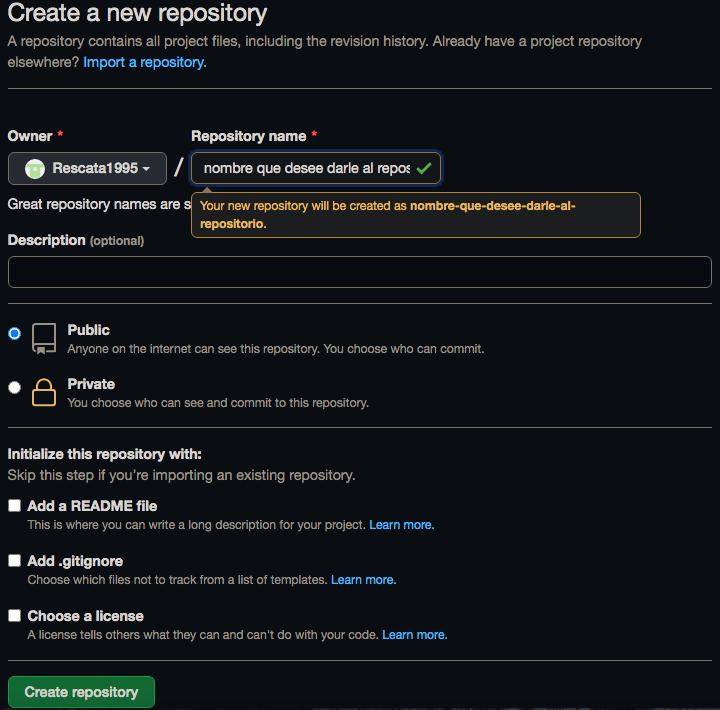
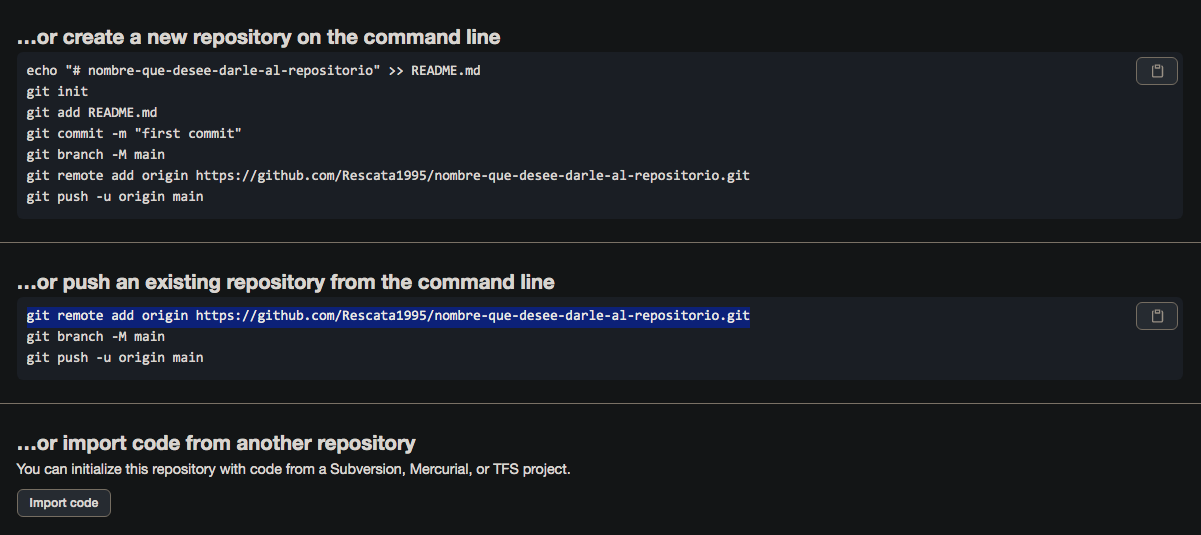
***Tutorial de GITHUB***

Todos los repositorios que hemos creado en GIT ahora nos interesa guardarlos en una nube o servidor externo (GITHUB ocupa GIT en la nube, eso hace). Si en caso dado se nos quema nuestro disco duro ya evitaremos completamente el riesgo de perder absolutamente todo o me encuentro trabajando en un proyecto grupal en el que necesito cooperar con más programadores; pues, todos nuestros repositorios estarán en la nube, el cual podemos retornar nuevamente a nuestra pc en cualquier momento. Esa interacción entre GIT & GITHUB es la que nos interesa. Pasos desde la creación de cuenta en GITHUB hasta la creación de un repositorio remoto vinculado a nuestro editor de texto y terminal:



Le damos un nombre al repositorio remoto que deseemos crear y decidimos si será de caracter público o privado y listo: create repository!

