20 DE ENERO DE 2023

INSTALACIÓN DE HERRAMIENTAS

PRÁCTICA 1

FERNANDO SILJESTROM BERENGUER

Contenido

1. Objetivos

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

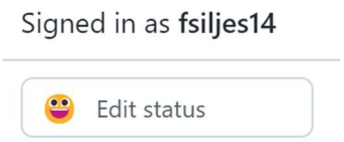
2. Desarrollo de la práctica

Como prerrequisitos de la práctica, debemos tener instalados Java, Maven, VSCode o Intellig y Docker.

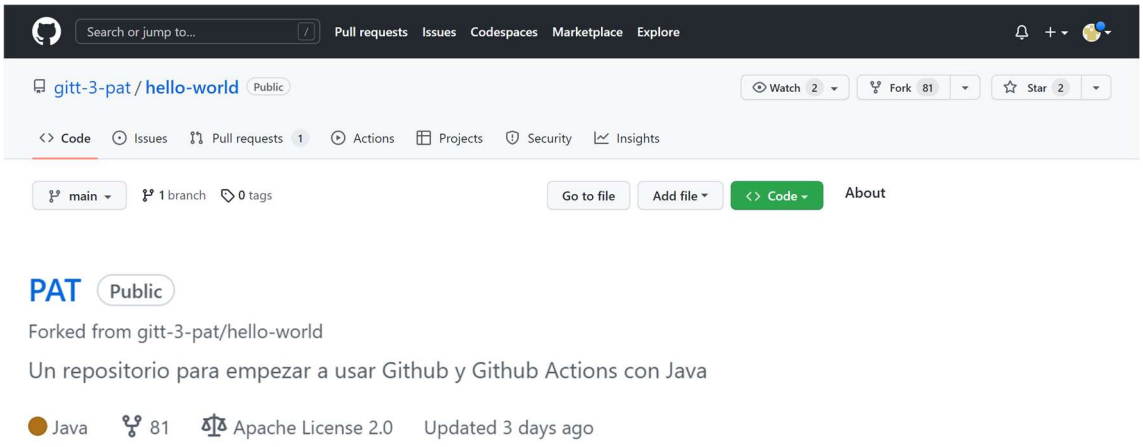
```
C:\Users\ferna>mvn --version
Apache Maven 3.8.7 (b89d5959fcde851dcb1c8946a785a163f14e1e29)
Maven home: C:\Program Files\apache-maven-3.8.7
Java version: 16.0.2, vendor: Oracle Corporation, runtime: C:\Program Files\Java\jdk-16.0.2
Default locale: es_ES, platform encoding: Cp1252
OS name: "windows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

```
C:\Users\ferna>java --version
java 17.0.1 2021-10-19 LTS
Java(TM) SE Runtime Environment (build 17.0.1+12-LTS-39)
Java HotSpot(TM) 64-Bit Server VM (build 17.0.1+12-LTS-39, mixed mode, sharing)
```

Lo primero que debemos hacer es crear una cuenta en Github para poder hacer el *fork* del repositorio que se nos proporciona.



A continuación, hacemos el *fork* del repositorio de Moodle, y si todo ha ido bien, debería aparecer en los repositorios de nuestro perfil. De esta forma, ya tenemos una copia del repositorio en nuestro GitHub personal.



Una vez hemos hecho el *fork*, clonamos el mismo repositorio utilizando el entorno del propio GitHub. Después de haber ejecutado el comando, aparecerá un repositorio en *tmp* con el nombre PAT.

