DataScience
in Human & Social Science for Women Empowerment

https://datascience-project.eu/

Ficha de formación

Título	Software
Palabras clave	SQL Github
Idioma	Español
Objetivos /metas / resultados de aprendizaje	**1. Entender cuándo utilizar SQL** **2. Importantes campos de aplicación de SQL** **3. Los comandos SQL básicos** **4. Cómo desarrollar código en equipo** 5. **Funcionalidades básicas de GitHub**
Curso de formación:	
Alfabetización en Ciencia de Datos	
Módulo de visualización de datos y análisis visual	
Introducción a la ciencia de datos para las ciencias humanas y sociales	
Ciencia de datos para el bien social	
Periodismo de datos y Storytelling	
● **Descripción**	● Este curso presentará brevemente los lenguajes y herramientas de programación más importantes que los científicos de datos utilizan a diario. ● Se esbozará el contexto y el propósito en el que se suelen utilizar y se presentarán los comandos más valiosos para los principiantes: ○ **SQL** se ha convertido en la piedra angular de la gestión de datos moderna. En este curso exploraremos diferentes formas en que SQL se puede utilizar para recuperar datos de bases de datos. ○ Discutiremos qué es **GitHub** , qué características ofrece, y cómo los desarrolladores de software pueden beneficiarse de él. ● Al final de este curso, los estudiantes conocerán el campo de actividad y los comandos más comunes.
Contenidos organizados en tres niveles	**2. Introducción a SQL** **2.1 Información general** **2.1.1 SQL: el lenguaje estándar mundial para la gestión y el análisis de bases de datos** El lenguaje de consulta estructurado (SQL) es un lenguaje de bases de datos ampliamente utilizado y diseñado para gestionar y manipular bases de datos relacionales. Es una potente herramienta que permite a los usuarios analizar, manipular y procesar datos de diversas maneras, lo que lo convierte en una opción popular en muchos sectores y áreas de

Co-funded by the Erasmus+ Programme of the European Union

With the support of the Erasmus+ programme of the European Union. This document and its contents reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

aplicación, como las finanzas, el comercio electrónico, la sanidad y la administración pública.

Una de las principales características de SQL es su flexibilidad. Ofrece un método estándar para mantener y procesar grandes cantidades de datos, lo que lo convierte en la primera opción para las empresas que necesitan gestionar grandes cantidades de información. Su flexibilidad también permite generar consultas e informes personalizados, lo que proporciona a los usuarios información valiosa sobre los datos con los que trabajan.

Además de su flexibilidad, SQL también es muy eficaz en su capacidad para procesar y manipular datos de forma rápida y eficiente. Con el uso de complejas sentencias SQL y técnicas de indexación, SQL puede localizar y recuperar datos rápidamente, lo que es especialmente importante cuando se trabaja con grandes bases de datos.

Otra ventaja de SQL es su facilidad de uso. Para la mayoría de los usuarios, es relativamente fácil de aprender y utilizar, lo que lo convierte en una opción popular para empresas y organizaciones de todos los tamaños. También hay numerosas fuentes disponibles para aprender el lenguaje, desde tutoriales y cursos on line hasta libros de texto y manuales.

A pesar de sus muchas ventajas, SQL tiene algunas limitaciones. Por ejemplo, no siempre es la mejor opción para trabajar con datos no estructurados, como imágenes, vídeos o archivos de audio. Además, puede no ser la opción más eficaz para determinados tipos de análisis, como los que requieren técnicas estadísticas avanzadas.

2.1.2 Propósito de SQL

Uno de los principales usos de SQL es recuperar datos de una base de datos. Esto puede incluir la selección de columnas específicas de datos, el filtrado de datos basado en criterios específicos o la combinación de datos de varias tablas. Por ejemplo, supongamos que tienes una base de datos de clientes y sus pedidos. Con SQL, puedes recuperar fácilmente una lista de todos los pedidos de un cliente concreto o de todos los pedidos de un periodo de tiempo determinado.

