Práctica 1 - Basics

¿Qué es Git?

Git es un sistema de control de versiones. ¿Qué es un sistema de control de versiones? De una manera simplificada, Git, es un programa que va guardando diferentes versiones de un proyecto. Esto es como si en un documento Word, cada vez que le diésemos a guardar, se guardase la nueva versión, pero también una copia de la antigua y los cambios que hay entre la nueva y la antigua. Esto es muy útil pues permite revertir a versiones anteriores si hay errores, si hay cambios indeseados o si se quiere recuperar partes del proyecto que han sido eliminadas.

También se puede ver quién ha hecho qué cambios y permite trabajar en paralelo sobre distintas versiones de un proyecto. Esta funcionalidad es muy útil. Imaginemos que tenemos una aplicación web que está en funcionamiento y a la que queremos introducir algunos cambios. Como es el jefe, delega esta tarea a un becario. Una opción sería trabajar sobre la versión del proyecto que está desplegada (activa) en ese momento pero, ¿y si el becario se equivoca e introduce errores? Para evitar este riesgo se crea una copia del proyecto sobre la que se puede trabajar de forma independiente (*checkout*). El becario puede trabajar sobre esta copia (o rama) sin interferir en la versión desplegada. Una vez hace los cambios que le ha pedido los guarda, que en Git se llama hacer un *commit*. Una vez ha terminado lo que le ha pedido, puede unir sus cambios a la versión desplegada.

¿Qué es GitHub?

GitHub es un OneDrive/Google Drive para código y que, internamente, utiliza Git.

Comandos de GIT

El primer paso en todo proyecto es crear un repositorio. Un repositorio (o *repo*, si quiere estar a la moda) es dónde se guardan todos los archivos de un proyecto. Una vez creado el proyecto, habrá que añadir código para darle funcionalidad. Para poder trabajar en el proyecto tengo que descargármelo en el ordenador. La forma de descargarme el proyecto es clonándolo. La diferencia entre descargárselo directamente y clonarlo es que, al clonarlo, Git irá registrando los cambios que voy haciendo al proyecto.

En esta primera foto se muestran los contenidos de la carpeta “pat”. Como se puede ver, la carpeta está vacía.

Ahora clono el repositorio “hello-world” en la carpeta “pat” con el siguiente comando

```
git clone https://github.com/JGCARDELUS-GITT-PAT/hello-world.git
```

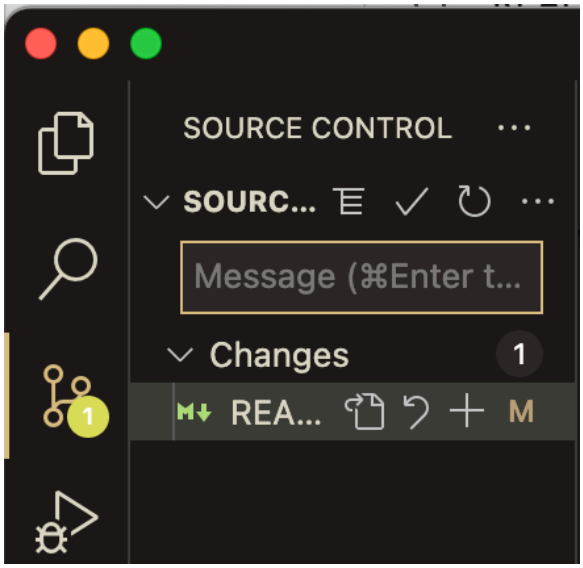
```
jgcardelus@jorges-macbook-air pat % git clone https://github.com/JGCARDELUS-GITT-PAT/hello-world.git
Cloning into 'hello-world'...
remote: Enumerating objects: 38, done.
remote: Counting objects: 100% (38/38), done.
remote: Compressing objects: 100% (23/23), done.
remote: Total 38 (delta 1), reused 31 (delta 0), pack-reused 0
Receiving objects: 100% (38/38), 58.97 KiB | 1.09 MiB/s, done.
Resolving deltas: 100% (1/1), done.
jgcardelus@jorges-macbook-air pat % ls
hello-world
jgcardelus@jorges-macbook-air pat %
```

Y ya tengo el proyecto descargado en el ordenador.

Ahora voy a modificar el proyecto para ver qué ocurre. Para eso abro el proyecto, por ejemplo, en VSCode (también conocido como el mejor editor de texto).

```
You, seconds ago • Uncommitted changes
## Mis cambios
Este es un cambio de prueba
```

Los cambios que hago, son añadir estas líneas en el archivo README.md. VSCode, como es un editor muy potente ya me avisa de que hay cambios sin guardar en el proyecto.



