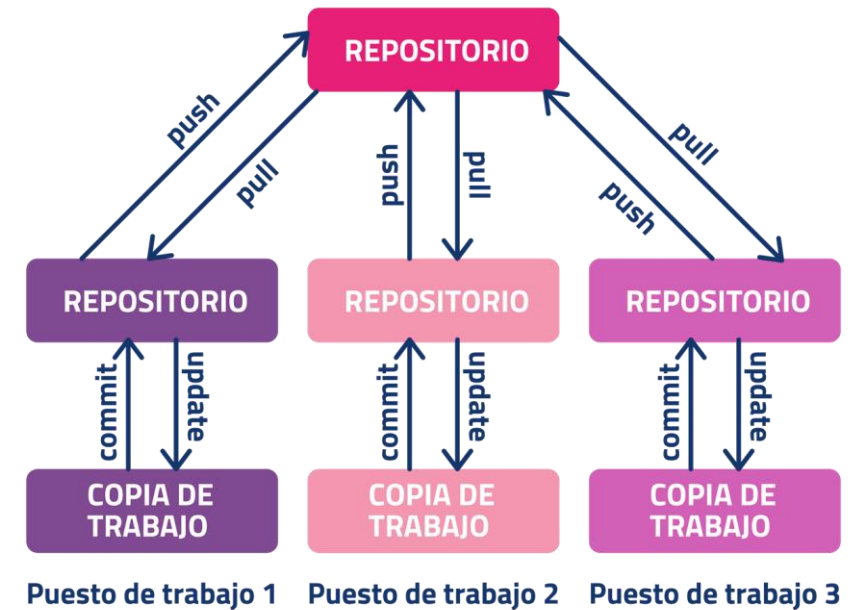
2.1.2. Ventajas de los repositorios de software

- Permite el trabajo en paralelo de dos o más usuarios de una aplicación o programa, sin que se den por válidas versiones con elementos que puedan entrar en conflicto entre sí.
- La seguridad de un repositorio de código en un servidor es máxima, ya que este garantiza la misma mediante diferentes métodos avanzados de ciberseguridad y la creación constante de copias de seguridad.
- Permite disponer y acceder a un historial de cambios. Cada programador tiene la obligación de señalar qué cambios ha realizado, quién los ha llevado a cabo y también cuándo se hicieron. Esto permite un mayor control sobre cada versión, permite corregir cambios y facilita que cada cambio ejecutado se vea como un nuevo estado.
- Facilita el entendimiento del proyecto, sus avances y el estado actual de la última versión. Algo posible gracias a que cada pequeño cambio efectuado por un programador, debe ir acompañado por un mensaje explicativo de la tarea realizada. Es así como se evita a los demás programadores perder tiempo cuestionando el porqué de los cambios ejecutados. De igual manera, facilita la corrección de errores si los hubiera y son detectados por otro programador.

2.2. Sistema de control de versiones distribuido

Los **Sistemas de Control de Versiones Distribuidos (en adelante SCVD)**, no dependen necesariamente de un servidor central para almacenar las versiones de los ficheros del proyecto. En los SCVD, cada programador tiene **una copia local o clon** del repositorio principal. Esto quiere decir que, cada programador mantiene un repositorio local propio que contiene todos los archivos y metadata presente en el repositorio principal. Todos pueden operar con su repositorio local sin ninguna interferencia.

Esta parte se podría incluir en el recurso educativo digital y referenciar la sección desde acá



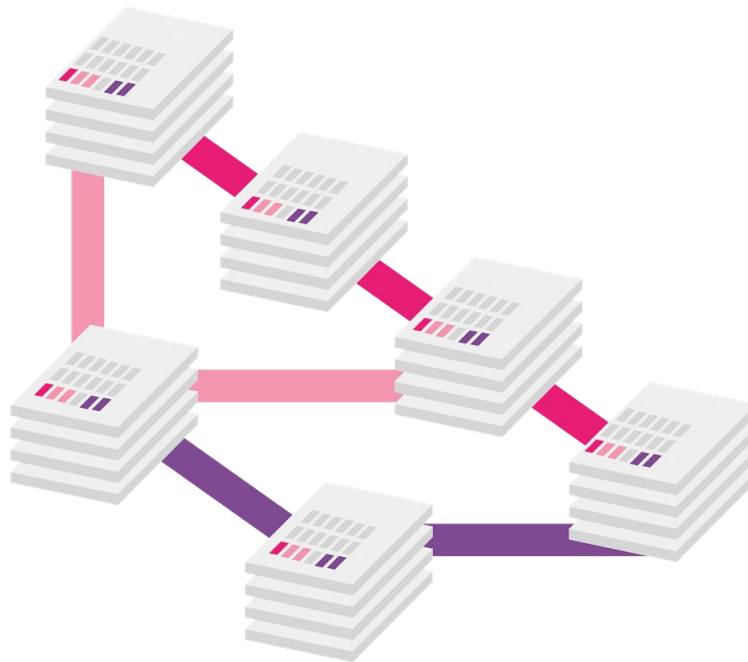
Todos los programadores pueden actualizar sus repositorios locales con nuevos datos del servidor central con una operación llamada 'pull' y persistir cambios en el repositorio principal con una operación llamada 'push' desde su repositorio local. El hecho de clonar un repositorio entero en su propio puesto de trabajo para tener un repositorio local, proporciona las siguientes ventajas:

- Todas las operaciones (excepto push y pull) son muy rápidas porque la herramienta sólo necesita acceder al disco duro, no a un servidor remoto. Por tanto, no siempre se necesita conexión a internet.
- Los nuevos cambios se pueden guardar (commit) localmente sin manipular los datos del repositorio principal. Una vez se tenga listo un conjunto de cambios, se pueden persistir (push) todos a la vez en el repositorio principal.
- Dado que cada programador tiene una copia completa del repositorio del proyecto, pueden compartir los cambios entre sí si fuera necesario obtener un feedback antes de persistir los cambios en el repositorio principal.
- Si el servidor central sufre algún percance en algún momento, los datos perdidos pueden ser recuperados fácilmente desde cualquiera de los repositorios locales de los colaboradores.

2.3. Git

Git es una herramienta software de código abierto para el control de versiones desarrollado por Linus Torvalds (Precursor del Kernel del Sistema Operativo Linux), diseñado para manejar con rapidez y eficiencia, desde proyectos pequeños hasta proyectos muy grandes. Adicionalmente, es una herramienta fácil de aprender, que ocupa poco espacio y de alto rendimiento, que supera a herramientas similares como Subversion, CVS, Perforce, ClearCase, entre otras.

Esta parte se podría incluir en el recurso educativo digital y referenciar la sección desde acá



A diferencia de otras herramientas que almacenan los datos como una lista de cambios basados en archivos. Git maneja sus datos como una serie de capturas instantáneas de un sistema de pequeños archivos donde cada vez que confirma o guarda el estado de un proyecto, Git almacena una referencia a esos archivos captados. Para ser eficiente, si los datos no han cambiado, únicamente se establece un enlace al archivo anteriormente almacenado.

Lo importante a tener en cuenta, es que Git tiene tres estados principales en los que pueden residir sus archivos: modificado (modified), preparado (staged) y confirmado (committed).

Modificado: significa que ha cambiado el archivo, pero aún no lo ha confirmado en su base de datos (archivo .git).

Preparado: significa que ha marcado un archivo modificado en su versión actual para pasar a su próxima captura de confirmación, es decir, falta la confirmación (commit).

Confirmado: significa que los datos se almacenan de forma segura en su base de datos local (archivo .git).

Esto nos lleva a tres secciones principales de un proyecto de Git: el Directorio de Trabajo (Working Directory), Área de Preparación (Staging Area) y Directorio de Git (.git Directory (Repository)), como se muestra en la siguiente Figura:

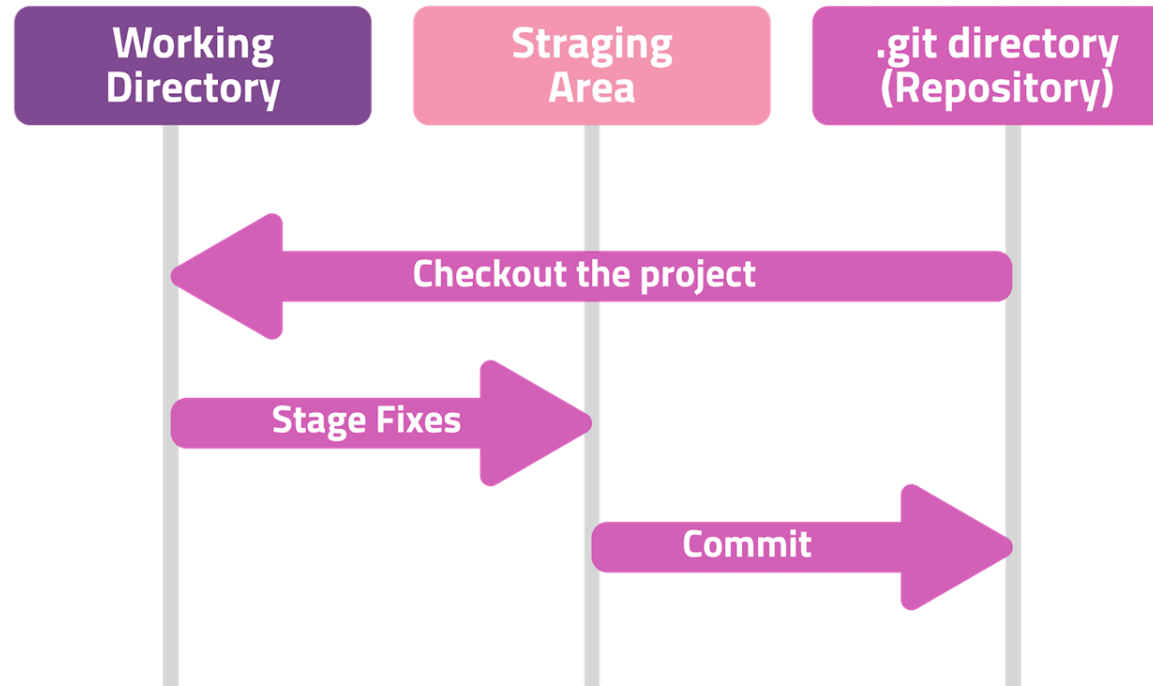


Imagen tomada de <https://git-scm.com/book/en/v2/Getting-Started-What-is-Git%3F> Imagen tomada de <https://git-scm.com/book/en/v2/Getting-Started-What-is-Git%3F>

Figure 6. Working tree, staging area, and Git directory.

El Directorio de Trabajo (Working Directory), es una comprobación única de una versión del proyecto. Estos archivos se crean o extraen de la base de datos comprimida en el directorio Git y se colocan en el disco para que los utilice o modifique.

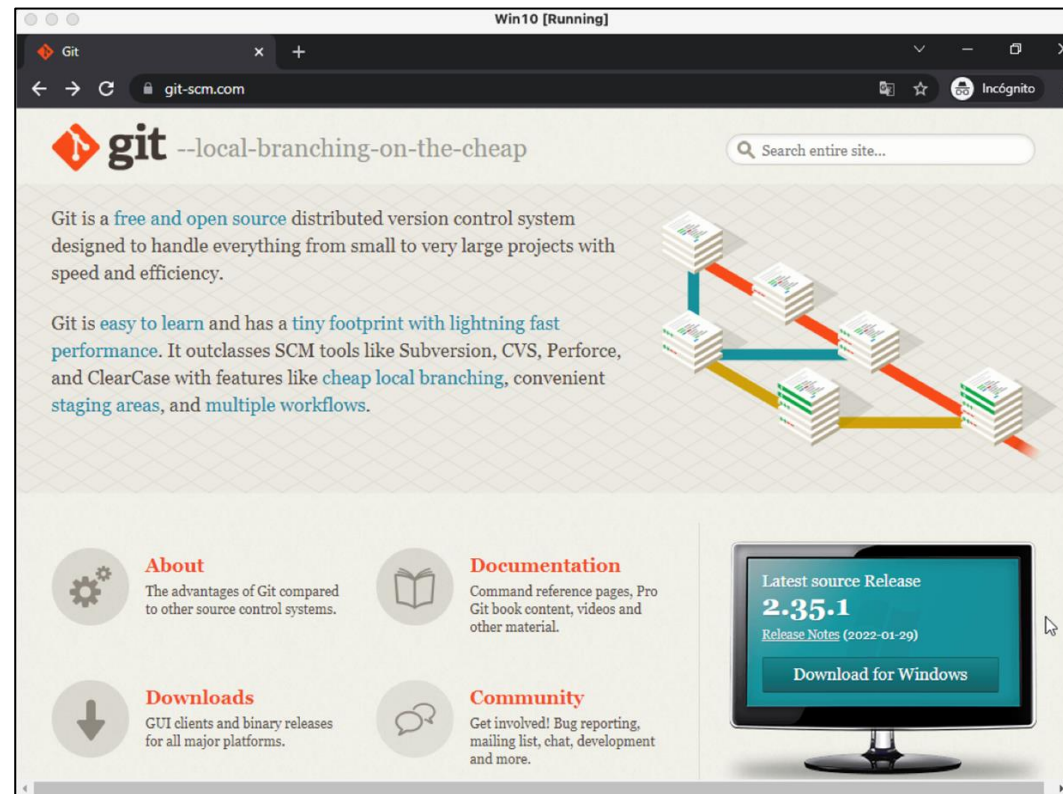
El Área de Preparación (Staging Area) es un archivo, generalmente contenido en su directorio Git, que almacena información sobre lo que se incluirá en su próxima confirmación. Su nombre técnico en la jerga de Git es "index", pero la frase "staging area" es la más empleada.