Otro uso de SQL es añadir nuevos datos a una base de datos, editar datos existentes o eliminar datos que ya no se necesitan.

Esto puede ser especialmente útil cuando se necesita actualizar grandes cantidades de datos a la vez. Por ejemplo, si necesitas actualizar la dirección de envío de todos los clientes que viven en un determinado código postal, puedes hacerlo rápida y fácilmente con SQL.

Además de gestionar datos, SQL también puede utilizarse para crear y gestionar tablas completas en bases de datos. Esto incluye la creación de nuevas tablas, la modificación de tablas existentes y la eliminación de tablas que ya no son necesarias. Por ejemplo, si deseas crear una nueva tabla para registrar los datos de ventas de tu empresa, puedes utilizar SQL para definir la estructura de la tabla y especificar los tipos de datos de cada columna.

Por último, SQL es especialmente adecuado para procesar grandes cantidades de datos. Esto se debe a que está diseñado para una alta eficiencia y puede manejar consultas y operaciones complejas sin esfuerzo. Esto lo convierte en la mejor opción para las empresas y organizaciones que necesitan gestionar y analizar grandes cantidades de datos con regularidad.

2.1.3 Sólo hay 4 cosas esenciales que necesitas para hacer un uso valioso de SQL

Si quieres empezar a utilizar SQL, puede que te estés preguntando qué herramientas y conocimientos necesitas para empezar. Afortunadamente, sólo necesitas 4 cosas esenciales.

En primer lugar, necesitas un sistema de gestión de bases de datos (SGBD). Un SGBD es un sistema informático que permite crear, gestionar y manipular bases de datos. Hay muchos SGBD diferentes, pero entre los más populares están MySQL, Oracle, PostgreSQL y Microsoft SQL Server. Estos sistemas permiten organizar y almacenar datos, así como acceder a ellos y manipularlos mediante SQL.

Lo **segundo** que necesitas es una base de datos. Una base de datos es una colección de datos organizados de una manera específica para facilitar el acceso y la edición. Hay muchos tipos diferentes de bases de datos, pero entre las más populares están Oracle, PostgreSQL, MySQL y SQL Server. Puedes descargar la base de datos de código abierto PostgreSQL desde su sitio web: https://www.postgresql.org/

Co-funded by the Erasmus+ Programme of the European Union

With the support of the Erasmus+ programme of the European Union. This document and its contents reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Lo **tercero** que necesitas es un cliente SQL. Un cliente SQL es una herramienta que te permite conectarte a una base de datos y ejecutar sentencias SQL. Hay muchos clientes SQL diferentes, pero entre los más populares están MySQL Workbench, SQL Developer y SQL Server Management Studio. También puedes utilizar un lenguaje de programación como Java, Python o C# para ejecutar sentencias SQL para una base de datos.

En **cuarto** y último lugar, se necesitan conocimientos básicos de sintaxis y conceptos SQL. Esto incluye saber cómo crear y manipular tablas, cómo utilizar las sentencias SELECT, INSERT, UPDATE y DELETE para interactuar con los datos y cómo utilizar las cláusulas WHERE para filtrar datos. Una vez que comprendas estos conceptos, podrás utilizar SQL para extraer, manipular y gestionar datos en una gran variedad de situaciones.

2.2 Las bases de datos relacionales son clave

Las bases de datos relacionales son una parte fundamental de la gestión moderna de datos en organizaciones de todos los tamaños. Se utilizan para almacenar, gestionar y analizar datos en diversos entornos, desde pequeñas empresas a grandes corporaciones, organismos públicos y organizaciones sin ánimo de lucro.

Una de las principales ventajas de una base de datos relacional es que grandes cantidades de datos se almacenan de forma estructurada y organizada, lo que permite recuperarlos (encontrarlos) muy rápidamente. Por ejemplo, puedes utilizar una base de datos relacional para almacenar y gestionar todos los datos de los empleados de tu empresa.

