



UNIVERSIDAD DE MÁLAGA

INGENIERÍA DEL SOFTWARE AVANZADA

Git (Individual)

Autor:

Mario Pascual González

Profesor asignado:

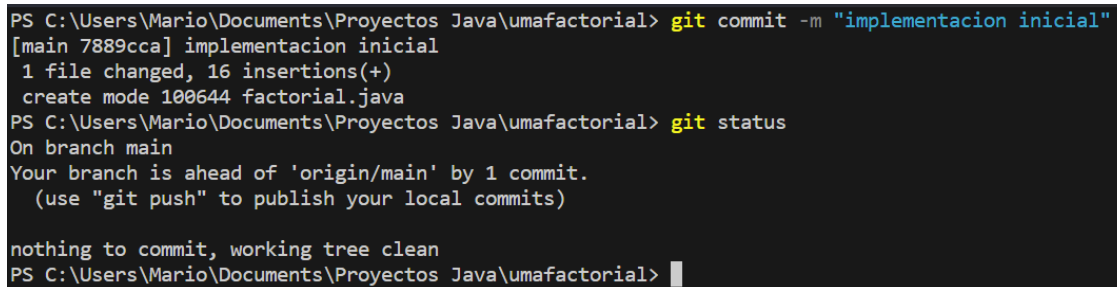
Dr. José Miguel Horcas Aguilera

Realización de la práctica

Primero se ha creado el repositorio 'umafactorial' en GitHub, habiendo creado un nuevo workspace en el IDE VSCode, se ha ejecutado con éxito el comando `git clone` (1). De manera inmediata se ha escrito el primer código de factorial java en el fichero 'factorial.java' (2).

Después de haber creado el código se ha validado que nos encontramos en la rama 'main', para, posteriormente, mover el fichero 'factorial.java' a la *staging area* y, finalmente, realizar un commit al repositorio git (3).

```
(-) git config user.name "MarioPasc"
(-) git config user.email "mario.pg02@gmail.com"
(1) git clone https://github.com/MarioPasc/umafactorial
(-) cd umafactorial
(-) git status
(3) git add factorial.java
(3) git commit -m "Initial commit"
(-) git status
```



```
PS C:\Users\Mario\Documents\Proyectos Java\umafactorial> git commit -m "implementacion inicial"
[main 7889cca] implementacion inicial
1 file changed, 16 insertions(+)
create mode 100644 factorial.java
PS C:\Users\Mario\Documents\Proyectos Java\umafactorial> 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
PS C:\Users\Mario\Documents\Proyectos Java\umafactorial> █
```

Figura 1: Primer commit y status del proyecto

Como se puede observar en la Figura 1, el commit ha sido realizado con éxito y nos indica que la rama que acabamos de editar (main) está una posición por delante del puntero HEAD del repositorio de github. Para publicar nuestros cambios al repositorio github en la web, deberíamos utilizar el comando `git push`.

A continuación se modifica el código para calcular el factorial en otra función y se procede a hacer un commit (4). El código es publicado en el repositorio público de github con éxito (5).

```
(-) git status
(4) git add factorial.java
(4) git commit -m "Refactorización"
(-) git status
(5) git push origin main
```

```

PS C:\Users\Mario\Documents\Proyectos Java\umafactorial> git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
(use "git push" to publish your local commits)

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:   factorial.java

no changes added to commit (use "git add" and/or "git commit -a")
PS C:\Users\Mario\Documents\Proyectos Java\umafactorial> git add factorial.java
PS C:\Users\Mario\Documents\Proyectos Java\umafactorial> git commit -m "refactorizacion"
[main f84236e] refactorizacion
 1 file changed, 10 insertions(+), 6 deletions(-)
PS C:\Users\Mario\Documents\Proyectos Java\umafactorial> git status
On branch main
Your branch is ahead of 'origin/main' by 2 commits.
(use "git push" to publish your local commits)

nothing to commit, working tree clean
PS C:\Users\Mario\Documents\Proyectos Java\umafactorial>

```

Figura 2: Segundo commit y status del proyecto. No se incluye *git push origin main* por la extensión del output

Como se puede observar en la Figura 2, el commit se ha realizado con éxito y ahora estamos dos pasos por delante. Al ejecutar `git push origin main`, podemos confirmar que ahora `git status` nos comunicará que nos encontramos a corde con el proyecto de la nube en github.