El Directorio .Git (.git Directory) es donde Git almacena los metadatos y la base de datos de objetos para su proyecto. Esta es la parte más importante de Git y es lo que se copia cuando clonas un repositorio desde otra computadora, o un repositorio remoto.

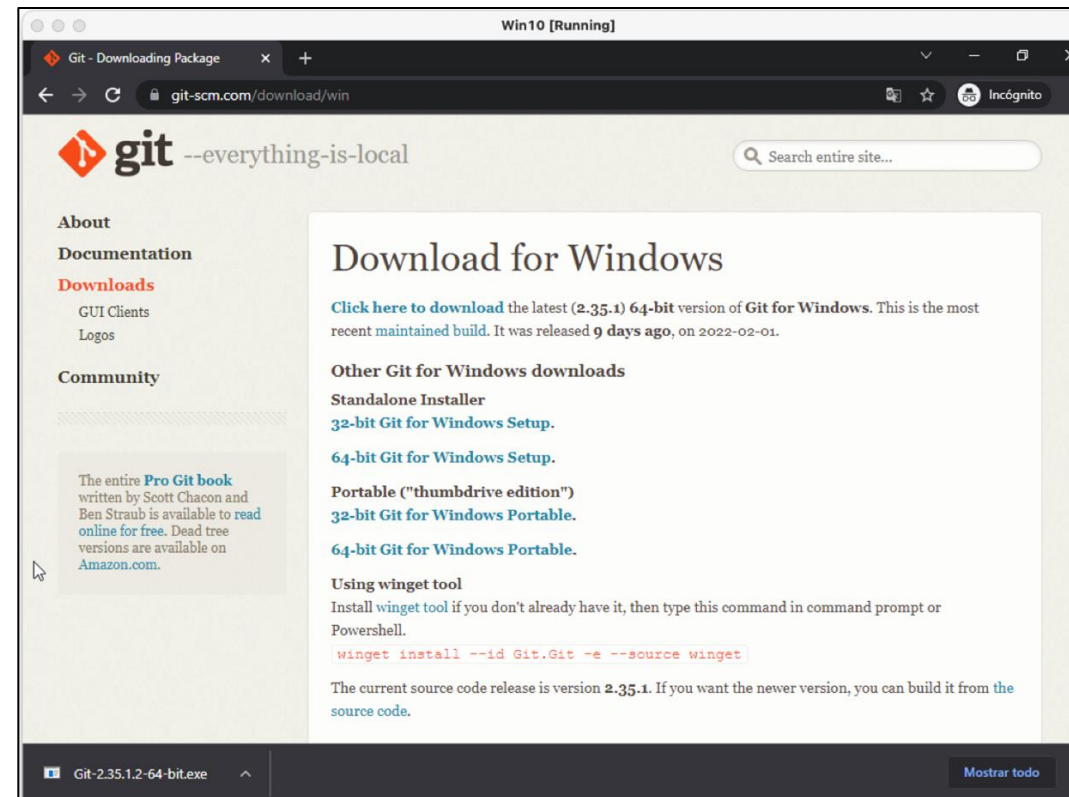
La mayoría de operaciones de Git se ejecutan en la máquina local, de tal forma que, todo el historial de su proyecto está en la base de datos de su disco local (directorio .git), permitiendo un alto rendimiento en la ejecución de sus funciones. Así mismo, para compartir el proyecto con otros miembros del equipo, se utilizan repositorios remotos en plataformas de social coding como GitHub y GitLab, entre las más populares.

2.3.1. Instalar Git

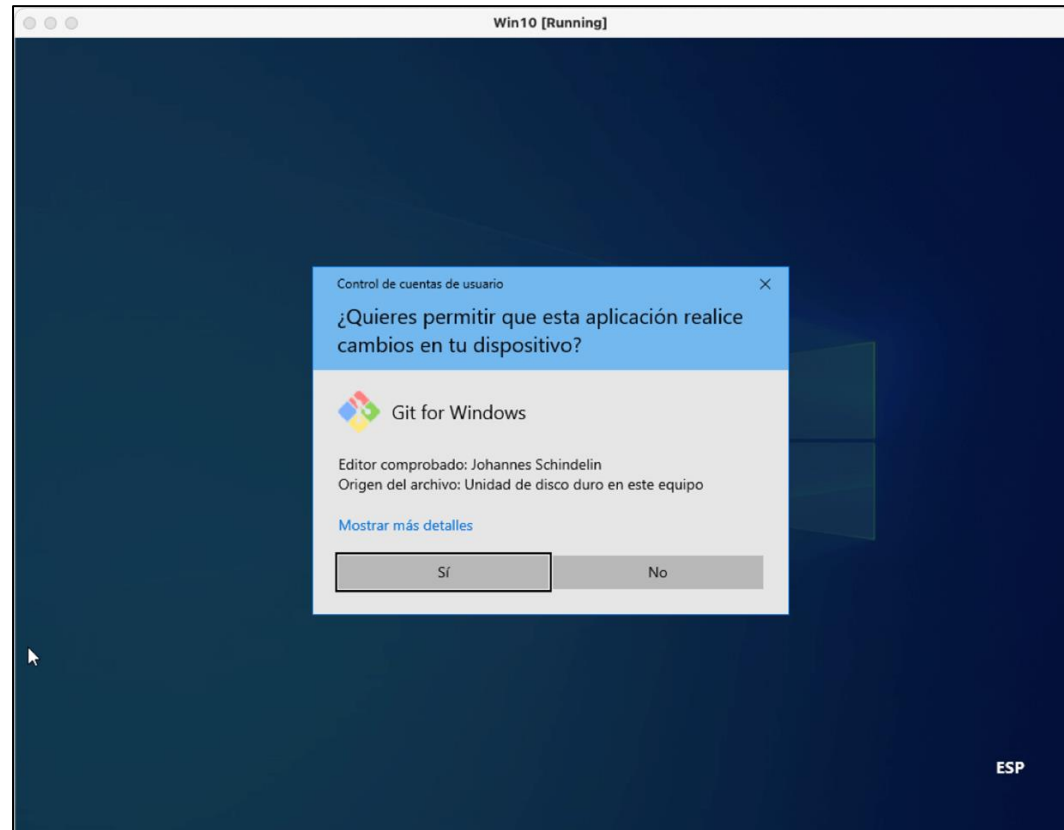
Para descargar Git, se ingresa a la página oficial <https://git-scm.com/> y descarga (download) el instalador correspondiente a su sistema operativo, para este caso en particular, la siguiente secuencia de pasos corresponde a la instalación de Git en Windows 10, donde se comentan las capturas de pantalla, es aquí donde hay cambios a tener en cuenta, y las demás se dejan por defecto al paso siguiente (next).

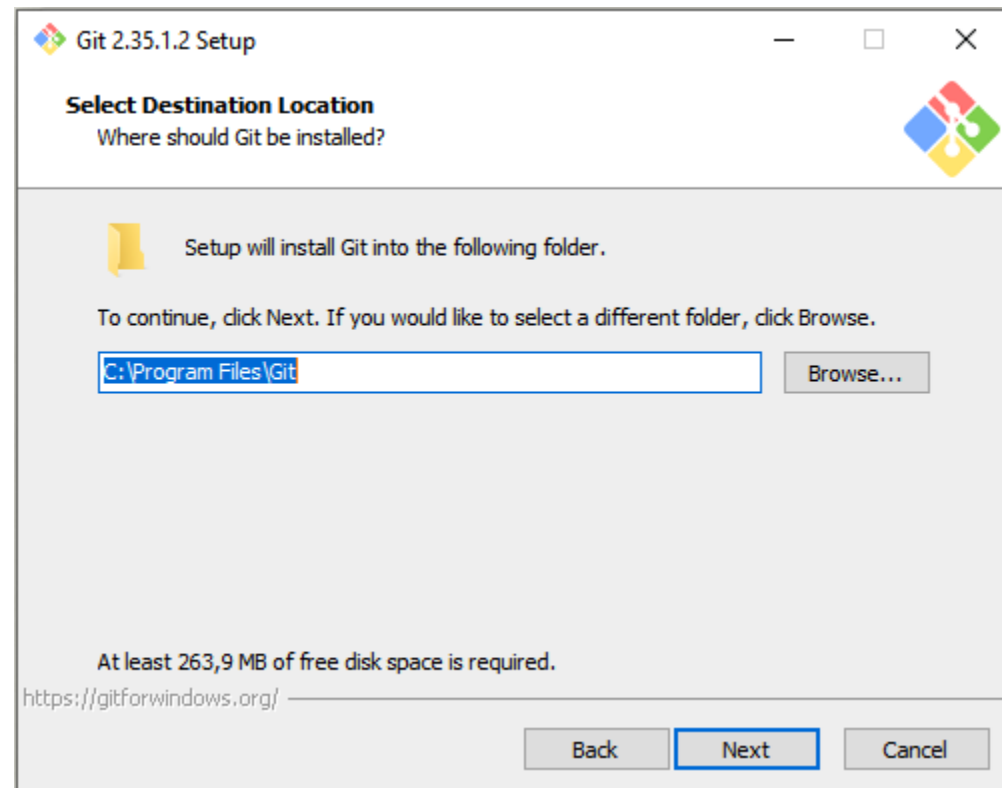
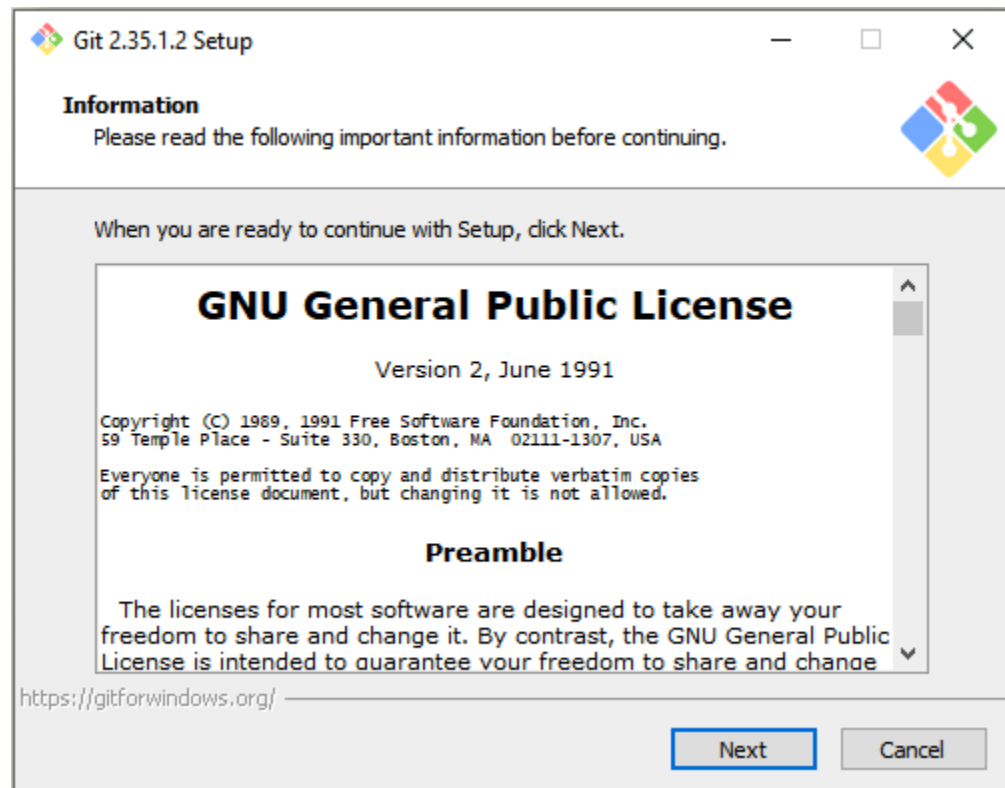


Descargar el instalador correspondiente a su sistema operativo, sea de 32 bits o 64 bits.

