Git y su implementación en IntelliJ

Jorge Alcides Diaz Lozano

Universidad de Cundinamarca Extensión Chía

Programación II

William Alexander Matallana Porras

19 de febrero de 2025

Tabla de contenido.

Objetivo

Proporcionar información para la implementación y uso de Git y GitHub dentro de IntelliJ IDEA, proporcionando un paso a paso detallado sobre la instalación, configuración y ejecución de los comandos primitivos de Git. Este documento permitirá la posibilidad de colaborar en proyectos y optimizar el flujo de trabajo dentro del entorno de desarrollo de IntelliJ.

Cómo crear un proyecto en IntelliJ

En primer lugar, se debe de descargar la versión de IntelliJ, teniendo también la posibilidad de obtener su forma gratuita de “IntelliJ Community”. La cual permitirá comenzar con su configuración posterior a su descarga.

En segundo lugar, se deben de implementar tanto un JDK 21 LTS, como la última versión de Git. Estos programas contribuirán en el desarrollo de diversas funciones tales como poder compartir código, administrar versiones y ejecutar programas en Java de manera eficiente.

Una vez instalados, se procede a configurar el entorno en IntelliJ. Para ello, al abrir IntelliJ por primera vez, se selecciona la opción "New Project". A continuación, se elige "Java" como lenguaje de programación y se verifica que el JDK 21 LTS esté correctamente asignado en la configuración del proyecto.

Posteriormente, se define la ubicación del proyecto y su estructura. IntelliJ permite configurar el proyecto con diferentes opciones.

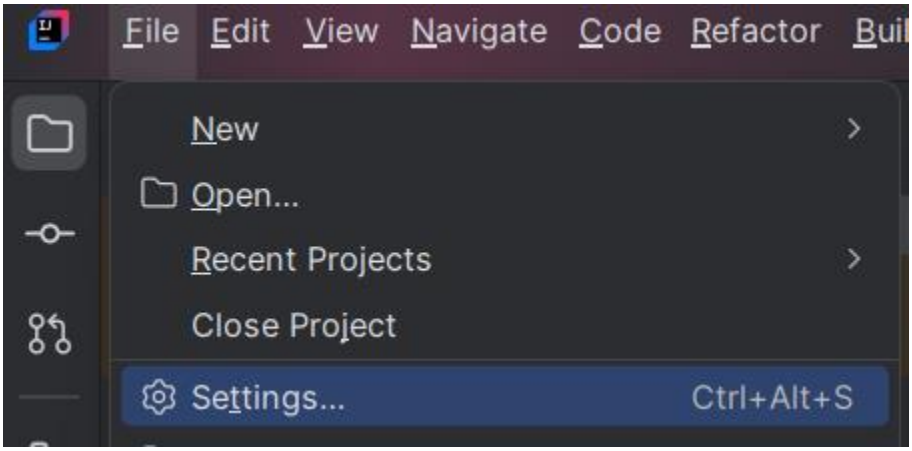
Finalmente, tras la creación del proyecto, se recomienda realizar un commit inicial en Git para establecer un punto de partida en el control de versiones. Desde la interfaz de IntelliJ, es

posible integrar un repositorio de GitHub, lo que facilita la colaboración y el seguimiento del código.

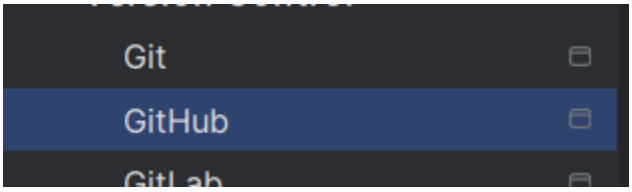
Con estos pasos, el entorno de desarrollo estará listo para programar en Java de manera eficiente.

Vincular GitHub con IntelliJ

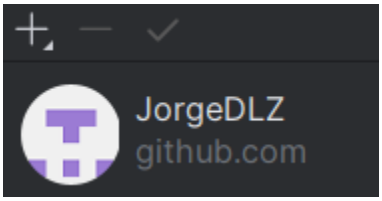
Una vez que se tengan los programas listos, se debe de vincular GitHub con IntelliJ para así poder acceder a todas las funcionalidades que ofrece GitHub. Para ello, se debe de buscar el apartado de configuraciones, como se muestra a continuación.



En este apartado se deberá de buscar la opción “GitHub”, la cual una vez se haya encontrado se deberá de seguir el proceso que se solicita para vincular la cuenta de GitHub.



Una vez se haya completado el proceso de loggeo deberá de aparecer la cuenta correspondiente en este mismo apartado.



