PRÁCTICA 1:
INTRODUCCIÓN A GITHUB

ALUMNO: PABLO MENÉNDEZ RUIZ DE AZÚA
CLASE: 3ºA GITT
ASIGNATURA: PROGRAMACIÓN DE APLICACIONES TELEMÁTICAS
PROFESOR: ÓSCAR SANZ SAN SEBASTIÁN
FECHA DE ENTREGA: 29/01/2022

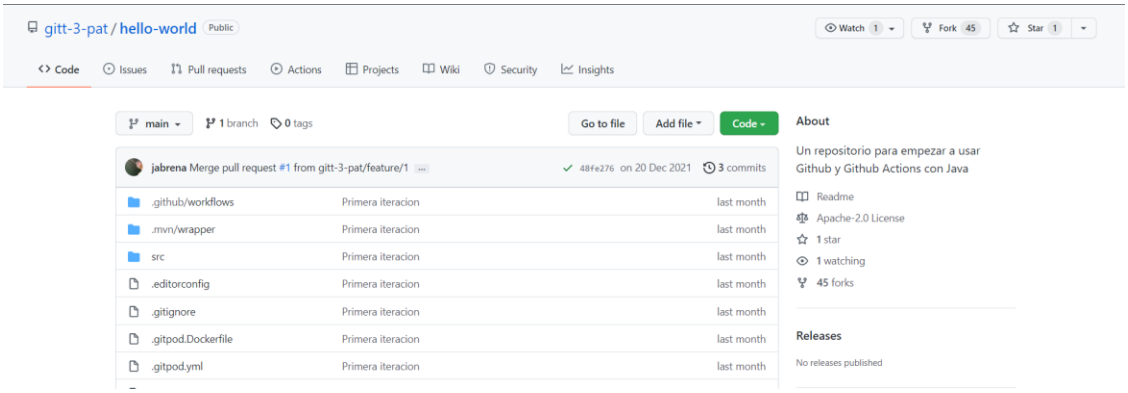
Índice

Objetivo

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

Primeros pasos

Para poder probar los diferentes comandos, vamos a utilizar un repositorio de prueba llamado hello-world que se encuentra en el github de la asignatura. Para copiar este repositorio a nuestro perfil y poder hacer cambios sobre él, hacemos un fork sobre el repositorio: https://github.com/gitt-3-pat/hello-world.



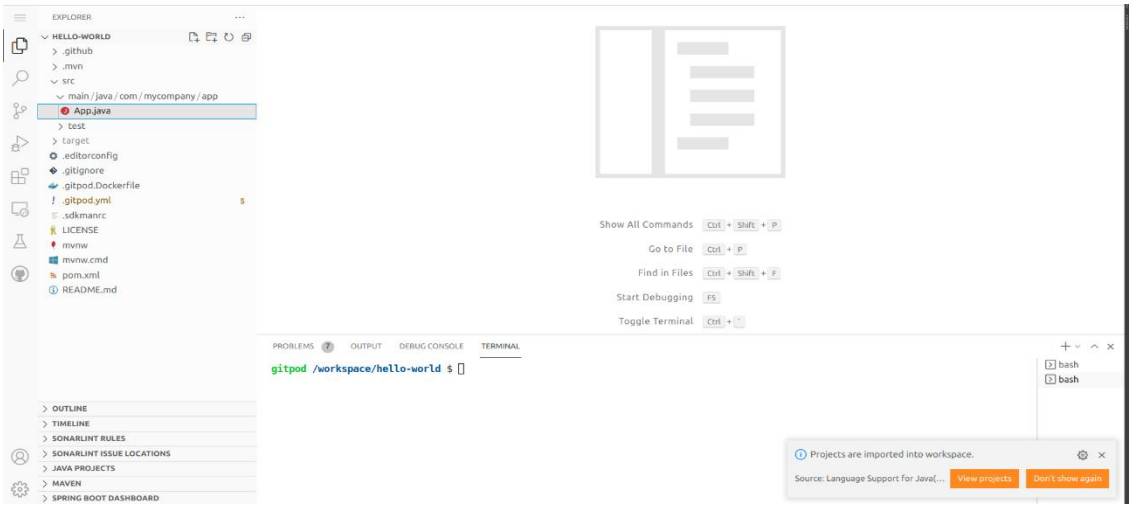
Una vez hecho el fork, tendremos el respositorio en nuestro perfil. En mi caso, estará en: https://github.com/Menendez6/hello-world. Para poder empezar a editar los archivos y utilizar los comandos de git, vamos a ver dos opciones. La primera es clonar el repositorio en nuestro ordenador y trabajar en local. Para ello, utilizamos el comando **git clone** como vemos en la siguiente imagen.

```
pablo@filete-ruso:~/Documentos/Tercero/PAT$ git clone https://github.com/Menendez6/hello-world.git
Clonando en 'hello-world'...
remote: Enumerating objects: 38, done.
remote: Counting objects: 100% (38/38), done.
remote: Compressing objects: 100% (23/23), done.
remote: Total 38 (delta 1), reused 31 (delta 0), pack-reused 0
Desempaquetando objetos: 100% (38/38), 58.95 KiB | 862.00 KiB/s, listo.
```