Procedemos ahora a crear la nueva rama 'recursivo' (5) e implementar el código (6) para posteriormente realizar un push con este al repositorio de github (7).

```

(5) git checkout -b recursivo
(-) git status # Comprobamos que estamos en la rama recursivo
(7) git add factorial.java
(7) git commit -m "Implementación recursiva"
(7) git push origin recursivo
(-) git status

```

Posteriormente se procede con los cambios propuestos en la rama main (8, 9) y se realiza un commit a la rama main (10).

```

(8) git checkout main
(-) git status
(10) git add factorial.java
(10) git commit -m "Correccion numeros negativos"
(10) git push origin main
(-) gitk --all

```

Ahora que el departamento de calidad ya ha aceptado el fichero original pero con un ligero cambio se volverá a recursivo, modificando el código ligeramente y haciendo un commit (11).

```

(11) git checkout recursivo
(-) git status
(11) git add factorial.java
(11) git commit -m "Optimizacion"
(12) git checkout main
(12) git merge recursivo
(-) gitk --all

```

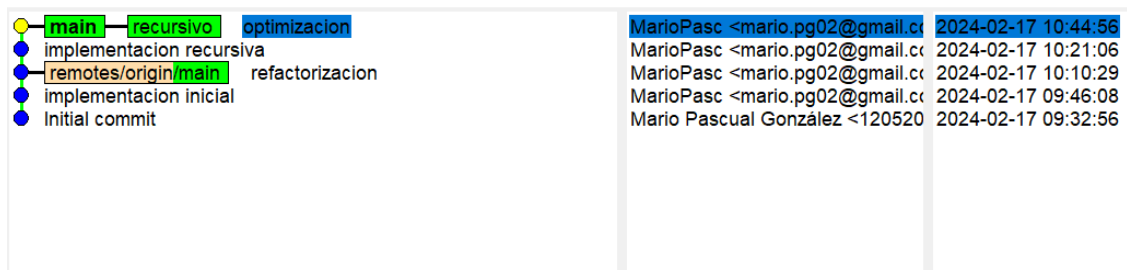


Figura 3: Comando gitk --all para el estado actual

Como podemos observar en la Figura 3, tenemos 2 ramas: 'main' y la rama remota 'origin/main'. La rama recursivo se ha fusionado con main, mostrando el último commit de recursivo a la cabeza de main (optimización). Por otra parte, el commit 'implementación recursiva', al haberse fusionado la rama recursivo con main no se ve asociado a ninguna rama.

Se crea la nueva rama refacIO, cambiando la entrada y la salida con un nuevo código refactorizado (13, 14). Posteriormente, debido al problema relacionado con los factoriales mayores de 20, se guardan los cambios temporalmente y se vuelve a la rama master. Finalmente, se vuelve a la rama refacIO, se recuperan los cambios, y se hace un commit (15, 16).

```

(13) git checkout -b refacIO
(-) git status
(14) git add factorial.java
(14) git commit -m "Refactorizacion de metodo entrada"
(15) git stash
(15) git checkout main
(16) git add factorial.java
(16) git commit -m "Cambio de long a biginteger"
(16) git push origin main
(16) git checkout refacIO
(16) git stash pop
(16) git add factorial.java
(16) git commit -m "Refactorizacion de metodo de salida"

```

Ahora nos proponemos a unir las ramas refacIO y main.

```

(17) git checkout main
(17) git merge refacIO

```

```
13 <<<<<< HEAD (Current Change)
14 public static BigInteger fact(BigInteger num) {
15     if (num.compareTo(BigInteger.ONE) <= 0) {
16         return BigInteger.ONE;
17     } else {
18         return num.multiply(fact(num.subtract(BigInteger.ONE)));
19     }
20 }
21 Run | Debug
22 public static void main(String []args) {
23     Scanner scanInput = new Scanner(System.in);
24     System.out.print("Introduzca un numero: ");
25     BigInteger num = BigInteger.valueOf(scanInput.nextLong());
26     if (num.compareTo(BigInteger.ZERO) < 0) {
27         System.out.println("El factorial no esta definido"
28             + " para numeros negativos");
29     } else {
30         BigInteger fac = fact(num);
31         System.out.println("El factorial de "+num+" es "+fac);
32     }
33 =====
34 public static long getNumber() {
35     Scanner scanInput = new Scanner(System.in);
36     System.out.print(s:"Introduzca un numero: ");
37     long num = scanInput.nextLong();
>>>>>> refacIO (Incoming Change)
```

Figura 4: Captura del código java de VSCode con las sugerencias de arreglos en el código para la unión de las ramas main y refacIO

Como podemos observar en la Figura 4, debemos arreglar los conflictos para poder proceder (18). Los cambios se han arreglado, por lo que se procede a hacer un commit y push a la rama origen (19).

