**Objetivos de Aprendizaje**

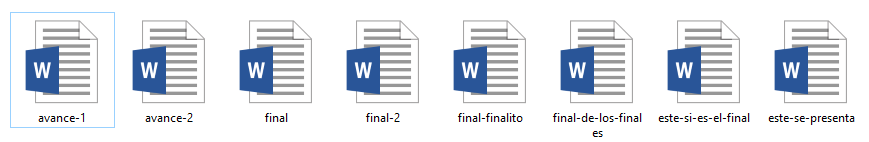
* Entender qué es Git y por qué es importante
* Crear tu cuenta en Github
* Diferenciar Git de Github
* Instalar y configurar Git
* Aprender el *"workflow"* de una developer
* Entender los conceptos básicos: local, remoto, working directory, staging area y branches
* Saber utilizar los comandos esenciales: init, status, add, commit, push y pull

El texto a continuación se basa en gran medida, con ciertos ajustes, en las siguientes páginas:

* <https://git-scm.com/book/en/v2>
* <https://github.com/wdi-sf-september-2014/notes/tree/master/git_intro>

**¿Qué es Git?**

Git es un Sistema de Control de Versiones.

**¿Qué es el Sistema de Control de Versiones?**  
El Sistema de Control de Versiones (VCS en inglés) es un sistema que registrará los cambios de uno, dos o varios archivos todo el tiempo para que si más adelante, necesitamos una versión, podamos recuperarla fácilmente. Por ejemplo, si alguna vez has elaborado un proyecto para una tarea ¿no te ha pasado algo así?:  
  
Gracias al VCS podemos hacer que eso no suceda debido a que permite:

- Revertir los archivos a un estado anterior.

- Revertir todo el proyecto a un estado anterior.

- Comparar los cambios que se hayan hecho a lo largo del proyecto.

- Ver quién introdujo un cambio o actualizó el archivo y cuándo.

- Y lo más importante, asegura que los archivos no se pierdan.

Ahora bien, Git nos ayudará a guardar y diferenciar todos los cambios que haremos durante el desarrollo de un proyecto, ya sea de uno o más archivos.

Cada vez que confirmas un cambio, o guardas el estado de tu proyecto en Git, él básicamente hace una foto del estado de todos tus archivos en ese momento, y guarda una referencia a esa foto. Para ser eficiente, si los archivos no se han modificado, Git no almacena el archivo otra vez, sólo un enlace al archivo idéntico anterior que ya almacenó.

Tanto la colección de archivos como sus versiones completas serán guardadas en un repositorio.

En resumen, Git piensa en tus archivos como un flujo de "imágenes". Por eso, a cada cambio que vayamos realizando le tomará una foto en el instante que le indiquemos y todo eso lo almacenará en un repositorio local.

**Ahora conozcamos algunas palabras claves que de seguro escucharás al momento de trabajar con Git:**

* Repositorio local: Cuando hablamos de un repositorio local hacemos referencia a los archivos y a sus versiones que se encuentran en nuestro equipo. Aquí encontraremos dos partes muy importantes.
  + working directory: Es nuestro directorio de trabajo.
  + staging area: Es el espacio en donde se encuentran los archivos a los que Git les debe de hacer seguimiento y que en algún momento se convertirán en una nueva versión del proyecto.
* Repositorio remoto: Git nos permite sincronizar nuestro repositorio local con un repositorio remoto o repositorio en la "nube".

**¿Qué es Github?**

Github es conocida como la "red social" de los developers debido a que una gran cantidad de ellos suben sus proyectos aquí. Esto les permite crear repositorios remotos que guardarán siempre una copia de su código y a la cual podrán acceder fácilmente mediante Git que es el controlador de versiones.

Pero Github no solo sirve para "subir" nuestro código, él también contribuirá al desarrollo colaborativo del proyecto brindándonos herramientas como por ejemplo:

* Un sistema de seguimiento de problemas que permite a los miembros de un equipo detallar un problema o una sugerencia que deseen hacer del proyecto.
* Una herramienta de revisión de código, donde se pueden añadir anotaciones en cualquier punto del proyecto y debatir sobre un cambio en específico.
* Un visor de versiones donde se puede comparar el progreso del proyecto en las distintas versiones.