Las bases de datos relacionales se basan en una idea simple pero poderosa: los datos pueden almacenarse en tablas, que constan de filas y columnas. Cada **tabla** representa un tipo concreto de datos, y una **fila** de la tabla representa un único elemento. Por ejemplo, en nuestra tabla de empleados, cada fila contiene un individuo distinto, y cada **columna** representa un **atributo** diferente del empleado (por ejemplo, nombre, dirección y número de teléfono). Una fila también se denomina **registro** y los atributos individuales se almacenan en campos. La fila de cabecera de la tabla también se denomina **esquema**,

porque en ella se indica qué datos se almacenan en cada columna.

Otra ventaja de una base de datos relacional es la capacidad de garantizar la integridad y coherencia de los datos. Esto significa que se puede establecer reglas y restricciones sobre cómo se introducen los datos en la base de datos y luego asegurarse de que esas reglas se cumplen. Por ejemplo, se puede exigir que se almacene una dirección para todos los empleados, o un número de teléfono único. Así se evitan errores e incoherencias en los datos que pueden causar problemas.

Existen otros tipos de bases de datos. Por ejemplo, las bases de datos NoSQL están diseñadas para manejar datos no estructurados y semiestructurados, como documentos, gráficos y pares clave-valor. Suelen utilizarse para análisis de big data y aplicaciones web en tiempo real.

Otros tipos son las bases de datos orientadas a objetos, las bases de datos gráficas o las bases de datos en memoria. La elección del tipo de base de datos depende de las necesidades y requisitos específicos de la aplicación utilizada.

2.3 Sentencias SQL

2.3.1 Estructura y sintaxis de las sentencias SQL

En este guión, aprenderemos sobre las **palabras clave** comunes

Co-funded by the
Erasmus+ Programme
of the European Union

With the support of the Erasmus+ programme of the European Union. This document and its contents reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

(también conocidas como cláusula) utilizadas en las sentencias de SQL y la **estructura general** y **sintaxis** de las sentencias SQL.
Palabras clave identifica el tipo de operación que se va a realizar. A continuación se indican las palabras clave más utilizadas en las sentencias SQL:

- **SELECT:** Se utiliza para recuperar datos de una o varias tablas
- **INSERT:** Se utiliza para insertar nuevos datos en una tabla
- **UPDATE:** Sirve para actualizar los datos existentes en una tabla
- **DELETE:** Sirve para eliminar datos de una tabla
- **ALTER:** Sirve para modificar la estructura de una tabla
- **DROP:** Sirve para eliminar una tabla o un objeto de la base de datos
- **CREATE:** Sirve para crear una nueva tabla u objeto en base de datos
- **USE:** Permite seleccionar una base de datos con la que trabajar
- **SHOW:** Permite mostrar información sobre un objeto de la base de datos

La **estructura y la sintaxis** de las sentencias SQL pueden variar en función del sistema de gestión de bases de datos (SGBD) específico que se utilice. Sin embargo, existen algunas directrices generales que se aplican a la mayoría de las sentencias SQL. Una sentencia SQL básica suele constar de los siguientes elementos:
Palabras clave: La palabra clave o cláusula que identifica el tipo de operación a realizar
Argumentos: Uno o más argumentos o parámetros que proporcionan información adicional sobre la operación a realizar
Punto y coma: Todas las sentencias SQL terminan en punto y coma (;)
Por ejemplo, una sentencia SELECT básica tendría el siguiente aspecto:

```
SELECT column1, column2 FROM table_name;
```

En esta sentencia, "SELECT" es la palabra clave, "column1, column2" son los argumentos, y "table_name" es la tabla en la que se recuperan los datos. El punto y coma al final de la sentencia indica el final de la sentencia SQL.

