



ETSI ICAI

PRÁCTICA 1

GitHub como plataforma de desarrollo y control de versiones

Javier González Santamaría

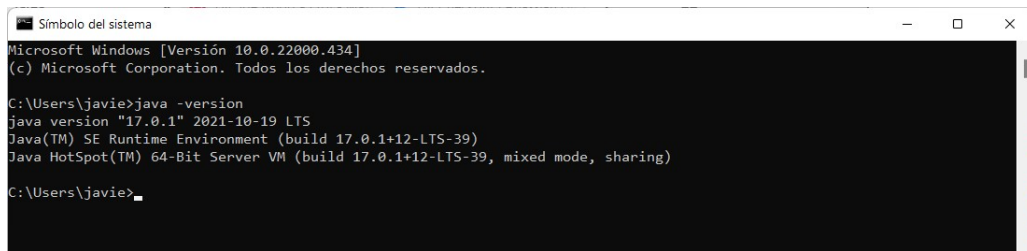
30 ENERO 2022

1. Introducción

El objetivo de esta práctica es conseguir las nociones necesarias para poder trabajar con el software de control de versiones de código fuente de Software y del mismo modo usar como plataforma de desarrollo la herramienta GitHub.

Como requisitos previos a la práctica, se necesita tener instalado en la máquina local java de la versión 17, Maven y un editor de texto, se ha elegido Visual Studio Code.

En la figura 1 se aprecia la versión que se ha instalado de java en el ordenador, mientras que en la figura 2 se tiene la versión de Maven instalada.

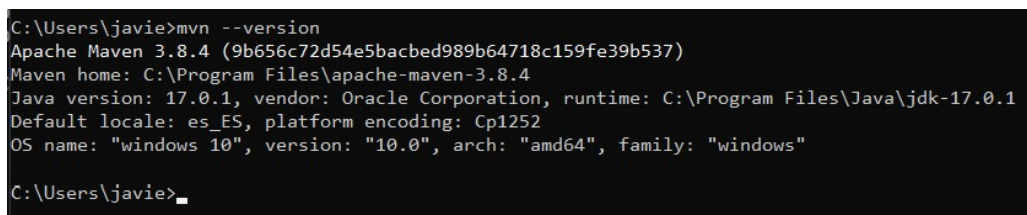


```
Símbolo del sistema
Microsoft Windows [Versión 10.0.22000.434]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\javie>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)

C:\Users\javie>
```

Figura 1: Versión de java instalada



```
C:\Users\javie>mvn --version
Apache Maven 3.8.4 (9b656c72d54e5bacbed989b64718c159fe39b537)
Maven home: C:\Program Files\apache-maven-3.8.4
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"

C:\Users\javie>
```

Figura 2: Versión de Maven instalado

Se ha creado una cuenta en la plataforma de GitHub para poder subir en esta las distintas versiones del código. Ya que con la herramienta git solo se nos almacenan de manera local en nuestra máquina y con GitHub se estaciona en los servidores de dicha empresa. En la figura 3 se aprecia el perfil creado en dicha plataforma.

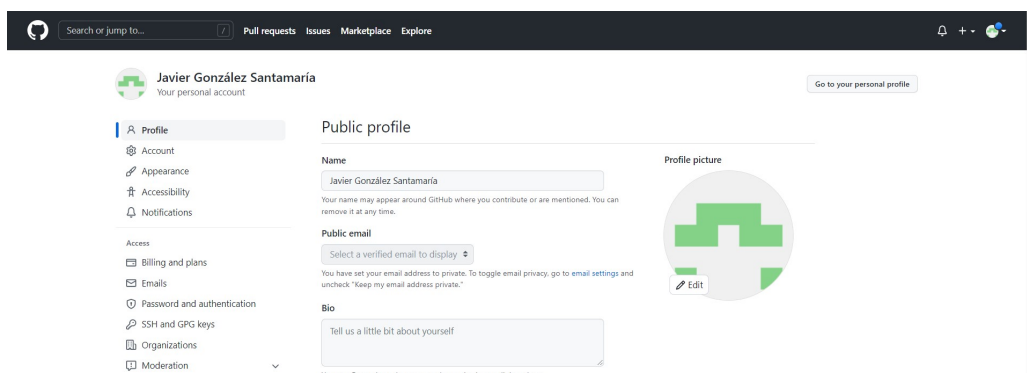


Figura 3: Perfil personal de GitHub

2. Comandos git

En este apartado se verán los distintos comandos que se han usado para esta práctica y las distintas aplicaciones que tienen estos. Cabe destacar que todos estos han sido invocados desde el programa Visual Studio Code en la terminal que se despliega en este editor. Más específicamente git bash. En la figura que se tiene a continuación se ve un ejemplo de dicha terminal donde se va a trabajar durante el resto de la práctica.

