

# GIT Básico: Curso Práctico para Principiantes

## Introducción a GIT y VCS

### SISTEMAS DE CONTROL DE VERSIONES

Un sistema de control de versiones (VCS por sus siglas en inglés) es un sistema que registra los cambios realizados en un archivo o conjunto de archivos a lo largo del tiempo, de modo que es posible recuperar versiones específicas en otro momento del desarrollo de un proyecto de software. Algunas de sus mayores ventajas son poder regresar a una versión anterior del proyecto  completo, comparar cambios entre distintas versiones, ver quien realizó modificaciones al proyecto, permitiendo reconocer cambios que puedan generar conflicto, trabajar sobre proyectos de otros desarrolladores y compartir los propios proyectos.

Un problema frecuente al desarrollar proyectos de software ocurre al copiar los archivos de un proyecto en una dirección local diferente y seguir el desarrollo desde allí, está práctica por común y sencilla que parezca es propensa a errores ya que es fácil olvidar en que ruta se
está trabajando y guardar cambios en la ruta equivocada, o sobre escribir archivos que no se debían modificar. Para afrontar este problema se desarrollaron los **VCS locales**, los cuales contienen una base de datos, en la que se lleva registro de todos los cambios realizados a los archivos.

Otro problema
recurrente al desarrollar proyectos de software es la necesidad de colaborar con desarrolladores en otros sistemas, por tal motivo, los VCS generalmente tienen un único servidor que contiene todos los

archivos versionados y varios clientes que descargan los archivos desde ese lugar central, este tipo de VCS son llamados **de tipo centralizado** (**CVCS** por sus siglas en inglés)

Otro tipo de VCS son los **de tipo distribuido** (**DVCS** por sus siglas en inglés), los VCS más populares, entre ellos Git, son de este tipo, se caracterizan porque el cliente no solamente descarga la última copia de los archivos, sino que se replica completamente el repositorio, de esta forma, si un servidor deja de funcionar y varios desarrolladores estaban colaborando a través de él, cualquiera de los repositorios disponibles en los clientes puede ser copiado al servidor con el fin de restaurarlo, de esta manera, cada clon es una copia completa de todos los datos.

Todos los sistemas de control de versiones proporcionan mecanismos de almacenamiento de los elementos que debe gestionar, la posibilidad de realizar cambios sobre los elementos almacenados, sean estas parciales, o incluso remover archivos. Así mismo, se cuenta con un registro histórico de las acciones realizadas con cada elemento o conjunto de elementos, pudiendo volver a un estado anterior del producto. Algunos de los términos más empleados al usar los VCS son:

- Repositorio: Lugar en el que se almacenan los datos actualizados e históricos de cambios, a menudo en un servidor. También se le llama depósito o repo.
- Módulo: Conjunto de directorios y/o archivos dentro de un repositorio que pertenecen a un proyecto común.
- Revisión: versión determinada de la información que se gestiona, algunos sistemas identifican dichas versiones con un contador, otros sistemas las identifican mediante un código de detección de modificaciones, tal como es el caso de Git, a la última versión se le suele identificar con el nombre **HEAD**.
- Rotular o tag: Consiste en darle a alguna versión de cada uno de los ficheros del módulo en desarrollo en un momento dado un nombre común, de forma que se pueda identificar rápidamente ese estado de desarrollo posteriormente bajo ese nombre.

- Línea base: Es una revisión aprobada de un documento o fichero fuente, a partir del cual se pueden realizar cambios subsiguientes.
- Desplegar o "Check-out": En un despliegue se crea una copia de trabajo local desde el repositorio, pudiéndose especificar una revisión concreta y predeterminadamente se suele obtener la última.
- Commit: Un commit ocurre cuando una copia de los cambios hechos a una copia local es escrita o integrada sobre el repositorio.
- Conflicto: Ocurre cuando el sistema no puede manejar adecuadamente cambios realizados por dos o más usuarios en un mismo archivo.
- Resolver: Es el acto de intervención del usuario para atender un conflicto entre diferentes cambios al mismo archivo.
- Exportación: Es la acción de copiar un árbol de directorios local, que no es en ese momento una copia de trabajo, en el repositorio por primera vez.
- Integración o "merge": Proceso por el que se unen dos conjuntos de cambios sobre un fichero o un conjunto de ficheros.
- Actualización, "sync" o "update": Integra los cambios que han sido hechos en el repositorio en la copia de trabajo local.
- Copia de trabajo o "workspace": Es la copia local de los ficheros de un repositorio, en un momento del tiempo o revisión específicos.
- Congelar: Significa permitir los últimos cambios para solucionar las fallas a resolver en una entrega y suspender cualquier otro cambio antes de una entrega, con el fin de obtener una versión consistente.
- Ramas o "branches": Herramienta de los VCS que permite abrir nuevas líneas de desarrollo de un proyecto sin alterar los archivos contenidos en otra rama del proyecto, de esta manera en un proyecto puede darse una rama "beta", en la cual se implementen cambios de nuevos desarrollos o funcionalidades, mientras que en otra rama "alfa" se conserve la versión principal del proyecto.

# GIT

Es un VCS de
software libre, diseñado por Linus Torvalds, pensado en la

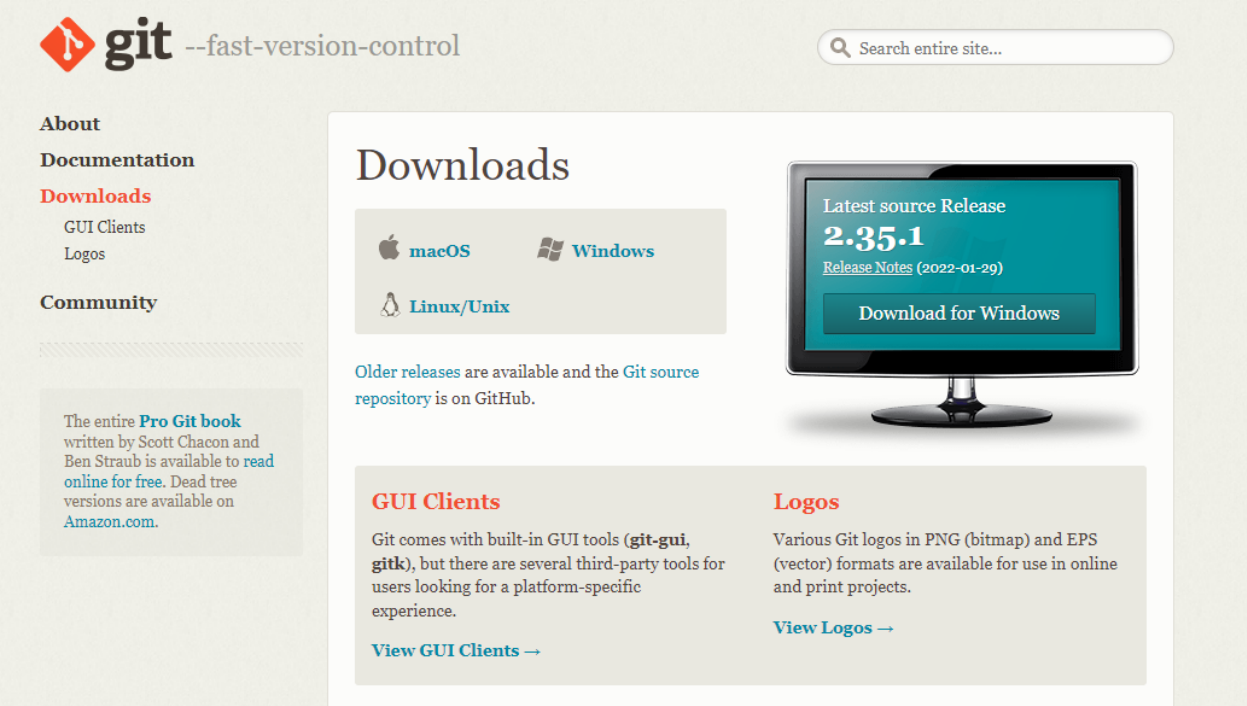
eficiencia, confiabilidad y compatibilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente. Su propósito es llevar registro de los cambios en archivos de computadora incluyendo coordinar el trabajo que varias personas realizan sobre archivos compartidos en un repositorio de código.

Algunas de sus
principales características son:

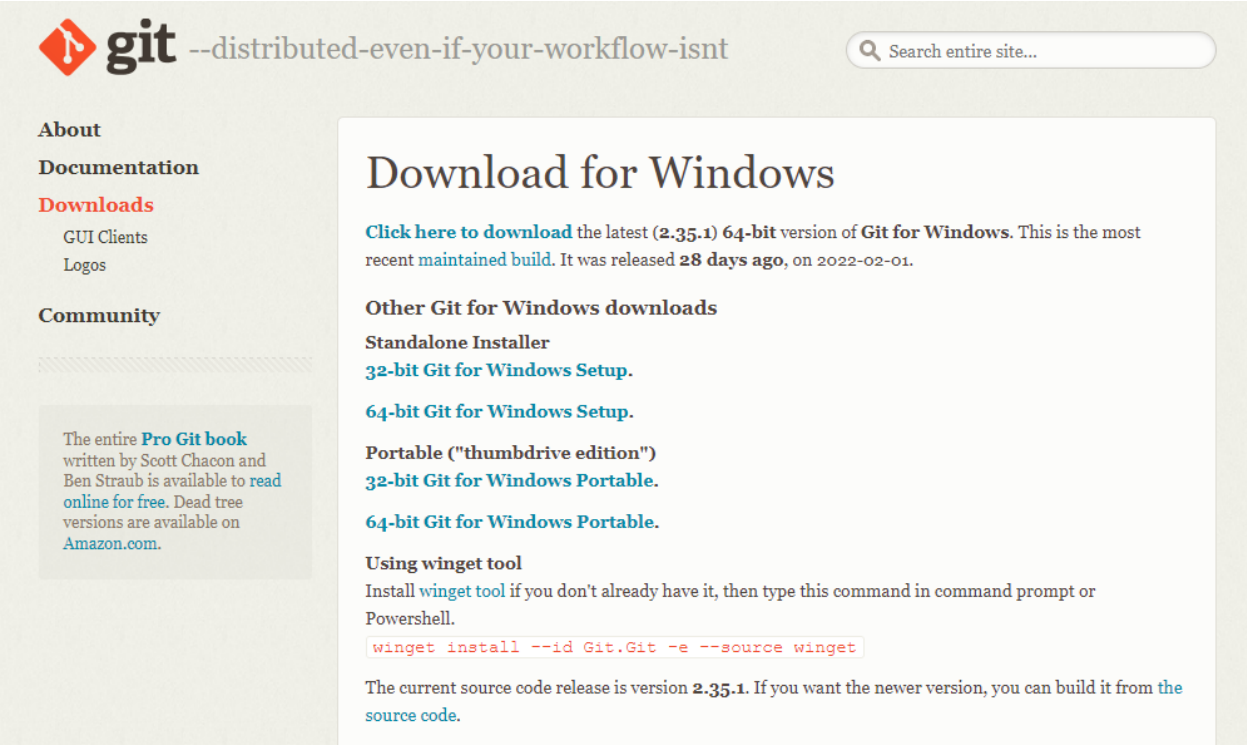
- Fuerte apoyo al desarrollo no lineal.
- Rapidez en la gestión de ramas y mezclado de diferentes versiones.
- Gestión distribuida, cada programador tiene una copia local del historial del desarrollo entero y los cambios se propagan entre los repositorios locales, de forma que los cambios se importan como ramas adicionales, pudiendo ser fusionados con los archivos en otras ramas.
- Gestión eficiente de proyectos grandes, dada la rapidez de gestión de diferencias entre archivos.
- Compatibilidad con Github y Microsoft Visual Estudio Code.

# Instalación GIT en Windows

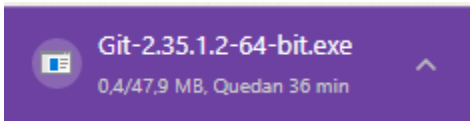
Se ingresa al sitio oficial de Git escribiendo la siguiente dirección en el navegador https://gitscm.com/downloads, dónde nos muestra la siguiente ventana.



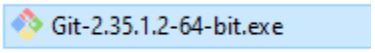

En ésta ventana.  Se debe seleccionar el sistema operativo y la versión para Windows si es de 32 o 64 bits, se presiona click en descargar el instalador está de color azul.



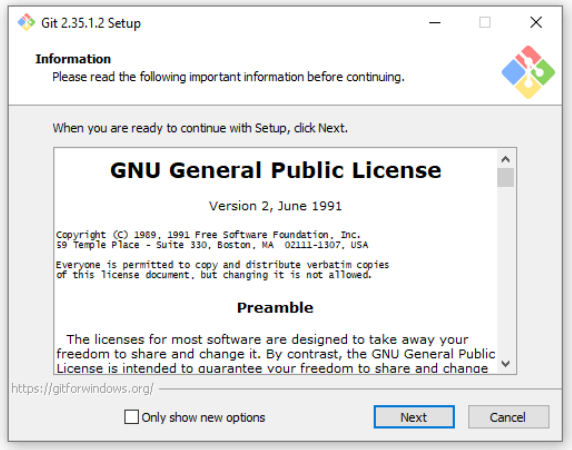
Esperamos que se descargue el instalador.

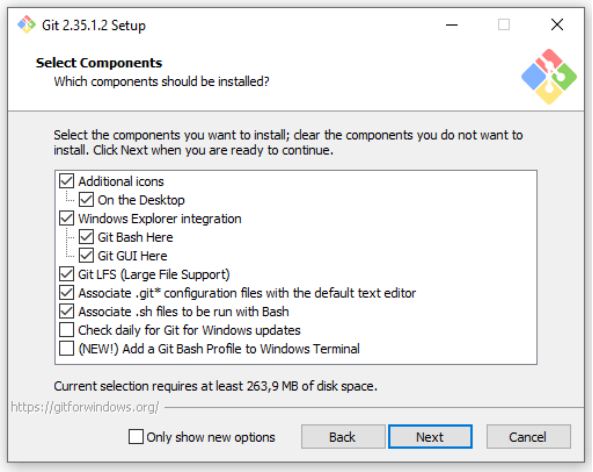


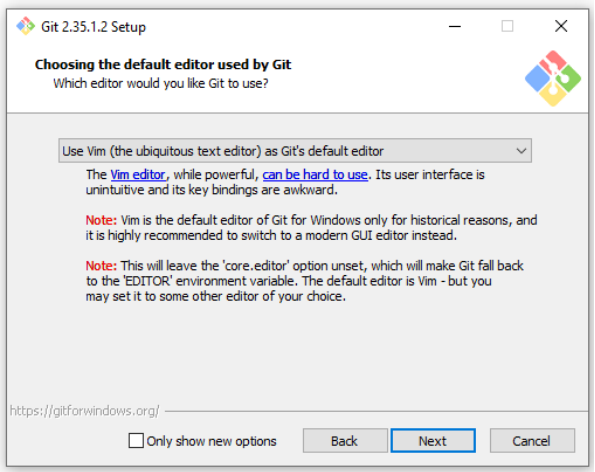

Para instalar el Git se debe dar doble click sobre el instalador

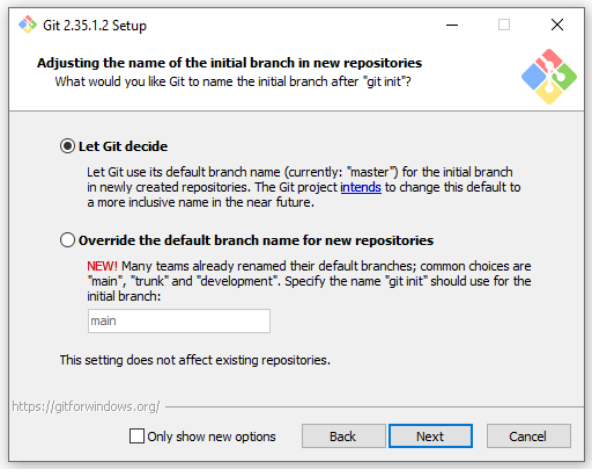


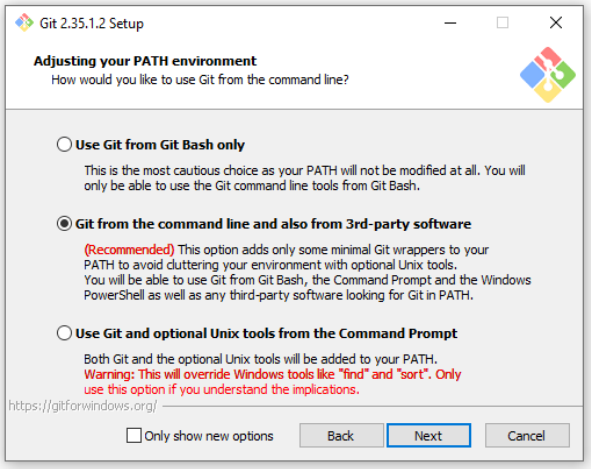

En las siguientes ventanas se presiona click en el botón Next

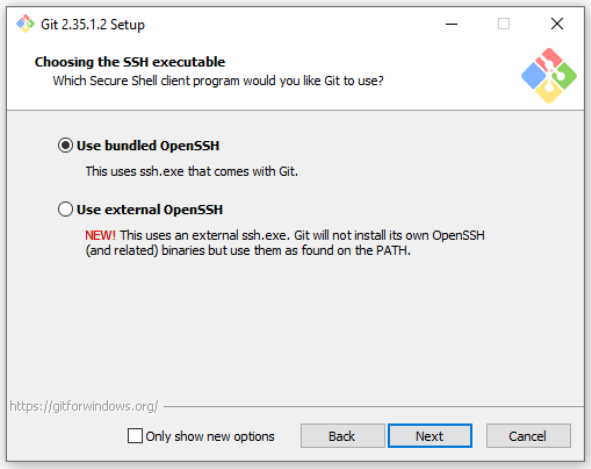


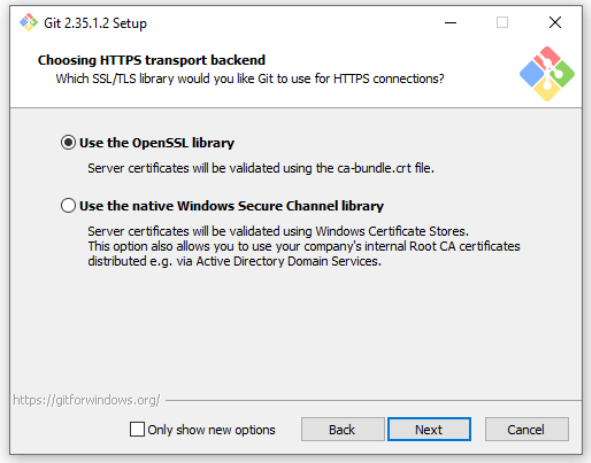



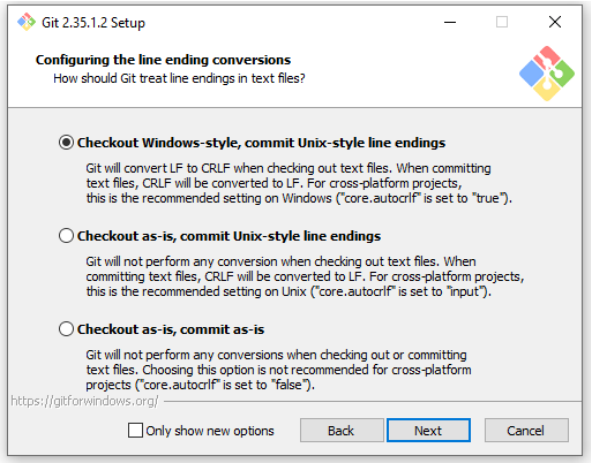





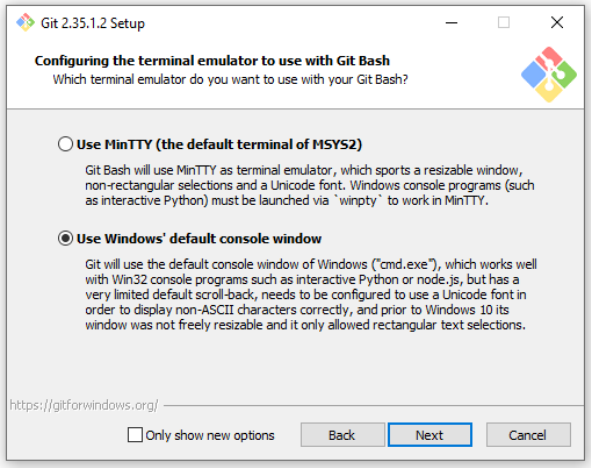



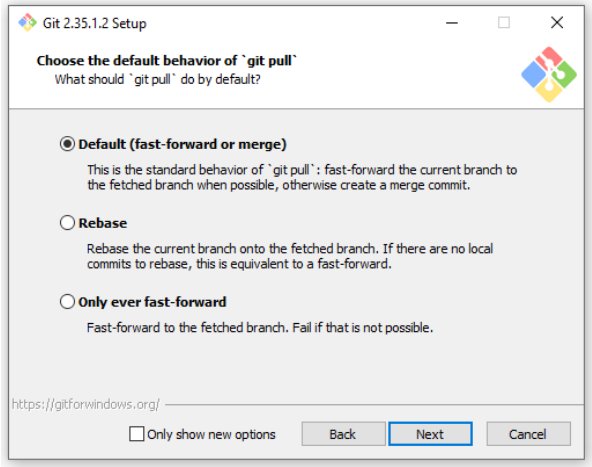



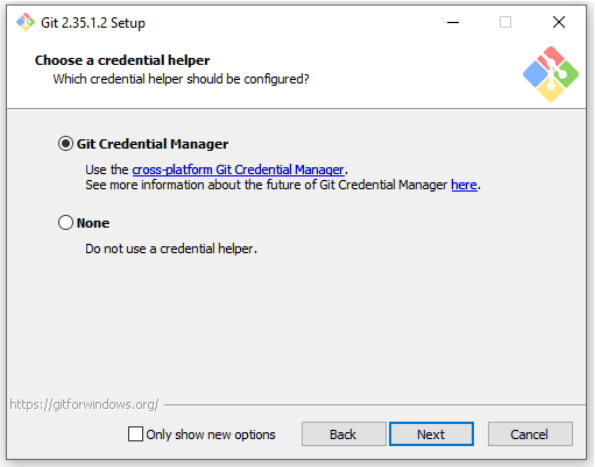



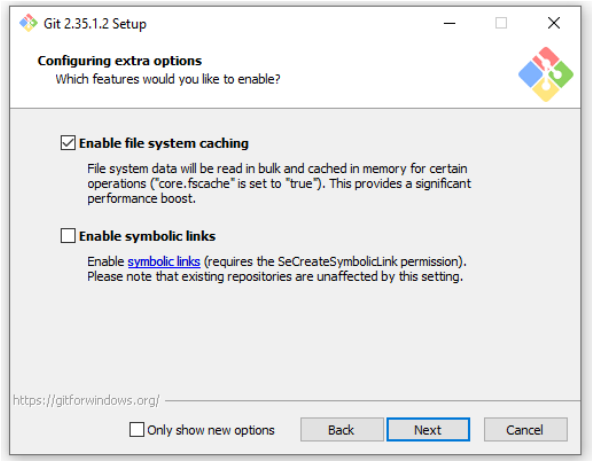



En ésta ventana se debe seleccionar Use Windows' default  console window permitiendo usar Git desde la línea de comandos "cmd.exe".  En las siguientes ventanas se presiona el botón Next, hasta que nos muestre el botón Install.

