

	<p style="text-align: center;">UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERIA, ESCUELA DE COMPUTACION</p>
<p style="text-align: center;">Ciclo II</p>	<p style="text-align: center;">DISEÑO Y PROGRAMACION DE SOFTWARE MULTIPLATAFORMA</p> <p style="text-align: center;">Guía de Laboratorio No. 1</p> <p style="text-align: center;">Introducción a Git</p>

I. RESULTADOS DE APRENDIZAJE

Que el estudiante:

- Configure el entorno de trabajo para poder iniciar a versionar los proyectos
- Utilice Git Bash para crear repositorios locales y llevarlos a repositorios remotos

II. INTRODUCCIÓN

SISTEMA DE CONTROL DE VERSIONES

Git, es un software de control de versiones diseñado por **Linus Torvalds**. La pregunta es **¿qué es control de versiones?** Pues bien, se define como control de versiones a la gestión de los diversos cambios que se realizan sobre los elementos de algún producto o una configuración del mismo es decir a la gestión de los diversos cambios que se realizan sobre los elementos de algún producto o una configuración, y para los que aún no les queda claro del todo, control de versiones es lo que se hace al momento de estar desarrollando un software o una página web.

Exactamente es eso que haces cuando subes y actualizas tu código en la nube, o le añades alguna parte o simplemente le editas cosas que no funcionan como deberían o al menos no como tú esperarías.

Y, entonces **¿a qué le llamamos sistema de control de versiones?** Muy sencillo, son todas las herramientas que nos permiten hacer todas esas modificaciones antes mencionadas en nuestro código y hacen que sea más fácil la administración de las distintas versiones de cada producto desarrollado; es decir Git.

GIT

Git fue creado pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente, es decir Git nos proporciona las herramientas para desarrollar un trabajo en equipo de manera inteligente y rápida y por trabajo nos referimos a algún software o página que implique código el cual necesitemos hacerlo con un grupo de personas.

Algunas de las características más importantes de Git son:

- Rapidez en la gestión de ramas, debido a que Git nos dice que un cambio será fusionado mucho más frecuentemente de lo que se escribe originalmente.
- Gestión distribuida; Los cambios se importan como ramas adicionales y pueden ser fusionados de la misma manera como se hace en la rama local.
- Gestión eficiente de proyectos grandes.
- Realmacenamiento periódico en paquetes.

Sistemas de Control de Versiones Distribuidos

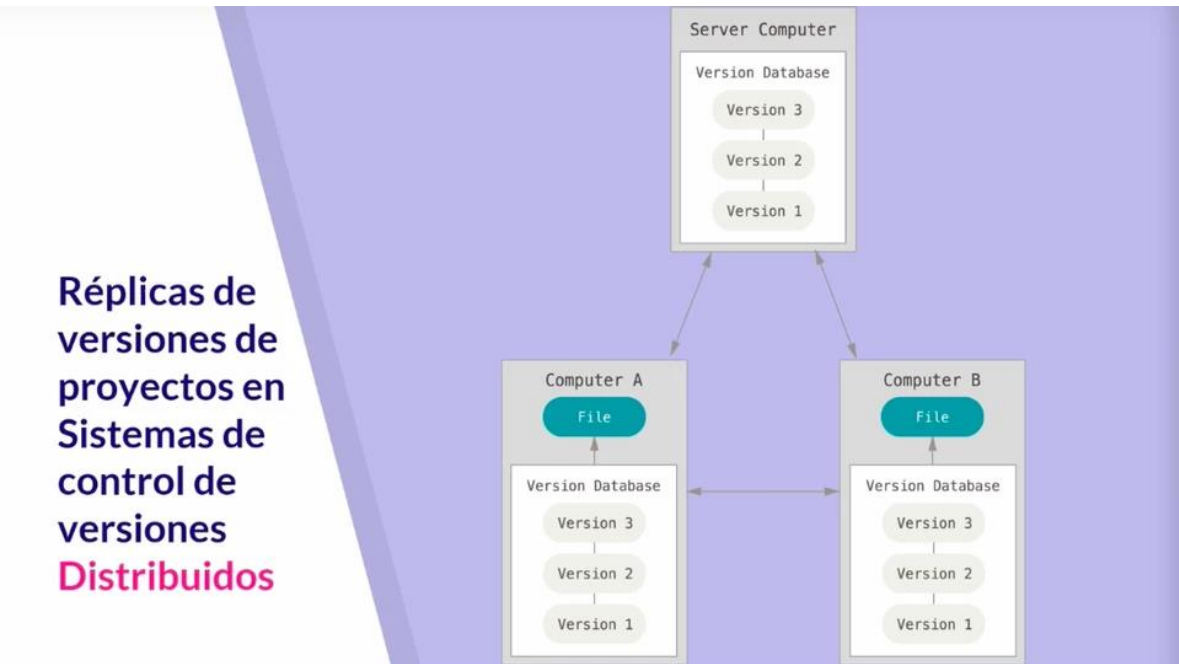
Los Sistemas de Control de Versiones Centralizados (VCS), como por ejemplo Subversion, que es una herramienta en la que se ha confiado para albergar el histórico de revisión de versiones,

es un punto centralizado, lo cual puede llegar a suponer una merma de trabajo si perdemos la conectividad de la red.