- (19) git add factorial.java
- (19) git commit -m "Mezcla de la rama refacIO"
- (19) git push origin main

Finalmente se va a crear un documento de texto (.txt) con todos los comandos git que se han utilizado con la finalidad de instruir a nuevos empleados en el uso de Git. Se crea el fichero y se usa un commit para subirlo al repositorio (20, 21). Debido a que ya se ha redactado este documento explicando los pasos y resultados de la práctica, este fichero de texto solo contendrá los comandos de git, mientras que **este documento pdf \LaTeX servirá como guía para que se entienda para qué sirven estos comandos usados y por qué se han usado.**

- (21) git add comandos-git.txt
- (21) git commit -m "Adición de hisotiral de linea de comandos"

Finalmente se sincronizarán todas las ramas del repositorio local con las del repositorio github.

- (22) git push --all origin

Finalmente se ejecutó el comando `gitk --all` para comparar el grafo propuesto en el documento de la práctica y mi grafo resultado.

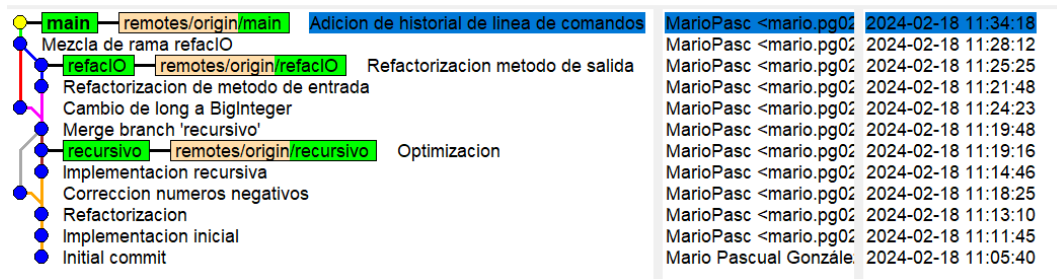


Figura 5: Captura del comando `gitk --all`

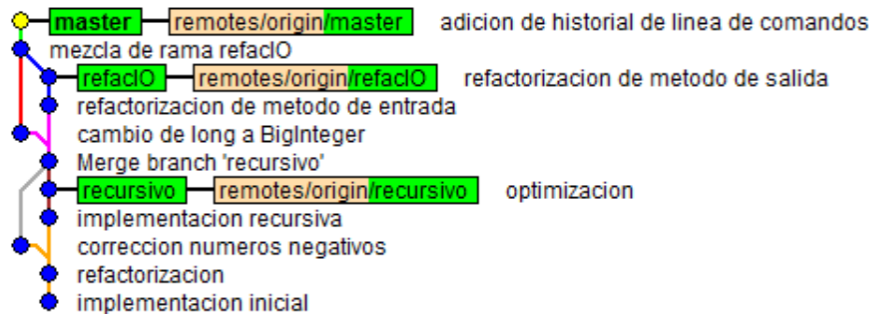


Figura 6: Captura del grafo propuesto como resultado

Como podemos observar al comparar las imágenes 5 y 6, los pasos de la práctica han sido realizados con éxito, consiguiendo la misma estructura que la propuesta por el Dr. José Miguel Horcas Aguilera al final del documento.

Se puede observar aún así un paso adicional en mi implementación 'Initial commit', sin embargo, no parece que se deba dar demasiada importancia, ya que no afecta a la estructura de las ramas.

Toda esta práctica se ha apoyado en las referencias online [1], [2]. El repositorio creado para esta práctica puede encontrarse en mi repositorio de GitHub.

NOTA: El código y todo su contenido asociado presentado en este documento son mi propio trabajo y han sido creados con el propósito de cumplir con una tarea académica. Este trabajo es el resultado de mi propia comprensión y esfuerzos en el contexto de los requisitos de la tarea.

Mario PASCUAL GONZÁLEZ

20 de febrero de 2024

Referencias

- [1] Git, “Git - documentation,” 2024, Último acceso: 2024-02-18. [Online]. Available: <https://git-scm.com/docs>
- [2] S. Chacon and B. Straub, *Pro Git*, 2nd ed. Apress, 2014, Último acceso: 2024-02-18. [Online]. Available: <https://git-scm.com/book/en/v2>