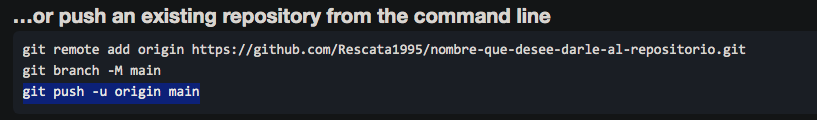


Luego, seleccionamos toda la ruta subrayada con fondo azul, la pegamos en nuestra terminal y presionamos “enter”, tal que así (el comando ***“git remote add origin”***... es el que tiene la función de vincular tu repositorio local con un repositorio remoto; de tal manera que, posteriormente, puedas exportar tus cambios guardados de un repositorio local en un repositorio remoto o, en su defecto, puedas importar los cambios guardados de un repositorio remoto en un repositorio local”):



Esto le dirá a GIT que ya estaremos trabajando de forma remota. Es decir, va a crear un identificador de todos nuestros archivos o de todo el trabajo que hemos hecho con GIT en la URL que hemos importado de GITHUB (nuestro repositorio local ahora estará en una nube).

Entonces, al terminar de registrar o guardar todos los commits correspondientes en nuestro repositorio local; es decir, cuando terminemos la tarea, importaremos la siguiente ruta (subrayada en fondo color azul) a nuestra terminal (y presionamos “enter”):



Esto cargará de forma automatica todos nuestros repositorios locales a GITHUB; de tal manera que, no sólo tengamos repositorios locales, sino que también en la nube (libre de riesgos de pérdida): sólo actualice su página en GITHUB.

Ahora bien, con esto ya tenemos vinculado al GIT local con el GIT remoto; es decir, no es necesario llamar de nuevo las anteriores rutas subrayadas si quisieramos guardar un nuevo commit en nuestro repositorio local y cargarlo en la nube de GITHUB. ¿Entonces qué se hace? pues se ejecuta el siguiente comando: “git push”. Listo, eso es todo. Así hemos cargado en la nube GITHUB nuestro último commit guardado en nuestro repositorio local. En el caso contrario, es decir, que quisieramos importar un “commit” del servidor externo de GITHUB (ya sea porque lo hayamos creado allá y no en nuestro repositorio local); pues, el comando a ejecutar sería: “git pull”. Por último, si deseamos compaginar por igual los “commits” que tenemos en ambos repositorios, local y remoto, debemos ejecutar el siguiente comando: “git fetch”. (la explicación de este último comando puede mejorar).

***Introducción a llaves SSH***

Aunque, en principio, github sea una plataforma o sitio web seguro (*HTTPS: Protocolo seguro de transferencia de hipertexto*) existe cierta vulnerabilidad a la que estamos expuestos: al introducir tu nombre de usuario y contraseña en la página web, éstas se guardan también en nuestro entorno local (digamos, en nuestro computador). Entonces, en el eventual caso de que sufras de un robo de tu dispositivo, estarás completamente expuesto a sufrir lo que se conoce como: ***“Password Cracking”***, estás expuesto a que “crackeen” (descifren) la contraseña de tu repositorio. “Password Cracking” es la ciencia que consiste en descifrar contraseñas o también se refiere al acto o proceso que se lleva a cabo para recuperar contraseñas que se han sido almacenadas en un equipo (cualquiera, desde dispositivos moviles a dispositivos de escritorio). En nuestro caso entonces estamos planteando la posible idea aterradora de que nuestro nombre de usuario y contraseñas de Github se vean expuestas a ser descifradas; seríamos vulnerables a ataques y quedaría en riesgo nuestra información sensible: servidor, proyectos (ya sean propios o de clientes), etc. Esta es la forma en la que sitios web, por ejemplo, son hackeados.