Por otro lado, se deberá de ingresar a la terminal de IntelliJ para ejecutar los diferentes comandos, los cuales permitirán ejercer diferentes funciones para el procesamiento de los codigos y sus respectivos repositorios de GitHub.

Antes de empezar se debe de vincular tanto el nombre de usuario como el correo electrónico proviniente de GitHub, para esto usaremos el siguiente comando.

Git config –list

El comando git config --list muestra todas las configuraciones de Git que están activas en el sistema. Se usa para verificar los valores actuales de configuración, como el nombre de usuario, correo electrónico, el editor predeterminado, las credenciales almacenadas y otras opciones personalizadas.

Una vez se haya ejecutado el comando en el panel de control introduciremos el siguiente comando para modificar nuestros datos.

```
git config user.name "Tu Nuevo Nombre"
git config user.email "tu.nuevo.correo@example.com"
```

git config --global user.name "user"

Configurar Nombre que salen en los commits.

```
C:\Users\Asus M3604Y>git config --global user.name
JorgeDLZ
```

git config --global user.email "email"

Establece la dirección de correo electrónico que Git utilizará globalmente para los commits

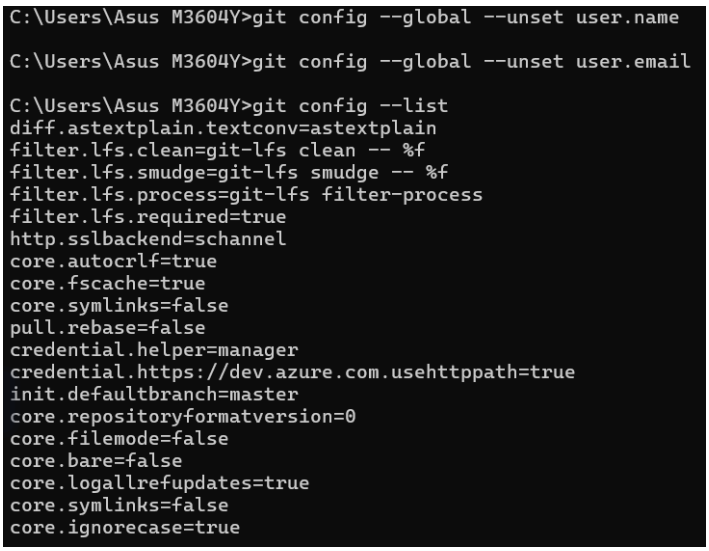
```
C:\Users\Asus M3604Y> git config --global user.email
jorgediazlozano11@gmail.com
```

git config --global --unset user.email

Elimina la configuración global del correo electrónico en Git. Esto significa que Git ya no usará un correo predeterminado para los commits hasta que se configure uno nuevo.

git config --global --unset user.name

Elimina la configuración global del nombre de usuario en Git. Después de ejecutarlo, Git no mostrará un nombre global en los commits hasta que se vuelva a configurar.



Una vez se ha implementado con éxito los datos previos, se puede continuar con el proceso de subir los proyectos al repositorio.

Comandos Git

1. git status

Muestra el estado actual del repositorio, indicando qué archivos han sido modificados, cuáles están en el área de preparación y cuáles no están bajo control de Git. Ayuda a verificar qué cambios se han hecho antes de confirmarlos (commit) o enviarlos al repositorio remoto.

```
On branch main
Your branch is up to date with 'origin/main'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore
        .idea/
        ComandosGit.iml
        src/
```

2. git add .

Agrega todos los archivos modificados y nuevos al área de preparación para ser incluidos en el próximo commit. Es útil cuando se han realizado múltiples cambios y se quieren registrar todos en un solo commit.

```
git add .
warning: in the working copy of '.gitignore', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'src/Main.java', LF will be replaced by CRLF the next time Git touches it
```

3. git commit -m "mensaje"

Crea un commit con los cambios añadidos al área de preparación, incluyendo un mensaje descriptivo sobre lo que se ha modificado. Permite registrar los cambios en el historial del repositorio de manera organizada.

```
git commit -m "ComandosGit"
[main baa7023] ComandosGit
 7 files changed, 78 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 .idea/.gitignore
 create mode 100644 .idea/misc.xml
 create mode 100644 .idea/modules.xml
 create mode 100644 .idea/vcs.xml
 create mode 100644 ComandosGit.iml
 create mode 100644 src/Main.java
```

4. git push origin main

Envía los commits locales al repositorio remoto en la rama main. Es necesario para que los cambios realizados en la máquina local estén disponibles en un servidor remoto, permitiendo la colaboración con otros desarrolladores.

```
git push origin main
Enumerating objects: 12, done.
Counting objects: 100% (12/12), done.
Delta compression using up to 16 threads
Compressing objects: 100% (9/9), done.
Writing objects: 100% (11/11), 1.99 KiB | 1020.00 KiB/s, done.
Total 11 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/JorgeDLZ/ComandosGit.git
   c77ad43..baa7023  main -> main
```

