

PRÁCTICA 1 PAT:

Primero nos descargamos los programas requeridos:

JAVA 17:

Windows x64 Installer	153.91 MB	https://download.oracle.com/java/17/archive/jdk-17.0.11_windows-x64_bin.exe (sha256)
-----------------------	-----------	---

Verifico habérmelo descargado correctamente:

```
PS C:\Users\victo> java -version
java version "17.0.12" 2024-07-16 LTS
Java(TM) SE Runtime Environment (build 17.0.12+8-LTS-286)
Java HotSpot(TM) 64-Bit Server VM (build 17.0.12+8-LTS-286, mixed mode, sharing)
PS C:\Users\victo> |
```

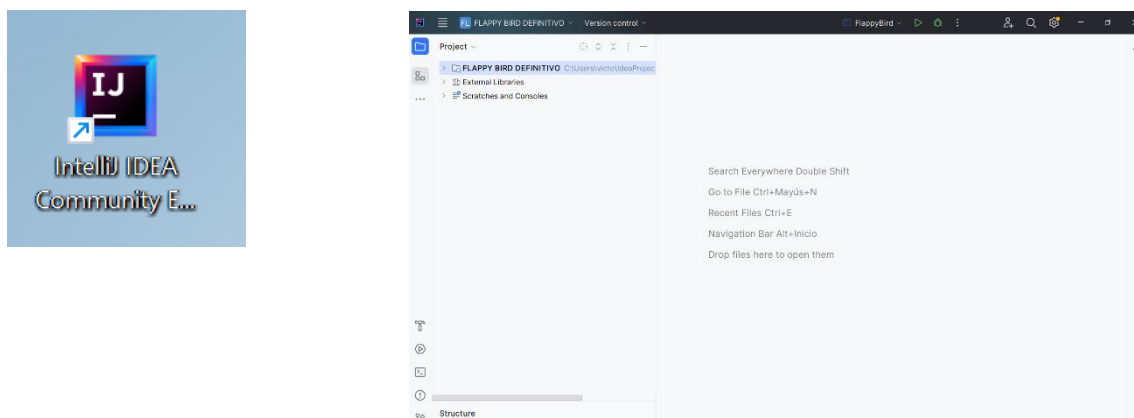
MAVEN:

Me lo descargo directamente desde el powershell de la terminal:

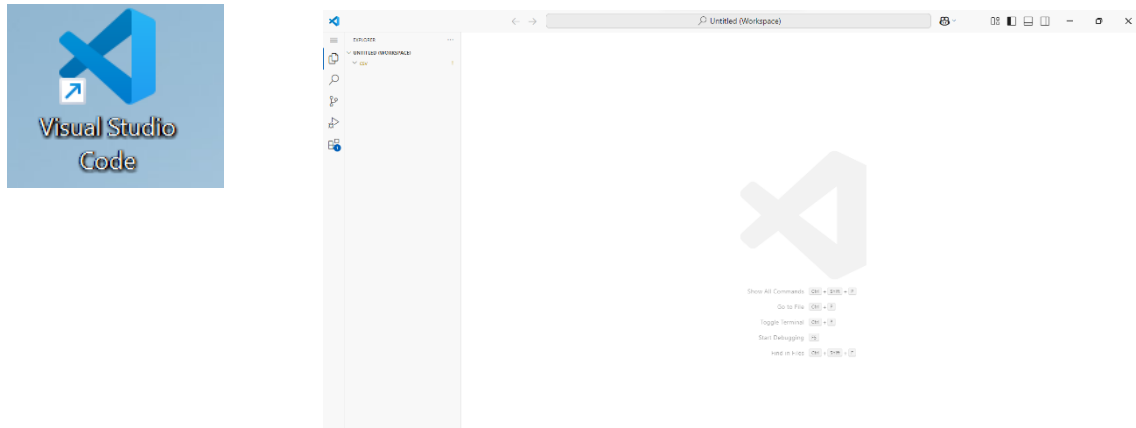
```
PS C:\Windows\system32> mvn -version
>>
Apache Maven 3.9.9 (8e8579a9e76f7d015ee5ec7bfc97d260186937)
Maven home: C:\ProgramData\chocolatey\lib\maven\apache-maven-3.9.9
Java version: 17.0.12, vendor: Oracle Corporation, runtime: C:\Program Files\Java\jdk-17
Default locale: es_ES, platform encoding: Cp1252
OS name: "windows 11", version: "10.0", arch: "amd64", family: "windows"
PS C:\Windows\system32> █
```

