

# Práctica 1: Plataforma GitHub & Git

Claudia Blanco García 201905597

3ºA GITT

El objetivo de esta primera práctica es familiarizarse con el uso de GitHub como plataforma de desarrollo y aprender a utilizar diferentes entornos de desarrollo como IntelliJ y Gitpod.

## Desarrollo de la práctica

Para empezar, es necesario instalar Java (versión 17) y Maven. También se utilizará IntelliJ (previamente instalado para otra asignatura) y Gitpod.

Se puede comprobar la versión de Java por consola:

```
C:\Users\claux>java -version
java version "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)
```

Se comprueba lo mismo para Maven:

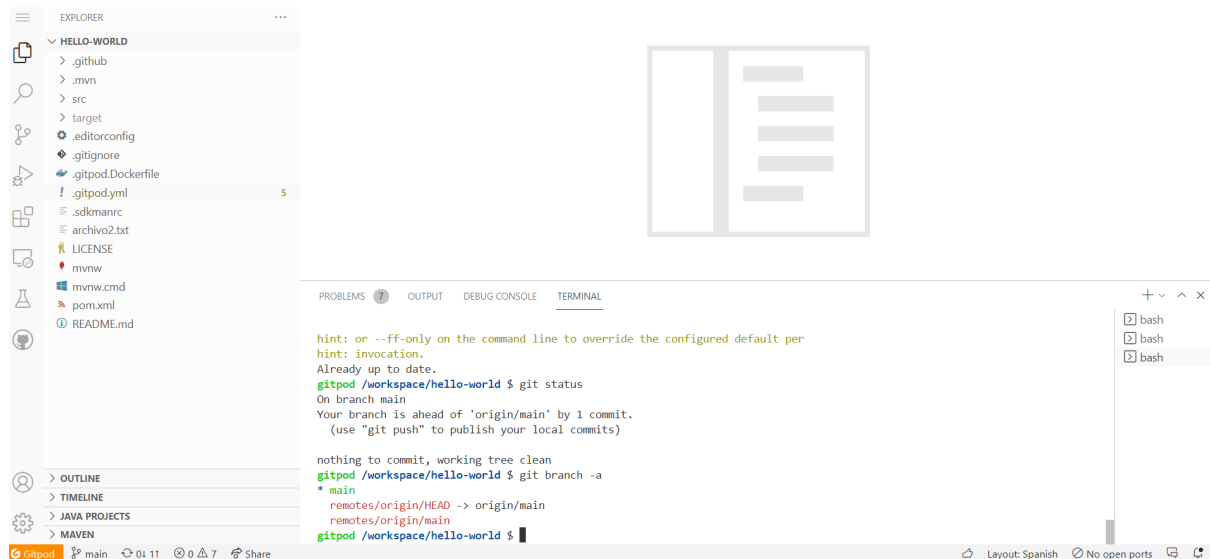
```
C:\Users\claux>mvn -version
Apache Maven 3.8.2 (ea98e05a04480131370aa0c110b8c54cf726c06f)
Maven home: C:\Program Files (x86)\apache-maven-3.8.2
Java version: 17.0.1, vendor: Oracle Corporation, runtime: C:\Program Files\Java\jdk-17.0.1
Default locale: es_ES, platform encoding: Cp1252
OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"
```

En primer lugar, se hace fork sobre el repositorio proporcionado para la práctica:

<https://github.com/gitt-3-pat/hello-world>



Una vez hecho esto, podemos abrir un entorno de desarrollo en Gitpod, que tendrá el siguiente aspecto:

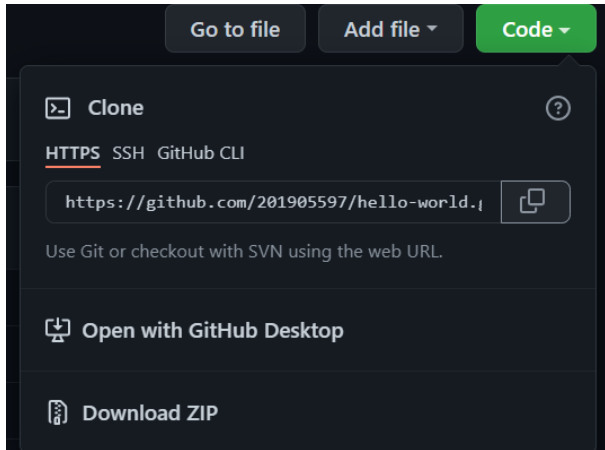


Gitpod es un editor de código en la nube. En esta primera práctica se hace un uso introductorio de dicha IDE, además de usar IntelliJ.

A continuación se prueban una serie de comandos de Git.

