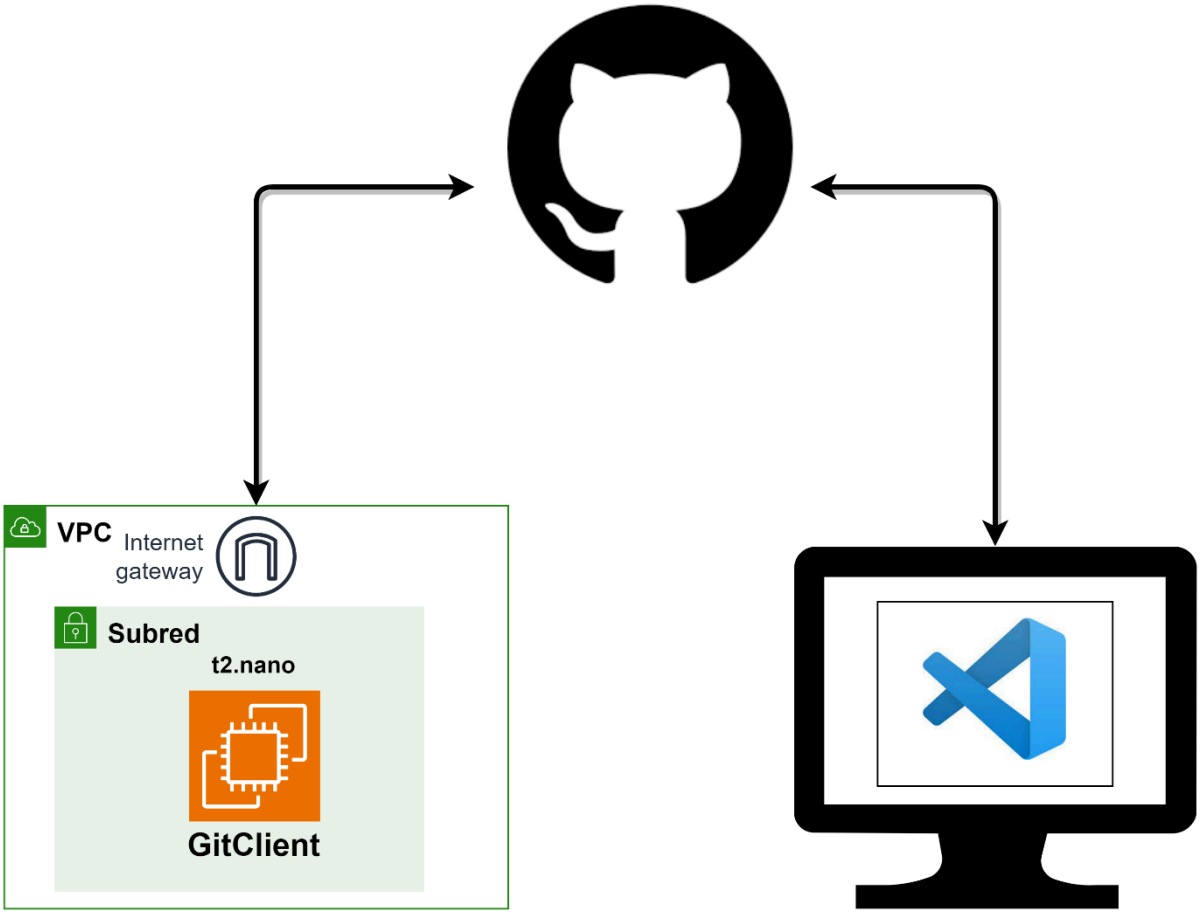
ACTIVIDAD 2.3 TRABAJANDO CON UN REPOSITORIO REMOTO EN GITHUB

1. INTRODUCCIÓN

En esta actividad crearemos un repositorio remoto en GitHub al que nos conectaremos con dos usuarios diferentes. Para simular que tenemos dos usuarios emplearemos nuestra máquina local y una máquina EC2. En la máquina remota usaremos git desde consola y realizaremos cambios en el repositorio. En nuestra máquina local usaremos la interfaz gráfica de git que viene incluida en VSCode. Crearemos ramas, las sincronizaremos con el repositorio remoto y resolveremos conflictos, y veremos cómo hacerlo tanto desde la consola como desde la interfaz de VSCode.