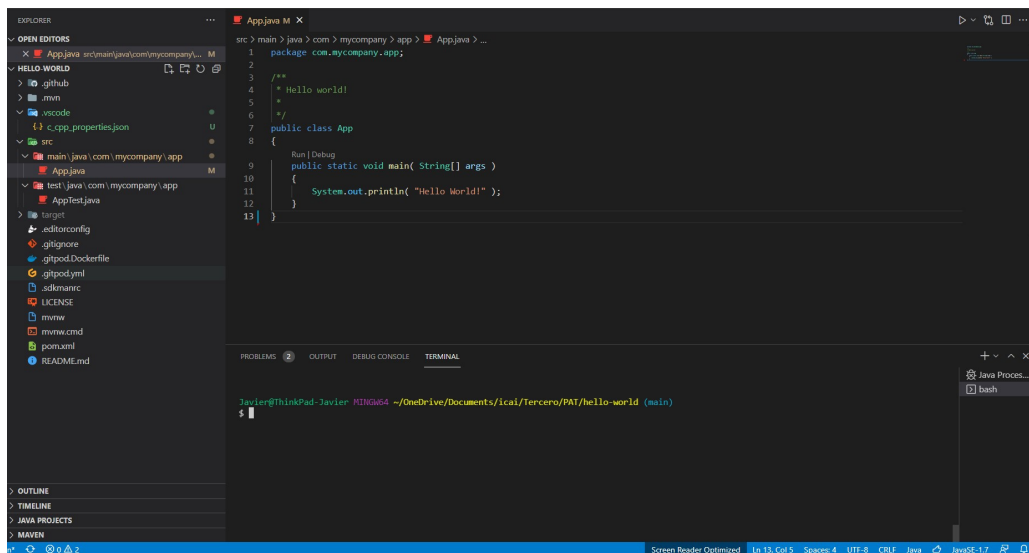


Figura 4: Terminal git bash desde VSCode

2.1. Fork y clone

Estos dos comandos son parecidos en el entorno de git salvo por una diferencia. La finalidad de ambos dos es de replicar un repositorio público en nuestro entorno, la diferencia reside en que el comando fork lo copia en nuestro repositorio de GitHub, es decir, crea una copia en remoto del proyecto en GitHub vinculado al original para poder realizar pull requests a este repositorio original. Mientras que el comando clone lo copia a nuestra máquina local (solo se tendría en local y no se crea un repositorio).

Como ejemplo se ha realizado un fork del repositorio “<https://github.com/gitt-3-pat/hello-world.git>” desde la web de GitHub. Para poder realizar dicho fork nos disponemos en la url antes puesta y pulsamos el apartado de “fork”, el cual nos creará automáticamente en nuestra cuenta esta copia como se ve en la figura 5 (se aprecia que arriba figura “forked from gitt-3-pat/hello-world”), el repositorio de la cuenta de “javierjg001” está vinculado con el original.

