

PRÁCTICA 1: Instalación de herramientas

1- Objetivo

El objetivo de esta práctica es adquirir unas nociones de como usar Github como plataforma de desarrollo y Git como sistema de control de versiones de código fuente de Software.

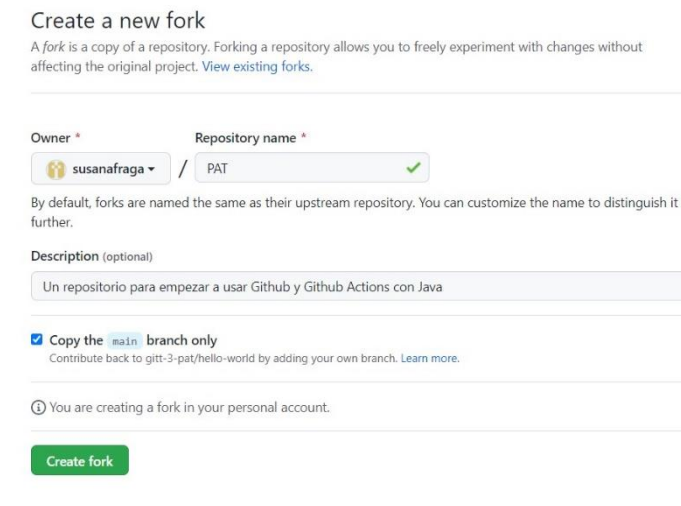
Github es una plataforma que permite crear proyectos de manera abierta tanto de herramientas como aplicaciones. Principalmente se usa para almacenar el código de aplicaciones desarrolladas por diferentes usuarios y además permite descargarlas e incluso colaborar con el desarrollo de las mismas. Tal y como su nombre indica, internamente hace uso de Git, un sistema de control de versiones.

En pocas palabras, se podría definir Git como un sistema que va guardando las distintas versiones que se van haciendo de un proyecto que está subido a github. Además de ir guardando las nuevas versiones con las modificaciones, mantiene las versiones antiguas por si algo falla o para fusionar distintas versiones en una mejorada.

A continuación, se hará uso de esta plataforma y se descubrirán distintos comandos de Git que se pueden realizar.

2- Prerrequisitos

Para poder desarrollar esta práctica se va a hacer uso de un repositorio de prueba de github llamado *hello-world* que está ubicado en el github de la asignatura (<https://github.com/gitt-3-pat/hello-world>). Un repositorio de github es en donde se pueden guardar todos los archivos de un proyecto. En primer lugar, para poder trabajar sobre él, vamos a copiarlo en nuestro perfil de github haciendo un **fork**.



The screenshot shows the 'Create a new fork' interface on GitHub. At the top, it says 'Create a new fork' and explains that a fork is a copy of a repository. Below this, there are two input fields: 'Owner' with a dropdown menu showing 'susanafraga' and 'Repository name' with a text input showing 'PAT'. A green checkmark is next to the repository name. Below these fields, there is a note about default naming and a 'Description (optional)' text area containing 'Un repositorio para empezar a usar Github y Github Actions con Java'. There is a checked checkbox for 'Copy the main branch only' with a link to 'Learn more'. At the bottom, there is a green 'Create fork' button and a note indicating the fork is being created in a personal account.

En mi caso, he decidido llamar al repositorio PAT y estará disponible en la siguiente dirección, que se corresponde con mi perfil de github: <https://github.com/susanafraga/PAT>.

La función **fork** hace que se cree una copia de un repositorio en tu propia cuenta de github, por lo que en este caso tendremos un duplicado de lo que se encontraba en el repositorio *hello-world*. Esta función nos permitirá realizar cambios sobre él en la nube con **Codespaces** (es un workspace en la nube). Para el resto de la práctica estaremos usando **Codespaces** y desde ahí probaremos los comandos correspondientes.

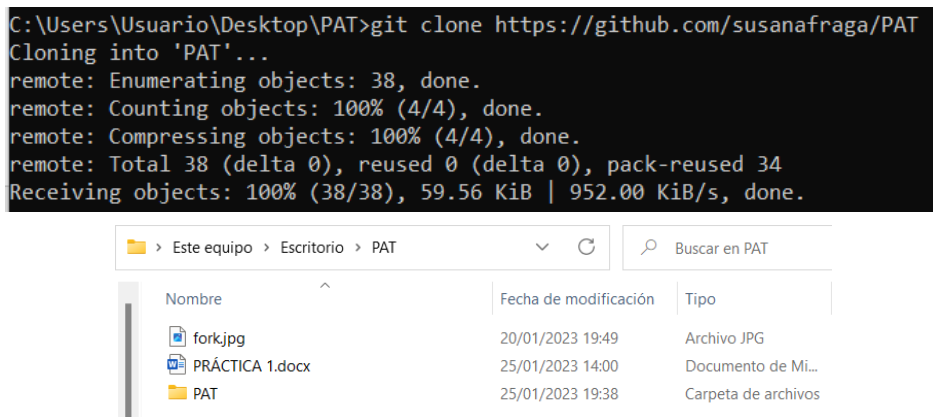
3- Desarrollo de la práctica

Comenzaremos probando los diversos comandos de Git en la nube con nuestro entorno de trabajo de **Codespaces** (menos *git clone* que se hará de manera local) y se explicará su funcionalidad.

Al final del documento se encuentran las capturas del software instalado.

GIT CLONE

Además de poder editar el repositorio en la nube existe otra forma de hacer cambios sobre el repositorio. Esto consiste en clonar el repositorio en nuestro ordenador y trabajar de forma local. Para ello se utiliza el comando **git clone** con la dirección del repositorio que se quiere copiar en el ordenador.