INTELLIJ IDEA:

Este ya lo tenía descargado de años pasados:

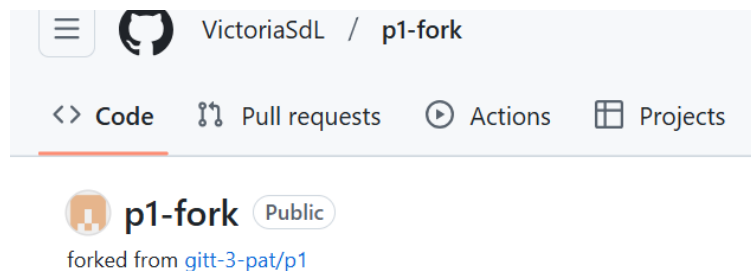


VISUAL STUDIO:



DESARROLLO PRÁCTICA:

Empiezo haciendo el fork del repositorio: <https://github.com/gitt-3-pat/p1>:



Un fork es una copia de un repositorio en tu cuenta de GitHub. Sirve para trabajar en un proyecto sin modificar el original.

Sirve para

1. Tener una copia independiente del proyecto.
2. Trabajar y hacer cambios sin afectar el repositorio original.
3. Poder experimentar o personalizar el proyecto para tu práctica.

Es útil para colaborar en proyectos abiertos o trabajar en tu propia versión de un proyecto.

A continuación me creo un code en codespace y verifico que estoy en la rama main:

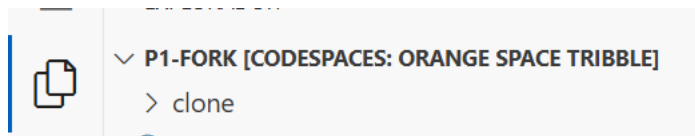
```
@VictoriaSdL → /workspaces/p1-fork (main) $ git status
On branch main
Your branch is up to date with 'origin/main'.
```

Ahora pruebo los siguientes comandos sobre el anterior repositorio:

GIT CLONE:

1. Creo una nueva carpeta y me desplazo a ella:

```
@VictoriaSdL →/workspaces/p1-fork (main) $ mkdir clone
@VictoriaSdL →/workspaces/p1-fork (main) $ cd clone
@VictoriaSdL →/workspaces/p1-fork/clone (main) $
```



2. Clono el mismo repositorio forkeado usando su URL

```
@VictoriaSdL →/workspaces/p1-fork/clone (main) $ git clone https://github.com/VictoriaSdL/p1-fork.git
Cloning into 'p1-fork'...
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (1/1), done.
remote: Total 6 (delta 0), reused 0 (delta 0), pack-reused 5 (from 1)
Receiving objects: 100% (6/6), done.
@VictoriaSdL →/workspaces/p1-fork/clone (main) $
```

Este log en la terminal muestra que el comando git clone se ejecutó correctamente y que el repositorio fue copiado a tu entorno local.

```
Receiving objects: 100% (6/6), done.
@VictoriaSdL →/workspaces/p1-fork/clone (main) $ cd p1-fork
@VictoriaSdL →/workspaces/p1-fork/clone/p1-fork (main) $ ls
README.md
```

Vemos que el archivo readme también se ha copiado bien

Explicación:

Git clone es un comando que sirve para copiar un repositorio remoto (en GitHub u otro servidor) a tu máquina local o entorno de trabajo.

Lo he usado para descargar mi repositorio forkeado (p1-fork) desde GitHub en un entorno en Codespaces. Ahora puedo trabajar en una copia completa del proyecto, editar archivos, y sincronizar cambios con el repositorio remoto.

GIT STATUS:

En mi repositorio clonado:

```

● @VictoriaSdL → /workspaces/p1-fork/clone/p1-fork (main) $ git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean

```

Esto significa que estoy en la rama main y que no hay cambios pendientes en el repositorio. Pues en efecto, el comando `git status` muestra el estado actual del repositorio. Indica:

1. Si hay archivos modificados, nuevos o eliminados.
2. Si hay cambios que no se han agregado al área de preparación (*staging area*).
3. En qué rama te encuentras

En este caso me ha permitido verificar el estado actual del repositorio clonado (en el que me encuentro) y confirmar si hay archivos pendientes de ser añadidos, modificados o listos para hacer un commit(=guardar los cambios de manera permanente en el historial del repositorio).