Para evitar todo esto, tenemos que agregarle una capa de seguridad mayor a nuestros sitios web de interés; donde, a veces, colocamos información confidencial o muy sensible y que deseamos proteger: esto se hace por medio de una combinación entre llaves públicas y privadas ***(llaves SSH)***, a este proceso (de cifrar por medio de llaves, especialmente una pública y una privada) más tarde se dará cuenta que se le conoce también como: ***cifradop o criptografia asimetrica.***

***El mecanismo funciona más o menos así:*** en nuestro entorno local (computadora) se debe crear una llave pública y una llave privada; y, una vez creadas, la llave pública debe ser enviada a github (en nuestro caso, nuestro sitio web de interés); pero puntualmente la llave pública es asignada a un repositorio remoto propio de github (al repositorio de nuestro interés *-se supone, el que se quiere cifrar*): todo este proceso se lleva a cabo conectandonos al repositorio por medio de un protocolo nuevo, ya no *HTTPS*; sino, ***SSH (Secure Shell)***. La llave privada sigue intacta en nuestra computadora, no es enviada nunca a ninguna parte.

dato: SSH es el mismo protocolo que se debe usar para conectarnos a servidores remotos, a otras computadoras que queramos controlar remotamente por consola.

Github, a partir de la llave pública que está correlacionada con tu llave privada y que le acabas de enviar; te regresa el favor y te envía, cifrado con tu propia llave pública (la que le mandaste), su propia llave pública de github (es un cifrado hecho, recuerde, a partir de la llave pública que te recibe github; y github, a partir de ahí, con ese cifrado, te genera una llave pública propia de él para ti). Entonces, github le manda su propia llave pública a su entorno local y la conecta: y así ambos quedan, mutuamente, conectados por medio del protocolo SSH, en lo que se conoce como una ***conexión de doble camino***, la cuál es 100% cifrada por SSH.

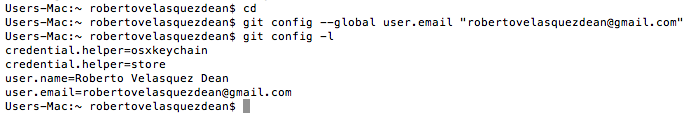
Por otro lado, la llave privada de su entorno local y que nunca es compartida, si desea, le agrega una contraseña encima (sobre ella): esto para tener una llave privada mucho más robusta y poderosa.

***Ahora sí, manos a la obra, veamos paso a paso todos estos procesos en un repositorio real de github.***

Aclaremos algo de entrada, *las llaves SSH no son por repositorio o por proyecto, sino por persona*; es decir, puedes crearlas en cualquier parte donde te situes parcialmente dentro de tu consola. Idealmente se crean en la carpeta home *(cd, cd ~)*.

Posteriormente, verificamos que el correo registrado en nuestra cuenta *github* coincida con el configurado (o potencialmente a configurar) en nuestro entorno local, puntualmente en *git.* El comando de git para registrarle un correo es el siguiente: ***git config --global user.email “inserte correo aquí -que debe coincidir con el registrado en su cuenta de github, recuerde”.***

Para verificar que efectivamente hemos registrado con exito el correo insertado, podemos corroborar esta información con el siguiente comando: ***“git config -l”***, donde podemos revisar también cuál es el nombre propiamente de usuario registrado en git. Hasta ahora el registro de comandos ejecutados dentro de su terminal se debe ver algo más o menos así:



Hecho esto, estando en sintonia nuestro correo electronico registrado en git y, posteriormente, nuestro correo electronico registrado en github; **Ya podemos crear nuestras llaves SSH y configurarlas en nuestro entorno local primeramente antes que conectarnos a github con SSH**

**(no olvide estar situado en *home*).**

**1.** Corra el siguiente comando en su terminal: ***ssh-keygen -t rsa -b 4096 -C “robertovelasquezdean@gmail.com”***

*Vamos por partes:*

***rsa*** es un algoritmo más, entre otros, para crear la llave ssh (siendo el más popular de hecho)

***-b + número*** es un elemento que nos indica, a partir de un número, qué tan compleja queremos que sea la llave.

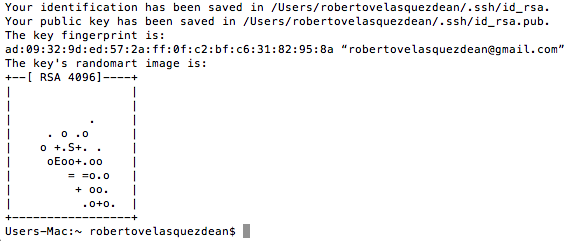
***-C “email...”*** se refiere a definir *qué correo electronico queremos que esté conectado a la llave en cuestión.* Debe intuir que es el mismo correo que hemos estado configurando, y en sintonia, en git y github.