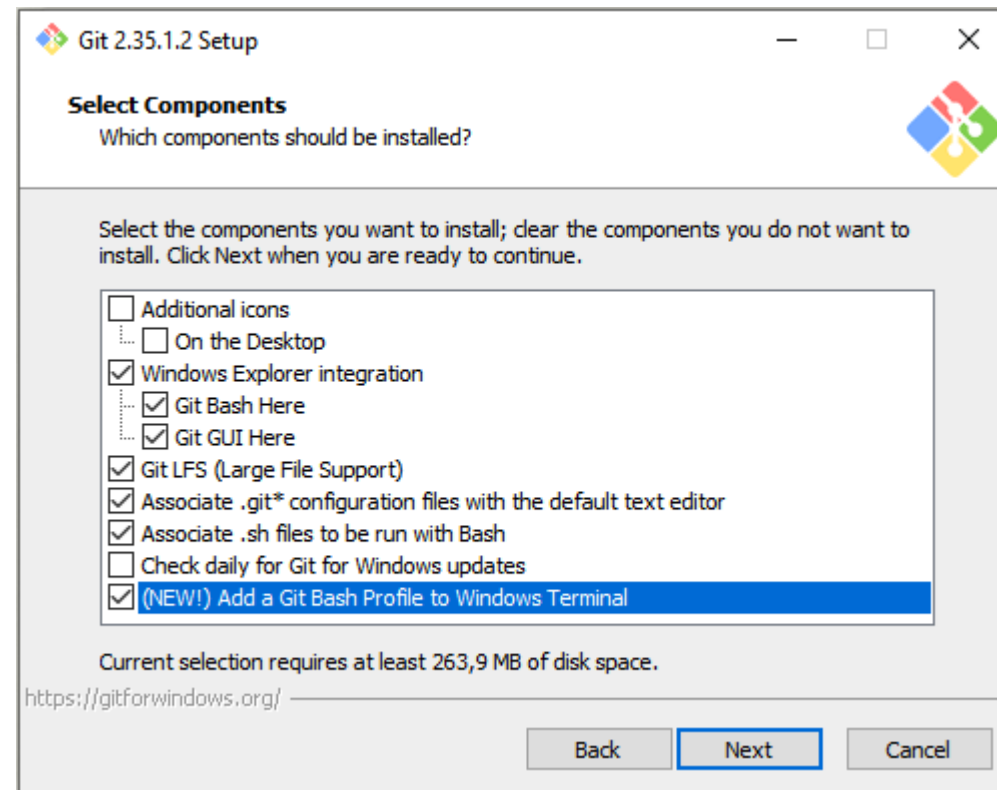


Permitir instalación de Git para Windows.

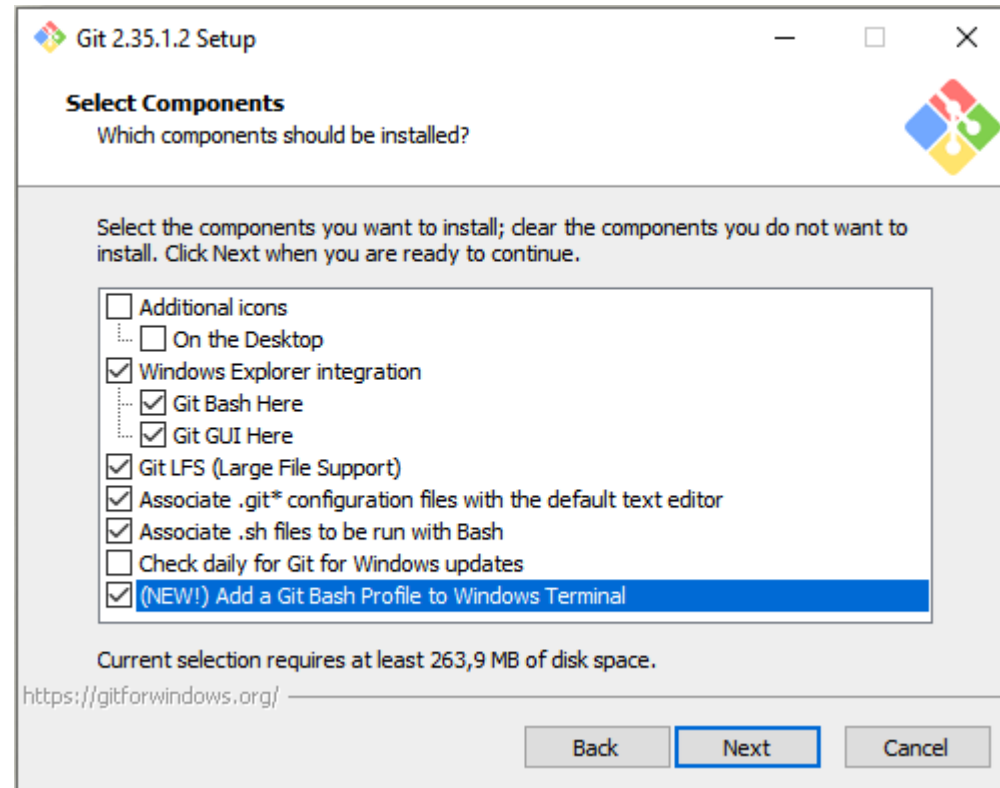


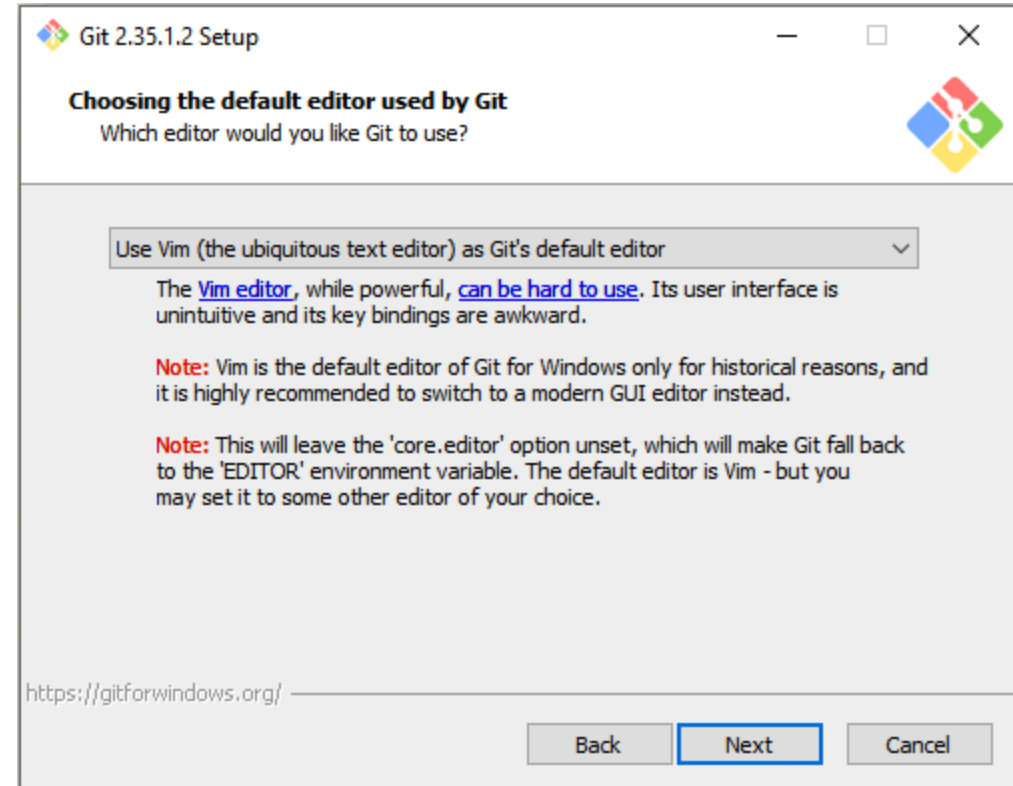
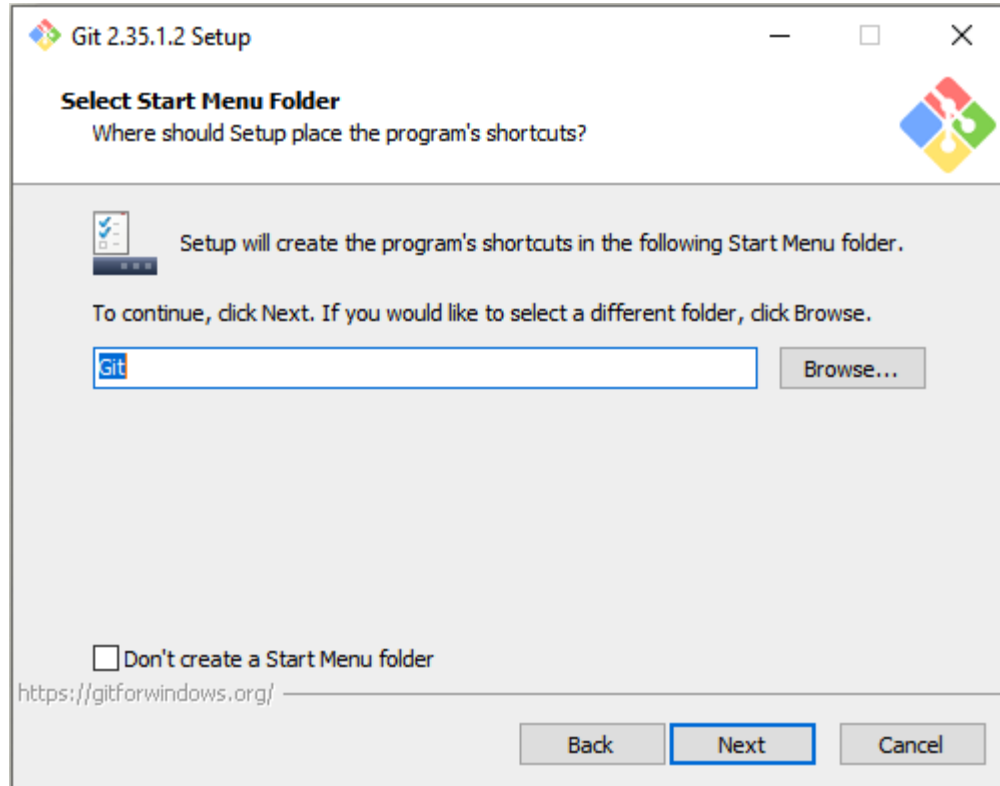


Seleccionar/Activar CheckBox "Add a Git Bash Profile to Windows Terminal" como interprete de comando de Linux en Windows.