Los **Sistemas de Control de Versiones Distribuidos (DVCS)** salvan este problema. Algunos ejemplos de sistemas distribuidos, aparte de **Git**, son **Mercurial**, **Bazaar** o **Darcs**. En este tipo de herramientas, los clientes replican completamente el repositorio.

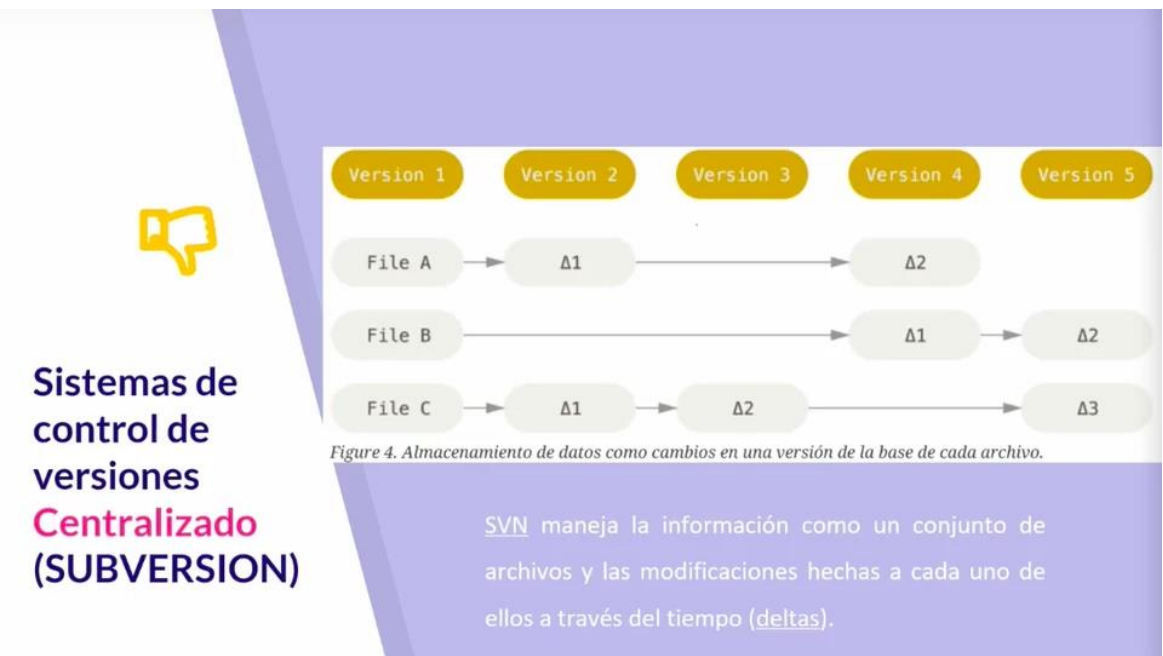
Funcionamiento de las réplicas

Las réplicas de versiones de proyectos en Sistemas de Control de Versiones Distribuidos funcionan como podemos ver en este gráfico:



Podemos tener una máquina que actúa como servidor, en el mismo se van almacenando las diferentes versiones de nuestro código, y cada uno de los clientes que participen en ese desarrollo tiene el histórico de las revisiones completa.

Sistemas de Control de Versiones Centralizados vs Sistemas de Control de Versiones Distribuidos



En **Sistemas de Control de Versiones Centralizados**, como Subversion, partimos de una versión, por ejemplo, de la versión 1 y tenemos tres ficheros, A, B y C.

De la versión 1 a la versión 2, las diferencias, como si fueran esa especie de incrementos que llamamos deltas, son almacenados por el sistema. De esta manera, por buscar algún símil, es como si Subversion trabajase con backups incrementales.

Después tenemos el funcionamiento de Sistemas de Control de Versiones de Git, en el que cada vez que hay cambios de ficheros éste es almacenado otra vez, y si no hay cambios es como si tuviésemos una especie de apuntador al fichero que no ha tenido cambios, en una revisión o en un hito del tiempo anterior.

Existen tres tipos de estado de un fichero Git:



Fundamentalmente, un proyecto Git se estructura en tres partes:

- El área del **working directory**, que es dónde vamos a tener todos nuestros ficheros, dónde estamos trabajando constantemente.
- El **staging area**, que es donde van los archivos que estamos modificando y que aceptamos para que vayan en una futura revisión.
- El **área de commit o el git directory**, que es dónde se almacenan la revisión completa. A lo largo de nuestro **curso de Git**, se explicará cómo podemos movernos a lo largo de esos tres estados, para qué sirven y por qué suponen una ventaja.

En resumen, podemos decir que Git es una herramienta:

- **Potente**
- **Rápida**
- **Multiplataforma**
- Que se puede utilizar a través de la **línea de comando o con múltiples clientes**
- Es la base o el primer eslabón de herramientas de estructura a plataforma tipo **GitHub o GitLab**, que son las plataformas que se utilizan, de manera masiva, para albergar proyectos de software libre y otro tipo de proyectos que pueda tener una organización, y que quieran delegar en GitHub o en GitLab como su Servicio de Gestión de Control de Versiones Distribuidas.

