Clase 5:
GIT

Índice

1. Introducción a Git
- Qué es?
- Cómo surgió?
- Pequeña biografía de su creador.
2. Instalación Git
3. Creación de repositorios locales
4. Agregado de archivos al repositorio local
5. Confirmación de archivos en el repositorio local

Git

¿Qué vamos a ver en esta clase?

- En esta clase, vamos a aprender una herramienta que nos acompañará a lo largo de nuestro camino como programadoras: Git
- Es muy útil para **compartir proyectos con distintas personas**, tener **varias versiones de un mismo proyecto y poder guardar proyectos en la nube. Es por esto que es una de las herramientas más requeridas por el mercado a la hora de trabajar con control de versiones y de forma colaborativa**.
- Como muchas de las herramientas que venimos viendo, **Git requiere de práctica** y tiene muchos accesorios que agregan funcionalidades, pero en esta clase nos centraremos en sus cuestiones principales.

Diferencia entre Git, Git Bash y Git Hub

Diferencia entre Git, Git Bash y Git Hub

¿Qué es Git?

Git es un sistema de control de versiones. Sirve para llevar un registro de los cambios que hacemos en nuestro código o archivos a lo largo del tiempo.

¿Para qué se usa?

- Para trabajar en equipo en proyectos de software.
- Para guardar el historial de cambios.
- Para saber quién hizo qué cambio y cuándo.
- Para poder volver atrás si algo sale mal.

Diferencia entre Git, Git Bash y Git Hub

¿Qué es Git Bash?

Git Bash es una herramienta o programa que instalamos en nuestra computadora para usar Git desde una consola (terminal/línea de comandos). Permite escribir comandos para interactuar con Git.

Git Bash no es Git. Es solo una forma de comunicarte con Git usando comandos.

¿Para qué se usa?

- Para escribir comandos, como “git init”, “git add”, “git commit”.
- Para tener una consola similar a Linux en Windows.
- Para usar Git de forma más flexible y profesional.

Diferencia entre Git, Git Bash y Git Hub

¿Qué es Git Hub?

GitHub es una plataforma en la nube (una página web) donde puedes guardar tus repositorios Git para compartirlos, colaborar y trabajar en equipo.

¿Para qué se usa?

- Para subir nuestros proyectos y tener una copia de seguridad en internet.
- Para trabajar con otras personas de manera colaborativa.
- Para mostrar nuestro trabajo a otros (como un portafolio).
- Para contribuir a proyectos de código abierto.

Diferencia entre Git, Git Bash y Git Hub

Elemento	¿Qué es?	¿Dónde se usa?	¿Para qué sirve?
Git	Un sistema de control de versiones	En tu computadora	Guardar cambios de tus archivos y proyectos
Git Bash	Una terminal para usar Git (en Windows)	En tu computadora	Escribir comandos para manejar Git
GitHub	Una plataforma en línea para repositorios	En internet (github.com)	Guardar, compartir y colaborar en proyectos Git

Pensemos el flujo de trabajo de la siguiente forma:

1. Creamos un proyecto → usamos Git para controlarlo.
2. Escribimos comandos en Git Bash.
3. Subimos el proyecto a GitHub para que otros lo vean o colaboren.

1. Introducción a Git

Git

Introducción a Git

¿Cómo se trabaja en forma colaborativa en un ambiente profesional? ¿Nos enviamos mails con código? ¿Mensajes de WhatsApp? ¿Usamos Google Drive? Mmm... ¿suena raro, no?

Definitivamente no usamos nada de todo esto, pero utilizamos una herramienta que tiene muchos de los beneficios que nos dan las que mencionamos anteriormente: trabajar en grupo de forma colaborativa, comunicar cambios, manejar distintas versiones.

Git

Introducción a Git

Seguramente, más de una vez te paso que tenes un archivo al cual le vas haciendo modificaciones las cuales llamas:

- **Archivo1.txt**
- **Archivo1-final.txt**
- **Archivo1-final-de-verdad.txt**

Esto en programación no es algo posible, ya que para tener un Backup necesitamos poder ir gestionando de manera paulatina las versiones del proyecto y también, poder compartirlas con nuestro equipo.

Y es por esto que se creó GIT... pero, que es?

Git

Introducción a Git

Que es?

Es un software de control de versiones, pensado para mantener eficientemente las actualizaciones sobre el código fuente.

Su propósito es llevar registro del cambio de archivos y coordinar el trabajo que varias personas realizan sobre los mismos.

Git

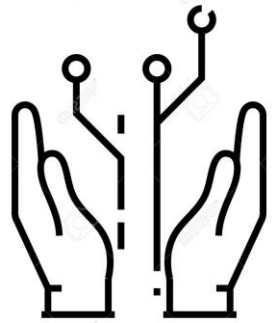
Introducción a Git

Sus Usos:

Con Git vas a poder tener un historial completo de versiones de un mismo archivo sin la necesidad de crear varias copias del mismo.

Pudiendo tener todos los cambios registrados, o incluso volver a la versión inicial cuando lo desees.

Además puedes compartir los archivos de una manera fácil y sencilla, pudiendo hacer seguimiento de los mismos para tener conocimiento sobre, por ejemplo:



- Que integrante del equipo fue el que realizó las modificaciones más recientes.

Git

Introducción a Git

¿Cómo surgió?

GIT fue desarrollado en 2005 por Linus Torvalds, el creador del kernel de Linux.

La historia de su origen está vinculada a las **necesidades de Torvalds y el equipo de desarrollo de Linux para gestionar el creciente y complejo proyecto del kernel de Linux.**

En un principio, utilizaban un sistema propietario llamado **BitKeeper** para manejar el control de versiones, pero cuando el acceso gratuito a BitKeeper fue retirado, surgió la necesidad de crear una alternativa libre y eficiente.

Así, Torvalds **desarrolló GIT para satisfacer las exigencias de velocidad, eficiencia, integridad y soporte para el desarrollo distribuido.**

Linus Torvalds

2. Instalación de Git

Git

Instalación de Git

Luego de tener instalada git bash (en el caso de Windows) procederemos a la instalación de git.

Para esto ingresamos en la página:

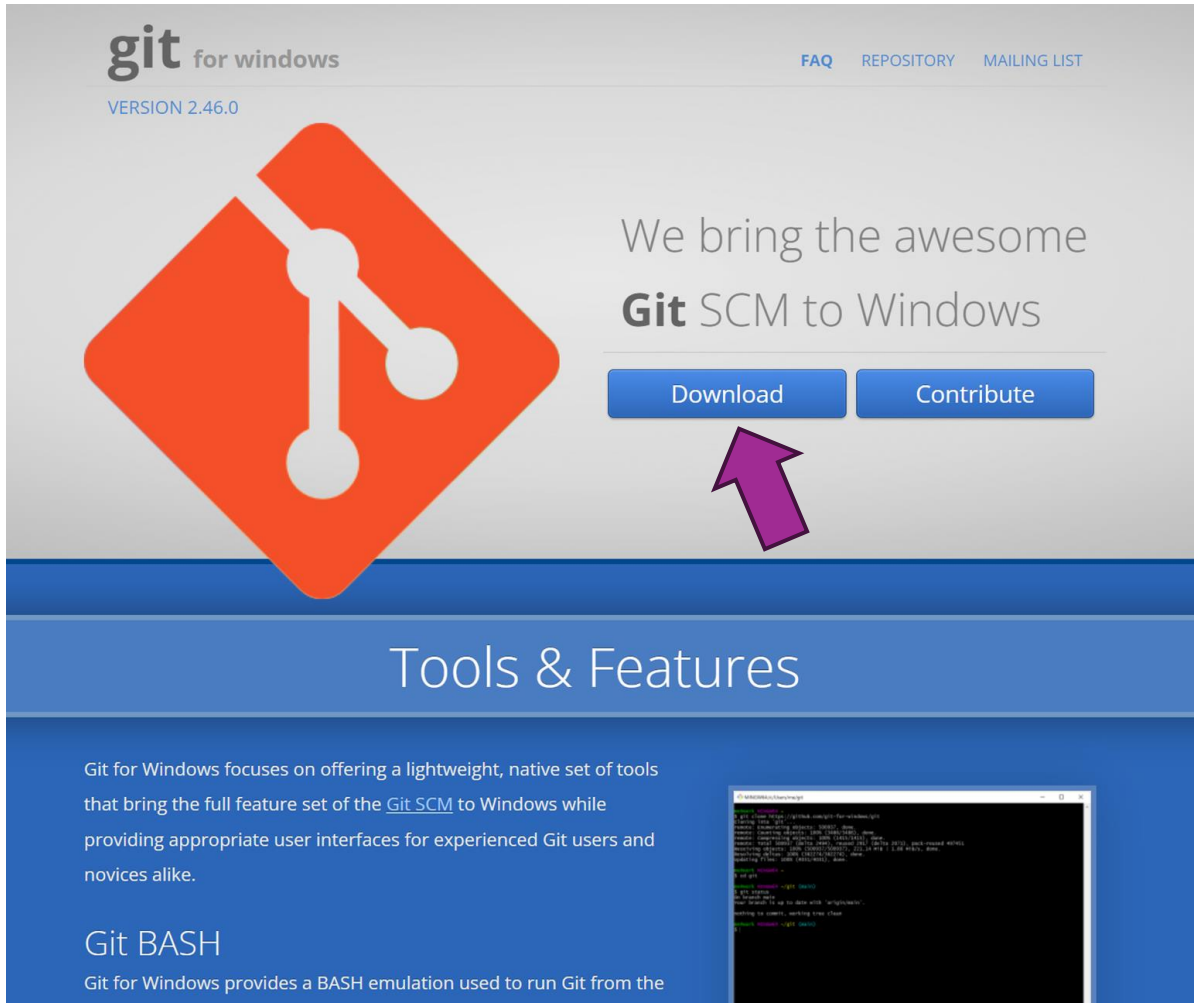
- https://gitforwindows.org

Git

Instalación de Git

En esta página debes dar click en **download** para que se inicie la descarga.

