25 DE ENERO DE 2023

PRÁCTICA 1

INSTALACIÓN DE HERRAMIENTAS

GONZALO ALONSO LIDÓN

INTRODUCCIÓN

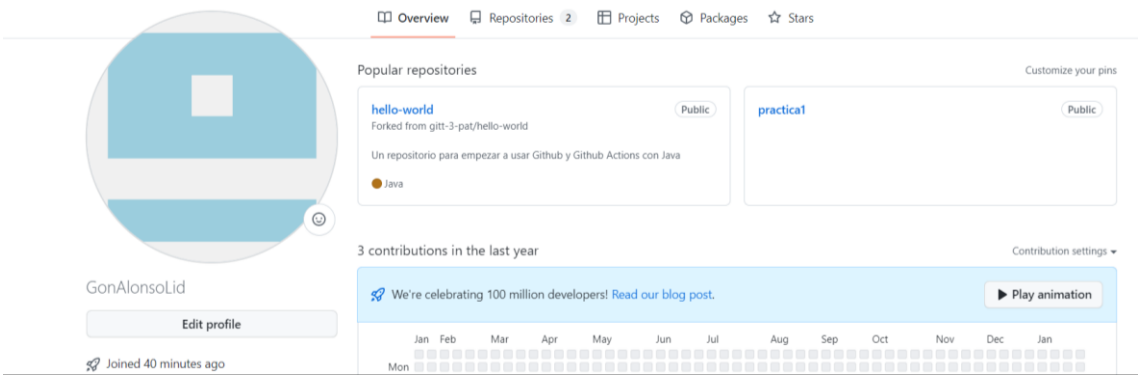
Para el correcto desarrollo de la asignatura deberemos de hacer uso de diferentes herramientas a lo largo del curso. Para ello es crucial que instalemos estas dentro de nuestro propio sistema, y hagamos pruebas sobre estas para comprobar su correcto funcionamiento.

Como practica inicial se procede a la instalación y "set-up" de todas estas herramientas, así como su entorno de trabajo, sobre todo el github y los repositorios que han de estar presentes en estas.

DESARROLLO

1. Instalación y creación de cuenta de GitHub, junto al repositorio personal.

En primer lugar, procedemos a la creación de una cuenta en GitHub como herramienta fundamental del curso. Esta creación la hacemos por medio del "e-mail" de la universidad de manera que acabamos con lo siguiente:



Como se puede ver, se ha procedido a la creación del repositorio personal, pero público, "PRACTICA 1".

De la misma manera, hemos hecho un fork al repositorio del profesor para tener acceso a este sin tener que buscarlo desde el enlace de la asignatura.

2. Desarrollo de los comandos vistos en la práctica.

- **"git clone"**:

El siguiente comando básicamente hace una copia del repositorio que se encuentre en el link al que hagamos referencia. Este repositorio al que hacemos referencia se copia en el nuestro, creando así un acceso directo a todo el contenido que tuviera. La siguiente imagen muestra la correcta copia del directorio en nuestro repositorio:

```
@GonAlonsoLid →/workspaces/hello-world (main) $ git clone https://github.com/gitt-3-pat/hello-world
Cloning into 'hello-world'...
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 | 1.35 MiB/s, done.
```

- **"git status"**:

El siguiente comando nos dice el estado del repositorio en el que estemos trabajando, así como el "staging area". Esto nos permitirá saber si nuestro repositorio esta trabajando correctamente. Procedemos a ejecutar el comando sobre nuestro repositorio:

```
@GonAlonsoLid →/workspaces/hello-world (main) $ 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 b
e committed)
        hello-world/

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

Como podemos ver en la imagen el código nos dice que el repositorio está funcionando correctamente, por lo que nos asegura que nuestro entorno esta preparado para que le hagamos peticiones, ya sean por medio de servidores HTTP u otros entornos.

- **"git add"**:

El siguiente comando nos permite guardar cambio sobre nuestro repositorio de manera fácil y rápida desde la línea de comandos.