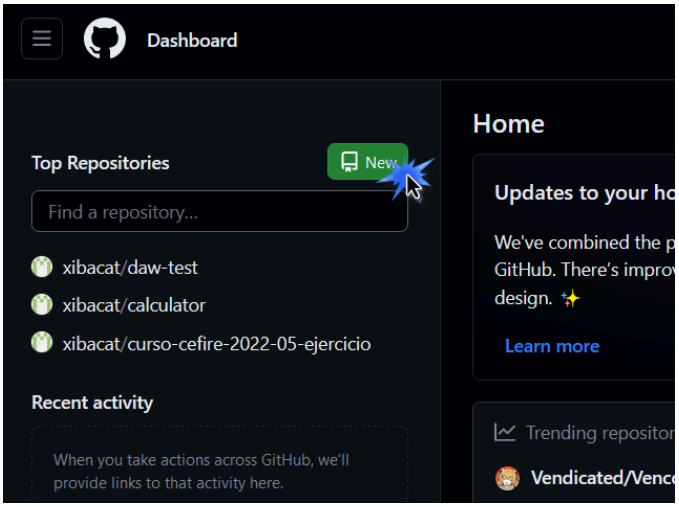
CONTENIDO

2. CREANDO EL ESCENARIO

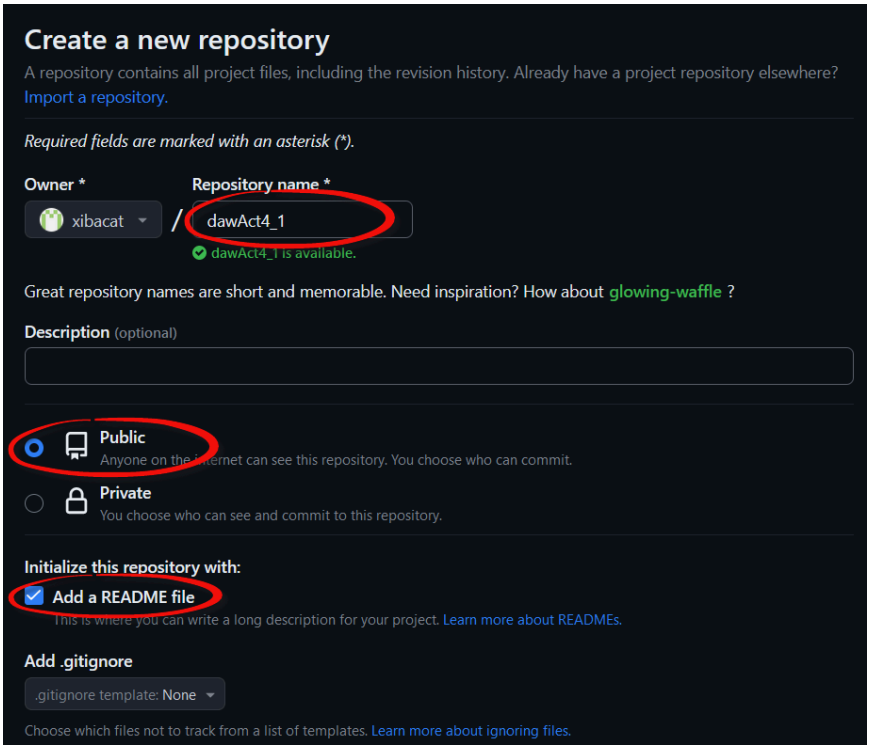
2.1 Configuración del Repositorio

En primer lugar, si no tienes cuenta en GitHub, debes crear una.

Crea un nuevo repositorio.



Dale un nombre y especifica que será un repositorio público. De esta manera cualquier persona podrá verlo en la web, pero sólo usuarios autorizados podrán modificarlo. Selecciona además que se añada un archivo README.md



Ya tenemos el repositorio, con un archivo README.md

2.2 Puesta a punto del cliente en una EC2

Crea una EC2 llamada “GitClient” tipo *t2.nano* usando un par de claves ***vockey***.

Conéctate a ella e **instala Git.**

Esta máquina se conectará a GitHub utilizando una conexión SSH. Para ello necesitamos transferir la clave privada a la máquina.

Primero crearemos un directorio *keys* en la máquina remota para guardar claves privadas

```
$ cd /home/ec2-user
$ mkdir keys
```

Después copiaremos el archivo *labsuser.pem* desde nuestra máquina local a la máquina GitClient usando el comando ***scp*** en la máquina local en la que estamos trabajando (nuestro ordenador, la sintaxis es la misma desde la consola de Windows o de Linux):

scp -i "credenciales_de_la_máquina_a_conectar " "archivo_a_transferir" <usuario>@<ip_destino>:<carpeta_destino>

Ejemplo:

```
> scp -i "labsuser.pem" "labsuser.pem"
ec2-user@54.174.102.0:/home/ec2-user/keys
labsuser.pem                                  100% 1678     0.4KB/s   00:03
```

El comando ***scp*** permite transferir archivos entre dos máquinas de forma segura. scp utiliza una sintaxis similar a la de ***ssh***: usa el parámetro ***-i*** para indicar la clave privada que usaremos para conectar con MyClient; continuación, se indica el archivo a transferir, que en este caso

es el mismo archivo de claves; y por último la máquina a conectar, indicando la ruta destino.

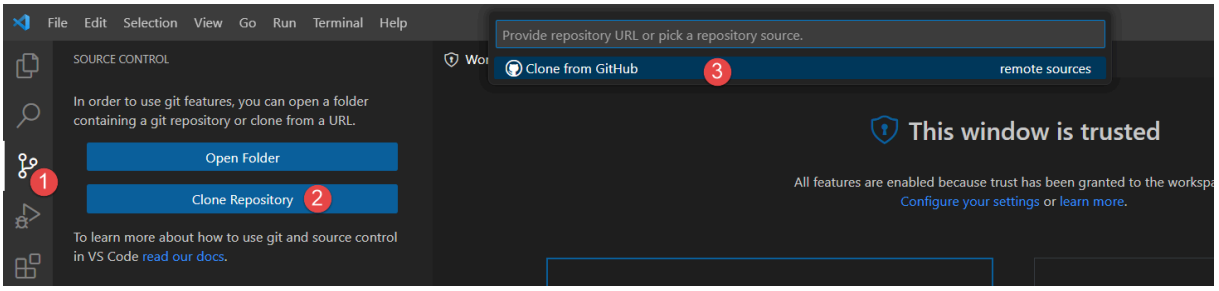
Para que funcione correctamente debes comprobar:

- Que te encuentras en la ruta de los ficheros, tanto el que queremos transferir como el que necesitamos para conectarnos (o que la ruta a los ficheros es correcta).
- Que la IP es la IP pública de la máquina GitClient.
- Que las credenciales para la conexión (*labsuser.pem*) sólo pueden ser accedidas por ti (al menos desde Linux es un requisito).

3. TRABAJANDO CON REMOTOS

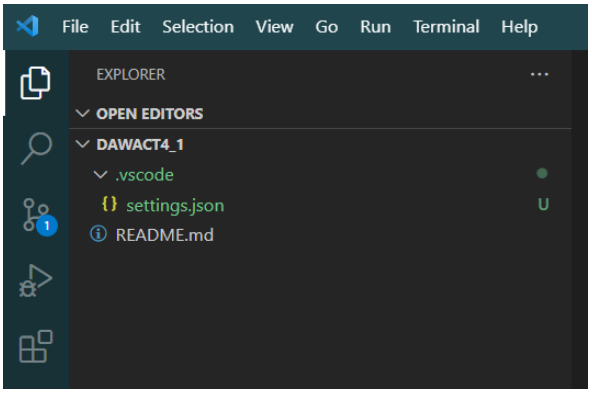
3.1 Clonación del repositorio en VSCode

Abre Visual Studio Code en tu máquina local y selecciona la pestaña lateral de control de versiones y pulsa Clone Repository.