Este programa está diseñado especialmente para los usuarios de windows

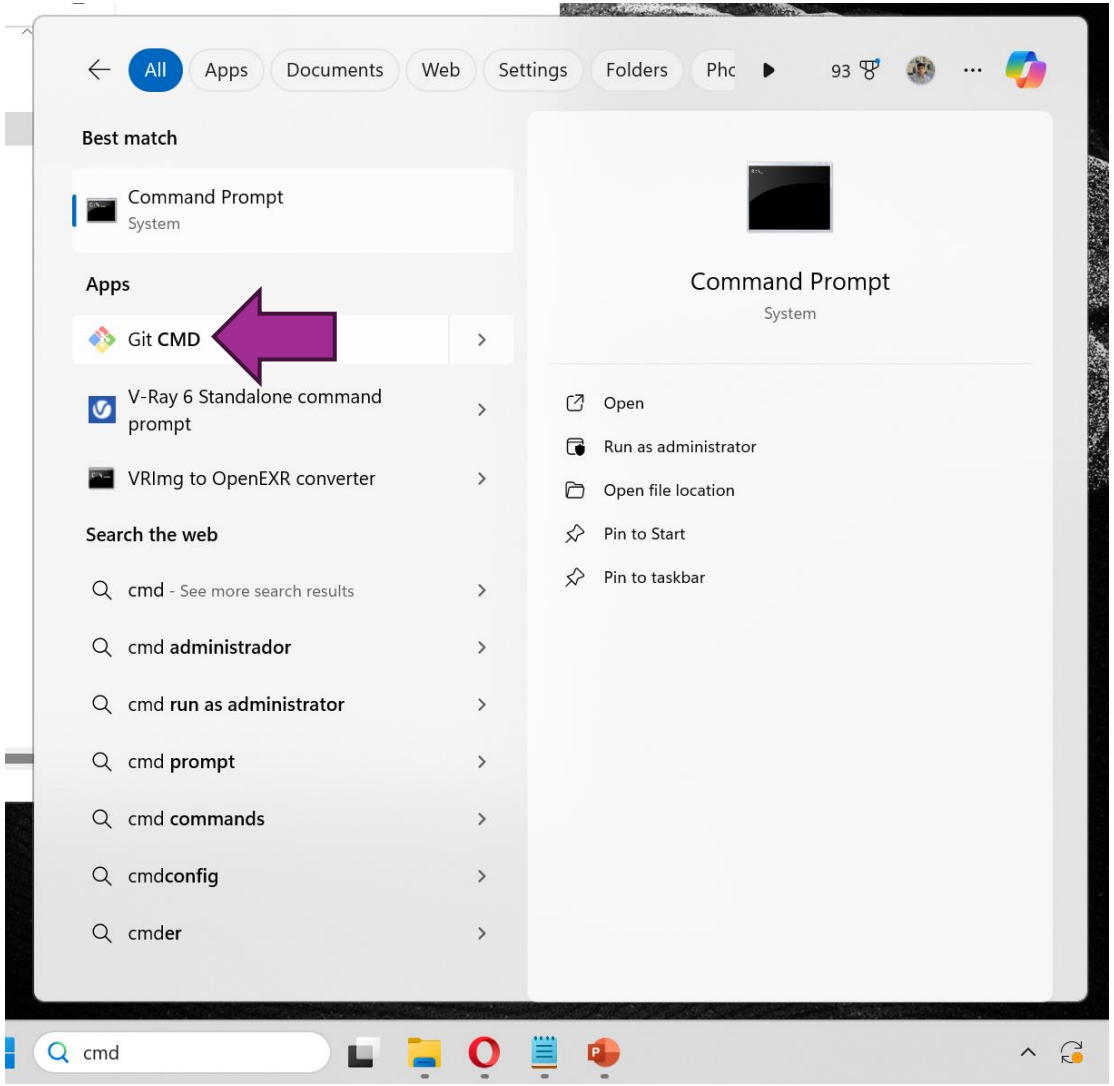


Git

Instalación de Git

Luego de tener instalada git bash (en el caso de Windows) y también el programa de git, pasamos a comprobar que efectivamente tenemos git en nuestra máquina, para eso vamos a la terminal:

- Escribimos cdm en la barra de tareas y abrimos la terminal. (En el caso de Windows abrimos la de git bash)



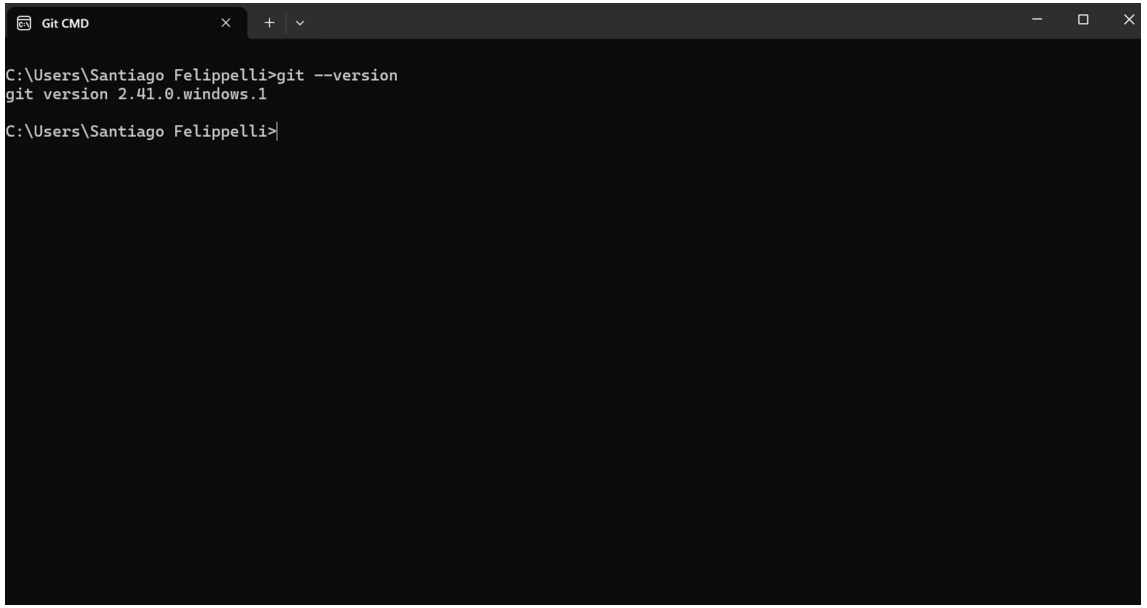
Git

Instalación de Git

- Una vez que se abrió la terminal, tenemos que escribir el siguiente comando:

git - -version

- Si el comando no devuelve un error, quiere decir que instalamos todo satisfactoriamente!



El comando git --version se utiliza **para mostrar la versión actual de GIT instalada en tu sistema.** Cuando lo ejecutas en la terminal o consola, devuelve el número de versión de GIT, lo que es útil para asegurarte de que tienes instalada la versión correcta o para verificar si necesitas actualizar a una versión más reciente.

3. Creando nuestro primer repositorio Local

Git

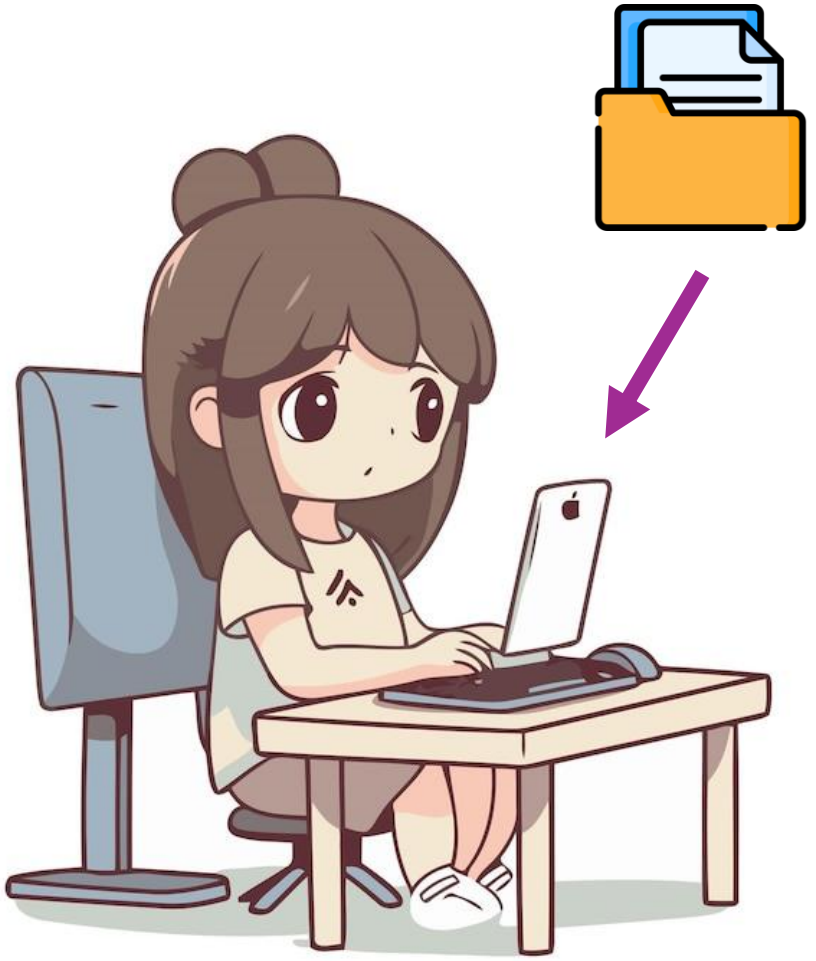
Creando nuestro primer repositorio Local

Imaginemos que tenemos una carpeta con varios archivos de extensión “.js”y por lo tanto queremos hacer un backup de los mismos con git.

Lo primero que debemos hacer es:

- Ir a la terminal
- Ubicarnos en dicha carpeta
- Escribir el comando **git init**
- Al escribir el comando, deberás recibir por parte de la consola un mensaje diciendo que **inicializaste un repositorio vacío** (el texto puede estar en inglés o español)

Pero.. Qué es lo que hizo este comando?



Git

Creando nuestro primer repositorio Local

git init

- Es un comando que inicializa un repositorio local, en la carpeta en la que tenes los archivos.
- Este repositorio local inicialmente está vacío, porque te falta indicar que archivo quieres agregar.

Para entender entonces cómo funciona, comencemos desde el principio...

