UNIVERSIDAD GERARDO BARRIOS.

Facultad: Ciencia y Tecnología.

Carrera: Ingeniería en Sistemas y Redes.

Asignatura: Programación Computacional II.

Docente: Inga. Gisela Espinoza.

Actividad: Investigación Bibliográfica | GIT y GitHub.

Estudiantes:

- Josué Amílcar López Benítez SMIS935820
- Andy Rodrigo Sorto Castillo SMIS927320

Fecha de entrega:

Domingo 14 / Marzo / 2021

INTRODUCCIÓN

Muchas veces necesitamos tener un mayor control sobre las versiones y los cambios que hacemos en nuestros proyectos, siendo esto algo muy esencial para cualquier empresa o desarrollador independiente que busca adaptarse a las nuevas metodologías de trabajo colaborativo o simplemente desea tener un registro fiable para reconocer el seguimiento de los cambios que realiza mientras esta programando un software individual o colectivamente… y es de lo anterior de donde surge la necesidad de buscar un programa/software que nos ayude con todo este gran proceso de la administración de nuestros recursos y además de ello nos proporcione un alivio al pesado trabajo de la clasificación por versiones, esto para que se nos permita tener un indexado de cada etapa implementada y de cada uno de los encargados que las han llevado a cabo.

Es acá donde entra GIT, que lo podemos categorizar dentro de la rama de los software de control de versiones (SCV) y cuyo propósito como ya su rama lo específica, es el de llevar registros de los cambios en los archivos de nuestra computadora a los que les hemos asignado el proceso de utilización de esta herramienta, sin embargo, viene con otra funcionalidad que nos proveería algunas ventajas, como la de tener una "línea de ensamblaje de software" que sería definido por múltiples usuarios entrando a un mismo repositorio donde seria alojado el proyecto "master", de esta forma, se agilizaría el poder compartir el trabajo entre los usuarios involucrados, trabajándolo de forma "remota" en la plataforma de GitHub que fácilmente se puede vincular a la interfaz de GIT que ha sido instalada en nuestro ordenador, asegurando con ello un trabajo colaborativo, efectivo, eficiente y un poco más organizado, anexándole también, el hecho de conservar las "memorias" de los cambios agregados en cada una de las versiones implementadas.

También debemos hablar acerca de su funcionamiento base, donde podemos entender que tendríamos un **directorio local** que puede ser constituido por una carpeta en nuestro pc en el cual iniciaríamos el proceso de GIT (navegando hasta la carpeta y utilizando 'git init'), donde posteriormente GIT nos ira registrando los cambios a los documentos que nosotros hemos atribuido el uso de GIT(utilizando 'git add .' para agregarlo a un repositorio local y poder ver el seguimiento de los recursos que hemos agregado), de esta forma tendríamos un "pase de viaje en el tiempo", en donde podríamos retroceder a versiones anteriores y con ello obtener una restauración del código almacenado en la versión que solicitamos…

LISTADO DE COMANDOS GIT

Recapitulando un poco, debemos definir que es GIT y de una forma simple lo presentaremos como un sistema de control de versiones o SCV que es de código abierto y ha sido el más utilizado. Todo gracias a que se nos permite rastrear los cambios realizados en los archivos en los que hemos trabajado, además, es una plataforma que permite que las empresas y los programadores puedan interactuar entre ellos para colaborar en el desarrollo de software y aplicaciones… De esta forma un proyecto GIT consta de tres secciones principales los cuales son:

- **El directorio de trabajo.**
- **El área de preparación.**
- **El directorio GIT**.

Así mismo, existen una serie de comandos para poder trabajar en los distintos procesos que trae consigo el utilizar GIT, por lo tanto, a continuación, hacemos una lista de los comandos más básicos y esenciales que debemos de conocer para poder utilizar esta herramienta de trabajo, manejándolo de manera correcta y de forma que podamos asegurar nuestros repositorios y proyectos:

- **git init:** Creará un nuevo repositorio local GIT en el directorio actual. Además como alternativa, podemos crear un repositorio dentro de un nuevo directorio especificando el nombre del proyecto y la forma de implementar esta última opción mencionada seria la siguiente:

```
git init [nombre del proyecto]
```

- **git clone:** Normalmente se utiliza para copiar un repositorio y si el repositorio está en un servidor remoto se utiliza de la siguiente forma:

```
git clone nombredeusuario@host:/path/to/repository
```

  En cambio, si pensamos copiar un repositorio local utilizamos la siguiente línea de código para ejecutarlo

```
git clone /path/to/repository
```

- **git add:** Se usa para agregar archivos al área de preparación (repositorio local). Por ejemplo, el siguiente comando de Git básico indexará un archivo de texto:

```
git add <nombre del archivo.txt>
```

- **git commit:** Capturara una instantánea de los cambios preparados en ese momento del proyecto, entonces, las instantáneas confirmadas pudieran considerarse como versiones "seguras" de un proyecto.

```
git commit –m “El mensaje que acompaña al commit va aquí”
```

- **git config:** Este comando puede ser usado para establecer una configuración específica de usuario, como el email, nombre de usuario y tipo de formato, etc. Por ejemplo, el siguiente comando se usa para establecer un email:

```
git config --global user.email tuemail@ejemplo.com
```

Debemos entender que la opción --global le dice a GIT que vas a usar ese correo electrónico para todos los repositorios locales. En cambio, si necesitamos utilizar diferentes correos electrónicos para diferentes repositorios, usamos el siguiente comando:

```
git config --local user. email tuemail@ejemplo.com
```

- **git status:** Utilizado para mostrar una lista de los archivos que se han cambiado junto con los archivos que están por ser preparados o confirmados, también podemos iniciar este comando de la siguiente forma:

```
git status -s
```

- **git push:** Es utilizado para enviar confirmaciones locales a la rama maestra del repositorio remoto al cual deseamos introducir los datos de nuestro proyecto y su estructura básica seria la siguiente:

```
git push origin < Reemplaza  con la rama en la que quieres enviar los cambios>
```

- **git checkout:** Con este comando podemos crear “ramas” , además de que nos puede ayudar a navegar entre ellas. Por ejemplo, el siguiente comando crea una nueva rama y automáticamente se cambia a ella:

```
command git checkout -b <insertar nombre de rama a crear>
```

Sin embargo, podemos utilizarlo simplemente para cambiar de una rama a otra, y se utilizaría de la siguiente manera:

```
git checkout <rama a la cual desea cambiar>
```