**En resumen, se podría decir que Github es como Dropbox pero para developers.**

**¿Cuál es la diferencia entre Git y Github?**

**Git y Github se complementan pero no son iguales.**

Github utiliza a Git como Sistema de Control de Versiones. Gracias a Git podremos guardar localmente los diferentes avances (estados) de nuestro proyecto y gracias a Github podremos subir esos avances a la nube y comenzar un trabajo colaborativo o trabajar en nuestros proyectos personales sin depender de la computadora donde se encuentre el proyecto o tener que estar guardándolo en un usb para copiarlo a otra computadora.

**¿Por qué es importante Git y Github?**

La programación se da, en su mayoría de veces, en equipos. Varias empresas tienen un equipo que se complementa para cada proyecto. Ante esta situación te preguntarás:

¿Cómo se organizan para trabajar todos en un solo proyecto? Tal vez tienen todo en un usb y se lo turnan para poder trabajar, o trabajan cada uno por su lado y al momento de tener que unir las partes tienen que ver detalladamente si se repiten o no alguna parte.

Esto no sería para nada eficiente, porque no se podría trabajar al mismo tiempo en algo en específico y se tendría que depender del avance de otro para que nosotros podamos avanzar.

Para estos casos existen herramientas como Git y Github, los cuales nos ayudan al momento de un trabajo colaborativo. Github es el lugar perfecto para trabajar conjuntamente en un proyecto con uno o varios compañeros, podrás crear un proyecto y añadir como colaboradores a los miembros de tu equipo; además, podrás traer a tu repositorio local los cambios que tus compañeros hayan subido al repositorio remoto y podrás ver automáticamente lo que acaba de añadir tu compañero.

**¿Cómo crear una cuenta en Github?**

Para crear tu cuenta de Github debemos seguir los siguientes pasos:

1. Debemos de dirigirnos a la página oficial de Github: <https://github.com/>
2. Ya en la página, haremos clic en **Sign up**.
3. Ahora deberás escribir un **username**, que podrá ser uno que ya tengas. Por ejemplo, si tienes un **username** que creaste para un juego online podrías usar ese. No hay límites de cómo debe de ser, pero ten en cuenta que será visible para todo el mundo y puede que te lo pidan en algún momento para añadirte como colaborador de un proyecto, así que ten cuidado con lo que pongas como **username**. Luego ingresa tu **correo**, una **contraseña** y haz clic en **Create an account**.
4. En la siguiente pantalla asegúrate de marcar la opción **Unlimited public repositories for free**, esta es la versión "free" de Github; luego haz clic en **continue**.
5. El paso 3 del registro puedes completarlo con tus datos y hacer clic en **Submit**, o solo hacer clic en **Skip this step**.

¡Y listo! Al hacer todos estos pasos ya tendríamos nuestra cuenta de Github creada.

Ahora Michelle, siguiendo los pasos anteriores, creará su cuenta en Github.

**¿Cómo creamos un repositorio remoto en Github?**

Sigue los pasos de Michelle para crear un repositorio en Github:

Por último tenemos la pantalla donde encontrarmos la url de nuestro repositorio remoto, con la cual (más adelante) vincularemos nuestro repositorio remoto con nuestro repositorio local.

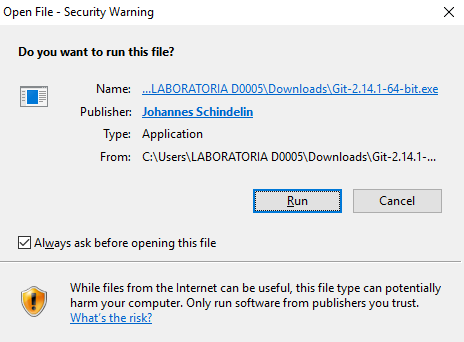
**¿Cómo instalamos Git?**

Si nos encontramos en Linux o Mac OS X no será necesario hacer la instalación de Git porque ya viene instalado, pero si nuestro sistema operativo es **Windows** debemos descargar e instalar Git en nuestro equipo.