Git

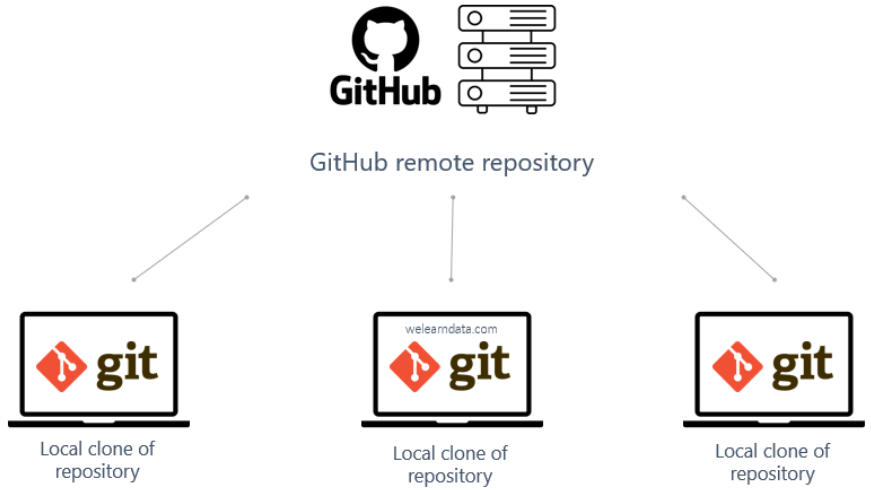
Creando nuestro primer repositorio Local

Repositorio

Cuando hablamos sobre “Repositorio”, podemos hablar de dos tipos:

- **Repositorio LOCAL**: Son los archivos hospedados en mi maquina, es decir, vive en mi computadora.
- **Repositorio REMOTO**: Son los archivos que se hospedan en Github

A continuación vamos a profundizar en el repositorio LOCAL.



Git

Creando nuestro primer repositorio Local

Repositorios
Local -> Vive en tu computadora
Remoto -> Se hospeda en Github

Repositorio

- Un Repositorio, ya sea local o remoto es un **almacén de archivos** (algo así como una estantería) En donde se irán almacenando tus archivos en pequeños paquetes llamados "**commits**"
- Dichos paquetes te permitirán hacer un seguimiento de los cambios que irás realizando, dado que cada uno de ellos tiene:
  - ✓Fecha de creación
  - ✓Un autor
- En otras palabras, los commits van a ser tu **historial de cambios del proyecto**

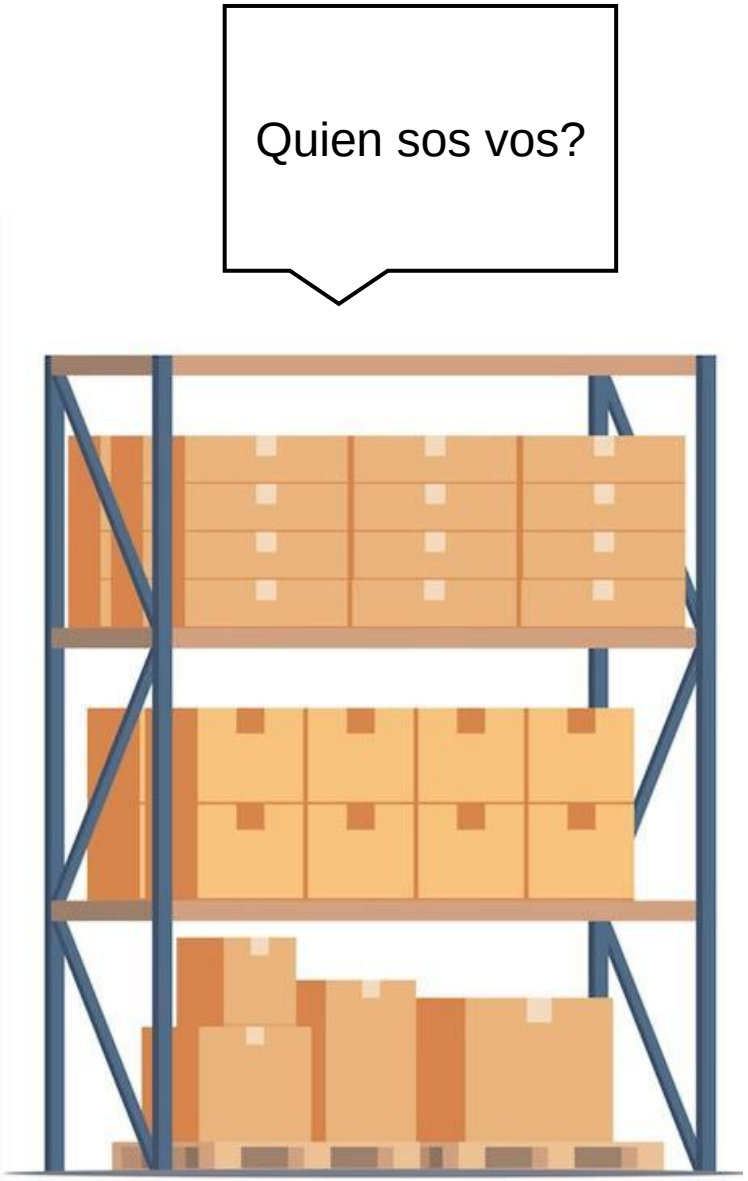
Git

Creando nuestro primer repositorio Local

Repositorio

- Es por esta razón que antes de cualquier cosa, necesitas decirle al repositorio **quien sos vos**, ya que es esta información la que le **permite a git hacer un correcto seguimiento de los cambios realizados.**
- Para **agregar tu identidad al repositorio** deberás escribir **dos comandos de git** (debes escribirlos en la terminal, parada en la carpeta correcta):

1. **git config user.name “mi-usuario”** Con este comando, vas a poder agregar tu nombre de usuario (con el que creaste tu cuenta en github.com) Para saber qué escribiste correctamente la instrucción, puedes cheaquear haciendo git config user.name y en la terminal deberas ver tu nombre de usuario



Git

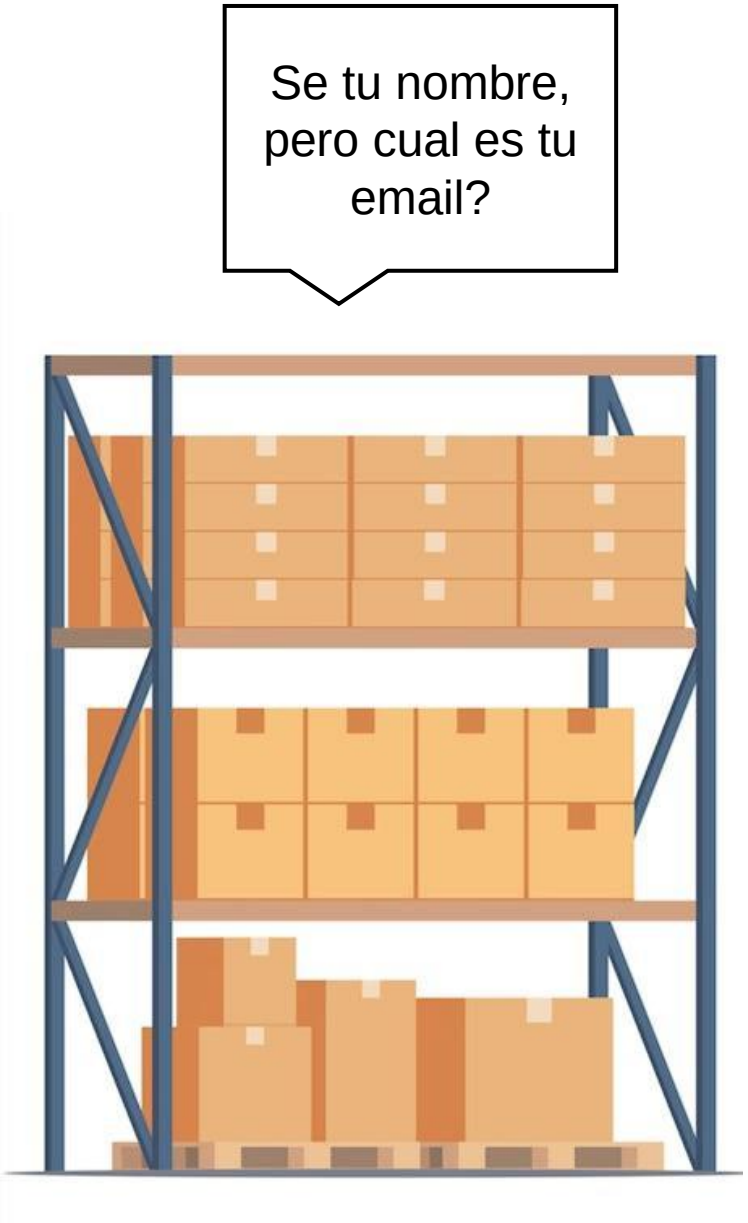
Creando nuestro primer repositorio Local

Repositorio

- Adicionalmente a tu nombre de usuario, el repositorio de git necesita que le dejes saber el correo electrónico con el que te registraste en github.
- Para ello vas a escribir el comando:

2. git congif user.email “miCorreo@email.com”

- Asi como hiciste con el comando anterior para que todo quede perfecto, vas a escribir de nuevo el comando pero sin tu email (**Git config user.email**) y lo que deberías ver en la terminal es el correo que ingresaste antes.
- De esta manera ahora tu repositorio sabe quien sos vos.



