



# MANEJO DE VERSIONES



UNIDAD 2 - IPC 2



# ¿Qué es control de versiones?

Se llama control de versiones a la gestión de cambios efectuados en un documento, programa, imagen, website y otros archivos que contengan información.

Los cambios se registran de forma automática y pueden ser identificados mediante números o combinaciones alfanuméricas.



Los sistemas de control de versiones son una categoría de herramientas de software que ayudan a un equipo de software a gestionar los cambios en el código fuente a lo largo del tiempo.

El software de control de versiones realiza un seguimiento de todas las modificaciones en el código por lo que, si se comete un error, los desarrolladores pueden ir atrás en el tiempo y comparar las versiones anteriores del código para ayudar a resolver el error al tiempo que se minimizan las interrupciones para todos los miembros del equipo.



# CONCEPTOS BÁSICOS

# REPOSITORIO

---

Un repositorio es como una carpeta para un proyecto. El repositorio del proyecto contiene todos los archivos de tu repositorio y almacena el historial de revisión de cada archivo.

Se puede ser propietario de repositorios individualmente o compartir la propiedad de los repositorios con otras personas en una organización.

# ETIQUETA (TAG)

---

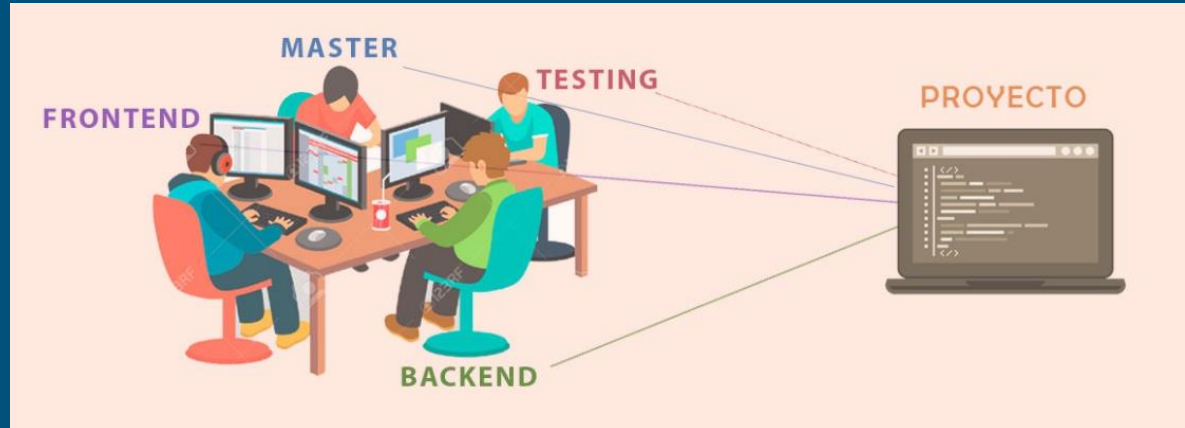
Es una palabra clave asignada a un dato almacenado en un repositorio. Las etiquetas son en consecuencia un tipo de metadato, pues proporcionan información que describe el dato (una imagen digital, un clip de vídeo o cualquier otro tipo de archivo informático) y que facilita su recuperación.

La diferencia entre las etiquetas y las palabras clave tradicionales es que las etiquetas son elegidas de manera informal y personal por los usuarios del repositorio.

# RAMA (BRANCH)

---

Son espacios o entornos independientes para que un Desarrollador sea Back-end, Front-end, Tester, etc. pueda usar y así trabajar sobre un mismo Proyecto sin chancar o borrar el conjunto de archivos originales del proyecto, dándonos flexibilidad para desarrollar nuestro proyecto de manera más organizada.



# RAMA (BRANCH)

---

Es recomendable contar con ciertas ramas creadas por defecto las cuales pueden ser:

## Develop

Esta rama es utilizada por el equipo de desarrollo principalmente para probar todos los nuevos desarrollos o refactors grandes.

## Staging

Esta rama es utilizada para que el equipo encargado de realizar el QA se encargue de generar todas las pruebas pertinentes que correspondan a nuevos desarrollos y correcciones, en ocasiones los cambios que pasan a esta rama pueden provenir de la rama develop y esto puede pasar cuando el equipo de desarrollo terminó de realizar pruebas en el branch.