ORDENES BÁSICAS

Iniciar un repositorio vacío en una carpeta específica.

```
git init
```

Añadir un archivo específico.

```
git add "nombre_de_archivo"
```

Añadir todos los archivos del directorio

```
git add .
```

Confirmar los cambios realizados. El “mensaje” generalmente se usa para asociar al commit una breve descripción de los cambios realizados.

```
git commit -am "mensaje"
```

Revertir el commit identificado por "hash_commit"

```
git revert "hash_commit"
```

Subir la rama(branch) “nombre_rama” al servidor remoto.

```
git push origin "nombre_rama"
```

Mostrar el estado actual de la rama(branch), como los cambios que hay sin hacer commit.

```
git status
```

Herramientas por utilizar:

git bash



Git Bash es un sistema de gestión de control de código fuente para Windows. Permite a los usuarios escribir comandos Git que facilitan la administración del código fuente a través del control de versiones y el historial de confirmación. Descargue e instale git bash en su equipo:

<https://git-scm.com/downloads>

Crear cuenta **Github**



<https://github.com/>

IV DESARROLLO

Antes de iniciar es importante definir el idioma en el cual se va trabajar (Español o English), pero no pueden ser ambos. **Lista de comandos más usados :** <https://cutt.ly/im3hLu8>

1. Crear una carpeta llamada “EjemploGit” en su equipo local, donde tendremos el repositorio.
2. Dentro de la carpeta **EjemploGit** crear archivo llamado “**holaMundo.html**” con el siguiente código, pueden utilizar su editor de texto de preferencia.

```
<HTML>

<HEAD>

<TITLE>ejemplo hola mundo</TITLE>

</HEAD>

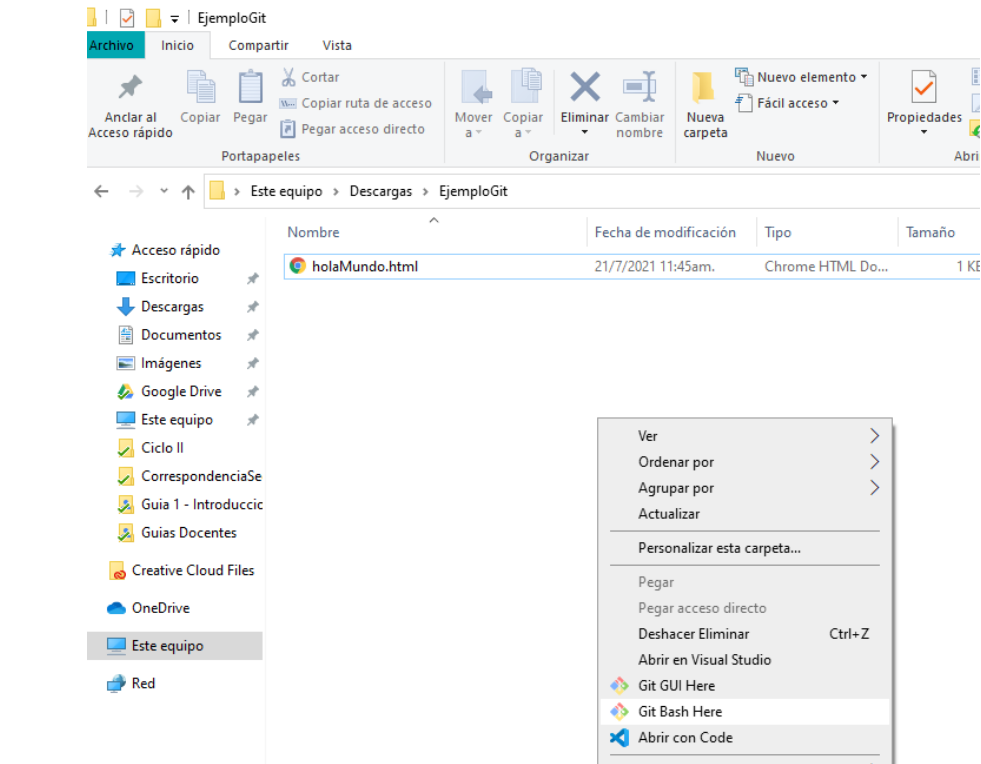
<BODY>

<P>Hola Mundo</P>

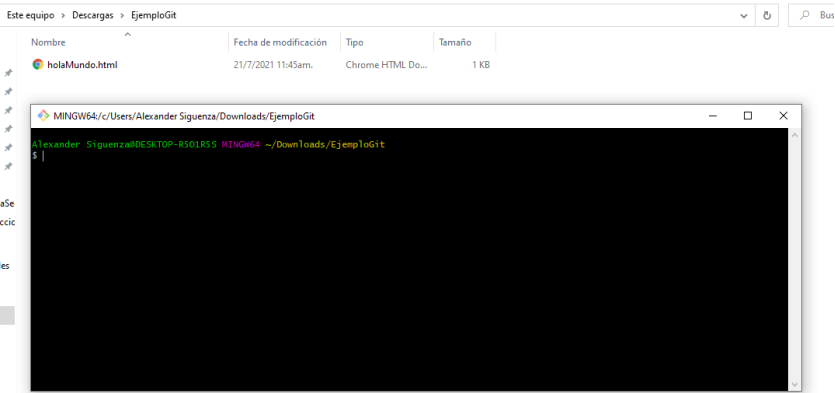
</BODY>

</HTML>
```

3. Ingresamos a dicha carpeta y damos clic derecho dentro de la carpeta vacía y veremos la opción que dice “Git Bash Here”.