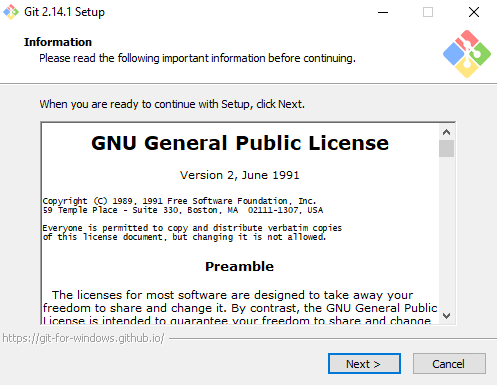
* [Descarga Git](https://git-scm.com/download/win)

Cuando haya terminado la descarga de nuestro programa comenzaremos con la instalación haciendo doble clic en el archivo que se acaba de descargar y luego nos aparecerán las siguientes pantallas:

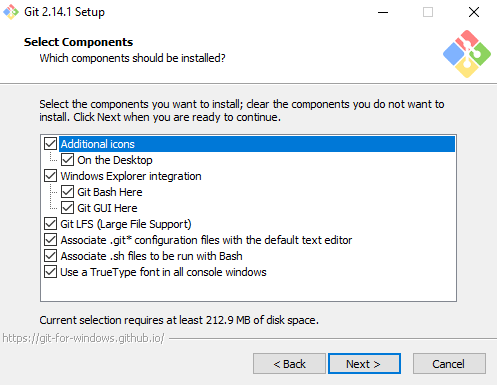
1. Hacemos clic en el botón **Run**



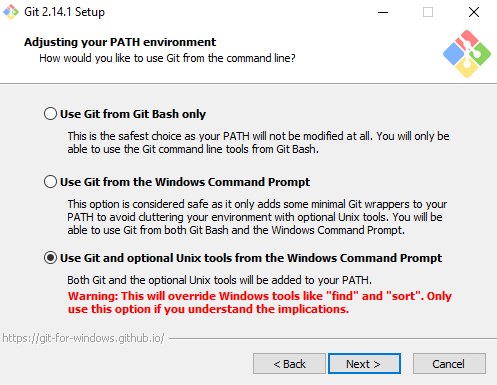
1. Ahora en **Next >**



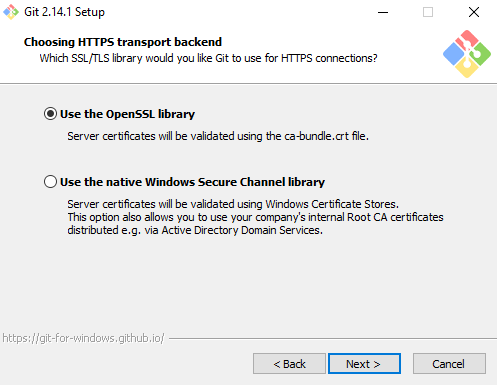
1. Nos aparecerá la siguiente pantalla, aquí seleccionaremos las siguientes opciones y hacemos clic en **Next >**



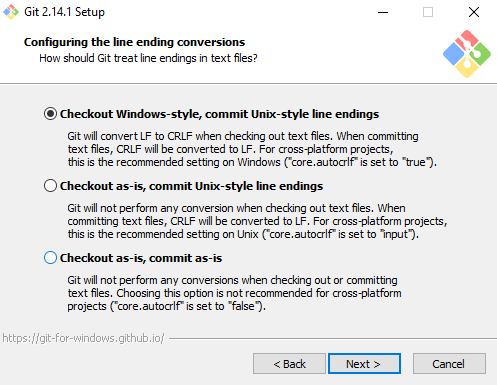
1. Seleccionamos la opción **Use Git and optional Unix tools from the Windows Command Prompt** y hacemos clic en **Next >**