```
@GonAlonsoLid →/workspaces/hello-world (main) $ git add file.txt
@GonAlonsoLid →/workspaces/hello-world (main) $
```

En la imagen anterior, se ha creado el archivo "file.txt" para que este luego se pueda guardar en nuestro repositorio sin ningún tipo de error por parte del GitHub. Este tipo de sentencias nos permiten "preparar" a un documento, archivo o grupo de archivos para que reciban una serie de actualizaciones por medio de confirmaciones posteriores. Añade un índice a archivos especificados y funciona como un "checkpoint".

- **"git commit":**

El comando commit hace que se guarde un documento con los cambios realizados en el directorio sobre documentos específicos. Por ello siempre va precedido del comando visto anteriormente "git add". Este guarda una captura de los cambios hechos, y no pueden ser alterados por el GitHub.

```
@GonAlonsoLid →/workspaces/hello-world (main) $ git commit
hint: Waiting for your editor to close the file...
```

Extra:

```
@GonAlonsoLid →/workspaces/hello-world (main) $ git commit -m "commit message"
[main 9b8f0f8] commit message
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 file.txt
@GonAlonsoLid →/workspaces/hello-world (main) $
```

Si sucedemos al comando "git commit" con un "-m + MENSAJE" estaremos creando un atajo con una confirmación de manera que confirmamos directamente la creación del documento con los cambios sin necesidad de abrir el editor de texto.

- **"git push"**:

El comando que veremos a continuación tiene la función de guardar en nuestro repositorio remoto todo lo que se encuentre en el repositorio local para que otros puedan tener acceso a ese código y trabajar sobre él. Es muy útil a la hora de compartir códigos o archivos de código sin necesidad de compartir el archivo directamente con la persona, limitando solo el compartir del link de nuestro repositorio remoto.

```
@GonAlonsoLid →/workspaces/hello-world (main) $ git push --all
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 2 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 292 bytes | 292.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/GonAlonsoLid/hello-world
   48fe276..9b8f0f8  main -> main
 * [new branch]      prueba -> prueba
```

Como podemos ver nos guarda el repositorio con el que estamos trabajando en local, en la dirección de uno de mis repositorios remotos, encontrado en https://github.com/GonAlonsoLid/hello-world.

- **"git checkout"**:

El siguiente comando nos permite volver a aquellos "checkpoints" o versiones de un mismo proyecto para alterarlos, o avanzar hasta el ultimo para seguir trabajando. Es muy útil a la hora de evaluar posibles errores en los códigos del proyecto, así como para ver el progreso de nuestro proyecto de una forma muy temporal. Para que este comando sea efectivo tenemos que hacer uso antes de un comando llamado "git branch", que crea esos puntos de referencia en el proyecto.

De la misma manera, estas "branches" se pueden crear también por medio de este comando añadiendo la terminación "-b NOMBRE DE BRANCH NUEVO", que interpreta ese nuevo punto de referencia como el punto actual del proyecto.

```
@GonAlonsoLid →/workspaces/hello-world (feature/1) $ git checkout main
D       file.txt
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
```

Como podemos ver en la imagen anterior, el sistema ha cambiado al punto de main, en este caso siendo el punto actual del proyecto.

INSTALACIÓN DEL SOFTWARE EN LOCAL

En las siguientes imágenes se prueba, por medio de los comandos pertinentes en el terminal del ordenador, que ambos Maven y Java están instalados correctamente en el sistema.

```
PS C:\Users\gonza> mvn --version
Apache Maven 3.8.7 (b89d5959fcde851dcb1c8946a785a163f14e1e29)
Maven home: C:\Program Files\apache-maven-3.8.7
Java version: 17.0.1, vendor: Oracle Corporation, runtime: C:\Program Files\Java\jdk-17.0.1
Default locale: es_ES, platform encoding: Cp1252
OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"
```

```
PS C:\Users\gonza> 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)
```

Para la instalación del software anterior se ha procedido también a la inclusión de las referencias en el “path” dentro de las variables del entorno como MAVEN_HOME y JAVA_HOME.

También se ha llevado a cabo la instalación de Docker Desktop como herramienta fundamental en local.

CONCLUSION

Con todo lo anterior, tenemos una introducción a las nociones básicas del entorno de GitHub para su uso a lo largo del curso. El uso del entorno de programación CODESPACE utilizado a lo largo de la practica puede ser el idóneo, pero también pueden utilizarse la combinación de herramientas instaladas en local entre las cuales esta Docker, Java, IntelliJ y Maven, para trabajar en nuestro ordenador y luego subirlo a nuestro repositorio remoto en caso de así desearlo.