



UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS

Introducción a las Ciencias de la Computación

Práctica 1. Instalación de Java y Git

PROFESOR

Manuel Alcántara Juárez

AYUDANTE

Víctor Emiliano Cruz Hernández

12 de agosto de 2024

Índice

1. Instalación de Java	3
1.1. Windows	3
1.2. Debian	7
1.3. Prueba de instalación	8
2. Manejo de Git	9
2.1. Token de autenticación	9
2.2. Prácticas en Github	10
2.3. .gitignore	15
3. Ejercicios	16

1. Instalación de Java

La instalación de Java viene con dos componentes principales. El JDK proporciona herramientas de software esenciales para desarrollar en Java, como un compilador y un depurador. El JRE se utiliza para ejecutar programas en Java. Además, hay dos opciones principales de instalación de Java para elegir. OpenJDK es la implementación de código abierto de Java. Oracle JDK es la versión original de Java y su mantenimiento completo corre a cargo de Oracle, los desarrolladores de Java.

Ambas versiones están reconocidas oficialmente por Oracle. Ambos también son desarrollados por Oracle, pero OpenJDK cuenta con contribuciones de la comunidad debido a su naturaleza de código abierto. Sin embargo, a partir de Java 11, las dos opciones ahora son funcionalmente idénticas, como lo detalla Oracle. La elección entre cuál instalar se reduce a elegir la licencia adecuada para sus circunstancias. Además, OpenJDK tiene la opción de instalar JRE por separado, mientras que OracleJDK viene empaquetado con su JRE.

Nosotros vamos a intalar el JDK. La instalación depende del sistema operativo, la instalación se puede encontrar en línea:

https://www.java.com/en/download/help/download_options.html

En general, se mostrará el procedimiento en un Windows 10 y un sistema GNU/Linux basado en Debian. En otros sistemas UNIX el procedimiento dependerá del gestor de paquetes del sistema operativo.

1.1. Windows

Descargamos el instalador del sitio oficial:

<https://www.oracle.com/java/technologies/downloads/#jdk22-windows>

Linux	macOS	Windows
Product/file description	File size	Download
x64 Compressed Archive	184.14 MB	https://download.oracle.com/java/22/latest/jdk-22_windows-x64_bin.zip (sha256)
x64 Installer	164.31 MB	https://download.oracle.com/java/22/latest/jdk-22_windows-x64_bin.exe (sha256)
x64 MSI Installer	163.06 MB	https://download.oracle.com/java/22/latest/jdk-22_windows-x64_bin.msi (sha256)

Figura 1: Seleccionamos el instalador.

Después ejecutamos el instalador con permisos de administrador:

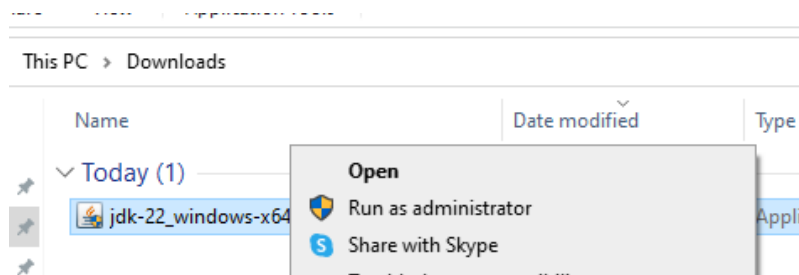


Figura 2: Ejecución del instalador de Java como administrador.

Nos mostrará el siguiente instalador, observamos bien la ruta que aparece abajo del ícono de la carpeta, después la vamos a necesitar.

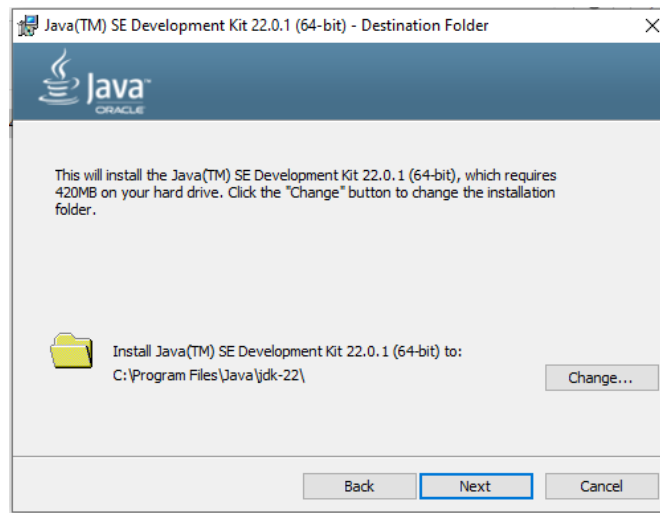


Figura 3: Menú principal del instalador de Java

Hacemos clic en Install y esperamos a que el instalador termine, después abrimos el Panel de Control:

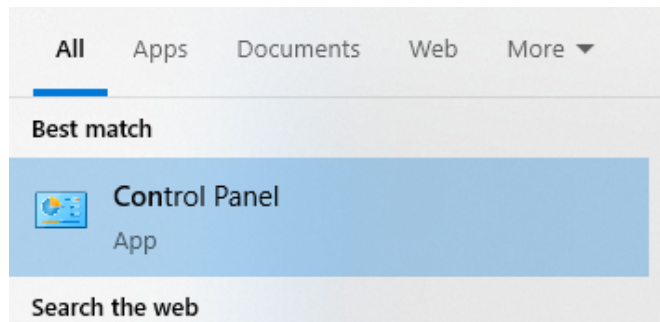


Figura 4: Abrimos el Panel de Control.

Hacemos clic en *System and Security* y después en *System*:

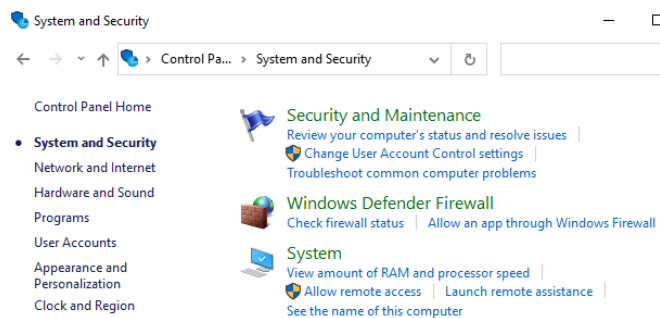


Figura 5: ControlPanel > System And Security > System

Hacemos clic derecho sobre *System* y después en *Open*, esto nos desplegará otro menú.

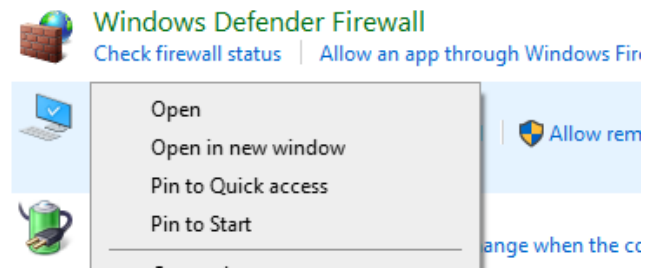


Figura 6: Clic derecho en *System*.

Después vamos a hacer clic en *Advanced system settings* en la parte izquierda del menú.

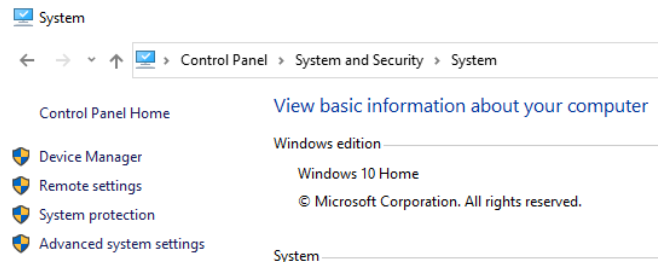


Figura 7: Advanced system settings.

Nos desplegará otro menú al que después vamos a hacer clic en *Environment Variables*:

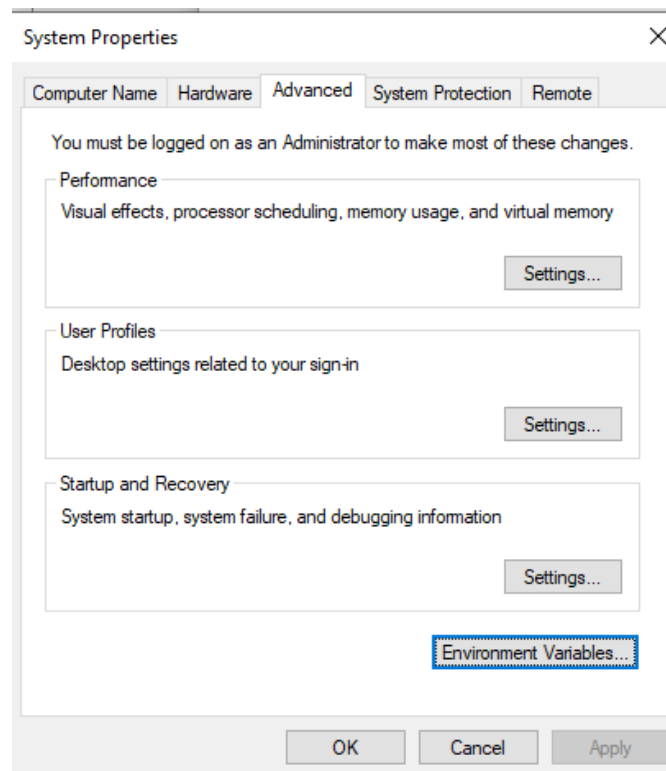


Figura 8: Menú desplegado con el botón *Environment Variables*

Después vamos a editar las variables de entorno de nuestro usuario (En mi caso *vicarus*):

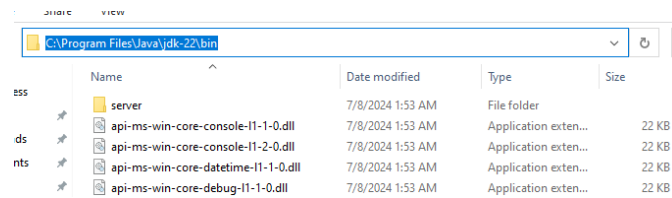


Figura 11: Ruta de los binarios de nuestra instalación

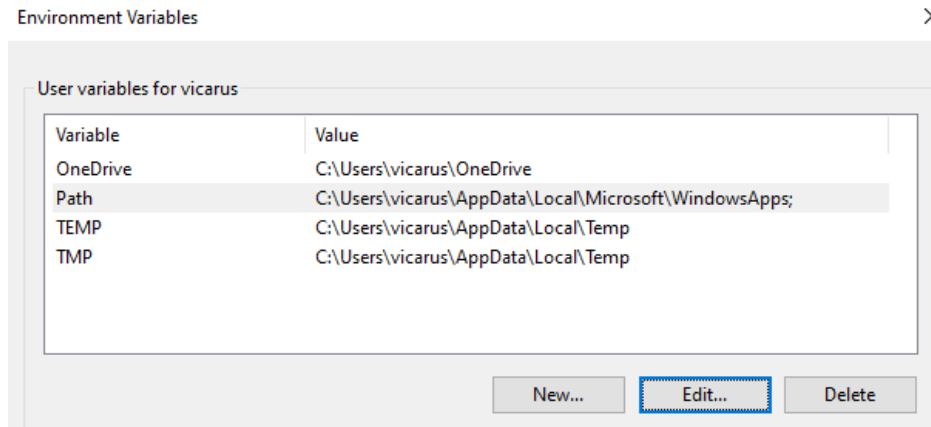


Figura 9: Hacemos clic en *Edit* después de seleccionar *Path*.

Después vamos a presionar el botón *New* para agregar una nueva ruta. Vamos a añadir la ruta donde estén los binarios de nuestra instalación de java.

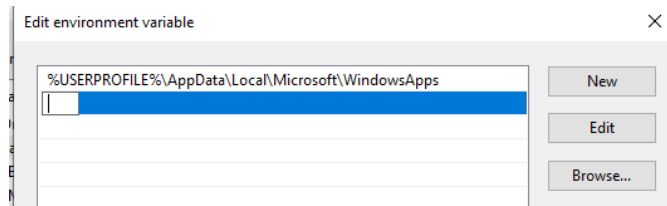


Figura 10: Enter Caption

En mi caso, la ruta `C:\ProgramFiles\Java\jdk-22\bin` es la que se va a añadir al Path. Después de añadirla al Path, hacemos clic en OK:

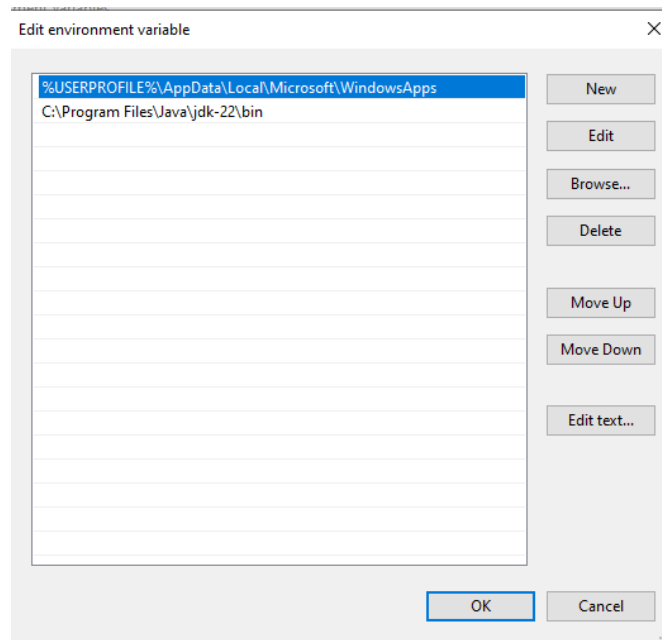


Figura 12: Hacemos clic en OK después de agregar el Path

Después de esto, abrimos una terminal presionando *Ctrl+R*, escribimos *cmd* en el prompt que se despliega y hacemos clic en OK.

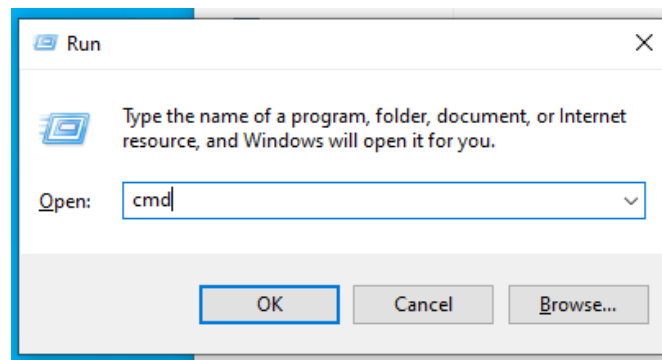


Figura 13: Enter Caption

Esto nos desplegará una terminal en la que si ejecutamos el comando `java -version`, obtenemos lo siguiente:

```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 10.0.19045.2965]
(c) Microsoft Corporation. All rights reserved.

C:\Users\vicarus>java -version
java version "22.0.1" 2024-04-16
Java(TM) SE Runtime Environment (build 22.0.1+8-16)
Java HotSpot(TM) 64-Bit Server VM (build 22.0.1+8-16, mixed mode, sharing)

C:\Users\vicarus>java -version_
```

Figura 14: Resultado de nuestro comando `java -version`.

1.2. Debian

la instalación de java en una distribución Debian se sigue de los siguientes pasos:

-
- 1 `sudo apt update`
 - 2 `sudo apt install default-jdk`

La instalación de Java en cualquier otra distribución UNIX puede encontrarse en línea.

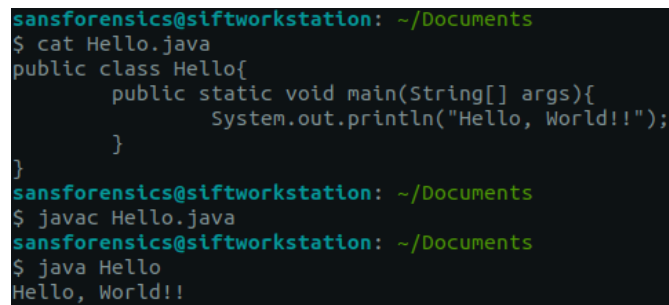
1.3. Prueba de instalación

Para probar que el código opera correctamente, podemos compilar y ejecutar el siguiente código desde la terminal:

```
// Hello.java
public class Hello{
    public static void main(String[] args) {
        System.out.println("Hello, World!!");
    }
}
```

Para ejecutarlo, con el editor de texto de nuestra preferencia guardamos el código de arriba en un archivo `Hello.java`, en nuestro caso esto fue en el directorio `~/Documents`, después ejecutamos los siguientes comandos:

- `javac Hello.java`: Tomamos el archivo escrito en Java y los compilamos en archivos de clase que se ejecutan en la máquina virtual Java.
- `java Hello`: El comando `java` inicia nuestro programa. Para ello, primero inicia la máquina virtual Java (JVM), carga la clase especificada `Hello` y llama al método `main()` de esa clase. El método debe declararse público y estático, no debe devolver ningún valor y debe aceptar un arreglo tipo `String` como parámetro.



```
sansforensics@siftworkstation: ~/Documents
$ cat Hello.java
public class Hello{
    public static void main(String[] args){
        System.out.println("Hello, World!!");
    }
}
sansforensics@siftworkstation: ~/Documents
$ javac Hello.java
sansforensics@siftworkstation: ~/Documents
$ java Hello
Hello, World!!
```

Figura 15: Compilación y ejecución del programa `Hello.java`

2. Manejo de Git

En esta sección vamos a ver algunos comandos básicos de git y uso de github para que se puedan subir las prácticas.