Inmediatamente luego de correr la línea de comando, con el comando para crear llave SSH, se nos mostrará esto:



*Este proceso ya me está generando la llave pública y privada a partir del comando ejecutado.* Posteriormente nos pregunta dónde deberiamos guardar las llaves (y he aquí la importancia de situarnos inicialmente desde el *home*), si usted desea sólo presiona *enter,* ya la consola sobreentiende que usted desea guardar sus llaves en su dirección actual. Es decir, en nuestro caso, las llaves serán guardadas en *home* y, más puntualmente, en una carpeta oculta llamada: ***“.ssh”***

Luego de presionar *enter* la consola pide un ***“passphrase”,*** *passphrase* en realidad es una manera *fancy* de escribir *“passwords con espacios”.* Es básicamente la contraseña, adicional y de texto, que le vas a poner *(me imagino que por temas de respaldo)* a tu llave pública y privada (a la carpeta *.ssh* para poder acceder a ella); sin embargo, es preciso decir que, esto es opcional, pero lo más recomendable es que sí se le coloque *(ahora no lo haremos así, presionamos enter dos veces para darle entender a la consola que no queremos contraseñas de respaldo sobre la carpeta .ssh).* Verá algo más o menos así:

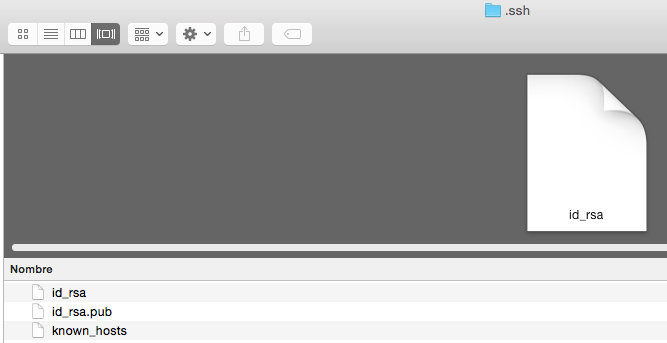


*Expliquemos por parte...*

Your identification... Quiere decir, tu identificación (llave privada) ha sido guardada en el archivo text: ***“id\_rsa”*** de la carpeta oculta: ***.ssh*** (se sobreentiende que en *home* como carpeta padre). La llave pública (your public key... ) también ha sido guardada en la carpeta oculta “.ssh”; sin embargo, ésta se guardó puntualmente en el archivo: ***“id\_rsa.pub”.***

Uno confirma que las llaves resultantes son de verdad por el ***key fingerprint*** (the key fingerprint is... )***: la huella.*** The ***key’s randomart image*** *es también otra forma de saber que tus llaves son de verdad y es una forma adicional, también, de compartir la llave pública.*

Ahora, vamos a situarnos dentro de la carpeta oculta *.ssh,* tal que así:



Como ya lo hemos dicho, dentro de esa carpeta se encuentra tanto nuestra llave pública como la llave privada. **id\_rsa** = archivo de texto donde está la llave privada; **id\_rsa\_pub** = archivo de texto donde está la llave pública.

Ahora, como nuestro interés es proteger a un sitio web, a nuestro repositorio en github más especificamente... Copiamos nuestra llave pública y la llevamos a github. **PERO STOP. PRIMERO VAMOS A CONFIGURAR LAS LLAVES SSH EN LOCAL.**

***Así es, primeramente, debemos agregar a la propia Mac las llaves (hacer que las identifique).*** Cómo se hace esto? En el mundo Mac es distinto que en Linus y en Windows, es de la siguiente forma:

*Primero revise antes que nada que el “ssh agent” esté corriendo*, **ssh Agent** es un programa auxiliar que realiza un seguimiento de las claves de identidad del usuario y sus frases de contraseña. El agente puede usar las claves para iniciar sesión en otros servidores sin que el usuario escriba una contraseña o frase de contraseña nuevamente. Esto implementa una forma de inicio de sesión único (SSO). Entonces, para revisar que todo esté bien aquí, ejecute el siguiente comando: **eval “$(ssh-agent -s)”***(presione enter)*