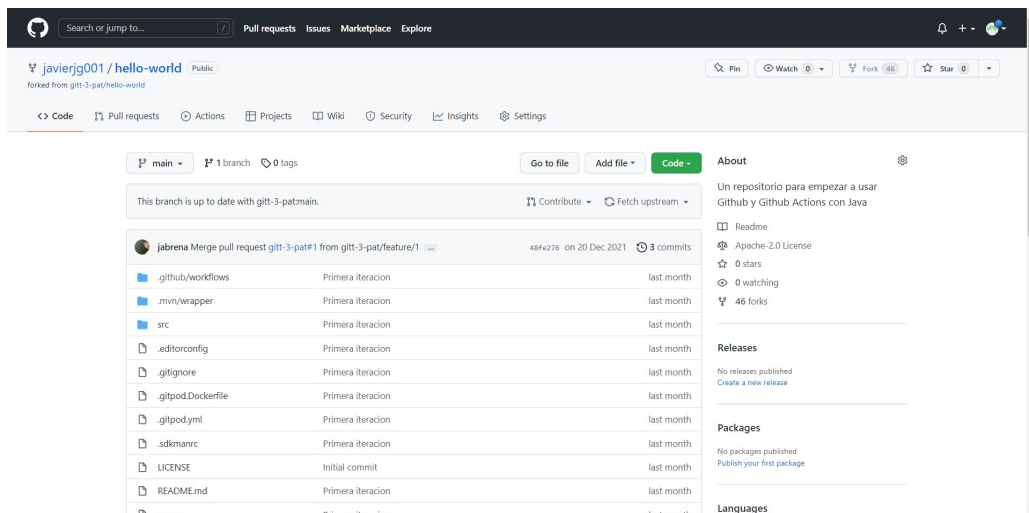
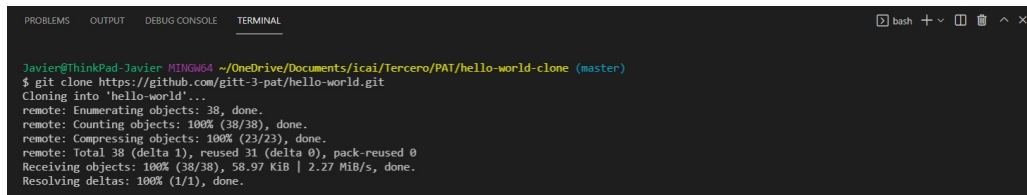


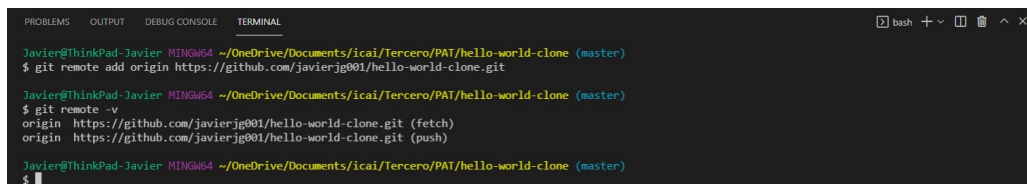
Figura 5: Repositorio copiado mediante fork

Más tarde se ha creado un clone desde el git bash para poder tener el repositorio en la máquina local como se ha explicado. Para ello, se tiene en la figura 6 el comando usado, ahora se tiene el repositorio de forma local, para poder subirlo a GitHub habría que usar los comandos descritos en la figura 7, el último de estos se usa para poder ver la información del GitHub y se comprueba que apunta hacia el repositorio creado “hello-world-clone”.



```
Javier@ThinkPad-Javier MINGW64 ~/OneDrive/Documents/ica/Tercero/PAT/hello-world-clone (master)
$ git clone https://github.com/gitt-3-pat/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
Receiving objects: 100% (38/38), 58.97 KiB | 2.27 MiB/s, done.
Resolving deltas: 100% (1/1), done.
```

Figura 6: Comando para clonar un repositorio



```
Javier@ThinkPad-Javier MINGW64 ~/OneDrive/Documents/ica/Tercero/PAT/hello-world-clone (master)
$ git remote add origin https://github.com/javierjg001/hello-world-clone.git

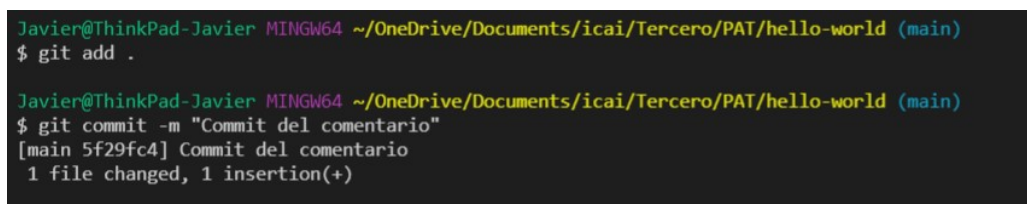
Javier@ThinkPad-Javier MINGW64 ~/OneDrive/Documents/ica/Tercero/PAT/hello-world-clone (master)
$ git remote -v
origin https://github.com/javierjg001/hello-world-clone.git (fetch)
origin https://github.com/javierjg001/hello-world-clone.git (push)

Javier@ThinkPad-Javier MINGW64 ~/OneDrive/Documents/ica/Tercero/PAT/hello-world-clone (master)
$
```