# RAMA (BRANCH)

---

## UAT

También conocido como User Acceptance Tests, en este punto el QA realizado en Staging a finalizado satisfactoriamente y los cambios están listo para que los usuarios finales puedan dar su Vo Bo y dependiendo de su Re-Test validar si los cambios pasan a la rama master(producción), o se tiene que realizar alguna corrección.

## Master

Hasta este punto todos los cambios y correcciones se encuentran listos para ser mandados a un ambiente de producción.

# CONFLICTO (CONFLICT)

---

Los conflictos de fusión ocurren cuando se hacen cambios contrapuestos en la misma línea de un archivo o cuando una persona edita un archivo y otra persona borra el mismo archivo.

# CAMBIO (CHANGE)

---

Un cambio (o diff, o delta) representa una modificación específica de un documento bajo el control de versiones. La granularidad de la modificación que es considerada como un cambio varía entre los sistemas de control de versiones.

# DESPLEGAR (CHECKOUT)

---

Es crear una copia de trabajo local desde el repositorio. Un usuario puede especificar una revisión en concreto u obtener la última. El término 'checkout' también se puede utilizar como un sustantivo para describir la copia de trabajo.

# CONFIRMAR (COMMIT)

---

Confirmar es escribir o mezclar los cambios realizados en la copia de trabajo del repositorio. Los términos 'commit' y 'checkin' también se pueden utilizar como sustantivos para describir la nueva revisión que se crea como resultado de confirmar.

# FUSIONAR, MEZCLAR (MERGE)

---

Una fusión o integración es una operación en la que se aplican dos tipos de cambios en un archivo o conjunto de archivos.

Un conjunto de archivos se bifurca, un problema que existía antes de la ramificación se trabaja en una nueva rama, y la solución se combina luego en la otra rama.

Se crea una rama, el código de los archivos es independiente editado, y la rama actualizada se incorpora más tarde en un único tronco unificado.

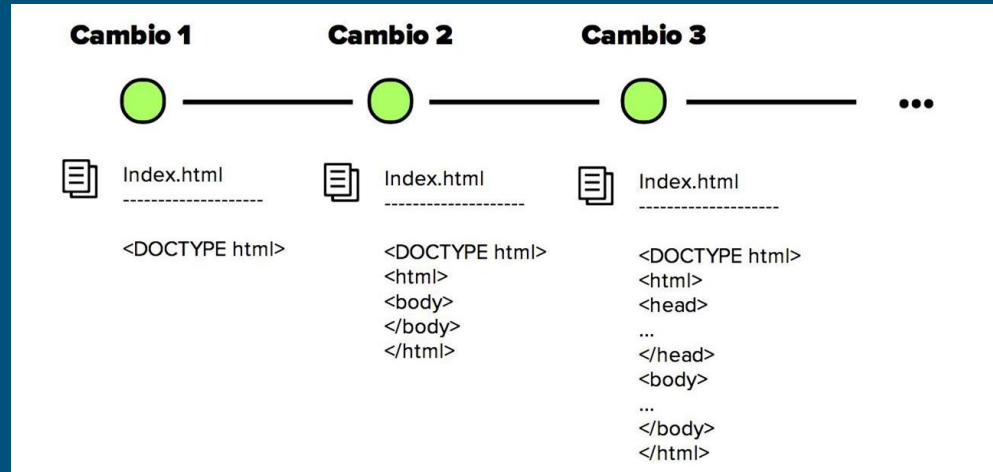
# Ventajas de los sistemas de control de versiones

Desarrollar software sin utilizar el control de versiones es arriesgado, equiparable a no tener copias de seguridad. El control de versiones también puede permitir que los desarrolladores se muevan más rápido y posibilita que los equipos de software mantengan la eficacia y la agilidad a medida que el equipo se escala para incluir más desarrolladores.

Las principales ventajas que deberías esperar del control de versiones son las siguientes.