- **git remote:** Este comando nos permite ver todos los repositorios remotos. A continuación se deja algunos ejemplos para utilizar este comando:

  Se ejecuta comando

```
git remote -v
```

  Para conectar el repositorio local a un servidor remoto, usamos este comando:

```
git remote add origin <insertar URL de repositorio remoto>
```

  Si lo que deseamos es borrar una conexión a un repositorio remoto especificado, utilizamos:

```
git remote <nombre-del-repositorio>
```

- **git Branch:** Se utiliza para listar, crear o borrar ramas. Por ejemplo, si queremos listar todas las ramas presentes en el repositorio, el comando debería verse así:

```
git branch
```

  Si deseamos borrar una rama, utilizaríamos la siguiente línea de código:

```
git branch -d <branch-name>
```

- **git pull:** Este comando sire para fusionar todos los cambios que se han hecho en el repositorio local con el directorio de trabajo local y lo ejecutamos de la siguiente manera:

```
git pull
```

- **git merge:** Nos permitirá fusionar una rama con otra rama activa:

```
git merge <nombre de la rama activa>
```

- **git diff:** Se Puede utilizar para hacer una lista de “conflictos”, misma donde podremos ver los conflictos con respecto al archivo base, y lo utilizamos de la siguiente manera:

```
git diff --base <nombre de archivo>
```

Si lo que deseamos es ver los conflictos que existen entre las ramas antes de fusionarlas, utilizamos la siguiente línea de código:

```
git diff <source-branch> <target-branch>
```

Además, si deseamos que se nos presente una lista general de todos los conflictos que puedan llegar a existir, usamos lo siguiente:

```
git diff
```

- **git tag:** Este comando marca commits específicos, por lo tanto es muy utilizado por desarrolladores, donde lo utilizan para marcar puntos de lanzamiento como v1.0 y v2.0.

```
git tag 1.1.0 <insert-id del commit>
```

- **git log:** Frecuentemente se usa para ver el historial del repositorio listando ciertos detalles de la confirmación y al ejecutar el comando se obtiene una salida como ésta:

```
commit 15f4b6c44b3c8344caasdac9e4be13246e21sadw
Author: Alex Hunter <alexh@gmail.com>
Date:   Mon Oct 1 12:56:29 2016 -0600
```

- **git reset:** Se puede utilizar para reiniciar el “índex” y el directorio de trabajo al último estado de confirmación; Su estructura básica seria la siguiente:

```
git reset - -hard HEAD
```

- **git rm:** Es utilizado para remover archivos del índex y del directorio de trabajo y se ejecuta de la siguiente manera:

```
git rm nombre del archivo.txt
```

- **git stash:** Se usa para guardar momentáneamente los cambios que no están listos para ser confirmados y de esta manera se puede volver al proyecto más tarde.

```
git stash
```

- **git show:** Es utilizad para mostrar la información sobre cualquier objeto git.

```
git show
```

- **git fetch:** Este comando permite al usuario buscar todos los objetos de un repositorio remoto que actualmente no se encuentran en el directorio de trabajo local y se puede usar de la siguiente manera:

```
git fetch origin
```

- **git ls-tree:** Nos permite ver un objeto del árbol junto con el nombre, modo de cada ítem y el valor blob de SHA-1. Si queremos ver el HEAD, lo utilizamos de la siguiente forma:

```
git ls-tree HEAD
```

- **git cat-file:** Se utiliza para ver la información de tipo y tamaño de un objeto del repositorio. Además usamos la opción -p junto con el valor SHA-1 del objeto para ver la información de un objeto específico, por ejemplo:

```
git cat-file –p d670460b4b4aece5915caf5c68d12f560a9fe3e4
```

- **git grep:** Permite que el usuario pueda buscar frases y palabras específicas en los árboles de confirmación, el directorio de trabajo y en el área de preparación (repositorio local). Para utilizarlo y buscar por www.hostinger.com en todos los archivos, se utiliza:

```
git grep “www.hostinger.com”
```

- **gitk:** este comando muestra la interfaz gráfica para un repositorio local y simplemente ejecuta de la siguiente manera:

```
gitk
```

- **git instaweb:** Nos permite explorar nuestro repositorio local en la interfaz GitWeb. Por ejemplo, lo podemos utilizar de esta forma:

```
git instaweb –http=webrick
```

- **git gc:** Este comando nos ayudara a limpiar archivos innecesarios y con ello optimizar el repositorio local.

```
git gc
```

- **git archive:** Permite que el usuario pueda crear archivos zip o rar que contengan los constituyentes de un solo árbol de repositorio. Por ejemplo:

```
git archive - -format=zip master
```

- **git prune:** Este comando sirve para eliminar los objetos que no tengan ningún apuntador entrante.

```
git prune
```

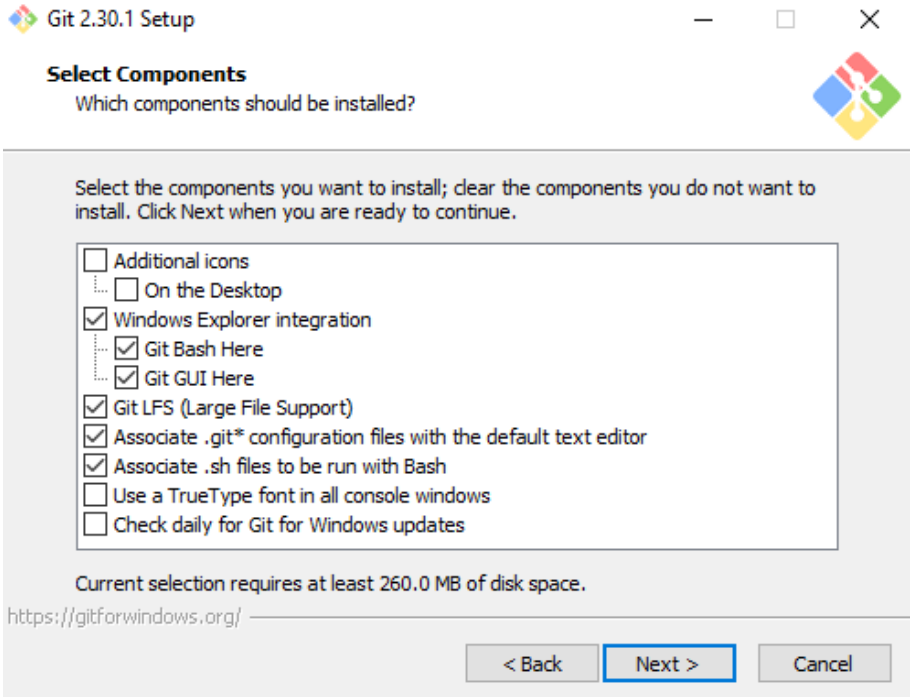
- **git fsck:** Se utiliza para realizar una comprobación de integridad del sistema de archivos git e identificar cualquier objeto corrupto.