Luego, para Macs modernos (Mac OS Sierra o superior a 10.12) toca hacer un proceso particular con el ***archivo*** ***config*:** **hay que modificarlo; y, si no existe, toca crearlo.** Yo no lo tengo (verifiqué incluso mis archivos ocultos de la propia carpeta oculta ***.ssh,*** donde tengo que buscarlo,y no se encuentra), tengo entonces que crearlo con un editor de texto de mi preferencia, tal que así: ejecutando el comando ***“vim config”. Ojo, este archivo debe crearse dentro de la carpeta oculta “.ssh”; es decir, tiene que situarse antes en esta carpeta para crear el archivo “config”.***

Dentro del archivo “config” **debe escribir lo siguiente, respetando completamente la identación y las mayusculas y minusculas**, tal que así:

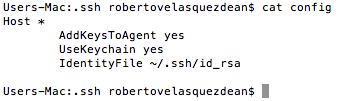
**Host \***

**AddKeysToAgent yes**

**UseKeychain yes**

**IdentityFile ~/.ssh/id\_rsa**

Luego, guarda. *(Si se fija, hemos hecho también un salto de línea al final y se ha respetado completamente).* En la terminal, se debe ver algo más o menos así:



Verificado lo anterior y ya teniendo nuestro archivo *config,* con su texto especial, dentro de la carpeta *.ssh.* **Regresamos a *home*** *(con comando cd o cd ~).*

Ahora para añadir la identidad de la llave privada a nuestro computador (para que su entorno local la identifique y pueda trabajar con ella), finalmente debe ejecutar este comando: **ssh-add -K ~/.ssh/id\_rsa.**

**Y listo, ya con esto hemos configurado nuestras llaves SSH en local, entorno local (nuestro PC). Debe verse algo así:**



Si se da cuenta, nos situamos primero en *home* antes de ejecutar el comando.

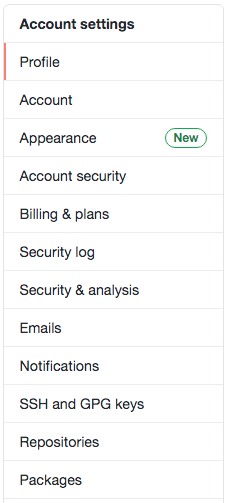
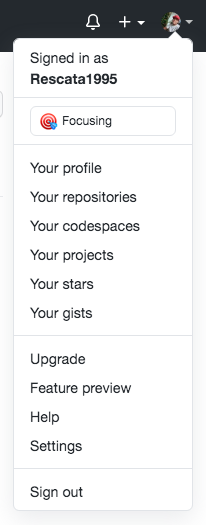
**Ahora sí, conexión a GitHub con SSH**

*Algo importante para aclarar antes de empezar:* si usted hace uso de varios computadores, digamos, tres. Los tres deben generar sus propias llaves SSH (su llave pública y su llave privada para cada tal) y hacer el mismo proceso que, anteriormente, ya fue explicado. Luego, con cada par de llaves SSH generadas en cada computadora, debe conectarse a su mismo repositorio en GitHub... lo que ahora explicaremos.

*pd. Es mejor tener llaves propias SSH para cada dispositivo; esto por un mismo tema de seguridad, no es bueno compartir llaves privadas de computador a otro (incluso aunque los computadores sean de un mismo dueño).*