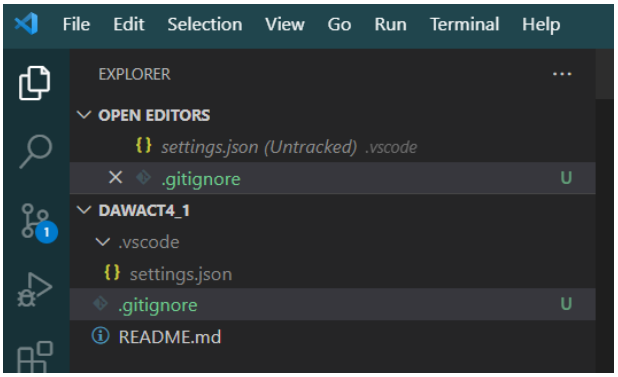


A continuación, seleccionaremos el repositorio recién creado. Es posible que VSC nos pida autenticarse en GitHub. Después seleccionamos la ruta en la que se creará el repositorio y abriremos el repositorio.

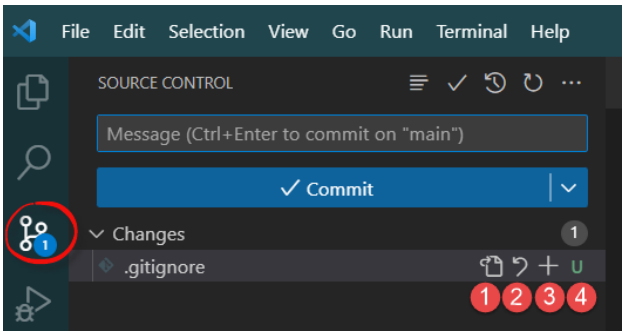
Veremos que tenemos el archivo README.md y quizás una carpeta *.vscode* que incluye las configuraciones del proyecto. Observa la U de ***Untracked*** porque ese archivo no estaba en el repositorio, lo ha creado VSC. (Si no tienes la carpeta .vscode créala tú).



Crea un archivo .gitignore para ignorar de la carpeta .vscode.



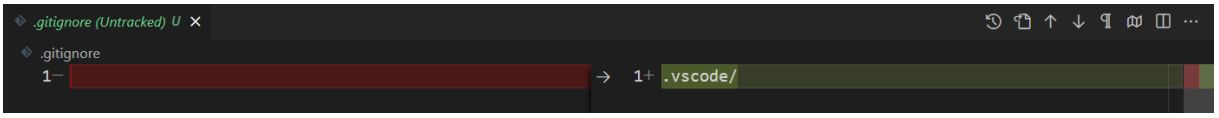
Observa que ahora la carpeta .vscode es ignorada mientras que el archivo .gitignore está marcado como ***Untracked***. Debemos añadirlo al repositorio. Para ello pulsamos en la pestaña de Git. Veremos todos los archivos modificados o sin seguimiento



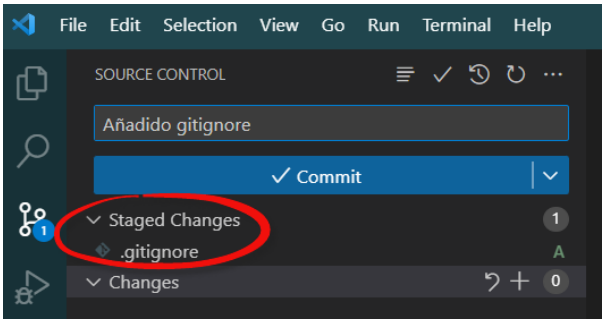
Fíjate en los iconos al lado del archivo, marcados con números rojos:

1. Abrir el archivo
2. Revierte los cambios (*git restore*)
3. Añade el fichero al stage (*git add*)
4. Indica el estado, en este caso **Untracked**

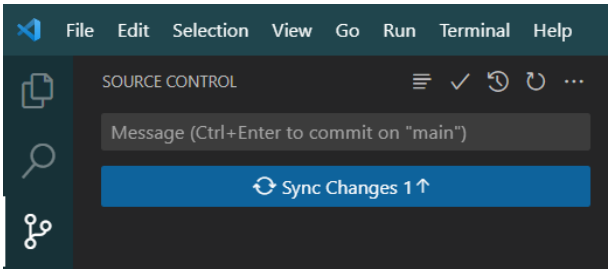
Si pulsamos el nombre del fichero veremos una vista tipo diff (adiciones y eliminaciones entre la última versión y el archivo modificado).



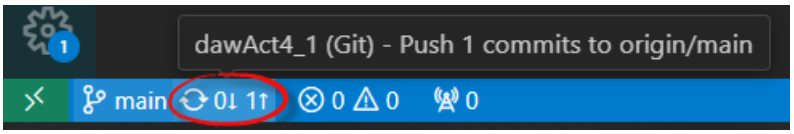
Añade el fichero al Stage. Observa que el archivo aparece en un nuevo espacio, el Stage, con el archivo preparado para hacer commit, y ha desaparecido de la sección Changes, de la misma forma que ocurría cuando usábamos *git status*.



Una vez hecho el commit, nos advierte de que tenemos un cambio sin sincronizar, es decir, tenemos un push pendiente. También se marca en la barra inferior a la izquierda



También se marca en la barra inferior a la izquierda, junto con la indicación de la rama en la que nos encontramos



Crea un archivo *index.html* que incluya un módulo de DAW. Por ejemplo:

Añádelo al repositorio y haz commit.

Sincroniza los cambios y comprueba que se reflejan en GitHub.