2.3.2 Sentencia SELECT

Un ejemplo sencillo de una sentencia **SELECT** con respecto a nuestra base de datos relacional podría ser el siguiente:

```
SELECT * FROM employees;
```

"**SELECT**" es la palabra clave y
"*" y "**FROM employees**" son los argumentos
Esta sentencia SQL está recuperando todos los datos de nuestra tabla llamada "employees". El * representa todo y en este caso todas las columnas.
En la mayoría de los casos, sin embargo, no queremos todos los datos almacenados, sino solo una parte de ellos. Por ejemplo, el nombre y el número de teléfono asociado de todos los empleados:

SELECT Name, TelNr **FROM** employees;

Veamos ahora los argumentos especiales **FROM** y **WHERE:**
FROM determina la tabla o tablas para esta consulta
WHERE se usa para añadir una condición a la consulta. Si la condición es un texto, se enciarra el texto en comillas simples
Por ejemplo, si sólo queremos el número de teléfono de los empleados que se apelliden Mouse, utilizaremos:
SELECT Name, TelNr **FROM** employees **WHERE** Name = 'Mouse';
Las sentencias SELECT son muy potentes y ofrecen varias perspectivas a los científicos de datos. Para obtener más argumentos especiales, consulta el anexo de esta guía.

2.3.3 Otras setencias SQL útiles

Los comandos DML – Lenguaje de manipulación de datos de SQL son: SELECT, INSERT, UPDATE, y DELETE y son esenciales para gestionar y manipular datos dentro de una base de datos relacional. Al aprender estos 4 comandos y cómo utilizarlos eficazmente, los usuarios pueden llegar a ser competentes en SQL y utilizarlo para manejar una amplia gama de tareas de base de datos.
SELECT: El comando SELECT se utiliza para recuperar datos de una o varias tablas de una base de datos. Permite a los usuarios especificar las columnas que desean recuperar y filtrar los datos en función de criterios específicos. Hemos trabajado con ellos en el último capítulo.
INSERT: El comando INSERT se utiliza para añadir nuevos datos a una tabla de una base de datos. Permite a los usuarios especificar los valores que se insertarán en la tabla y las columnas en las que se insertarán.

INSERT INTO table_name (column1, column2 …)
VALUES (value1, value2 … valueX);

INSERT INTO va seguida del nombre de la tabla en la que se van a insertar los datos, en el ejemplo "nombre_tabla".
La siguiente parte de la sentencia especifica las columnas de la tabla en las que se van a insertar los datos, lo que se representa mediante "(column1, column2 ...)". A continuación, se utiliza la

Co-funded by the Erasmus+ Programme of the European Union

With the support of the Erasmus+ programme of the European Union. This document and its contents reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

palabra clave VALUES para especificar los datos reales que se insertarán en el campo. Los valores deben coincidir con el orden de las columnas especificado en la sección anterior y se representan mediante "(value1, value2 ... valueX)".
Por ejemplo, si insertamos datos de nuestra tabla denominada "employees" con columnas "Name", "Adresse" y "TelNr", la sentencia SQL para insertar un nuevo registro podría ser la siguiente:
INSERT INTO employees (Name, Adresse, TelNr) VALUES ('John Smith', 'High Street', 50573);
Hay que tener en cuenta que no es necesario especificar todas las columnas de una tabla en la sentencia INSERT INTO. Si se omite una columna, la base de datos asignará un valor por defecto o insertará un valor NULL en esa columna, dependiendo de cómo se haya creado la tabla.
Además, el orden en que se especifican los valores debe coincidir con el orden de las columnas. Si el orden es diferente o si falta algún valor, la base de datos informará de un error.
UPDATE: El comando UPDATE se utiliza para modificar los datos existentes en una tabla de una base de datos. Permite a los usuarios especificar los nuevos valores que deben sustituir a los existentes y filtrar los datos en función de criterios específicos.
UPDATE table_name
SET column1 = value1, column2 = value2 ... columnN = valueN
[**WHERE** condition];
UPDATE table_name especifica el nombre de la tabla a actualizar.
SET column1 = value1, column2 = value2 ... columnN = valueN especifica los valores de una o más columnas de la tabla en nuevos valores. Cada columna y su valor correspondiente están separados por un signo (=), y las columnas múltiples están separadas por comas.
[WHERE condition] es una cláusula opcional que especifica la(s) condición(es) que debe(n) cumplirse para que se produzla la actualización. Si no se especifica ninguna condición, se actualizarán todas las filas de la tabla.
Ejemplos de condiciones válidas:
WHERE Name = 'Mouse';
WHERE Age > 50;
WHERE category = 'Electronics' AND stock_quantity > 0;
WHERE order_date < '2022-01-01';
DELETE: El comando DELETE se utiliza para eliminar datos de una tabla de una base de datos. Permite a los usuarios filtrar