Git

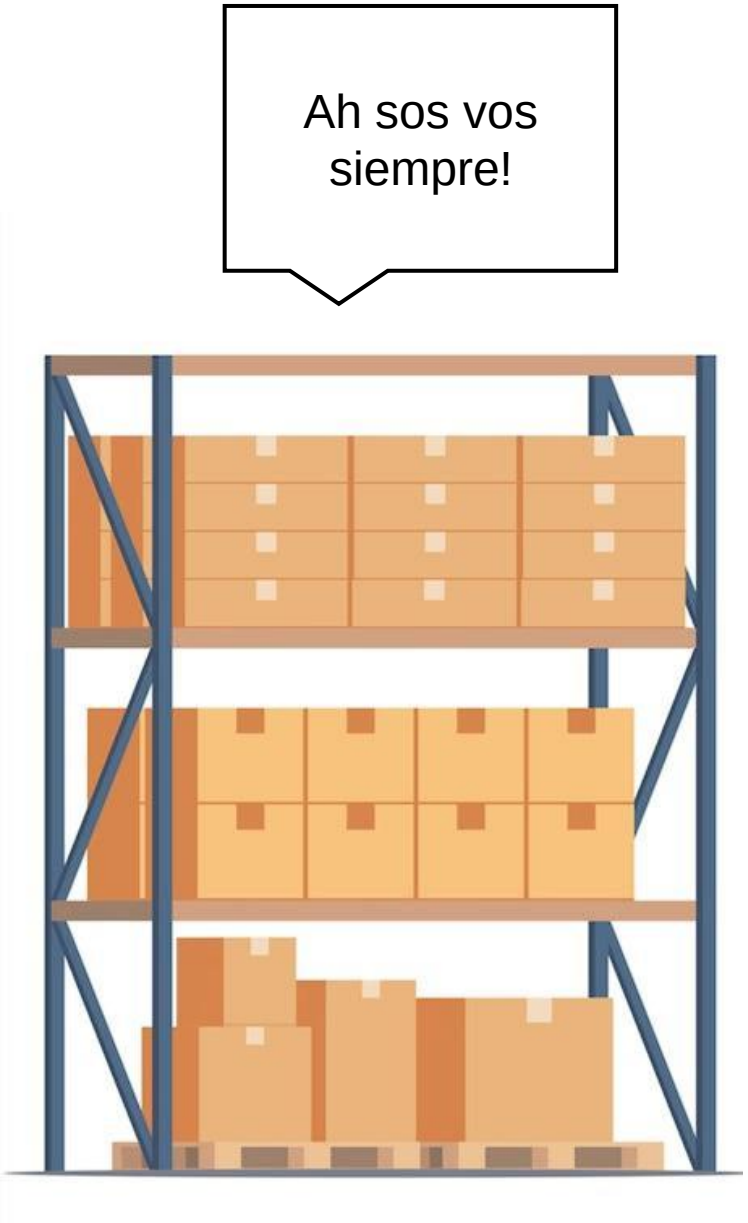
Creando nuestro primer repositorio Local

Repositorio

Si es tu computadora propia y no quieres estar configurando esto a cada rato, puedes hacer los dos comandos como:

- **git config - -global user.name “mi-usuario”**
- **git config - -global user.email “miCorreo@email”**

De esta manera, estarás indicando que cualquier repositorio existente dentro de tu computadora tendrá tu firma de tu nombre y de tu email.



4. Agregando archivos al repositorio local

Git

Agregando archivos al repositorio local

Luego de todo lo visto, seguro te preguntaras, bien como agrego mis archivos al repositorio?

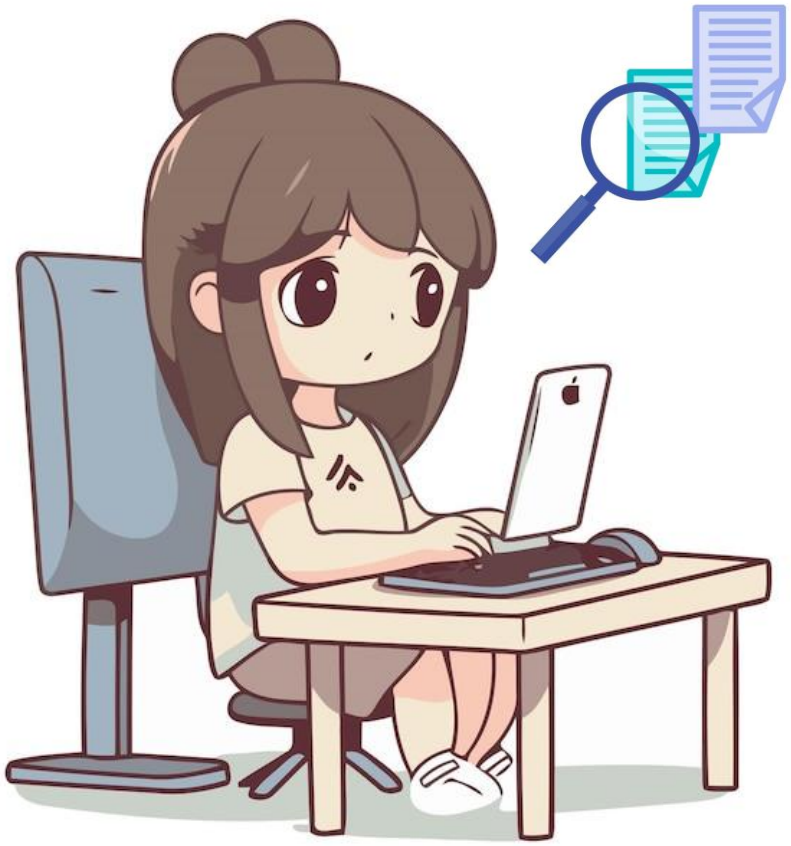
Y no necesariamente todos los archivos, sino los que yo quiero subir.

Git

Agregando archivos al repositorio local

Cuando tienes un **repositorio local ya inicializado** en la carpeta de tu proyecto, el mismo ya es un “**almacén de archivos**” que desde el inicio está vacío, ya que espera que vos le digas cuales son los archivos que quieres agregar.

repositorio local ya inicializado



Git

Agregando archivos al repositorio local

Estos archivos (que pueden ser varios) cada vez que los agregas, se guardan en pequeños paquetes que llamamos “commits”, los cuales tienen consigo:

- Una marca de tiempo
- Firma de un autor

Con estos datos, dejan un historial de los cambios del proyecto.

Git

Agregando archivos al repositorio local

Los Commits, son la base del sistema de git, ya que son los que nos permiten realizar un completo seguimiento y control sobre la versiones de nuestros archivos

Git

Agregando archivos al repositorio local

Pero... sigamos con la pregunta, cómo podemos agregar archivos al repositorio?

Sencillo, como la mayoría de los demás procesos de Git, esto se logra mediante un comando escrito en la terminal.

(Ten en cuenta que solo funciona si estas dentro de una carpeta que tiene un repositorio ya inicializado).

Git

Agregando archivos al repositorio local

El comando a escribir es:

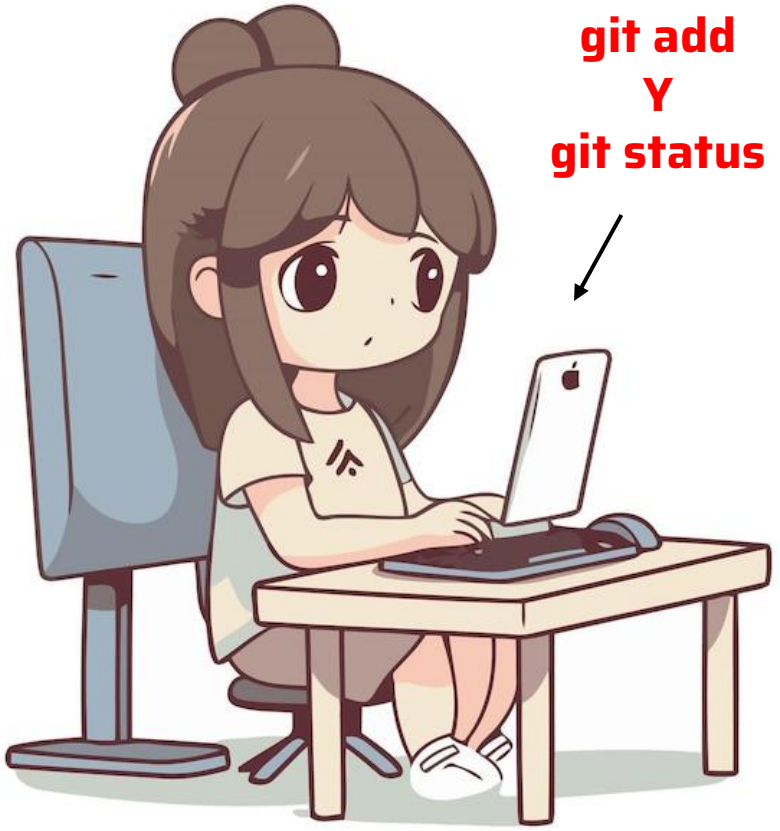
git add

A este, debes indicarle el o los archivos que quieres agregar.

Otro comando que es bueno tener a mano es

git status

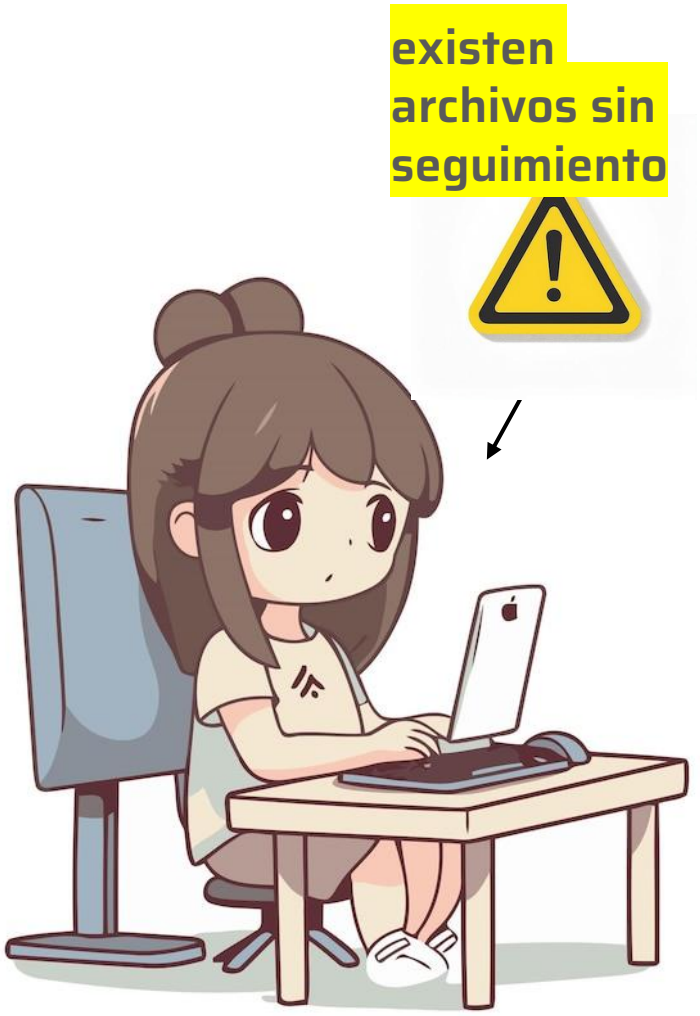
Este sirve para saber el status de nuestros archivos y el estado de los mismos respecto al repositorio.