3.2 Autorización de GitClient en GitHub usando SSH

Al ser un repositorio público, cualquier usuario puede clonarlo, pero no va a poder modificarlo si no está autorizado. Vamos a autorizar a la máquina GitClient a acceder a nuestro repositorio usando SSH. Para ello necesitamos llevar a cabo dos pasos:

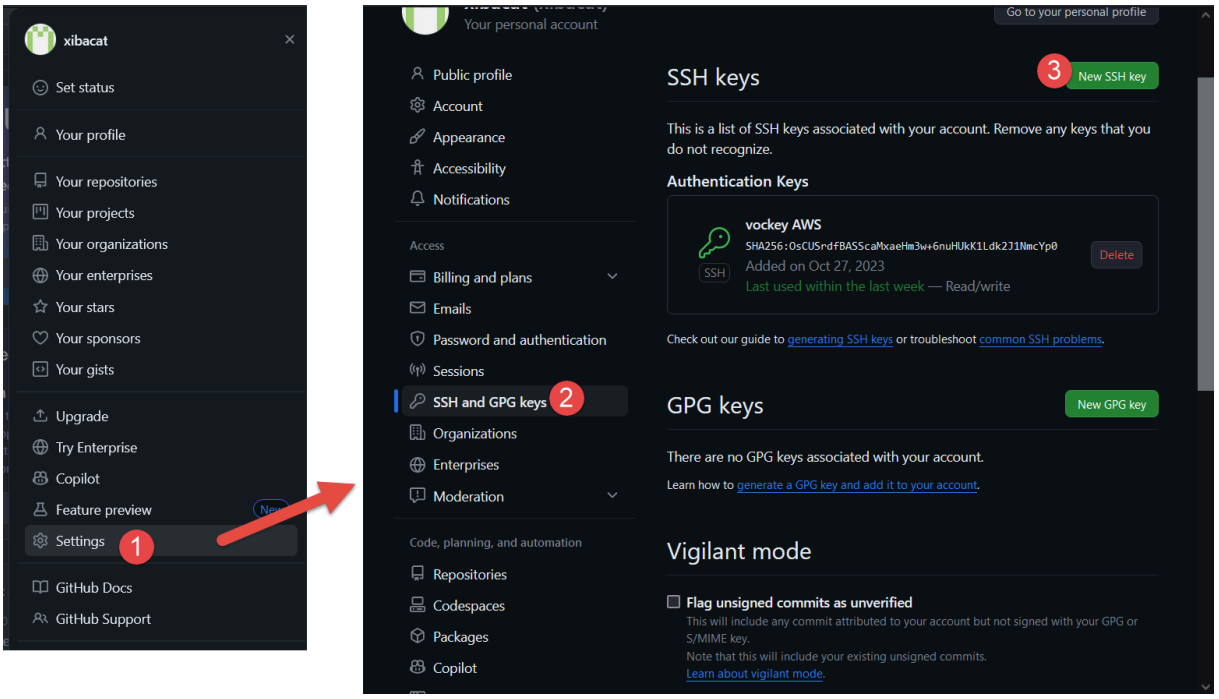
1. Dar de alta la clave pública en GitHub

2. Usar la clave privada de GitClient para identificarse en GitHub

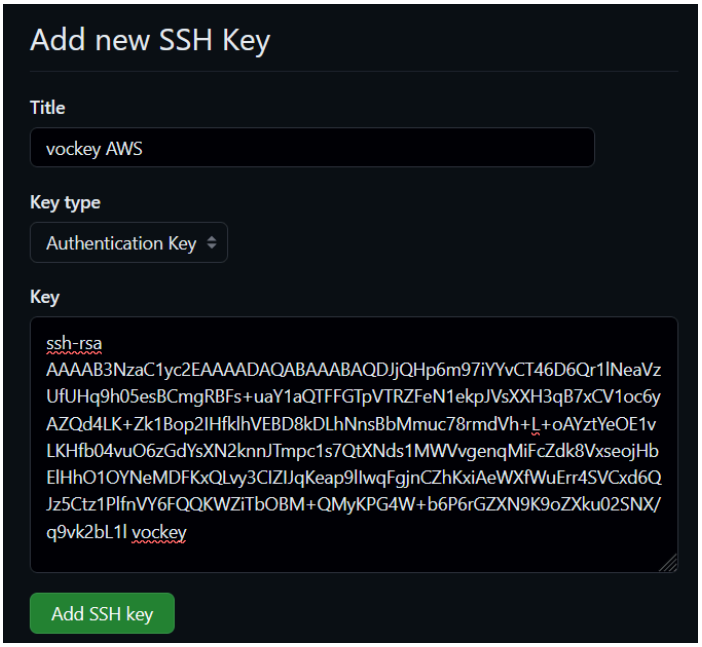
Dar de alta la clave pública en GitHub

En la máquina GitClient, muestra el contenido del archivo `/home/ec2-user/.ssh/authorized_keys` y copia el contenido en el portapapeles.

En GitHub, abre tu perfil (arriba a la derecha) y pulsa en *Settings*. En el menú de la izquierda selecciona *SSH and GPG keys* y pulsa *New SSH key*



En la siguiente ventana le daremos un nombre a la clave (*vockey AWS*) y copiaremos el contenido del archivo *authorized_keys*.



Uso de la clave privada de GitClient para identificarse en GitHub

El primer paso es agregar nuestra clave privada a *ssh-agent*, que es una utilidad auxiliar que administra nuestras claves y nos permite no tener que identificarnos cada vez en el repositorio, creando un inicio de sesión único.

Primero iniciamos el agente en segundo plano:

```
$ eval "$(ssh-agent -s)"
> Agent pid 59566
```

Ahora agregamos nuestra llave privada SSH a *ssh-agent*, pero primero debe ser de solo lectura:

```
$ chmod 400 /home/ec2-user/keys/labsuser.pem
$ ssh-add /home/ec2-user/keys/labsuser.pem
> Identity added: /home/ec2-user/keys/labsuser.pem
(/home/ec2-user/keys/labsuser.pem)
```

Ahora testaremos que podemos conectarnos mediante SSH a GitHub.

```
$ ssh -T git@github.com
```

Puedes ver la habitual advertencia de host desconocido:

```
[ec2-user@ip-172-31-25-207 ~]$ ssh -T git@github.com
The authenticity of host 'github.com (140.82.114.4)' can't be established.
ECDSA key fingerprint is SHA256:p2QAMXNIC1TJYWeIOttrVc98/R1BUFWu3/LiyKgUfQM.
ECDSA key fingerprint is MD5:7b:99:81:1e:4c:91:a5:0d:5a:2e:2e:80:13:3f:24:ca.
Are you sure you want to continue connecting (yes/no)? yes
```