# Un completo historial de cambios a largo plazo de todos los archivos.

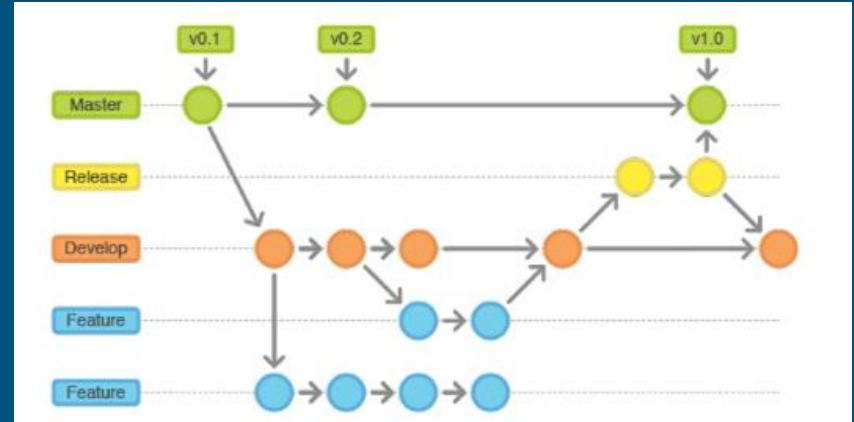
Tener el historial completo permite volver a las versiones anteriores para ayudar a analizar la causa raíz de los errores y es crucial cuando se tiene que solucionar problemas en las versiones anteriores del software.





# Creación de ramas y fusiones

La creación de una "rama" en las herramientas de VCS mantiene múltiples flujos de trabajo independientes los unos de los otros al tiempo que ofrece la facilidad de volver a fusionar ese trabajo, lo que permite que los desarrolladores verifiquen que los cambios de cada rama no entran en conflicto.



Changes <span>1</span>	History
Select branch to compare...	
Regla 3	
monica-261 • Jan 8, 2021	
Merge pull request #2131 from DiegoArmira/main	
Leonel Avila • Jan 8, 2021	
Merge pull request #2135 from Luiskr1993/patch-1	
ezequiel-brito • Jan 8, 2021	
Update Sintactico3D.py	
Luis Carlos Valiente Salazar • Jan 8, 2021	
Merge pull request #2134 from alerod620/main	
Alejandro • Jan 8, 2021	
Merge pull request #2132 from glendyco/patch-1	
ezequiel-brito • Jan 8, 2021	
Update sintactico3D.py	
Glendy Marilucy Contreras González • Jan 8, 2021	
Merge pull request #2130 from GermanM0000/patch-2	
ezequiel-brito • Jan 8, 2021	
Sintáctico 3d	
Augusto German Mazariegos Salguero • Jan 8, 2021	
Merge pull request #2120 from 3liezerSong/patch-11	

## Regla 3

monica-261 6f042146e ± 2 changed files

parser\fase2\team...\optimizacion.html

parser\fase2\team05...\optimizacion.py

		@@ -24,13 +24,94 @@
24	24	<th>Reglas</th>
25	25	</tr>
26	26	<tr>
27	-	<td>Regla 1:   t130 = b
28	- 	b = t130
29	-.  	Se optimiza por: t130 = b
	27	+<td>Regla 3: if t35=="tbProducto": goto .L0
	28	+  goto .L1
	29	+  label .L0
	30	+  se optimiza por:   if t35!="tbProducto": goto .L1
	31	+  t36=True
30	32	</td>
31	33	</tr> <tr>
32	-	<td>Regla 1:   t130 = b
33	- 	b = t130
34	-.  	Se optimiza por: t130 = b
	34	+<td>Regla 3: if t36 == True: goto .L3
	35	+  goto .L4
	36	+  label .L3
	37	+  se optimiza por:   if t36 != True: goto .L4
	38	+  t12 = "SELECT COUNT(*) from tbProducto;"
	39	+</td>
	40	+</tr> <tr>

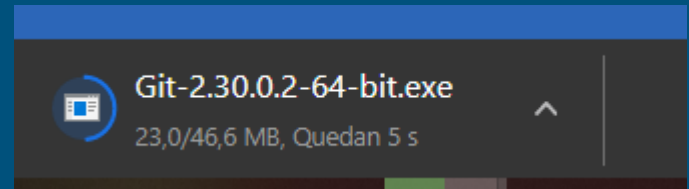
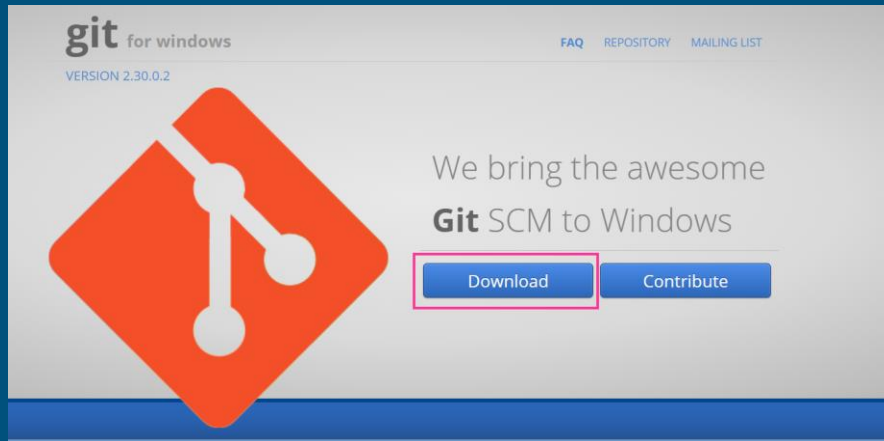
¿Qué es Git?