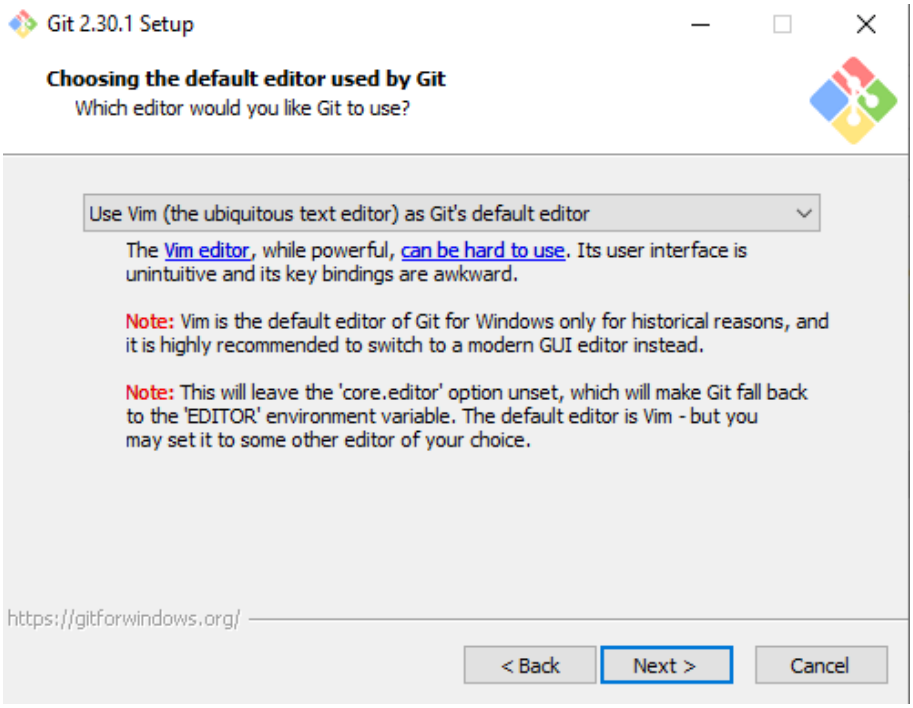
```
git fsck
```

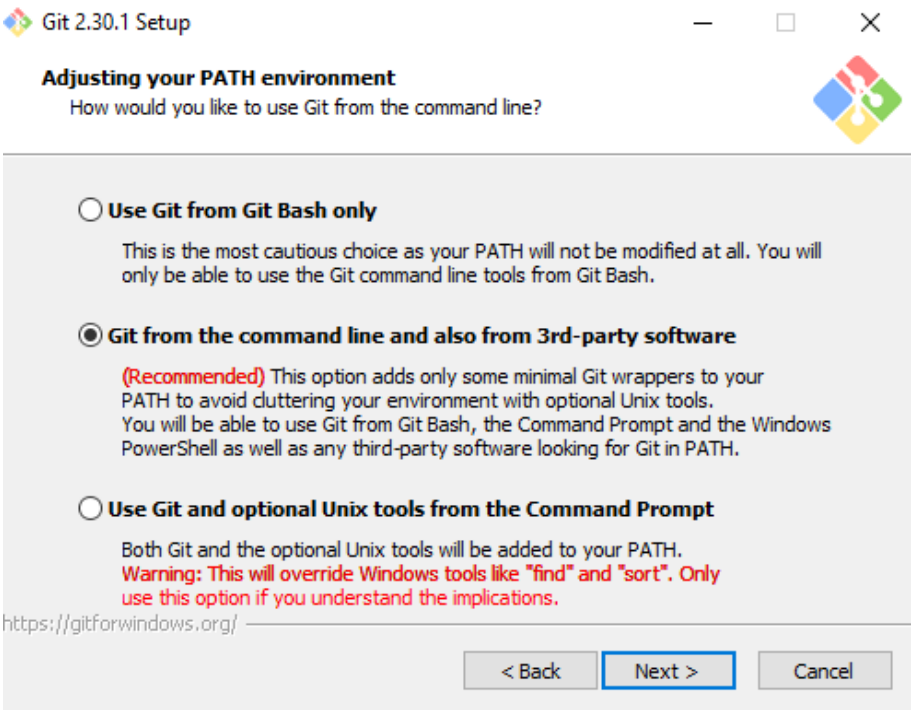
- **git rebase:** Comúnmente se utiliza para aplicar ciertos cambios de una rama en otra. Por ejemplo, lo podemos utilizar de la siguiente manera:

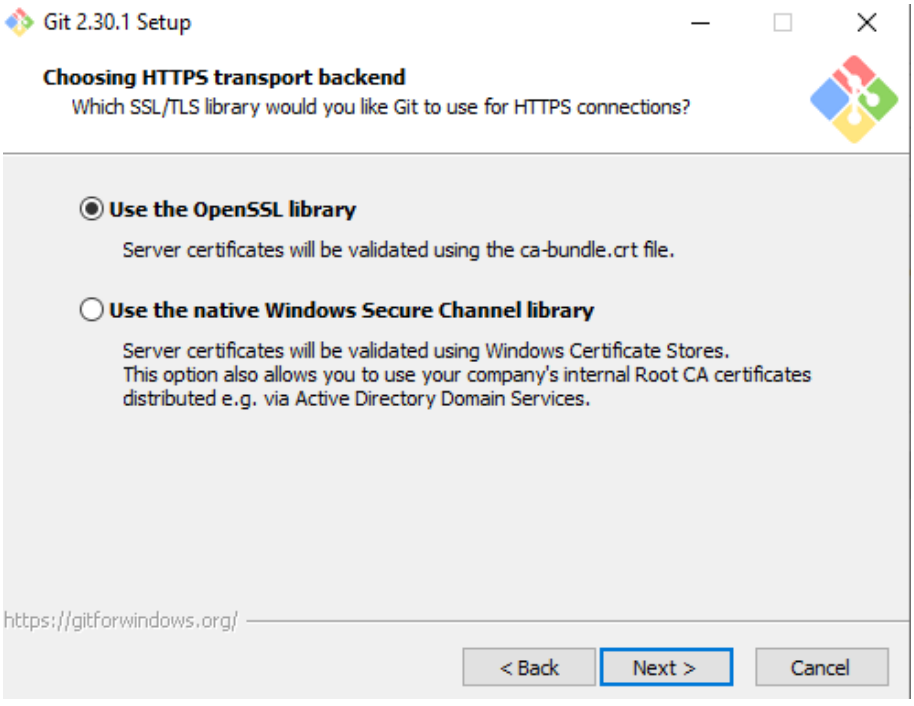
```
git rebase master
```

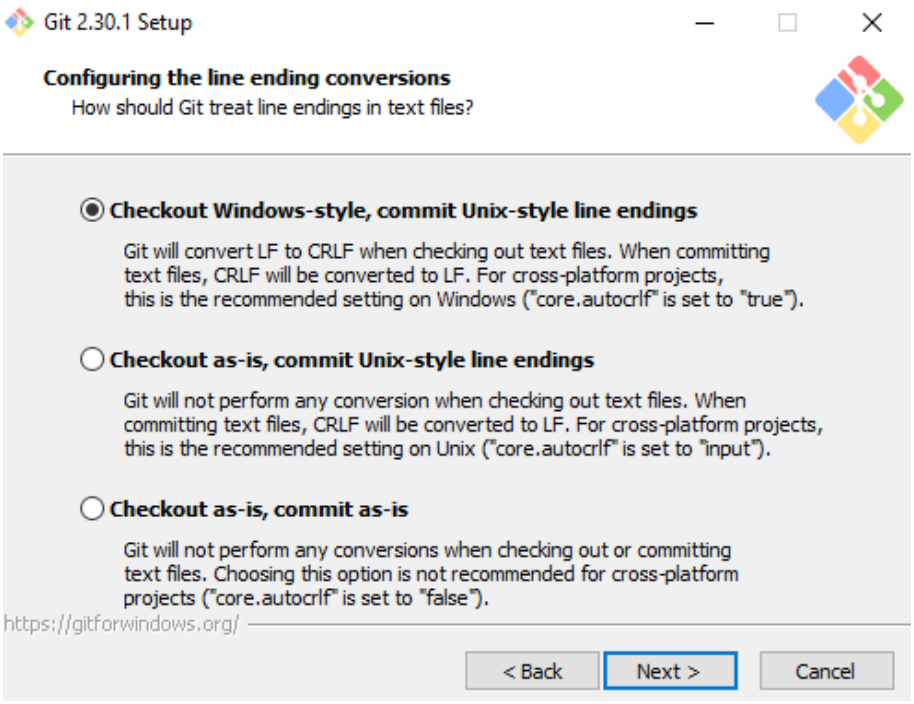
CONFIGURACIÓN GIT

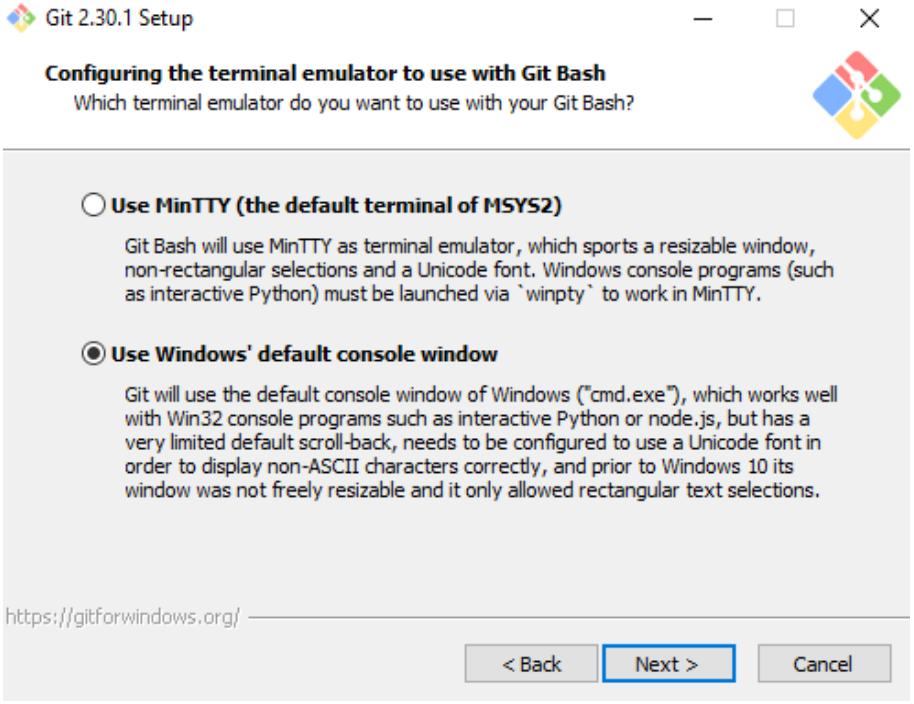


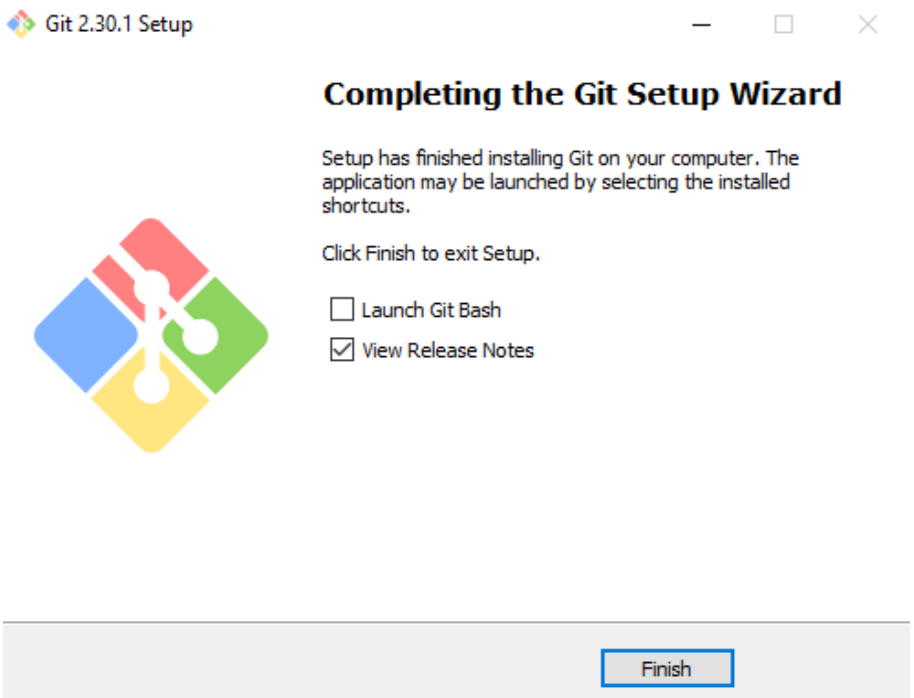


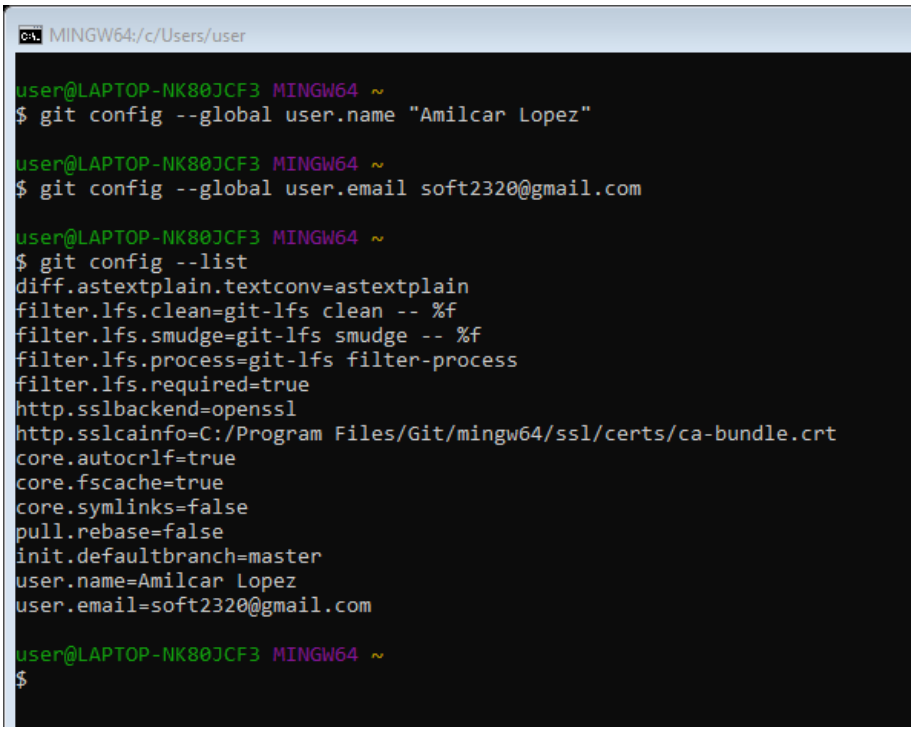

Git 2.30.1 Setup
Adjusting your PATH environment
How would you like to use Git from the command line?
Use Git from Git Bash only
This is the most cautious choice as your PATH will not be modified at all. You will only be able to use the Git command line tools from Git Bash.
Git from the command line and also from 3rd-party software
(Recommended) This option adds only some minimal Git wrappers to your PATH to avoid cluttering your environment with optional Unix tools.
You will be able to use Git from Git Bash, the Command Prompt and the Windows PowerShell as well as any third-party software looking for Git in PATH.
Use Git and optional Unix tools from the Command Prompt
Both Git and the optional Unix tools will be added to your PATH.
Warning: This will override Windows tools like "find" and "sort". Only use this option if you understand the implications.
https://gitforwindows.org/
< Back
Next >
Cancel


Git 2.30.1 Setup
Choosing HTTPS transport backend
Which SSL/TLS library would you like Git to use for HTTPS connections?
Use the OpenSSL library