En esta imagen se puede ver cómo VSCode me avisa de que he modificado README.md y en el icono de la izquierda muestra que hay un cambio sin guardar.

Pero pongamos que somos el típico programador que usa VIM en una distro basada en Arch Linux al que, obviamente, le gusta sufrir. ¿Cómo comprobamos esto a mano? Podemos llegar a esta misma información en la terminal con

```
git status
```

```
jgcardelus@jorges-macbook-air hello-world % 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:   README.md

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

En la foto superior se puede ver la salida de ese comando. Como se puede ver, Git nos indica en rojo, que se ha modificado README.md. También se puede ver cómo nos incita a ejecutar git add o git restore. ¿Qué son estos comandos? Como he mencionado al principio, para guardar un cambio en el repositorio se hace un *commit* pero, antes de hacer el commit, hay que indicarle a Git qué cambios queremos que se guarden en ese commit. 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”.

```
git stage .
```

```
jgcardelus@jorges-macbook-air hello-world % git add .
jgcardelus@jorges-macbook-air 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)
        modified:   README.md

jgcardelus@jorges-macbook-air hello-world %
```

Ahora se puede ver cómo los cambios están en verde, indicando que están listos para ser guardados.

Pongamos ahora, que antes de hacer el commit, alguien ve las líneas que he escrito en README.md, no le gustan y me pide que deje todo como estaba. ¿Cómo se hace? Lo primero que hay que hacer es quitar de “staged” a README.md. Para ello se ejecuta el siguiente comando

```
git restore --staged README.md
```

Al ver el estado de los archivos, podemos ver que README.md vuelve a estar en rojo, indicando que no está “staged”.

```
jgcardelus@jorges-macbook-air hello-world % 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:   README.md

no changes added to commit (use "git add" and/or "git commit -a")
jgcardelus@jorges-macbook-air hello-world %
```

Seguimos, todavía, necesitando revertir los cambios que hemos hecho. Para eso, se ejecuta el comando

```
git restore README.md
```

Que revierte los cambios hechos y deja el archivo como estaba al inicio del commit.

```
jgcardelus@jorges-macbook-air hello-world % git restore README.md
jgcardelus@jorges-macbook-air hello-world % cat README.md
# hello-world

Un repositorio para empezar a usar Github y Github Actions con Java

## ¿Como probar en el cloud?

[![](https://gitpod.io/button/open-in-gitpod.svg)](https://gitpod.io/#https://github.com/gitt-3-pat/hello-world)

## Comandos git básicos

```
git clone https://github.com/gitt-3-pat/hello-world
git status
git add .
git commit -m "TU MENSAJE"
git push

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

Referencias

- https://git-scm.com/
jgcardelus@jorges-macbook-air hello-world %
```

Si examina el contenido de README.md tras ejecutar git restore, se puede ver que ya no están mis líneas de prueba. Para los siguientes ejemplos sí me interesa que haya cambios, así que voy a volver a modificar README.md y añadirlo al commit.

Finalmente llegamos al commit. Un commit, como he dicho antes 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

```
git commit -m "Qué has hecho en este commit?"
```

```
jgcardelus@jorges-macbook-air hello-world % git commit -m "Modified README.md with awesome content"
[main b62e8c2] Modified README.md with awesome content
 1 file changed, 3 insertions(+)
jgcardelus@jorges-macbook-air hello-world %
```

Al ejecutarlo, Git me dice que ha creado un commit (el cual tiene un id asociado). Ese commit modifica un archivo (README.md) del proyecto inicial añadiéndole 3 líneas. También le he puesto un mensaje al commit para saber en un futuro qué era lo que modificaba (o añadía) ese commit.

¿Eso es todo para guardar los cambios? No. Todavía me queda actualizar los cambios el repositorio original. Hasta ahora estos commits están guardados en mi copia local pero yo quiero que se actualicen en GitHub, ¿cómo lo hago? Para eso tengo que ejecutar

```
git push
```

Al ejecutarlo se envían los cambios y se actualiza el repositorio original.

```
jgcardelus@jorges-macbook-air hello-world % git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 350 bytes | 350.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/JGCARDELUS-GITT-PAT/hello-world.git
   48fe276..b62e8c2  main -> main
jgcardelus@jorges-macbook-air hello-world %
```

Como se puede ver en la salida del comando git push, el repositorio original ha pasado del commit 48fe276 al b62e8c2, que si recuerda, es el ID del commit que hemos hecho antes. Con este mensaje, nos informan de que los cambios se han actualizado en el repositorio original.