De esta manera obtenemos el mismo repositorio con todo lo que contiene en nuestro ordenador y podremos editarlo con programas que admitan programación en java como pueden ser IntelliJ o Visual Studio Code. A diferencia de hacerlo en la nube, cuando estamos trabajando en local, no se realiza un guardado en github de manera automática como si ocurre al usar Codespaces, por lo que, si se quisiese usar en github, habría que subirlo de nuevo.

Es por ello, que para probar el resto de los comandos se usará la opción de Codespaces.

GIT STATUS

El segundo comando que se va a probar es el comando **git status**. Este comando muestra el estado en el que se encuentra el directorio en el que estamos trabajando y permite ver si tenemos que añadir al repositorio nuevos cambios que hemos hecho. Es decir, nos indica si se han producido cambios en el directorio.

```
@susanafraga →/tmp $ cd /tmp/PAT
@susanafraga →/tmp/PAT (main) $ git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
```

Como se puede ver en la imagen anterior, nos encontramos en directorio *main* del repositorio y, como no se ha hecho ningún cambio en ningún archivo, no tenemos nada que modificar (*nothing to commit*). En cambio, en este caso, pasa lo siguiente:

```

● @susanafraga → /workspaces/PAT (main X) $ git status
On branch main
Your branch is up to date with 'origin/main'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    comandos
    hello-world/

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

```

Como vemos, ha detectado que se han añadido dos archivos y nos dice que debemos usar el comando **git add** para confirmar el cambio o modificación.

GIT ADD

Para que el cambio que se ha realizado del archivo se guarde en el repositorio se deben usar tres comandos seguidos. El primero de ellos es **git add**, tal y como se nos indicaba en **git status**. Se puede usar de dos maneras distintas: seleccionando el archivo que queremos guardar o directamente aplicarlo a todos los archivos que se han modificado. Vamos a usar la segunda opción porque es más sencilla y guarda los dos archivos directamente. Para ello, se hace lo siguiente:

```

● @susanafraga → /workspaces/PAT (main X) $ git add .

```

GIT COMMIT

Tras seleccionar con **git add** los archivos que queremos guardar en el repositorio, pasamos a usar **git commit**. Este comando lo que hace es confirmar una instantánea de los cambios realizados en el archivo y guardarla en el directorio git (no en tu repositorio). Es decir, recoge los cambios realizados y los guarda para que, con el próximo comando, se actualicen y añadan a tu repositorio.

```

● @susanafraga → /workspaces/PAT (main) $ git commit -m "Prueba de commit"
[main 315e44f] Prueba de commit
2 files changed, 3 insertions(+)
create mode 100644 comandos
create mode 160000 hello-world

```

En este caso, como se puede ver, se confirma que dos archivos se han modificado y se han producido 3 inserciones. Para comprobar si haría lo mismo para eliminarlo, hemos decidido eliminar el segundo archivo *hello-world* haciendo el mismo procedimiento que antes: **git status**, **git add**, **git commit**:

```

● @susanafraga → /workspaces/PAT (main) $ git commit -m "Prueba de commit"
[main b10864a] Prueba de commit
1 file changed, 1 deletion(-)
delete mode 160000 hello-world

```

GIT PUSH

Ya como último paso, se debe usar el comando **git push**. Este comando sirve para guardar en el repositorio los archivos que se han modificado (tanto añadido como eliminado). Se puede hacer también de dos maneras. Una primera indicando la dirección a la que se quiere

enviar y guardar directamente. La segunda, sin poner ningún argumento ya que git entiende que lo quieres guardar y enviar a la rama principal (main). En este caso, como queremos guardarlo en el main, se usará la segunda manera:

```
● @susanafraga → /workspaces/PAT (main) $ git push
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 2 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 572 bytes | 572.00 KiB/s, done.
Total 5 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 1 local object.
To https://github.com/susanafraga/PAT
48fe276..b10864a main -> main
```

Para comprobar que se ha guardado correctamente, vamos a nuestro repositorio y comprobamos que se encuentra el archivo:

 comandos	Prueba de commit	16 minutos ago
--	------------------	----------------

GIT CHECKOUT

Por último, vamos a ver qué ocurre con el comando **git checkout**. Este comando nos permite movernos de una rama a otra dentro del repositorio. Por ejemplo, probando el comando siguiente, se crea una nueva rama y además te lleva a ella:

```
● @susanafraga → /workspaces/PAT (main) $ git checkout -b feature/1
Switched to a new branch 'feature/1'
```

En cambio, si queremos movernos de rama, pero ya existe o está creada, servirá con hacer lo siguiente:

```
● @susanafraga → /workspaces/PAT (feature/1) $ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
```

ENTORNO DE DESARROLLO

JAVA

```
C:\Users\Usuario>java --version
java 16.0.2 2021-07-20
Java(TM) SE Runtime Environment (build 16.0.2+7-67)
Java HotSpot(TM) 64-Bit Server VM (build 16.0.2+7-67, mixed mode, sharing)
```

DOCKER

```
C:\Users\Usuario>docker --version
Docker version 20.10.22, build 3a2c30b
```

MAVEN

```
C:\Users\Usuario>mvn --version
Apache Maven 3.8.7 (b89d5959fcde851dcb1c8946a785a163f14e1e29)
Maven home: C:\Program Files\apache-maven-3.8.7
Java version: 16.0.2, vendor: Oracle Corporation, runtime: C:\Program Files\Java\jdk-16.0.2
Default locale: es_ES, platform encoding: Cp1252
OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"
```

EDITOR DE CÓDIGO FUENTE: INTELIJ

 IntelliJ IDEA Community Edition 2021.3.2	16/03/2022 13:03	Acceso directo	2 KB
--	------------------	--------------------------------	------