Figura 7: Comandos para subir el repositorio local a GitHub

2.2. Commits, status, log y regreso a otros commits

Cabe destacar que a partir de este momento, se usará y entregará el repositorio que ha sido creado haciendo fork (<https://github.com/javierjg001/hello-world>), y el otro creado con el clone (hello-world-clone) se dejará en la cuenta de “javierjg001” de GitHub. En este apartado se van a explicar los distintos comandos para crear y subir commits a GitHub. En cuanto a la hora de realizar commits, que son actualizaciones que se añaden al repositorio de GitHub, se tienen distintos comandos. Empezando por “git add .” (con el punto se añaden todos los ficheros, si solo se quiere añadir uno, habría que cambiar ese punto por el nombre del fichero correspondiente), con este comando se añaden dichos ficheros al “staging area” o área de preparación, que es el anterior área antes de ser añadidos definitivamente como un commit al proyecto final o al GitHub, esta área se usa para que el añadido a GitHub se realice en dos pasos por lo tanto puede ser más ordenado y escalado. Se ha añadido un comentario en el fichero App.java y se procederá a añadir un commit. Para poder añadir definitivamente el fichero se usa el comando git commit -m “Mensaje”(figura 8). Con el cual se añaden todos los elementos del staging area al repositorio como un commit con el mensaje correspondiente.



```
Javier@ThinkPad-Javier MINGW64 ~/OneDrive/Documents/ica/Tercero/PAT/hello-world (main)
$ git add .

Javier@ThinkPad-Javier MINGW64 ~/OneDrive/Documents/ica/Tercero/PAT/hello-world (main)
$ git commit -m "Commit del comentario"
[main 5f29fc4] Commit del comentario
1 file changed, 1 insertion(+)
```

Figura 8: Comando git add . y git commit -m

Ahora si se quiere subir al GitHub se ha de usar el comando git push, con el cual se actualiza el repositorio con los cambios hechos, como se ve en la figura 9.

```

Javier@ThinkPad-Javier MINGW64 ~/OneDrive/Documents/icai/Tercero/PAT/hello-world (main)
$ git push
Enumerating objects: 17, done.
Counting objects: 100% (17/17), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (9/9), 614 bytes | 307.00 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/javierjg001/hello-world.git
48fe276..5f29fc4 main -> main

```

Figura 9: Comando git push

Posteriormente, con el comando “git pull” se pueden descargar todos los cambios hechos en el repositorio a nuestra máquina local, como se tienen en la figura 10, aunque no se actualizará nada ya que todo está al día.

```

Javier@ThinkPad-Javier MINGW64 ~/OneDrive/Documents/icai/Tercero/PAT/hello-world (main)
$ git pull
Already up to date.

```

Figura 10: Comando git pull

Con el comando “git status” se obtiene información de cómo está actualmente el repositorio, es decir, si se tienen untracked files (ficheros que no se manejan sus versiones) o ficheros en staging area (añadidos pero aún no se ha realizado el commit). En la figura 11 se ve reflejado lo explicado.

```

Javier@ThinkPad-Javier MINGW64 ~/OneDrive/Documents/icai/Tercero/PAT/hello-world (main)
$ 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

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

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

Javier@ThinkPad-Javier MINGW64 ~/OneDrive/Documents/icai/Tercero/PAT/hello-world (main)
$ █