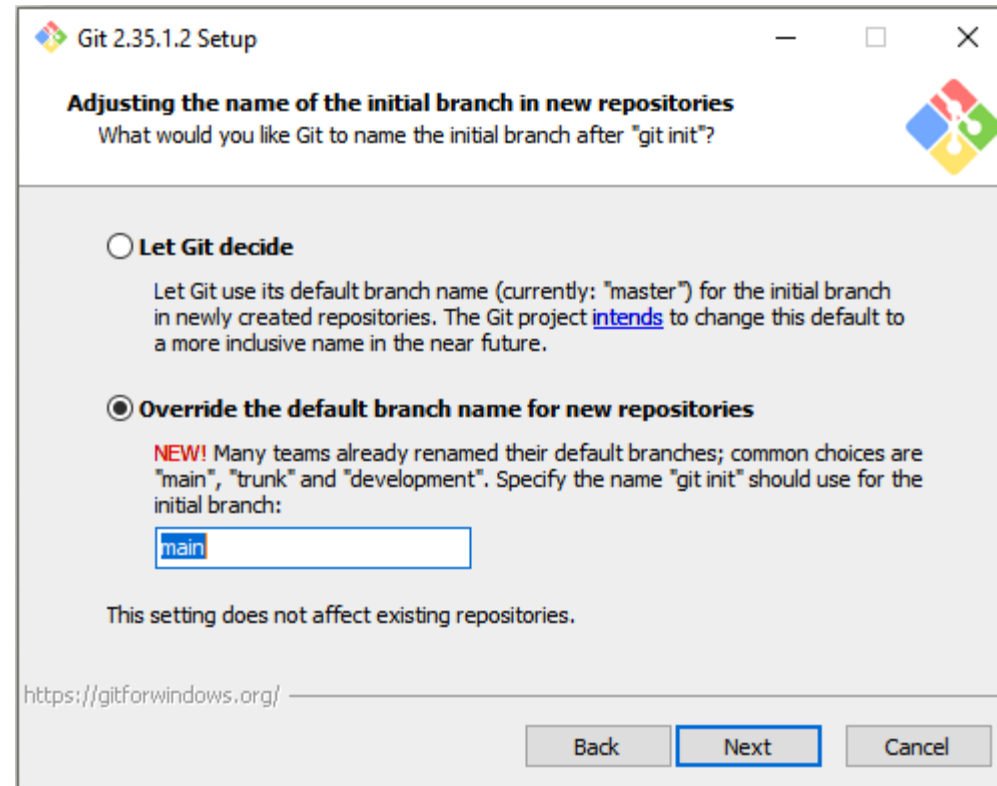


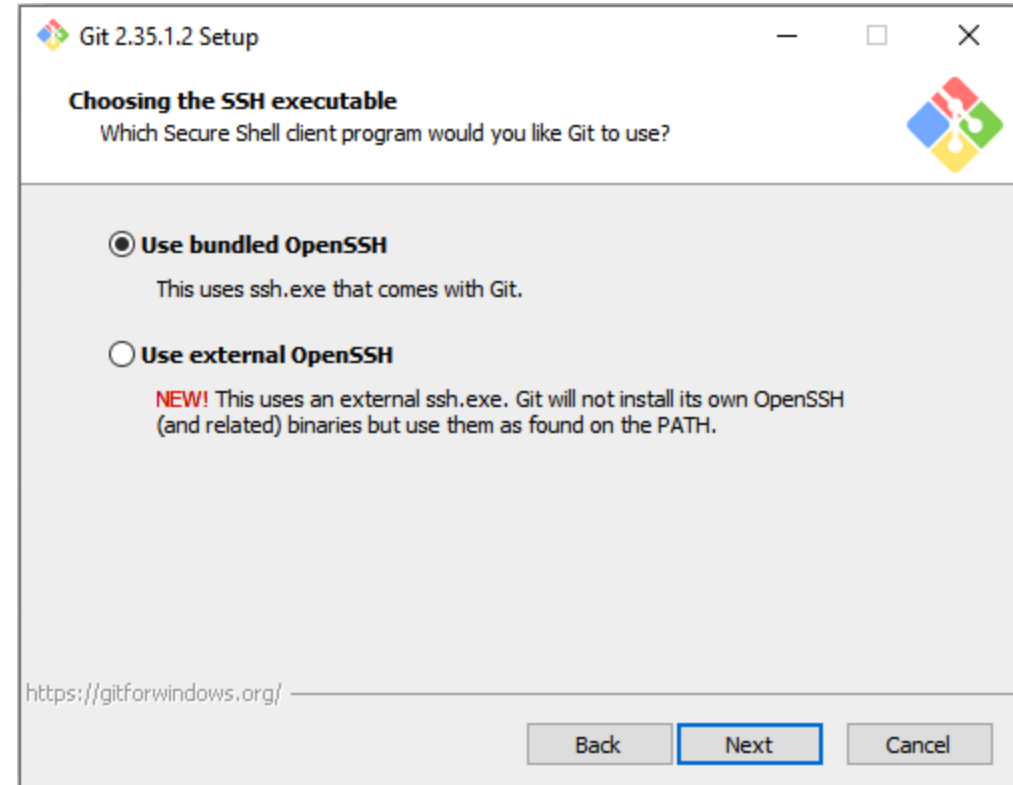
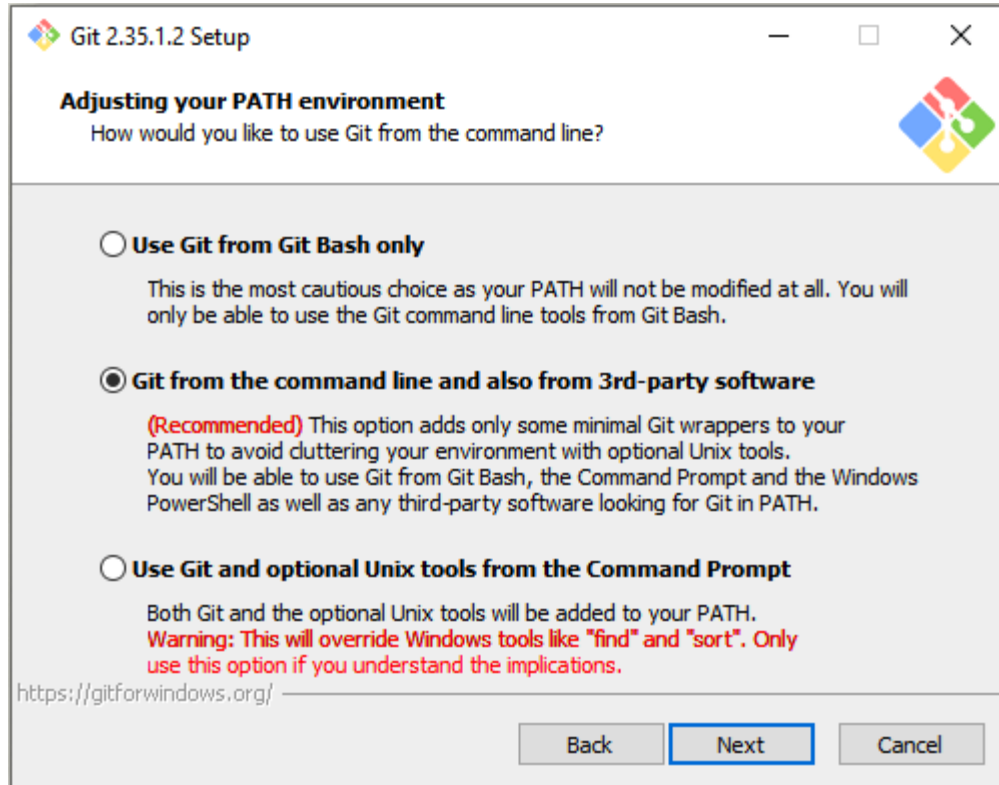
Seleccionar/Activar CheckBox "Add a Git Bash Profile to Windows Terminal" como interprete de comando de Linux en Windows.

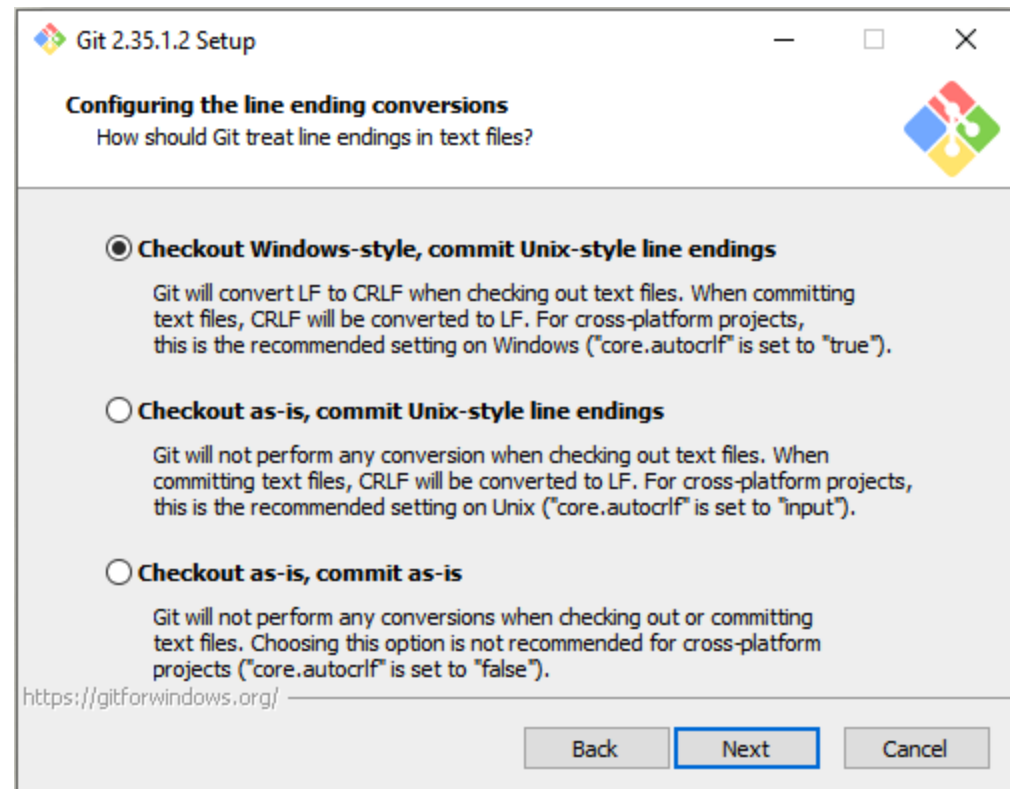
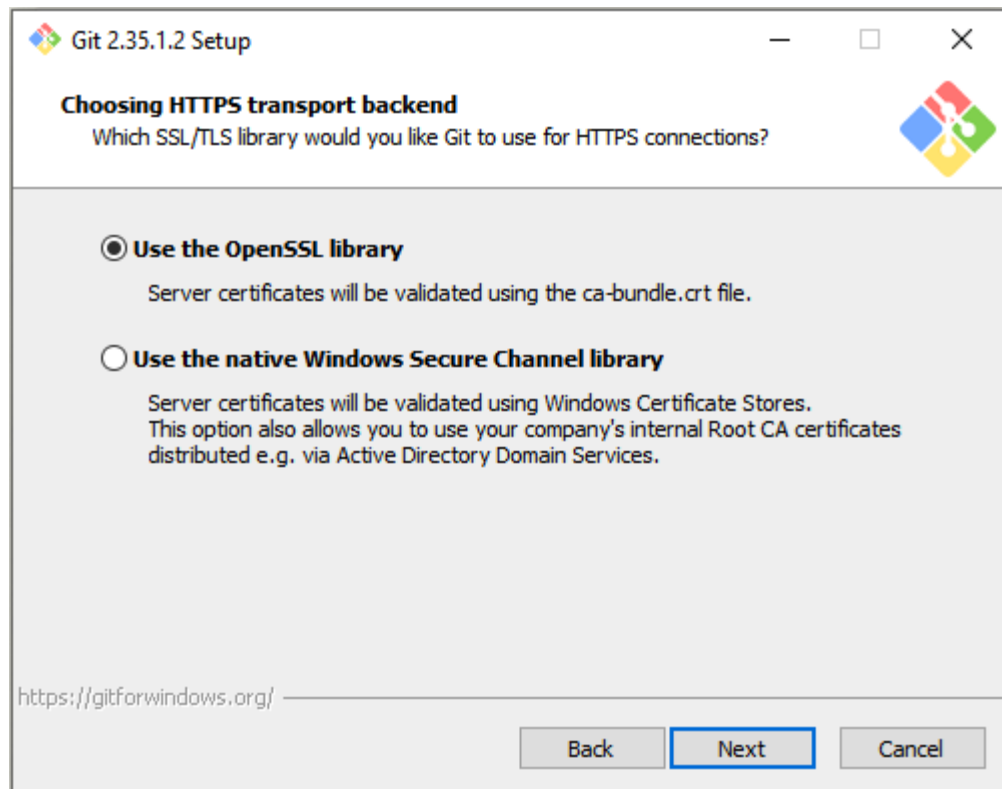


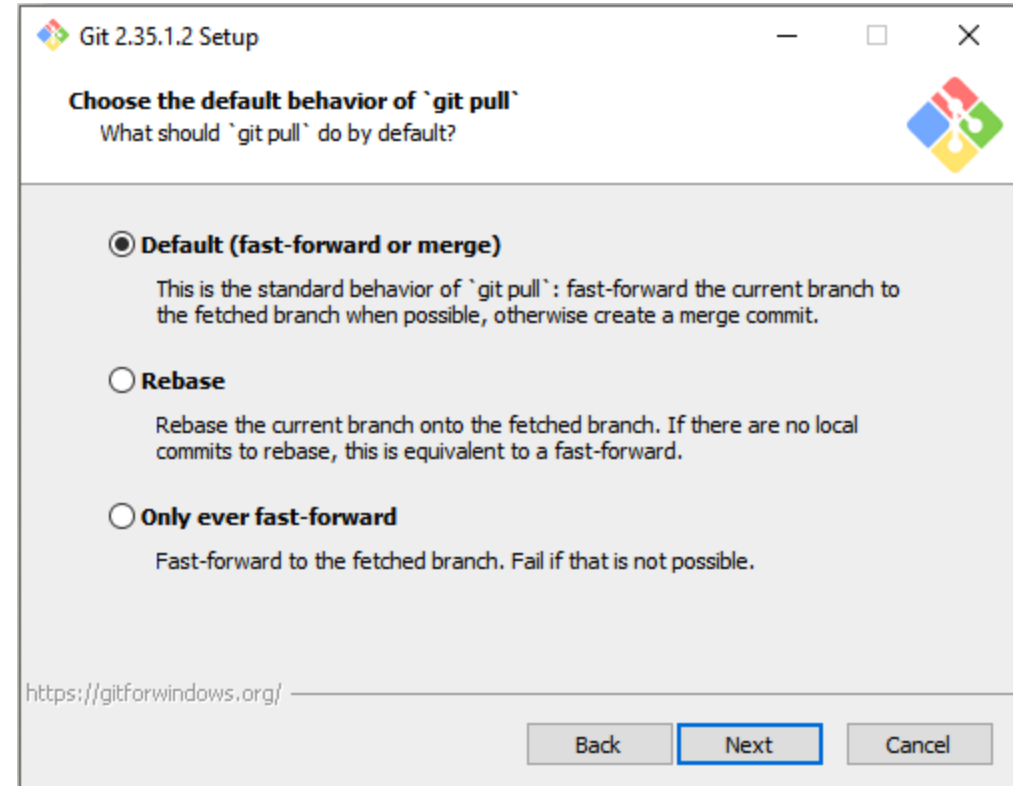
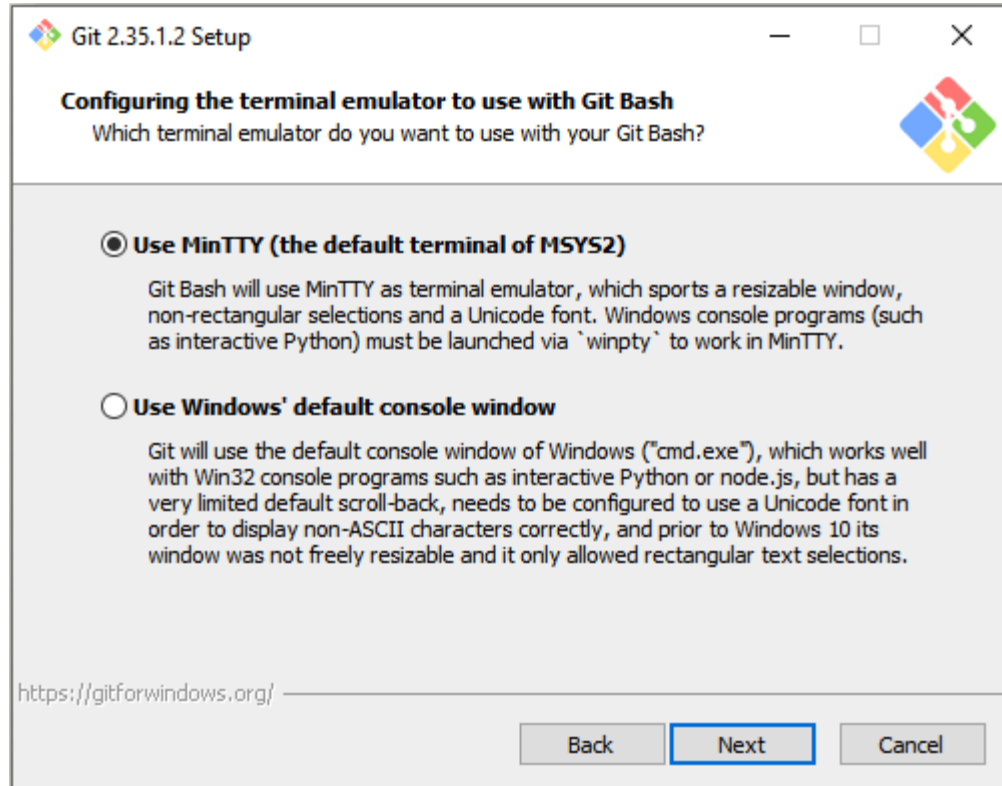