Comentado [1]: Add an example of how to specify this condition

Comentado [2R1]: done

Co-funded by the Erasmus+ Programme of the European Union

With the support of the Erasmus+ programme of the European Union. This document and its contents reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

	los datos en función de criterios específicos y eliminar todos o un subconjunto de los datos que coinicidan con estos criterios. **DELETE FROM** table_name **WHERE** {condición}; DELETE FROM table_name especifica el nombre de la tabla de la que deben eliminarse filas. WHERE {condición} es una cláusula opcional qu especifica la(s) condición(es) que debe(n) cumplirse para que se eliminen las filas. Si no se especifica ninguna condición, se borrarán todas las filas de la tabla! Ejemplos para condiciones válidas: **WHERE** NameID = 123456; Esta sentencia eliminaría la fila con un NameID de 12345 **WHERE** NameID **IN** (12345, 23456, 34567); Esta sentencia eliminaría todas las filas en las que el "NameID" es 12345 or 23456 or 34567. Es importante tener cuidado al utilizar la sentencia DELETE, ya que elimina datos de una tabla de forma permanente. Se recomienda hacer una copia de seguridad de los datos o utilizar una transacción para deshacer los cambios si es necesario. En conclusión, SQL es un lenguaje de bases de datos potente y flexible que se utiliza ampliamente en muchos sectores y áreas de aplicación diferentes. Su facilidad de uso, flexibilidad y eficacia lo convierten en una opción excelente para las empresas y organizaciones que necesitan procesar y mantener grandes cantidades de datos. Con sus numerosas ventajas y recursos disponibles para su aprendizaje, SQL es una herramienta valiosa para cualquiera que trabaje con bases de datos relacionales
Autoevaluación (preguntas y respuestas de elección múltiple)	1. ¿Quién utiliza SQL? A) Los humanos para trabajar con datos almacenados en una base de datos relacional B) Los desarrolladores web para programar sitios web C) Administradores de bases de datos para gestionar los datos 2. ¿Para qué sirve SQL? A) Para crear y eliminar datos en una base de datos B) Para pagos en tiendas web C) Para recuperar datos en un entorno que no sea una base de datos 3. ¿Qué palabras clave pueden iniciar una sentencia SQL? A) FROM B) WHERE

Comentado [3]: Here too an example would be useful, e.g. listing the name of the column

Comentado [4R3]: done

Con formato: Fuente: Negrita

Con formato: Fuente: Negrita

Con formato: Fuente: Negrita

Co-funded by the Erasmus+ Programme of the European Union

With the support of the Erasmus+ programme of the European Union. This document and its contents reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