Si todo se ha realizado correctamente obtendrás un saludo como este, usando tu nombre de usuario.

```
[ec2-user@ip-172-31-25-207 ~]$ ssh -T git@github.com
Hi xibacat! You've successfully authenticated, but GitHub does not provide shell access.
```

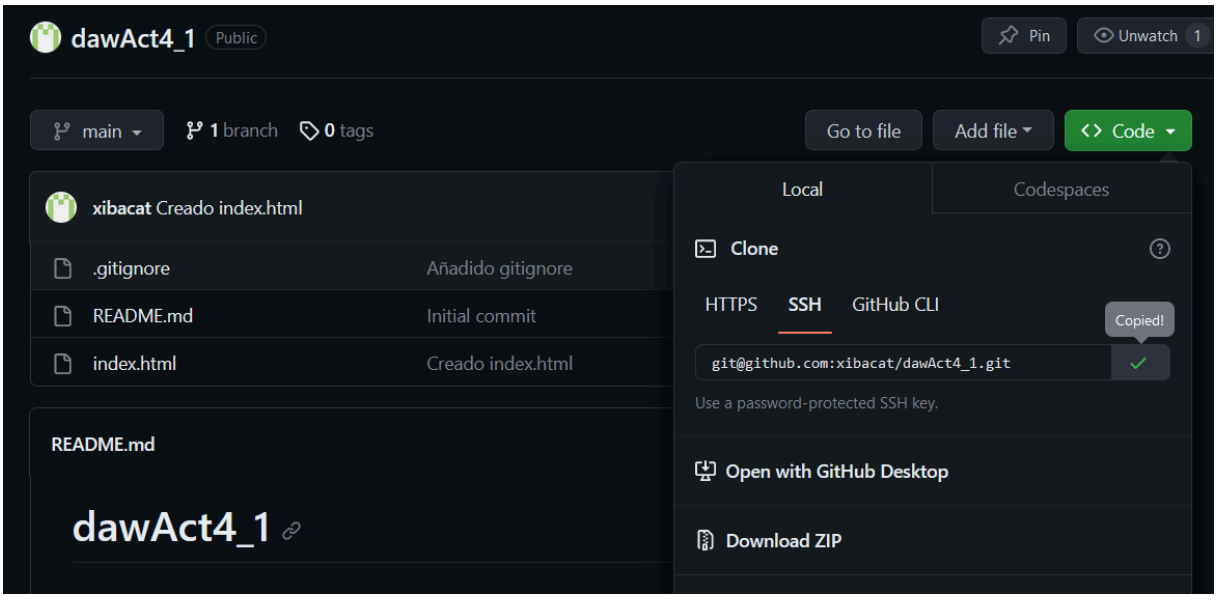
Este proceso de identificación lo deberás hacer cada vez que inicies sesión en la máquina.

Si quieres crear una clave diferente de la de AWS, puedes seguir los pasos indicados aquí:

https://docs.github.com/es/authentication/connecting-to-github-with-ssh/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent

3.3 Clonando el repositorio en GitClient

Ahora clonaremos el repositorio. En la página principal del repositorio en GitHub pulsa el botón Code, selecciona la pestaña SSH y copia la dirección SSH del repositorio:



En la máquina GitClient ejecuta *git clone* usando esa dirección:

```
$ git clone git@github.com:xibacat/dawAct4_1.git
```

Comprueba que el repositorio se ha clonado mirando los ficheros con un *ls -a*.

```
[ec2-user@ip-172-31-25-207 dawAct4_1]$ ls -a
.  ..  .git  .gitignore  index.html  README.md
```

Ejecuta ***git remote -v*** para verificar el remoto asociado a este repositorio (se nombra predeterminadamente como *origin* al hacer el clone):

```
[ec2-user@ip-172-31-25-207 dawAct4_1]$ git remote -v
origin  git@github.com:xibacat/dawAct4_1.git (fetch)
origin  git@github.com:xibacat/dawAct4_1.git (push)
```

Comprueba que puedes escribir en el repositorio haciendo un push:

```
[ec2-user@ip-172-31-25-207 dawAct4_1]$ git push
Warning: Permanently added the ECDSA host key for IP address '140.82.112.4' to the list of known hosts.
Everything up-to-date
```

Como no tienes cambios pendientes, no realiza ningún cambio.

3.4 Trabajando en local y actualizando el remoto desde GitClient

Ahora, estamos listos para comenzar a trabajar con nuestro repositorio local ***dawAct4_1***:

1. Edita el archivo index.html y escribe el nombre de un nuevo módulo del ciclo de Desarrollo de Aplicaciones Web. Guarda
2. Haz el primer commit. Deberías obtener algo como eso:

```
[ec2-user@ip-172-31-25-207 dawAct4_1]$ nano index.html
[ec2-user@ip-172-31-25-207 dawAct4_1]$ git commit -a -m "nuevo módulo añadido"
[main a83f21d] nuevo módulo añadido
 Committer: EC2 Default User <ec2-user@ip-172-31-25-207.ec2.internal>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

    git config --global --edit

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

 1 file changed, 1 insertion(+)
```

Hemos olvidado establecer el nombre y el correo electrónico en la instancia de GitClient, lo haremos ahora y en este caso los configuramos solo para el repositorio actual (*--local*):

```
$ git config --local user.name GitClient
$ git config --local user.email gitclient@daw.org
```

Tal y como indica el mensaje, corregimos el autor del commit con el comando git commit *--amend --reset-author*: se abrirá el editor, solo hay que guardar y salir (si se abre Vim, sal con *Ctrl+C* y escribe *:qa*)

```
[ec2-user@ip-172-31-25-207 dawAct4_1]$ git config --local user.name "GitClient"
[ec2-user@ip-172-31-25-207 dawAct4_1]$ git config --local user.email gitclient@daw.org
[ec2-user@ip-172-31-25-207 dawAct4_1]$ git config --global core.editor nano
[ec2-user@ip-172-31-25-207 dawAct4_1]$ git commit --amend --reset-author
[main ee3d732] nuevo módulo añadido
 Committer: GitClient <ec2-user@ip-172-31-25-207.ec2.internal>
```