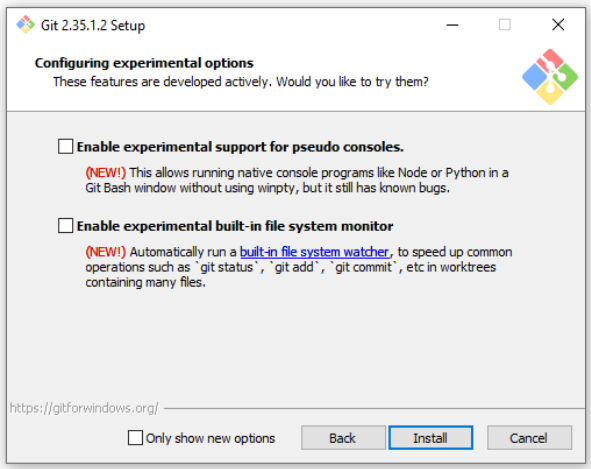


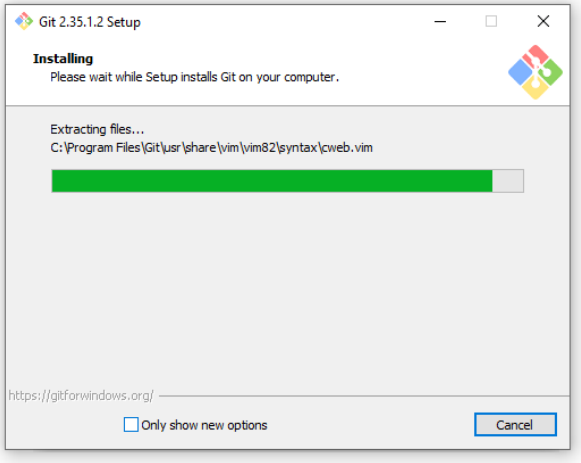



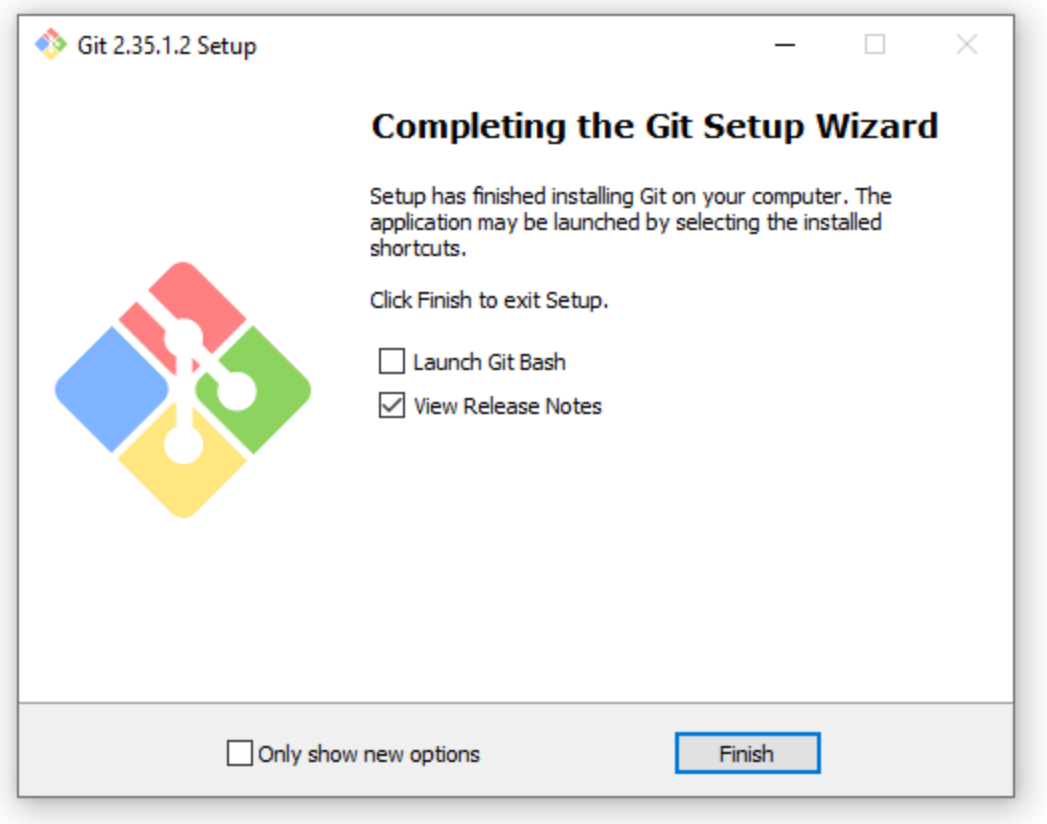




Click en el botón Install y esperar que termine la instalación

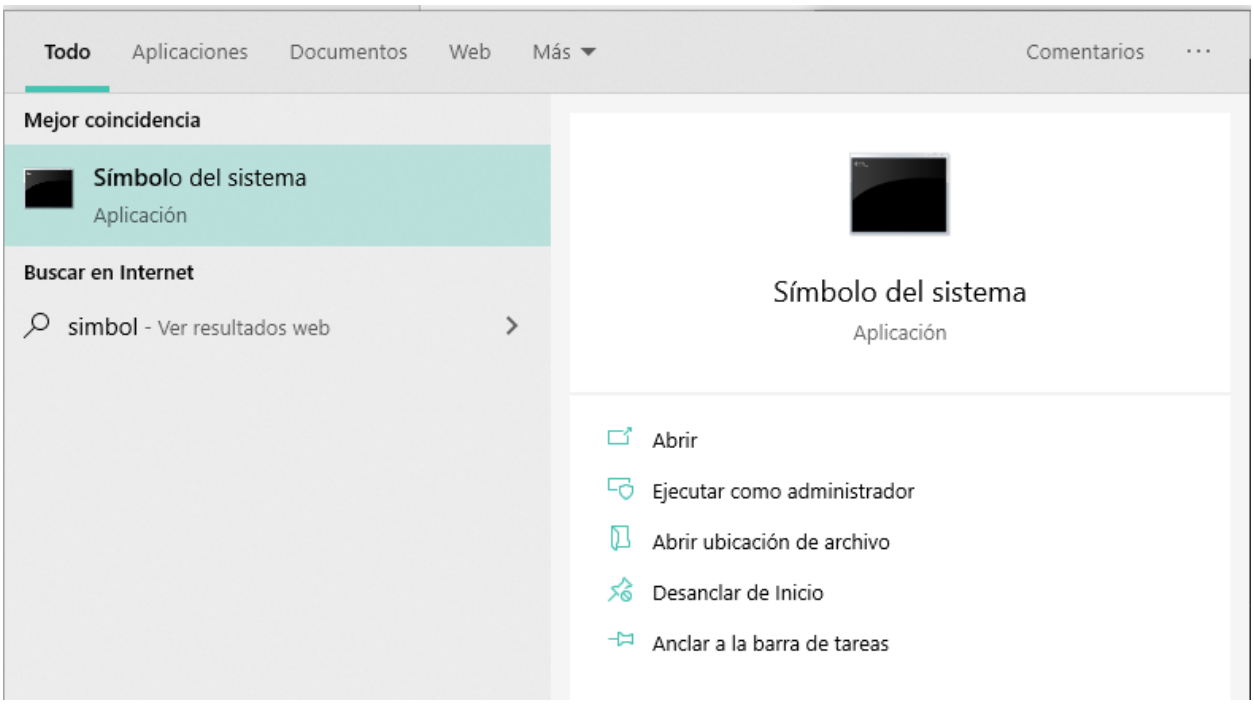


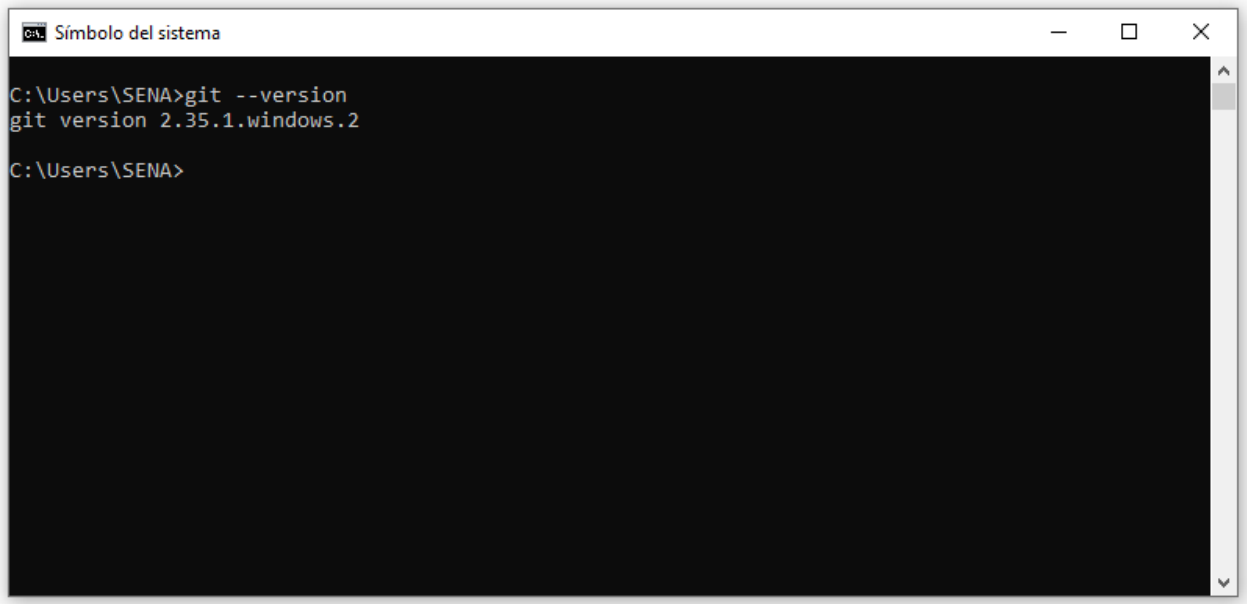
Click en Finish




Para revisar que Git esté instalado se abre una ventana de línea de comandos (Símbolo del sistema)




En la ventana del Símbolo del sistema, se escribe el siguiente comando de Git para verificar que está ya instalado. `git --version` así, y presionamos la tecla **ENTER**:



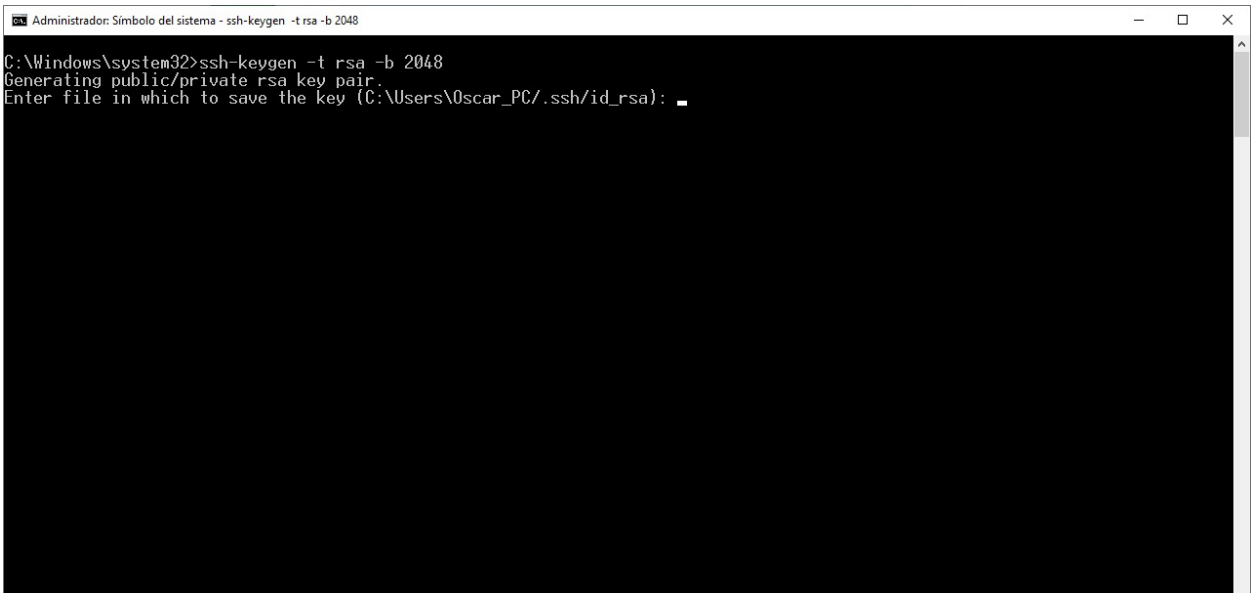
# Como crear y vincular la llave SSH para iniciar sesión con GitHub

## Creación de la llave SSH

Crearemos una llave SSH de tipo RSA 2048 bits usando el siguiente comando dentro del CMD de Windows

```
ssh-keygen -t rsa -b 2048
```

Una vez ejecutado el comando, nos pedirá la ruta donde se almacenará la llave SSH

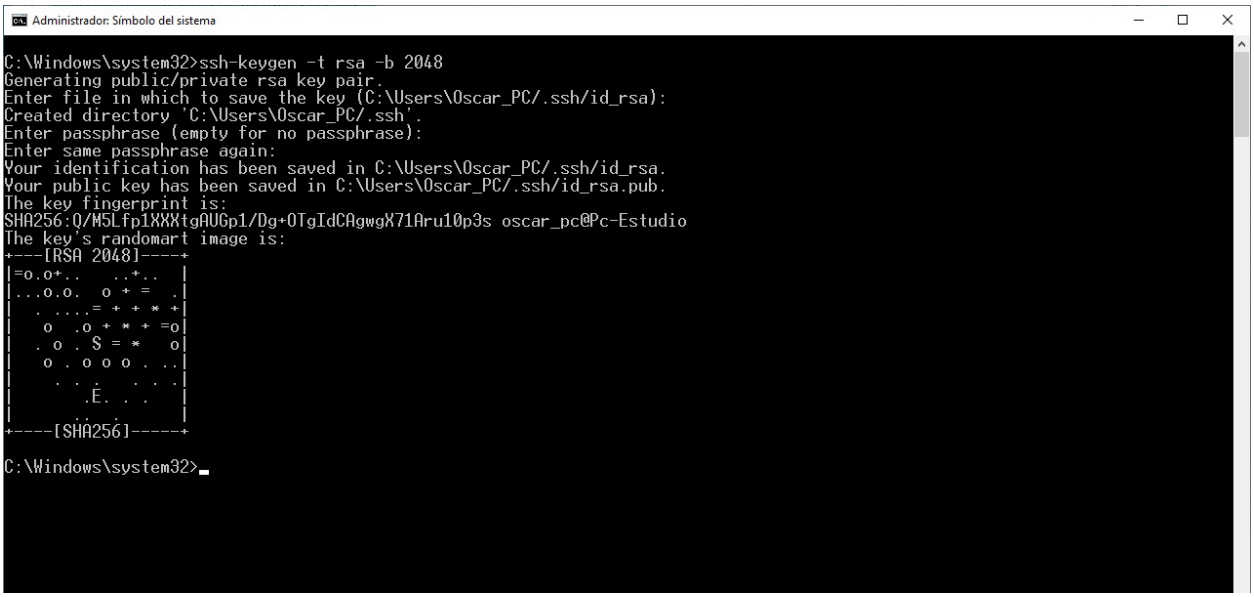



Presionamos **ENTER,** para aceptar la ruta por defecto

Después, nos pedirá ingresar una clave

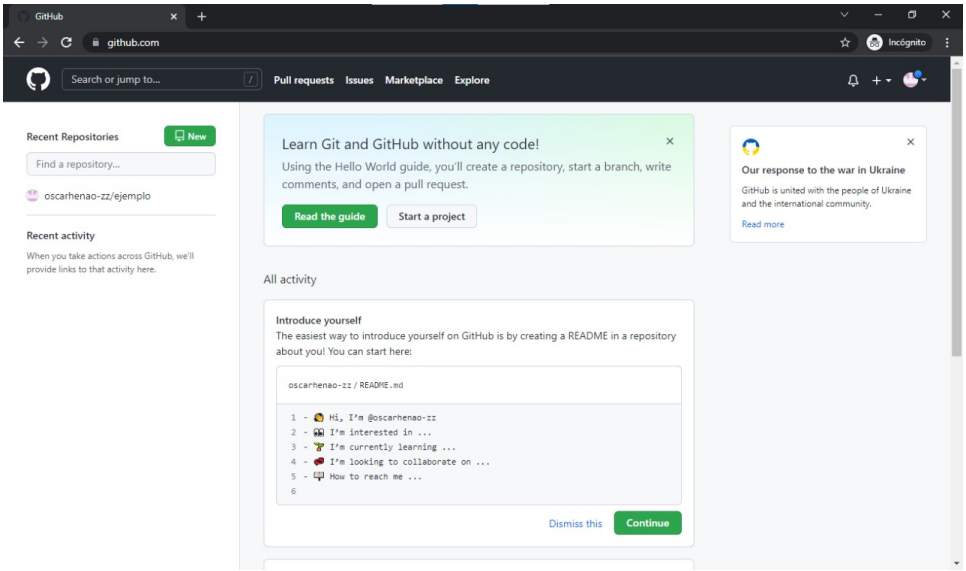
Para propósito de este tutorial, dejaremos el espacio de la clave vacío y presionamos **ENTER**

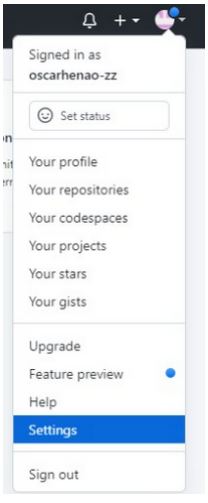
Nos generará una llave en la ubicación por defecto, y nos mostrará el siguiente resultado en consola



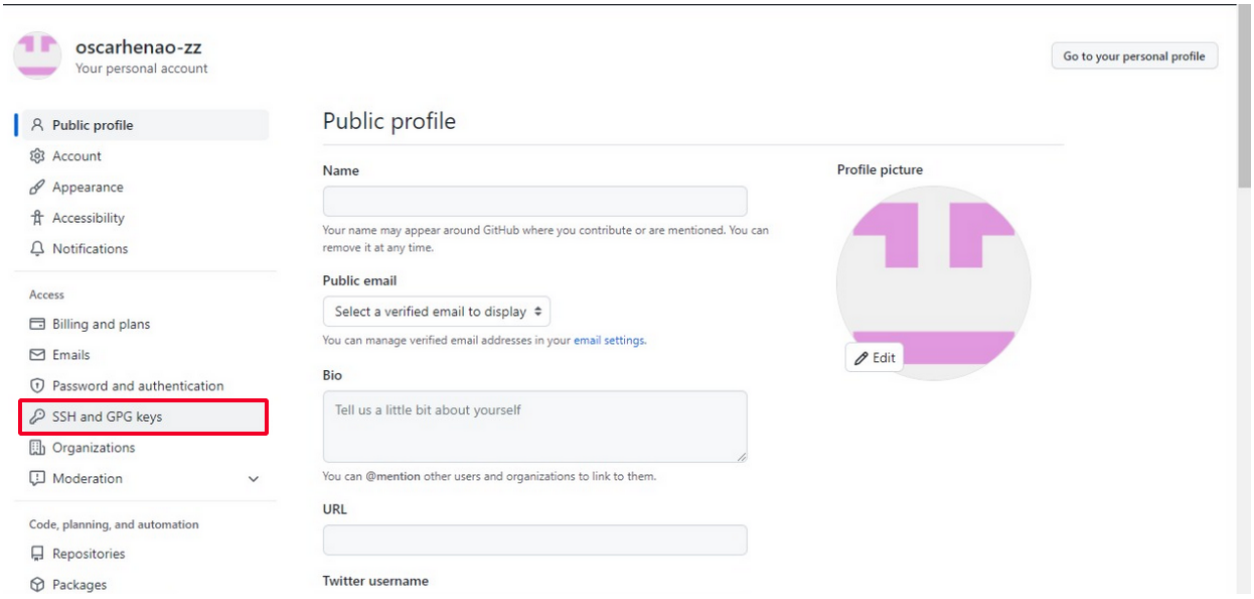

# Registrar la llave SSH con GitHub

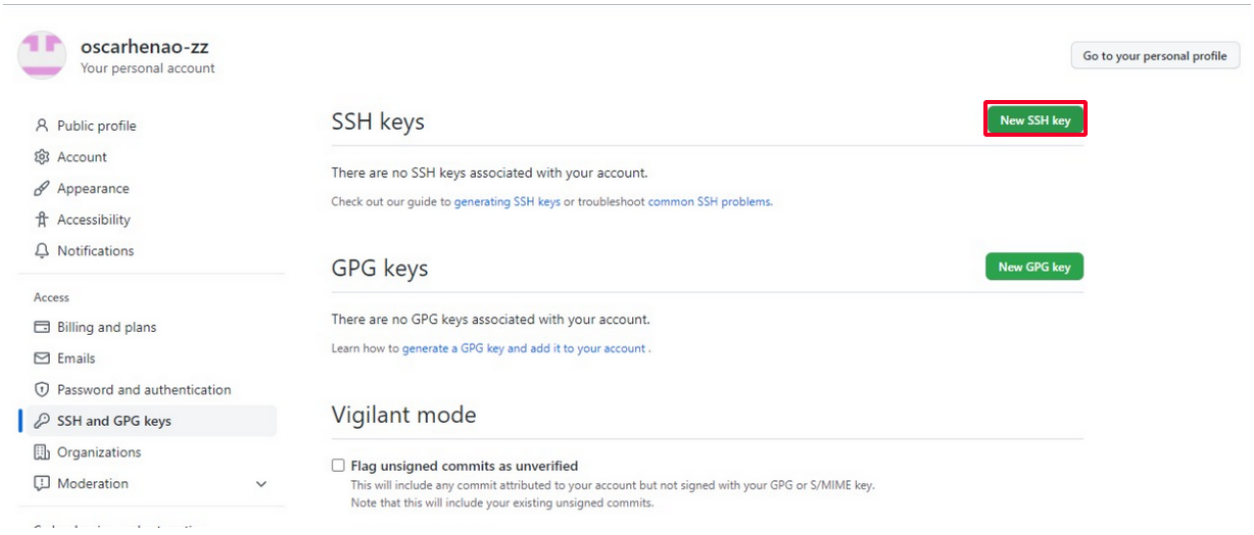
Ingresamos a nuestra cuenta de Github y nos dirigimos a la configuración



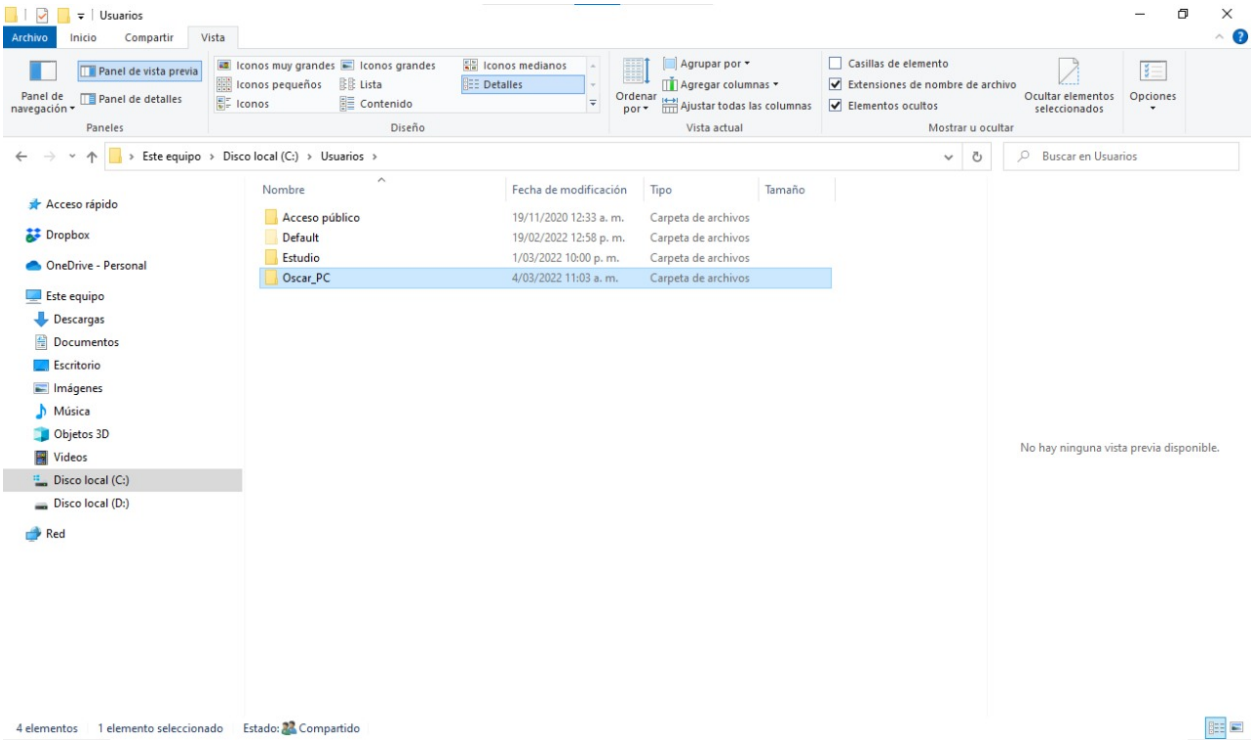



Dentro de configuración elegimos, “SSH y GPG Keys”, e ingresamos a “New SSH Key”

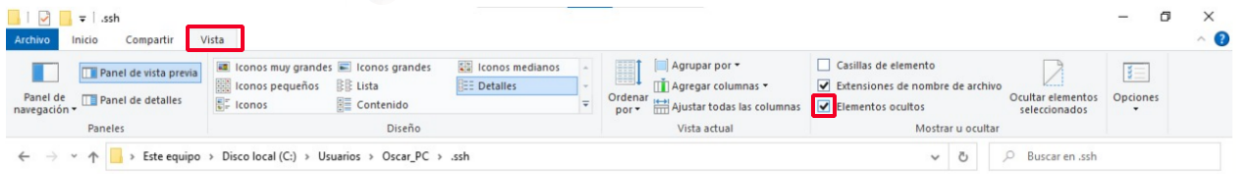


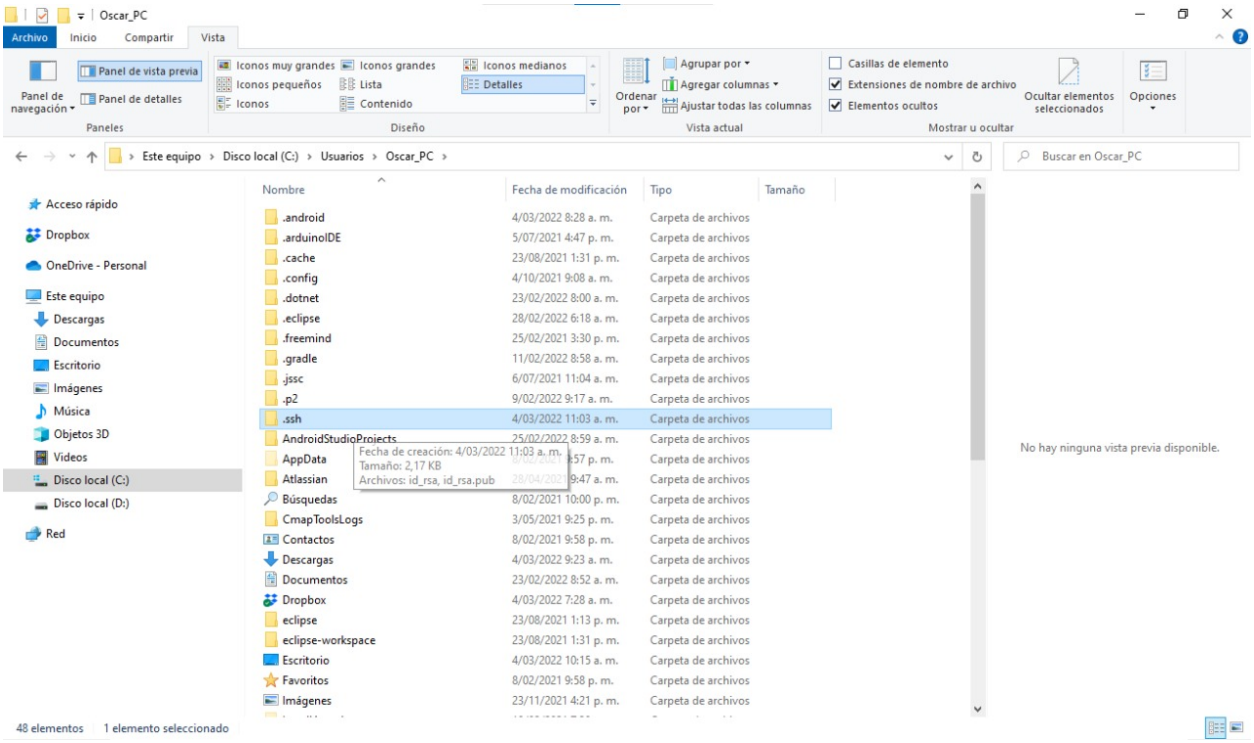

Una vez hecho esto, nos movemos a nuestro computador para buscar la carpeta de nuestro usuario, que generalmente se encuentra ubicada en **C:/Users/NombreUsuario** donde **NombreUsuario** corresponde al nombre de usuario de tu equipo