	C) SELECT
Recursos (videos, enlaces de referencia)	https://www.postgresql.org/ https://www.youtube.com/watch?v=HXV3zeQKqGY
Material relacionado	
Contenidos organizados en 3 niveles	**3. Entender GitHub** **3.1 Propósito de GitHub** **3.1.1 Gestión de código de Software de gran tamaño** Para todos los desarrolladores de software, la gestión del código es una parte esencial de la programación. Los sistemas de control de versiones permiten a los desarrolladores controlar los cambios en su código, colaborar con otros y gestionar sus proyectos de forma eficiente. Un sistema de control de versiones popular es Git, y GitHub es un servicio de control de versiones basado en la nube para proyectos de desarrollo de software construido sobre Git. GitHub ofrece una interfaz fácil de usar para crear y gestionar repositorios Git, así como herramientas para colaborar con otros desarrolladores en un proyecto. GitHub puede utilizarse tanto para proyectos personales como comerciales y es gratuito en su edición básica. GitHub puede caracterizarse como una especie de red social para desarrolladores de software. Los miembros pueden seguirse unos a otros, valorar su trabajo, recibir actualizaciones sobre proyectos concretos y comunicarse de forma pública o privada. Permite a los desarrolladores compartir su trabajo, aprender de los demás y crear una comunidad. GitHub también ofrece un completo conjunto de herramientas de gestión de proyectos que facilitan a los desarrolladores la gestión de sus proyectos. Incluye herramientas para el seguimiento de problemas, la organización de tareas y la colaboración con otras personas. GitHub se integra con muchas otras herramientas y servicios, incluidos los de integración y despliegue continuos. Esto permite a los desarrolladores automatizar el proceso de prueba, construcción y despliegue de sus cambios de código. Desde finales de 2018, GitHub Inc es propiedad de Microsoft. **3.1.2 ¿Qué es el control de versiones?**

Comentado [5]: I thought PostgreSQL is also a database management system? Not sure about this though

Comentado [6R5]: Yes, it is both!

Con formato: Color de fuente: Automático

Con formato: Sangría: Izquierda: 1,25 cm

Co-funded by the Erasmus+ Programme of the European Union

With the support of the Erasmus+ programme of the European Union. This document and its contents reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Cuando se trabaja en proyectos de software complejos, hacer un seguimiento de los cambios y gestionar múltiples versiones del código puede ser una tarea ardua. Los sistemas de control de versiones como Git ofrecen una solución a este problema, ya que rastrean y registran los cambios.

Git ofrece funcionalidades para recuperar versiones antiguas de un proyecto, comparar, analizar, fusionar cambios y mucho más. Este proceso se denomina **version control (control de versiones)**, y ayuda a los desarrolladores a hacer un seguimiento del historial de su código. Git no es el único sistema de control de versiones disponible. También existen otros sistemas de control de versiones como Perforce, Mercurial, CVS y SVN. Sin embargo, Git se ha convertido en el sistema de control de versiones más popular y utilizado entre los desarrolladores debido a su flexibilidad, velocidad y facilidad de uso.

Una de las características únicas de Git es que es un sistema de control de versiones descentralizado. A diferencia de otros, Git no depende de un servidor central para mantener las versiones antiguas de los archivos. En su lugar, funciona de forma completamente local, almacenando esos datos como carpetas en el disco duro del usuario. Esto se denomina **repository (repositorio)**. Esto permite a los desarrolladores trabajar en su código sin conexión y facilita el seguimiento de los cambios sin depender de un servidor central.

Si un usuario quiere trabajar con otros en el mismo código, puede proporcionar una copia de su repositorio on line para que todo el equipo pueda acceder a ella. Esto permite que varios desarrolladores trabajen en el mismo código sin sobrescribir los cambios de los demás.

3.2 Cómo utilizarlo

3.2.1 Instalar en tu dispositivo

Para utilizar Github de forma productiva, necesitas Git en tu ordenador local. Hay muchos programas Git que se pueden utilizar de forma gratuita o de pago:

Windows - "Git para Windows" proporciona un cliente GUI y un emulador de línea de comandos BASH.
https://git-scm.com/downloads/win o
https://gitforwindows.org/

Linux - sólo tienes que abrir un nuevo terminal e instalar Git a través del gestor de paquetes de tu distribución de Linux. Para Ubuntu, el comando es, por ejemplo, sudo apt-get install git

Mac OS - La forma más sencilla es instalar homebrew y luego simplemente ejecutar brew install git desde tu terminal. https://brew.sh/ o https://git-scm.com/downloads/mac

3.2.2 Crear una cuenta y configurar GitHub

Antes de poder utilizarlo, crear una cuenta en Github www.github.com

Hay varias configuraciones para el aspecto y la funcionalidad del cliente. Decidir de acuerdo a sus preferencias para trabajar de manera eficiente.