En general, se asume que ya se tiene una cuenta de github para poder hacer las prácticas. Además de esto, es necesario tener un token de autenticación asociado a la cuenta:

2.1. Token de autenticación

Para crear un token de autenticación vamos a seguir la documentación:

<https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/managing-your-personal-access-tokens#creating-a-personal-access-token-classic>

Una vez iniciada la sesión en nuestra cuenta, vamos a dar clic en el ícono de nuestro perfil en la parte superior derecha. Esto nos despliega un menú con la opción de *Settings*:

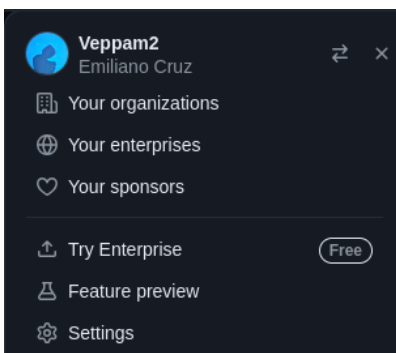


Figura 16: Menú tras hacer clic en nuestro ícono.

Damos clic en *Settings* y en la columna izquierda, hasta abajo damos clic en developer settings:

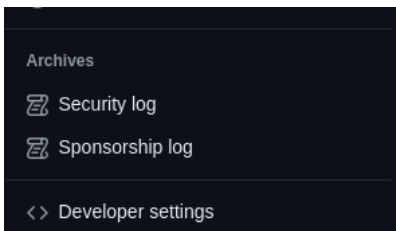


Figura 17: Damos clic en *Developer settings*

De la misma forma, en la columna izquierda damos clic en *Personal access tokens* y después en *Tokens(classic)*

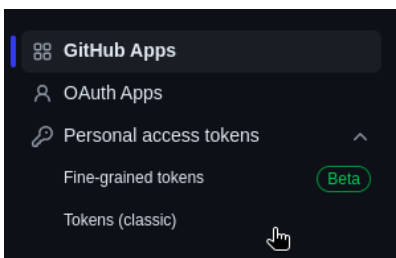


Figura 18: Clic en *Personal access tokens*

Después damos clic en *Generate new token (classic)*

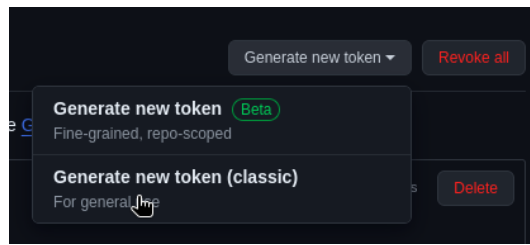


Figura 19: Clic en *Generate new token (classic)*

Esto nos desplegará un menú, donde podremos nombrar nuestro token, ajustar la vigencia del mismo y los permisos que se le otorgan. Para nuestro caso, marcando **repo**, es suficiente:

Figura 20: Permisos y vigencia de nuestro nuevo token

Finalmente nos deslizamos hasta abajo y encontramos un botón *Generate Token*, esto nos generará nuestro nuevo token:

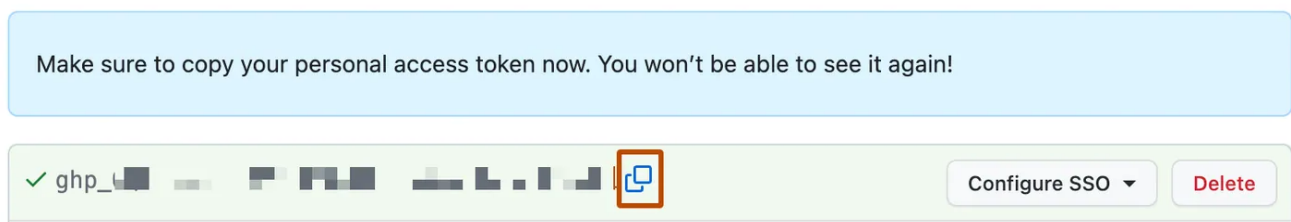


Figura 21: Nuevo token.

Es importante guardar este token pues no podrá ser desplegado de nuevo. Este token servirá para poder descargar localmente nuestras prácticas en github.