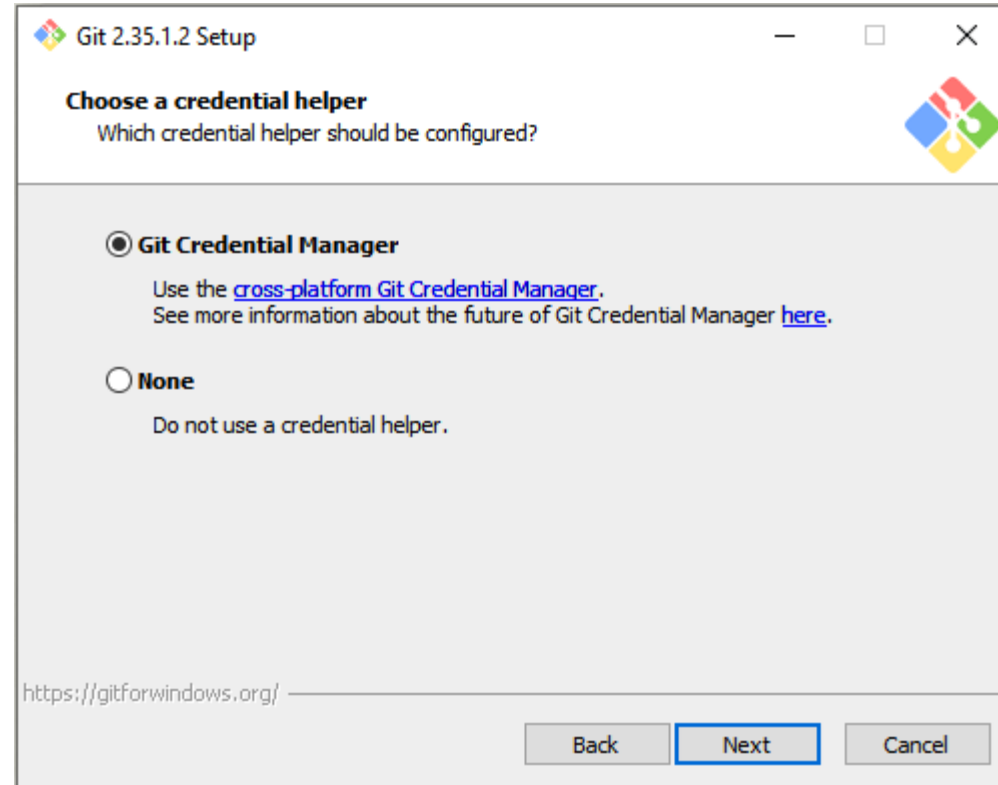
Seleccionar la opción para establecer “**main**” como rama principal cuando se crea el repositorio principal.



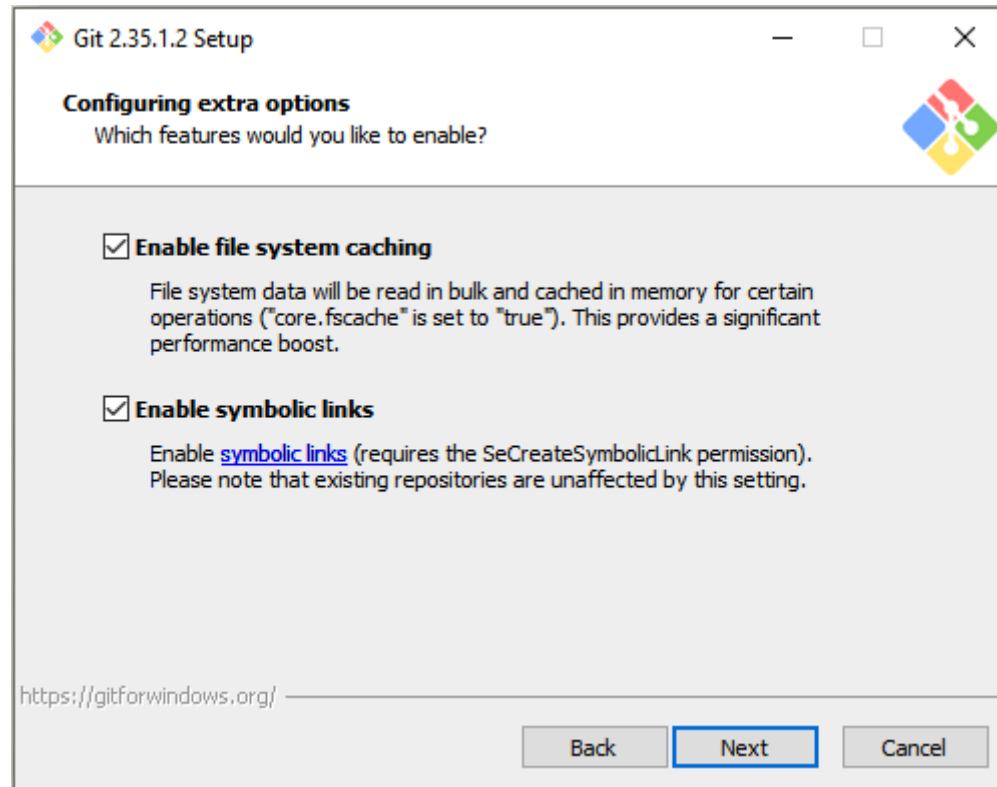


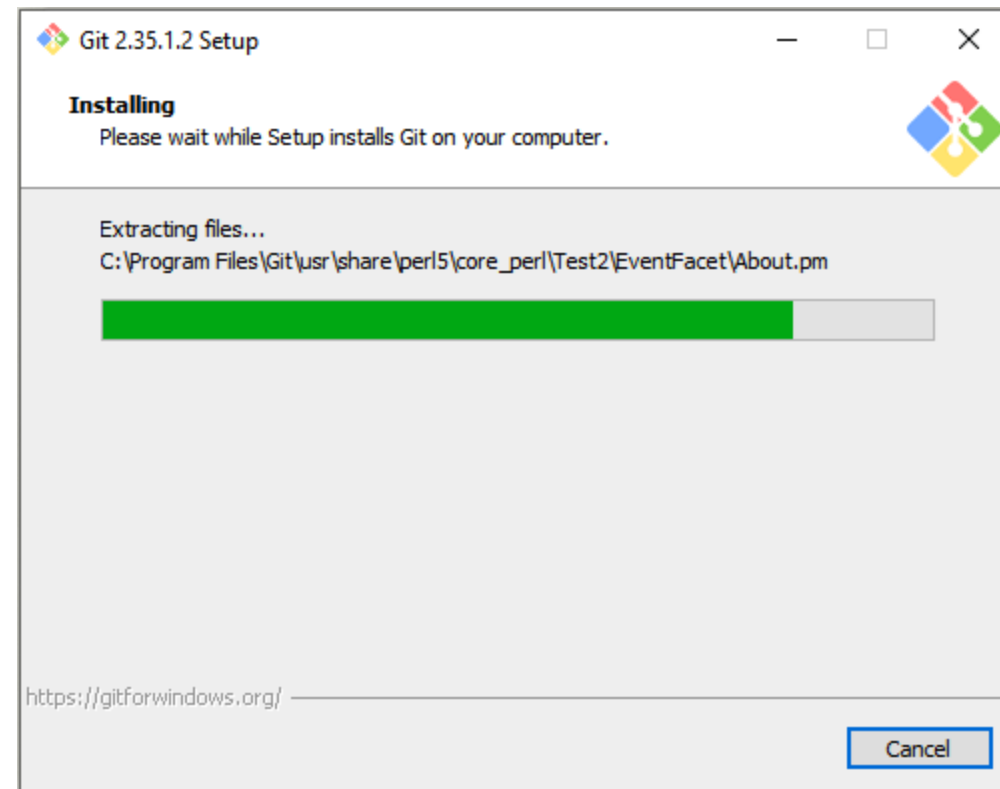
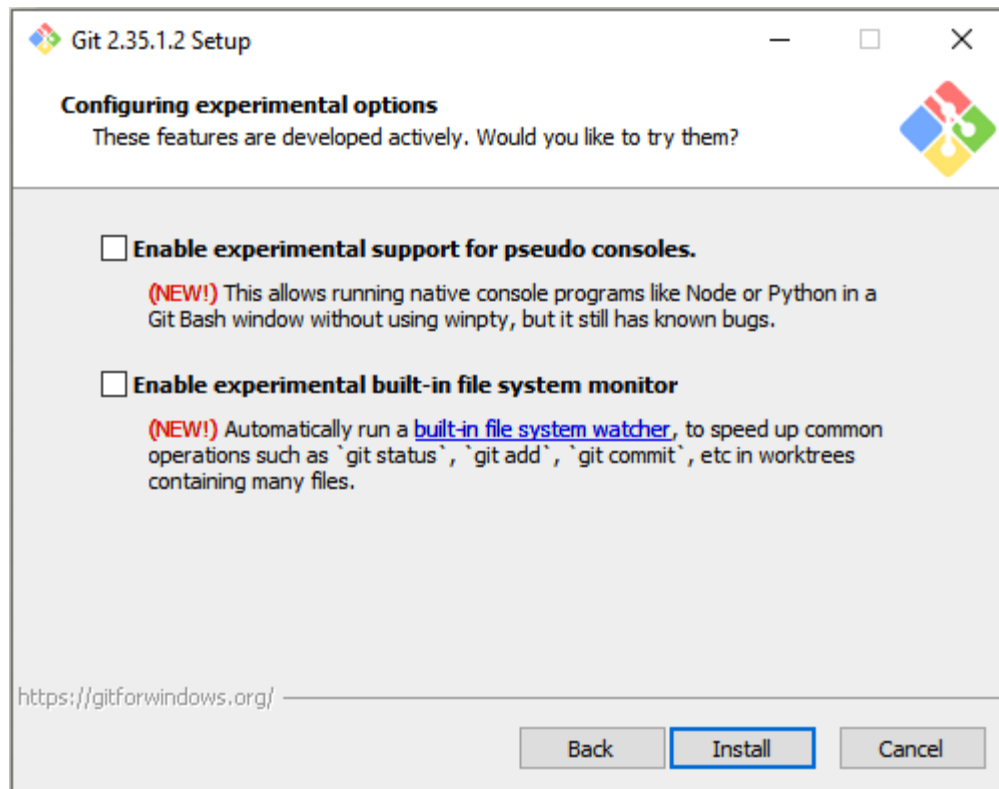