Sin embargo, es importante la configuración del **nombre del usuario** y la **dirección del correo electrónico** correspondiente abrir "Git Bash" y ejecutar estos comandos en la línea de comandos:
git config --global user.name “Mary”
git config --global user.email “mary.smith@email.eu”

Todas las actividades en Git están vinculadas al nombre de usuario y la dirección de correo electrónico especificados en la configuración. Esto hace que las modificaciones sean rastreables porque otros usuarios siempre saben quién hizo los cambios específicos y proporciona una visión general en proyectos donde muchos desarrolladores están trabajando.

3.3 Cómo utilizar GitHub

3.3.1 Términos comunes en GitHub

Cuando se trabaja con Git, es esencial comprender su terminología para poder utilizarlo con eficacia.

A **repository**, or repo for short, is a folder in which all files and their version histories are stored. It is the central location for managing and organizing code. A repository can be hosted on a remote server, such as GitHub or Bitbucket, or it can be stored locally on your computer.
Un **respository** (**repositorio)**, o repo para abreviar, es una carpeta en la que se almacenan todos los archivos y sus historiales de versiones. Es la ubicación central para gestionar y organizar el código. Un repositorio puede estar alojado en un

servidor remoto, como GitHub o Bitbucket, o puede almacenarse localmente en tu ordenador.

Branch (rama) es un espacio de trabajo en el que puedes hacer cambios aislados que no afectarán a otros y que tiene su propio historial. Es como una línea de tiempo independiente en la que puedes experimentar con cambios sin afectar al código base principal. Los desarrolladores pueden trabajar simultáneamente en distintas ramas, lo que facilita la colaboración y evita conflictos.

Commit es un registro guardado de las modificaciones realizadas en un archivo dentro del repositorio. Es el estado de nuestro repositorio en un momento dado. Una instantánea a la que el código puede ser restaurado. Los commits son esenciales en Git porque te permiten hacer un seguimiento de los cambios a lo largo del tiempo y volver a una versión anterior de tu código si es necesario.

Después de realizar actualizaciones en un repositorio, otros desarrolladores pueden descargar todos los cambios con una **pull request** o PR. Una pull request es una solicitud para fusionar cambios de una rama a otra. Esto permite a los desarrolladores revisar y aprobar los cambios antes de que se fusionen en el código base principal.

Push es el proceso de añadir un cambio local al repositorio remoto. Cuando envías tus cambios, están disponibles para que otros los vean y descarguen. Este es un paso esencial para colaborar con otros desarrolladores.

Una vez aprobada una pull request, la confirmación se fusionará de una rama a otra. La fusión combina los cambios de una rama en otra, normalmente la rama principal. Este proceso garantiza que todos los cambios se incorporen al código base principal.

Clone es una copia totalmente funcional de un proyecto. Copiar un repositorio desde un servidor remoto es clonar. Esto le permite tener una copia completa en la máquina local, lo que facilita trabajar en su código sin depender de una conexión a Internet.

3.3.2 Primeros pasos en Github

Co-funded by the Erasmus+ Programme of the European Union

With the support of the Erasmus+ programme of the European Union. This document and its contents reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Al utilizar Git, primero hay que crear una carpeta
mkdir learning-git (=**m**a**k**e **dir**ectory)
En la que se pueda crear el repositorio para cada programa o proyecto
cd learning-git (= **c**hange **d**irectory)
git init

A continuación, puedes enlazar un repositorio local y uno on line con
git remote add origin
https://github.com/[username]/[projectname].git
Git funciona con el concepto de "área de preparación". Al principio, el área de preparación está vacía. Se pueden añadir archivos (o incluso líneas sueltas y partes de archivos) con el comando git add y finalmente confirmar todo (crear una instantánea) con git commit:
git add learning_script.txt
git commit –m «firstsnapshot»