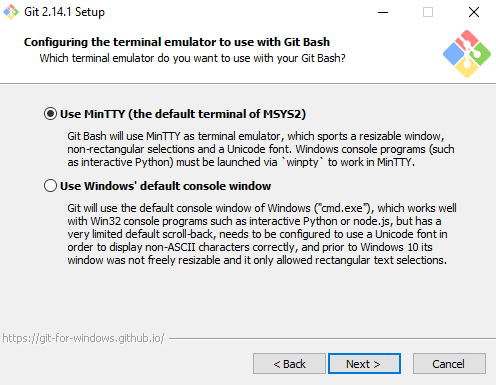
1. Seleccionamos la opción **Use the OpenSSL library** y continuamos (**Next >**)



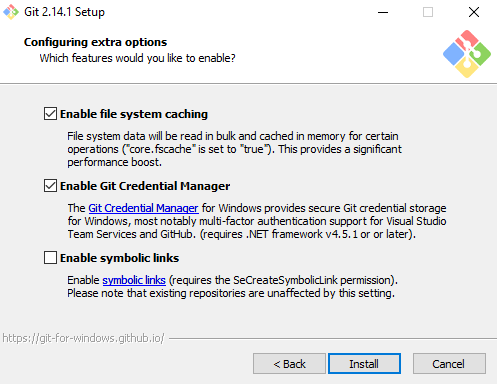
1. Elegimos la primera opción y seguimos



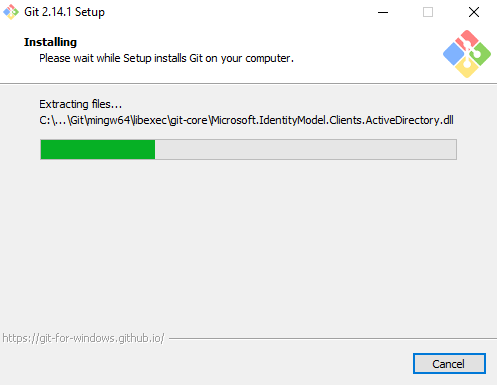
1. Nos aseguramos de escoger la opción **Use MinTTY**



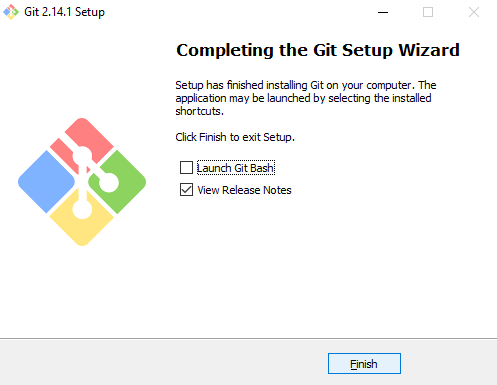
1. Seleccionamos las dos primeras opciones y hacemos clic en **Install**



1. Ahora comenzará a instalarse



1. Y ya tendremos instalado Git en nuestro equipo al hacer clic en **Finish**



**¿Qué es Git Bash?**

Cuando terminemos de instalar Git, en nuestras nuevas aplicaciones encontraremos una que se llama Git Bash.

En el sistema operativo Windows, debido a que el Command Prompt (o terminal de Windows) no es muy amigable y no soporta comandos UNIX, Git nos proporciona una herramienta muy buena llamada Git Bash que nos permitirá ejecutar comandos como si estuviéramos en Linux o Mac OS X.

Por lo tanto, Git Bash será la "línea de comandos" que utilizaremos cuando trabajemos con Git en Windows, en cambio, si nos encontramos en Linux o Mac OS X utilizaremos nuestra terminal.

**¿Cómo configuramos Git?**

Ya tenemos instaladoGit en nuestro equipo, ahora personalizaremos nuestro entorno de Git.

Recuerda que si quieres ejecutar comandos de Git y te encuentras en Mac OS X o Linux deberás ejecutarlos en **la línea de comandos**, en cambio, si tu sistema operativo es Windows abriremos **Git Bash** para ejecutar los comandos de Git.