Git

Agregando archivos al repositorio local

Por ejemplo, si ejecutas “**git status**” inmediatamente **después de inicializar el repositorio** verás en la terminal un mensaje avisándote que “**existen archivos sin seguimiento**”



Esto quiere decir que git se dio cuenta que tenes uno o más archivos que todavía no fueron agregados al repositorio.

Debes tener en cuenta que para Git un archivo sin seguimiento no es necesariamente un archivo nuevo, puede ser un archivo que ya existía dentro del repositorio, pero que sufrió cambios.

Git

Agregando archivos al repositorio local

Vamos ahora a agregar los archivos:

Imaginemos que tenemos tres archivos:

- App.js
- Funciones.js
- Usuarios.js

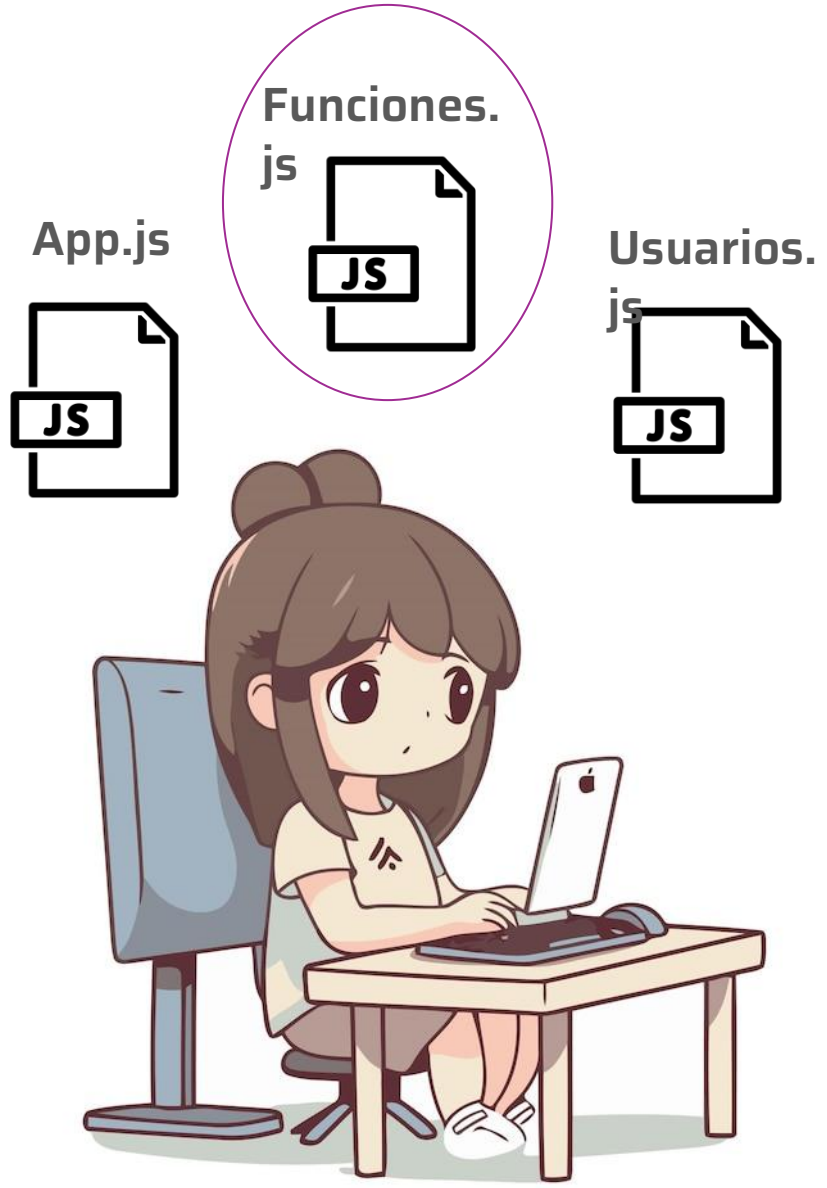
Pero deseas agregar al repositorio el archivo de funciones.js y ninguno mas.

Para esto, tendrás que escribir en la terminal:

git add funciones.js

Si esta bien ingresado el comando, no debería haber errores.

Si luego de esto hacemos “**git status**” deberías ver un mensaje que diga que el archivo funciones.js fue agregado, pero que los otros dos están sin seguimiento.



Git

Agregando archivos al repositorio local

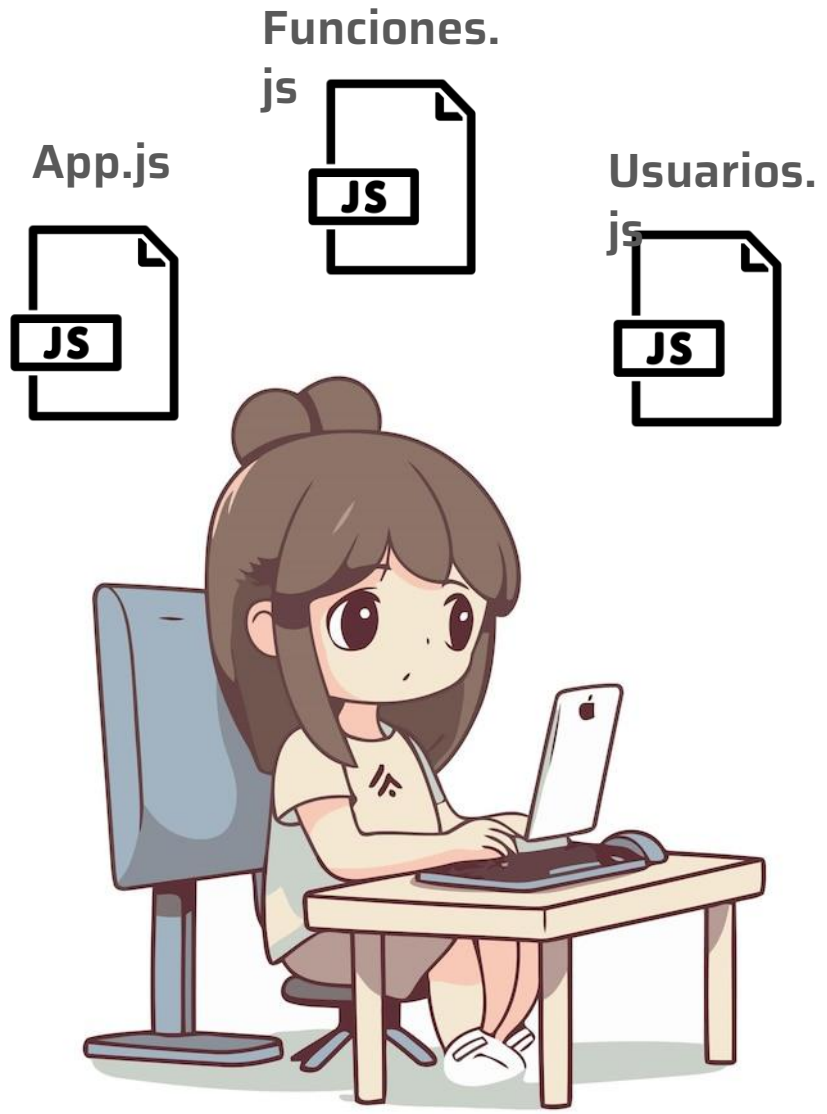
Entonces... tenemos que agregar uno a uno los archivos? Eso nos llevaría mucho tiempo!

No se preocupen, podemos abreviar el proceso haciendo uso del mismo comando, pero en vez de escribir el nombre de un archivo en especifico, vas a escribir un punto:

Git add .

El punto simboliza que deseas agregar todos los archivos presentes en el repositorio, es decir, archivos nuevos y también archivos que hayan sufrido cambios.

Si hacemos esto y luego escribimos "git status", vas a ver que ahora todos los archivos han sido agregados al repositorio con éxito y que no quedo ninguno sin seguimiento.



Git

Agregando archivos al repositorio local

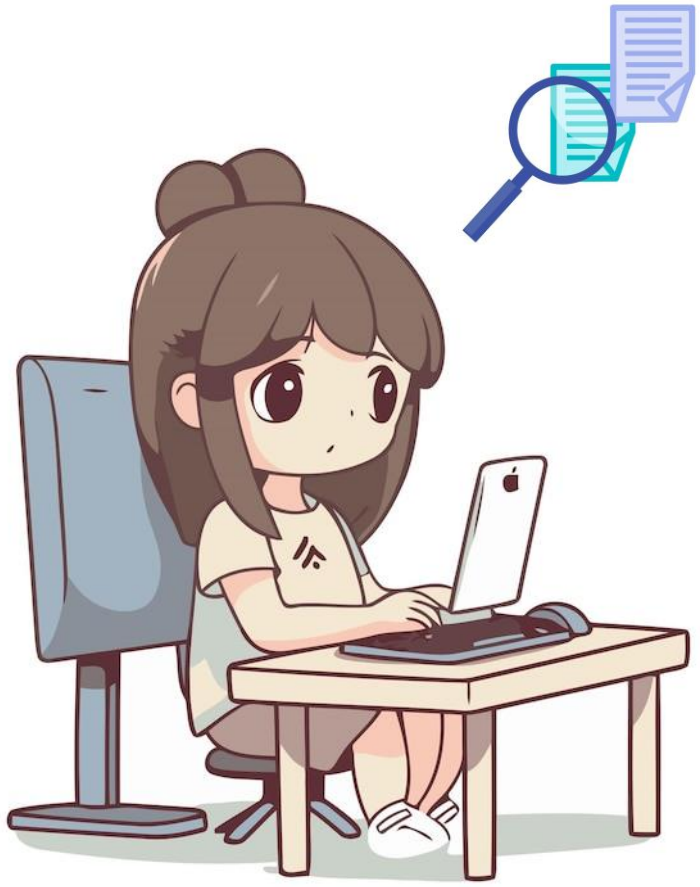
Genial, ya vimos como subir los archivos pero... Que sucede cuando un archivo que ya agregaste al repositorio sufre alguna modificación?