2.2. Prácticas en Github

Las prácticas se subirán al google classroom, estas serán un link a un repositorio en github, para ello vamos a ver cómo clonar una práctica a nuestra computadora y subir nuestros cambios como un flujo de trabajo básico.

Para cada práctica, se va a enviar una invitación a la que se deberá de aceptar para que se cree un repositorio en su cuenta con las instrucciones a realizar. En nuestro caso, vamos a aceptar la invitación para la práctica Hola-Mundo.

FCiencias-Org-classroom-icc2025-1

Accept the assignment —

Hola-Mundo

Once you accept this assignment, you will be granted access to the `hola-mundo-Veppam` repository in the [FCiencias-Org](#) organization on GitHub.

Accept this assignment

Figura 22: Invitación para la práctica.

Una vez que hemos aceptado la invitación, en nuestro perfil vamos a ver una copia de la tarea, normalmente el archivo `README.MD` tendrá las instrucciones:



Figura 23: Archivo `README.md` de nuestro repositorio.

Ahora vamos a clonar nuestro repositorio. En la parte derecha de la página de nuestro repositorio encontramos un botón verde `<> Code`, al darle clic, nos desplegarán distintas opciones para clonar nuestro repositorio a nuestra computadora. En este caso, vamos a hacerlo via `HTTPS`.

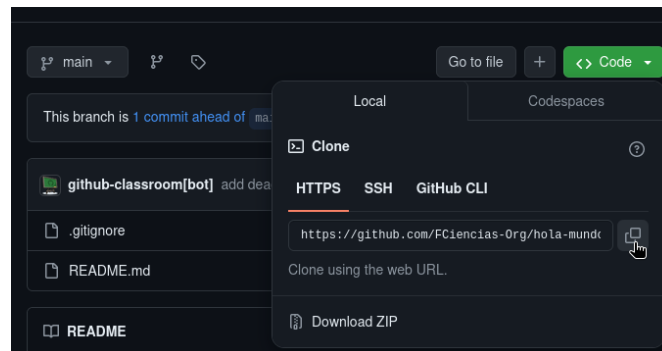


Figura 24: Link para clonar nuestro repositorio via `HTTPS`.

Después de copiar el link de nuestro repositorio, vamos a clonar el repositorio en nuestra computadora con el comando:

```
git clone <link-del-repositorio>
```

Tendremos que insertar nuestras credenciales. En este caso será nuestro nombre de usuario, y la contraseña será el token que creamos en la sección pasada.

```
sansforensics@siftworkstation: ~/Documents/Practicas
$ git clone https://github.com/FCiencias-Org/hola-mundo-Veppam.git
Cloning into 'hola-mundo-Veppam'...
Username for 'https://github.com': Veppam
Password for 'https://Veppam@github.com':
remote: Enumerating objects: 7, done.
remote: Counting objects: 100% (7/7), done.
remote: Compressing objects: 100% (7/7), done.
Receiving objects: 100% (7/7), done.
Resolving deltas: 100% (1/1), done.
remote: Total 7 (delta 1), reused 2 (delta 0), pack-reused 0
sansforensics@siftworkstation: ~/Documents/Practicas
$ ls
hola-mundo-Veppam
sansforensics@siftworkstation: ~/Documents/Practicas
$
```

Figura 25: Clonamos localmente nuestro repositorio

Nos cambiamos a nuestro repositorio con el comando `cd hola-mundo-Veppam`. Y observamos que tiene distintos directorios y archivos. Además nos encontramos en la rama `main`

```
sansforensics@siftworkstation: ~/Documents/Practicas/hola-mundo-Veppam
$ ls -la
total 20
drwxrwxr-x 3 sansforensics sansforensics 4096 Jul 12 00:54 .
drwxrwxr-x 3 sansforensics sansforensics 4096 Jul 12 00:54 ..
drwxrwxr-x 8 sansforensics sansforensics 4096 Jul 12 00:54 .git
-rw-rw-r-- 1 sansforensics sansforensics 290 Jul 12 00:54 .gitignore
-rw-rw-r-- 1 sansforensics sansforensics 317 Jul 12 00:54 README.md
sansforensics@siftworkstation: ~/Documents/Practicas/hola-mundo-Veppam
$ git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
sansforensics@siftworkstation: ~/Documents/Practicas/hola-mundo-Veppam
$
```

Figura 26: Estado actual de nuestro repositorio local. Con el comando `git status` vemos el estado de nuestro repositorio.

