**GITHUB**

Github es una plataforma/servidor que aloja nuestros repositorios de Git.

**Clone/Fork**

En Github puedes crear repositorios remotos y puedes clonar o descargar ese mismo repositorio para enlazarlo con el repositorio git local en tu equipo. Si le damos a descargar no tendremos el repositorio git dentro del directorio que nos bajamos, en cambio, si clonamos (sea con HTTPS o SSH) tendremos el repositorio .git dentro del directorio listo.

También podemos hacer **fork** a un proyecto de otra persona/entidad/grupo. El fork nos va a clonar el proyecto para poder trabajar desde nuestro repositorio como si fuera un proyecto a parte.

**SSH y HTTPS**

Para comunicarse entre el repositorio local y el repositorio remoto (Github), es necesario utilizar uno de los dos protocolos de comunicación y seguridad, SSH o HTTPS.

Para utilizar SSH necesitarás crear una llave pública y una llave privada con **ssh-keygen**. HTTPS es mucho más sencillo de utilizar porque no necesitas crear ni configurar absolutamente nada, solo que cada vez que vayas a ejecutar una operación que requiera una comunicación entre el remoto y el local te pedirá usuario y contraseña para autenticar.

Lo interesante de SSH es la comodidad de no tener que estar escribiendo una contraseña cada vez que quieres hacer algo en Github además de asegurarte que nadie podrá ver la contraseña.

Los dos protocolos son seguros, solo escoge el que mejor te convenga.

**Crear llave SSH:**

* Creamos la llave -> **ssh-keygen -t rsa -b 4096 -C “Email de Github”**
  + Nos pedirá que especifiquemos si queremos cambiar el directorio y nombre de la llave; mejor dejarlo tal cual de viene por defecto (Users/usuario/.ssh/id\_rsa).
  + También nos pedirá una contraseña para la llave. Podemos añadirla o dejarla sin contraseña pero es recomendable por seguridad añadir alguna contraseña.
* Copiamos la llave -> **pbcopy < ~/.ssh/id\_rsa.pub**
* Pegamos la llave ssh copiada en configuración de Github.

**Añadiendo un repositorio remoto**

Es necesario crear una conexión entre tu repositorio local y tu repositorio remoto para poder compartir el código en Github. Para eso vamos a dar uso del comando **git remote**, dándole como nombre habitual al repo remoto “origin”.

Para agregar un repositorio remoto tenemos que escribir en la terminal **git remote add [repositorio remoto] [SSH/HTTPS].** Ej:

* **git remote add origin git@github.com:DCueto/git-y-github.git**

Para eliminar el repositorio remoto podemos utilizar el comando **git remote remove [repositorio remoto]**.

* **git remote** -> Muestra los repositorios remotos creados.
* **git remote -v** -> Muestra las conexiones existentes.

**Traer cambios desde el repositorio remoto al repositorio local**

Antes de poder enviar cambios a Github, es necesario traer todos los cambios en el repositorio remoto para tener el mismo contenido.

Para traer los cambios del repositorio remoto al local vamos a utilizar el comando **git fetch**. En este comando tenemos que especificar el repositorio remoto del cual queremos coger esos datos y especificar la rama en nuestro repositorio local donde queremos que se alojen. Ej:

* **git fetch origin master** -> git fetch [repositorio remoto] [rama en repo local]

Con este comando solo trae esos datos pero no los mezcla en nuestra rama principal. Nosotros tenemos que hacer el merge de la rama que ha creado el fetch (“origin/master”) con ese nuevo contenido, con la rama principal. Tenemos que ejecutar el siguiente merge:

* **git merge origin/master**

Si la rama que hemos descargado contiene algún commit (generalmente suele ser así), nos saltará un error: **“fatal: refusing to merge unrelated histories”**. Para solucionarlo tenemos un flag **--allow-unrelated-histories** que nos permite relacionar y mezclar los commits que tiene el repositorio remoto con los de nuestro repositorio local.