Git es un proyecto de código abierto maduro y con un mantenimiento activo que desarrolló originalmente Linus Torvalds, el famoso creador del kernel del sistema operativo Linux, en 2005. Un asombroso número de proyectos de software dependen de Git para el control de versiones, incluidos proyectos comerciales y de código abierto.



# INSTALACIÓN

Desde la página de Git, <https://gitforwindows.org/> , se descarga el instalador más reciente.



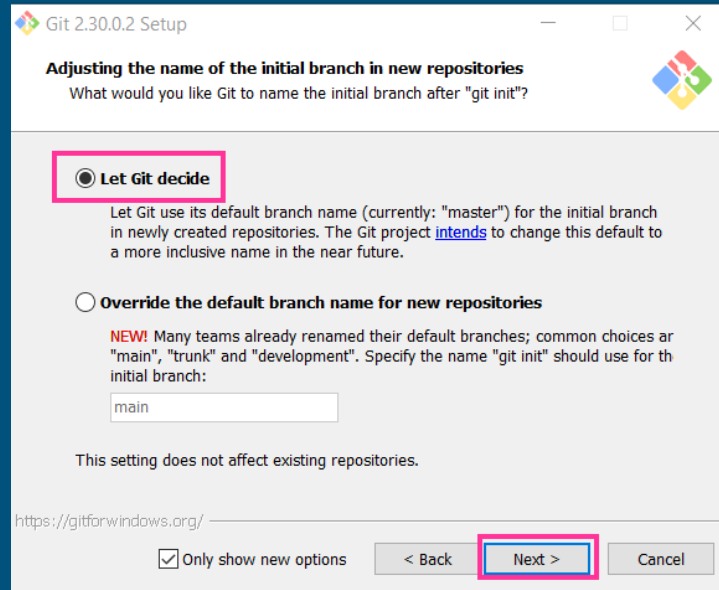
# INSTALACIÓN

Tras la descarga, se le dan permisos a la aplicación y muestra la pantalla de configuración



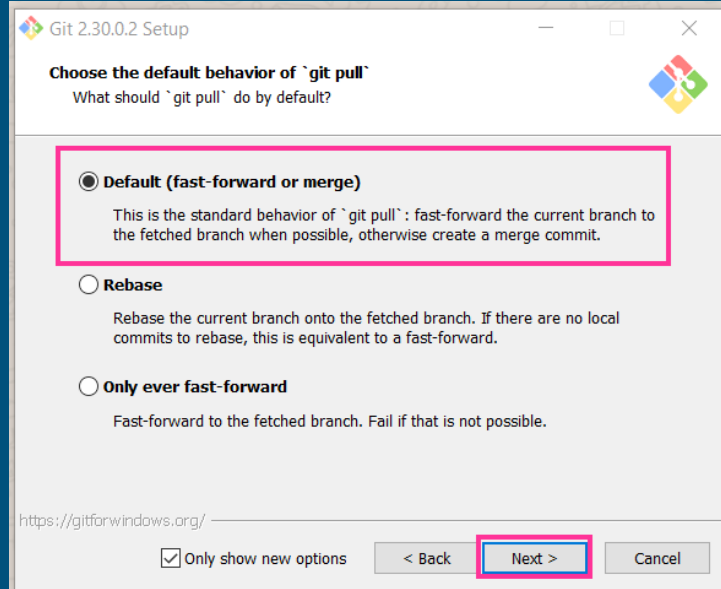
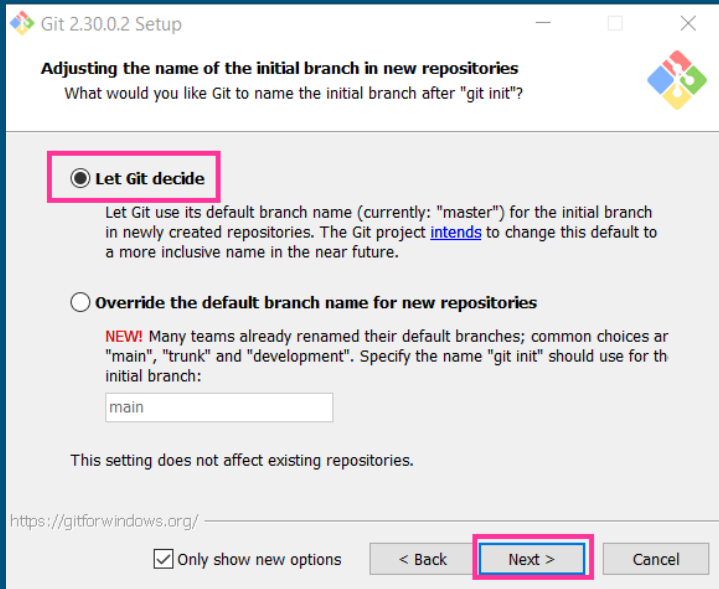
# INSTALACIÓN