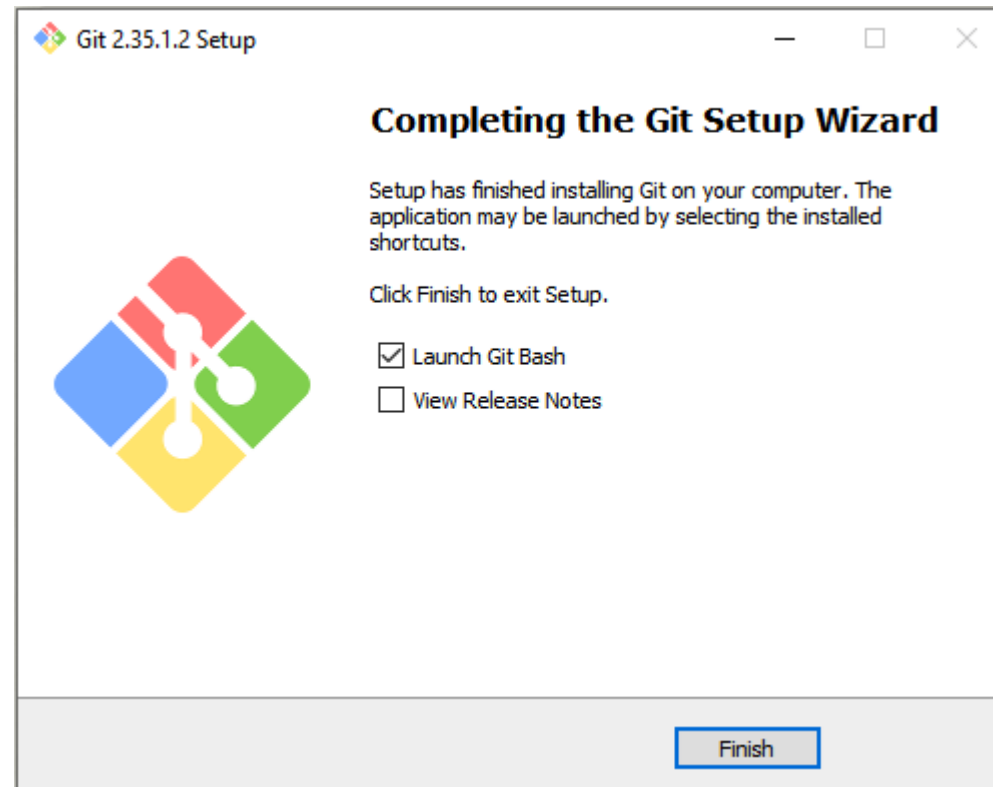


Activar "Enable symbolic links", requerido para activar permisos del sistema.

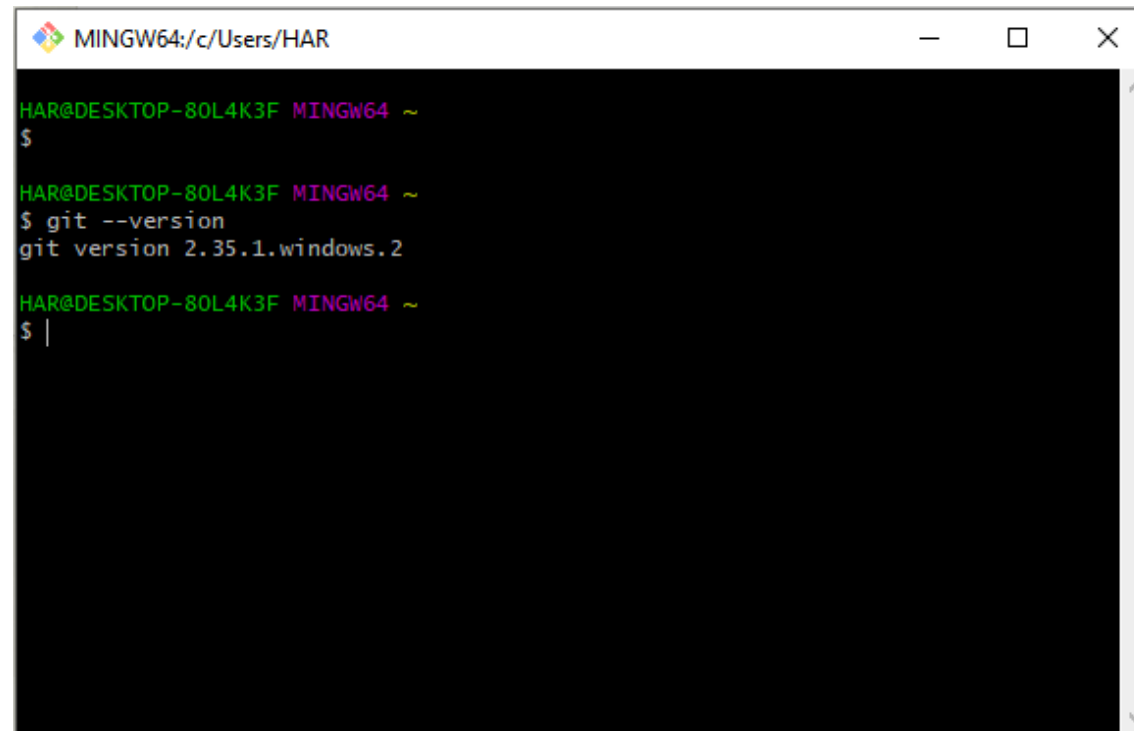




Seleccionar, ejecutar Git Bash y finalizar la instalación.



Git Bash es un emulador de la terminal de Linux en Windows, en el cual ejecutamos el comando **git --version** para verificar la instalación de Git, como se muestra en la siguiente imagen.



```
MINGW64:/c/Users/HAR

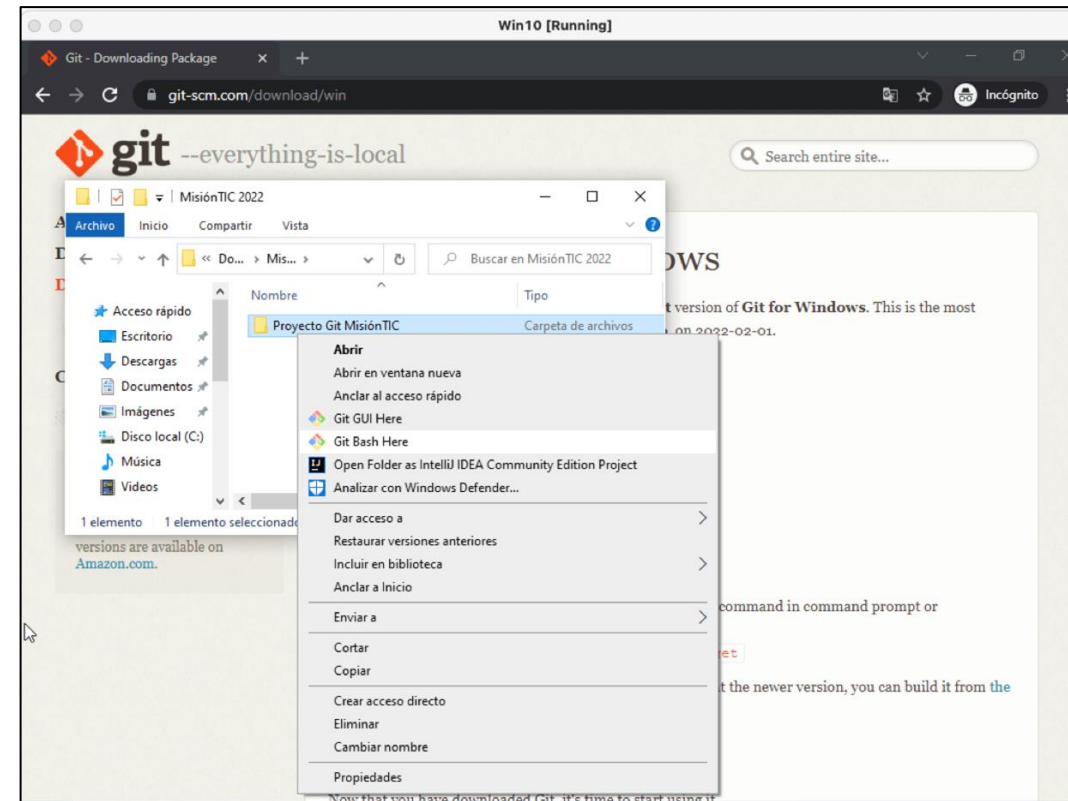
HAR@DESKTOP-80L4K3F MINGW64 ~
$

HAR@DESKTOP-80L4K3F MINGW64 ~
$ git --version
git version 2.35.1.windows.2

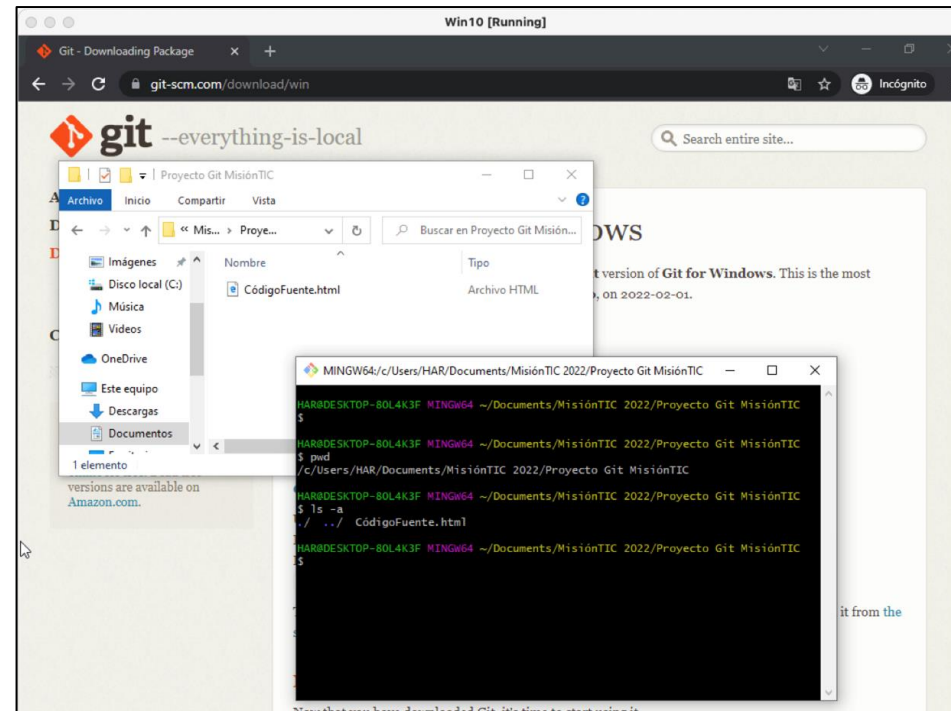
HAR@DESKTOP-80L4K3F MINGW64 ~
$ |
```

2.3.2. Crear Proyecto Git y Hacer Commit

Git se utiliza principalmente en líneas de comandos, el primer paso es ubicar el directorio donde se va a trabajar el proyecto; en tal caso, se pueden ejecutar comandos desde la terminal de Git Bash o en el explorador de archivos, ubicar el directorio del proyecto, dar clic derecho y seleccionar “**Git Bash Here**” para abrir la ruta del directorio seleccionado en la terminal de Git Bash, como se muestra en la siguiente imagen:

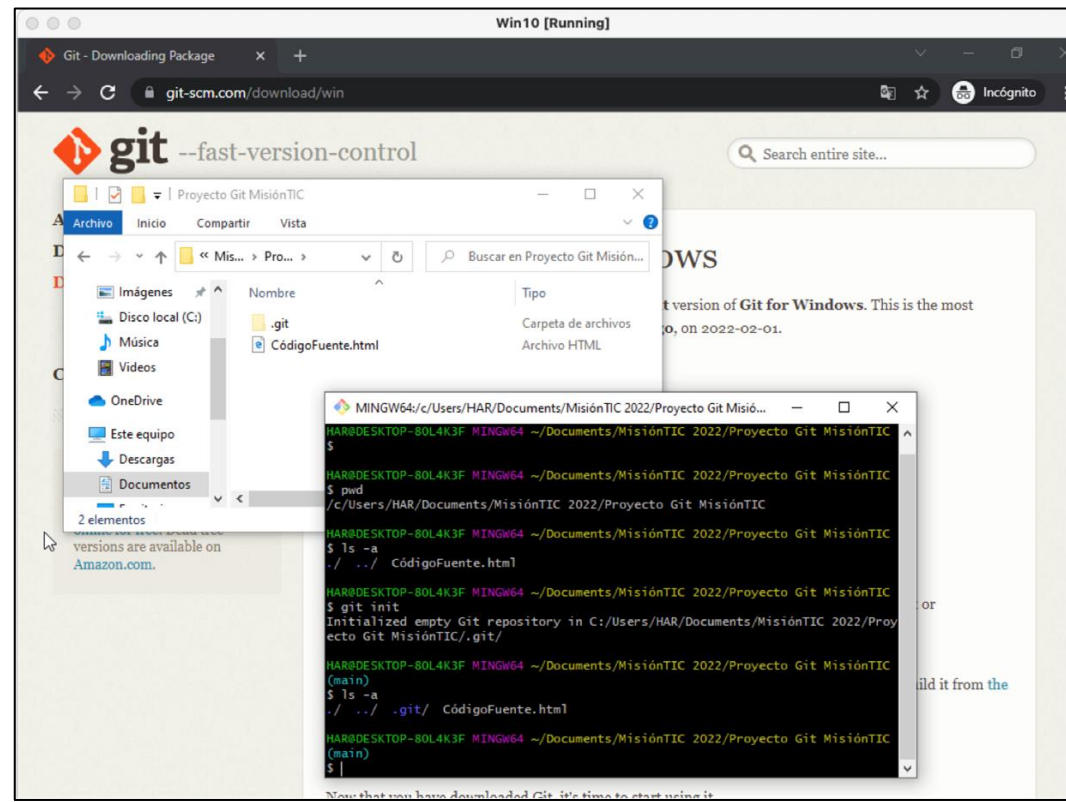


En la terminal de Git Bash, con el comando **pwd** puedo verificar la ruta del directorio seleccionado. Con el comando **ls -a**, puedo visualizar los archivos disponibles en el directorio seleccionado. La siguiente imagen muestra en la terminal de Git Bash el directorio "Proyecto Git Misión TIC" que contiene el archivo "CódigoFuente.html", el cual puedo visualizar tanto en el explorador de archivos como en la terminal de Git Bash.