Server certificates will be validated using the ca-bundle.crt file.
Use the native Windows Secure Channel library
Server certificates will be validated using Windows Certificate Stores.
This option also allows you to use your company's internal Root CA certificates distributed e.g. via Active Directory Domain Services.
https://gitforwindows.org/
< Back
Next >
Cancel

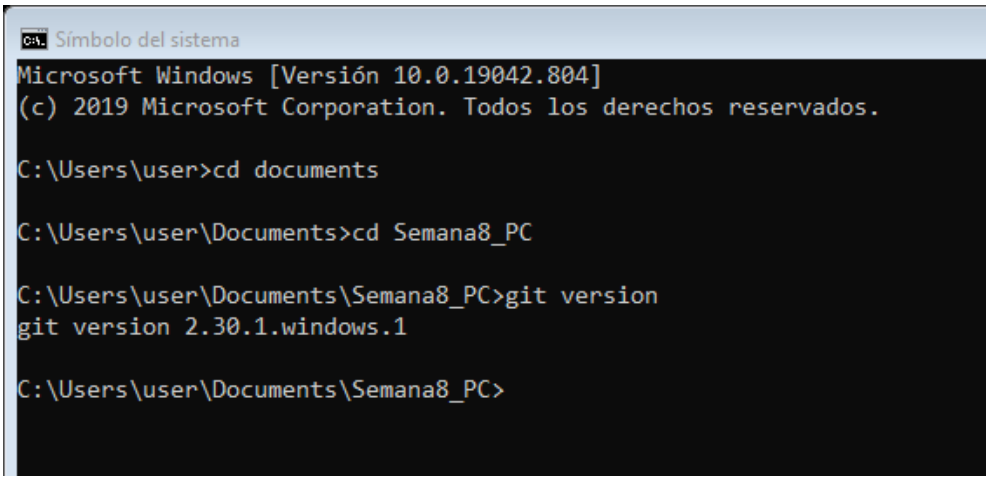

Git 2.30.1 Setup
Configuring the line ending conversions
How should Git treat line endings in text files?
Checkout Windows-style, commit Unix-style line endings
Git will convert LF to CRLF when checking out text files. When committing text files, CRLF will be converted to LF. For cross-platform projects, this is the recommended setting on Windows ("core.autocrlf" is set to "true").
Checkout as-is, commit Unix-style line endings
Git will not perform any conversion when checking out text files. When committing text files, CRLF will be converted to LF. For cross-platform projects, this is the recommended setting on Unix ("core.autocrlf" is set to "input").
Checkout as-is, commit as-is
Git will not perform any conversions when checking out or committing text files. Choosing this option is not recommended for cross-platform projects ("core.autocrlf" is set to "false").
https://gitforwindows.org/
< Back
Next >
Cancel