Si seguimos las indicaciones del archivo `README.md` tenemos que crear un archivo `Hello.java` que despliegue el mensaje *Hello, World!!*. Todo lo que trabajemos dentro del directorio `hola-mundo-Veppam` va a ser rastreado por git, si es que así se lo indicamos, por ejemplo, después de crear nuestro archivo `Hello.java` dentro del directorio `src` que nosotros creamos de nuestro repositorio, el estado de nuestro repositorio es el siguiente:

```
sansforensics@siftworkstation: ~/Documents/Practicas/hola-mundo-Veppam
$ git status
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)
  src/

nothing added to commit but untracked files present (use "git add" to track)
sansforensics@siftworkstation: ~/Documents/Practicas/hola-mundo-Veppam
$
```

Figura 27: Estado de nuestro repositorio después de agregar nuestro archivo `src/Hello.java`.

Notamos que nuestro archivo `src/Hello.java` está en estado *Untracked*. Esto quiere decir que git todavía no está haciendo un registro del estado de este archivo, para que nuestro archivo esté incluido dentro de nuestro repositorio, ejecutamos el comando `git add <ruta-archivo>`. Esto hará, que en nuestro caso, el archivo `src/Hello.java` esté en estado *staged*, como lo mostramos:

```
sansforensics@siftworkstation: ~/Documents/Practicas/hola-mundo-Veppam
$ git add src/Hello.java
sansforensics@siftworkstation: ~/Documents/Practicas/hola-mundo-Veppam
$ 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:   src/Hello.java

sansforensics@siftworkstation: ~/Documents/Practicas/hola-mundo-Veppam
$
```

Figura 28: Estado de nuestro repositorio después de añadir nuestro archivo. Observamos el cambio de estado de nuestro archivo.

En este momento, lo único que hemos hecho es indicarle a git que nuestro archivo está dentro del área de preparación. Pero aún no le hemos indicado que queremos confirmar los cambios, es decir, que con todo lo que está en el área de preparación vamos a hacer un nuevo estado en nuestro repositorio **local** (Aún no remoto).

Para indicar que queremos confirmar los cambio, creamos un nuevo commit con el comando:

```
git commit -m "<mensaje-descriptivo>"
```

Si es la primera vez que hacen esta acción, puede saltar un mensaje como el siguiente:

```
sansforensics@siftworkstation: ~/Documents/Practicas/hola-mundo-Veppam
$ git commit -m "Nuevo archivo Hello.java"
Author identity unknown

*** Please tell me who you are.

Run

  git config --global user.email "you@example.com"
  git config --global user.name "Your Name"

to set your account's default identity.
Omit --global to set the identity only in this repository.

fatal: unable to auto-detect email address (got 'sansforensics@siftworkstation.(none)')
sansforensics@siftworkstation: ~/Documents/Practicas/hola-mundo-Veppam
$
```

Figura 29: Mensaje de configuración

Basta con ejecutar los dos comandos que se mencionan con las respectivas credenciales. Después de ello, volvemos a ejecutar nuestro comando para hacer un commit y observamos lo siguiente:

```
sansforensics@siftworkstation: ~/Documents/Practicas/hola-mundo-Veppam
$ git commit -m "Nuevo archivo Hello.java"
[main 53b8b79] Nuevo archivo Hello.java
 1 file changed, 5 insertions(+)
 create mode 100644 src/Hello.java
sansforensics@siftworkstation: ~/Documents/Practicas/hola-mundo-Veppam
$
```

Figura 30: Commit confirmado

Ahora vemos que el estado de nuestro repositorio local ha cambiado y tenemos confirmados los cambios a nuestro archivo:

```
sansforensics@siftworkstation: ~/Documents/Practicas/hola-mundo-Veppam
$ 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
sansforensics@siftworkstation: ~/Documents/Practicas/hola-mundo-Veppam
$ git log --oneline
53b8b79 (HEAD -> main) Nuevo archivo Hello.java
b166afa (origin/main, origin/HEAD) add deadline
1d2f56a Initial commit
sansforensics@siftworkstation: ~/Documents/Practicas/hola-mundo-Veppam
$
```

Figura 31: Cambios en nuestro repositorio y nuevo commit.

