Silvia García Lage
3º GITT + BA, PAT
29/01/2023

Práctica 1: Instalación de herramientas

URL de mi repositorio: https://github.com/201901639/hello-world.git

1. Comandos a probar en github

- **git clone**

Lo primero que vamos a hacer en todos los proyectos es crear un repositorio. En el repositorio se guardan todos los archivos de un proyecto. Una vez creado el proyecto, añadiremos código para darle funcionalidad. Nos descargaremos el proyecto para poder trabajar con él. Para descargarme el proyecto utilizaremos el primer comando a probar en el esta práctica que es:

```
@201901639 ➜ /workspaces/hello-world (main) $ git clone https://github.com/201901639/hello-world.git
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 | 1.04 MiB/s, done.
```

La diferencia entre descargárselo directamente y clonarlo es que, al clonarlo, Git irá registrando los cambios que voy haciendo al proyecto.

- **git status**

El comando git status muestra el estado del directorio de trabajo y del área del entorno de ensayo. Permite ver los cambios que se han preparado, los que no y los archivos en los que Git no va a realizar el seguimiento. El resultado del estado no muestra ninguna información relativa al historial del proyecto.

```
@201901639 ➜ /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 be committed)
        Lab1
        hello-world/

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

- **git add .**

Para indicarle a Git los cambios a realizar se usa git add. Al ejecutar git add, los archivos modificados pasan al estado “staged”, esto es, listos para el commit. Si un archivo está “staged” le ha indicado a Git que quiere que formen parte del commit. Ejecutando el siguiente comando le indico a Git que quiero que todos los archivos modificados estén “staged”.

```
@201901639 ➜/workspaces/hello-world (main) $ git add .
warning: adding embedded git repository: hello-world
hint: You've added another git repository inside your current repository.
hint: Clones of the outer repository will not contain the contents of
hint: the embedded repository and will not know how to obtain it.
hint: If you meant to add a submodule, use:
hint:
hint:   git submodule add <url> hello-world
hint:
hint: If you added this path by mistake, you can remove it from the
hint: index with:
hint:
hint:   git rm --cached hello-world
hint:
hint: See "git help submodule" for more information.
```

```
@201901639 ➜/workspaces/hello-world (main) $ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   Lab1
        new file:   hello-world
```

- **git commit**

Un commit es la forma de guardar los cambios. Los commits, además, suelen llevar una descripción indicando qué se ha hecho o modificado para que otros programadores o tu yo del futuro sepan qué contiene ese commit. Para realizar un commit hay que utilizar el siguiente comando.

```
@201901639 ➜/workspaces/hello-world (main) $ git commit -m "Qué has hecho en este commit?"
[main fb416eb] Qué has hecho en este commit?
 2 files changed, 11 insertions(+)
 create mode 100644 Lab1
 create mode 160000 hello-world
```

- **git push**

Para actualizar los cambios el repositorio original tengo que ejecutar git push

```
@201901639 →/workspaces/hello-world (main) $ git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 2 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 958 bytes | 958.00 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/201901639/hello-world
   48fe276..fb416eb  main -> main
```

- **git checkout -b feature/1**

Una rama es una copia del proyecto, más específicamente, del proyecto en un commit particular. La rama tiene el beneficio de que es independiente de las otras ramas, por lo que no se ve afectada por cambios y commits de las otras. Esta función se suele utilizar para trabajar sobre partes específicas del proyecto sin interferencias. Pongamos que quiero crear un documento extra de información, aparte del README. Para ello, puedo crear una rama y realizar los cambios sobre ella.

```
@201901639 →/workspaces/hello-world (main) $ git checkout -b feature/1
Switched to a new branch 'feature/1'
```

- **git checkout main**

Quiero unir las ramas, para que la principal (main) tenga el documento de información. Para eso tengo que ejecutar git merge desde la rama principal. Como estoy en feature/1, tengo que pasar primero a main.

```
@201901639 →/workspaces/hello-world (feature/1) $ git checkout main
M       Lab1
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
```

Silvia García Lage
3º GITT + BA, PAT
29/01/2023

2. Instalación de herramientas

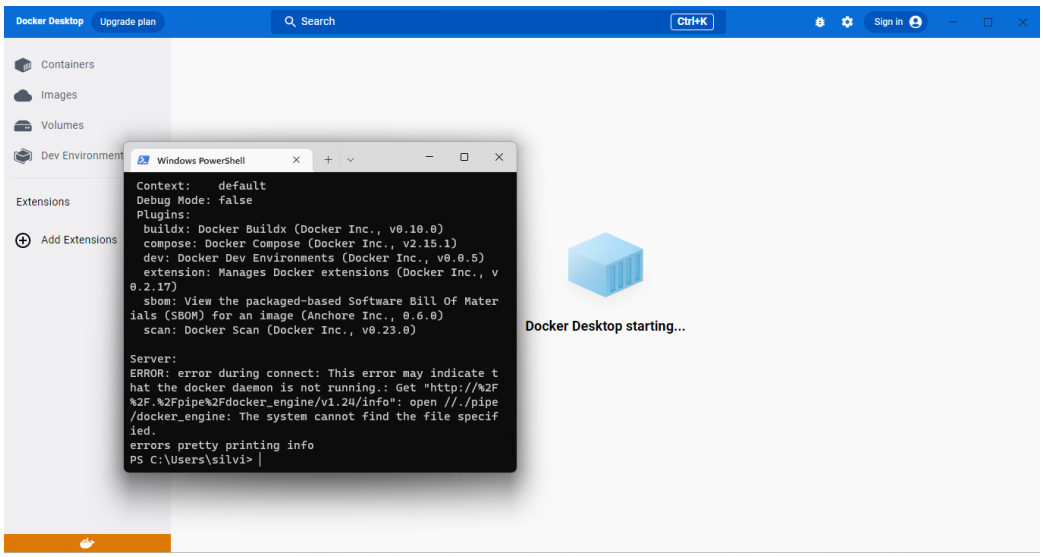
- Java

```
PS C:\Users\silvi> java -version
java version "1.8.0_361"
Java(TM) SE Runtime Environment (build 1.8.0_361-b09)
Java HotSpot(TM) Client VM (build 25.361-b09, mixed mode)
PS C:\Users\silvi>
```

- Maven

```
PS C:\Users\silvi> mvn -version
Apache Maven 3.8.7 (b89d5959fcde851dcb1c8946a785a163f14e1e29)
Maven home: C:\Program Files\apache-maven-3.8.7
Java version: 16.0.2, vendor: Eclipse Foundation, runtime: C:\Program Files\Eclipse Foundation\jdk-16.0.2.7-hotspot
Default locale: es_ES, platform encoding: Cp1252
OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"
PS C:\Users\silvi>
```

- Docker



- VS code

```
PS C:\Users\silvi> code --version
1.74.3
97dec172d3256f8ca4bfb2143f3f76b503ca0534
x64
```