Al clonar el repositorio, se crea una carpeta con el nombre del repositorio en la dirección donde lo hayamos clonado. Dentro de esta carpeta estarán los mismos archivos que hay en el repositorio. Para editar los archivos, podemos utilizaremos o bien Visual Studio Code o Intellij. Además, como vamos a trabajar con archivos de java, necesitaremos tener instalado en nuestro ordenador la última versión de java (17) y el gestor de proyectos Maven.

```
pablo@filete-ruso:~/Documentos/Tercero/PAT/hello-world$ mvn -version
Apache Maven 3.6.3
Maven home: /usr/share/maven
Java version: 17.0.1, vendor: Oracle Corporation, runtime: /usr/lib/jvm/java-17-oracle
Default locale: es_ES, platform encoding: UTF-8
OS name: "linux", version: "5.13.0-27-generic", arch: "amd64", family: "unix"
```

La segunda opción para editar los archivos es hacerlo en la nube. Para ello, utilizaremos gitpod, que nos permite abrir repositorios de github en la nube, de la misma forma que lo haríamos en local con Visual Studio Code. Para abrir un repositorio, es tan fácil como sincronizar gitpod con tu cuenta de github y abrir uno de tus repositorios, de esta forma se crea un entorno de trabajo.



Desarrollo de la práctica

Una vez tenemos ya nuestro repositorio abierto en un entorno de desarrollo, ya sea en local o en la nube, vamos a empezar a probar comandos de git. Comenzamos por el comando más básico que es "**git status"**. Este comando nos sirve para ver la rama en la que estamos, si tenemos que añadir al repositorio nuevos cambios que hemos hecho y si hay algún commit ya hecho para que podamos subir los archivos al repositorio.

```
gitpod /workspace/hello-world $ git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
```

Como vemos en la imagen anterior, estamos en la rama "main" y no hemos hecho cambios a ningún archivo. Sin embargo, si modificamos un archivo y hacemos "git status", obtenemos lo siguiente.

```
gitpod /workspace/hello-world $ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   src/main/java/com/mycompany/app/App.java

no changes added to commit (use "git add" and/or "git commit -a")
```

Como vemos, ha detectado que hemos cambiado el archivo App.java y nos pide o bien que modifiquemos los cambios en el repositorio o que dejemos el documento como estaba.

En primer lugar, para entender todos los comandos, vamos a probar el "git restore". En nuestro caso, como habíamos cambiado el archivo App.java, introducimos el comando ***git restore*** */workspace/hello-world/src/main/java/com/mycompany/app/App.java*. De esta forma, el archivo App.java vuelve a su versión anterior y se revierte el cambio realizado.

Sin embargo, si queremos que ese cambio se guarde en nuestro repositorio de github, tenemos que hacer uso de una combinación de tres comandos o pasos. Para que se entienda mejor, voy a hacer la analogía con el envío de un paquete. En primer lugar, utilizamos el comando "**git add"** para seleccionar los archivos que hemos modificado y cuyos cambios queremos que se guarden en la nube. En el caso del paquete, sería como seleccionar los elementos que queremos enviar o meter en la caja. Si solo queremos actualizar un archivo, por ejemplo, el App.java, solo hay que escribir el comando *git add /workspace/hello-world/src/main/java/com/mycompany/app/App.java*. En el caso en el que queramos añadir todos los archivos que han sufrido cambios utilizamos el comando "git add .".