### Comando *git clone*

El comando `git clone` sirve para clonar o copiar un repositorio existente en otro directorio. Para esta práctica, después de hacer Fork, se ha utilizado `git clone` a partir de la URL del repositorio en GitHub:



```
C:\Users\claux\OneDrive\Escritorio\pat\patr>git clone https://github.com/201905597/hello-world.git
Cloning into '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
Unpacking objects: 100% (38/38), done.
```

### Comando *git status*

El comando `git status` muestra el estado del área de trabajo o *workspace*: los cambios que se han preparado, los commits que se han realizado...

Si se ejecuta este comando en consola justo después de clonar el repositorio (sin haberlo modificado), se obtiene lo siguiente:

```
C:\Users\claux\OneDrive\Escritorio\pat\patr>git status
On branch feature/1

No commits yet

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

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

A lo largo de este documento se muestran otros ejemplos de uso de este comando.

### Comando *git add*

El comando `git add [fichero]` selecciona el fichero y lo añade para que posteriormente se incluya en el commit; es decir, “prepara” el fichero para el commit.

A continuación se muestran algunos ejemplos de uso de este comando:

Añadimos un archivo de nuestro ordenador a partir de la ruta (consola).

Primero se crea el archivo (`archivo1.txt`) manualmente y se guarda en el directorio `hello-world` (en máquina).

En consola ejecutamos lo siguiente:

```
C:\Users\claux\OneDrive\Escritorio\pat\patr\hello-world>git add archivo1.txt
```

Con git status se comprueba que todo ha ido bien:

```
C:\Users\claux\OneDrive\Escritorio\pat\patr\hello-world>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:   archivo1.txt
```

Se puede observar que el nuevo archivo aparece como “*Changes to be committed*”.

### Comando *git commit*

Este comando sirve para que los archivos modificados (desde el IDE, por ejemplo) registren dichas modificaciones en el repositorio.

Volviendo al ejemplo anterior (archivo1):

```
C:\Users\claux\OneDrive\Escritorio\pat\patr\hello-world>git commit -m "Prueba de commit de archivo1"
[main 9a8fc60] Prueba de commit de archivo1
 1 file changed, 1 insertion(+)
 create mode 100644 archivo1.txt
```

Se ejecuta git status:

```
C:\Users\claux\OneDrive\Escritorio\pat\patr\hello-world>git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```

### Comando *git push*

El comando git push sirve para enviar los archivos ya preparados (con add y commit) a un servidor remoto; es decir, para guardarlos en GitHub.

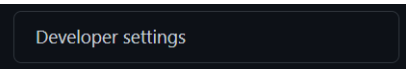
Para poder usar este comando correctamente, es necesario crear un token de acceso personal (*personal access token*) en GitHub. Esto sirve como alternativa al uso de contraseñas al autenticarse por consola.

Para crear el token se siguen los siguientes pasos:

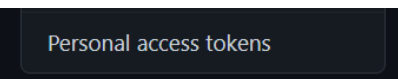
1. Vamos a settings (ajustes) de GitHub

A dark rectangular button with the word "Settings" in white text, circled with a green oval.

2. Vamos a developer settings

A dark rectangular button with the text "Developer settings" in white.

3. Elegimos Personal access tokens

A dark rectangular button with the text "Personal access tokens" in white.

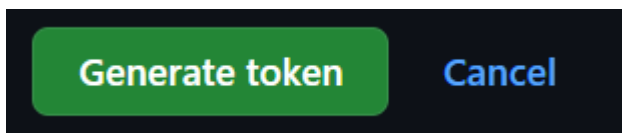
#### 4. Elegimos Generate new token



5. Le damos un nombre al token (*tokenpush*), una fecha de expiración y seleccionamos *repo* para poder usar el token para el control del repositorio:

A dark-themed form titled 'New personal access token'. It includes a text input for the token name containing 'tokenpush', a dropdown for expiration set to '30 days', and a section for selecting scopes. The 'repo' scope is selected, showing sub-options like 'repo:status', 'repo\_deployment', 'public\_repo', 'repo:invite', and 'security\_events'. Other scopes like 'workflow' and 'write:packages' are also listed but not selected.

#### 6. Finalmente, lo generamos.



Después de esto aparecerá el token en pantalla: se podrá ver y copiar solo una vez, por seguridad.

Una vez creado el token, se podrá utilizar para cierto control del repositorio.

Se ejecuta lo siguiente:

```
C:\Users\claux\OneDrive\Escritorio\pat\patr\hello-world>git push origin main
```

Se piden los credenciales de GitHub:

A light-themed 'GitHub Login' dialog box. It has a title bar with 'GitHub Login' and a close button. The main content area has the 'GitHub Login' text. Below it are two input fields: 'Username or email' and 'Password'. At the bottom, there are two buttons: 'Login' (with a checkmark icon) and 'Cancel' (with an 'X' icon). Below the buttons, there are links for 'Don't have an account? Sign up' and 'Forgot your password?'.