3. Ejecuta git log para ver el historial del repositorio. Observa que el repositorio remoto origin (en rojo) está un commit por detrás del repositorio local (verde) porque todavía no hemos sincronizado los cambios.

```
[ec2-user@ip-172-31-25-207 dawAct4_1]$ git log
commit ee3d732a82ad771ece554fd532169581cea71cdf (HEAD -> main)
Author: GitClient <ec2-user@ip-172-31-25-207.ec2.internal>
Date:   Sat Oct 28 15:49:25 2023 +0000

    nuevo módulo añadido

commit ee8ce51b88dfca30c6fa3a19874f2e5b4fdc828f (origin/main, origin/HEAD)
Author: xibacat <xibanez.sai@gmail.com>
Date:   Fri Oct 27 12:34:25 2023 +0200

    Creado index.html
```

4. Actualiza el repositorio remoto (push). Ahora ejecuta de nuevo ***git log***, mostrará que los repositorios remotos (en rojo) y local (verde) están sincronizados, están apuntando a la misma confirmación

```
[ec2-user@ip-172-31-25-207 dawAct4_1]$ git log
commit ee3d732a82ad771ece554fd532169581cea71cdf (HEAD -> main, origin/main, origin/HEAD)
Author: GitClient <ec2-user@ip-172-31-25-207.ec2.internal>
Date:   Sat Oct 28 15:49:25 2023 +0000

    nuevo módulo añadido
```

5. Haz una rama *version1*. Ve a la rama *version1* y agrega a *index.html* el nombre de un módulo más. Haz commit de los cambios.
6. Muestra el estado del repositorio:

```
2126244 (HEAD -> version1) Primer commit en version1
ee3d732 (origin/main, origin/HEAD, main) nuevo módulo añadido
```

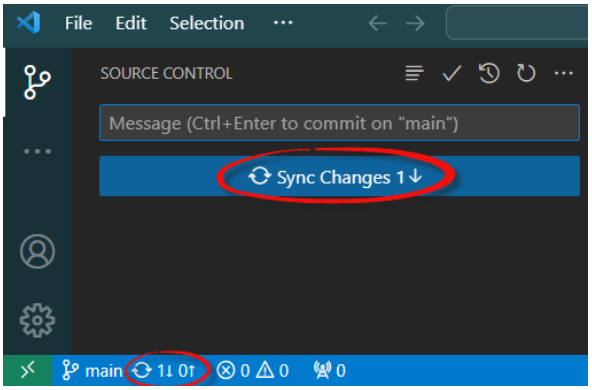
7. Observa que en local tenemos la rama *version1* y la rama *main*. En *origin* solo tenemos la rama main, porque como no hemos hecho push todavía el repositorio remoto no conoce de la existencia de la segunda rama
8. Actualiza el repositorio remoto (push) con la nueva rama, *versión1*. Para agregar la nueva rama al control remoto, el comando sería ***git push origin version1***.
9. Muestra de nuevo el estado del repositorio:

```
2126244 (HEAD -> version1, origin/version1) Primer commit en version1
ee3d732 (origin/main, origin/HEAD, main) nuevo módulo añadido
```

Como podemos ver, ya tenemos ambas ramas sincronizadas en repositorios remotos y locales.

3.5 Cambiando de rama en VSCode

Cuando abramos VSCode veremos que ya ha detectado que hay cambios en el repositorio sin sincronizar y que necesitamos hacer pull. Esto es porque por defecto tiene activada la opción Autofetch.



Sincroniza los cambios con el repositorio y cambia a la rama version1 pulsando sobre *main* y eligiendo la rama *origin/version1* en el menú que aparece. En próximas ocasiones ya tendrás una rama *version1* local, pero como esta es la primera vez debes hacer checkout a la rama de origin.

Introduce un nuevo módulo en *index.html*. A estas alturas deberías tener 4 módulos.

Guarda los cambios, haz *commit* y haz *push* (sincroniza).

3.6 Resolviendo conflictos en consola

Seguiremos trabajando en GitClient sin sincronizar con el remoto, de manera que se generará un conflicto que tendremos que resolver.

Asegúrate de que HEAD está en la rama version1:

```
[ec2-user@ip-172-31-25-207 dawAct4_1]$ git log --oneline
2126244 (HEAD -> version1, origin/version1) Primer commit en version1
ee3d732 (origin/main, origin/HEAD, main) nuevo módulo añadido
```

Vamos a añadir un módulo nuevo (y diferente del que añadimos en VSCode).

Haz commit y push.

```
[ec2-user@ip-172-31-25-207 dawAct4_1]$ git commit -a -m "Otro módulo más en version1"
[version1 147b35b] Otro módulo más en version1
 1 file changed, 1 insertion(+)
[ec2-user@ip-172-31-25-207 dawAct4_1]$ git push origin version1
To github.com:xibacat/dawAct4_1.git
 ! [rejected]        version1 -> version1 (fetch first)
error: failed to push some refs to 'github.com:xibacat/dawAct4_1.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

Obtendrás un error al no estar sincronizados los commits locales con los del repositorio. Es decir, el repositorio remoto tiene commits en esa rama que no tenemos en nuestro repositorio local.

Podríamos haber sabido esto previamente ejecutando ***git remote show origin***

```
[ec2-user@ip-172-31-25-207 dawAct4_1]$ git remote show origin
* remote origin
  Fetch URL: git@github.com:xibacat/dawAct4_1.git
  Push  URL: git@github.com:xibacat/dawAct4_1.git
  HEAD branch: main
  Remote branches:
    main     tracked
    version1 tracked
  Local branch configured for 'git pull':
    main merges with remote main
  Local refs configured for 'git push':
    main     pushes to main     (up to date)
    version1 pushes to version1 (local out of date)
```

Como sugiere el mensaje de error del *push*, deberíamos hacer *pull* antes del *push*. Vamos a usar la opción ***--rebase=false***, que especifica que, si puede, después del *fetch* hará un *merge*.