La configuración será decidida por GitHub y se presiona siguiente.



# INSTALACIÓN

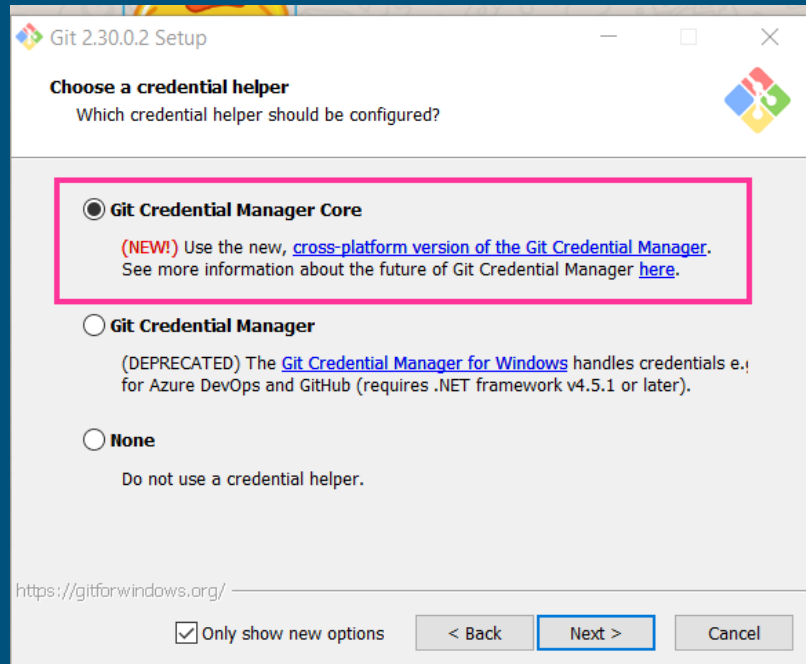
Como configuración se dejará la predeterminada