Se visualizará una pantalla similar a esta:



4. Dentro de la carpeta donde crearemos nuestro repositorio comenzaremos con algunos comandos básicos que son muy necesarios. Necesitamos ejecutar el siguiente comando para iniciar nuestro usuario que ya creamos en <https://github.com/>

```
Alexander Siguenza@DESKTOP-R501R5S MINGW64 ~
$ git config --global user.name "alexander.siguenza"

Alexander Siguenza@DESKTOP-R501R5S MINGW64 ~
$ git config --global user.email "alexander.siguenza@gmail.com"

Alexander Siguenza@DESKTOP-R501R5S MINGW64 ~
$ git config --global -l
core.autocrlf=true
user.name=alexander.siguenza
user.email=alexander.siguenza@gmail.com

Alexander Siguenza@DESKTOP-R501R5S MINGW64 ~
$ |
```

```
git config --global user.name "Usuario"
git config --global user.email "usuario@dominio.com"
```

5. Para poder revisar la configuración de nuestro git solo tendremos que escribir el siguiente comando


```
git config --global -l
```


6. Caso tengamos otro usuario ya configurado, pero queremos agregar uno nuevo entonces con el siguiente comando.

```
git config --global --replace-all user.name "Your New Name"
git config --global --replace-all user.email "Your new email"
```

7. Cconfigurado nuestro usuario, pasamos a iniciar nuestro repositorio local con el siguiente comando

```
git init
```

Nombre	Fecha de modificación	Tipo	Tamaño
 holaMundo.html	21/7/2021 11:45am.	Chrome HTML Do...	1 KB


 MINGW64:/c/Users/Alexander Siguenza/Downloads/EjemploGit

```
Alexander Siguenza@DESKTOP-R501R5S MINGW64 ~/Downloads/EjemploGit
$ git init
Initialized empty Git repository in C:/Users/Alexander Siguenza/Downloads/EjemploGit/.git/

Alexander Siguenza@DESKTOP-R501R5S MINGW64 ~/Downloads/EjemploGit (master)
$ |
```

8. Luego digitar el siguiente comando agregando todos los cambios que le realicemos a nuestro repositorio local.

```
git add -A
```

 MINGW64:/c/Users/Alexander Siguenza/Downloads/EjemploGit

```
Alexander Siguenza@DESKTOP-R501R5S MINGW64 ~/Downloads/EjemploGit
$ git init
Initialized empty Git repository in C:/Users/Alexander Siguenza/Downloads/EjemploGit/.git/

Alexander Siguenza@DESKTOP-R501R5S MINGW64 ~/Downloads/EjemploGit (master)
$ git add -A

Alexander Siguenza@DESKTOP-R501R5S MINGW64 ~/Downloads/EjemploGit (master)
$ |
```

9. Ahora para hacer commit (donde subiremos los cambios a nuestro repositorio en línea) usaremos el siguiente comando

```
git commit -m "primer version"
```

```
MINGW64:/c:/Users/Alexander Siguenza/Downloads/EjemploGit

Alexander Siguenza@DESKTOP-R501R55 MINGW64 ~/Downloads/EjemploGit
$ git init
Initialized empty Git repository in C:/Users/Alexander Siguenza/Downloads/EjemploGit/.git/

Alexander Siguenza@DESKTOP-R501R55 MINGW64 ~/Downloads/EjemploGit (master)
$ git add -A

Alexander Siguenza@DESKTOP-R501R55 MINGW64 ~/Downloads/EjemploGit (master)
$ git commit -m "primer version"
[master (root-commit) b5548b2] primer version
1 file changed, 13 insertions(+)
create mode 100644 holaMundo.html

Alexander Siguenza@DESKTOP-R501R55 MINGW64 ~/Downloads/EjemploGit (master)
$ |
```

Escribe mejores mensajes de commit:

1. Verbo en imperativo. <https://www.ejemplos.co/verbos-en-imperativo/>
 2. No uses puntos ni puntos suspensivos.
 3. Hazlo corto y conciso. Por debajo de 50 caracteres.
 4. Añade un cuerpo de mensaje si es necesario.
10. Observamos que se ha agregado un archivo al repositorio local, tenemos la opción de corroborar los commit que hemos hecho en esta rama con los siguientes comandos.

```
git log
git log --oneline --decorate --all --graph
```

```
Alexander Siguenza@DESKTOP-R501R55 MINGW64 ~/Downloads/EjemploGit (master)
$ git log
commit b5548b2fbd04c24ab6a5540caccb6201daafe973 (HEAD -> master)
Author: alexander.siguenza <alexander.siguenza@gmail.com>
Date:   Wed Jul 21 15:14:31 2021 -0600

    primer version

Alexander Siguenza@DESKTOP-R501R55 MINGW64 ~/Downloads/EjemploGit (master)
$ git log --oneline --decorate --all --graph
* b5548b2 (HEAD -> master) primer version

Alexander Siguenza@DESKTOP-R501R55 MINGW64 ~/Downloads/EjemploGit (master)
$ |
```