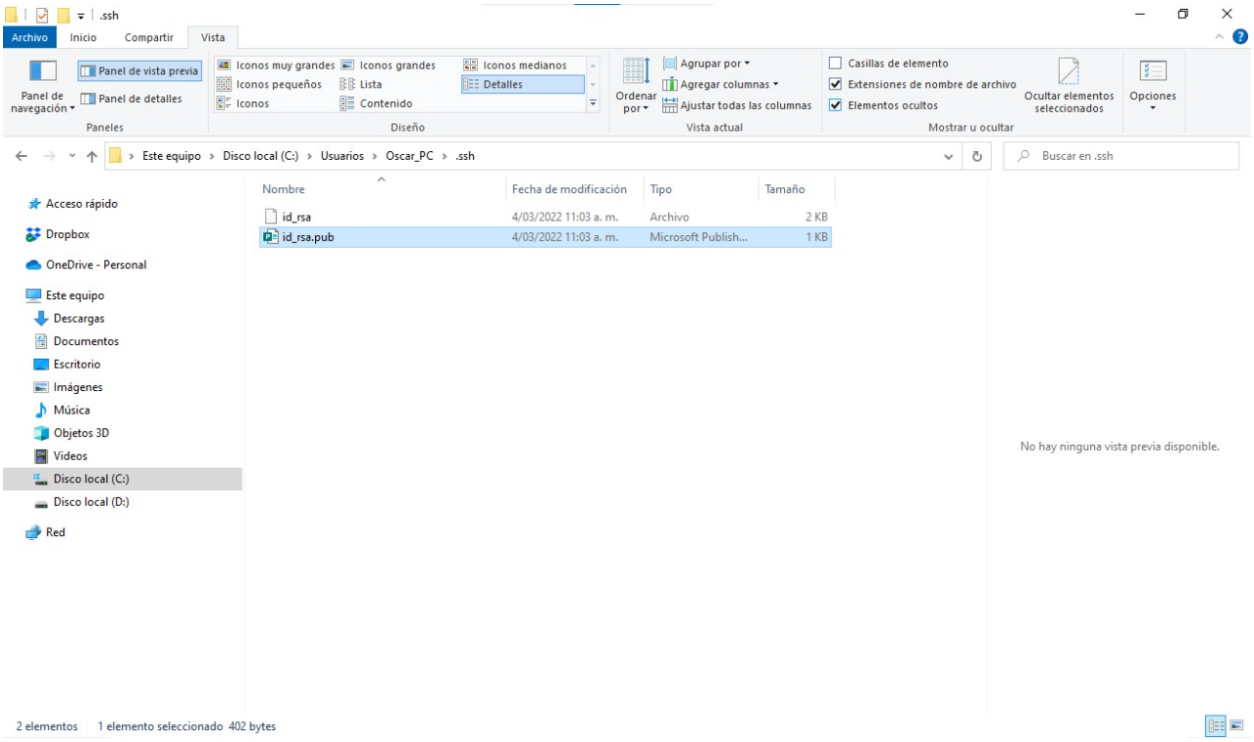

Dentro del explorador de Windows, debemos asegurarnos de activar la opción “Elementos ocultos” dentro de la pestaña de vistas



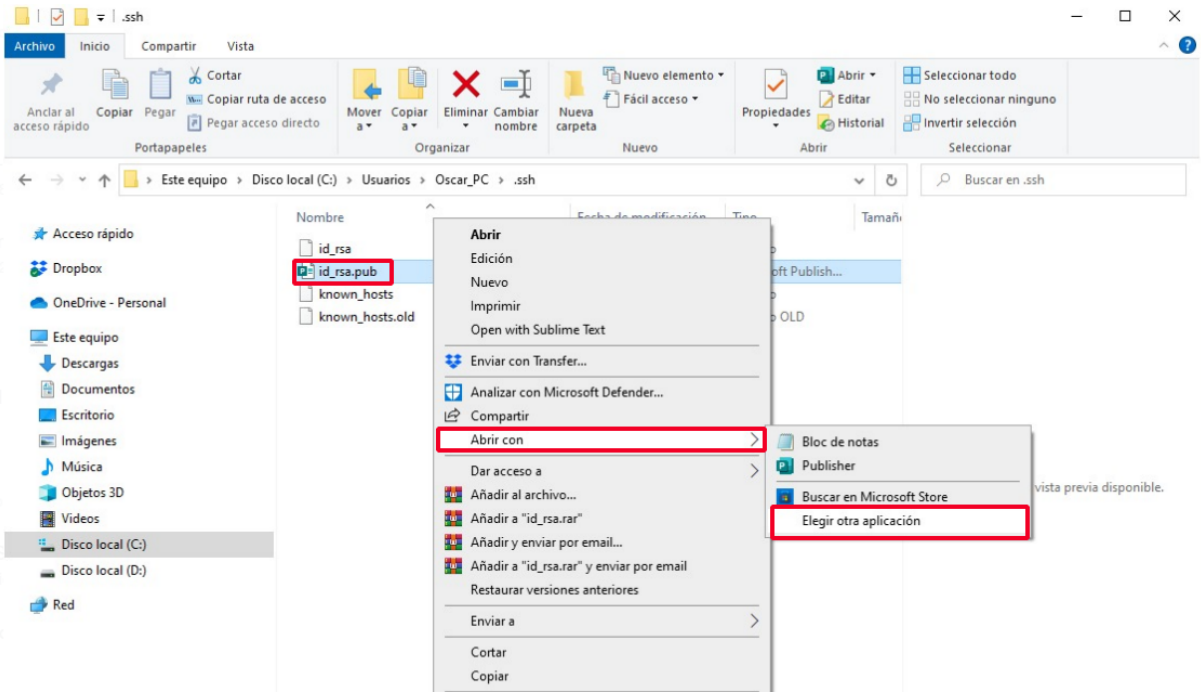

Una vez hecho esto, ingresamos a la carpeta **.ssh**

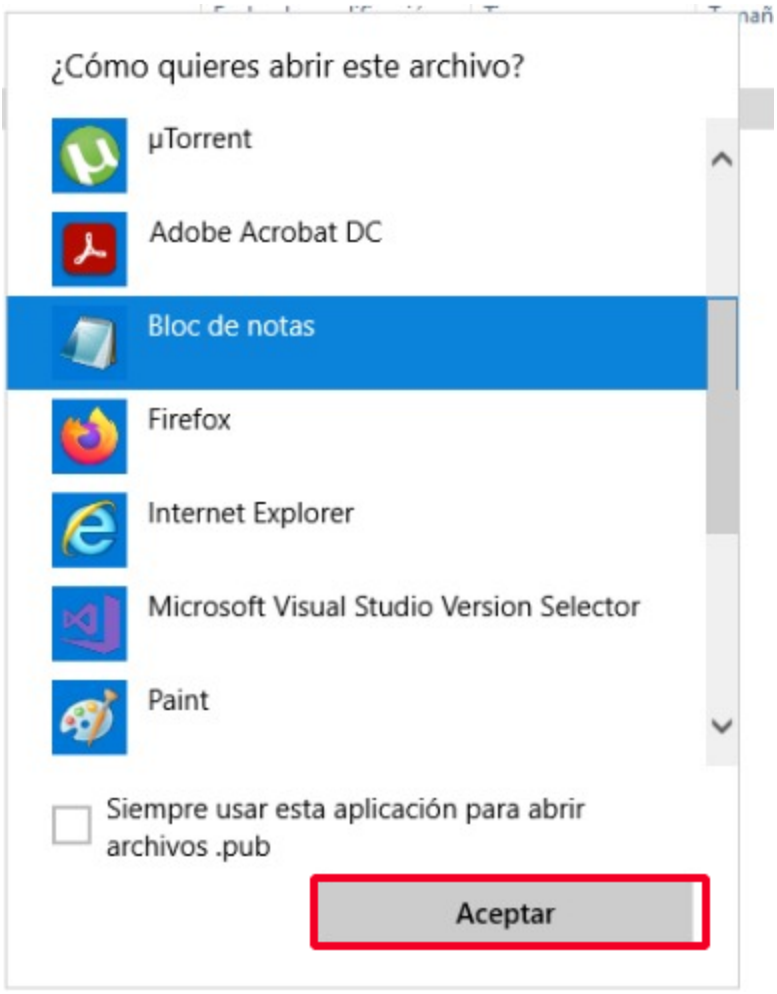



Dentro de ella, vemos dos archivos, pero solo nos interesa abrir el archivo **id_rsa.pub**

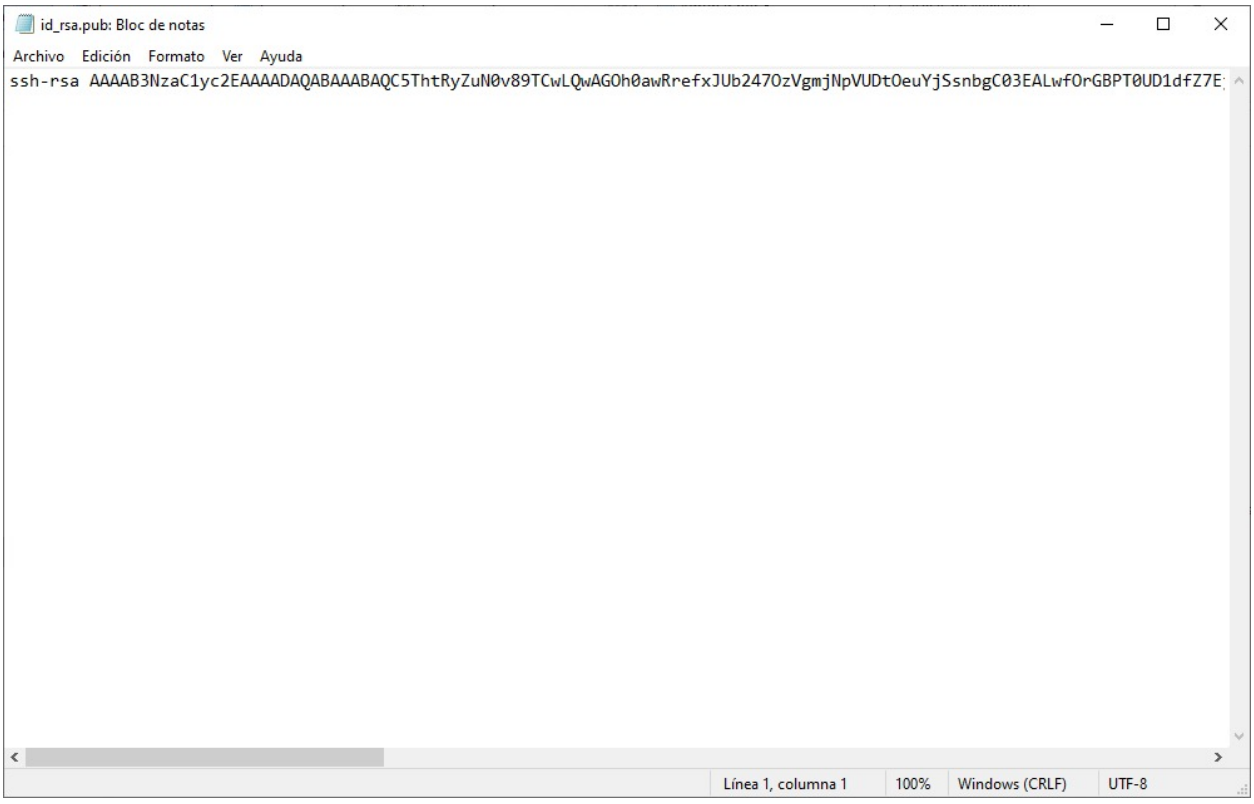


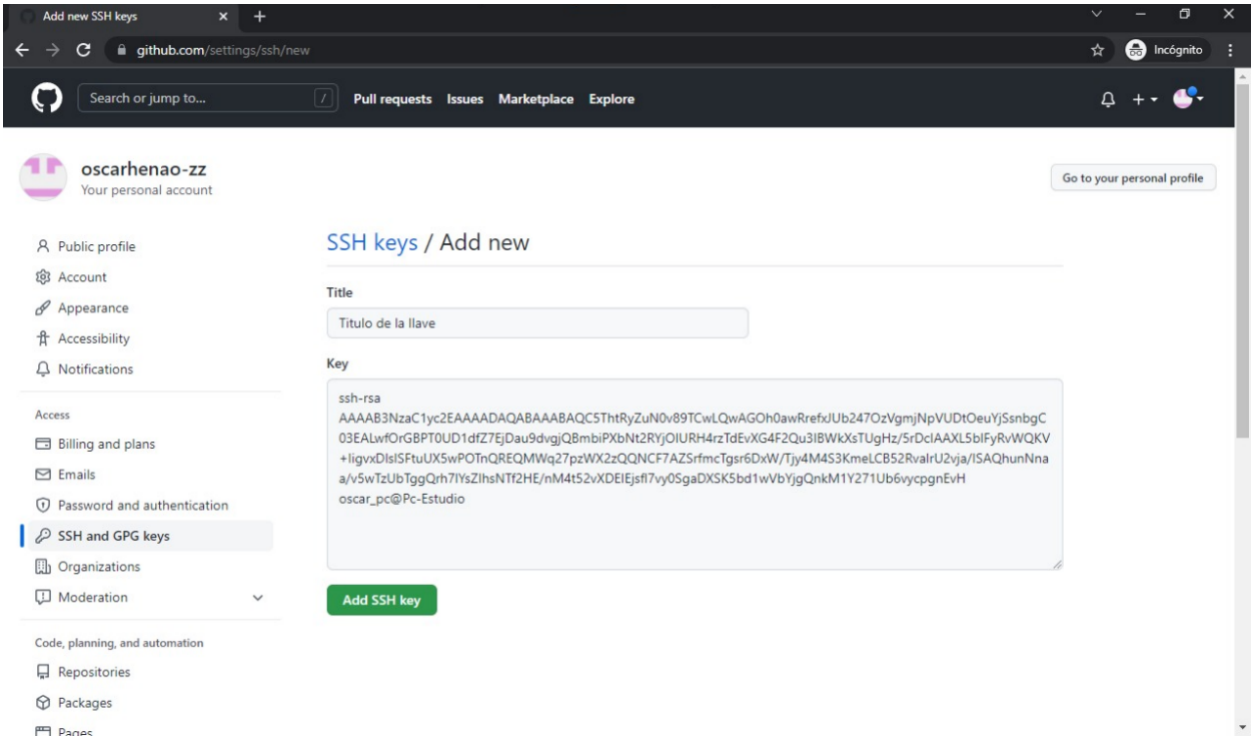
Utilizamos el comando de **Abrir con** para abrir con el programa de Bloc de notas de Windows



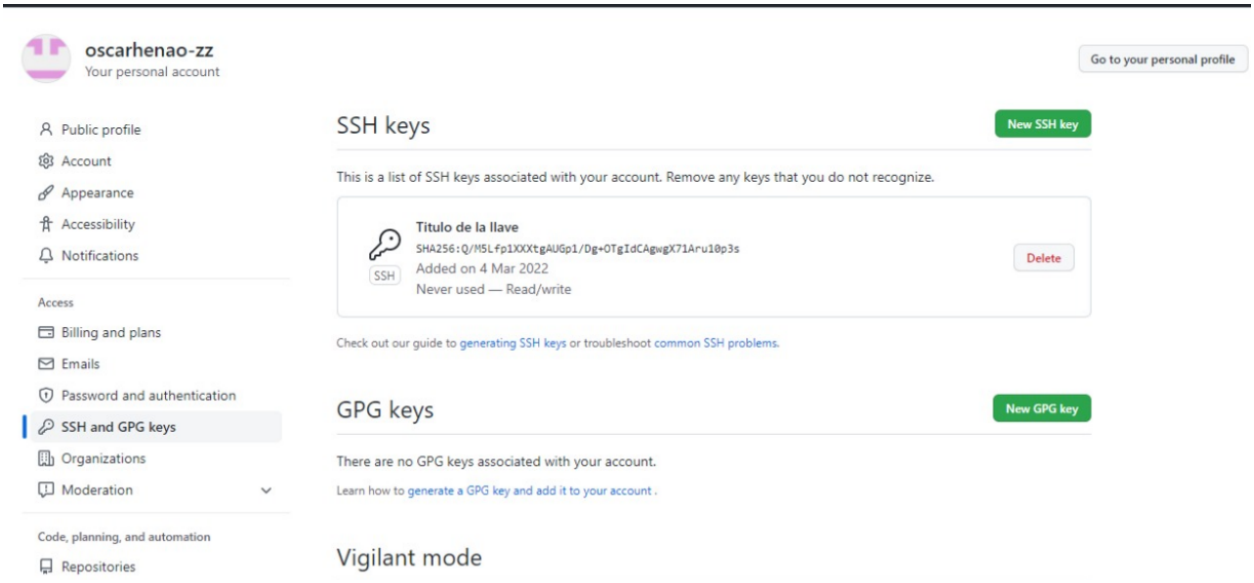



Una vez dentro, hacemos click en algún sitio y presionamos **Ctrl + E,** para seleccionar todo el contenido y lo pegamos dentro de la ventana SSH de la página de GitHub