```

Figura 11: Comando git status

Se tiene el comando “git log” para ver los commits hechos con anterioridad, tiene varias opciones según lo que se necesite. Si no se añade nada, muestra dichos commits ordenados cronológicamente y en detalle, si se añade –oneline en vez de verse en detalle solo se observaría los datos principales, si se añade –graph se puede visualizar con las distintas ramas en las que han sido subidos estos commits, otro ejemplo puede ser si se añade -n 1, en este caso se verá solo el último commit (si se desean ver los dos últimos habría que cambiar el número a 2 y así según lo que se quiera ver). Un ejemplo de este comando se tiene en la figura 12. En esta misma figura se tiene también que uno de los commits es una mezcla de la rama “feature/1” a la rama principal, se explicará en mayor detalle las ramas posteriormente.

```

Javier@ThinkPad-Javier MINGW64 ~/OneDrive/Documents/icai/Tercero/PAT/hello-world (main)
$ git log --graph
* commit 5f29fc479d2c43288ed8cf2e9b7adc5277a61a51 (HEAD -> main, origin/main, origin/HEAD)
  Author: javierjg001 <56931870+javierjg001@users.noreply.github.com>
  Date: Sat Jan 29 20:58:51 2022 +0100

  Commit del comentario

* commit 48fe2767a1dec83ae423cca0c0f8c92204ba6f65
  \ Merge: 5038239 5b68377
  | Author: Juan Antonio Breña Moral <bren@juanantonio.info>
  | Date: Mon Dec 20 19:26:31 2021 +0100
  |
  | Merge pull request #1 from gitt-3-pat/feature/1
  |
  | Primera iteracion
  |
  * commit 5b683771defff11ef7f0086ef25c48a3cb95deed
  / Author: Juan Antonio Breña Moral <bren@juanantonio.info>
  | Date: Mon Dec 20 19:25:31 2021 +0100
  |
  | Primera iteracion
  |
  * commit 5038239e93d2da05a94667e8f9e66a16b5353076
  | Author: Juan Antonio Breña Moral <bren@juanantonio.info>

```

Figura 12: Comando git log

Puede llegar el caso de querer volver a commits anteriores o querer eliminar dichos, para ello se tiene el comando “git checkout” (que más adelante se verá como también este comando puede crear nuevas ramas y moverse por ellas). Para poder volver a versiones anteriores del programa se tendrá que saber el hashcode del commit al que se quiere volver. Se puede obtener dicho hash en la descripción del comando log, por ejemplo, en la figura 12 el hash del commit “Primera iteracion” es “5b68377”. La sintáxis del comando por lo tanto será “git checkout hashcode” como se tiene en la figura 13, que lo que se ha realizado es una vuelta a dicho commit (Primera iteracion) como se observa mediante el log y se deshace la vuelta por lo tanto se termina en el mismo punto con el último commit subido (Commit del comentario).

```

Javier@ThinkPad-Javier MINGW64 ~/OneDrive/Documents/icai/Tercero/PAT/hello-world (main)
$ git checkout 5b68377
Note: switching to '5b68377'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

    git switch -c <new-branch-name>

Or undo this operation with:

    git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at 5b68377 Primera iteracion

Javier@ThinkPad-Javier MINGW64 ~/OneDrive/Documents/icai/Tercero/PAT/hello-world ((5b68377...))
$ git log --graph --oneline
* 5b68377 (HEAD) Primera iteracion
* 5038239 Initial commit

Javier@ThinkPad-Javier MINGW64 ~/OneDrive/Documents/icai/Tercero/PAT/hello-world ((5b68377...))
$ git switch -
Previous HEAD position was 5b68377 Primera iteracion
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

Javier@ThinkPad-Javier MINGW64 ~/OneDrive/Documents/icai/Tercero/PAT/hello-world (main)
$ git log --graph --oneline
* 5f29fc4 (HEAD -> main, origin/main, origin/HEAD) Commit del comentario
* 48fe276 Merge pull request #1 from gitt-3-pat/feature/1
  \
  | * 5b68377 Primera iteracion
  /
  * 5038239 Initial commit

```

Figura 13: Comando git checkout

2.3. Pull requests

Ya que el repositorio con el que se está trabajando ha sido creado con un fork, los cambios que se han realizado pueden ser sugeridos al propietario del repositorio original para que se actualice. (En este caso solo se ha añadido un comentario por simplificación). Para ello se realiza un pull request desde la rama main hasta la principal del repositorio original desde la página web de GitHub como se ve en la figura 14. Una vez creado, al usuario gitt-3-pat le aparecerá dicho pull request y podrá o aceptarlo o rechazarlo.