En este momento, ya terminamos de agregar los cambios a nuestro **repositorio local**, pero remotamente en nuestro repositorio de github, estos cambios no se han reflejado, para esto, ejecutamos el comando

```
git push origin <nombre-de-la-rama>
```

```
sansforensics@siftworkstation: ~/Documents/Practicas/hola-mundo-Veppam
$ git push origin main
Username for 'https://github.com': Veppam
Password for 'https://Veppam@github.com':
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 465 bytes | 465.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/FCiencias-Org/hola-mundo-Veppam.git
   b166afa..53b8b79  main -> main
sansforensics@siftworkstation: ~/Documents/Practicas/hola-mundo-Veppam
```

Figura 32: Subimos nuestros cambios al repositorio remoto.

Si refrescamos la página de nuestro repositorio remoto, podremos ver que el archivo se encuentra ahí:

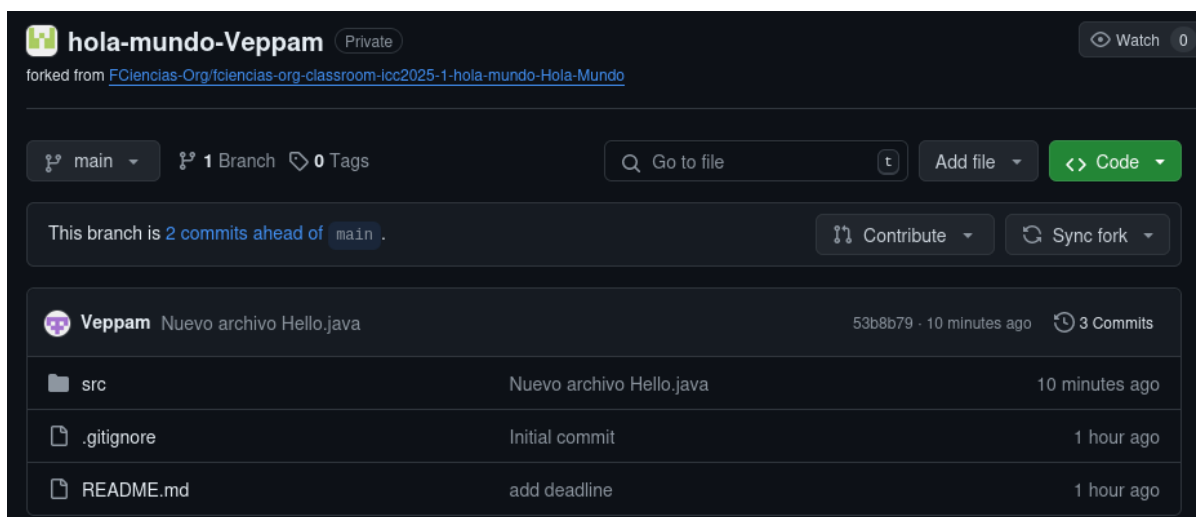


Figura 33: Cambios en nuestro repositorio remoto

En general, el uso de git es más extenso y sus funcionalidades se verán en el laboratorio, de la misma forma, git y github no son lo mismo, pero son herramientas que nos ayudan para el desarrollo de software.

2.3. .gitignore

El archivo `.gitignore` especifica archivos dentro del repositorio que git debe de ignorar. Una observación importante es el estado de los archivos que se indican en el `.gitignore`, los archivos que ya se encuentren rastreados en el repositorio, ya sea que estén en estado `staged` o `committed`, seguirán incluidos en el historial del repositorio. Archivos que estén en estado `untracked` y sean incluidos en el `.gitignore`, serán invisibles para el repositorio.

Algunas consideraciones del formato del archivo `.gitignore`:

- Cada línea del archivo especifica un archivo o directorio a ignorar.
- Cada archivo a ignorar se indica con un patrón, la documentación se encuentra en:

https://git-scm.com/docs/gitignore#_pattern_format

- Algunos archivos que se deben de incluir en las prácticas son los archivos `.class` entre otros que se indiquen.
- El archivo `.gitignore` se maneja como cualquier otro archivo dentro del repositorio, esto incluye el monitoreo de los cambios y estado de los archivos que git maneja.

3. Ejercicios

1. Clonar el repositorio y dentro de él crear un directorio `src` donde incluyan su archivo `HelloWorld.java` como el que se incluyó de ejemplo.
2. Incluir capturas de la terminal donde se muestre que el código compiló y se ejecutó correctamente en un directorio `imgs` a la misma altura del directorio `src`.
3. Configurar el `.gitignore` para evitar subir los archivos `.class` de su práctica
4. Agregar los archivos `HelloWorld.java`, `.gitignore` y las captura de las terminales a su repositorio local.
5. Subir la práctica a github classroom.