5. git branch

Lista todas las ramas del repositorio y señala en cuál se encuentra actualmente. Permite visualizar y gestionar las ramas del proyecto, facilitando la organización del desarrollo en diferentes funcionalidades o versiones.

```
git branch
* Rama1
  main
```

6. git switch <nombre_rama>

Cambia a la rama especificada. Se usa para moverse entre distintas ramas dentro del repositorio, lo que permite trabajar en diferentes características sin afectar el código principal.

```
git switch main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
```

7. git pull

Descarga y fusiona los cambios del repositorio remoto en la rama local actual. Permite mantener el código local actualizado con las últimas modificaciones realizadas por otros colaboradores en el repositorio remoto.

```
git pull
Already up to date.
```

8. git init

El comando git init inicializa un nuevo repositorio de Git en un directorio. Se usa cuando se quiere comenzar a usar Git en un proyecto nuevo o convertir una carpeta existente en un repositorio. Crea una carpeta oculta llamada .git, donde se almacenará toda la información del repositorio.

```
git init
Reinitialized existing Git repository in C:/Users/Asus M3604Y/IdeaProjects/ComandosGit/.git/
```

9. git revert

El comando git revert se utiliza para deshacer cambios en un repositorio sin perder el historial. En lugar de eliminar un commit, crea uno nuevo que invierte los cambios realizados en el commit especificado. Se usa cuando se quiere deshacer un cambio sin alterar la línea de tiempo del proyecto, lo que es útil en entornos colaborativos.

10. git clone

Clona (copia) un repositorio remoto en la computadora local, descargando todo su historial y archivos. Se usa cuando se quiere trabajar con un proyecto ya existente alojado en plataformas como GitHub.

```
C:\Users\Asus M3604Y>git clone https://github.com/usuario/repo.git mi_carpeta
Cloning into 'mi_carpeta'...
```

11. git branch -D <nombre_rama>

Elimina una rama de manera forzada en el repositorio local. Se usa cuando ya no se necesita una rama y se quiere eliminar sin importar si tiene cambios sin fusionar.

```
git branch -D Rama1
Deleted branch Rama1 (was baa7023).
```

12. git switch -c <nombre_rama>

Crea una nueva rama y cambia a ella en un solo paso. Se usa cuando se quiere comenzar a trabajar en una nueva funcionalidad o corrección sin afectar la rama actual.

```
Switched to a new branch 'rama2'
```

13. git fetch --all

Descarga información actualizada de todas las ramas del repositorio remoto, pero sin fusionarlas con las ramas locales.

```
~\IdeaProjects\ComandosGit git:[rama2]
git fetch --all
```

14. git branch -r

Muestra todas las ramas remotas disponibles en el repositorio. Sirve para ver qué ramas existen en el repositorio remoto antes de traerlas a la máquina local.

```
git branch -r
  origin/HEAD -> origin/main
  origin/main
```

15. git log

Muestra el historial completo de commits en la rama actual, incluyendo el hash del commit, el autor, la fecha y el mensaje del commit.

```
commit baa702318460c8de5c25d00237741a51844929f0 (HEAD -> rama2, origin/main, origin/HEAD, main)
Author: JorgeDLZ <jorgediazlozano11@gmail.com>
Date:   Thu Feb 20 17:28:50 2025 -0500

    ComandosGit

commit c77ad432a6374fa6c39c86bcdd36288fcb14c59e
Author: JorgeDLZ <jorgediazlozano@gmail.com>
Date:   Thu Feb 20 16:39:52 2025 -0500

    first commit
```

16. git reflog

Muestra el historial de movimientos de la referencia HEAD, incluyendo cambios de rama, commits, resets y merges, incluso si han sido eliminados o deshechos.

```
baa7023 (HEAD -> rama2, origin/main, origin/HEAD, main) HEAD@{0}: checkout: moving from main to rama2
baa7023 (HEAD -> rama2, origin/main, origin/HEAD, main) HEAD@{1}: checkout: moving from Rama1 to main
baa7023 (HEAD -> rama2, origin/main, origin/HEAD, main) HEAD@{2}: checkout: moving from main to Rama1
baa7023 (HEAD -> rama2, origin/main, origin/HEAD, main) HEAD@{3}: commit: ComandosGit
c77ad43 HEAD@{4}: Branch: renamed refs/heads/master to refs/heads/main
c77ad43 HEAD@{6}: commit (initial): first commit
```