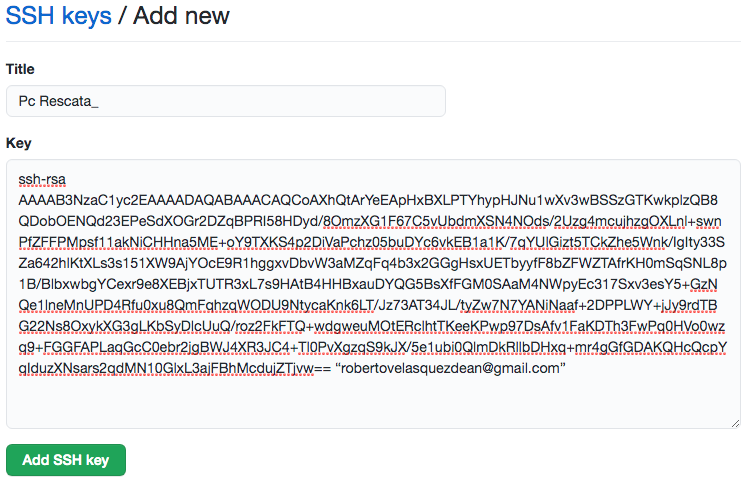
**Ahora sí! Vamos con GitHub.**

Tome su llave pública SSH, copiela y vayase a GitHub. Estando en *GitHub* se va a la sesión de ***Settings***y, posteriormente, se va al apartado que dice: ***“SSH and GPG keys”****, tal que así:*

***1) 2)***

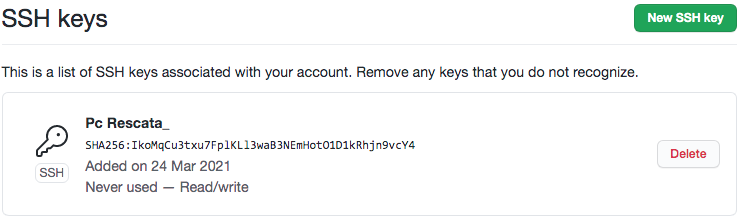
Mientras no tenga una SSH pública asociada, verá algo más o menos así: *“There are no SSH keys associated with your account”.* Presione entonces en: *.* ***Tome su llave pública*** *y...*

*Proceda, entonces, así (intuya que se debe colocar en cada espacio según el ejemplo a continuación):*



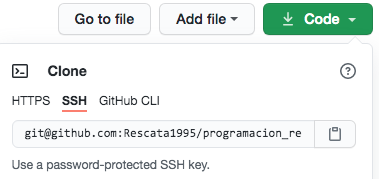
Luego de agregada la llave pública, luego de haber presionado en “Add SSH key”, proceda e introduzca su contraseña de *GitHub* (se la pedirán)*.*

Posteriormente, debe ver algo así:



La anterior imagen significa que ha registrado con exito su llave pública SSH en GitHub. *“Never used”* significa que nunca ha sido usada y *“Delete”* que puede ser borrada.

Luego, debe situarse en el repositorio de GitHub de su interés; y, si desea asegurarlo con una llave *SSH*, ya no más con *HTTPS,* debe hacer lo siguiente: presione *Code* y luego vayase a *SSH,* tal que así:



**Copiamos la URL contenida** y nos vamos de nuevo a nuestra terminal, puntualmente a donde tenemos situado, **ahora sí**, **nuestro repositorio local**: el que está vinculado con el repositorio remoto de GitHub de interés (que acaba de copiarle la URL de SSH)

Estando ahí, en su repositorio local de interés, debe ejecutar el siguiente comando: ***git remote -v***

*git remote -v* lo que nos muestra es la URL actual de nuestro repositorio local de interés; lo que queremos es, justamente, cambiarlo por la URL SSH del repositorio remoto, de interés, de GitHub. Veamos el proceso paso a paso:



Tenemos que nos salen dos repositorios con nombre de *origin,* este nombre nos interesa; ya que, necesitamos reemplazar las URL de todos los repositorios locales *(los Origin’s)* por la URL SSH copiada... ejecutando el siguiente comando: ***git remote set-url origin + pega* la URL SSH del repositorio remoto de nuestro interés de GitHub (el que habíamos copiado anteriormente).** Tal que así:



*Con este cambio, ya no tenemos una URL en protocolo HTTPS (ya no en nuestro repositorio remoto de interés); sino que, hemos pasado nuestra URL a protocolo SSH.* Revisemos:



*Si se da cuenta, ya no sale al inicio del repositorio “https://...”*; es decir, ya no estamos bajo ese protocolo efectivamente. Listo, lo hemos logrado.

Ahora, nos traemos esta última versión de nuestro repositorio remoto (ahora con protocolo SSH) a nuestro repositorio local, esto lo logra con el comando: *git pull.* Ejecute.

*un dato: ejecute comando en consola:* **defaults write com.apple.finder AppleShowAllFiles TRUE** *(presione Enter),*

*luego ejecute comando:* **KillAll Finder** *(presione Enter).* Todo este proceso para ver archivos ocultos. Sólo copie y pegue (y respete las mayusculas y minusculas).