En lugar de escribir la contraseña, se escribe el token creado (copiado al portapapeles previamente). Es posible que de error y se vuelvan a pedir los credenciales por consola:

```
Logon failed, use ctrl+c to cancel basic credential prompt.
Username for 'https://github.com': 201905597@alu.comillas.edu
Password for 'https://201905597@alu.comillas.edu@github.com':
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 343 bytes | 171.00 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/201905597/hello-world.git
 48fe276..9a8fc60  main -> main
```

Con git status se comprueba que todo ha ido bien:

```
C:\Users\claux\OneDrive\Escritorio\pat\patr\hello-world>git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
```

Desde Gitpod se puede hacer lo mismo que se ha hecho hasta ahora en consola.

1. Abrimos el proyecto en Gitpod y comprobamos que el archivo 1 está:

```
gitpod /workspace/hello-world $ ls
archivo1.txt  LICENSE  mvnw  mvnw.cmd  pom.xml  README.md  src  target
```

2. Creamos un archivo 2 en la terminal en la nube usando el comando *touch*:

```
gitpod /workspace/hello-world $ touch archivo2.txt
```

3. Añadimos el archivo nuevo (*git add*) y hacemos *commit*. Con *git status* vamos haciendo comprobaciones:

```
gitpod /workspace/hello-world $ git add archivo2.txt
gitpod /workspace/hello-world $ 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:   archivo2.txt
```

```
gitpod /workspace/hello-world $ git commit -m "Commit del archivo2"
[main 57ee548] Commit del archivo2
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 archivo2.txt
```

4. Hacemos *push*:

```
gitpod /workspace/hello-world $ git push origin main
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 16 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 280 bytes | 280.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/201905597/hello-world.git
 9a8fc60..57ee548  main -> main
```

5. Comprobamos que los cambios están subidos en GitHub: efectivamente aparece el nuevo archivo.

 archivo1.txt	Prueba de commit de archivo1	20 hours ago
 archivo2.txt	Commit del archivo2	2 minutes ago

### Comando *git checkout*

El comando `git checkout` permite, entre otras cosas, cambiar entre ramas y restaurar ramas. Por ejemplo, si se ejecuta `git checkout -b rama_nueva` se crea una rama nueva llamada `rama_nueva` y se cambia a ella.

```
C:\Users\claux\OneDrive\Escritorio\pat\patr\hello-world>git checkout -b rama_nueva
Switched to a new branch 'rama_nueva'

C:\Users\claux\OneDrive\Escritorio\pat\patr\hello-world>git branch
  main
* rama_nueva
```

Se ejecuta `git status` en la rama nueva:

```
C:\Users\claux\OneDrive\Escritorio\pat\patr\hello-world>git status
On branch rama_nueva
nothing to commit, working tree clean
```

Para probar, hacemos un cambio en esta nueva rama: añadimos un archivo 3:

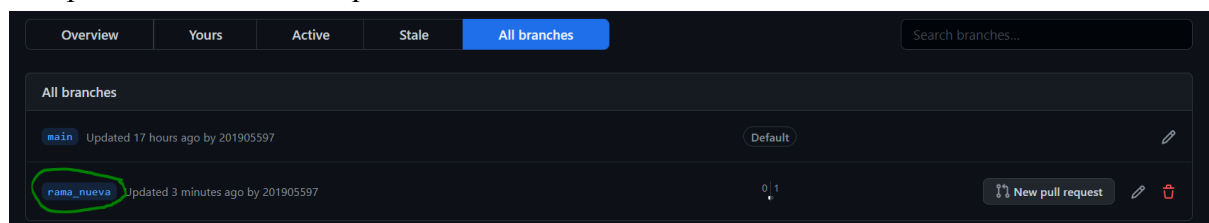
```
C:\Users\claux\OneDrive\Escritorio\pat\patr\hello-world>git add archivo3.txt
```

Hacemos commit y push:

```
C:\Users\claux\OneDrive\Escritorio\pat\patr\hello-world>git commit -m "Commit del archivo3"
[rama_nueva aaf5f03] Commit del archivo3
1 file changed, 1 insertion(+)
create mode 100644 archivo3.txt

C:\Users\claux\OneDrive\Escritorio\pat\patr\hello-world>git push -u origin rama_nueva
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 351 bytes | 351.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
remote:
remote: Create a pull request for 'rama_nueva' on GitHub by visiting:
remote:   https://github.com/201905597/hello-world/pull/new/rama_nueva
remote:
To https://github.com/201905597/hello-world.git
 * [new branch]      rama_nueva -> rama_nueva
Branch 'rama_nueva' set up to track remote branch 'rama_nueva' from 'origin'.
```

Comprobamos en GitHub que la nueva rama está:



Pasamos de nuevo a la rama `main` ejecutando `git checkout main`:

```
C:\Users\claux\OneDrive\Escritorio\pat\patr\hello-world>git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
```

### Comando *git branch*

El comando *git branch* permite visualizar las ramas existentes en local y en remoto y la rama en la que nos encontramos.