17. git log --oneline

Muestra el historial de commits en una sola línea por commit, facilitando la lectura. Se usa para obtener un resumen rápido del historial del repositorio.

```
git log --oneline
baa7023 (HEAD -> rama2, origin/main, origin/HEAD, main) ComandosGit
c77ad43 first commit
```

18. git merge

Fusiona otra rama en la rama actual, combinando sus cambios en un solo commit. Si hay conflictos, Git pedirá resolverlos antes de completar la fusión.

```
git merge
Already up to date.
```

19. git rebase

En lugar de fusionar, git rebase reaplica los commits de la rama actual sobre otra rama, manteniendo un historial más limpio.

```
git rebase main
Current branch main is up to date.
```

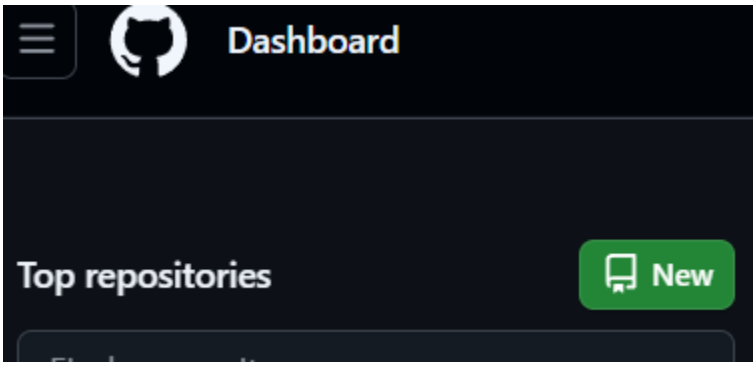
20. git push origin --delete <nombre_rama>

Elimina una rama en el repositorio remoto (GitHub,

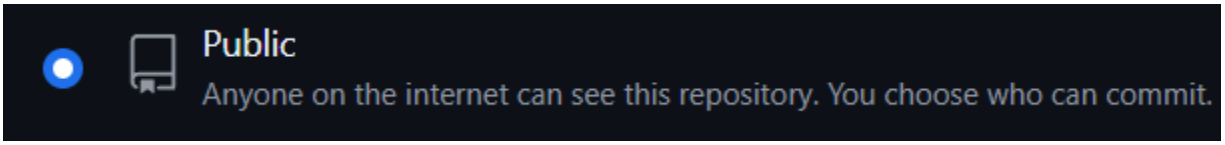
```
git push origin --delete rama2
To https://github.com/JorgeDLZ/ComandosGit.git
 - [deleted]         rama2
```

Cómo subir un proyecto a un repositorio de GitHub

Por último, para subir un proyecto a un repositorio de GitHub. Se deberán seguir los siguientes pasos. Una vez estando en GitHub deberemos de crear un nuevo directorio.



Después de haberle dado a crear un nuevo repositorio, debemos asignarle un nombre y dejarlo en su estado público para así poder compartir los proyectos de una manera más eficiente.



Una vez se haya creado el repositorio debemos de copiar los comandos que GitHub nos proporciona para poder vincular nuestro proyecto con el repositorio. Una vez copiados procedemos a pegarlos en el terminal de nuestro proyecto.

```
echo "# RepositorioP" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/JorgeDLZ/RepositorioP.git
git push -u origin main
```

```
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 250 bytes | 250.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/JorgeDLZ/RepositorioP.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
```

Una vez se haya vinculado el proyecto con el repositorio, como último paso se debe de hacer uso de los comandos antes mencionados. Se debe seguir el siguiente orden para una correcta ejecución.

Git status

Git init

Git add .

Git commit –m “mensaje”

Git push origin main

Conclusión:

En conclusión, se puede determinar que por medio de la implementación de las herramientas externas tales como Git y/o GitHub, se pueden compartir ideas de una manera eficiente a la hora de la realización de proyectos usando IntelliJ. Proporcionado así, una alternativa que funcione de manera que se puedan estructurar diferentes metodologías para la colaboración en la realización de proyectos informáticos, contribuyendo a la eficiencia en la realización de los mismos.

Referencias:

https://git-scm.com/book/es/v2/Ap%C3%A9ndice-C:-Comandos-de-Git-Seguimiento-B%C3%A1sico

https://blog.hubspot.es/website/comandos-github

https://www.jetbrains.com/help/idea/github.html

https://www.youtube.com/watch?v=6WtA2dUDvPY

OpenAI. (2025). ChatGPT (Febrero 2025) [Modelo de lenguaje de IA]. OpenAI. https://openai.com .Se utilizó ChatGPT (OpenAI, 2025) para generar explicaciones y complementar información en aproximadamente un 30-40% del contenido del documento.

https://gist.github.com/dasdo/9ff71c5c0efa037441b6