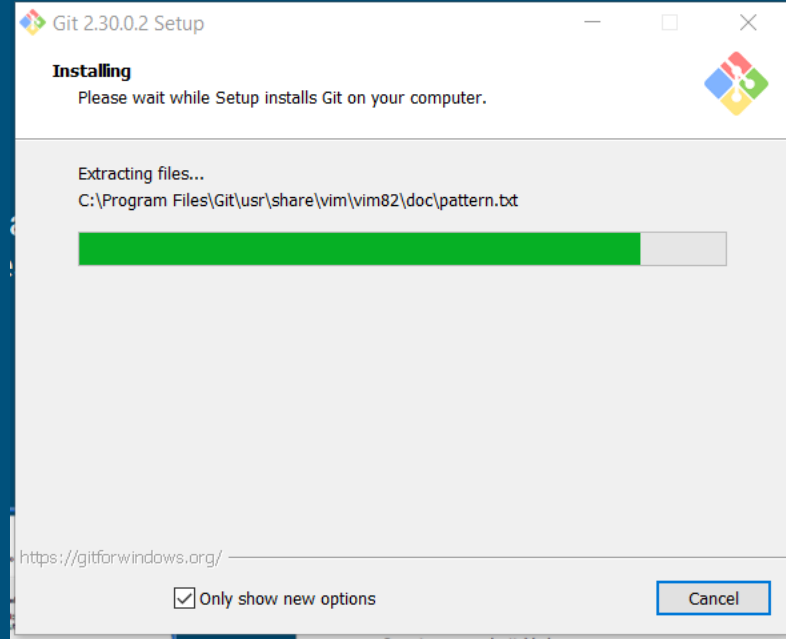
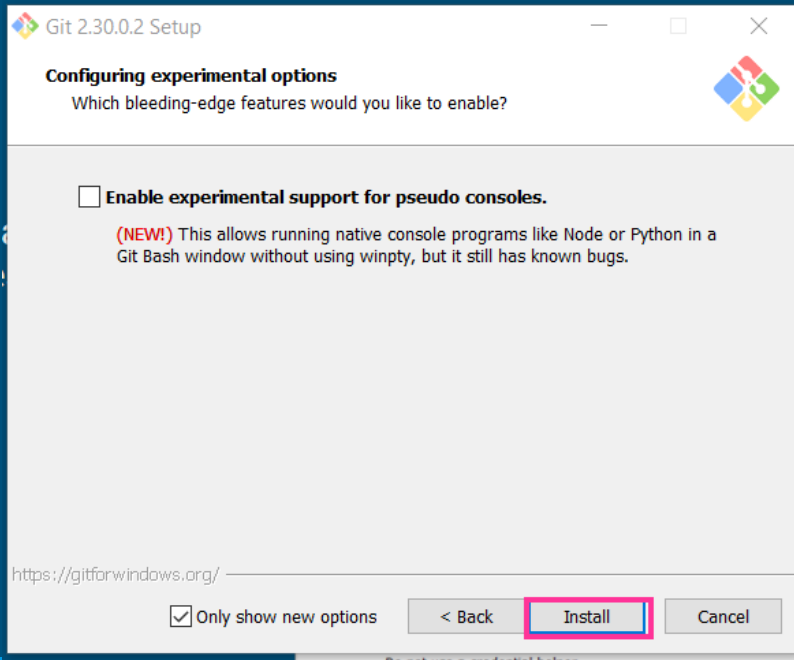
# INSTALACIÓN

Para la parte de credenciales, puede seleccionarse cualquiera de las tres opciones



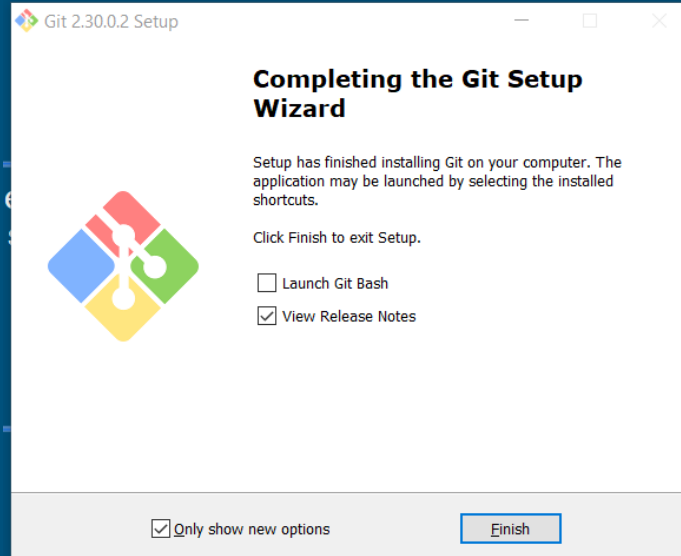
# INSTALACIÓN

Se puede activar soporte para pseudo consolas, en esta ocasión no se selecciona y se procede a instalar