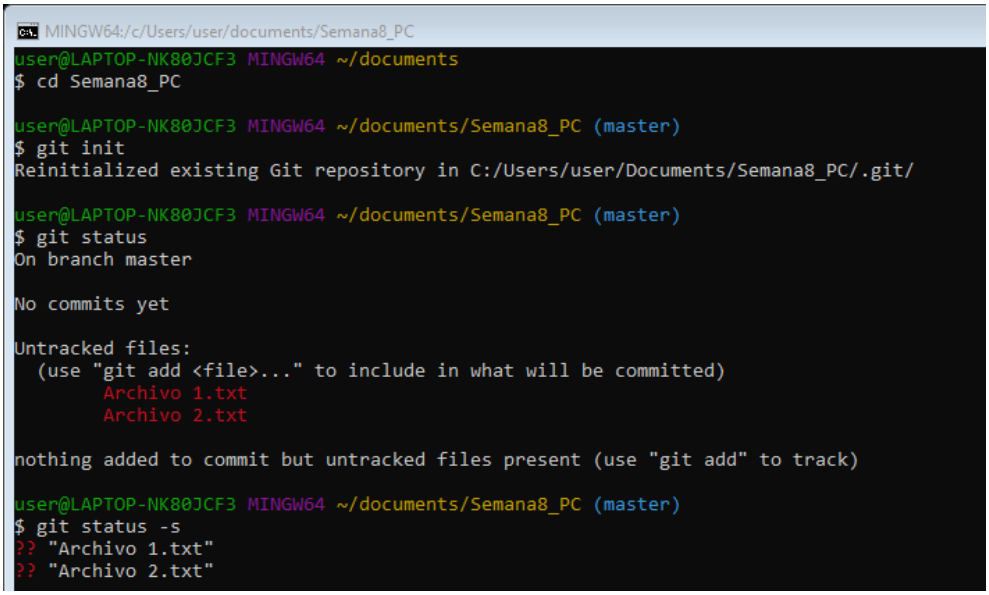

Git 2.30.1 Setup
Configuring the terminal emulator to use with Git Bash
Which terminal emulator do you want to use with your Git Bash?
Use MinTTY (the default terminal of MSYS2)
Git Bash will use MinTTY as terminal emulator, which sports a resizable window, non-rectangular selections and a Unicode font. Windows console programs (such as interactive Python) must be launched via `winpty` to work in MinTTY.
Use Windows' default console window
Git will use the default console window of Windows ("cmd.exe"), which works well with Win32 console programs such as interactive Python or node.js, but has a very limited default scroll-back, needs to be configured to use a Unicode font in order to display non-ASCII characters correctly, and prior to Windows 10 its window was not freely resizable and it only allowed rectangular text selections.
https://gitforwindows.org/
< Back
Next >
Cancel

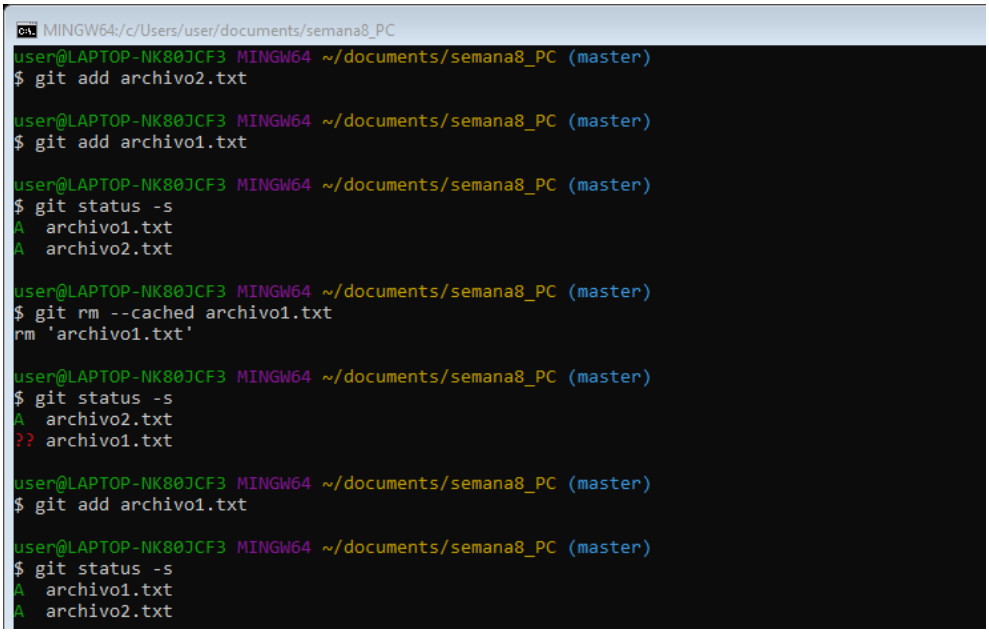

Git 2.30.1 Setup
Completing the Git Setup Wizard
Setup has finished installing Git on your computer. The application may be launched by selecting the installed shortcuts.
Click Finish to exit Setup.
Launch Git Bash
View Release Notes
Finish