Cuando pasa esto con cualquier archivo, git asume que ese archivo es “**nuevo**” y por lo tanto **pasará a ser nuevamente un archivo que no tiene seguimiento.**

Porque sucede esto? Porque esto es justamente el famoso “control de versiones”. **Para git cada cosa nueva que suceda sobre un archivo, es una indicación de cambios efectuados.**

En este caso lo único que tienes que hacer para que git agregue ese archivo (junto con sus cambios) es ejecutar de nuevo **git add** (con el nombre del archivo o con el “.” en caso de que modificaras varios archivos)

control de versiones

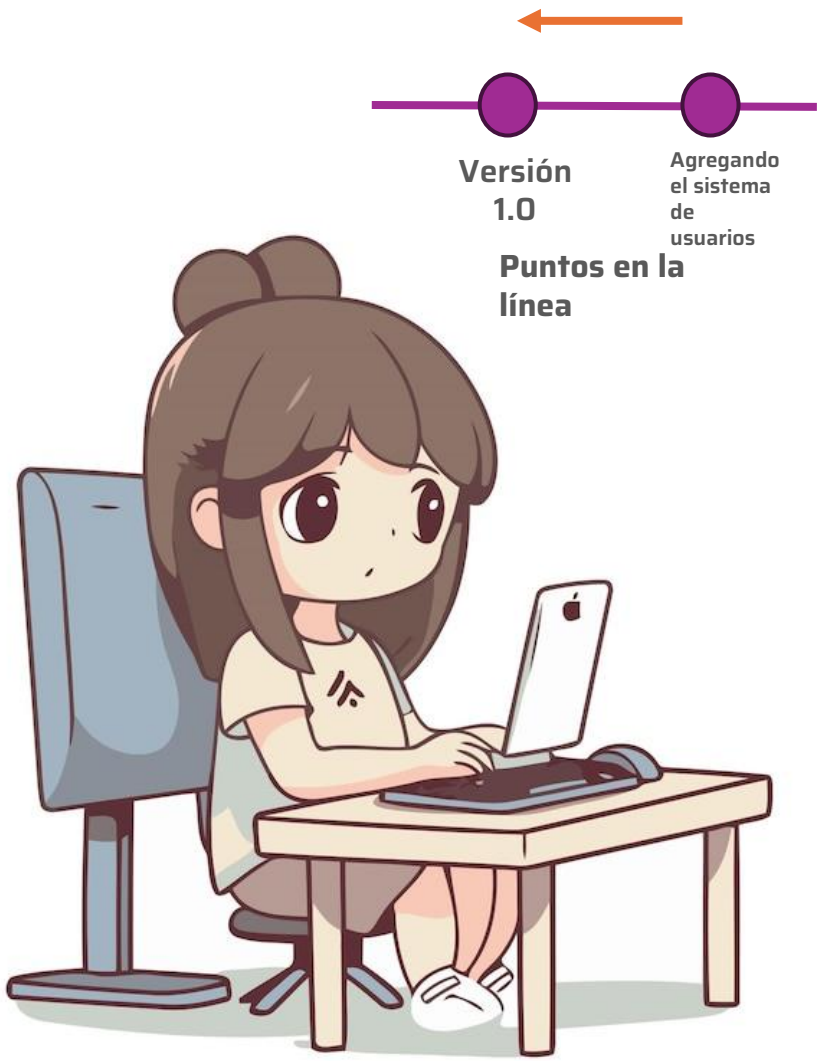


5. Confirmando archivos

Git

Confirmando archivos

Llegó el momento en el que oficialmente vamos a confirmar que los archivos agregados al repositorio serán un punto en la línea de tiempo a la que podrás volver, si es que así lo necesitas.



Git

Confirmando archivos

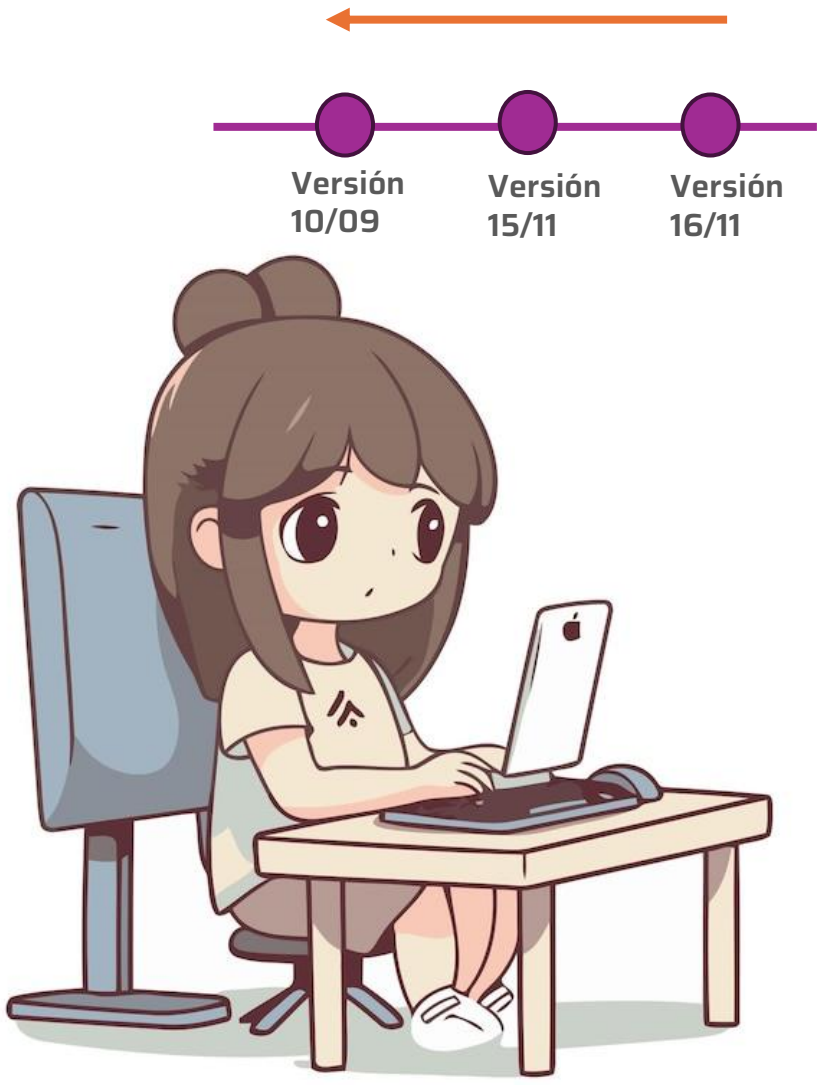
Vamos a comenzar a crear “commits”.

Un commit, es una **confirmación** a través de la cual le estamos **diciendo al repositorio que los archivos que fuimos agregando** (que pueden ser varios) **los deseamos oficialmente como un pequeño paquete de adiciones o modificaciones, que tendrán a su vez una *marca indeleble de tiempo* y estarán *firmados por un autor.***

Git

Confirmando archivos

Los commits generan puntos cronológicos en la línea del tiempo del proyecto, que nos permiten identificar el estado del mismo hasta ese momento específico y a su vez, volver sobre los mismos, si es que así lo quisiéramos en algún momento determinado.



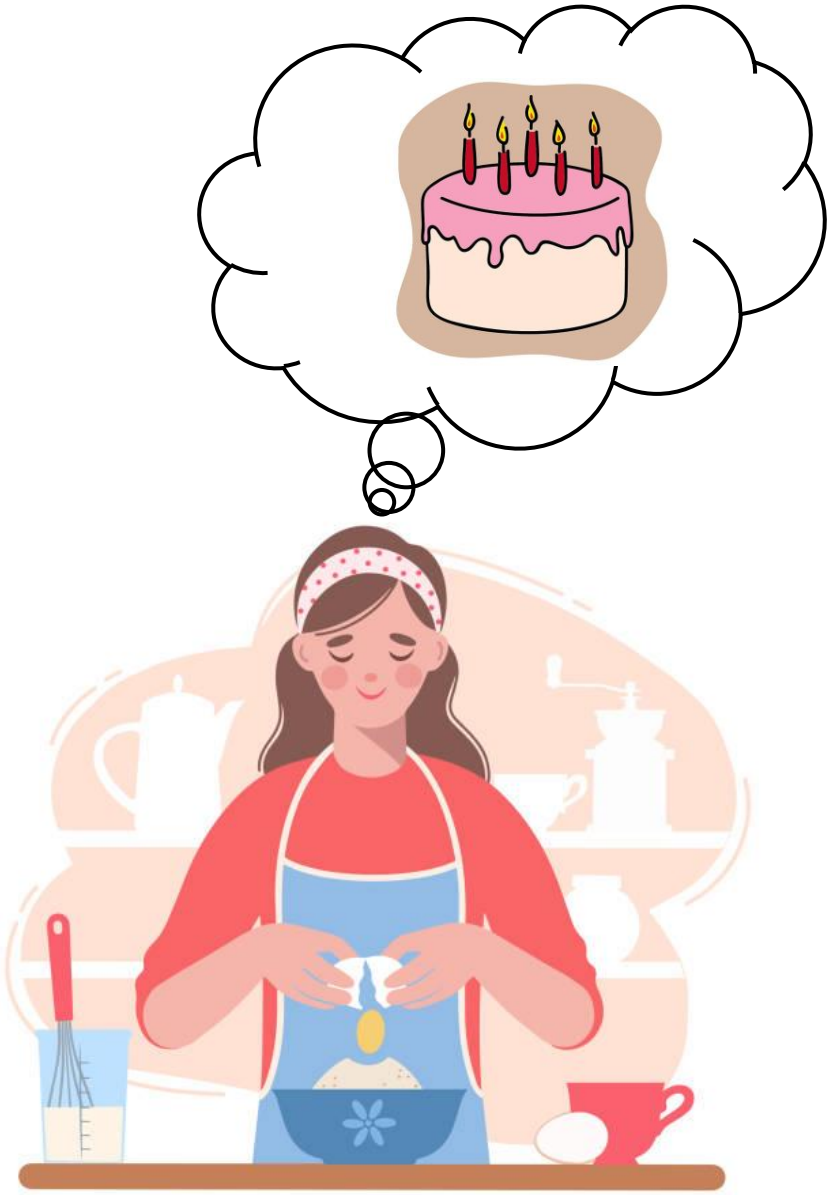
Git

Confirmando archivos

Imagina que estás preparando la masa para una **torta**.