Para subir código a un repositorio remoto, primero conéctate a:
git remote add origin https://github.com/learning.git
Después, los commits locales pueden ser transferidos al servidor, realizado cada vez que queramos actualizar el repositorio remoto
git push server_origin local_master
Una vez hecho esto, otros desarrolladores pueden descargar los cambios del repositorio remoto con un solo comando
git pull origin master
Para clonar un programa o proyecto remoto utiliza
git clone https://github.com/tutorial.git

3.3.3 Comandos relacionados con Branch

Cuando se desarrolla una nueva funcionalidad de software, la mejor práctica es trabajar en una copia del proyecto original, denominada rama. Una rama es una copia separada del código base que permite a los desarrolladores realizar cambios sin afectar a la versión activa del código.
Cada rama tiene su propio historial y aísla los cambios realizados en ella de otras ramas hasta que se decide fusionarlas. Las ventajas de este enfoque son:
1. Una versión estable (viva) del código no se ve afectada por errores no deseados: Cuando se trabaja en una rama separada, cualquier fallo o error que se produzca durante el proceso de desarrollo no afecta a la versión en vivo del código.

Co-funded by the Erasmus+ Programme of the European Union

With the support of the Erasmus+ programme of the European Union. This document and its contents reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

	2. Un equipo de desarrolladores puede trabajar en muchas funciones al mismo tiempo: Las ramas permiten a varios desarrolladores trabajar en distintas funciones al mismo tiempo sin interferir en el trabajo de los demás. 3. Cada desarrollador puede trabajar en su propia rama sin riesgo de que su código base se vea modificado por el trabajo de otro desarrollador: Trabajar en ramas separadas garantiza que los desarrolladores puedan trabajar de forma independiente y evitar conflictos entre sus cambios de código. 4. Se pueden desarrollar varias versiones de la misma función en distintas ramas y compararlas para encontrar la mejor versión: Las ramas permiten a los desarrolladores experimentar con distintos enfoques de una función y comparar los resultados antes de incorporar los cambios al código principal. En resumen, el uso de ramas es una buena práctica en el desarrollo de software, ya que ayuda a evitar conflictos y errores, facilita el trabajo en equipo y permite experimentar y comparar diferentes enfoques de una función. La rama por defecto de cada repositorio se llama master. Para crear más ramas, se utiliza el comando git branch <name> Cambia a la rama recién creada utilizando git checkout <name> Para fusionar dos ramas cambia a una y utiliza git merge <name> Para eliminar una rama utiliza git branch –d <name> En conclusión, muchas empresas utilizan GitHub. Así que, si estás buscando trabajo, te irá bien si ya estás familiarizado con GitHub. GitHub es también una plataforma de aprendizaje y colaboración. Explórala y amplía tus conocimientos y tu comunidad.
Autoevaluación (preguntas y respuestas de evaluación múltiple)	1. ¿Por qué es importante el control de versiones? A) Desarrollar código rápidamente B) Desarrollar un programa con un equipo en un proceso trazable C) Necesario para vender el proyecto 2. ¿Quién puede utilizar GitHub? A) Cualquiera que tenga una cuenta Git B) Cualquiera que lo haya comprado

Co-funded by the Erasmus+ Programme of the European Union

With the support of the Erasmus+ programme of the European Union. This document and its contents reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

	C) Sólo desarrolladores capacitados 3. ¿Qué es branch? A) Una empresa autorizada B) Un proyecto específico C) Una copia del proyecto original
Recursos (videos, enlaces de referencia)	https://www.github.com https://git-scm.com/downloads/win https://gitforwindows.org/ https://brew.sh/ https://git-scm.com/downloads/mac
Material relacionado	
PPT relacionado	
Bibliografía	
Proporcionado por	[Women in AI Austria]