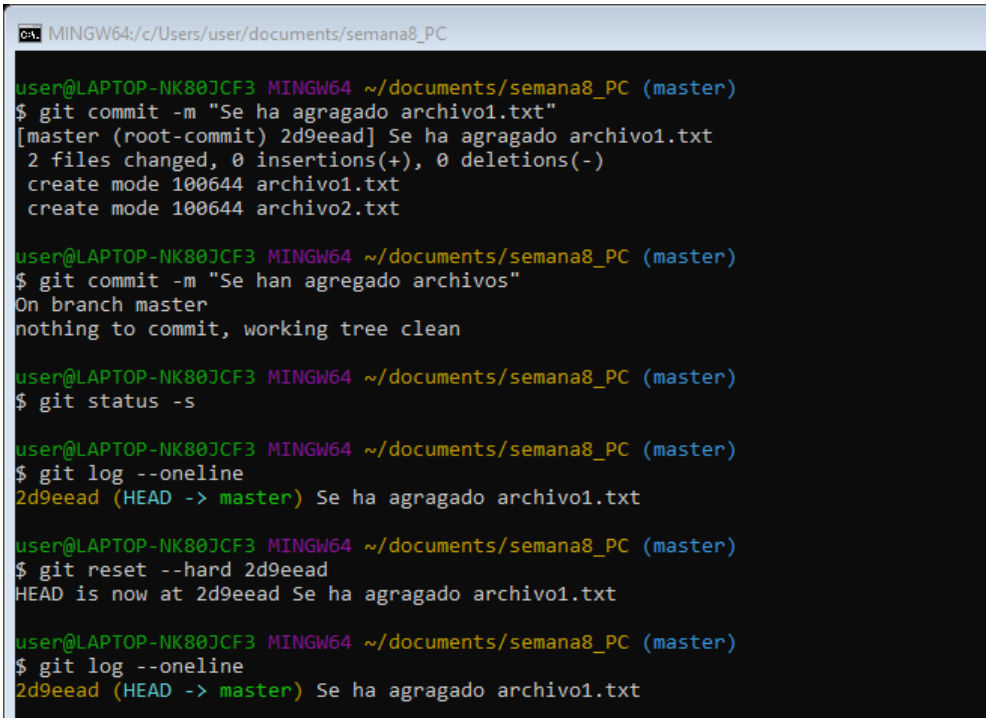

MINGW64:/c/Users/user
user@LAPTOP-NK80JCF3 MINGW64 ~
$ git config --global user.name "Amilcar Lopez"
user@LAPTOP-NK80JCF3 MINGW64 ~
$ git config --global user.email soft2320@gmail.com
user@LAPTOP-NK80JCF3 MINGW64 ~
$ git config --list
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %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=false
pull.rebase=false
init.defaultbranch=master
user.name=Amilcar Lopez
user.email=soft2320@gmail.com
user@LAPTOP-NK80JCF3 MINGW64 ~
$

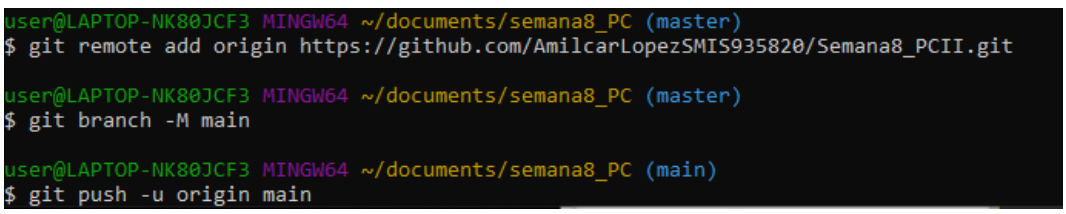
EVIDENCIA DE TRABAJO EN LA CONFERENCIA

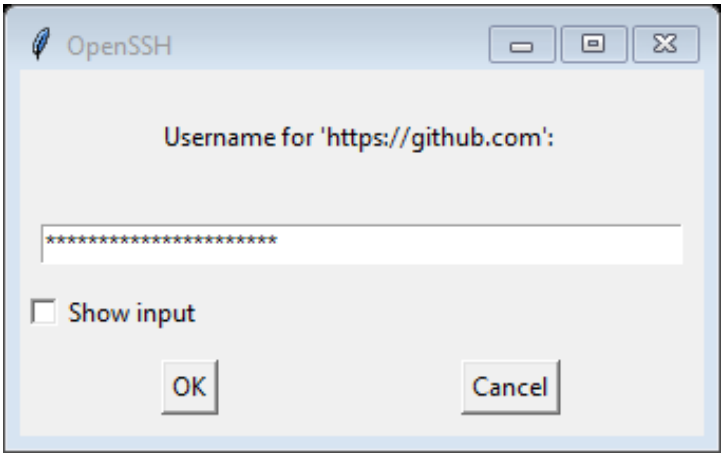


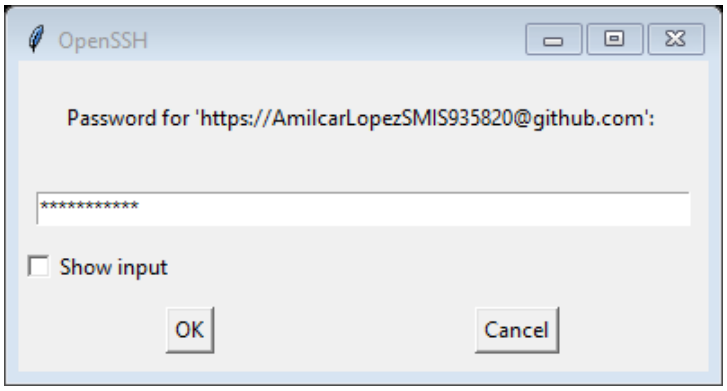



MINGW64:/c/Users/user/documents/semana8_PC
user@LAPTOP-NK80JCF3 MINGW64 ~/documents/semana8_PC (master)
$ git add archivo2.txt
user@LAPTOP-NK80JCF3 MINGW64 ~/documents/semana8_PC (master)
$ git add archivo1.txt
user@LAPTOP-NK80JCF3 MINGW64 ~/documents/semana8_PC (master)
$ git status -s
A archivo1.txt
A archivo2.txt
user@LAPTOP-NK80JCF3 MINGW64 ~/documents/semana8_PC (master)
$ git rm --cached archivo1.txt
rm 'archivo1.txt'
user@LAPTOP-NK80JCF3 MINGW64 ~/documents/semana8_PC (master)
$ git status -s
A archivo2.txt
?? archivo1.txt
user@LAPTOP-NK80JCF3 MINGW64 ~/documents/semana8_PC (master)
$ git add archivo1.txt
user@LAPTOP-NK80JCF3 MINGW64 ~/documents/semana8_PC (master)
$ git status -s
A archivo1.txt
A archivo2.txt