```
[ec2-user@ip-172-31-25-207 dawAct4_1]$ git pull origin version1 --rebase=false
Warning: Permanently added the ECDSA host key for IP address '140.82.114.3' to the list of known hosts.
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (1/1), done.
remote: Total 3 (delta 2), reused 3 (delta 2), pack-reused 0
Unpacking objects: 100% (3/3), 299 bytes | 299.00 KiB/s, done.
From github.com:xibacat/dawAct4_1
 * branch            version1   -> FETCH_HEAD
   2126244..6043f51  version1   -> origin/version1
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.
```

Pero el *pull* no se puede hacer porque tenemos un conflicto, provocado porque el commit de VSCode y el que hemos hecho desde GitClient modifican la misma línea de *index.html*.

Debemos solucionarlo. En este caso nos quedaremos las dos líneas. Abrimos el fichero con nano y observa cómo indica la línea que tenemos en local (HEAD) y la del repositorio con el

hash de commit. Eliminamos las anotaciones de git (<<<, ===, >>>, etc), conservamos las dos líneas y guardamos.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>

<body>
  <h1>Módulos de Desarrollo de Aplicaciones Web</h1>
  <ul>
    <li>Despliegue de Aplicaciones Web</li>
    <li>Programación</li>
    <li>Bases de datos</li>
<<<<<<< HEAD
    <li>Diseño de interfaces web</li>
=======
    <li>Lenguaje de marcas</li>
>>>>>>> 6043f51a4270e4665b81a6120d44ec8a0328cded
  </ul>
</body>

^G Get Help     ^O Write Out    ^W Where Is     ^K Cut Text     ^J Just
^X Exit         ^R Read File    ^\ Replace      ^U Uncut Text   ^T To
```

Hacemos commit de los cambios que resuelven el conflicto y comprobamos el log:

```
[ec2-user@ip-172-31-25-207 dawAct4_1]$ git log  --graph
*   commit 1fea5708c79e1dec8857d18d4326263f1e25f724 (HEAD -> version1)
|\  Merge: 147b35b 6043f51
| | Author: GitClient <gitclient@daw.org>
| | Date:   Sat Oct 28 19:00:44 2023 +0000
| |
| |     Conflicto entre usuario VSCode y GitClient resuelto
| |
| * commit 6043f51a4270e4665b81a6120d44ec8a0328cded (origin/version1)
| | Author: xibacat <xibanez.sai@gmail.com>
| | Date:   Sat Oct 28 20:13:02 2023 +0200
| |
| |     4 módulos en index
| |
* | commit 147b35b06d5d256697fddef826b54745bf9aa837
|/  Author: GitClient <gitclient@daw.org>
|   Date:   Sat Oct 28 18:42:40 2023 +0000
|
|       Otro módulo más en version1
|
* commit 2126244d941012ab06fb38a3defaee6d2642069b
| Author: GitClient <gitclient@daw.org>
| Date:   Sat Oct 28 16:21:17 2023 +0000
|
|     Primer commit en version1
|
```

Ahora ya podemos hacer *push*:

```
[ec2-user@ip-172-31-25-207 dawAct4_1]$ git push origin version1
Warning: Permanently added the ECDSA host key for IP address '140.82.113.3' to the list of known hosts.
Enumerating objects: 10, done.
Counting objects: 100% (10/10), done.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 629 bytes | 629.00 KiB/s, done.
Total 6 (delta 4), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (4/4), completed with 2 local objects.
To github.com:xibacat/dawAct4_1.git
   6043f51..1fea570  version1 -> version1
[ec2-user@ip-172-31-25-207 dawAct4_1]$ git log --oneline
1fea570 (HEAD -> version1, origin/version1) Conflicto entre usuario VSCode y GitClient resuelto
```

3.7 Fusionando ramas y resolviendo conflictos desde VSCode

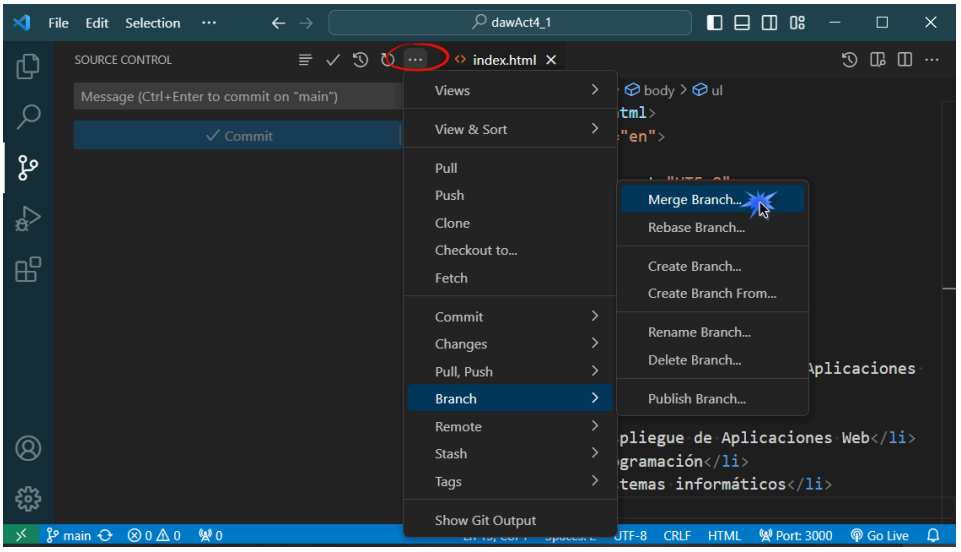
Ahora vamos a añadir una modificación en la rama *main* desde VSCode y después trataremos de fusionar la rama *version1* con *main*. Se generará un conflicto que tendremos que solucionar usando la interfaz de VSCode.

En primer lugar sincronizaremos VSCode con el repositorio remoto y después cambiaremos a la rama *main*.