Ya sea cualquiera de los dos comandos podremos ver los commit realizados, solo que para el caso del segundo los podremos observar de forma ordenada y más limpia

11. Para listar todos los repositorios remotos usaremos el siguiente comando

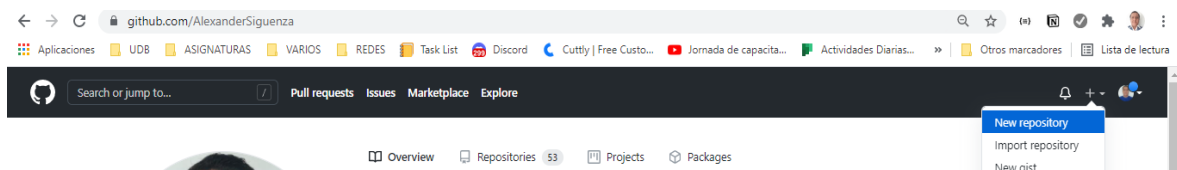
```
git remote --v
```

```
MINGW64:/c:/Users/Alexander Siguenza/Downloads/EjemploGit

Alexander Siguenza@DESKTOP-R501R55 MINGW64 ~/Downloads/EjemploGit (master)
$ git remote --v

Alexander Siguenza@DESKTOP-R501R55 MINGW64 ~/Downloads/EjemploGit (master)
$ |
```

12. En este caso como no tenemos ninguno, no saldrá nada, tenemos que ir a la página de <https://github.com/> debemos de ingresar con las credenciales creadas, y **Create a new repository**.



13. Debemos de colocar un nombre al repositorio, aconsejable es llamarle como el proyecto local **“EjemploGit”**, se tienen dos opciones **public o private** para efectos de pruebas no es necesario hacerlo privados. Son las dos opciones a modificar, **Initialize this repository with**, quedan vacías. Para finalizar presionar **“Create repository”**

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner * Repository name *

AlexanderSiguenza / EjemploGit ✓

Great repository names are short and unique. EjemploGit is available. [Inspiration?](#) How about [animated-succotash](#)?

Description (optional)

☒ **Public**
Anyone on the internet can see this repository. You choose who can commit.

☐ **Private**
You choose who can see and commit to this repository.

Initialize this repository with:
Skip this step if you're importing an existing repository.

☐ **Add a README file**
This is where you can write a long description for your project. [Learn more.](#)

☐ **Add .gitignore**
Choose which files not to track from a list of templates. [Learn more.](#)

☐ **Choose a license**
A license tells others what they can and can't do with your code. [Learn more.](#)

[Create repository](#)

14. Una vez creado el repositorio remoto usaremos el siguiente comando que se muestra al haber completado la creación del repositorio

...or create a new repository on the command line

```
echo "# EjemploGit" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/AlexanderSiguenza/EjemploGit.git
git push -u origin main
```

Si observamos los comandos, son los que ya se realizaron, para crear el repositorio local.

```
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/AlexanderSiguenza/EjemploGit.git
git push -u origin main
```

El comando es el siguiente, por supuesto será diferente para cada uno, por el nombre de usuario

```
git remote add origin https://github.com/AlexanderSiguenza/EjemploGit.git
```