Si lo ejecutamos después del paso anterior, nos muestra que estamos en *main* (en verde), pero también muestra la otra rama existente en local:

```
C:\Users\claux\OneDrive\Escritorio\pat\patr\hello-world>git branch
* main
  rama_nueva
```

También existe una opción para ver las ramas en remoto:

```
C:\Users\claux\OneDrive\Escritorio\pat\patr\hello-world>git branch -r
origin/HEAD -> origin/main
origin/main
```

Y también se pueden ver las ramas en local y en remoto:

```
C:\Users\claux\OneDrive\Escritorio\pat\patr\hello-world>git branch -a
* main
  rama_nueva
remotes/origin/HEAD -> origin/main
remotes/origin/main
```

Este comando también permite borrar ramas:

Si creamos una rama nueva llamada *nueva\_rama2* y la queremos eliminar, bastará con ejecutar lo siguiente:

```
C:\Users\claux\OneDrive\Escritorio\pat\patr\hello-world>git branch -D nueva_rama2
Deleted branch nueva_rama2 (was 9a8fc60).
```

### Comando *git merge*

El comando *git merge* permite unificar las ramas independientes creadas con *git branch* en una sola rama.

Por ejemplo, imaginemos que queremos unir la rama *main*, que contiene los archivos 1 y 2, con la *rama\_nueva*, que contiene los archivos 1 y 3. Este comando nos permite unir a la rama actual (*main*) la rama nueva, que tendremos que especificar:

```
C:\Users\claux\OneDrive\Escritorio\pat\patr\hello-world>git merge rama_nueva
Updating 9a8fc60..aaf5f03
Fast-forward
 archivo3.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 archivo3.txt
```




Para que este cambio aparezca en GitHub tenemos que hacer push, aunque antes es necesario ejecutar el comando *git pull*. El comando *git pull* sirve para extraer y descargar contenido desde un repositorio remoto y actualizar con su contenido el repositorio local.

```
C:\Users\claux\OneDrive\Escritorio\pat\patr\hello-world>git pull
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (1/1), done.
remote: Total 3 (delta 1), reused 3 (delta 1), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/201905597/hello-world
  9a8fc60..57ee548  main    -> origin/main
Merge made by the 'recursive' strategy.
 archivo2.txt | 0
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 archivo2.txt
```

Ya podemos hacer push:

```
C:\Users\claux\OneDrive\Escritorio\pat\patr\hello-world>git push origin main
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 335 bytes | 335.00 KiB/s, done.
Total 2 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/201905597/hello-world.git
  57ee548..48214be  main -> main
```

Verificamos que los tres archivos de texto (1, 2 y 3) aparecen en GitHub:

 archivo1.txt	Prueba de commit de archivo1	21 hours ago
 archivo2.txt	Commit del archivo2	20 minutes ago
 archivo3.txt	Commit del archivo3	4 hours ago

## Comando git cherry-pick + rebase

El comando *git cherry-pick* tiene diferentes aplicaciones. Es un comando de mayor complejidad que los mencionados anteriormente. Aunque quizá no quede al alcance de esta primera práctica, a continuación se explica un ejemplo de uso de este comando:

1. El programador, sin darse cuenta, hace varios commit sobre la rama equivocada; por ejemplo, sobre la rama de otro compañero
2. Para evitar repetir todo el proceso y la programación en su propia rama, hace *cherry-pick* para “copiar y pegar” esos commit en su rama
3. Es necesario también usar el comando *rebase* para “reordenar” el historial de commits tal y como debería haber quedado si el programador no se hubiese equivocado

## Uso de IntelliJ

Para concluir, cabe destacar que las operaciones de add, commit y push que se han detallado en este documento pueden hacerse también de manera automática desde IntelliJ, con los siguientes botones:



## Conclusiones

Esta práctica ha servido de introducción a GitHub y al desarrollo colaborativo de código. Cabe destacar que los comandos utilizados son solo algunos de los muchos disponibles en Git, y que a lo largo del curso quizá se utilicen muchos más.

También se han añadido al repositorio algunas capturas de pantalla para dejar evidencia de la instalación del software.

El link al repositorio es el siguiente: <https://github.com/201905597/hello-world>