GIT ADD:

El comando `git add .` agrega todos los archivos y cambios del directorio actual al área de preparación (*staging area*) para incluirlos en el próximo commit.

```

● @VictoriaSdL → /workspaces/p1-fork/clone/p1-fork (main) $ git add .
○ @VictoriaSdL → /workspaces/p1-fork/clone/p1-fork (main) $

```

Miro el estado actual del directorio:

```

● @VictoriaSdL → /workspaces/p1-fork/clone/p1-fork (main) $ git add .
● @VictoriaSdL → /workspaces/p1-fork/clone/p1-fork (main) $ git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean

```

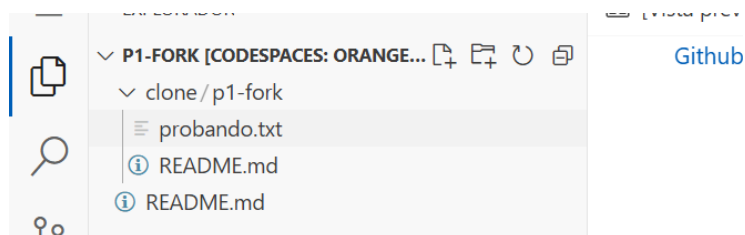
Esto se debe a que no hay archivos nuevos, modificados o eliminados en tu repositorio y por lo tanto `git add .` no tiene nada que preparar para un commit.

Para que tenga sentido la continuación creamos un archivo:

```

● @VictoriaSdL → /workspaces/p1-fork/clone/p1-fork (main) $ echo "Probando" >> probando.txt
○ @VictoriaSdL → /workspaces/p1-fork/clone/p1-fork (main) $

```



Hemos usado el comando `echo` este permite: Añadir texto a un archivo sin sobrescribir el contenido y si el archivo no existe (como es el caso), lo crea.

Ahora si que tiene sentido hacer git add seguido de git commit, pues tenemos cambios que añadir:

```
● @VictoriaSdL → /workspaces/p1-fork/clone/p1-fork (main) $ git add .
○ @VictoriaSdL → /workspaces/p1-fork/clone/p1-fork (main) $
● @VictoriaSdL → /workspaces/p1-fork/clone/p1-fork (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:   probando.txt
```

Este mensaje indica que el archivo probando.txt ha sido añadido al área de preparación (*staged*) y está listo para incluirse en el próximo commit.

GIT COMMIT:

= El comando git commit guarda los cambios que has preparado (en el área de preparación o *staging area*) en el historial del repositorio local, permitiéndote revisar, deshacer o regresar a versiones anteriores.

Lo he usado para:

- guardar cambios (mi archivo de texto: probando.txt) de forma segura en mi repositorio local.
- Crear un historial claro de lo que se hizo en cada etapa del proyecto.

```
● @VictoriaSdL → /workspaces/p1-fork/clone/p1-fork (main) $ git commit -m "Añadir archivo probando.txt"
[main 2e4265b] Añadir archivo probando.txt
1 file changed, 1 insertion(+)
create mode 100644 probando.txt
```

Este mensaje indica que el archivo probando.txt fue añadido al repositorio con 1 línea de contenido y se guardó exitosamente en el historial del proyecto. El siguiente paso es sincronizar estos cambios con el repositorio remoto usando git push.

```
● @VictoriaSdL → /workspaces/p1-fork/clone/p1-fork (main) $ 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
```

GIT PUSH:

El comando git push se utiliza para enviar los cambios locales (commits: como el que hice para probando.txt) al repositorio remoto en GitHub.

(Un repositorio remoto es una copia de mi proyecto que está almacenada en un servidor en la nube o en otro equipo. En el caso de GitHub, el repositorio remoto es la versión de mi proyecto que está alojada en esas plataformas.)

```

@VictoriaSdL →/workspaces/p1-fork/clone/p1-fork (main) $ git push
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), 317 bytes | 317.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/VictoriaSdL/p1-fork.git
07720b5..2e4265b main -> main

```

Esto indica que los cambios locales se comprimieron, enviaron y aplicaron correctamente en el repositorio remoto (main en GitHub). De esta forma, ahora los cambios serán visibles en el repositorio remoto para mi y para otros colaboradores.