Con este comando se configura el repositorio remoto, para que el código este versionado tanto de **manera local y remoto**. Al revisar nuevamente el comando **git remote -v**, podemos observar el repositorio configurado.


```
MINGW64:/c/Users/Alexander Siguenza/Downloads/EjemploGit

Alexander Siguenza@DESKTOP-R501R55 MINGW64 ~/Downloads/EjemploGit (master)
$ git remote --v

Alexander Siguenza@DESKTOP-R501R55 MINGW64 ~/Downloads/EjemploGit (master)
$ git remote add origin https://github.com/AlexanderSiguenza/EjemploGit.git

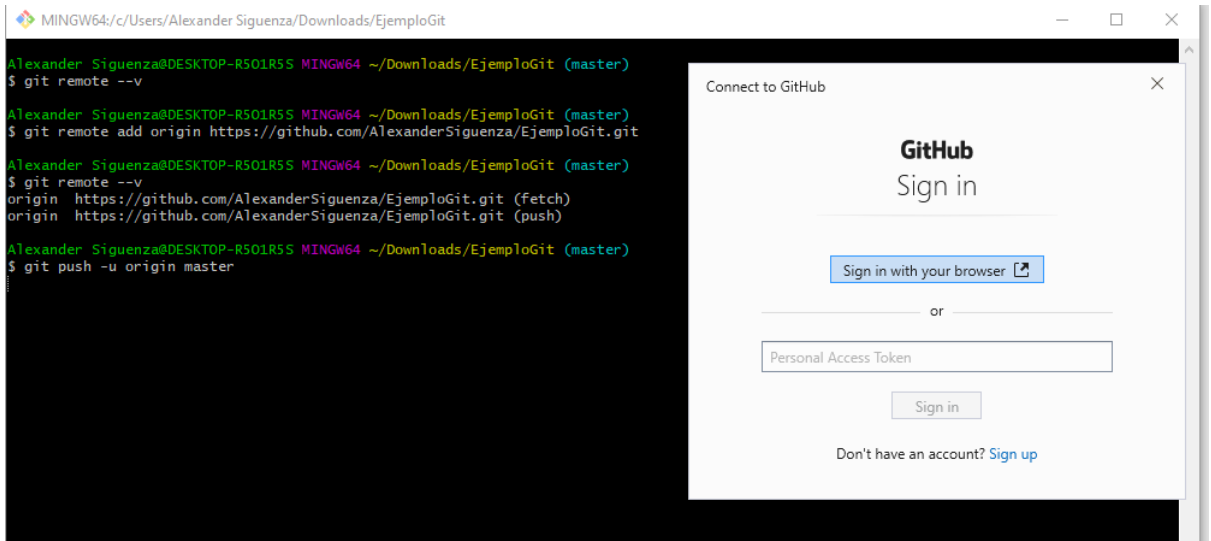
Alexander Siguenza@DESKTOP-R501R55 MINGW64 ~/Downloads/EjemploGit (master)
$ git remote --v
origin https://github.com/AlexanderSiguenza/EjemploGit.git (fetch)
origin https://github.com/AlexanderSiguenza/EjemploGit.git (push)

Alexander Siguenza@DESKTOP-R501R55 MINGW64 ~/Downloads/EjemploGit (master)
$ |
```

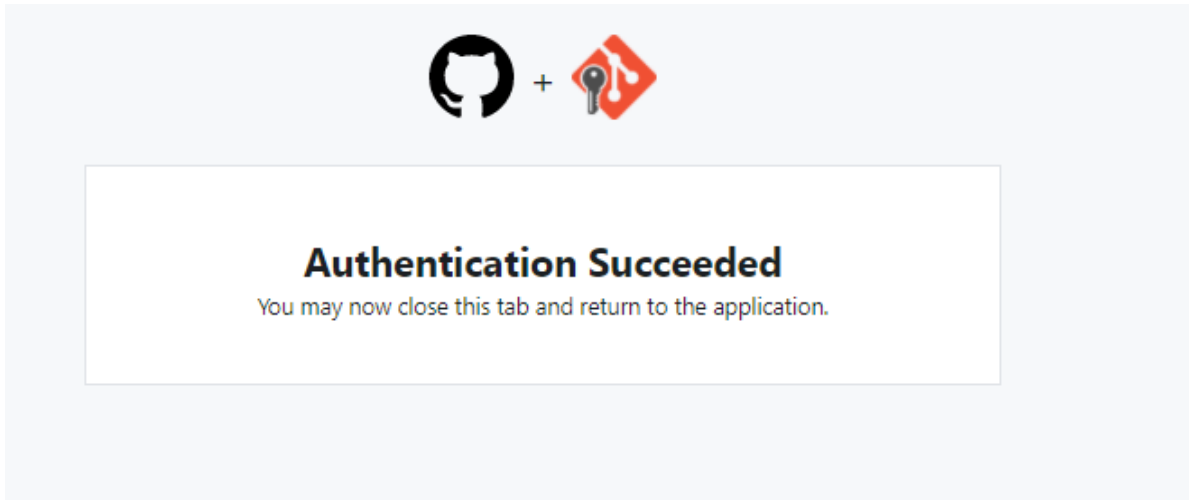
15. Solo queda un paso más para enviar los cambios locales al repositorio remoto, eso lo haremos con el siguiente comando.

```
git push -u origin master
```

Al ejecutar el comando, nos pedira ingresar los accesos de Github, presionamos la opcion **“Sign in with your browser”**




Como se tiene activa la sesión en el navegador, la autenticación será automática:



Despues de la autentificacion los archivos del proyectos deben de subirse al repositorio remoto.

```
MINGW64:/c:/Users/Alexander Siguenza/Downloads/EjemploGit
Alexander Siguenza@DESKTOP-R501R5S MINGW64 ~/Downloads/EjemploGit (master)
$ git remote --v
Alexander Siguenza@DESKTOP-R501R5S MINGW64 ~/Downloads/EjemploGit (master)
$ git remote add origin https://github.com/AlexanderSiguenza/EjemploGit.git
Alexander Siguenza@DESKTOP-R501R5S MINGW64 ~/Downloads/EjemploGit (master)
$ git remote --v
origin https://github.com/AlexanderSiguenza/EjemploGit.git (fetch)
origin https://github.com/AlexanderSiguenza/EjemploGit.git (push)
Alexander Siguenza@DESKTOP-R501R5S MINGW64 ~/Downloads/EjemploGit (master)
$ git push -u origin master
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 6 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 291 bytes | 291.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/AlexanderSiguenza/EjemploGit.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
Alexander Siguenza@DESKTOP-R501R5S MINGW64 ~/Downloads/EjemploGit (master)
$ |
```