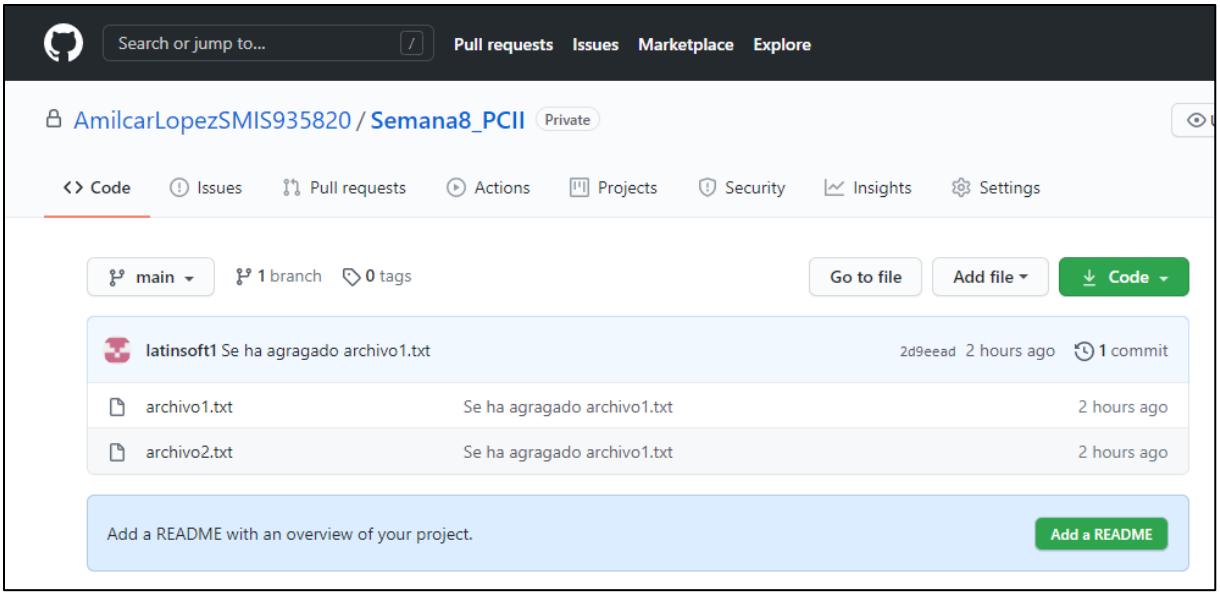

MINGW64:/c/Users/user/documents/semana8_PC
user@LAPTOP-NK80JCF3 MINGW64 ~/documents/semana8_PC (master)
$ git commit -m "Se ha agragado archivo1.txt"
[master (root-commit) 2d9eead] Se ha agragado archivo1.txt
2 files changed, 0 insertions(+), 0 deletions(-)
create mode 100644 archivo1.txt
create mode 100644 archivo2.txt
user@LAPTOP-NK80JCF3 MINGW64 ~/documents/semana8_PC (master)
$ git commit -m "Se han agregado archivos"
On branch master
nothing to commit, working tree clean
user@LAPTOP-NK80JCF3 MINGW64 ~/documents/semana8_PC (master)
$ git status -s
user@LAPTOP-NK80JCF3 MINGW64 ~/documents/semana8_PC (master)
$ git log --oneline
2d9eead (HEAD -> master) Se ha agragado archivo1.txt
user@LAPTOP-NK80JCF3 MINGW64 ~/documents/semana8_PC (master)
$ git reset --hard 2d9eead
HEAD is now at 2d9eead Se ha agragado archivo1.txt
user@LAPTOP-NK80JCF3 MINGW64 ~/documents/semana8_PC (master)
$ git log --oneline
2d9eead (HEAD -> master) Se ha agragado archivo1.txt


user@LAPTOP-NK80JCF3 MINGW64 ~/documents/semana8_PC (master)
$ git remote add origin https://github.com/AmilcarLopezSMIS935820/Semana8_PCII.git
user@LAPTOP-NK80JCF3 MINGW64 ~/documents/semana8_PC (master)
$ git branch -M main
user@LAPTOP-NK80JCF3 MINGW64 ~/documents/semana8_PC (main)
$ git push -u origin main






```
user@LAPTOP-NK80JCF3 MINGW64 ~/documents/semana8_PC (main)
$ git push -u origin main
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 235 bytes | 13.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/AmilcarLopezSMIS935820/Semana8_PCII.git
 * [new branch]      main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.
```



Nota: En el mensaje del commit debería decir “se han agregado los archivos”, sin embargo, me equivoque al escribir el mensaje, fuera de eso, todo ha funcionado bien…

CONCLUSIONES

Al principio puede que usar Git nos parezca algo difícil, pero una vez nos acostumbramos se vuelve indispensable para cada proyecto y hasta nuestra forma de trabajar se vuelve más eficiente, por eso muchas compañías exitosas utilizan el control de versiones para trabajar, como Google o Facebook, pues facilita el trabajo en equipo, además de ello, ya no existiría la necesidad de enviarse archivos comprimidos de un correo a otro, ya que todo se encontrara centralizado en un solo lugar de trabajo; donde cualquiera del equipo con los permisos necesarios puede tener acceso.

Como se mencionó previamente, Git nos proporciona una función que nos ayudará a guardar el código fuente de nuestro proyecto en un punto en el tiempo, agregándole una descripción de cada punto con los cambios que se han hecho en el código o progresos que ha tenido dicho proyecto. Este punto de guardado es conocido como Commit y Git nos permitiría movernos de manera libre entre cada uno de estos, ya sea que necesitemos regresar a un estado anterior del proyecto o que queramos regresar a la versión más actualizada, podremos hacerlo y con ello visualizar todos los commits hechos en un log o historial, donde se mostrara la descripción o razón del commit y en un orden inversamente cronológico.

Si nos centramos en la idea de como nos beneficiaria tanto como universitarios y futuros ingenieros en sistemas, podemos expresar lo siguiente:

- La introducción a Git nos ayudaría como alumnos a respaldar nuestro trabajo y además podríamos facilitar la comunicación del progreso del proyecto con nuestros compañeros de equipo e inclusive con el tutor de la catedra.
- Git también podría ayudarnos en el trabajo para lograr un mejor rendimiento y trabajar de una manera más eficiente, reduciendo con ello los tiempos durante la corrección de errores.

Git además nos permite gestionar cambios en un proyecto mediante un sencillo flujo de trabajo local y podemos entenderlo de la siguiente manera:

1. Modificamos uno o más archivos en nuestro directorio.
2. Luego los añadimos a nuestra área de preparación (repositorio local).
3. Confirmamos los cambios de los archivos, los cuales se toman del área de preparación y se guardan de en una base de datos local o en un repositorio de GitHub.

También debemos entender que el directorio en el que trabajaríamos sería una copia de una versión de nuestro proyecto sobre el cual podemos hacer modificaciones (modified), luego las añadimos al área de preparación, para luego confirmar (committed) lo que tenemos preparado y guardarlo en nuestro repositorio remoto en GitHub.

Por esta y muchas más razones, el uso de Git es muy eficiente y recomendable para cualquier desarrollador, ya sea desde una perspectiva de estudiante o un ambiente laboral...