```
@fsiljes14 ➜/tmp $ git clone https://github.com/fsiljes14/PAT
Cloning into 'PAT'...
remote: Enumerating objects: 38, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 38 (delta 0), reused 0 (delta 0), pack-reused 34
Unpacking objects: 100% (38/38), 59.54 KiB | 1.98 MiB/s, done.
@fsiljes14 ➜/tmp $
```

```
@fsiljes14 ➜/tmp/PAT (main) $ git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
```

En la imagen anterior se puede observar como nos encontramos dentro del repositorio que hemos clonado y probamos el comando *git status*. Como era de esperar, no aparece ningún cambio sin hacer *commit* ya que todavía no hemos hecho nada.

Sin embargo, si ahora creamos un fichero en el repositorio, ya cuando ejecutamos status van a aparecer cambios. En mi caso, creé *comandos.txt* y *fernando.txt*, los cuales se muestran en la siguiente imagen.

```
@fsiljes14 ➜/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.txt
        fernando.txt
```

Una vez vemos que hemos modificado archivos, para que se guarden correctamente en el repositorio hay que seguir una serie de pasos. En primer lugar, debemos añadir estos cambios para que se guarden en la nube, y se puede hacer de varias maneras. Si solo queremos añadir el cambio de un fichero, ejecutamos el comando *add* seguido de la ruta del archivo del que

queremos guardar los cambios. En mi caso, como quiero guardar los cambios de ambos archivos, puedo poner *add .* y de esta forma se guardan los cambios de todos los archivos modificados.

```
@fsiljes14 ➜ /workspaces/PAT (main X) $ git add .
```

Después de ejecutar *add*, hay que confirmar los cambios haciendo *commit,* y lo identificamos como *"practica1"*

```
@fsiljes14 ➜ /workspaces/PAT (main) $ git commit -m "practica1"
[main 5fe0ace] practica1
 2 files changed, 3 insertions(+)
 create mode 100644 comandos.txt
 create mode 100644 fernando.txt
```

Finalmente, lo único que tenemos que hacer para terminar la ejecución de los cambios es indicar la dirección del envío de estos. Como quiero que se envíen a la rama principal, ejecuto el comando *git push origin main.* Si quiero enviarlo a una rama distinta, nada más tengo que añadir después de *origin* el nombre de la rama.

```
@fsiljes14 ➜ /workspaces/PAT (main) $ git push origin main
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 2 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (4/4), 424 bytes | 424.00 KiB/s, done.
Total 4 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/fsiljes14/PAT
   48fe276..5fe0ace  main -> main
```

Por último, el comando *git checkout* nos permite crear ramas y navegar por ellas. Para movernos entre ellas, simplemente hay que poner el comando seguido del nombre de la rama a la que nos queremos cambiar. Si no le ponemos nada después, nos muestra información de la rama en la que estamos, que en mi caso es la *main*.

```
@fsiljes14 ➜ /workspaces/PAT (main) $ git checkout
Your branch is up to date with 'origin/main'.
```

Como comandos adicionales, podemos experimentar con *git pull*, *git branch* o *git merge.*

Git pull es muy valioso cuando estamos trabajando con más personas a la vez. De esta forma, antes de ponerte a trabajar ejecutas este comando para obtener la última versión del repositorio, ya que ha podido haber cambios.

Por otro lado, *git branch* sirve para mostrar, borrar o crear ramas. Si ponemos simplemente *git branch,* mostramos las ramas existentes.

```
@fsiljes14 ➜/workspaces/PAT (main) $ git branch
* main
```

En este caso, vemos que solo existe la rama *main.* Sin embargo, vamos a crear alguna otra para mostrar el funcionamiento de estos comandos.

```
@fsiljes14 ➜/workspaces/PAT (main) $ git checkout -b rama_fernando
Switched to a new branch 'rama_fernando'
```

```
@fsiljes14 ➜/workspaces/PAT (rama_fernando) $ git branch
  main
* rama fernando
```

```
@fsiljes14 ➜/workspaces/PAT (main) $ git branch -D rama_fernando
Deleted branch rama_fernando (was 5fe0ace).
```

Para acabar, en la anterior secuencia de imágenes se muestra la creación y cambio a la rama *rama_fernando,* se han mostrado las ramas existentes y por último hemos borrado la rama que habíamos creado.