En la introducción menciono que se pueden trabajar sobre versiones distintas del mismo proyecto en paralelo. ¿Cómo se hace? El primer paso es crear una rama (*branch*). 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.

```
git checkout -b info
```

```
jgcardelus@jorges-macbook-air hello-world % git checkout -b info
Switched to a new branch 'info'
```

Mi rama la voy a llamar info, indicando el propósito de esa rama. Tras ejecutar ese comando, crea la rama y me permite trabajar sobre ella. En esta rama voy a crear el documento info.md.

```
jgcardelus@jorges-macbook-air hello-world % touch info.md
jgcardelus@jorges-macbook-air hello-world % nano info.md
jgcardelus@jorges-macbook-air hello-world % cat info.md
This is an info documento with absolutely no info.
```

Creado.

git status me muestra que he realizado un cambio en mi rama “info”.

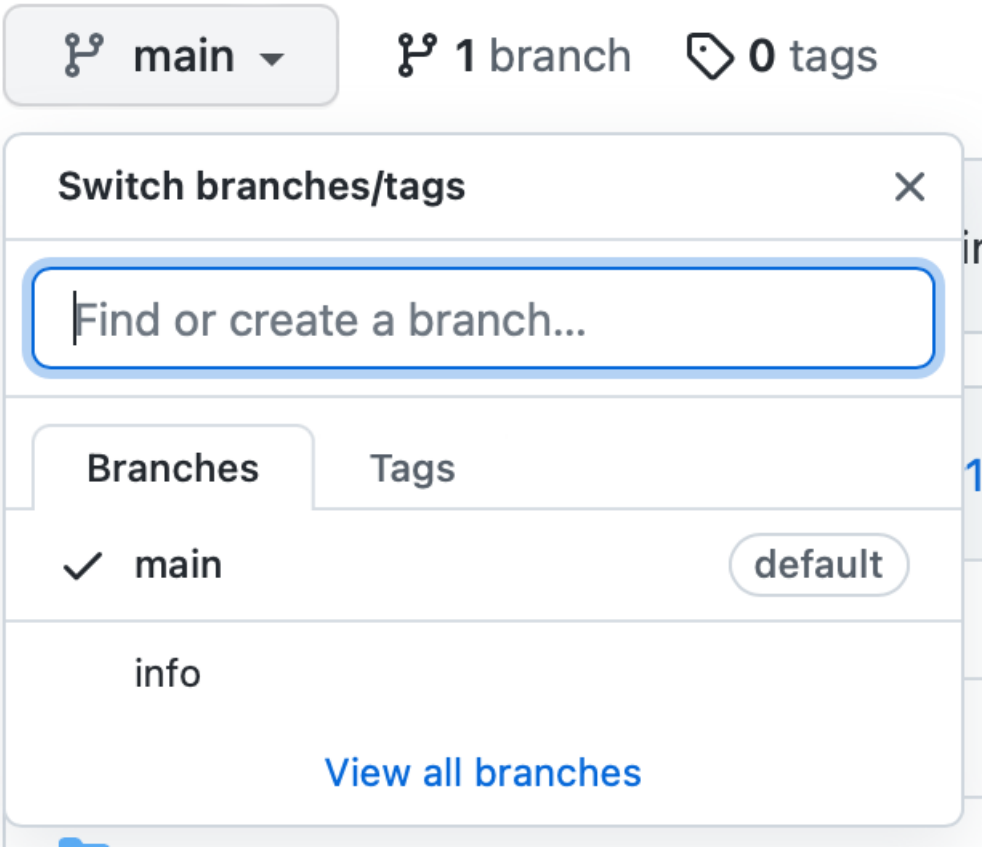
```
jgcardelus@jorges-macbook-air hello-world % git status
On branch info
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        info.md
```