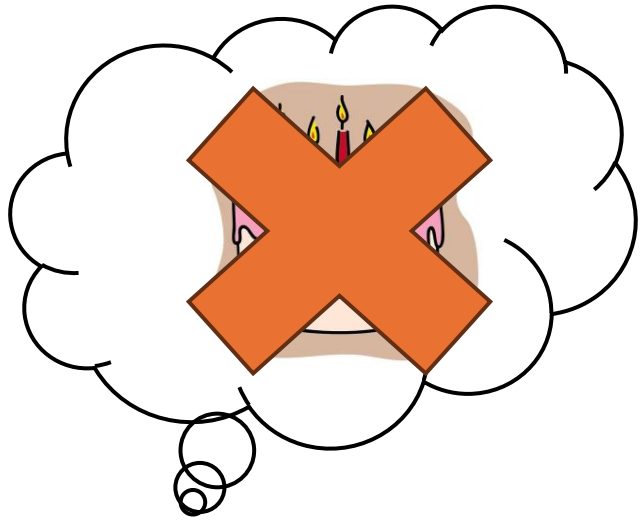
Lo primero que harías, una vez teniendo los ingredientes listos, es mezclarlos en un bowl.

Ósea que: agregamos la harina, luego los huevos y sin querer cometes el error de agregar aceite en vez de manteca.



Git

Confirmando archivos



En la vida real, este error nos llevaría a tener que tirar a la basura la mezcla y volver a empezar.

Pero con git esto no es del todo así.

Cada vez que estamos en un estado importante, podemos crear un commit.

Git

Confirmando archivos

Por ejemplo podemos crear un commit de: “mezcla terminada”

Luego otro que tenga por nombre: “mezcla en el horno”.

Después tendríamos otro que se llame “torta horneada”.

Y finalmente uno que se llame: “torta decorada”.

- **Mezcla terminada**
- **Mezcla en el Horno**
- **Torta horneada**

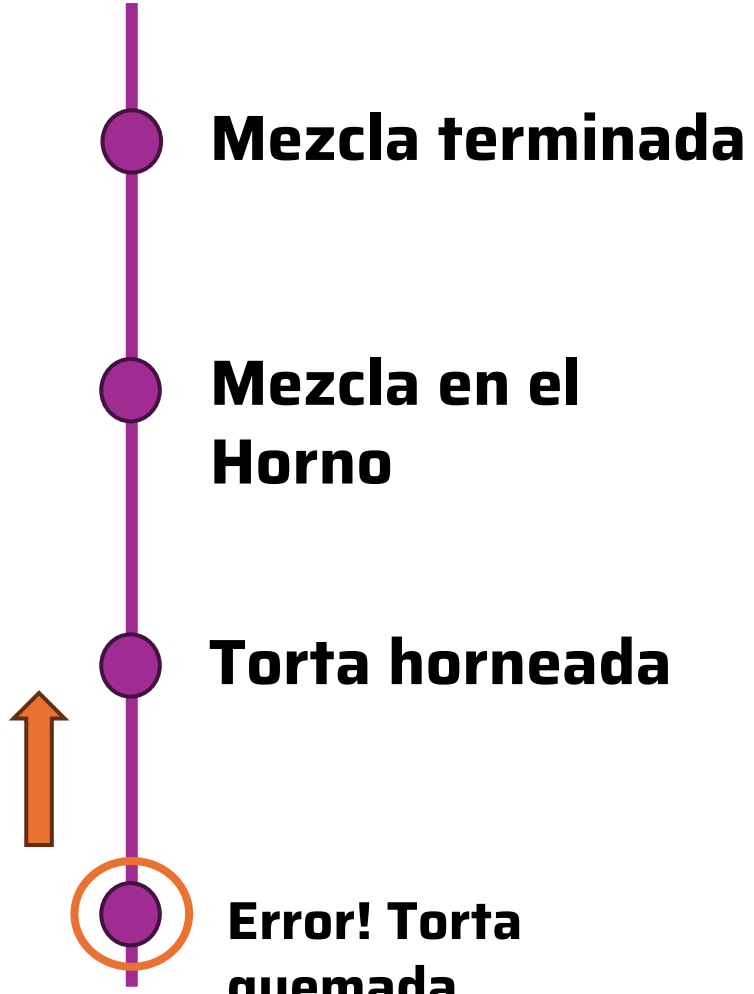
Git

Confirmando archivos

Pero si cometimos un error, como por ejemplo “se nos quemó la torta”

Podemos volver con mucha facilidad a la versión anterior.

Básicamente, es como poder viajar en el tiempo y solucionar los errores no previstos!



Git

Confirmando archivos

Para poder crear un commit, tenes que haber agregado previamente los archivos modificados al repositorio y luego escribir el comando:

git commit –m “mensaje”

El mensaje puede ser cualquier cosa, pero la idea es que con el mensaje describas de manera resumida el trabajo hecho hasta ese momento.

Git

Confirmando archivos

Pensa que arrancaste tu proyecto y en el mismo tienes dos archivos:

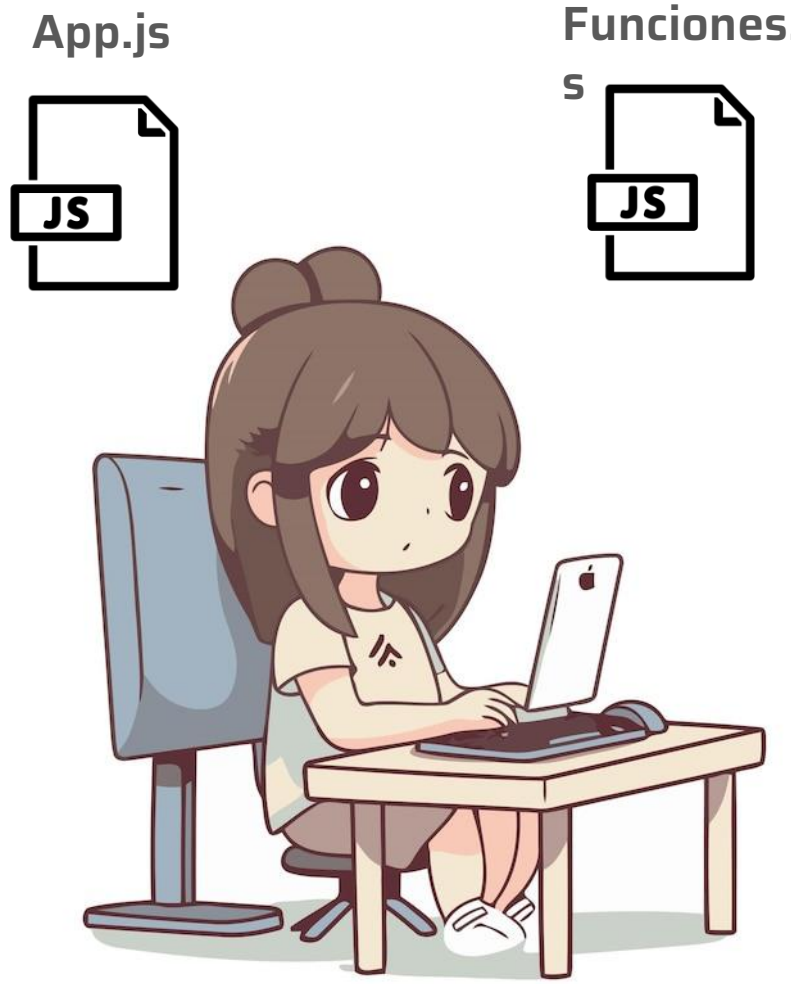
- App.js
- Funciones.js

Entonces seguramente, después de iniciar un repositorio y de agregarlos vas a querer commitearlos.

Para ello, ejecutaremos el comando:

git commit –m “primer commit del proyecto”

Commitear = Hacer un commit



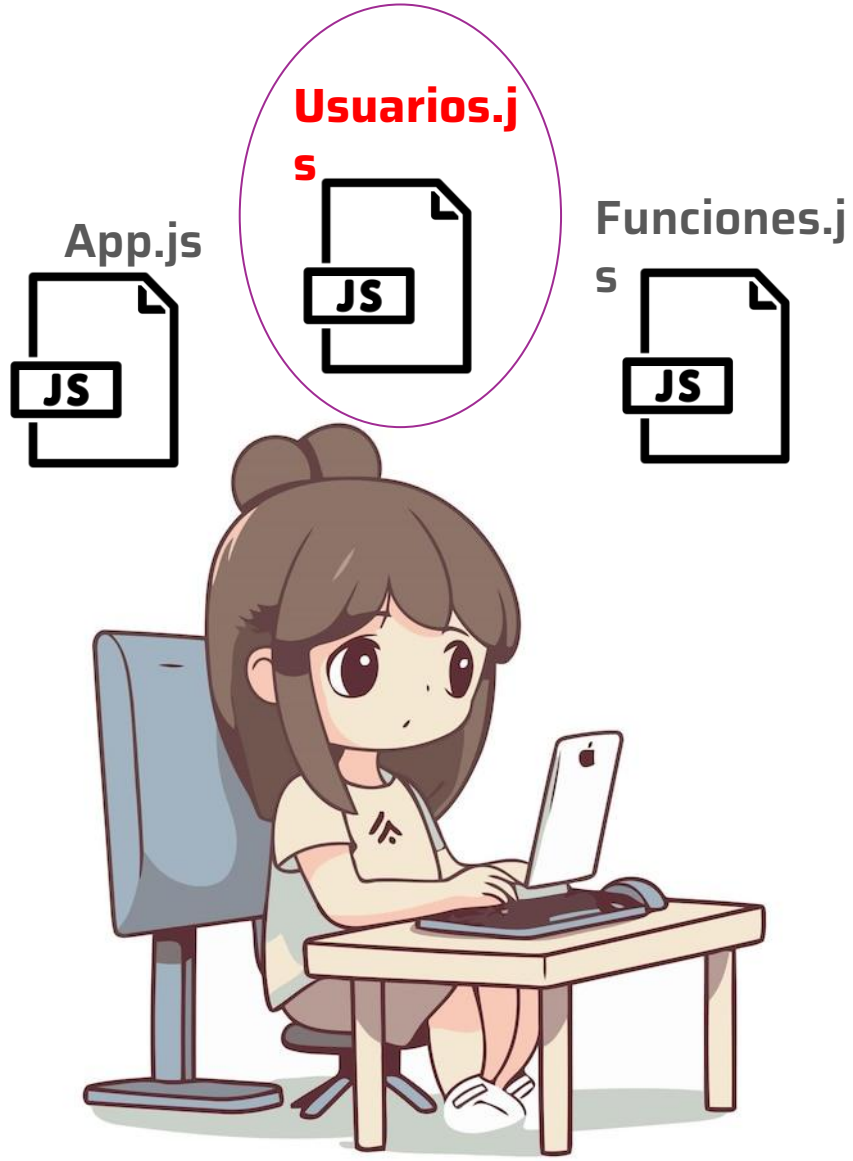
Git

Confirmando archivos

Con esto generas un **punto en la línea del tiempo de tu proyecto** el cual tiene los archivos que hasta el momento haz agregado al repositorio.