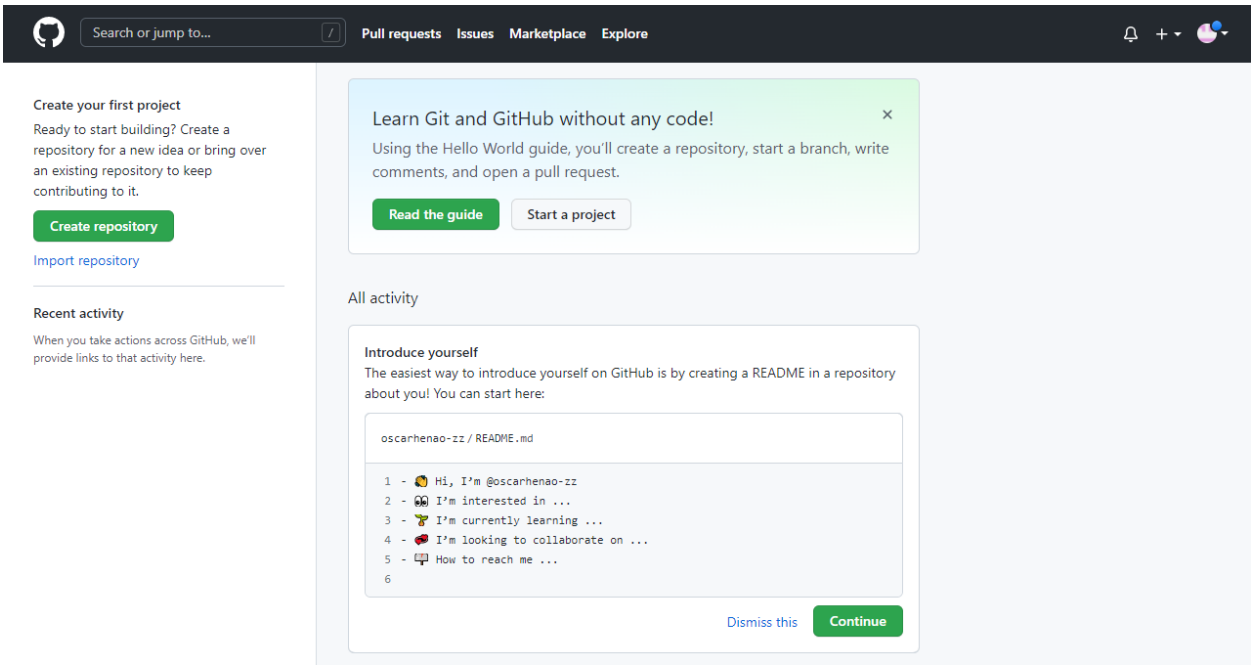




El resultado es el siguiente

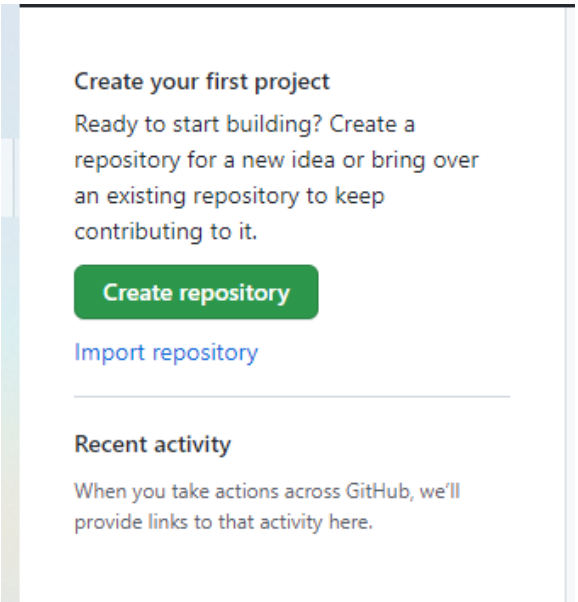



# Como crear ramas y solicitar revisión por compañeros

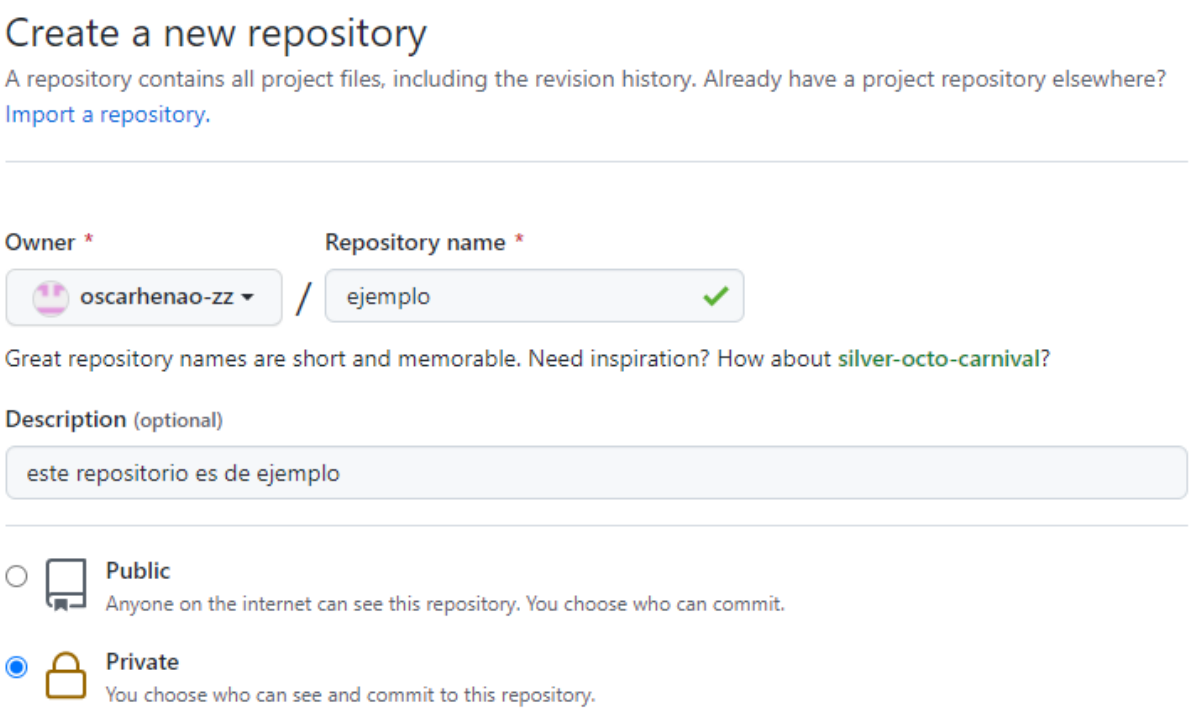
Desde nuestra pagina de GitHub



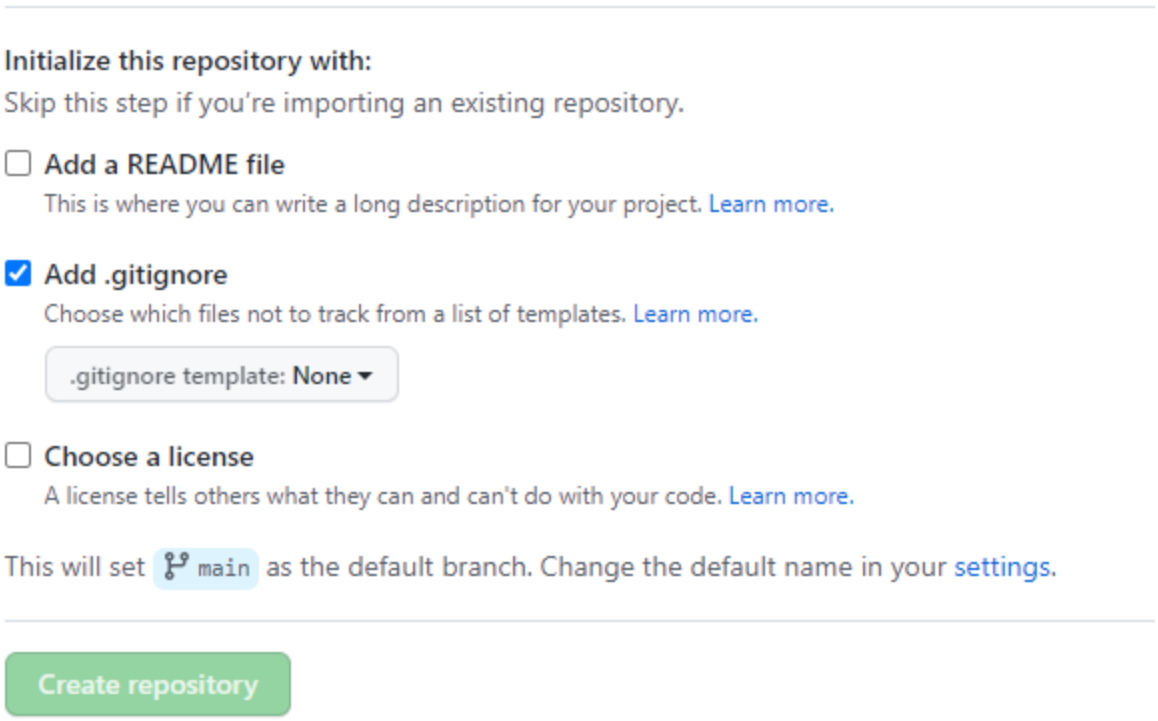
Damos click en **Create Repository**



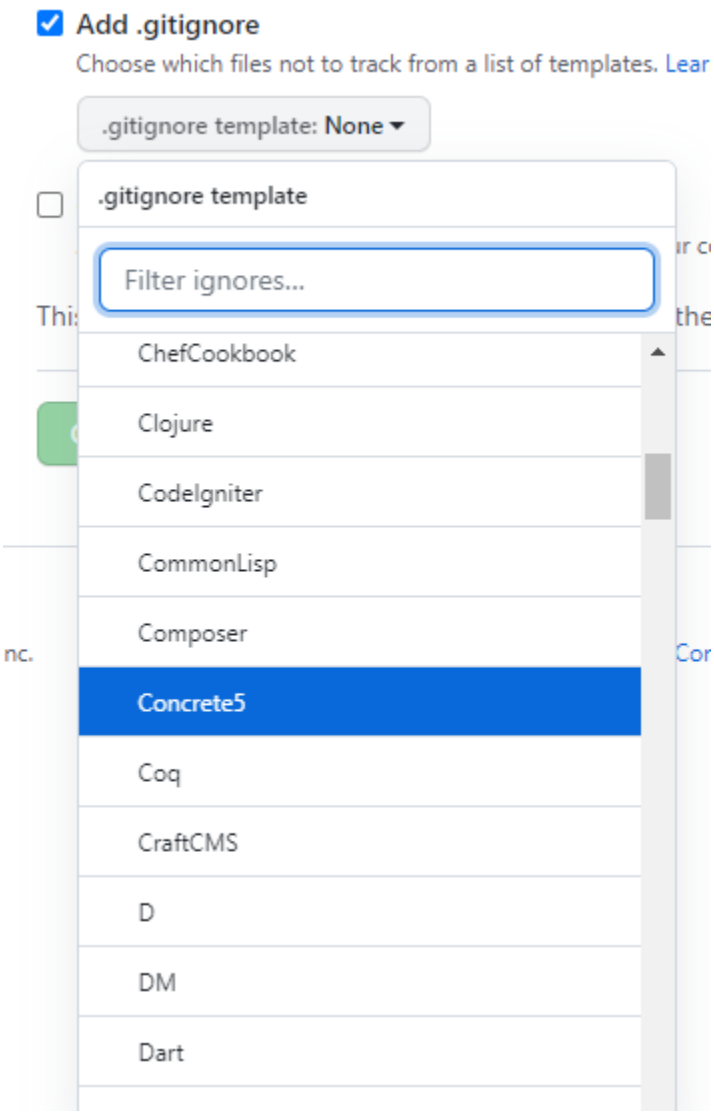

Rellenamos los datos, para efectos del tutorial, lo crearemos privado



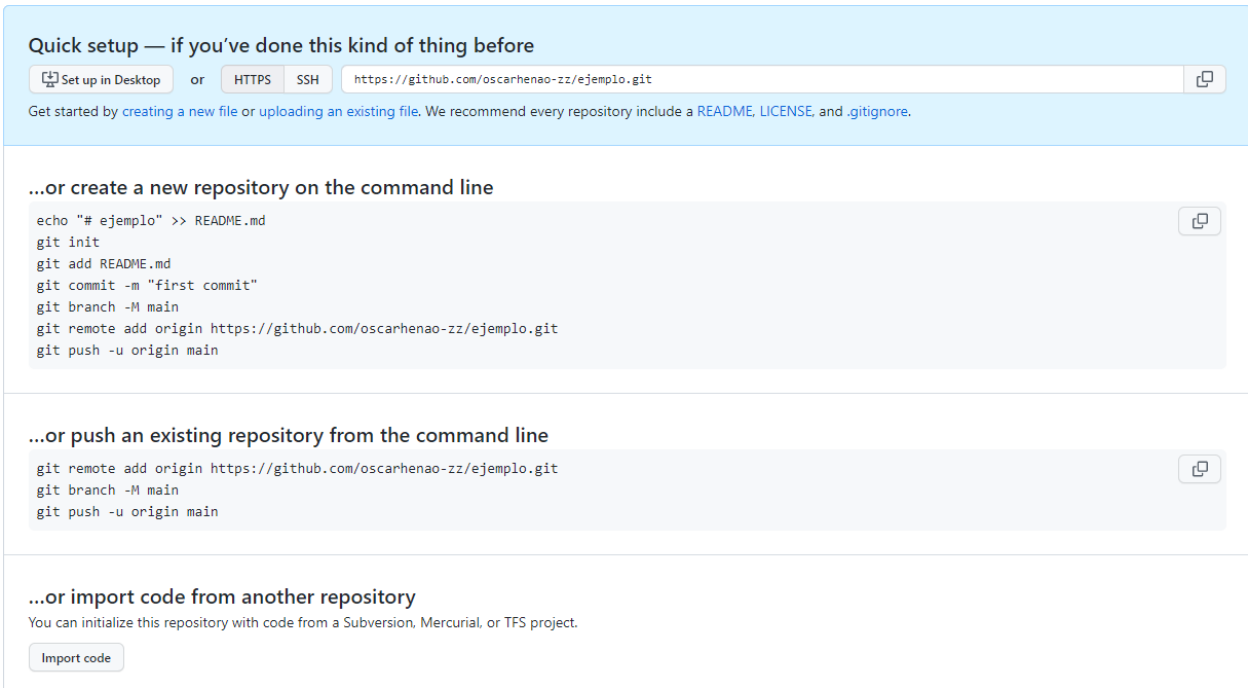

Configuramos el proyecto




Agregamos el Git Ignore según nuestro proyecto, o lo dejamos vacío

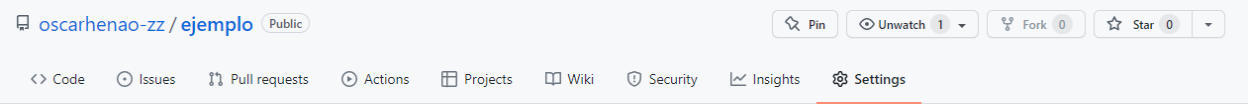



Una vez dentro del proyecto, podemos apreciar el inicio de GitHub con algunas instrucciones de como iniciar

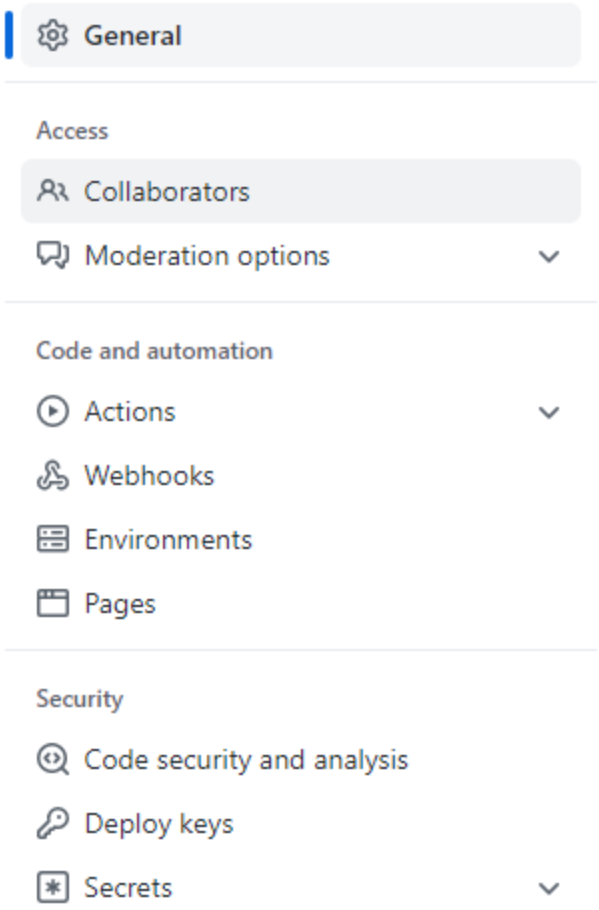



⚠️ Acceso público solo permite la lectura libre del proyecto de GIT, más no la modificación directa, no hay nada de que preocuparse.

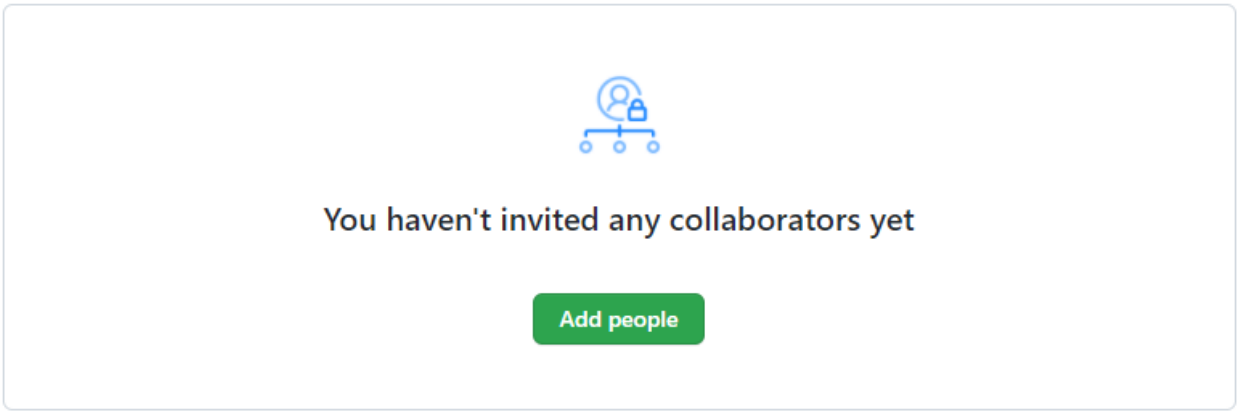
Para habilitar la edición a tus compañeros de proyecto dentro de GitHub, primero debemos agregarlos como colaboradores



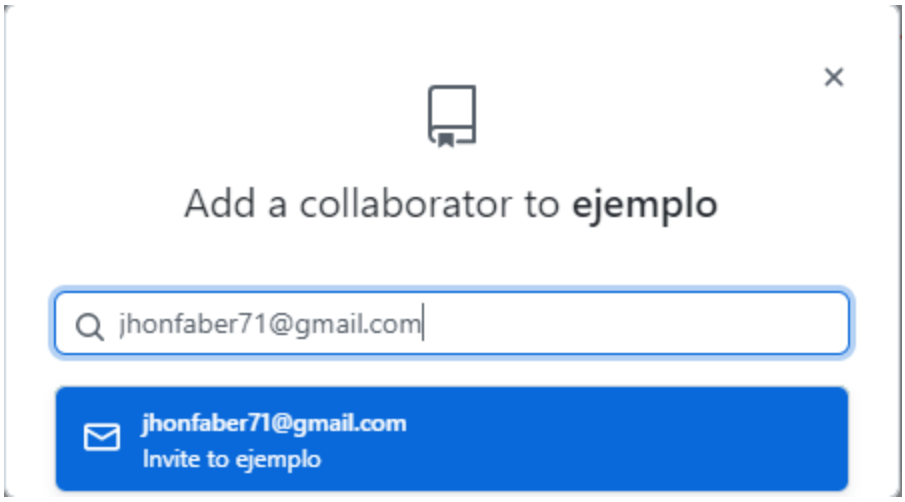

Dentro de la configuración del proyecto, accedemos a **Collaborators**



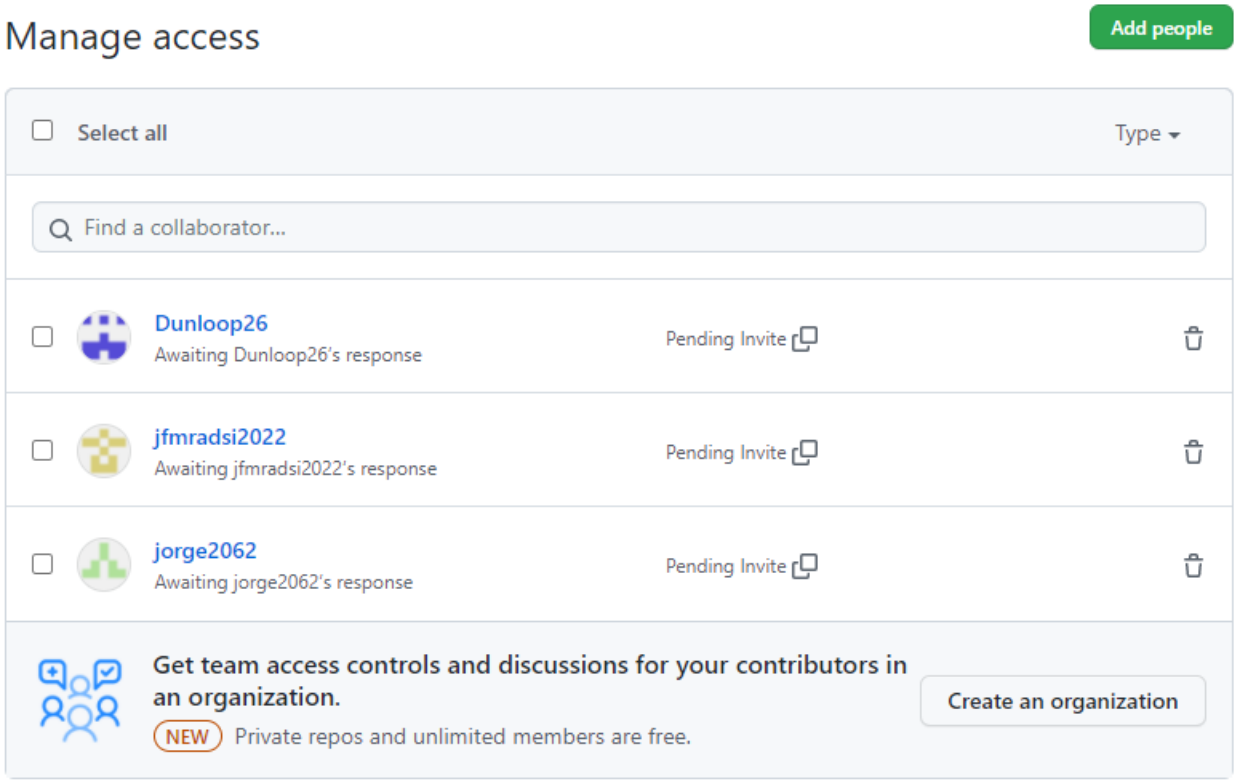

Para agregar colaboradores damos clic en **Add People**




Agregamos los correos electrónicos de nuestros compañeros de equipo




Una vez invitados, podemos manejar todo el acceso



# Manual de comandos de GIT

`git init`

Inicializa el repositorio de git dentro de una carpeta de archivos

`git add <archivo-1> <archivo-2>`

Agrega los archivos sin seguimiento y con cambios a la fase de STAGING dentro de git.

Usa `git add .` para agregar todos los archivos dentro del espacio de trabajo a la fase de staging

`git commit -m <mensaje>`

Hace un registro de los archivos en la fase de STAGING dentro del historial de git, dejando una marca temporal o **commit** que nos permite regresar a esta versión del proyecto.

`git checkout <commit-id>`

Permite la revisión del proyecto desde una marca en el tiempo o **commit** especificado.

`git branch <rama_nueva>`

Crea una rama desde el **commit** en el que me encuentro actualmente

`git checkout -b <rama_nueva>`

Este comando crea una rama a partir de un commit, y hace su checkout automáticamente.

Es equivalente a `git branch <rama_nueva> && git checkout <rama_nueva>`.

`git log`

Muestra el historial de todos los commits sobre la rama actual.

Si se desea ver el historial sobre una rama o commit `git log <rama_o_commit-id>`.

`git branch`

Muestra todas las ramas del un proyecto de git local.

Para mostrar las ramas de todo el proyecto (locales y remotas), se usa `git branch -a`

`git restore`

Restaura los cambios actuales del proyecto.

# ¿Cómo iniciar un proyecto en GIT desde la consola?

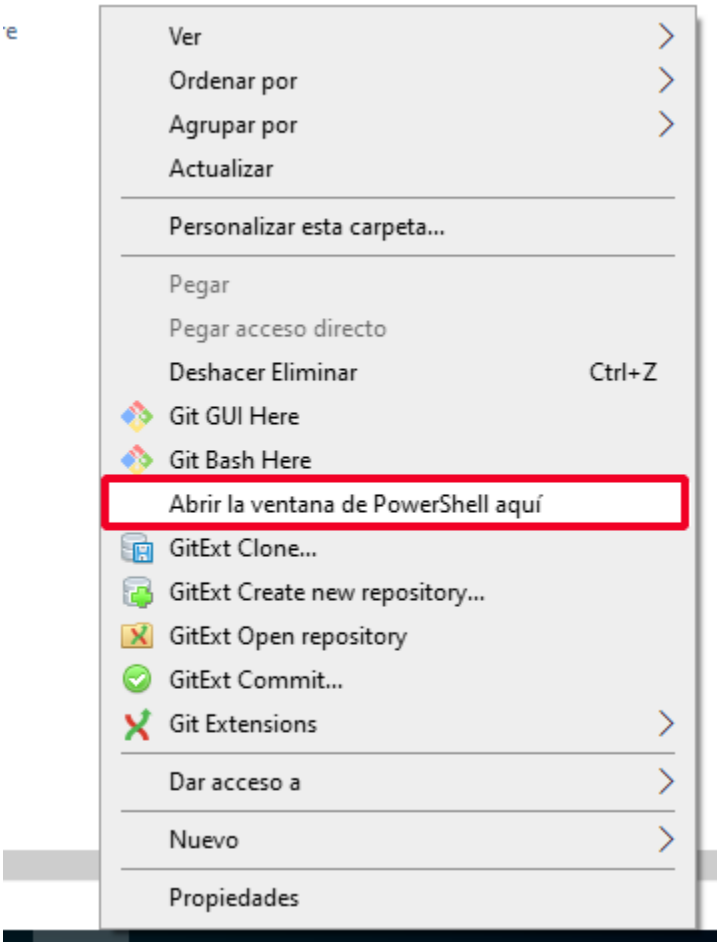
Para iniciar un nuevo proyecto de GIT es importante tener claro lo siguiente:

- El proyecto es separado por carpetas, es decir, depende de qué carpeta elijas para comenzar el proyecto
- GIT comenzará el proyecto con todos los archivos que tengas dentro de esa carpeta
- GIT creará una rama del proyecto por defecto, llamada master o main, en algunos casos

# Primeros pasos

Abrimos la terminal de windows presionando las siguientes combinaciones

Presiona **Windows + E** para lanzar el explorador de archivos
A continuación, presionamos la tecla **Mayus + Click Derecho** para desplegar el menú contextual y elegimos la opción **“Abrir con Linea de Comandos / Powershell”**



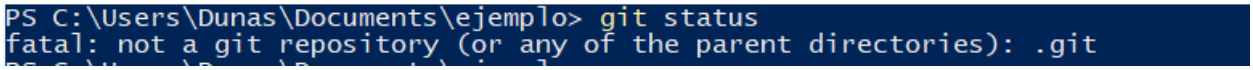

Una vez ubicados en la carpeta, podemos empezar un proyecto de GIT desde la línea de comandos

# Creación de un proyecto de GIT

Para empezar escribiremos el comando

```
git status
```

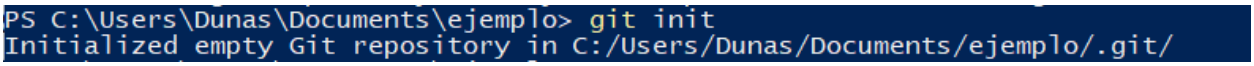
Recibiremos el siguiente mensaje




Esto indica que nuestra carpeta aún no tiene el proyecto de GIT inicializado

Inicializamos git con el siguiente comando

```
git init
```

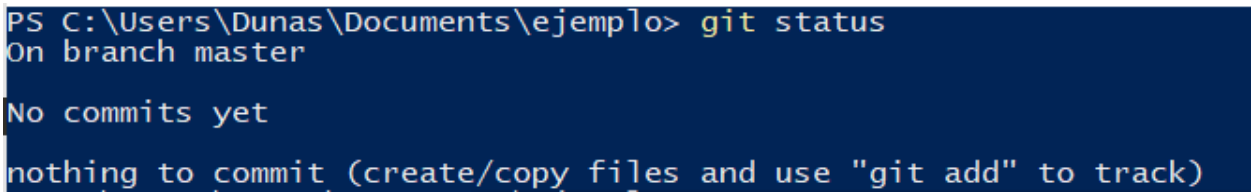



A partir de este momento, nuestro proyecto en git ya se encuentra inicializado

# Trabajando con GIT

Una vez ubicados dentro de nuestro proyecto de GIT, ejecutamos el siguiente comando

```
git status
```



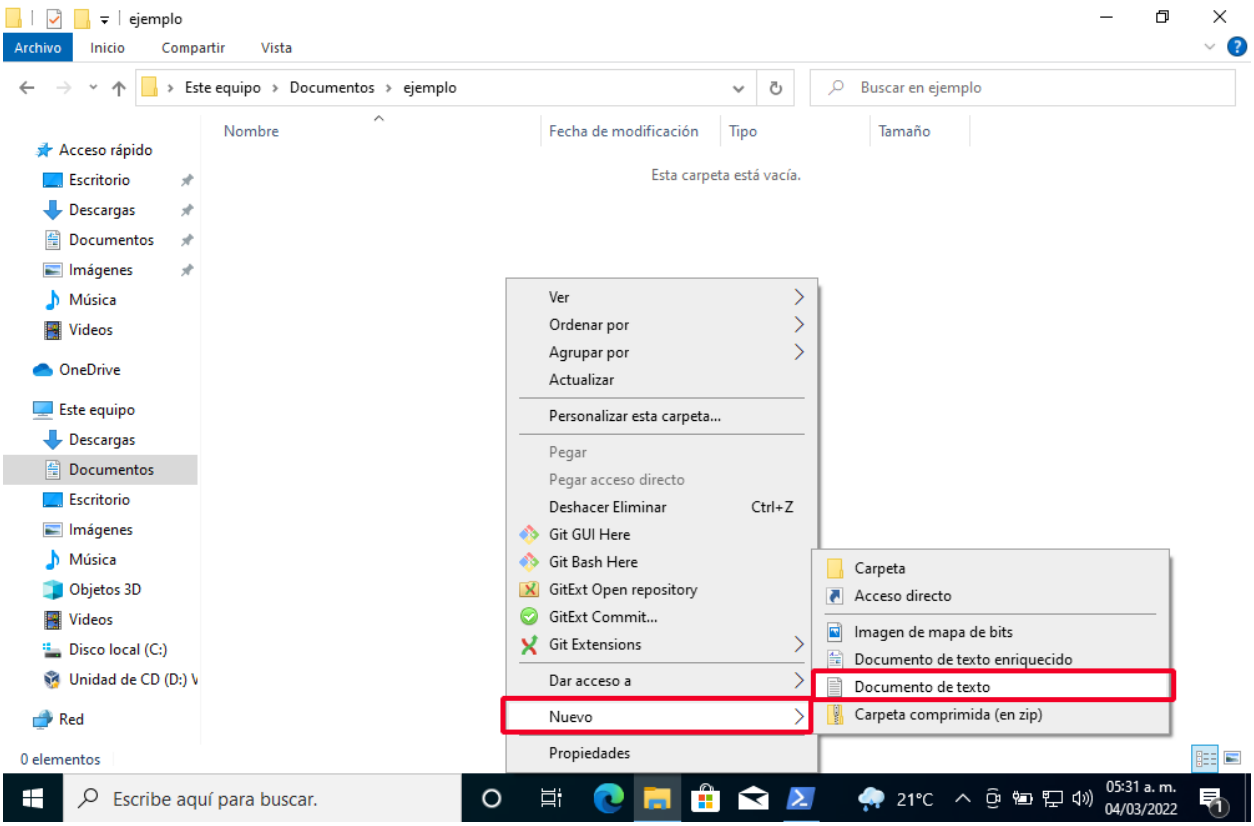
Este comando nos permite ver el estado actual del proyecto de GIT, desde archivos nuevos, archivos modificados, hasta archivos que han sido eliminados desde el commit anterior.