¿Cómo subo ahora esta rama al repositorio original? Igual que antes, hay que añadir los cambios al “stage”, hacer un commit y realizar un push. La diferencia es que el push tiene que ser a la rama que quieres crear. Por tanto, los comandos son

```
git add info.md
git commit -m "Very interesting concept of an info doc"
git push origin info
```

```
jgcardelus@jorges-macbook-air hello-world % git add info.md
jgcardelus@jorges-macbook-air hello-world % git commit -m "Very interesting concept of an info doc"
[info 44dddc1] Very interesting concept of an info doc
 1 file changed, 1 insertion(+)
 create mode 100644 info.md
jgcardelus@jorges-macbook-air hello-world % git push origin info
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), 343 bytes | 343.00 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.
remote:
remote: Create a pull request for 'info' on GitHub by visiting:
remote:      https://github.com/JGCARDELUS-GITT-PAT/hello-world/pull/new/info
remote:
To https://github.com/JGCARDELUS-GITT-PAT/hello-world.git
 * [new branch]      info -> info
```

La salida nos muestra que se han guardado los cambios y que se ha creado una nueva rama. Si vamos a GitHub, podemos ver cómo hay dos ramas ahora.



Una vez terminado mi documento de info 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 info, tengo que pasar primero a main.

```
git checkout main
```

Ahora que estoy en main, puedo fusionar las dos ramas con git merge

```
git merge info
```

```
jgcardelus@jorges-macbook-air hello-world % git merge info
Updating b62e8c2..44dddc1
Fast-forward
 info.md | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 info.md
```

Acabo de fusionar ambas ramas en mi copia local así que, para terminar, actualizo los cambios.

```
git push
```

En GitHub puedo ver cómo la rama “main” tiene ahora el documento info.md.

main | 2 branches | 0 tags | Go to file | Add file | Code

This branch is 2 commits ahead of gitt-3-pat:main. Contribute | Fetch upstream

JGCardelus Very interesting concept of an info doc 44dddc1 7 minutes ago 5 commits

.github/workflows	Primera iteracion	last month
.mvn/wrapper	Primera iteracion	last month
src	Primera iteracion	last month
.editorconfig	Primera iteracion	last month
.gitignore	Primera iteracion	last month
.gitpod.Dockerfile	Primera iteracion	last month
.gitpod.yml	Primera iteracion	last month
.sdkmanrc	Primera iteracion	last month
LICENSE	Initial commit	last month
README.md	Modified README.md with awesome content	40 minutes ago
info.md	Very interesting concept of an info doc	7 minutes ago
mvnw	Primera iteracion	last month
mvnw.cmd	Primera iteracion	last month
pom.xml	Primera iteracion	last month

Falta un problema por resolver, la rama “info” sigue existiendo, ¿cómo se elimina?

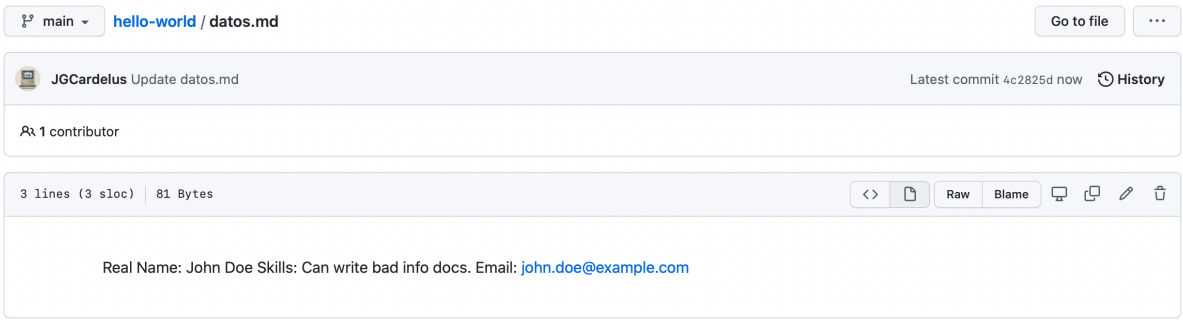
```
git branch -d info
git push origin --delete info
```

El primer comando elimina la rama local, el segundo la remota.

main | 1 branch | 0 tags | Go to file | Add file | Code

¿Y qué ocurre si alguien modifica el repositorio original y yo quiero actualizar mi repositorio local con esos cambios?

Si alguien hace cambios en el repositorio original (o usted mismo desde otro sistema) puede actualizar la copia local con git pull. Por ejemplo, voy a crear un documento desde GitHub. En este documento pondré mis datos para que, cuando Google quiera inevitablemente contratarme, pueda localizarme.




```
jgcardelus@jorges-macbook-air hello-world % ls
LICENSE         info.md         mvnw.cmd        src
README.md       mvnw            pom.xml         target
jgcardelus@jorges-macbook-air hello-world %
```

Como se puede ver en GitHub está el documento datos, pero no en mi ordenador.