Lo primero que debemos de hacer es establecer nuestro **username** y **dirección de correo** debido a que Gitutilizará estos datos para la confirmación de cambios (o como los conoceremos más adelante: commits), podemos configurar estos datos por medio del comando git config, este nos permite obtener y establecer variables de configuración que controlan el aspecto y funcionamiento de Git, como por ejemplo si ejecuto el comando git config --list en mi terminal me devuelve esto:

user.name = michellesegv

user.email = michelle@laboratoria.la

(Que vienen a ser los datos que ya he configurado)

Ahora nosotras vamos configurar nuestro entorno de Git:

* Para configurar nuestro **username** debemos escribir la siguiente línea que debe de tener nuestros datos, y luego presionaremos la tecla "enter":
* git config --global user.name Tu-username
* Para configurar nuestro **correo electrónico** debemos escribir la siguiente línea con nuestros datos y luego presionaremos la tecla "enter":
* git config --global user.email tucorreo@gmail.com

Los datos con los que llenes las líneas anteriores deben de ser los mismos que usaste al crear tu cuenta de Github, porque como te comentaba con ellos se registrarán los commits que hagas en Git y que luego pasarán al historial de nuestro proyecto en Github.

**¿Cómo usar Git y Github?**

Hemos leído las ventajas que nos pueden brindar Git y Github, con ellos podremos subir un repositorio local a uno remoto, trabajar con versiones y colaborativamente, etc. Ahora hagamos la "magia" de trabajar con Git y Github, vamos a subir nuestro repositorio local a uno remoto y trabajar colaborativamente.

A continuación tienes la lista de los comandos más comunes que utilizarás trabajando con Git y Github. Si te sigue pareciendo muy "teórico" no te preocupes, más abajo habrá un video de Michelle utilizándolos.

* init: Este comando le indica a Git que comenzará a hacer seguimiento de la carpeta actual. En otras palabras, comenzará a vigilar esta carpeta. El comando para iniciar el seguimiento de la carpeta sería:
* $ git init

Al ejecutar esta línea, Git creará una carpeta oculta llamada .git en el working directory, en la raíz del proyecto.

* add: Con este comando estaremos indicando qué archivos queremos que pasen de nuestro *working directory* al *staging area*. Es decir que los archivos ya han sido modificados y están **preparados** para la nueva versión del proyecto.

Por ejemplo, si quieremos añadir a nuestro *staging area* el archivo HTML llamado **index.html** debemos de ejecutar la siguiente línea:

$ git add index.html

Si queremos que todos los archivos del directorio actual se añadan al *staging area* (definir que ya están preparados), ejecutamos la siguiente línea:

$ git add .

* commit: Con commit **confirmaremos** los archivos que declaramos como preparados con el comando anterior (add) .Con los archivos confirmados se creará una nueva versión del proyecto.  
  La estructura del comando es la siguiente:
* $ git commit -m 'añadiendo un comentario'

El texto que va dentro de las comillas ('') puede ser cualquiera, te recomiendo que escribas un resumen de lo que estás subiendo en ese momento para que tengas una mejor organizacion, además, un buen comentario te servirá si quieres volver a una funcionalidad en especial sin tener que leer todo el código y solo guiarte por el comentario de tu commit. Por ejemplo:

$ git commit -m 'Creando estructura HTML'

* remote: Con remote **vincularemos** nuestro repositorio local con nuestro repositorio que se encuentra en la "nube" por medio de la **url**. Este paso no se debe de hacer cada vez que creemos una nueva versión, solo cuando queramos especificar el repositorio remoto que tendremos. Entonces, para enlazar nuestro repositorio local con nuestro repositorio remoto debemos de ejecutar el siguiente comando:
* $ git remote add nombre-remoto url

Normalmente, la mayoría de los developers, como **nombre-remoto** utiliza origin y **url** es la "ruta" (url) de nuestro repositorio en Github. Por ejemplo, hace un momento he creado el repositorio "prueba" en Github y si quiero vincular mi repositorio local con ese repositorio remoto tendría que ejecutar la siguiente línea de comando:

$ git remote add origin https://github.com/michellesegv/prueba.git

* push: Por medio de push **enviaremos** nuestra nueva versión que confirmamos en el commit a nuestro repositorio remoto. El comando es el siguiente:
* $ git push nombre-remoto master

En **nombre-remoto** pondremos el mismo nombre-remoto que le hemos asignado al momento de añadir la url de nuestro proyecto (en nuestro caso sería **origin**) y seguido pondremos **master**, más adelante conoceremos el trabajo en "ramas" que tiene Github. Las **ramas** son una copia paralela que podemos crear de nuestro código. Por defecto, los repositorios traen una "rama" llamada **master** y ahora trabajaremos solo en ella, así que subiremos nuestros cambios a ella especificando **master**como el nombre de la rama a la que queremos subir nuestra versión, entonces el comando a ejecutar sería:

$ git push origin master

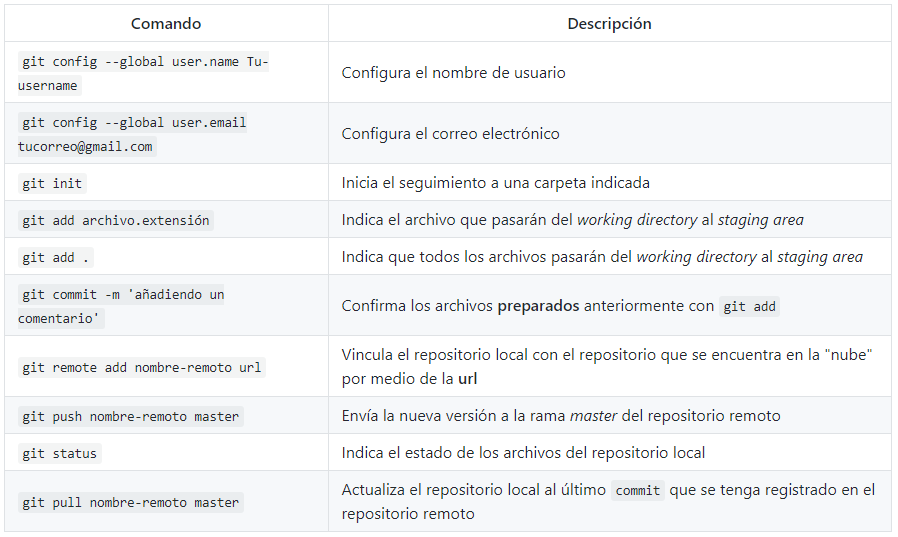
* status: Con status conoceremos el estado de los archivos de nuestro repositorio local. Es decir, nos dirá si existe un archivo que se ha modificado y no se ha actualizado en el repositorio remoto o si hay un archivo que todavía no se "sube" y nos informará si se encuentra en nuestro working directoy o staging area. Este comando nos resultará muy útil para determinar qué archivos debemos declarar como "preparados" (por medio del comando add) y luego "confirmarlos" en la nueva versión (por medio del comando commit) que subiremos a nuestro repositorio remoto. El comando para ver el estado de nuestro repositorio local es:
* $ git status
* pull: Sirve para actualizar nuestro repositorio local al últimocommit que tengamos registrado en el repositorio remoto. Este comando nos servirá mucho al momento de trabajar colaborativamente porque podremos tener los avances que nuestros colaboradores han subido al repositorio remotro a nuestro repositorio local y mantenerlo actualizado. El comando es el siguiente:
* $ git pull nombre-remoto master

En nuestro caso, como nuestro **nombre-remoto** ya lo declaramos como **origin** y la rama en la que estamos trabajando se llama **master**, el comando sería:

$ git pull origin master

**Detalle de comandos**

A continuación veremos una lista de los comandos de Git usados en esta lección:



Ahora veamos cómo Michelle utiliza Git y Github para manejar el control de versiones de sus proyectos y colaborar con otras profesoras de Laboratoria.