## Creando archivos dentro de nuestro proyecto

Creamos un archivo de texto

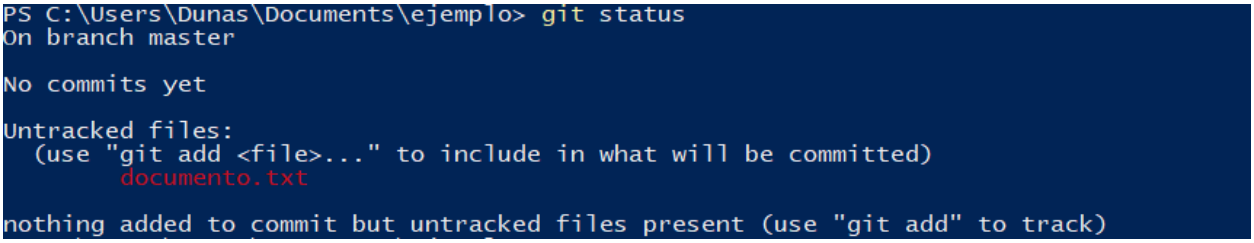


Utilizamos **click derecho** para abrir el menú contextual sobre nuestra carpeta.
Una vez en ella, nos dirigimos a **Nuevo > Documento de texto**, para crear un archivo de texto vacío dentro de nuestra carpeta del proyecto




Nombramos a ese archivo de texto **documento.txt**

Una vez creado nuestro proyecto, con el comando `git status`, veremos que nuestro archivo de texto aparece dentro de nuestro proyecto GIT como un **nuevo cambio**






Visualizamos nuestro documento.txt de **color rojo** y dentro de la categoría **“Untracked files”.**
Esto significa que nuestro archivo **no se encuentra en seguimiento**, es decir, es un archivo nuevo.
A partir del siguiente commit, se seguirán los cambios realizados al archivo.

Agregamos nuestro documento de texto a la fase de **staging** con el siguiente comando

```
git add documento.txt
```

En este momento aceptamos los cambios de nuestro proyecto agregando el nuevo archivo de texto.

Ahora solo falta realizar **nuestro primer commit**

Para ello se utiliza el siguiente comando

```
git commit -m "Nuevo documento de texto"
```

el comando `git commit` me permite registrar una “fotografía” del proyecto con los cambios actuales, de tal forma que, yo pueda **regresar a estos cambios**

Para utilizar este comando siempre es necesario proveer un mensaje después de escribir `-m` para fines de documentación de los cambios en nuestro proyecto.

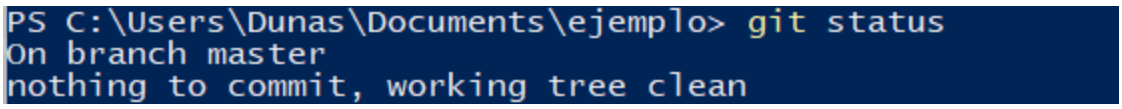
El comando completo sería el siguiente

```
git commit -m "<nuestro_mensaje>"
```

> 💡 En caso de no proveer el mensaje con `-m` tendremos que escribir el mensaje usando un editor externo configurado desde el sistema y GIT.

Una vez hecho el commit aceptando el nuevo documento.txt a través de `git add documento.txt` (Aceptamos los cambios a documento.txt) y después `git commit -m "Nuevo documento de texto"` (Registramos el punto en el tiempo que modificamos nuestro nuevo archivo) encontramos que nuestro proyecto regresa a la normalidad.

```
git status
```




> 💡 Este mensaje nos indica que nuestro proyecto no tienen **ningún cambio actual**, en este caso encontramos que ya se hizo el commit aceptando el cambio anterior, reiniciando el estado actual

## Visualizando nuestro historial de commits con GIT

Una vez realizado nuestro primer commit, podemos visualizar el historial de cambios usando el siguiente comando

```
git log
```

```
PS C:\Users\Dunas\Documents\ejemplo> git log
commit b4da01653d0b1648c0364d37074b9f82502aeb92 (HEAD -> master)
Author: user <user@myemail.com>
Date:   Fri Mar 4 05:53:00 2022 -0600

    Nuevo documento de texto
```

TODO: <Imagen del primer log>

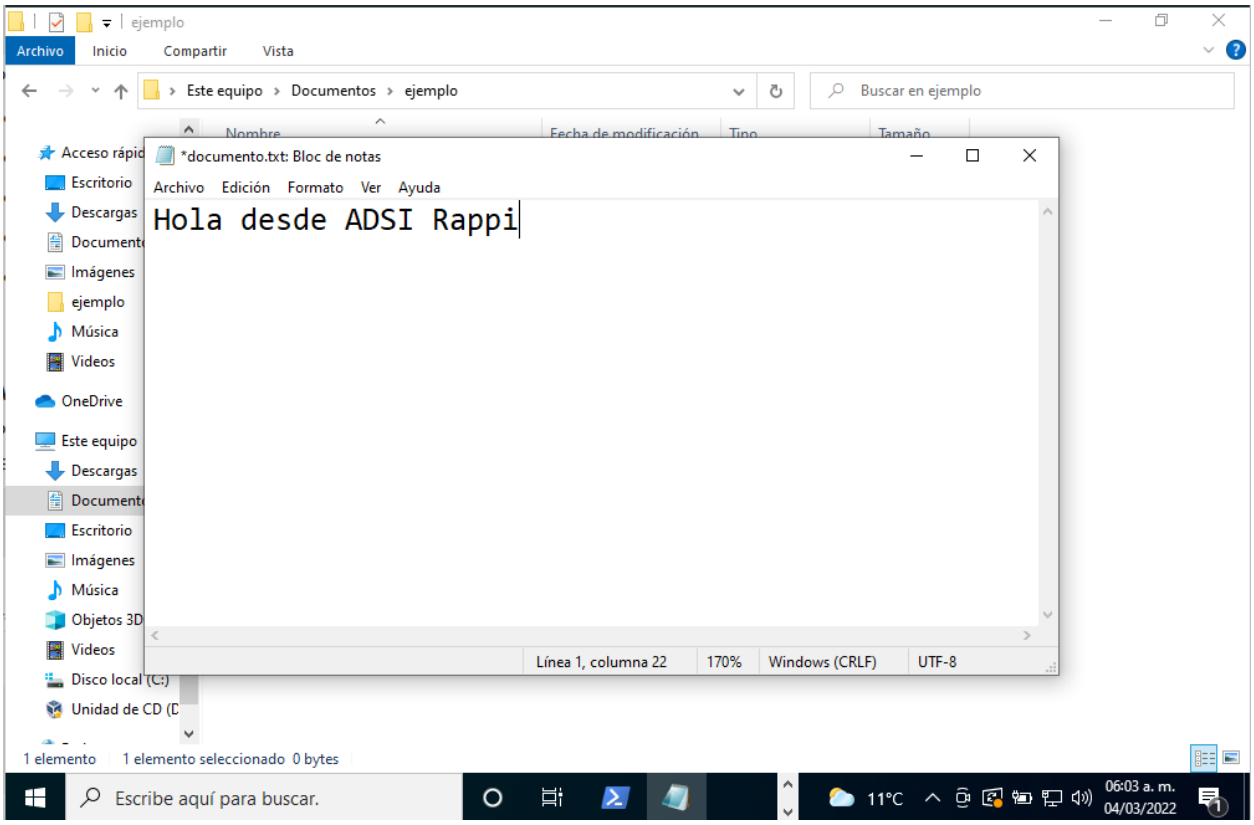
Podemos visualizar nuestro commit realizado anteriormente junto con la siguiente información

- El HASH del Commit (Que es un ID único generado según el contenido del proyecto)
- La posición de la rama actual (en este caso master)
- El autor del commit, incluyendo nombre de usuario y correo electrónico
- La fecha y la hora en la que se realizó dicho commit
- El mensaje con el cuál describimos los cambios hechos en el proyecto

## Modificando archivos dentro de nuestro proyecto

Para este segmento, vamos a agregar o modificar elementos dentro de nuestro documento de texto.

> 💡 Abrimos nuestro archivo y agregamos algo de texto para continuar




Una vez modificado nuestro archivo de texto, ejecutamos el siguiente comando

```
git status
```

```
PS C:\Users\Dunas\Documents\ejemplo> git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   documento.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

Podemos ver que nuestro archivo aparece nuevamente pero esta vez, bajo la sección **“Changes not staged for commit”**, es decir, existen **cambios nuevos** dentro de nuestro archivo

Ahora que nuestro archivo se encuentra actualmente en GIT, se consideran solamente los cambios nuevos sobre nuestro archivo

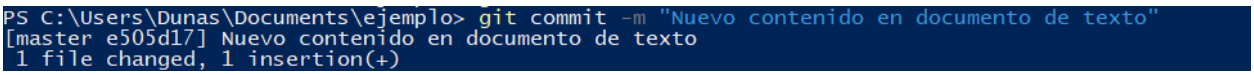
Ahora, vamos a aceptar nuestros cambios ejecutando

```
git add .
```

💡 Con **git add .** indicamos a GIT que queremos aceptar los cambios de **todo el proyecto desde la carpeta en la que estoy**, es decir, todos los archivos, carpetas y subcarpetas modificadas dentro de GIT

Y registramos nuestro cambio en una “fotografía” de cambios de GIT, con el siguiente comando

```
git commit -m "Nuevo contenido en documento de texto"
```




Visualizamos nuestro historial de cambios nuevamente con

```
git log
```

Y podemos ver lo siguiente




Tenemos nuestros dos commits registrados dentro de GIT con los mensajes respectivos

> 💡 Podemos ver que el commit más reciente aparece siempre **en la parte superior**

# Trabajando con ramas en GIT

Vamos a empezar creando nuestra primera rama.

Una rama es una versión de nuestro proyecto alterna, donde podemos desarrollar cambios independientes sin afectar los cambios pertenecientes a otra rama

Utilizamos el siguiente comando

```
git branch nueva_rama
```

Este comando creará una rama llamada **”nueva_rama”** dentro de nuestro proyecto de GIT

> 💡 Es importante no incluir nombres de ramas con espacios en GIT

Para visualizar las ramas de nuestro proyecto, podemos usar

```
git branch
```

```
PS C:\Users\Dunas\Documents\ejemplo> git branch
* master
  nueva_rama
PS C:\Users\Dunas\Documents\ejemplo>
```

Si observamos atentamente, encontramos un asterisco ( * ) ubicado sobre el nombre “**master”**

Esto nos indica que estamos ubicados sobre la rama de cambios llamada **“master”**

Necesitamos cambiar nuestra rama actual de “**master”** a "**nueva_rama”**

Para esto, ejecutamos el siguiente comando

```
git checkout nueva_rama
```

Este comando nos permite entrar a la rama llamada **“nueva_rama”**

```
PS C:\Users\Dunas\Documents\ejemplo> git checkout nueva_rama
Switched to branch 'nueva_rama'
```

Para comprobarlo nuevamente, usamos el comando `git branch`

```
PS C:\Users\Dunas\Documents\ejemplo> git branch
  master
* nueva_rama
```

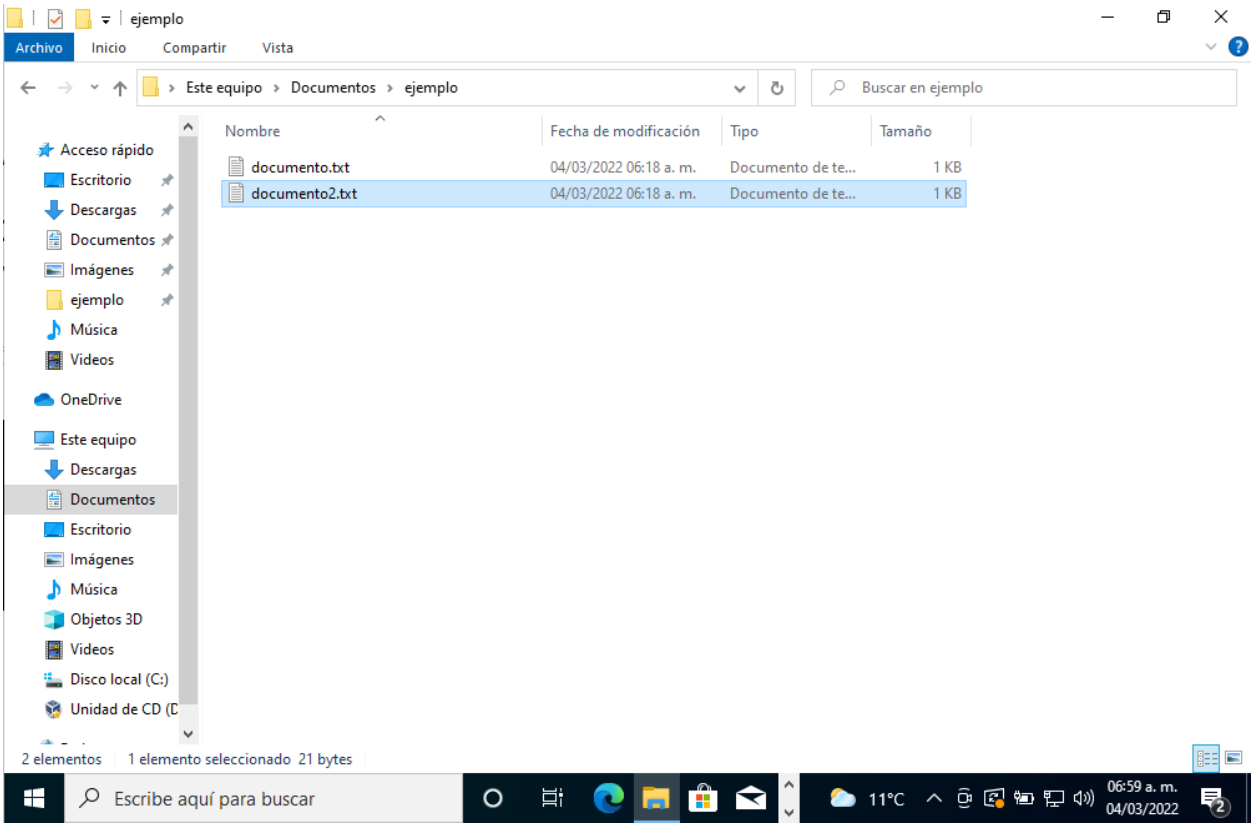
Adicionalmente podemos comprobarlo utilizando el comando `git status`

```
PS C:\Users\Dunas\Documents\ejemplo> git status
On branch nueva_rama
nothing to commit, working tree clean
```

Ahora todos los cambios sobre el proyecto pertenecerán a la rama **“nueva_rama”**

## Cambios sobre nuestra rama

Vamos a hacer una copia de nuestro documento de texto con el nombre de documento2.txt




Y ejecutamos el comando `git status`

```
PS C:\Users\Dunas\Documents\ejemplo> git status
On branch nueva_rama
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        documento2.txt

nothing added to commit but untracked files present (use "git add" to track)
```

💡 Encontramos el **nuevo documento** “documento2.txt” dentro de la sección de “**Untracked files”** ya que es un archivo nuevo para GIT

Utilizamos los comandos `git add` para aceptar los cambios, y el comando `git commit` para registrar los cambios dentro de git

```
git add .
git commit -m "Copia documento de texto llamada documento2"
```

A continuación, usamos el comando `git log` para ver el historial de cambios

```
PS C:\Users\Dunas\Documents\ejemplo> git log
commit c95b70fe936a250f06e7e7d0f44f575b8941abb6 (HEAD -> nueva_rama)
Author: user <user@myemail.com>
Date:   Fri Mar 4 07:23:23 2022 -0600

    Copia de documento de texto llamada documento2

commit e505d173a6d3d4e22cf71a3c07b40b699f5c1fca (master)
Author: user <user@myemail.com>
Date:   Fri Mar 4 06:12:31 2022 -0600

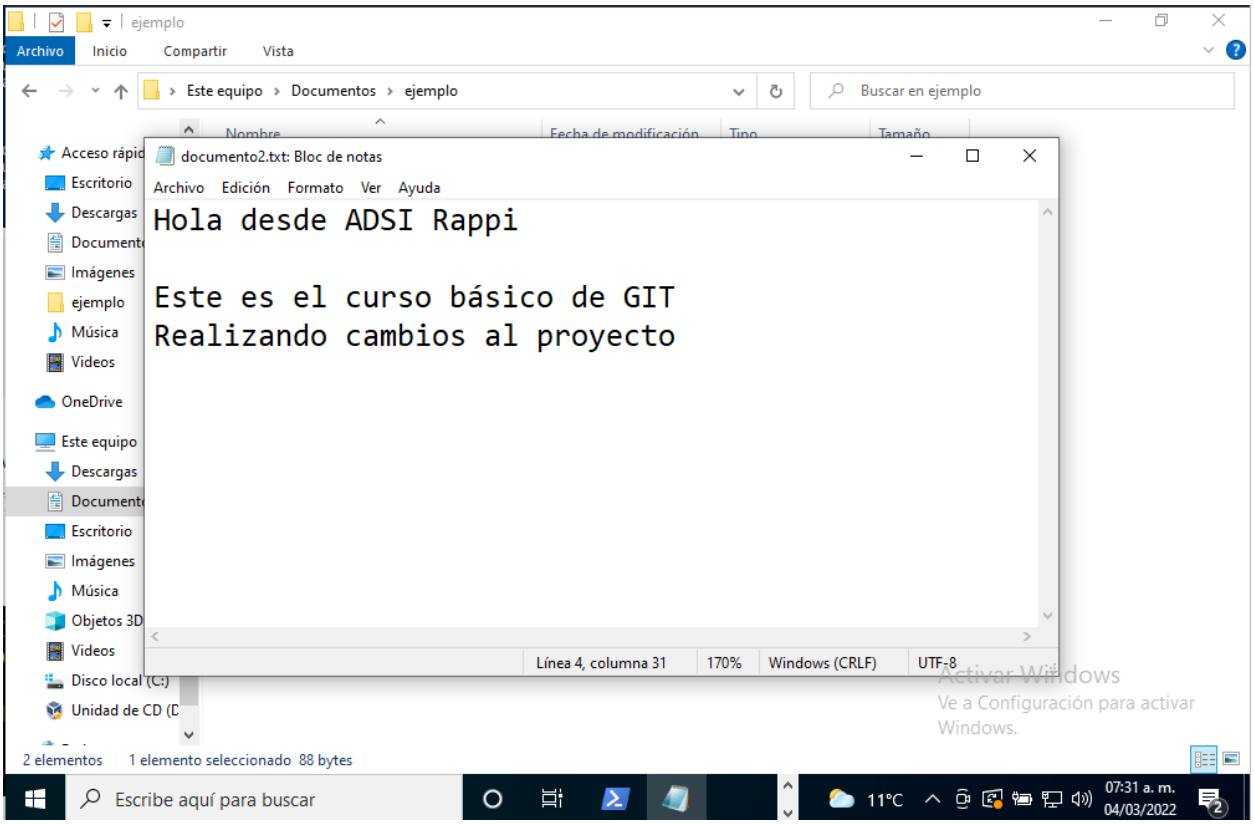
    Nuevo contenido en documento de texto

commit b4da01653d0b1648c0364d37074b9f82502aeb92
Author: user <user@myemail.com>
Date:   Fri Mar 4 05:53:00 2022 -0600

    Nuevo documento de texto
```

Además de la rama, podemos apreciar la palabra **HEAD**, esta palabra indica el lugar en donde **nos encontramos ubicados en el historial.**
Este caso el commit ID ( **c9b770...abb6** ), la rama **“nueva_rama”**, y la palabra **“HEAD”** hacen referencia al mismo lugar dentro del **historial de GIT**

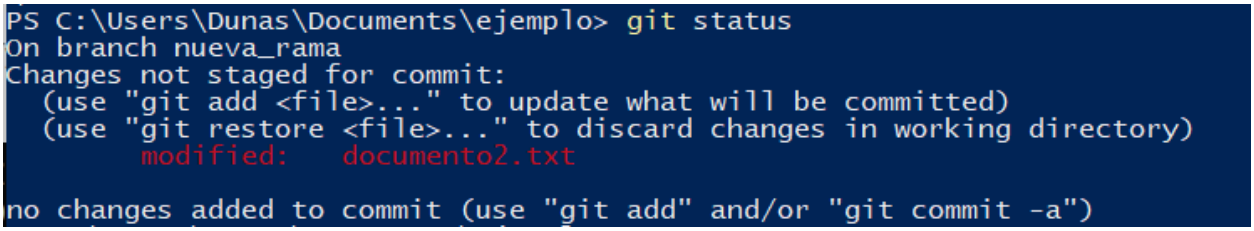
Ahora, vamos a realizar una pequeña modificación a nuestro archivo de texto **“documento2.txt”**




Para ver los cambios rápidamente dentro de nuestra consola usamos `git diff`
Este comando nos permite ver que lineas borramos (con color rojo) y que lineas agregamos dentro de nuestro documento (con color verde).

Ejecutamos el comando

```
git status
```




Podemos ver que existen cambios activos en el documento2, precisamente los cambios que realizamos anteriormente

Para aceptarlos y agregarlos al historial de cambios de GIT, usamos lo siguiente

```
git add .
git commit -m "Cambios a documento2"
```

```
PS C:\Users\Dunas\Documents\ejemplo> git log
commit 0c5a8e05bc2c42def2c6a6afaece1cf5c9877721 (HEAD -> nueva_rama)
Author: user <user@myemail.com>
Date:   Fri Mar 4 07:39:08 2022 -0600

    Cambios a documento2

commit c95b70fe936a250f06e7e7d0f44f575b8941abb6
Author: user <user@myemail.com>
Date:   Fri Mar 4 07:23:23 2022 -0600

    Copia de documento de texto llamada documento2

commit e505d173a6d3d4e22cf71a3c07b40b699f5c1fca (master)
Author: user <user@myemail.com>
Date:   Fri Mar 4 06:12:31 2022 -0600

    Nuevo contenido en documento de texto

commit b4da01653d0b1648c0364d37074b9f82502aeb92
Author: user <user@myemail.com>
Date:   Fri Mar 4 05:53:00 2022 -0600