Si ejecuto git pull

```
git pull
```

```
jgcardelus@jorges-macbook-air hello-world % git pull
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 743 bytes | 185.00 KiB/s, done.
From https://github.com/JGCARDELUS-GITT-PAT/hello-world
   44dddc1..bcce3bd  main       -> origin/main
Updating 44dddc1..bcce3bd
Fast-forward
 datos.md | 3 +++
 1 file changed, 3 insertions(+)
 create mode 100644 datos.md
jgcardelus@jorges-macbook-air hello-world %
```

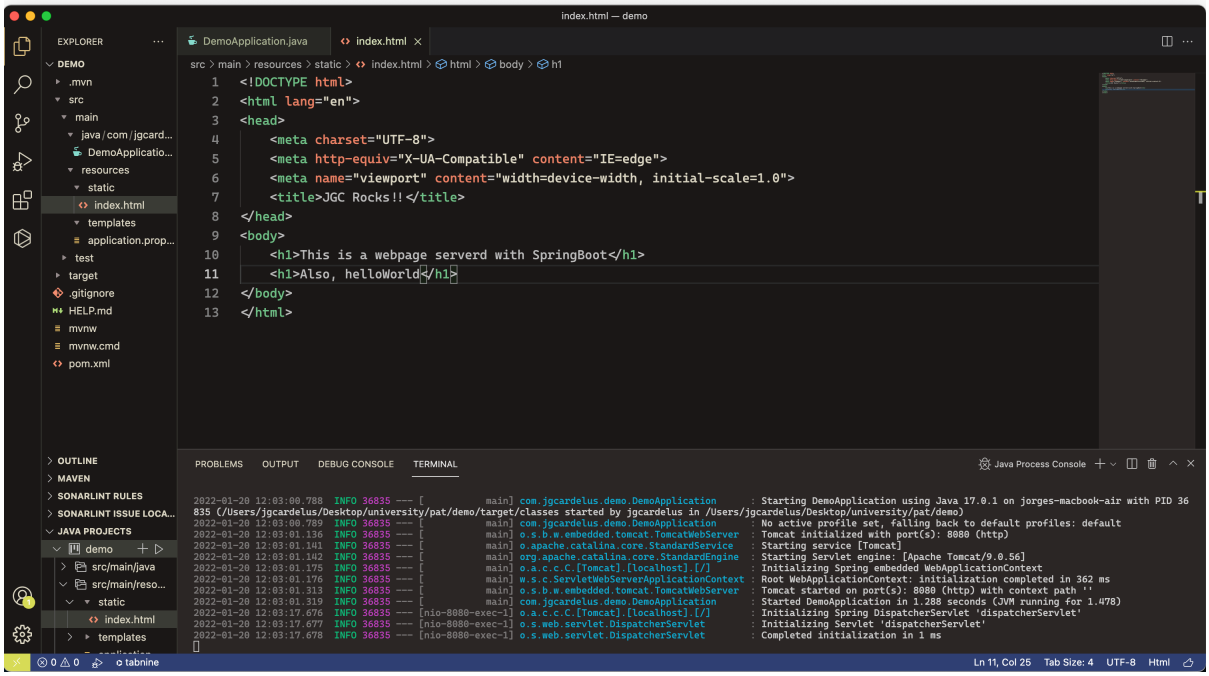
```
jgcardelus@jorges-macbook-air hello-world % ls
LICENSE         datos.md        mvnw            pom.xml         target
README.md       info.md         mvnw.cmd        src
jgcardelus@jorges-macbook-air hello-world %
```

El archivo ya está disponible. ¿Es magia? No, es Linus Torvalds que tiene mucho tiempo libre.

Normalmente, no se usan los comandos directamente desde la terminal sino que los IDEs vienen con esta funcionalidad integrada. Todo esto lo podría haber hecho desde VSCode de una manera mucho más sencilla e intuitiva; lo que no significa que no sea útil conocerse los comandos. Es especialmente útil para actualizar repositorios en servidores remotos, donde normalmente todo se hace a través de una terminal.

Prueba de Java 17, Maven y VSCode

```
jgcardelus@jorges-macbook-air hello-world % java --version
java 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)
jgcardelus@jorges-macbook-air hello-world %
```

This is a webpage serverd with SpringBoot

Also, helloWorld