Si quieres hacer el fetch y el merge al mismo tiempo y que git se encargue de hacerlo automáticamente, tenemos un comando que se llama **git pull**. Tan solo tenemos que especificar el repositorio remoto y la rama en la que queremos que haga el merge.

* **git pull origin [rama donde hacer el merge]** -> **git pull origin master**

**Enviando cambios al repositorio remoto**

Ya sabemos cómo traer los datos del repositorio remoto al repositorio local. Para hacer lo contrario, enviar datos del repositorio local al repositorio remoto, vamos a dar uso al comando **git push**. Cuando el proceso del comando termine de completarse tendremos todo el repositorio en Github.

* **git push [repositorio remoto] [rama]** -> Ej: **git push origin master**

El push no solo puede enviar ramas, también nos permite enviar tags de las ramas. Solo hay que especificar el flag **--tags** después de la rama: **git push origin master --tags**.

**Crear un template para Issues**

En GitHub podemos crear templates para que a la hora de generar un Issue tenga un formato y ofrezca el feedback que buscamos para resolver el error.

1. Crear archivo llamado **issue\_template.md**
2. Definir los lineamientos para reportar el issue.
3. Hacer el commit.

Ahora, al crear un issue, automáticamente nos lo generará con esta plantilla y nos podremos asegurar de que nos puedan dar la información que necesitamos.

**Crear un template para Pull Request**

GitHub te permite usar templates a la hora de generar Pull Request.

1. Crear archivo llamado **pull\_request\_template.md**
2. Definir los lineamientos usando sintaxis markdown.
3. Hacer commit.

Con la plantilla todo será más ordenado cuando otros generen contribuciones a tu proyecto.

**Ignorando archivos no deseados**

Si tienes archivos que no pueden ser públicos, como archivos de configuración con contraseñas, lo ideal es que no los subas a tu repositorio. Estos archivos los puedes poner en el archivo **.gitignore**.

Es recomendable dar uso a [**https://gitignore.io**](https://gitignore.io) para poder comprobar que tipo de archivos en nuestro proyecto deberíamos de incluir en el .gitignore.

**Pull Request**

Un Pull Request es una solicitud al administrador del repositorio para que realice un merge de los cambios que realizó en su rama. Es la base de la colaboración Open Source en GitHub.

**¡Consejo!** -> Si eres el dueño del repositorio, debes proteger la rama master, para que siempre que alguien externo, pida permiso para hacer aportes al repositorio. Esto evita que te puedan hacer inyecciones de cambios directamente a master sin nuestra supervisión. (Puedes proteger otras ramas a parte de “master” para exigir un pull request).

Pasos del **Pull Request**:

1. Haces el commit de tu código en otra rama.
2. Vas a GitHub y le das a hacer nuevo Pull Request comparando con la rama que quieres que haga merge con master u otra rama protegida en nuestro proyecto.
3. Añades el reviewer para que compruebe tu Pull Request.
4. El reviewer comprueba ese pull request y si encuentra algún fallo te lo notificará para que puedas remediarlo. Cuando el reviewer considere que todo está correcto aprobará el request y hará el merge con la rama que pediste hacer el merge.

**Issues y Milestones**

**Issues**: Una de las características sociales de GitHub es la sección de Issues donde podemos reportar fallos y dar feedback.

**Milestones**: Cuando un proyecto es demasiado complejo y hay demasiados Issues, lo mejor es tener un Milestone. Un Milestone nos ayuda a agrupar issues que tienen una o muchas características en común y tener una barra de progreso para ir solucionando todos esos issues o feedback.

**GitHub Pages**

GitHub Pages te ofrece la posibilidad de alojar tu página web en GitHub, sin tener que alojarlo en un servidor externo. El único “inconveniente” es que no podemos tener nada que contenga un backend, pero sí que permite alojar archivos estáticos (HTML, CSS, Javascript...). Por ejemplo, podemos tener nuestro portafolio en GitHub Pages ;).

Solo es necesario ir a Settings de nuestro repositorio e indicar que rama queremos que utilice para el GitHub Page.

PD: También podemos añadir un dominio personalizado en GitHub Pages, en vez del que nos ofrece automáticamente GitHub.