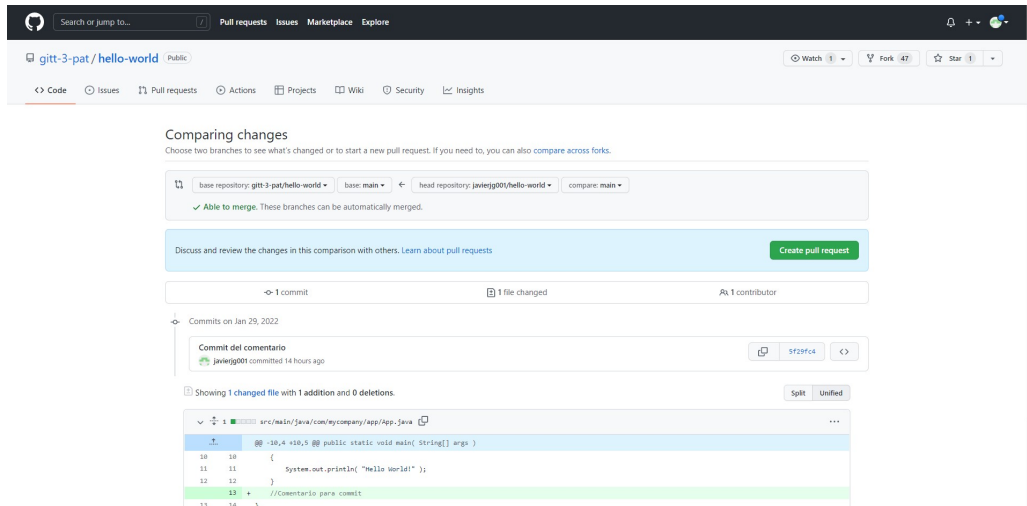


Figura 14: Creación del pull request

2.4. Crear y mezclar ramas

En este apartado se va a explicar la creación de las distintas ramas y cómo poder mezclarlas. Las ramas en git son los distintos “caminos” que se pueden crear del repositorio para poder trabajar en paralelo con la rama principal (que normalmente se trata de main) y más adelante, cuando se considera que se ha terminado esa rama mezclarla con la principal. Esto se usa en los grandes grupos para que no haya pérdidas a la hora de subir o bajar la misma versión del código, por lo tanto, dos programadores que trabajen en el mismo programa se podrán crear cada uno una rama y trabajar desde esta a la vez y en paralelo y una vez terminado su trabajo añadir los cambios al main y se da por concluida la rama.

Para poder crear ramas se tienen dos posibles comandos, el primero ya se ha visto y es git checkout, pero en este caso se le tiene que añadir -b nombre-rama, por lo tanto sería “git checkout -b nombre-rama”. Otro comando es “git branch nombre-rama”, la diferencia reside en que en el primero el usuario se mueve a esa rama y en el segundo comando se queda en la rama desde donde se ha creado. Para ver las distintas ramas que se tienen y en dónde se encuentra el usuario se puede usar “git branch”. Y para poder moverse entre ramas se tiene el comando “git checkout nombre-rama”. Si se desea eliminar la rama sin realizar merge (volcado a otra rama) se usará el comando “git branch -d nombre-rama”. Todo esto se ve reflejado en la figura 15. Cabe destacar que para crear la rama en GitHub se debe usar el comando “git push -u origin nombre-rama”.

```
Javier@ThinkPad-Javier MINGW64 ~/OneDrive/Documents/ica/Tercero/PAT/hello-world (main)
$ git checkout -b feature-backend
Switched to a new branch 'feature-backend'

Javier@ThinkPad-Javier MINGW64 ~/OneDrive/Documents/ica/Tercero/PAT/hello-world (feature-backend)
$ git branch
* feature-backend
  main

Javier@ThinkPad-Javier MINGW64 ~/OneDrive/Documents/ica/Tercero/PAT/hello-world (feature-backend)
$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

Javier@ThinkPad-Javier MINGW64 ~/OneDrive/Documents/ica/Tercero/PAT/hello-world (main)
$ git branch -d feature-backend
Deleted branch feature-backend (was 5f29fc4).

Javier@ThinkPad-Javier MINGW64 ~/OneDrive/Documents/ica/Tercero/PAT/hello-world (main)
$ git branch
* main
```