Pero... qué sucede si agregas un archivo nuevo? Por ejemplo, digamos que adicionas el archivo “usuarios.js”.

Seguramente si hacemos “**git status**” verás en la terminal el mensaje de que sobre este ultimo archivo no se está haciendo seguimiento.



Git

Confirmando archivos

Por lo tanto, lo que tenes que hacer es agregar el archivo con el comando:

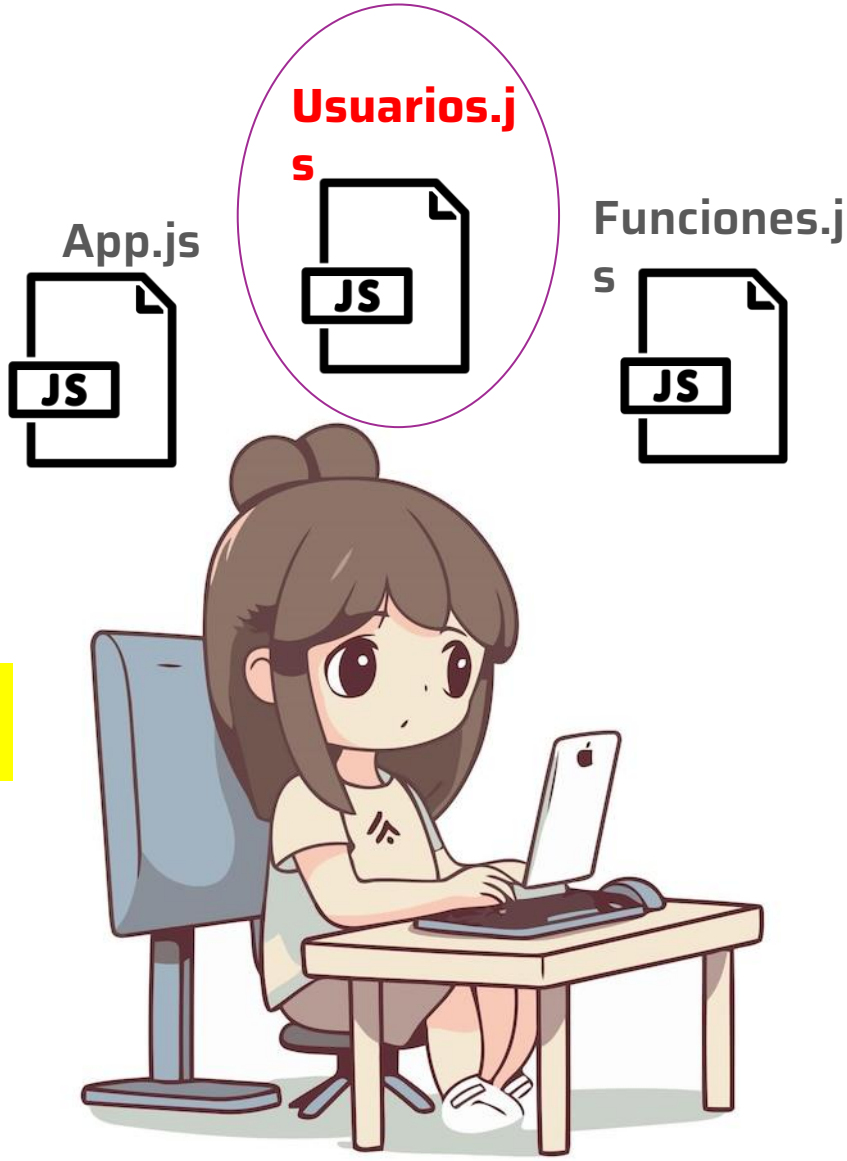
Git add

Y posteriormente volver a realizar un nuevo **commit.**

Es decir, en un commit, entran los archivos en los que hayas hecho previamente “git add”

Esta vez el commit podría tener un mensaje:

git commit – m “agrego el archivo usuarios.js”



Git

Confirmando archivos

En esta instancia, ahora tu proyecto tiene 2 commits:

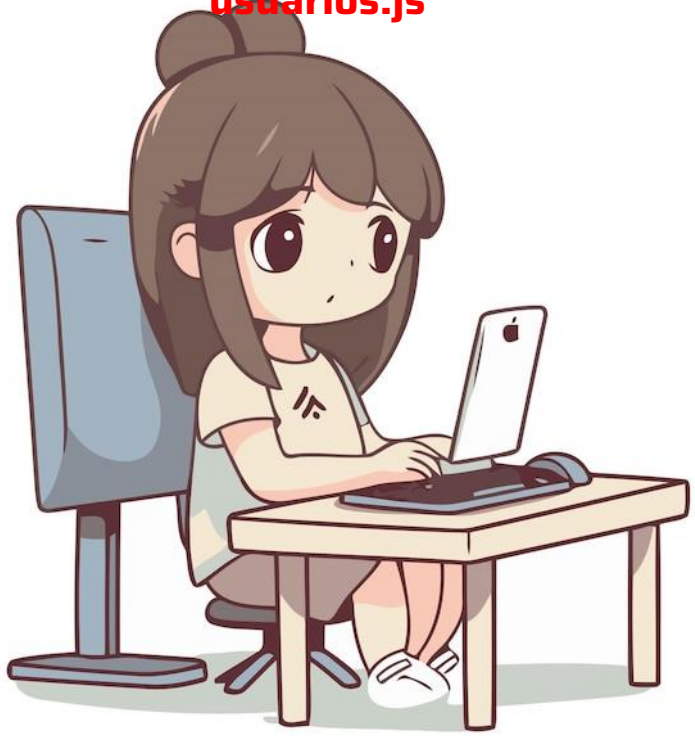
- El creado inicialmente.
- Y este último que acabas de generar.

Como veras, el proceso se vuelve repetitivos, es decir, cada vez que creas un archivo lo agregas y posteriormente lo commiteas. Y así, tantas veces como archivos nuevos vayas generando

2 COMMITS

1. **git commit –m “primer commit del proyecto”**

1. **git commit – m “agrego el archivo usuarios.js”**



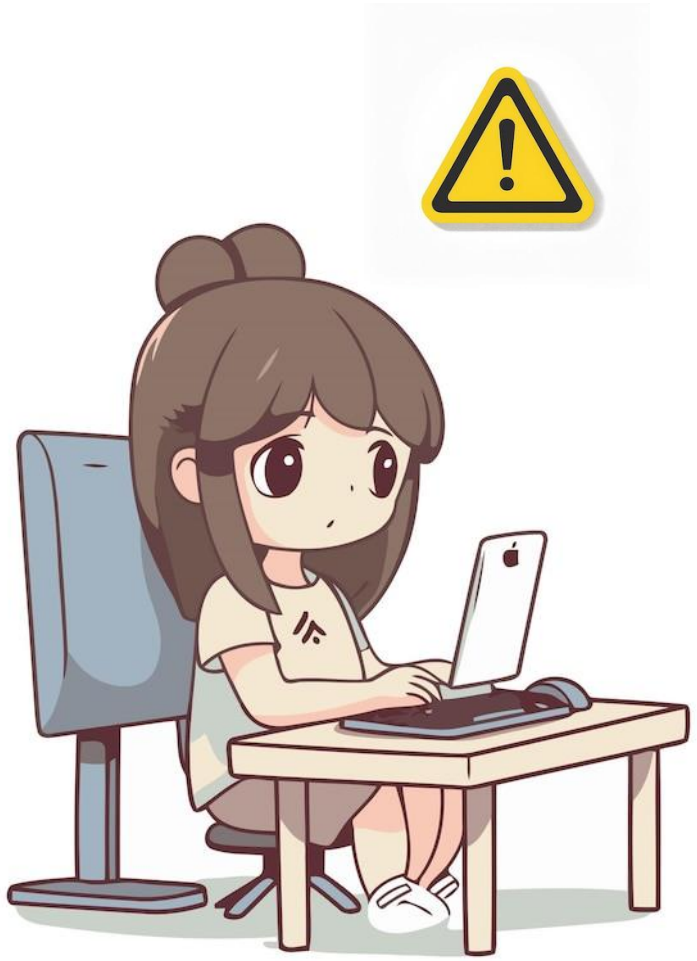
Git

Confirmando archivos

Pero… Qué sucedería si ahora es un archivo que ya existe dentro del repositorio el que cambia?

Si ejecutas a esta altura un “**git status**” verías un mensaje de alerta de que dicho archivo ha sido modificado, el cual si bien ya es parte del repositorio, es necesario volver a agregar con **git add**, para actualizar el mismo en el repositorio y, posteriormente, crear un commit para confirmar que dicho cambio lo queres oficialmente con un punto en la línea del tiempo del proyecto.

El archivo a sido



Git

Confirmando archivos

De esta manera podes ir actualizando el estado de tu proyecto, sin perder ningún archivo que vayas agregando al mismo.

Así de fácil podemos tener todos nuestros archivos ordenados y seguros!

Nos falta ver el otro lado de git, pero tranquilas, que eso estará en nuestra próxima clase!

Momento de poner a prueba lo aprendido!