Una vez, hemos seleccionado los archivos a guardar o actualizar en nuestro repositorio, debemos hacer un "commit". Básicamente el "commit" es como meter todos los elementos que queremos enviar dentro del paquete y cerrarlo. "git commit" creará una instantánea de los cambios y la guardará en el directorio git.

```
gitpod /workspace/hello-world $ git commit -m "Segundo commit"
[main 1d0f928] Segundo commit
 1 file changed, 1 insertion(+)
```

Finalmente, siguiendo la analogía del paquete, lo único que nos queda es poner la dirección a la que lo queremos enviar y enviarlo. Para ello, utilizamos el comando **git push.** Si no le pasas ningún argumento a "git push", se entiende que lo vas a enviar a la rama principal. Sin embargo, si quieres hacer push a otra rama, debes utilizar *git push origin rama1***.**

```
gitpod /workspace/hello-world $ git push
Enumerating objects: 17, done.
Counting objects: 100% (17/17), done.
Delta compression using up to 16 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (9/9), 582 bytes | 582.00 KiB/s, done.
Total 9 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/Menendez6/hello-world.git
   05b0cd3..1d0f928  main -> main
```

Una vez hemos entendido los comando básicos para actualizar uno o varios ficheros, vamos a ver cómo trabajar con ramas.

Para crear una rama utilizamos el comando **git checkout.** Además, este comando también nos ayuda a navegar por ellas. Si utilizamos el comando **git checkout -b <nombre-rama>,** se crea automáticamente una rama y se cambia a ella. Para movernos de una rama a otra, utilizamos simplemente **git checkout <nombre-rama>.**

```
gitpod /workspace/hello-world $ git checkout rama1
Switched to branch 'rama1'
```

Además, para trabajar con ramas, existe otro comando que es el **git branch.** Este comando, se utiliza para listar, crear o borrar ramas. Si queremos listar todas las ramas del repositorio:

```
gitpod /workspace/hello-world $ git branch
  main
  rama1
* rama2
```

Si queremos borrar una rama, utilizamos **git branch -d <branch-name>**

```
gitpod /workspace/hello-world $ git branch -D rama2
Deleted branch rama2 (was 1d0f928).
```

Otro comando muy importante es el **git pull.** Se utiliza para fusionar todos los cambios que se han hecho en el repositorio remoto con el repositorio local. Muy útil cuando estás trabajando con más gente en un proyecto. Por ejemplo, yo antes he modificado el repositorio desde gitpod pero previamente me lo había clonado en local. Por lo tanto, he hecho una serie de cambios que mi ordenador no ha guardado en local. Para ello hacemos pull.

```
pablo@filete-ruso:~/Documentos/Tercero/PAT/hello-world$ git pull
remote: Enumerating objects: 35, done.
remote: Counting objects: 100% (35/35), done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 27 (delta 7), reused 25 (delta 5), pack-reused 0
Desempaquetando objetos: 100% (27/27), 1.53 KiB | 142.00 KiB/s, listo.
Desde https://github.com/Menendez6/hello-world
   48fe276..1d0f928  main       -> origin/main
 * [nueva rama]      rama1      -> origin/rama1
Actualizando 48fe276..1d0f928
Fast-forward
 src/main/java/com/mycompany/app/App.java | 2 ++
 1 file changed, 2 insertions(+)
```

Ahora ya tengo el repositorio actualizado en mi ordenador.

Si queremos fusionar dos ramas, se utiliza el comando **git merge.**

```
gitpod /workspace/hello-world $ git merge rama1
Auto-merging src/main/java/com/mycompany/app/App.java
CONFLICT (content): Merge conflict in src/main/java/com/mycompany/app/App.java
Automatic merge failed; fix conflicts and then commit the result.
```

Como vemos, no podemos mergear porque nos aparece un conflicto. Para ver el conflicto podemos utilizar el comando **git diff**. Solucionamos el conflicto aceptando los dos cambios, en este caso. Una vez solucionado el conflicto, para terminar de hacer merge, tenemos que hacer el proceso anterior de actualizar un archivo. Es decir, add + commit + push.

Una vez hecho eso, si volvemos a hacer merge, nos dirá que está todo actualizado.

```
gitpod /workspace/hello-world $ git merge rama1
Already up to date.
```

Por último, un comando de git muy útil que en esta práctica no hemos utilizado es **git init.** Este comando nos sirve para inicializar un repositorio en local. Es decir, en vez de clonar el repositorio, lo creamos en nuestro ordenador y luego ya lo subimos con los comandos ya vistos.