    Nuevo documento de texto
PS C:\Users\Dunas\Documents\ejemplo>
```

Encontramos 4 commits y una pequeña particularidad

Si prestamos atención a la rama **“master”** se encuentra detrás de los cambios realizados en la rama **“nueva_rama**”

Podemos afirmar que la rama **“master”** está desactualizada con relación a la rama **“nueva_rama”**

Procederemos a la siguiente sección

# Fusionar cambios entre ramas

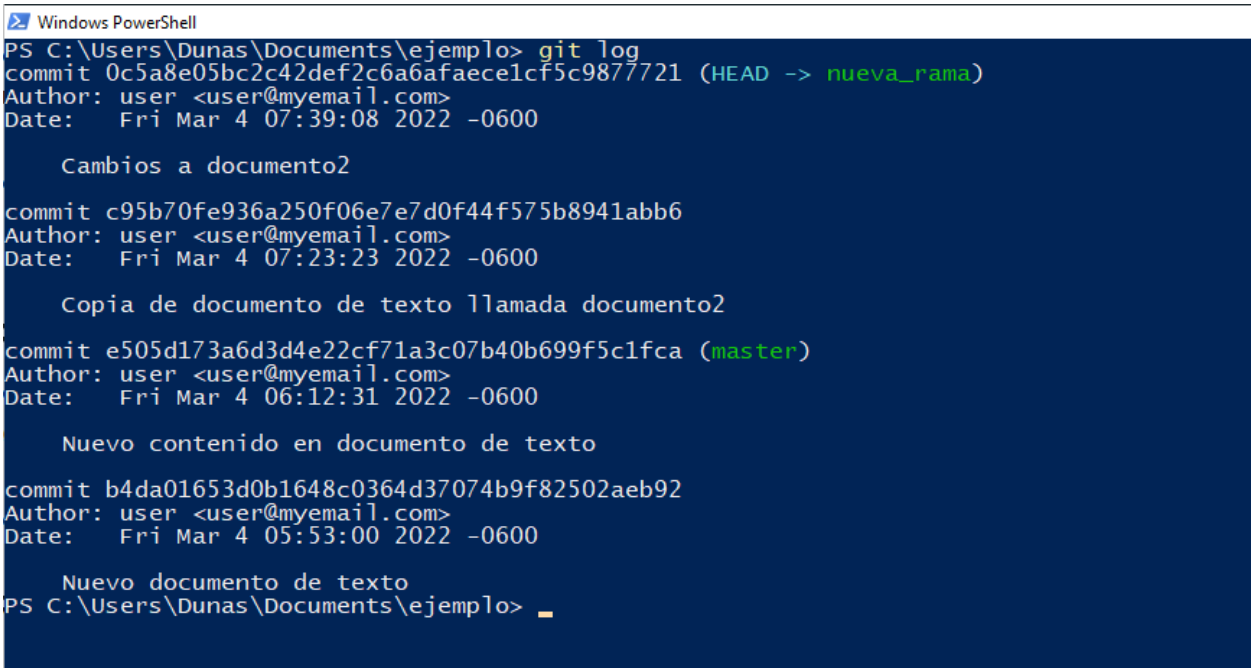
Para actualizar nuestra rama **“master”** debemos hacer una **fusión entre las dos ramas**, la actualizada, junto con la que se encuentra desactualizada

Pero primero, Existe un pequeño concepto que debemos entender antes

# Revisión de cambios a través del historial de GIT

Para poder hacer la revisión de cambios debemos contar con el **nombre de la rama, el ID del commit, o una palabra reservada usada por GIT**

Para obtener cualquiera de estos datos, ejecutamos el comando `git log`




Podemos visualizar diferentes commits, y en algunos de ellos, encontramos palabras como **“HEAD”, “master”, “nueva_rama”**

Estas palabras y los commits ID, son los que podemos utilizar para hacer una revisión de GIT

Para revisar, por ejemplo, primera versión del proyecto, copiamos el commit ID correspondiente al primer commit (El commit de **“Nuevo documento de texto”** de la parte inferior)

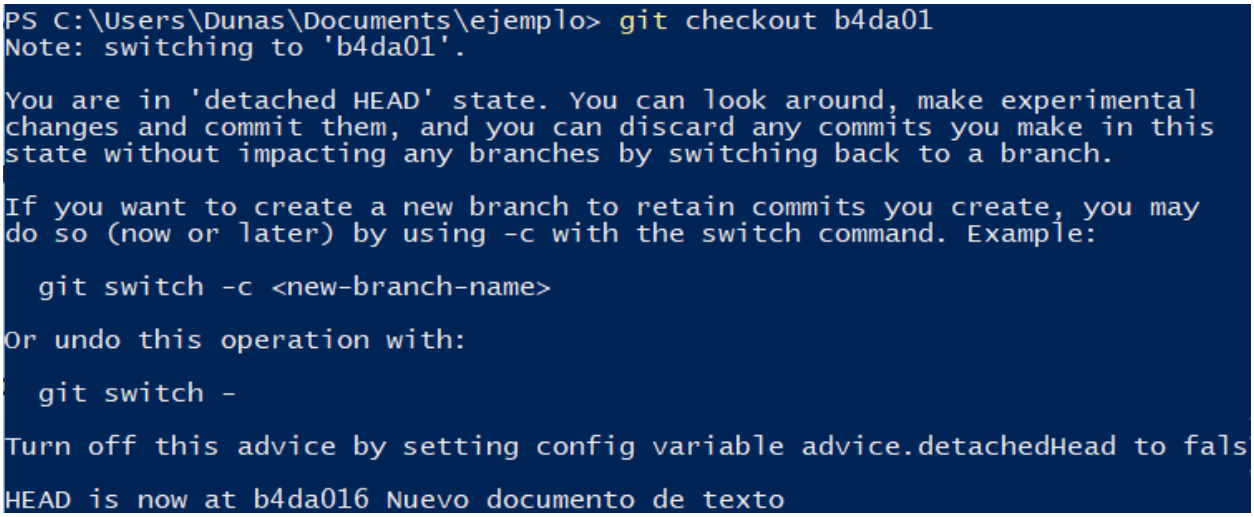
Usamos el siguiente comando

```
git checkout b4da01653d0b1648c0364d37074b9f82502aeb92
```

O simplemente

```
git checkout b4da01
```

Que es la versión corta del commit ID




Nos saldrá el siguiente mensaje de confirmación, indicándonos que **no nos encontramos en ninguna rama del proyecto**, por lo que nuestros cambios no seran completamente persistentes a no ser, que al final de todo el proceso de cambios y commits, usemos el comando `git switch -c nuevos_cambios` reemplazando **“nuevos_cambios”** por el nombre que desees.

Podemos ver también una advertencia ejecutando el comando `git status`

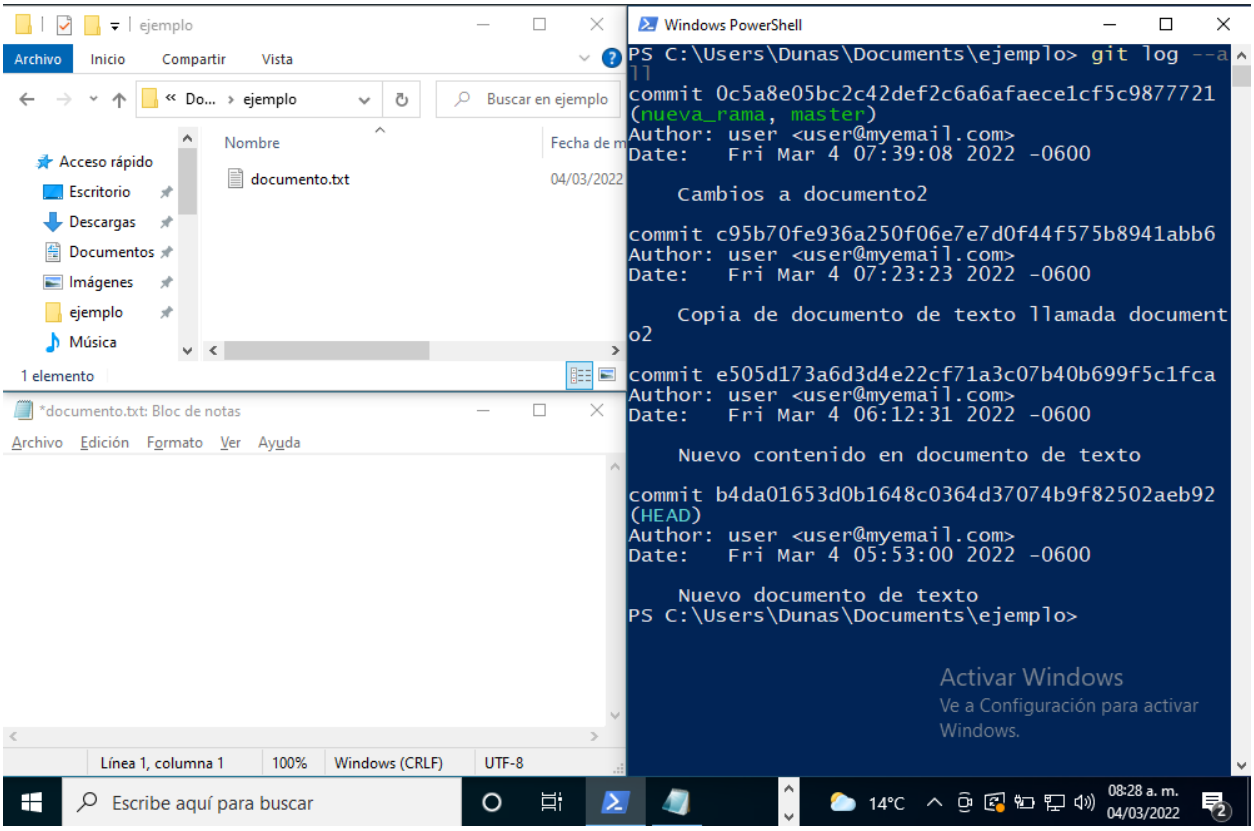
```
PS C:\Users\Dunas\Documents\ejemplo> git status
HEAD detached at b4da016
nothing to commit, working tree clean
PS C:\Users\Dunas\Documents\ejemplo>
```

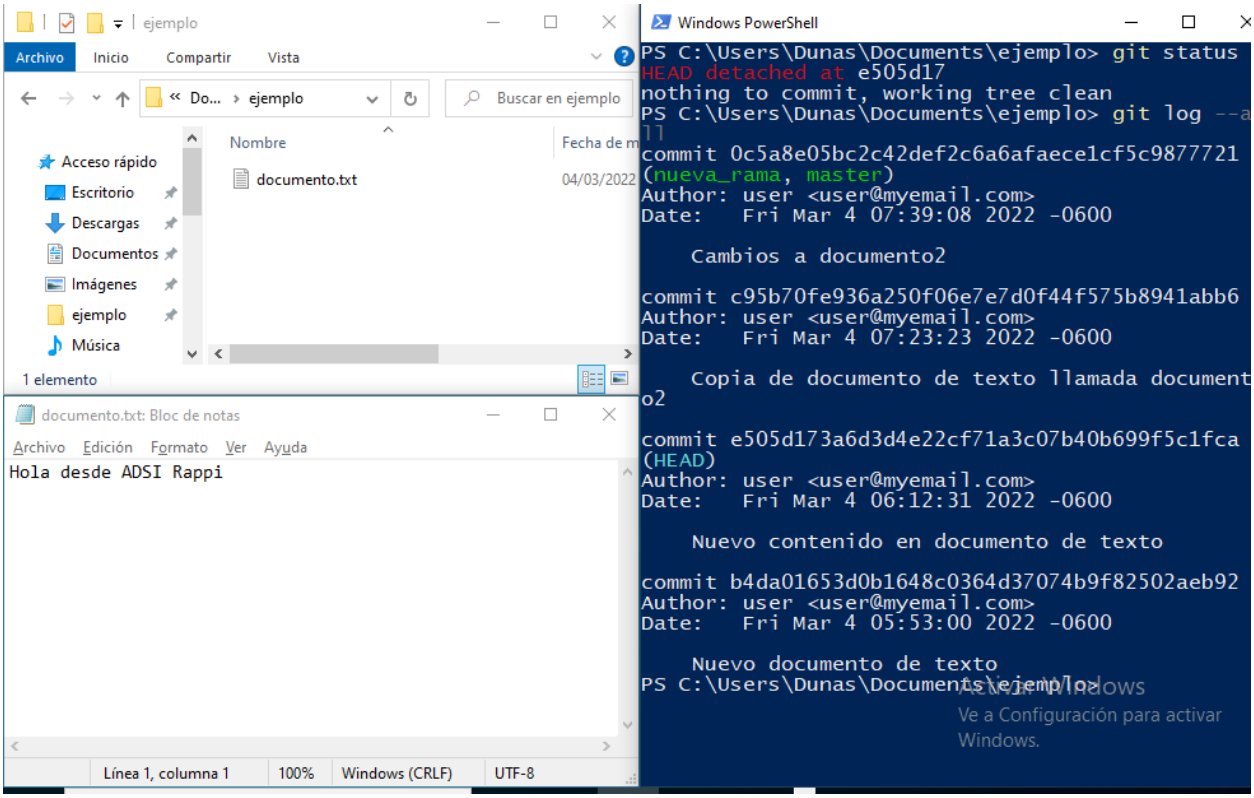
Como nos movimos a una versión del proyecto anterior dentro del historial de GIT, podemos encontrar que el proyecto nos muestra **todos los archivos tal y como les hicimos el registro a GIT en ese momento**, es decir, nos regresamos a un punto en el tiempo en donde no tenemos “documento2.txt” creado, ni hemos agregado cambios a “documento.txt” aún

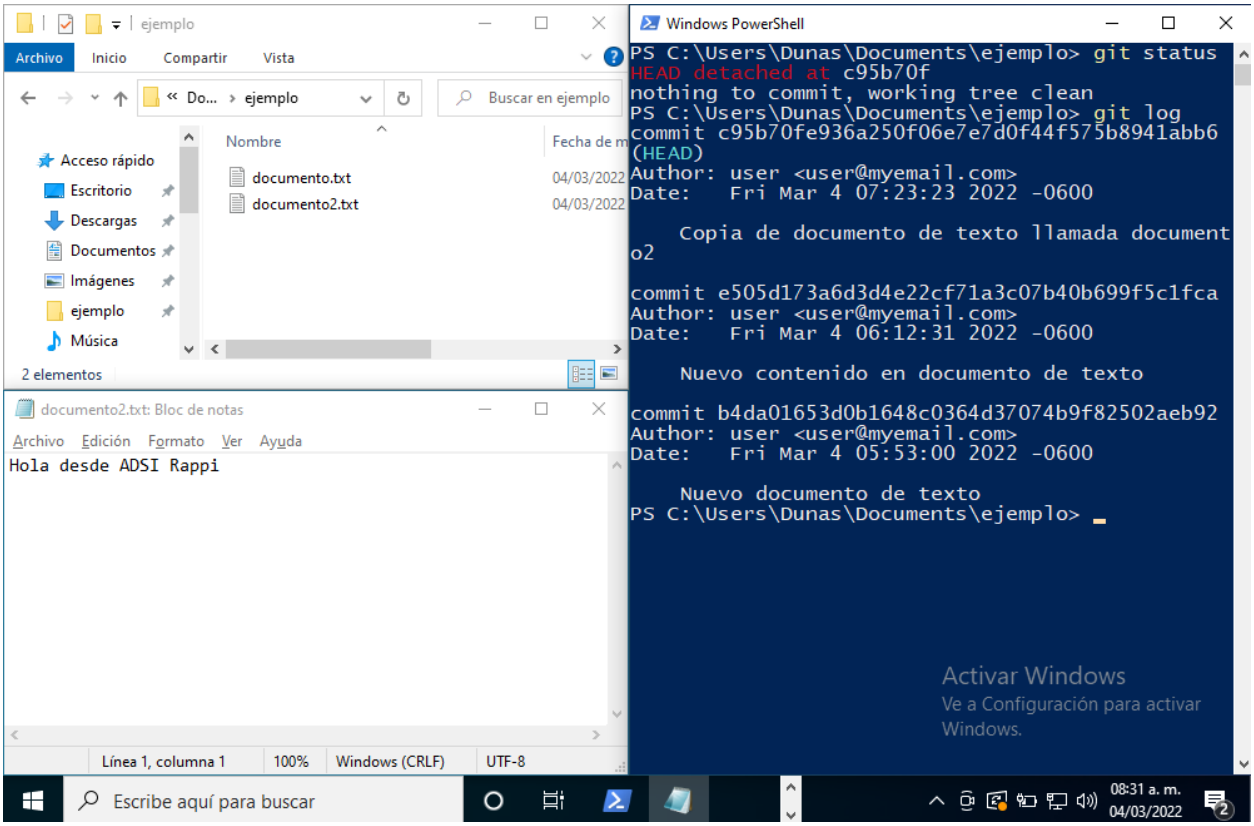


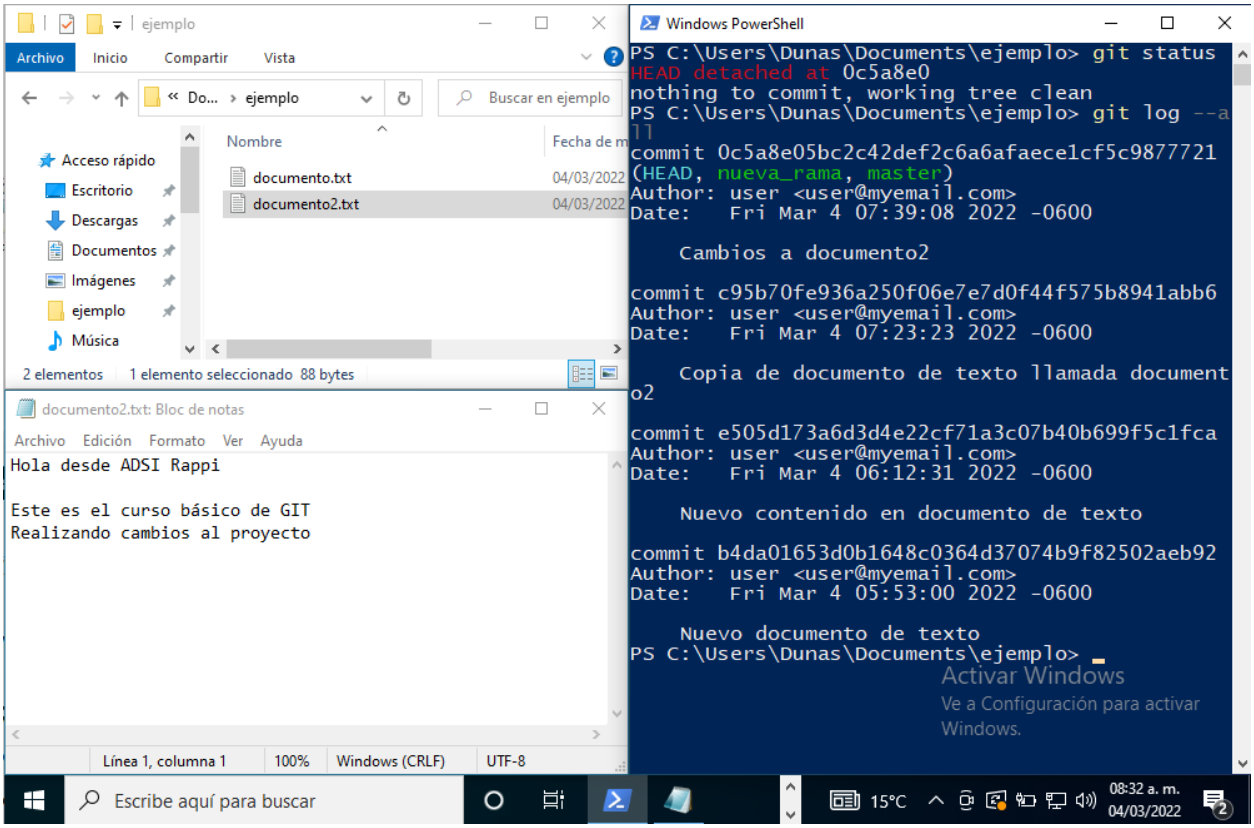
Si ejecutamos `git log` veremos que solo nos mostrará los commits anteriores.
Para ver todos los commits, utilizamos el comando `git log --all`

Navegando rápidamente por los commits, podemos ver el progreso de nuestro proyecto sobre la misma carpeta


ejemplo
Archivo Inicio Compartir Vista
Do... › ejemplo
Buscar en ejemplo
Acceso rápido
Escritorio
Descargas
Documentos
Imágenes
ejemplo
Música
Nombre
documento.txt
04/03/2022
1 elemento
*documento.txt: Bloc de notas
Archivo Edición Formato Ver Ayuda
Línea 1, columna 1 100% Windows (CRLF) UTF-8
Windows PowerShell
PS C:\Users\Dunas\Documents\ejemplo> git log --all
commit 0c5a8e05bc2c42def2c6a6afaece1cf5c9877721 (nueva_rama, master)
Author: user <user@myemail.com>
Date: Fri Mar 4 07:39:08 2022 -0600
Cambios a documento2
commit c95b70fe936a250f06e7e7d0f44f575b8941abb6
Author: user <user@myemail.com>
Date: Fri Mar 4 07:23:23 2022 -0600
Copia de documento de texto llamada document o2
commit e505d173a6d3d4e22cf71a3c07b40b699f5c1fca
Author: user <user@myemail.com>
Date: Fri Mar 4 06:12:31 2022 -0600
Nuevo contenido en documento de texto
commit b4da01653d0b1648c0364d37074b9f82502aeb92 (HEAD)
Author: user <user@myemail.com>
Date: Fri Mar 4 05:53:00 2022 -0600
Nuevo documento de texto
PS C:\Users\Dunas\Documents\ejemplo>
Activar Windows
Ve a Configuración para activar Windows.
Escribe aquí para buscar
14°C
08:28 a. m.
04/03/2022


ejemplo
Archivo Inicio Compartir Vista
Do... › ejemplo
Buscar en ejemplo
Acceso rápido
Escritorio
Descargas
Documentos
Imágenes
ejemplo
Música
Nombre
documento.txt
04/03/2022
1 elemento
documento.txt: Bloc de notas
Archivo Edición Formato Ver Ayuda
Hola desde ADSI Rappi
Línea 1, columna 1 100% Windows (CRLF) UTF-8
Windows PowerShell
PS C:\Users\Dunas\Documents\ejemplo> git status
HEAD detached at e505d17
nothing to commit, working tree clean
PS C:\Users\Dunas\Documents\ejemplo> git log --all
commit 0c5a8e05bc2c42def2c6a6afaece1cf5c9877721 (nueva_rama, master)
Author: user <user@myemail.com>
Date: Fri Mar 4 07:39:08 2022 -0600
Cambios a documento2
commit c95b70fe936a250f06e7e7d0f44f575b8941abb6
Author: user <user@myemail.com>
Date: Fri Mar 4 07:23:23 2022 -0600
Copia de documento de texto llamada document o2
commit e505d173a6d3d4e22cf71a3c07b40b699f5c1fca (HEAD)
Author: user <user@myemail.com>
Date: Fri Mar 4 06:12:31 2022 -0600
Nuevo contenido en documento de texto
commit b4da01653d0b1648c0364d37074b9f82502aeb92
Author: user <user@myemail.com>
Date: Fri Mar 4 05:53:00 2022 -0600
Nuevo documento de texto
PS C:\Users\Dunas\Documents\ejemplo>
Activar Windows
Ve a Configuración para activar Windows.


ejemplo
Archivo
Inicio
Compartir
Vista
Do... › ejemplo
Buscar en ejemplo
Acceso rápido
Escritorio
Descargas
Documentos
Imágenes
ejemplo
Música
Nombre
Fecha de m
documento.txt
04/03/2022
documento2.txt
04/03/2022
2 elementos
documento2.txt: Bloc de notas
Archivo Edición Formato Ver Ayuda
Hola desde ADSI Rappi
Línea 1, columna 1
100%
Windows (CRLF)
UTF-8
Windows PowerShell
PS C:\Users\Dunas\Documents\ejemplo> git status
HEAD detached at c95b70f
nothing to commit, working tree clean
PS C:\Users\Dunas\Documents\ejemplo> git log
commit c95b70fe936a250f06e7e7d0f44f575b8941abb6
(HEAD)
Author: user <user@myemail.com>
Date: Fri Mar 4 07:23:23 2022 -0600
Copia de documento de texto llamada document
o2
commit e505d173a6d3d4e22cf71a3c07b40b699f5c1fca
Author: user <user@myemail.com>
Date: Fri Mar 4 06:12:31 2022 -0600
Nuevo contenido en documento de texto
commit b4da01653d0b1648c0364d37074b9f82502aeb92
Author: user <user@myemail.com>
Date: Fri Mar 4 05:53:00 2022 -0600
Nuevo documento de texto
PS C:\Users\Dunas\Documents\ejemplo>
Activar Windows
Ve a Configuración para activar
Windows.
Escribe aquí para buscar
08:31 a. m.
04/03/2022




Vamos a regresar a la sección principal, y continuar con el proceso

## Requisitos para hacer una fusión de dos ramas de GIT

Para hacer esta fusión es necesario tener en cuenta:

- Debemos ubicarnos sobre la rama que queremos **actualizar con otros cambios**, en este caso debemos ubicarnos en **“master”** y pedir los cambios de **“nueva_rama”**
- **No debe de existir ningún cambio dentro de la rama actual**, para evitar que los datos que no he registrado como commit dentro de GIT, se pierdan

Para empezar esta fusión, nos ubicamos primero en la rama **“master”** con el siguiente comando

```
git checkout master
```

```
PS C:\Users\Dunas\Documents\ejemplo> git checkout master
Previous HEAD position was b4da016 Nuevo documento de texto
Switched to branch 'master'
```

💡 Siguiendo con el proceso, podemos ver un mensaje que nos notifica que nos encontrábamos en el commit **b4da016**, y que nos hemos transladado a la rama **“master”**

Para realizar la fusión, solo hace falta un paso

```
git merge rama_nueva
```

Con este comando le pedimos a GIT que nos traiga los cambios de **"nueva_rama”** y los integre sobre nuestra rama actual **“master”**

```
PS C:\Users\Dunas\Documents\ejemplo> git merge nueva_rama
Updating e505d17..0c5a8e0
Fast-forward
 documento2.txt | 4 ++++
 1 file changed, 4 insertions(+)
 create mode 100644 documento2.txt
```

A continuación podemos ver que se ha realizado una actualización desde **e505d17** (rama “**master”**), hasta **0c5a8e0** (rama **“nueva_rama”**)

Si ejecutamos el comando `git log` encontraremos que la rama **“master”** ahora se encuentra en la misma ubicación que la rama **“nueva_rama”**

```
PS C:\Users\Dunas\Documents\ejemplo> git log
commit 0c5a8e05bc2c42def2c6a6afaece1cf5c9877721 (HEAD -> master, nueva_rama)
Author: user <user@myemail.com>
Date:   Fri Mar 4 07:39:08 2022 -0600

    Cambios a documento2

commit c95b70fe936a250f06e7e7d0f44f575b8941abb6
Author: user <user@myemail.com>
Date:   Fri Mar 4 07:23:23 2022 -0600

    Copia de documento de texto llamada documento2

commit e505d173a6d3d4e22cf71a3c07b40b699f5c1fca
Author: user <user@myemail.com>
Date:   Fri Mar 4 06:12:31 2022 -0600

    Nuevo contenido en documento de texto

commit b4da01653d0b1648c0364d37074b9f82502aeb92
Author: user <user@myemail.com>
Date:   Fri Mar 4 05:53:00 2022 -0600