GIT CHECKOUT:

El comando git checkout se utiliza para:

1. Cambiar de rama:

Puedes moverte entre diferentes ramas en el repositorio.

```

@VictoriaSdL →/workspaces/p1-fork/clone/p1-fork (main) $ git checkout -b nueva_rama
Switched to a new branch 'nueva_rama'
@VictoriaSdL →/workspaces/p1-fork/clone/p1-fork (nueva_rama) $

```

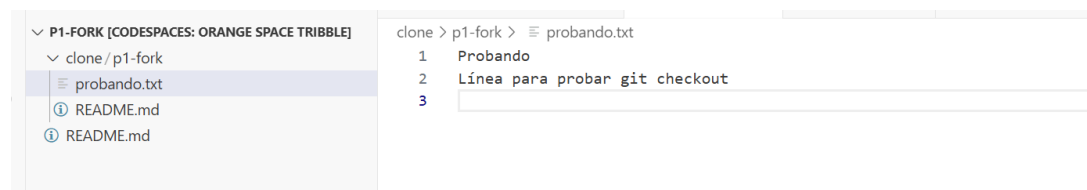
Ejecutando ese código:

- Se crea una nueva rama llamada “nueva_rama”.
- Se cambia automáticamente a esa rama para trabajar en ella.

2. Restaurar archivos:

Restaura un archivo al estado que tenían en el último commit o en una rama específica.

Para probarlo añado una línea a mi documento .txt:



```

@VictoriaSdL →/workspaces/p1-fork/clone/p1-fork (main) $ git checkout -- probando.txt
@VictoriaSdL →/workspaces/p1-fork/clone/p1-fork (main) $

```

clone > p1-fork > probando.txt

```

1 Probando
2

```

Vemos como en efecto git checkout borra automáticamente el último cambio realizado que había sido la escritura de la última línea: “Línea para probar git checkout”.

Por último redacte el readme con la ayuda de esto: <https://docs.github.com/es/get-started/writing-on-github/getting-started-with-writing-and-formatting-on-github/basic-writing-and-formatting-syntax>)

Práctica 1: Introducción a Git y GitHub

Este proyecto corresponde a la Práctica 1 de PAT, donde he aprendido diferentes comandos de git y me he instalado entornos de desarrollo como IntelliJ IDEA y Visual Studio. Esta práctica esta explicada más precisamente en el pdf adjuntado. Aquí daremos una visión general

Requisitos previos

1. JAVA 17: Descarga e instalación verificadas.
2. Apache Maven: Instalado directamente desde PowerShell.
3. IntelliJ IDEA: Ya instalado de prácticas previas.
4. Visual Studio: Configurado para otros proyectos.

Windows x64 Installer 151.91 MB https://download.oracle.com/java/7/archive/jdk-7u011-windows-x64_bin.exe (sha256)

```
PS C:\Windows\system32> mvn -version
>>
Apache Maven 3.9.9 (8e579a9e76f7d015ee5ec7bfc97d260186937)
Maven home: C:\ProgramData\chocolatey\lib\maven\apache-maven-3.9.9
Java version: 17.0.12, vendor: Oracle Corporation, runtime: C:\Program Files\Java\jdk.17
Default locale: es_ES, platform encoding: cp1252
OS name: "windows 11", version: "10.0", arch: "amd64", family: "windows"
PS C:\Windows\system32>
```



Desarrollo de la práctica

1. Hacer un fork del repositorio original

Repositorio original: <https://github.com/gitt-3-pat/p1>

Un fork es una copia independiente del repositorio original. Esto nos permite:

- Trabajar en el proyecto sin modificar el original.
- Personalizar y adaptar el proyecto a nuestras necesidades.
- Colaborar en proyectos abiertos.

2. Clonar el repositorio

Una vez hecho el fork, cloné el repositorio forkeado en Codespaces.

Comandos a probar:

```
git status
```

Este comando muestra el estado actual del repositorio. Te permite ver qué archivos han cambiado, cuáles están listos para ser confirmados y cuáles no.

```
git add .
```

Sirve para agregar todos los archivos modificados al área de preparación. Esto prepara los cambios para el próximo commit.

```
git commit -m "TU MENSAJE"
```

Con este comando confirmas los cambios que agregaste al área de preparación. El mensaje entre comillas describe los cambios realizados.

```
git push
```

Este comando sube los commits confirmados en tu repositorio local al repositorio remoto.

```
git checkout -b feature/1
```

Crea una nueva rama llamada `feature/1` y cambia automáticamente a esa rama. Esto se utiliza para trabajar en cambios específicos sin afectar la rama principal.

```
git checkout main
```