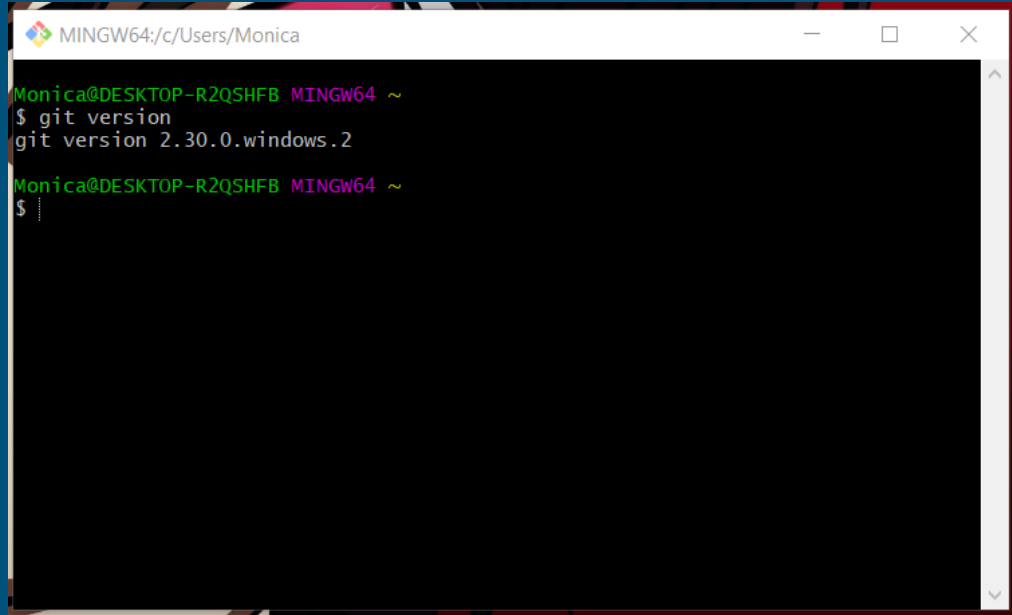
# INSTALACIÓN

Se finaliza la instalación y se puede escoger lanzar Git Bash automáticamente



# INSTALACIÓN

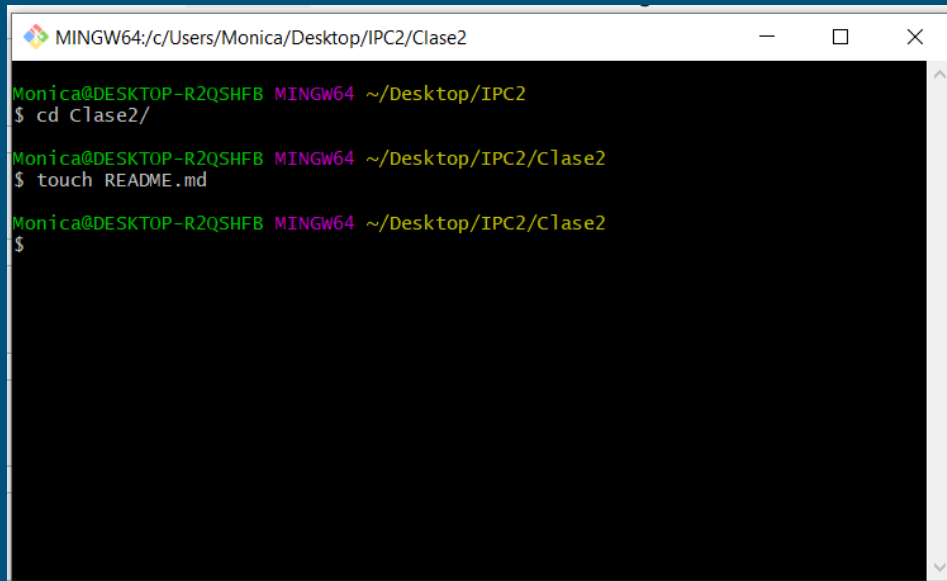
El instalador provee también la herramienta de Git Bash, se puede verificar la versión de git instalada a través del comando **git version**



```
MINGW64:/c/Users/Monica
Monica@DESKTOP-R2QSHFB MINGW64 ~
$ git version
git version 2.30.0.windows.2
Monica@DESKTOP-R2QSHFB MINGW64 ~
$
```