    Nuevo documento de texto
PS C:\Users\Dunas\Documents\ejemplo>
```

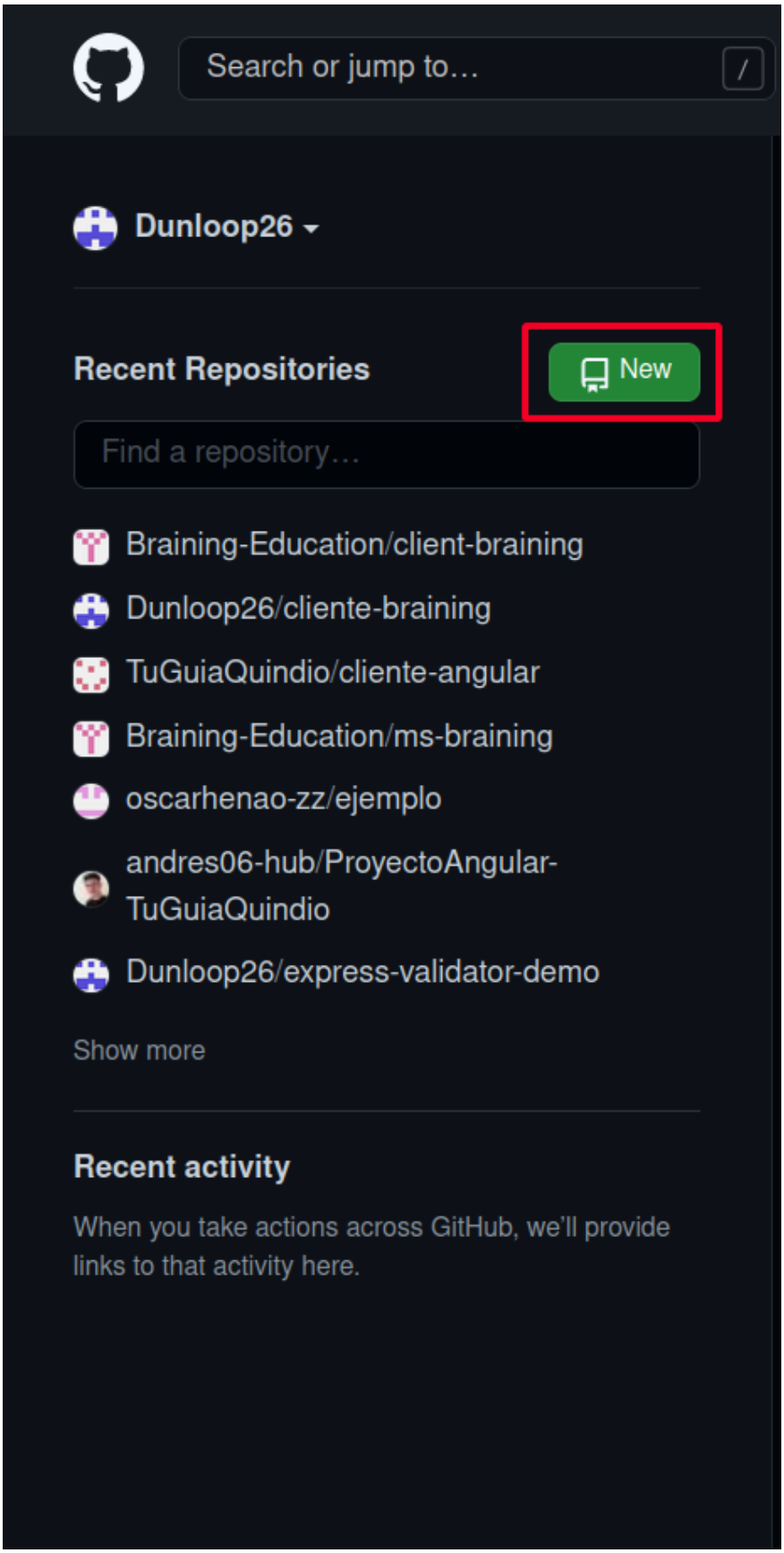
Podemos ver que ahora ambas ramas se encuentran en el mismo lugar.

# Registrar un repositorio remoto de GitHub dentro de nuestro proyecto de GIT

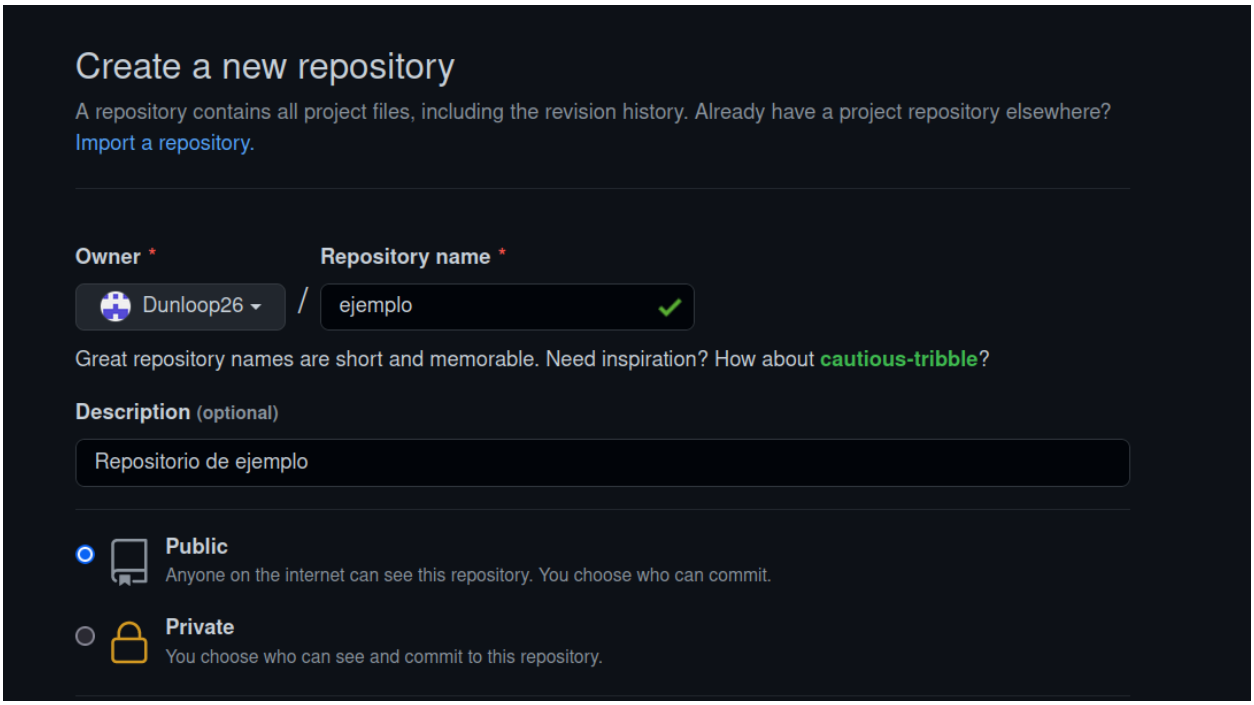
## Creando nuestro proyecto remoto en GitHub

Para empezar iniciaremos sesión en nuestra cuenta de GitHub en https://www.github.com

Creamos un proyecto nuevo de GIT dentro de GitHub.

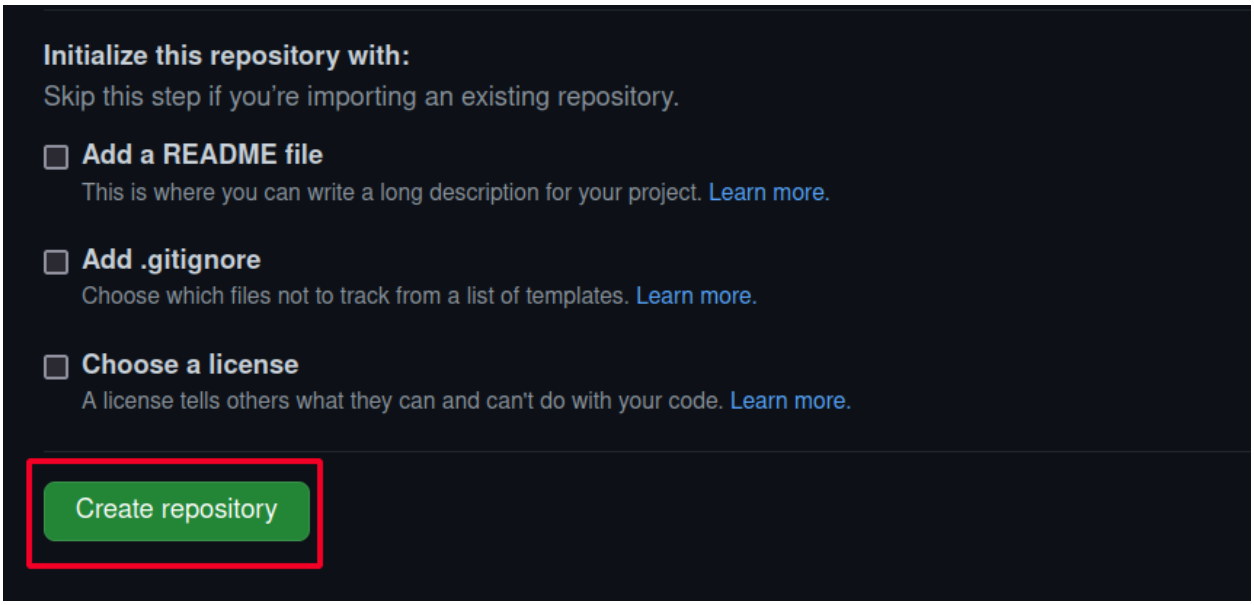

Search or jump to…
/
Dunloop26
Recent Repositories
New
Find a repository…
Braining-Education/client-braining
Dunloop26/cliente-braining
TuGuiaQuindio/cliente-angular
Braining-Education/ms-braining
oscarhenao-zz/ejemplo
andres06-hub/ProyectoAngular-TuGuiaQuindio
Dunloop26/express-validator-demo
Show more
Recent activity
When you take actions across GitHub, we'll provide links to that activity here.

Configuramos nuestro proyecto dentro de GitHub, esta es la configuración de ejemplo

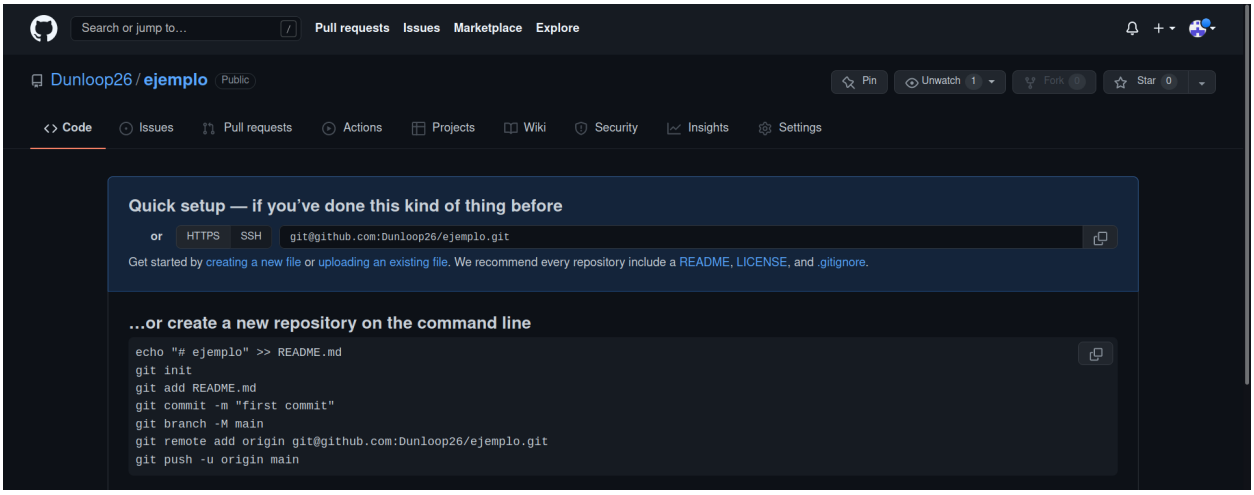



Sin README, sin .gitignore, y sin elegir licencia

Damos clic en **Create repository**




Al crear el repositorio veremos lo siguiente




Ya estamos listos para registrar el remoto dentro de nuestro proyecto de GIT Local

## Registrando el remoto en nuestro proyecto de GIT local

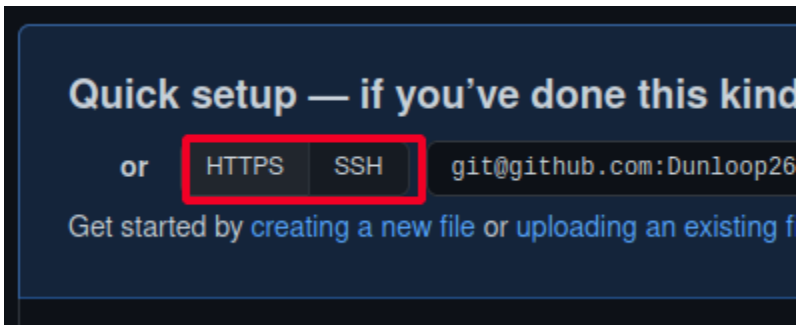
Este paso es muy importante si queremos que nuestro proyecto pueda mantener una copia en la nube

## Obteniendo la URL del proyecto de GitHub

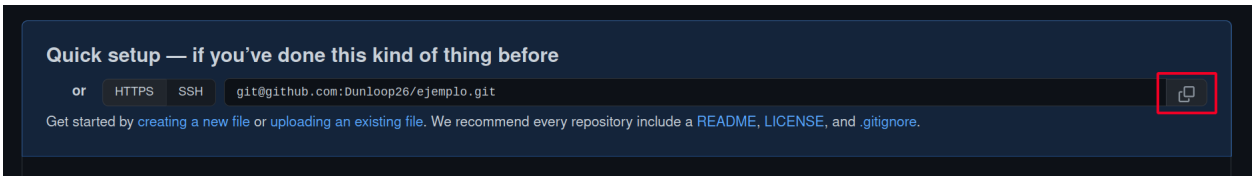
Primero, debemos elegir entre **HTTPS** o **SSH**

- **HTTPS:** Esta opción nos permite clonar y subir nuestro proyecto de GitHub de manera sencilla, usando usuario y contraseña, o PAT (Personal Access Token) distribuido por el administrador del proyecto
- **SSH:** Esta opción nos permite usar nuestra llave SSH registrada como medio de autenticación con GitHub sin necesidad de proveer credenciales cada vez

> 💡 Solo usaremos **SSH** si hemos realizado los pasos anteriores para registrar una nueva llave SSH, de otra forma, elegimos **HTTPS**




Copiamos la URL que encontramos a la derecha después de elegir nuestra opción




Para este caso corresponde a **"git@github.com:Dunloop26/ejemplo.git"**

## Registrando el remoto

Para iniciar el registro de nuestro proyecto en la nube usamos el comando

```
git remote add <nombre_remoto> <url_https|url_ssh>
```

En este caso utilizaremos el nombre de remoto **“origin”** y la url ssh **”git@github.com:Dunloop26/ejemplo.git”**

```
git remote add origin git@github.com:Dunloop26/ejemplo.git
```

> ⚠️ Debemos reemplazar la url por la que obtenemos desde la página inicial de nuestro proyecto de GIT

Al hacer esto, podemos realizar operaciones comunes de interacción con la nube como `git push` `git pull` `git fetch` entre otros

Para subir nuestro proyecto por primera vez, nos aseguramos de renombrar la rama a **”main”** con

```
git branch -M main
```

Una vez hecho esto, procedemos a subir nuestro proyecto con el siguiente comando