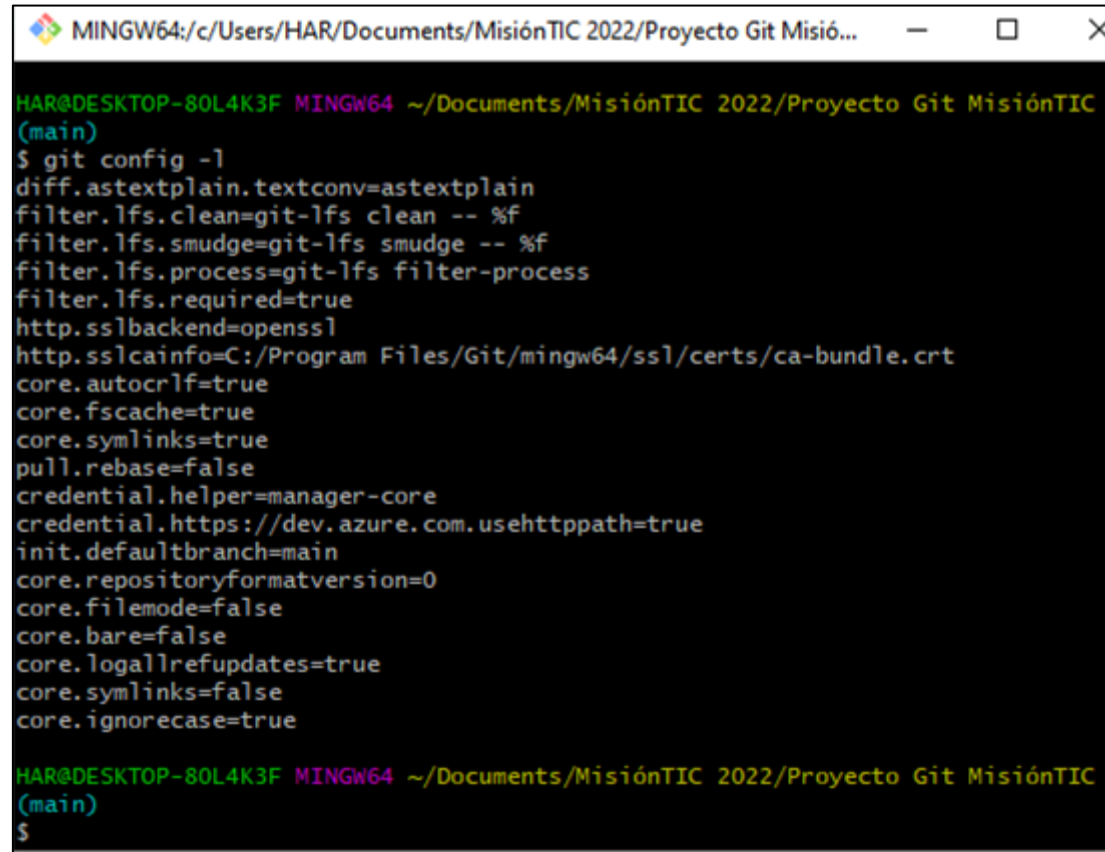


Nota: Los comandos **pwd** y **ls -a**, se pueden utilizar en la terminal de MacOS y Linux, sin embargo, *NO* funcionan en el cmd de Windows, debido a que son comandos propios del Kernel de Linux.

Una vez identificado el directorio del proyecto con su respectiva ubicación en la terminal de Git Bash, ejecuto el comando **git init** para crear el nuevo repositorio local de Git, creando por defecto la rama principal "**main**". La siguiente imagen permite visualizar tanto en el explorador de archivos como en la terminal de Git Bash, el directorio oculto **".git"** creado y disponible para iniciar el registro en el control de versiones de forma local.



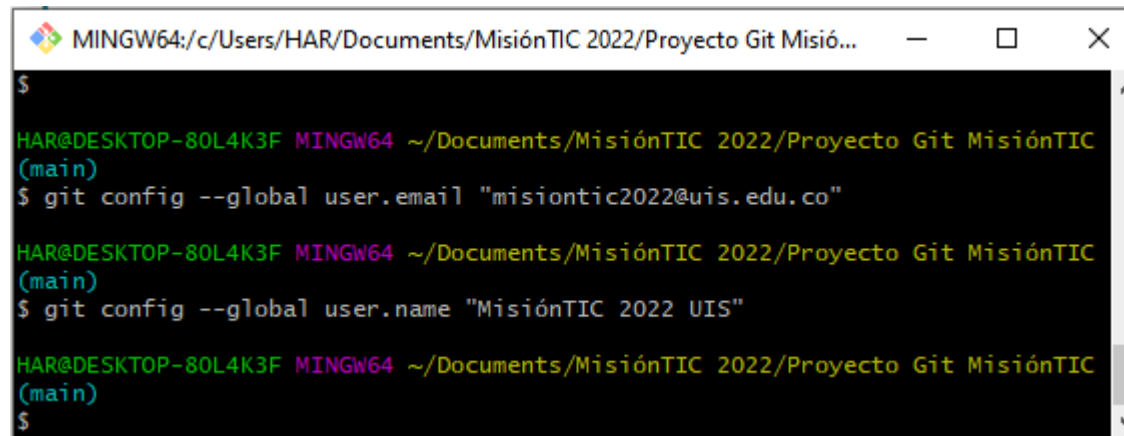
Con el comando **git config -l**, podemos visualizar la configuración del repositorio .git, en el cual se debe identificar el correo y nombre de usuario previo a realizar el primer commit.



```
MINGW64:/c/Users/HAR/Documents/MisiónTIC 2022/Proyecto Git Misió...
HAR@DESKTOP-80L4K3F MINGW64 ~/Documents/MisiónTIC 2022/Proyecto Git MisiónTIC
(main)
$ git config -l
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
http.sslbackend=openssl
http.sslcainfo=C:/Program Files/Git/mingw64/ssl/certs/ca-bundle.crt
core.autocrlf=true
core.fscache=true
core.symlinks=true
pull.rebase=false
credential.helper=manager-core
credential.https://dev.azure.com.usehttppath=true
init.defaultbranch=main
core.repositoryformatversion=0
core.filemode=false
core.bare=false
core.logallrefupdates=true
core.symlinks=false
core.ignorecase=true

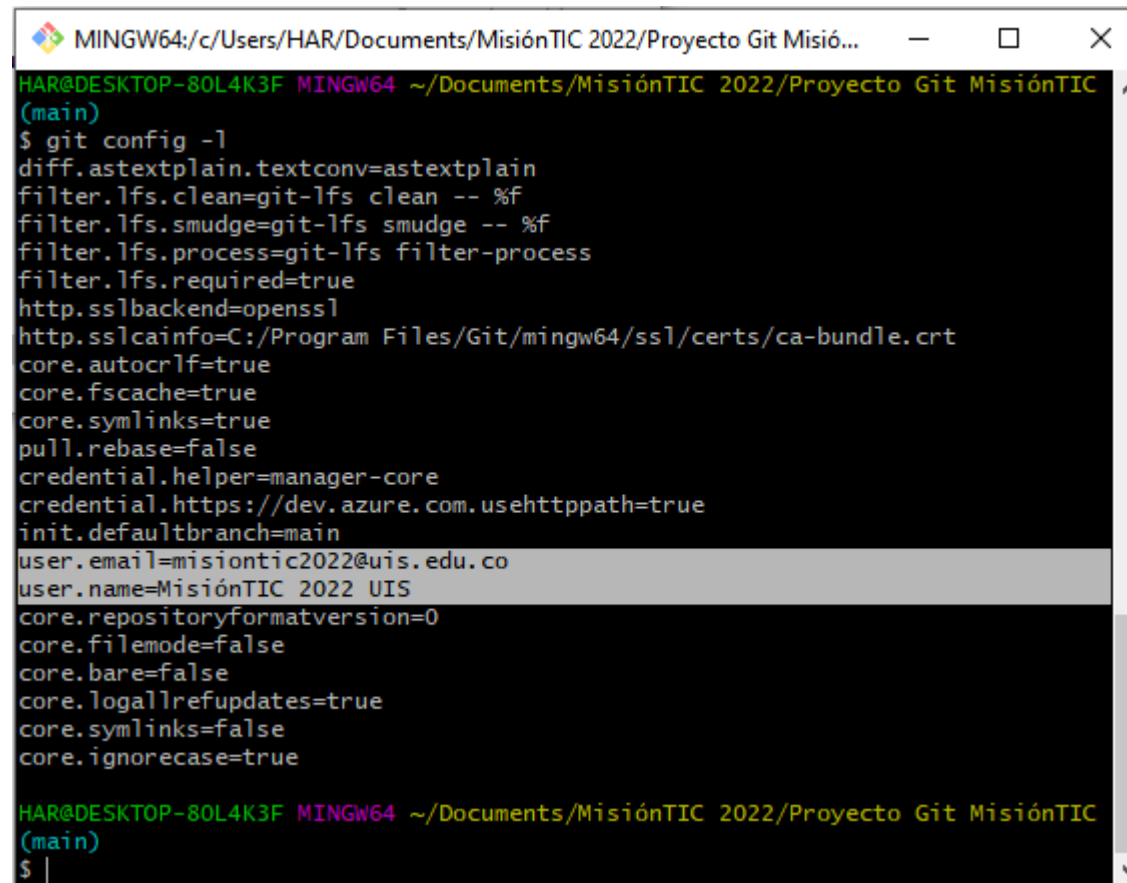
HAR@DESKTOP-80L4K3F MINGW64 ~/Documents/MisiónTIC 2022/Proyecto Git MisiónTIC
(main)
$
```

Con el comando **git config --global user.email "misiontic2022@uis.edu.co"**, establecemos el correo electrónico del usuario; y con el comando **git config --global user.name "MisiónTIC 2022"**, establecemos el nombre de usuario como se muestra en la siguiente imagen:

A screenshot of a terminal window titled "MINGW64:/c/Users/HAR/Documents/MisiónTIC 2022/Proyecto Git Misió...". The terminal shows the following commands and output:

```
$  
HAR@DESKTOP-80L4K3F MINGW64 ~/Documents/MisiónTIC 2022/Proyecto Git Misió...  
(main)  
$ git config --global user.email "misiontic2022@uis.edu.co"  
HAR@DESKTOP-80L4K3F MINGW64 ~/Documents/MisiónTIC 2022/Proyecto Git Misió...  
(main)  
$ git config --global user.name "MisiónTIC 2022 UIS"  
HAR@DESKTOP-80L4K3F MINGW64 ~/Documents/MisiónTIC 2022/Proyecto Git Misió...  
(main)  
$
```

