EXPLICACIÓN PRÁCTICA - GIT/GITHUB

¿PARA QUÉ SIRVE GIT?

“El control de versiones es un sistema que registra los cambios realizados sobre un archivo o conjunto de archivos a lo largo del tiempo, de modo que puedas recuperar versiones específicas más adelante.”

https://git-scm.com/book/es/v1/Empezando-Acerca-del-control-de-versiones

SISTEMA DE CONTROL DE VERSIONES DISTRIBUIDO

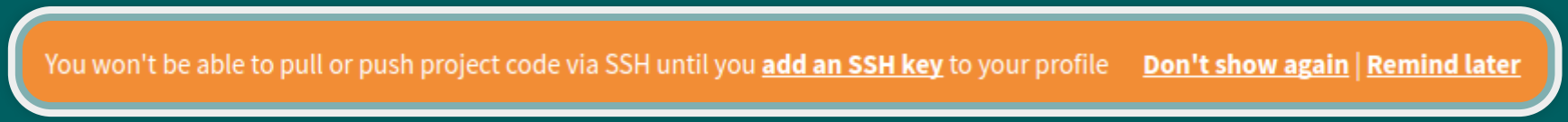
GITHUB

Github es una aplicación opensource que nos permite administrar repositorios en git mediante una interfaz web.

- Perfil
- Repositorios propios (públicos o privados).
- **Generación y configuración de clave SSH**
- Red social de desarrolladores/as.
- Una pagina web por proyecto.
- Toda la ayuda de Github.

USANDO CLAVE SSH

- Agregar la clave SSH es para toda la plataforma de Git.
- Sirve para que el servidor te reconozca y no pida usuario y contraseña cada en cada push.
- **Nunca cargar en tu cuenta personal la clave pública de una PC que es compartida** ya que cualquiera podría modificar en tu nombre.



AGREGAR CLAVE SSH

1. Crearla (si no la han creado ya):

```
$ ssh-keygen -t rsa
```

   Con esto se han creado las claves pública y privada. Las clave privada nunca debe ser compartida. La clave privada te identifica frente a otros dispositivos.

2. Copiar la clave pública al servidor Gitlab. Ésta se ubica en el archivo ~/.ssh/id_rsa.pub

```
$ cat ~/.ssh/id_rsa.pub
```

OPCIONES AL CREAR UN NUEVO REPOSITORIO

OPCIONES AL CREAR UN NUEVO REPOSITORIO

PRACTICA!

CREAREMOS UN REPO Y TRABAJAREMOS EN SIMULTANEO DESDE 2 PC

INICIALIZACIÓN EN “COMPUTER A”: OPCION 1

```
A@pcA$ mkdir python1
A@pcA$ cd python1
A@pcA$ git init
A@pcA$ git remote add origin https://github.com/minuevorepo.git
```

Creamos una carpeta, inicializamos y le decimos a git cuál es su servidor-proyecto

INICIALIZACIÓN EN “COMPUTER A”: OPCION 2

```
A@pcA$ mkdir python1
A@pcA$ cd python1
A@pcA$ git init
A@pcA$ touch README.md
A@pcA$ git add README.md
A@pcA$ git status
```

Creamos el archivo README.md y lo agregamos, esto lo pone en área “Staging”

INICIALIZACIÓN EN “COMPUTER A”: OPCION 3

```
A@pcA$ mkdir python1
A@pcA$ cd python1
A@pcA$ git init
A@pcA$ touch README.md
A@pcA$ git add README.md
A@pcA$ git status
A@pcA$ git commit –m 'Primer commit'
A@pcA$ git log
A@pcA$ git push origin master
```

Los “commiteamos”, o sea, quedan listos para subir y “pusheamos”, subimos los cambios a la rama master

WORKFLOW REALIZADO HASTA AHORA

¿QUÉ HACEMOS EN “COMPUTER B”?

```
B@pcB$ git clone https://github.com/minuevorepo.git
B@pcB$ echo “Hola mundo” >> README.md
B@pcB$ git add .
B@pcB$ git commit “cambio en readme”
B@pcB$ git push origin master
```

Clonamos el proyecto desde el servidor, editamos el readme, add, commit, push

SEGUIMOS TRABAJANDO Y RECIBIMOS LOS CAMBIOS EN A

```
A@pcA$ echo Chau mundo” >> README.md
A@pcA$ git commit -m “nuevos cambios”
A@pcA$ git pull
```

Actualizamos el repositorio (git fetch + git merge = git pull)

CONFLICTO!

```
! [rejected] master -> master (fetch first)
```

Y AHORA!?

RESOLVEMOS LOS CONFLICTOS EN A

```
A@pcA$ echo “Otra cosa” >> README.md
A@pcA$ git commit -m “nuevos cambios”
A@pcA$ git pull
A@pcA$ git status
En la rama master
Your branch and 'origin/master' have diverged,
and have 1 and 1 different commit each, respectively.
  (use "git pull" to merge the remote branch into yours)
nothing to commit, working directory clean
```

Vemos un nuevo estado

RESOLVEMOS LOS CONFLICTOS EN A

```
# editamos el archivo como queremos que quede
A@pcA$ git add README.md
A@pcA$ git commit # Sin mensaje
A@pcA$ git push origin master
```

Una vez resuelto: add, commit, push

SIEMPRE VA A SER ASÍ DE COMPLICADO?

NO!

¿POR QUÉ LOS CONFLICOS?

Depende cuán bien se maneje el sistema de ramas (branches) del proyecto. Considerar que siempre que se trabaje sobre el mismo archivo al mismo momento existirán conflictos, por ende, hay que coordinar bien la división de tareas y modularizar al máximo el código.

¿POR QUÉ LOS CONFLICOS?

Lo ideal sería tener las ramas:

PARA FINALIZAR

En git existe un archivo muy importante que en proyectos grandes siempre se usa:

.gitignore

.GITIGNORE

Es un archivo de texto en el que se agregan carpetas y archivos que no querramos que *git* los 'trackee', es decir, que no los suba ni baje del repositorio. Por ejemplo:

- Archivos de configuracion (Base de datos, editor de texto)
- Dependencias (PySimpleGUI, pattern, etc)
- Mas cosas que no querramos

LINKS DE INTERÉS

- Listado de comandos
- Github guides
- Una guía sencilla de git
- Libro: Pro git