Caso contrario no este activa la sesión, se debe de ingresar con usuario/contraseña



Sign in to GitHub
to continue to **Git Credential Manager**

Username or email address

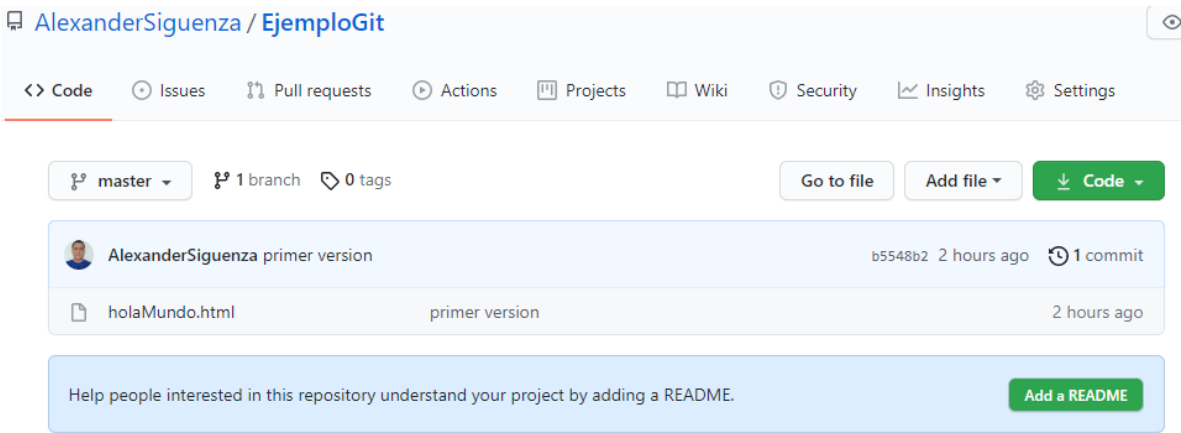
Password

[Forgot password?](#)

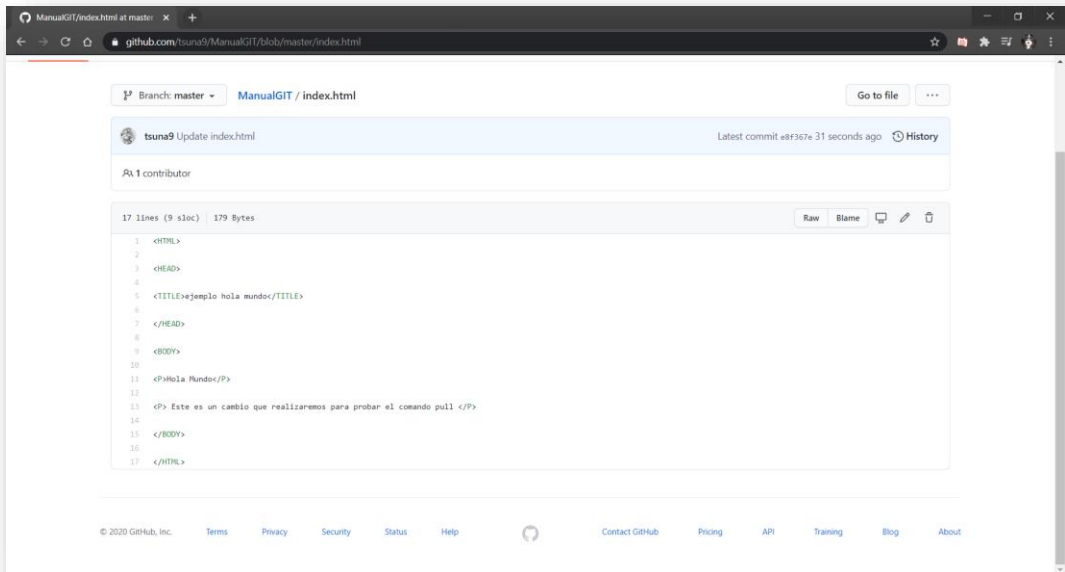
Sign in

New to GitHub? [Create an account.](#)

16. Podemos observar que los cambios han sido enviados, ahora solo queda corroborar si el archivo que creamos anteriormente se ha subido al repositorio remoto.