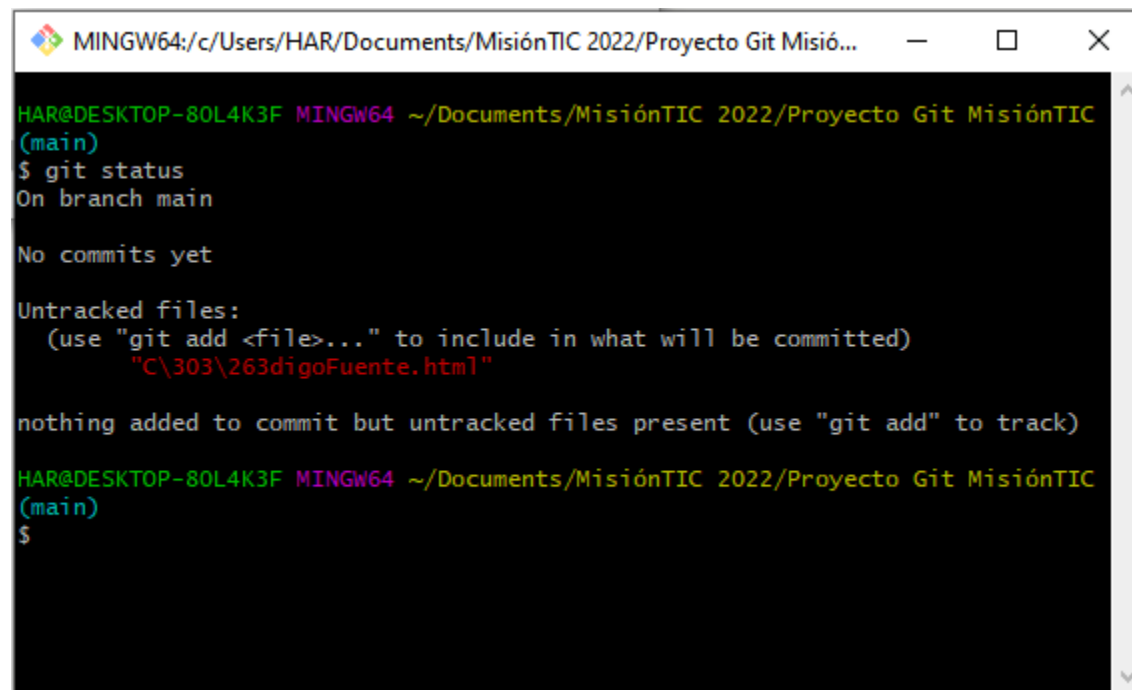
Si revisamos nuevamente la configuración del repositorio .git con el comando **git config -l**, se debe visualizar el correo electrónico y nombre de usuario asignado.



```
MINGW64: c:/Users/HAR/Documents/MisiónTIC 2022/Proyecto Git MisiónTIC
HAR@DESKTOP-80L4K3F MINGW64 ~/Documents/MisiónTIC 2022/Proyecto Git MisiónTIC
(main)
$ git config -l
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
http.sslbackend=openssl
http.sslcainfo=C:/Program Files/Git/mingw64/ssl/certs/ca-bundle.crt
core.autocrlf=true
core.fscache=true
core.symlinks=true
pull.rebase=false
credential.helper=manager-core
credential.https://dev.azure.com.usehttppath=true
init.defaultbranch=main
user.email=misiontic2022@uis.edu.co
user.name=MisiónTIC 2022 UIS
core.repositoryformatversion=0
core.filemode=false
core.bare=false
core.logallrefupdates=true
core.symlinks=false
core.ignorecase=true

HAR@DESKTOP-80L4K3F MINGW64 ~/Documents/MisiónTIC 2022/Proyecto Git MisiónTIC
(main)
$ |
```

Con el comando **git status**, se obtiene información referente a la rama en la cual se está trabajando, último commit y directorios, archivos o cambios en archivos de texto plano que están pendientes por guardar para el seguimiento en el control de versiones. En la siguiente imagen y para este caso en particular, en color rojo se identifica el archivo html que está pendiente por guardar.



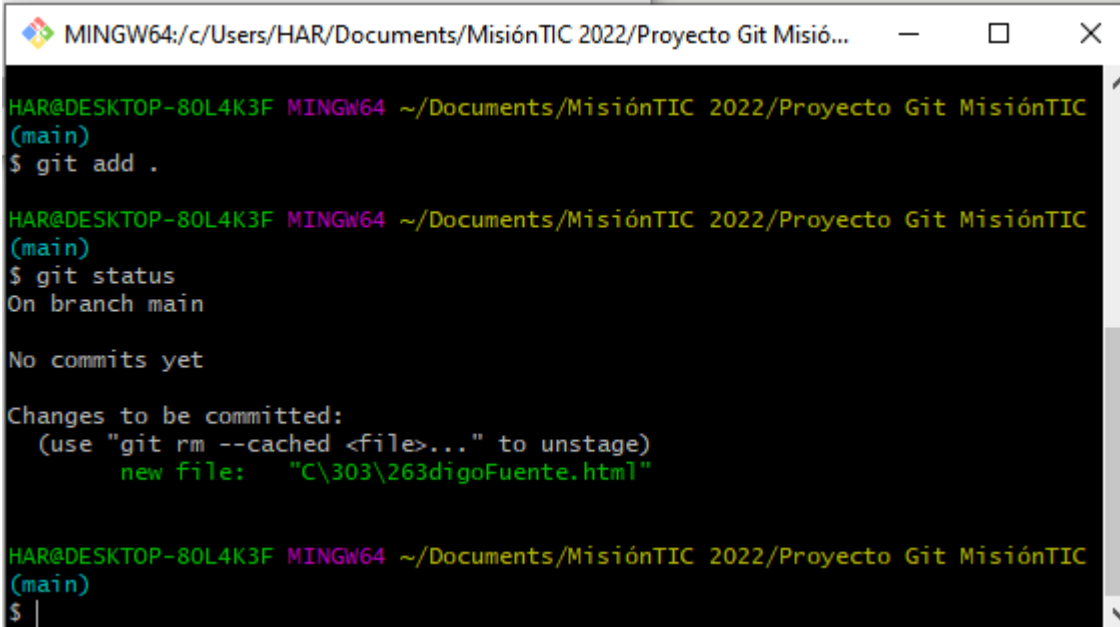
```
MINGW64:/c:/Users/HAR/Documents/MisiónTIC 2022/Proyecto Git Misió...
HAR@DESKTOP-80L4K3F MINGW64 ~/Documents/MisiónTIC 2022/Proyecto Git MisiónTIC
(main)
$ git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        "C\303\263digoFuente.html"

nothing added to commit but untracked files present (use "git add" to track)
HAR@DESKTOP-80L4K3F MINGW64 ~/Documents/MisiónTIC 2022/Proyecto Git MisiónTIC
(main)
$
```


Con el comando **git add .** (incluyendo el espacio punto), preparamos en el Staging todos los archivos nuevos o modificados para realizar el commit. Si ejecutamos nuevamente el comando **git status**, se observa que el archivo html se muestra en color verde, es decir, está listo para hacer el commit como se muestra en la siguiente figura:



```
MINGW64:/c:/Users/HAR/Documents/MisiónTIC 2022/Proyecto Git Misió...
HAR@DESKTOP-80L4K3F MINGW64 ~/Documents/MisiónTIC 2022/Proyecto Git MisiónTIC
(main)
$ git add .

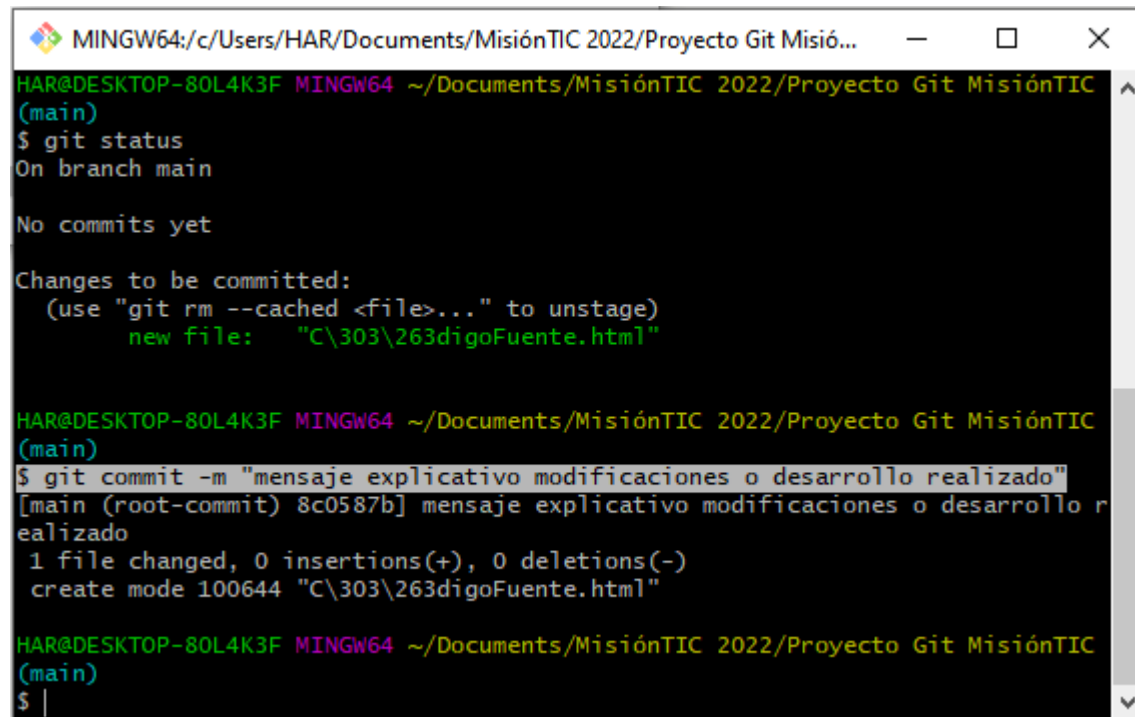
HAR@DESKTOP-80L4K3F MINGW64 ~/Documents/MisiónTIC 2022/Proyecto Git MisiónTIC
(main)
$ git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   "C:\303\263digoFuente.html"

HAR@DESKTOP-80L4K3F MINGW64 ~/Documents/MisiónTIC 2022/Proyecto Git MisiónTIC
(main)
$ |
```

Para guardar los cambios realizados o los archivos agregados, ejecutamos la siguiente instrucción **git commit -m "mensaje explicativo modificaciones o desarrollo realizado"**, donde el contenido entre comillas establece un breve mensaje tipo resumen, describiendo el trabajo guardado.



```
MINGW64:/c/Users/HAR/Documents/MisiónTIC 2022/Proyecto Git MisiónTIC
HAR@DESKTOP-80L4K3F MINGW64 ~/Documents/MisiónTIC 2022/Proyecto Git MisiónTIC
(main)
$ git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   "C:\303\263digoFuente.html"

HAR@DESKTOP-80L4K3F MINGW64 ~/Documents/MisiónTIC 2022/Proyecto Git MisiónTIC
(main)
$ git commit -m "mensaje explicativo modificaciones o desarrollo realizado"
[main (root-commit) 8c0587b] mensaje explicativo modificaciones o desarrollo r
ealizado
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 "C:\303\263digoFuente.html"

HAR@DESKTOP-80L4K3F MINGW64 ~/Documents/MisiónTIC 2022/Proyecto Git MisiónTIC
(main)
$ |
```

Nota 1: Posterior al primer commit, se establece un ciclo repetitivo para guardar cualquier modificación en el repositorio principal del proyecto, el cual podemos establecer con la siguiente serie de comandos:

- 1) **git status** => para identificar cualquier cambio en el repositorio del proyecto, o si hay archivos en el staging pendientes por guardar.
- 2) **git add .** => Prepara todos los archivos en el staging, para confirmar el guardado de los cambios establecidos.
- 3) **git commit -m "mensaje"** => Confirma el guardado de los cambios establecidos, incluyendo un mensaje que permite identificar el punto de control de cambios establecido.

Nota 2: Recordar que la explicación anterior, hace referencia a establecer el Control de Cambios en nuestra máquina local. Para compartir el repositorio con los demás miembros del equipo del proyecto, es necesario utilizar un repositorio remoto como GitHub o GitLab, entre los más populares.

2.3.3. Filtros y Archivo Gitignore

Un archivo “.gitignore” muestra de manera específica, archivos SIN seguimiento intencional que Git debe ignorar, es decir, archivos que son excluidos en el Control de Versiones. Cada línea en el archivo “.gitignore” establece un patrón, el cual establece omitir en el control de versiones rutas o archivos específicos.

El contenido del archivo “.gitignore”, puede estar precedido primordialmente de tres símbolos específicos que establecen un patrón coincidente:

- 1) * (asterisco) => Se utiliza como coincidencia comodín, por ejemplo: todos los archivos de texto, entonces el patrón a seguir se establece como ***.txt**
- 2) / (slash) => Se usa para ignorar las rutas relativas al archivo “.gitignore”.
- 3) # (numeral) => Es utilizado para agregar comentarios.

Ejemplo Contenido archivo .gitignore:

```
# Ignora archivos del sistema Mac
```

```
.DS_store
```

```
# Ignora la carpeta node_modules
```

```
node_modules
```

```
# Ignora todos los archivos de texto
```

```
*.txt
```

```
# Ignorar contenido específico de los diferentes IDE
```

```
### IntelliJ IDEA ###
```

```
.idea
```

```
*.iws
```

```
*.iml
```

```
*.ipr
```

```
### NetBeans ###
```

```
/nbproject/private/
```

```
/nbbuild/
```

```
/dist/
```

```
/nbdist/
```

```
/.nb-gradle/
```

```
build/
```

```
!**/src/main/**/build/
```

```
!**/src/test/**/build/
```

```
### VS Code ###
```

```
.vscode/
```

Nota: Para mayor información de los archivos “.gitignore” puede visitar el siguiente link <https://git-scm.com/docs/gitignore>

**Continúa con la segunda parte
de la guía de estudio del ciclo 2**