Figura 15: Comandos para trabajar con ramas

En cuanto al merge, mezcla de ramas, se puede realizar desde GitHub o desde local, en cuanto se hace desde la web se crea un commit para representar este merge mientras que si se realiza de forma local no se crea el commit, por lo tanto el historial de commits queda más limpio y ordenado. Se va a realizar un ejemplo de creación de rama y merge en local, para ello se debe de crear la rama como se ha visto anteriormente y crear el merge con el comando “git merge nombre-rama”, se subirá un cambio a la rama secundaria (otro comentario). En la figura 16 se tiene la creación de la rama y ya una vez añadido el comentario se añaden al staging area los archivos. En la figura 17 se crea el commit del comentario a la rama secundaria. Por último, en la figura 18 se tienen los comandos para cambiar de rama a main (ya que se debe de realizar el merge desde la rama destino), se crea el merge de la rama secundaria al main y se suben los cambios a git (no se verá reflejado en git la creación de la rama debido a que esto se ha hecho de forma local). Como se ha subido otro commit y se había creado un pull request en este se verá reflejado el segundo commit que se acaba de hacer también.

```
Javier@ThinkPad-Javier MINGW64 ~/OneDrive/Documents/icai/Tercero/PAT/hello-world (main)
$ git checkout -b feature-secundaria
Switched to a new branch 'feature-secundaria'

Javier@ThinkPad-Javier MINGW64 ~/OneDrive/Documents/icai/Tercero/PAT/hello-world (feature-secundaria)
$ git add .
```

Figura 16: Comandos para la creación de la rama y añadido al staging area

```
Javier@ThinkPad-Javier MINGW64 ~/OneDrive/Documents/icai/Tercero/PAT/hello-world (feature-secundaria)
$ git commit -m "Comentario subido a la rama secundaria"
[feature-secundaria 7a81791] Comentario subido a la rama secundaria
1 file changed, 1 insertion(+)
```

Figura 17: Comando para la creación del commit

```
Javier@ThinkPad-Javier MINGW64 ~/OneDrive/Documents/icai/Tercero/PAT/hello-world (feature-secundaria)
$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

Javier@ThinkPad-Javier MINGW64 ~/OneDrive/Documents/icai/Tercero/PAT/hello-world (main)
$ git merge feature-secundaria
Updating 5f29fc4..7a81791
Fast-forward
 src/main/java/com/mycompany/app/App.java | 1 +
 1 file changed, 1 insertion(+)

Javier@ThinkPad-Javier MINGW64 ~/OneDrive/Documents/icai/Tercero/PAT/hello-world (main)
$ git push
Enumerating objects: 17, done.
Counting objects: 100% (17/17), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (9/9), 629 bytes | 69.00 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/javierjg001/hello-world.git
 5f29fc4..7a81791 main -> main

Javier@ThinkPad-Javier MINGW64 ~/OneDrive/Documents/icai/Tercero/PAT/hello-world (main)
$
```

Figura 18: Comando para merge y subida del código a GitHub

Cabe destacar que cuando se trabaja con varias ramas pueden provocarse conflictos, en este caso solo se añadían comentarios por lo que no había ninguno de estos, pero en el momento que muchos programadores intentan arreglar el mismo error o trabajan sobre el mismo archivo pueden aparecer estos. Y para poder arreglarlos se tienen varias opciones, la primera es aceptar el cambio que estaba antes (accept current change). La segunda es aceptar el cambio nuevo que se introduce (accept incoming change). La tercera es aceptar los dos cambios (accept both changes) y la cuarta el comparar los cambios para poder observar de manera visual en qué difieren (compare changes).

3. Conclusión

En esta práctica se ha podido trabajar con las herramientas git y GitHub y por lo tanto, entender su funcionamiento que es algo básico ya que en el mundo laboral proyectado son ampliamente usadas en cualquier tipo de proyecto. También se considera que se han conseguido los objetivos propuestos al principio de este informe. Esta practica por lo tanto será la base de las futuras ya que en todas estas se usarán las herramientas con las que se han trabajado en esta.