```
git push -u origin main
```

💡 El comando `git push` acompañado de la opción `-u` crea un enlace desde nuestro proyecto local **main**, hasta el remoto de origen **main**, lo cual, nos permite hacer operaciones comunes `git pull` `git push` sin la necesidad de especificar una rama, ya que con el enlace activo `local:main -> remoto:main` esto se hace de manera automática.

# CMMI

Marca externa



GC-F-004v.04



# ¿QUÉ ES CMMI?

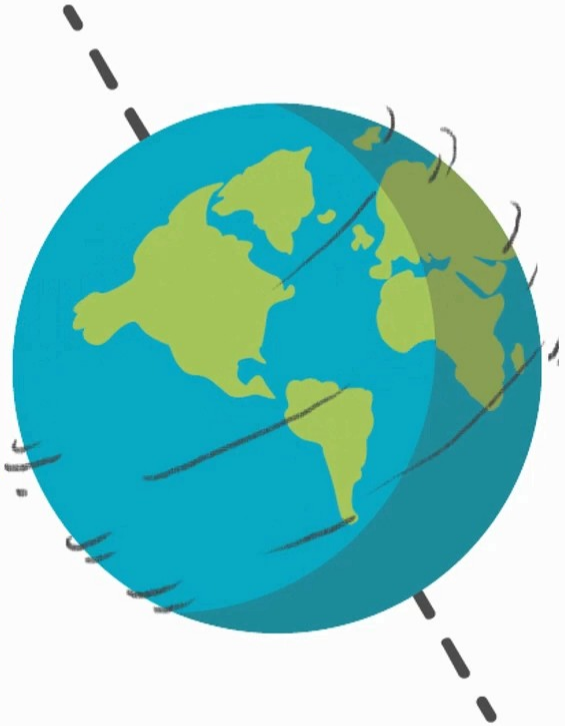
## CAPABILITY MATURITY MODEL INTEGRATION (MODELO DE MADUREZ DE CAPACIDADES DE INTEGRACIÓN)

CMMI ES UN MODELO QUE CONTIENE LAS MEJORES PRÁCTICAS Y QUE PROVEE A LAS ORGANIZACIONES DE AQUELLOS ELEMENTOS QUE SON ESENCIALES PARA QUE LOS **PROCESOS DE NEGOCIO** DE LAS MISMAS SEAN EFECTIVOS.



# CCMI NACE A FINALES DE LOS AÑOS 80

## EN LA INDUSTRIA MILITAR DE EU Y AMÉRICA LATINA



SU PRIMERA FINALIDAD INICIAL FUE LA DE **EVALUAR LOS PROCESOS** DE SOFTWARE MÁS CRÍTICOS.



# 1991

## SE CREÓ POR PRIMERA VEZ EL MODELO CMMI PARA SOFTWARE

LA PROLIFERACIÓN DE NUEVOS MODELOS DIO LUGAR A CONFUSIÓN, POR LO QUE EL GOBIERNO DECIDIÓ FINANCIAR UN PROYECTO DE 2 AÑOS A FIN DE CREAR UN SOLO MARCO EXTENSIBLE PARA

INGENIERÍA DE SISTEMAS

INGENIERIA DE SOFTWARE

Y EL DESARROLLO DE PRODUCTOS



# ANTECEDENTES

**1984**

El congreso del gobierno Americano aprueba la creación de un organismo de investigación para el desarrollo de modelos de mejora en el Desarrollo de sistemas de software

**1985**

SEI (Software Engineering Institute) empieza a trabajar en un marco de madurez de procesos para evaluar a las empresas productoras de software.

SEI publica la versión 1.0 del modelo de madurez de capacidades para el software (SW-CMM, Capability Maturity Model for Software)

1991

1993

SEI publica la version 1.1 de SW-CMM

Publicación de la version 1.2

1997

2000

SW-CMM fue integrado y relevado por el nuevo modelo CMMI

2005

Limite para migrar de SW-CMMI a CMMI

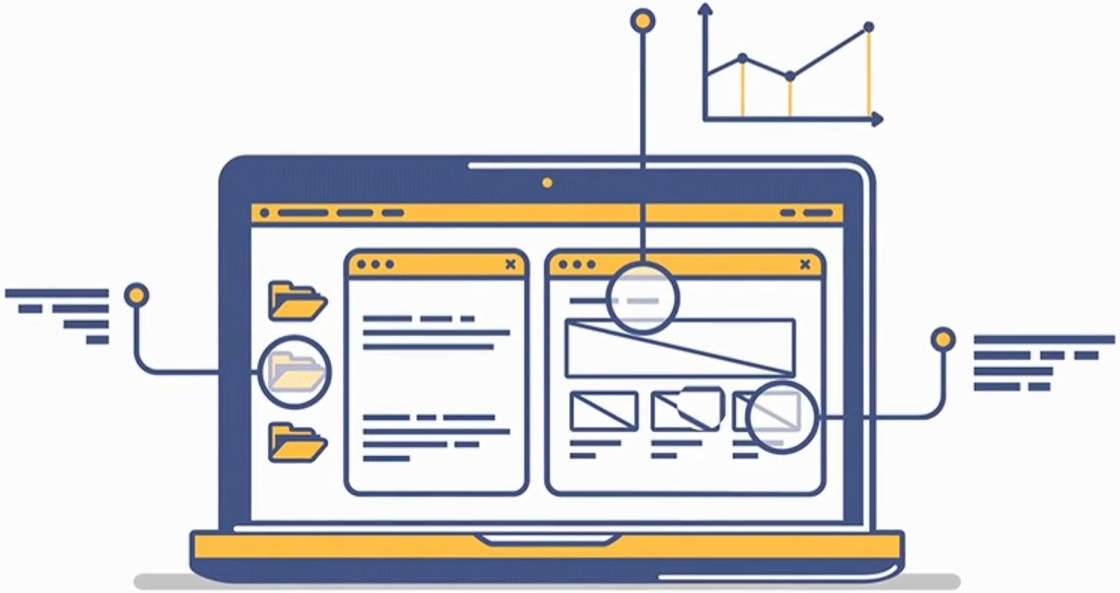
2006

Se libera la version CMMI para el Desarrollo (DEV-CMMI)

Se libera CMMI para la adquisición. (ACQ-CMMI)

2007

Actualidad



EL MODELO CMMI SE USA PARA EVALUAR EL NIVEL DE MADUREZ DE UNA COMPAÑÍA EN TÉRMINOS DE DESARROLLO INFORMÁTICO.

# CMMI TIENE COMO OBJETIVO:

**ALENTAR A LAS COMPAÑÍAS PARA QUE MONITOREEN Y MEJOREN CONTINUAMENTE SUS PROCESOS, Y EVALÚEN EL NIVEL DE MADUREZ DE LOS MISMOS EN UNA ESCALA DE CINCO NIVELES ESTABLECIDA POR EL CMMI.**

EN LA ACTUALIDAD HAY DOS ÁREAS DE INTERÉS CUBIERTAS POR LOS MODELOS CMMI

DESARROLLO

ADQUISIÓN

# OBJETIVOS CMMI



EL MODELO TIENE

# 4 ÁREAS DE CONOCIMIENTO




ING.SISTEMAS

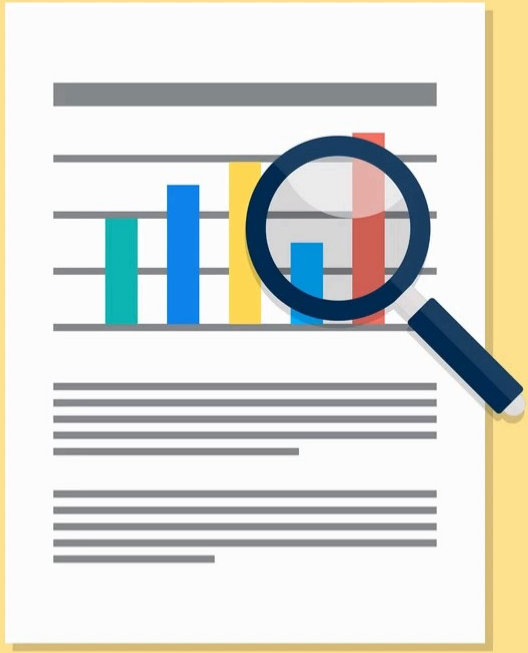
ACUERDOS CON PROVEEDORES

ING.SOFTWARE

IPPD

# CARACTERÍSTICAS

LAS PRÁCTICAS QUE DEBEN SER REALIZADAS POR CADA ÁREA CLAVE DE PROCESO ESTÁN ORGANIZADAS EN 5 CARACTERÍSTICAS COMUNES, LAS CUALES CONSTITUYEN PROPIEDADES QUE INDICAN SI LA IMPLEMENTACIÓN Y LA INSTITUCIONALIZACIÓN DE UN PROCESO CLAVE ES EFECTIVO, REPETIBLE Y DURADERO.



# ESTAS 5 CARACTERISTICAS SON:




01 COMPROMISO DE LA REALIZACIÓN

02 LA CAPACIDAD DE REALIZACIÓN

03 LAS ACTIVIDADES REALIZADAS

04 LAS MEDICIONES Y EL ANÁLISIS

05 LA VERIFICACIÓN DE LA IMPLEMENTACIÓN

# DESVENTAJAS

- FALTA DE ADECUACIÓN AL ENFOQUE DE SERVICIOS QUE EXPERIMENTA EL SECTOR DE LAS TI (PROCESOS DE DESARROLLO DE PRODUCTOS DE SOFTWARE)
- ALTO ESFUERZO DE IMPLEMENTACIÓN QUE EXIGE
- EL PROCESO DE EVALUACIÓN ES MUY COSTOSO EN TIEMPO Y ESFUERZO.
- LA COMPLEJIDAD DE LA EVALUACIÓN PUEDE ATENTAR CONTRA LA DEFINICIÓN DE OBJETIVOS CONCRETOS DE LA EMPRESA



# VENTAJAS

- REDUCCIÓN DEL COSTE DE DESARROLLO.
- LOCALIZACIÓN Y RESOLUCIÓN DE DEFECTOS.
- MEJORA EN LA FIABILIDAD DE LA PLANIFICACIÓN.
- **AUMENTO DE LA PRODUCTIVIDAD.**
- REDUCCIÓN DE LOS TRABAJOS DERIVADOS DE CORRECCIONES.
- AUMENTO DE LA EFECTIVIDAD SOBRE LA PLANIFICACIÓN REALIZADA.
- **MEJORA EN LA CALIDAD DEL PRODUCTO.**
- SE PUEDE APLICAR A MÁS DE UNA DISCIPLINA.
- MEJOR ATENCIÓN A LAS ÁREAS DE INGENIERÍA
- MEJORA DE LA COMUNICACIÓN PARA QUE CADA PARTICIPANTE CUMPLA CON SUS RESPONSABILIDADES
- MEJORA LA PLANIFICACIÓN
- **LOS CLIENTES VIVEN MÁS INFORMADOS**



# ¿EN QUE SE DIFERENCIA

# CMMI

# DE OTROS MODELOS



SE DIFERENCIA DE OTROS MODELOS POR EL HECHO DE ESTAR BASADO EN PRÁCTICAS AJUSTABLES A **CUALQUIER** DOMINIO DE PRODUCCIÓN




Y POSEER UN ENFOQUE GLOBAL E INTEGRADO DE LA ORGANIZACIÓN, CON EL PROPÓSITO DE

**ALCANZAR LOS OBJETIVOS DEL NEGOCIO.**



CMMI PERMITE A EMPRESAS COMPLEJAS COMPUESTAS POR VARIAS ÁREAS DE NEGOCIO INSTAURAR DE UNA FORMA MÁS SENCILLA UN SISTEMA DE **ASEGURAMIENTO DE LA CALIDAD**.

EL CCMI TIENE **DOS REPRESENTACIONES**, ESTAS REPRESENTACIONES PERMITEN A LA ORGANIZACIÓN PERSEGUIR DIFERENTES OBJETIVOS DE MEJORA.

LA PRESENTACIÓN Y ORGANIZACIÓN DE LA INFORMACIÓN ES DIFERENTE PARA CADA UNA, SIN EMBARGO, EL CONTENIDO ES EL MISMO.



# POR ETAPAS (STAGED)

MIDEN LA MADUREZ DE UN GRUPO DE PROCESOS DENTRO DE UNA ORGANIZACIÓN





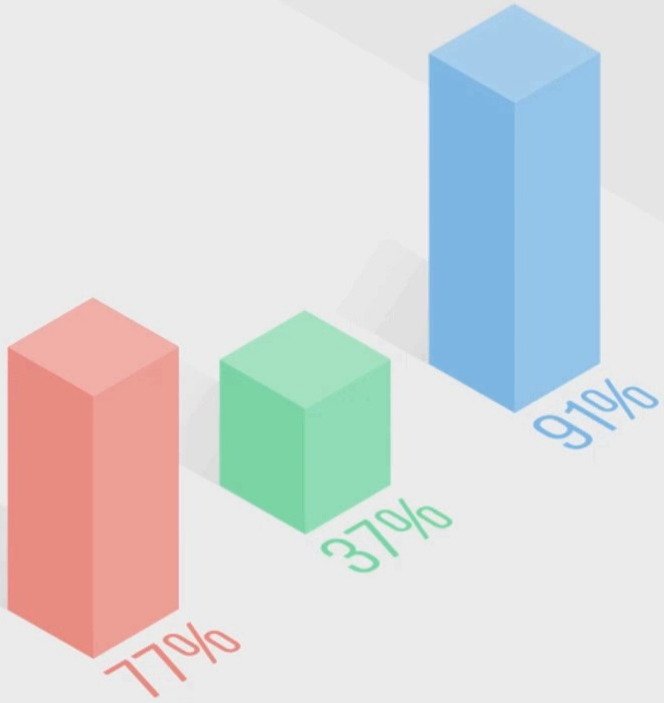


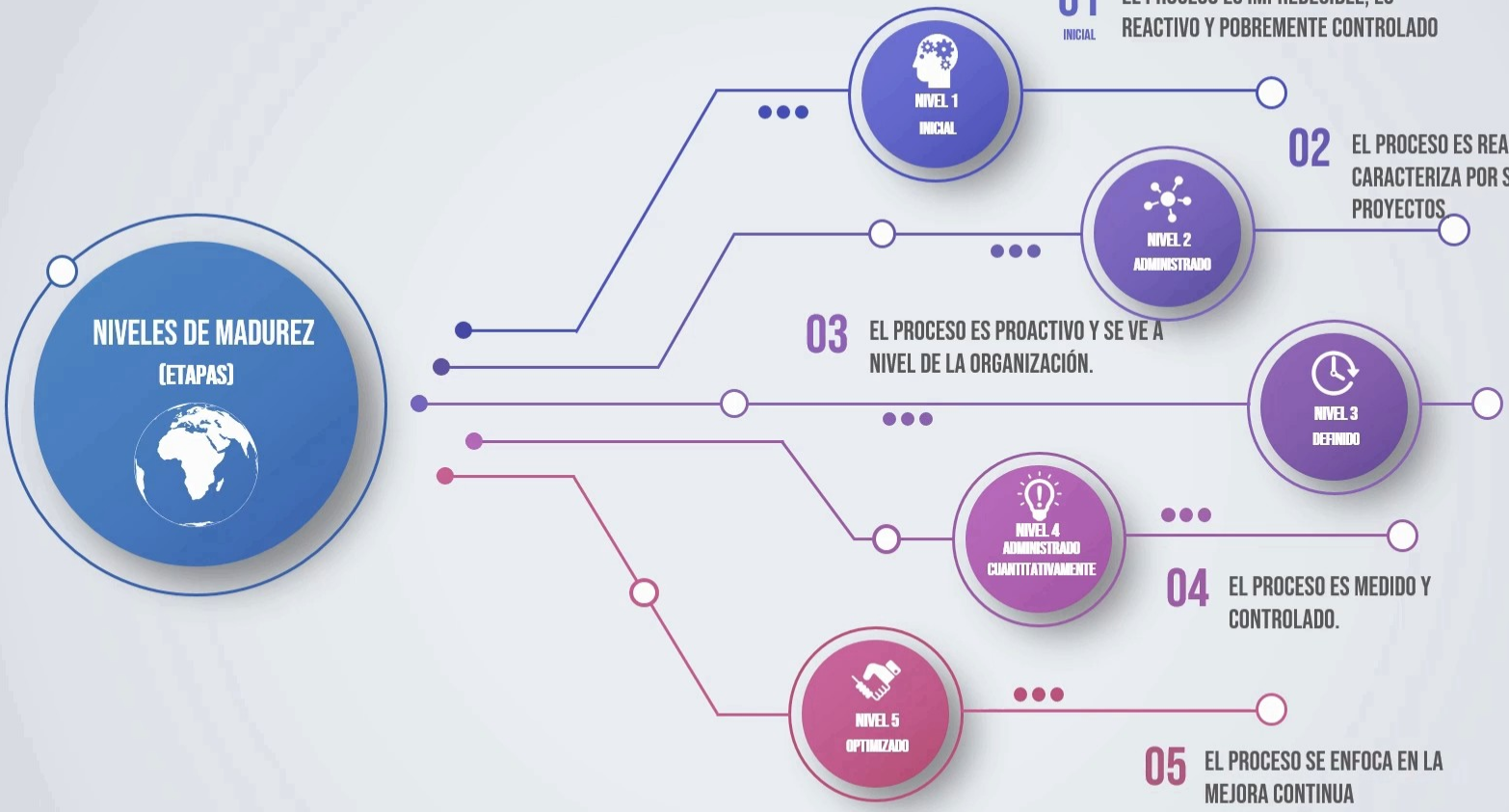


# CONTINUO

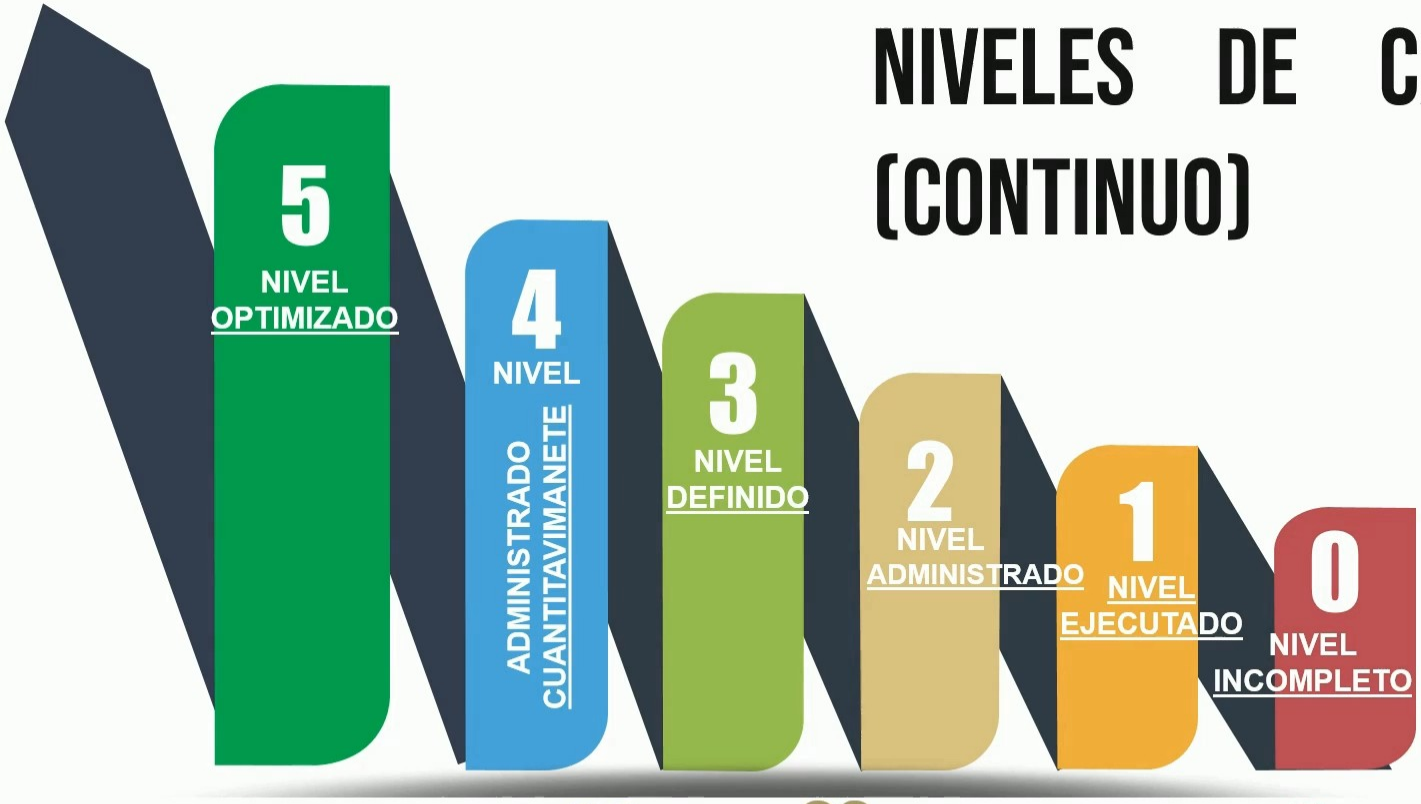
MIDEN LA MADUREZ DE UN PROCESO EN PARTICULAR DENTRO DE UNA ORGANIZACIÓN

# 5 NIVELES DE CAPACIDAD




NIVELES DE MADUREZ
(ETAPAS)
NIVEL 1
INICIAL
NIVEL 2
ADMINISTRADO
NIVEL 3
DEFINIDO
NIVEL 4
ADMINISTRADO
CUANTITATIVAMENTE
NIVEL 5
OPTIMIZADO
01
INICIAL
EL PROCESO ES IMPREDECIBLE, ES REACTIVO Y POBREMENTE CONTROLADO
02
EL PROCESO ES REACTIVO Y SE CARACTERIZA POR SU APLICACIÓN A PROYECTOS
03
EL PROCESO ES PROACTIVO Y SE VE A NIVEL DE LA ORGANIZACIÓN.
04
EL PROCESO ES MEDIDO Y CONTROLADO.
05
EL PROCESO SE ENFOCA EN LA MEJORA CONTINUA

# NIVELES DE CAPACIDAD (CONTINUO)




**05 OPTIIMIZADO**
El proceso se enfoca en la mejora continua.

**04 ADMINISTRADO CUANTITATIVAMENTE**
El proceso es medido y controlado.

**03 DEFINIDO**
El proceso es proactivo y se ve a nivel de la organización.

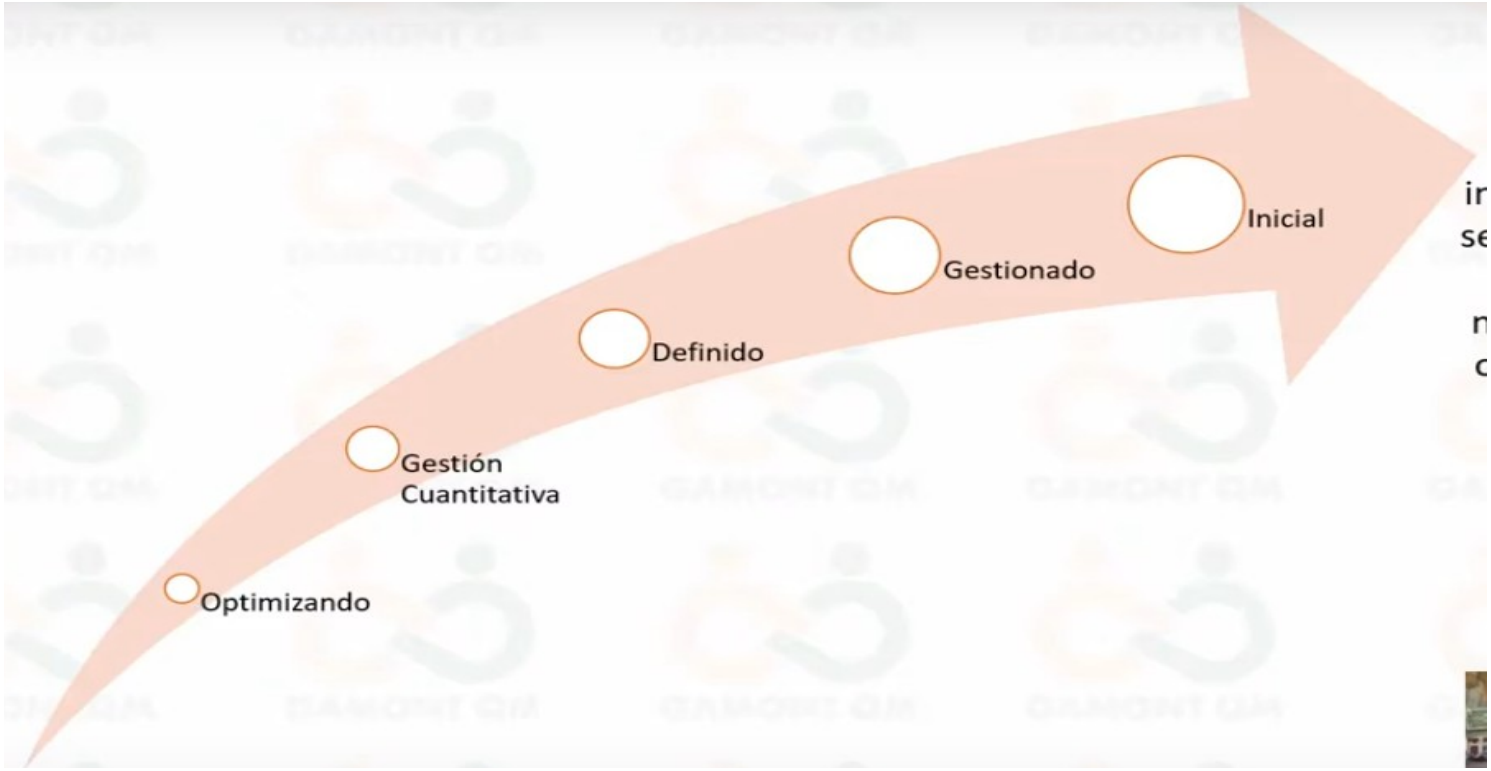
**02 ADMINISTRADO**
El proceso es reactivo y se caracteriza por su aplicación a proyectos.

**01 EJECUTADO**
El proceso se ejecuta y se producen productos basados en productos de entrada identificados.

**0 INCOMPLETO**
El proceso no se ejecuta o se hace parcialmente.

# NIVEL DE MADUREZ Y EL RIESGO

GRAVEDAD DEL IMPACTO DE EVENTOS NEGATIVOS




POSIBILIDAD DE OCURRENCIA DE EVENTOS NEGATIVOS

# EMPRESAS CERTIFICADAS CON CMMI MUNDIALMENTE



# EMPRESAS CERTIFICADAS CON CMMI MUNDIALMENTE



# EMPRESAS CERTIFICADAS CON CMMI MUNDIALMENTE



EN 2007 EL PROCESO DE EVALUACIÓN INCLUYÓ LA EVALUACIÓN DE LOS PROYECTOS DE IBM EN APPLICATION SERVICES MOSTRANDO EL CONTINUO LIDERAZGO Y ÉXITO DE IBM EN LA ENTREGA DE SERVICIOS SUPERIORES A SUS CLIENTES A NIVEL MUNDIAL

IBM® INDIA®



# MÉTODO DE DETERMINAR EL NIVEL DE MADURACIÓN

| Nivel de Madurez | Idea descriptiva | Grupo | Individuo | Concepto Clave |
|---|---|---|---|---|
| Inicial | El Sistema triunfa por genialidades | El Equipo de Fútbol de Llanero al Real Madrid | El Empleado del Mes | Los Super Héroes, los Centuriones |
| Gestionado | El éxito se basa en el cumplimiento de las órdenes (cadena de mando) | La armada, el ejército | El Soldado de Infantería | Los Generales del ejército |
| Definido | El cumplimiento de las reglas y procedimientos | La Burocracia Funcional, las dictaduras ilustradas | El Burócrata Ilustrado | Gestión por Funciones |
| Gestión Cuantitativa | La organización Científica, basada en el uso del Método) | El Reloj Suizo, jerárquico mecaniza | El Gestor de Proceso | Gestión por Procesos, no por funciones |
| Optimizando | El sistema sociocultural que se adapta y evoluciona en su contexto, como el cuerpo humano | Holocracia, equipos empoderados | El Gestor de Valor | Organismo complejo y biológico (ser vivo) |



# G R A C I A S

Línea de atención al ciudadano: 018000 910270
Línea de atención al empresario: 018000 910682

@SENAcomunica

www.sena.edu.co

**TALLER DE PRIORIDAD DE OPERADORES.**

Nombre:

Resuelva las siguientes operaciones teniendo en cuenta el orden de precedencia de los operadores matemáticos vistos en clase.

13 - 4 * (5 – 2) + 3 * (2 + 8)
16 + 3 * (6 – 4) – 3 * 5
23 – 8 + 6 * 2 – 3 * 4
6 * ( 7 * 5 – 4 * 6 ) + 81
23 - 5 * (6 – 2) + 9
4 * 3 + 7 − 2 * 4 + 3 * (9 – 5)
3 * (5 + 4) 6 + 3 * (8 – 4)
8 * (3 + 6) – 7 * (2 + 3)
6 + 3 * 5 – 4 * (7 – 2)
4 * (7 + 2) – 3 * 9
3 * 9 + 7 + 6 – 5 * 3
5 * (2 + 6) + 7 – 4 * 3
6 * 7 – 4 * 6 + 8 – 3
23 – 5 * (6 – 2) + 9
4 * 3 + 7 – 2 * 4 + 3 * (9 – 5)
40 – (15 – 6 – 8 ) - 10
5 * 7 + 5 * 2 – 5 * 4 + 5 * 10 – 5 * 3 + 5 * 12

EJERCICIO – ANALISIS DE CONSULTA SQL

Dada la siguiente consulta:

```sql
SELECT  aca.fecha_registro as Dia_que_se_asigno,
        aca.fecha_deseada as Fecha_de_la_cita,
        aca. tipo_id_paciente as Tipo_Documento,
        aca.paciente_id as Documento,
        aca.sw_atencion as Valor1_Canceleda,
        aca.cargo_cita as cups,
        ct.descripcion as descripcion,
        CONCAT(p.primer_nombre,' ', p.segundo_nombre,' ', p.primer_apellido,' ',p.segundo_apellido) NOMBRE_COMPLETO,
        p.residencia_telefono as telefono1,
        p.celular_telefono as telefono2,
        pl.plan_descripcion as entidad,
        st.usuario as asigno_cita,
        tc.descripcion as especialidad

FROM    agenda_citas_asignadas aca JOIN pacientes p ON aca.paciente_id = p.paciente_id
                                   JOIN planes pl ON aca.plan_id = pl.plan_id
                                   JOIN agenda_citas ac ON aca.agenda_cita_id = ac.agenda_cita_id
                                   JOIN agenda_turnos atr ON ac.agenda_turno_id = atr.agenda_turno_id
                                   JOIN tipos_consulta tc ON atr.tipo_consulta_id = tc.tipo_consulta_id
                                   JOIN system_usuarios st ON aca.usuario_id = st.usuario_id
                                   JOIN cups ct ON aca.cargo_cita = ct.cargo

WHERE   aca.fecha_deseada BETWEEN '2021-01-01' AND '2022-03-02';
```

De respuesta a las siguientes preguntas.

1. ¿Cuáles son las tablas que se pueden evidenciar en la consulta?

2. Identifique todas las columnas que se están llamando en la consulta y ubíquelas en la tabla que corresponde.

3. Identifique las llaves primarias y las foráneas.

4. Realice el modelo relacional.

5. Realice la base de datos en el gestor BD de su preferencia.

6. Una vez creada la base de datos, ingrese información.

7. Ejecute la consulta y verifique su funcionalidad.

| | |
|---|---|
| **ACTA DE REUNION**<br>PT-AR-01<br>FICHA 2339490<br>PROYECTO: | |

**MOTIVO DE LA REUNION: Avances de la semana**

**FECHA: 2 de mayo de 2022**

**HORA: 7:08 am**

**PROYECTO/MODULO:**

**LUGAR:** AMBIENTE 8

**PARTICIPANTES:**

| NOMBRE | CARGO |
|---|---|
| Alvaro Rodriguez Fernandez | Instructor |
| | |
| | |
| | |
| | |

**ORDEN DEL DIA:**

*El objetivo de la reunión es ver el avance del desarrollo de la semana, se abordarán los temas y compromiso del equipo en el desarrollo del proyecto.*

**DESARROLLO DE LA REUNION:**

***Documentación:***

- 

**Front-End:**

- 

**Back-End:**

- 

***COMPROMISOS:*** *Terminar informe general del sistema, Historias de usuarios, Diagramas de casos de usos.*

| Actividad | Responsable | Fecha Entrega | Estado |
|---|---|---|---|
| | | | |

| | |
|---|---|
| **ACTA DE REUNION**<br>PT-AR-01<br>FICHA 2339490<br>PROYECTO: |  |

| | | | |
|---|---|---|---|
| | | | |
| | | | |
| | | | |

Se firma la presente Acta por quienes en ella intervinieron.

| Nombres y Apellidos : | Firma : |
|---|---|
| Alvaro Rodriguez Fernandez | |
| | |
| | |
| | |
| | |

# Sesión # 11: Componente Práctico

Implementar una super clase abstracta llamada Producto, la cual tenga los siguientes atributos y métodos.

Atributos:

- Código.
- Descripción.
- Precio de compra.
- Precio de venta.
- Cantidad en bodega.
- Cantidad mínima requerida en bodega.
- Cantidad máxima de inventario permitida.
- Porcentaje de Descuento.

Métodos abstractos:

- Solicitar pedido: devuelva true si debe solicitar el producto al proveedor y false en caso contrario.
- Calcular total a pagar : devuelva la cantidad total a pagar al proveedor dado una cantidad de unidades de compra.
- Mostrar: devuelva una cadena descriptiva con la información del producto separada por guión medio (-).

Adicionalmente se desea dos subclases para los siguientes tipos de productos:

- Prendas de vestir (como lo son blusas, jeans, camisas, etc) el cual debe tener los siguientes parámetros adicionales:
    - Talla: S, M, L, etc
    - Permite planchado: verdadero o falso.

- Calzado (como lo son tenis, calzado formal, sandalias, etc) el cual debe tener el siguiente parámetro adicional:
    - Talla: 35, 36, 37, etc

Cada subclase debe implementar los métodos abstractos de la clase Producto.

Diseñar un programa que:

- Consulte el número de productos de tipo de prendas de vestir a manejar.
- Consulte el número de productos de tipo calzado a manejar.
- Cree un vector de productos de prendas de vestir en el cual se guardarán las instancias de cada uno de ellos.
- Cree un vector de productos de calzado en el cual se guardarán las instancias de cada uno de ellos.
- Solicite los datos requeridos para cada tipo de producto a guardar.

Implementar en la clase principal un menú con las siguientes opciones:

- Consultar producto: solicite el código de un producto a vender, el tipo (calzado o prenda) y muestre la información del mismo.
- Verificar productos a pedir: recorrer los vectores de productos y arrojar una alerta en caso tal se deba solicitar el pedido al proveedor.
- Calzado con mayor cantidad de unidades: indicar cuál es el código del producto que

tiene mayor cantidad de unidades en bodega, si hay varios que cumplen con la condición, indicar el primero de ellos.

- Prenda con mayor cantidad de unidades: indicar cuál es el código del producto que tiene mayor cantidad de unidades en bodega, si hay varios que cumplen con la condición, indicar el primero de ellos.
- Modificar cantidad mínima requerida en bodega: leer código de producto, el tipo (calzado o prenda) y el nuevo valor para la cantidad de unidades mínimas requeridas, validar en el método set de la variable correspondiente que la nueva cantidad no sea menor a 0, en caso de serlo establecer la cantidad mínima como 0.
- Vender producto: solicite el código de un producto a vender, el tipo (calzado o prenda), y las unidades correspondientes, calcule el valor de la factura con descuento y sin descuento. Validar que existan las unidades necesarias para la venta y modificar las unidades existentes después de la venta utilizando el método set de la variable correspondiente.

Para tener en cuenta: el porcentaje de descuento es el mismo para todos los productos, y es del 1%.