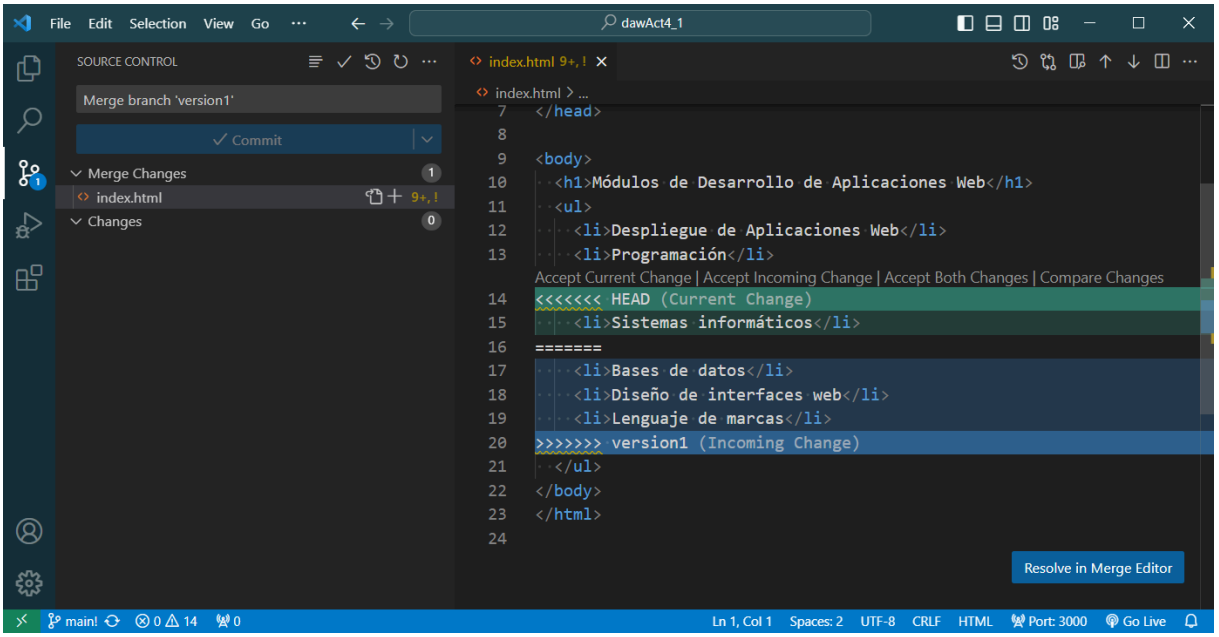
En esta rama debemos tener dos módulos en index.html y vamos a introducir un módulo más diferente de los anteriores.

Añadimos, guardamos y hacemos commit.

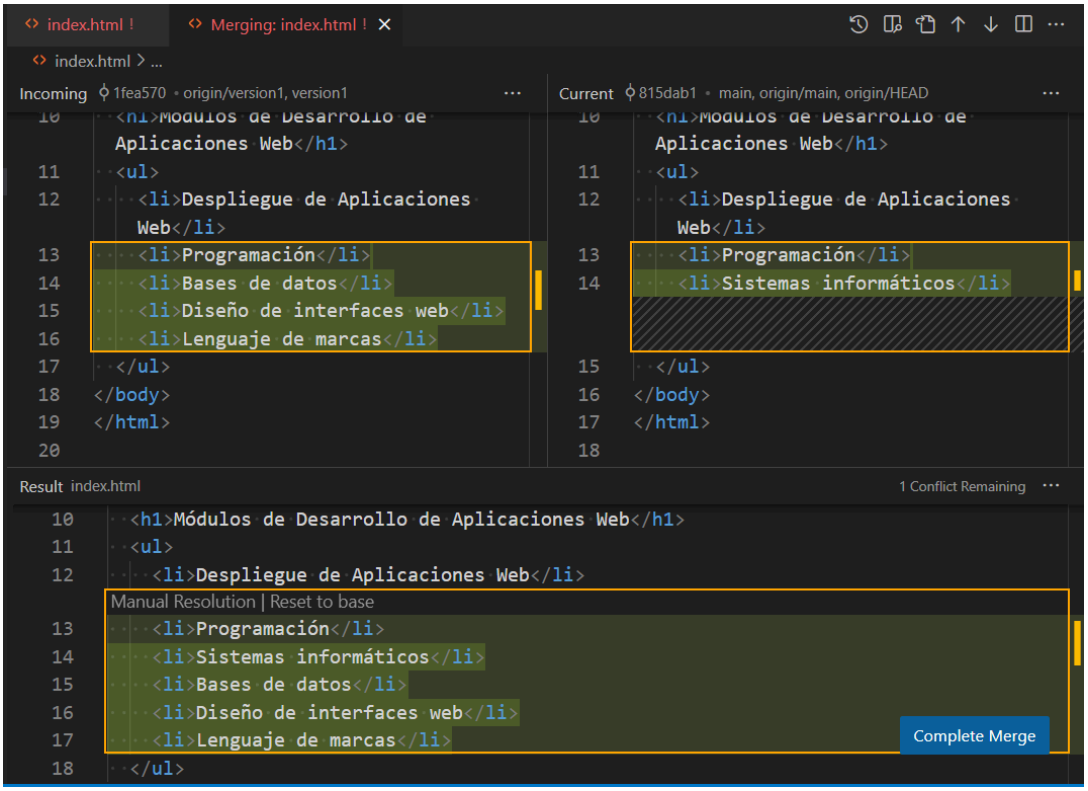
Pinchamos en los tres puntos del panel de Source Control → Branches → Merge Branch



En el menú que aparece seleccionamos version1. Y nos aparecerá una advertencia de que hay conflictos. Si cerramos la advertencia veremos esto:



En estos casos deberemos leer bien el código y entender los cambios para resolver el conflicto. En nuestro caso queremos quedarnos todos los cambios, nos interesan todos los módulos, con lo que pincharemos en *Accept Both Change*. En otros casos más complejos puede ser de utilidad el Merge Editor, que nos muestra el código de cada rama y el resultado.



Guarda el fichero, mételo en el Stage y haz commit. Después sincroniza.