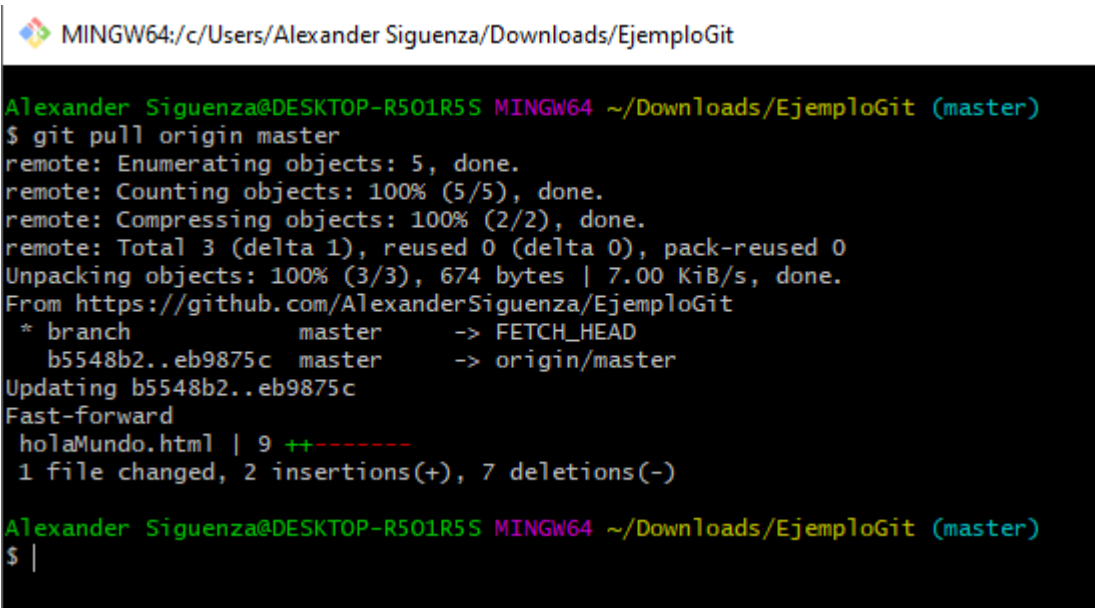


17. Ahora modificamos dicho archivo para traer esos cambios a nuestro repositorio local.



Se agregó un cambio al archivo remoto, entonces ahora con el siguiente comando traeremos dichos cambios al local.

```
git pull origin master
```



18. Para verificar los cambios aplicados:

```
git log --oneline --decorate --all --graph
```

MINGW64:/c:/Users/Alexander Siguenza/Downloads/EjemploGit

```
Alexander Siguenza@DESKTOP-R501R5S MINGW64 ~/Downloads/EjemploGit (master)
$ git log --oneline --decorate --all --graph
* eb9875c (HEAD -> master, origin/master) Update holaMundo.html
* b5548b2 primer version

Alexander Siguenza@DESKTOP-R501R5S MINGW64 ~/Downloads/EjemploGit (master)
$ |
```

19. Si se realiza cambios locales del archivo se debe de subir estos cambios al remoto, retomando los pasos anteriores.

Local:

- 1- git add -A
- 2- git commit -m "comentario"

Remoto:

- 1- git push -u origin master

Faltas muchos más escenarios que practicar, pero con estas practica se tiene lo esencial.

V. DISCUSION DE RESULTADOS

Para todos los ejercicios lo que deben de entregar es la Url del repositorio público, por ejemplo: <https://github.com/AlexanderSiguenza/pruebasGit.git>

1. Crear un repositorio local y luego llevarlo a Github, en una carpeta llamada **PruebasGit** , dentro de la carpeta crear una página html con el nombre **primerversion.html** con el siguiente párrafo.
 - a. "Git fue creado pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente, es decir Git nos proporciona las herramientas para desarrollar un trabajo en equipo de manera inteligente y rápida y por trabajo nos referimos a algún software o página que implique código el cual necesitemos hacerlo con un grupo de personas"
2. Crear un repositorio local y versionarlo al repositorio remoto, con una estructura de carpetas de la siguiente manera: (principal) **GuiLabDPS**
 - a. (**Subcarpetas**) guia01, guia02, guia03, guia04, guia05, guia06, guia07, guia08, guia09, guia10
3. Realizar cambios en el archivo **primerversion.html**, desde el repositorio remoto, ingresar a github y buscar el repositorio y agregar a la página el siguiente Párrafo.
 - a. "Los Sistemas de Control de Versiones Distribuidos (DVCS) salvan este problema. Algunos ejemplos de sistemas distribuidos, aparte de Git, son Mercurial, Bazaar o Darcs. En este tipo de herramientas, los clientes replican completamente el repositorio."

4. Luego de agregar el párrafo, deben de actualizar el repositorio local, con los cambios realizados en el remoto.
5. Para estructura **GuiLabDPS** , clonar la página **primerversion.html** a la carpeta **guia01** y buscar 1 colaborador para trabajar la página en conjunto con un compañero, ambos deben de realizar cambios a ambos párrafos, puede ser agregar texto o quitar, no deben de comunicarse que harán , los cambios los van a realizar por separado, lo que se busca es que existan **conflictos**.
6. Luego de los cambios realizados por ambos colaboradores, el dueño del repositorio llevará los cambios al repositorio remoto y luego pedirá al colaborador hacer lo mismo, tomar captura de pantalla y comentar que sucede.
7. Para estructura **GuiLabDPS** , clonar la página **primerversion.html** a la carpeta **guia02** y hacer tres ramas de trabajo , **desarrollo** , **producción**, **pruebas** , agregar un nuevo párrafo a la rama desarrollo “rama desarrollo” luego de hacer los cambios , guardarlos en el repositorio local y llevarlos a la rama pruebas guardar los cambios y llevarlos a la rama producción , para después llevar el repositorio local al repositorio remoto.