# CREANDO UN REPOSITORIO

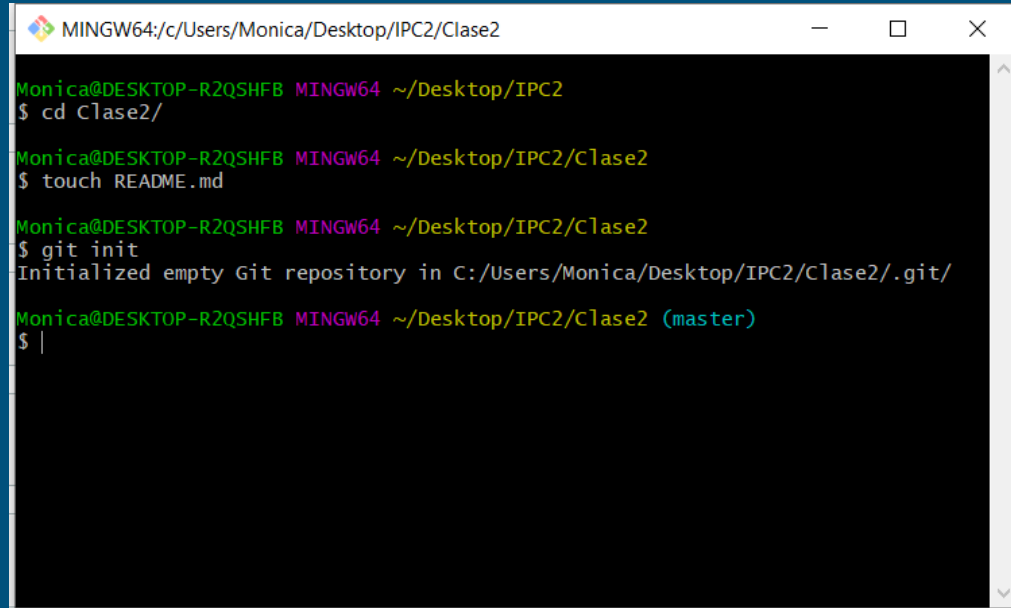
Desde la línea de comandos de Git Bash, se abre la carpeta la cual se versionara y en él, se crea el archivo **README.md**



```
MINGW64:/c/Users/Monica/Desktop/IPC2/Clase2
Monica@DESKTOP-R2QSHFB MINGW64 ~/Desktop/IPC2
$ cd clase2/
Monica@DESKTOP-R2QSHFB MINGW64 ~/Desktop/IPC2/Clase2
$ touch README.md
Monica@DESKTOP-R2QSHFB MINGW64 ~/Desktop/IPC2/Clase2
$
```

# CREANDO UN REPOSITORIO

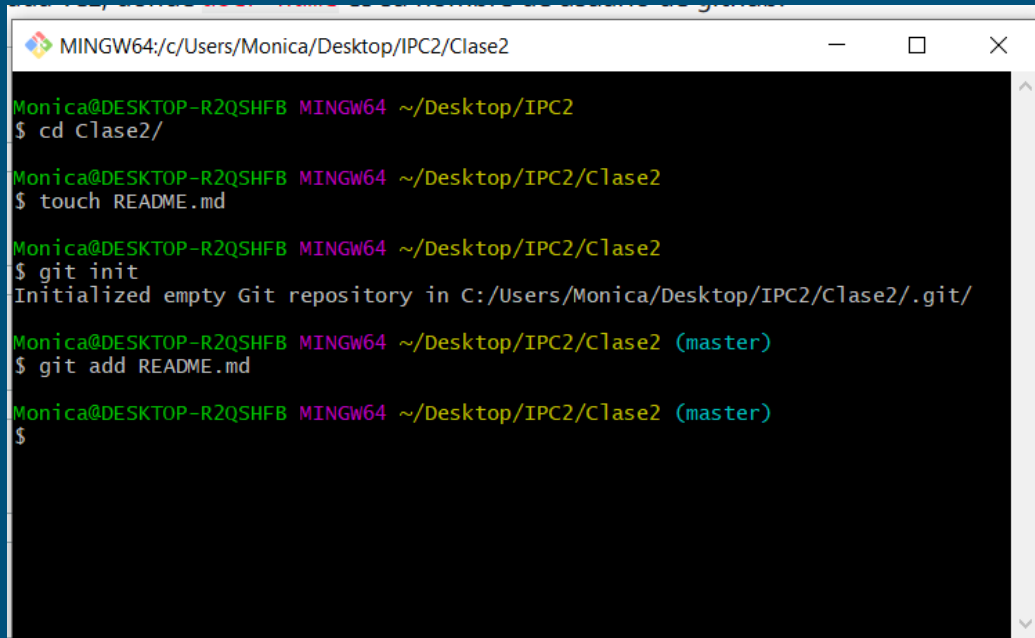
Se procede a inicializar el repositorio a través del comando **git init**



```
MINGW64:/c/Users/Monica/Desktop/IPC2/Clase2
Monica@DESKTOP-R2QSHFB MINGW64 ~/Desktop/IPC2
$ cd Clase2/
Monica@DESKTOP-R2QSHFB MINGW64 ~/Desktop/IPC2/Clase2
$ touch README.md
Monica@DESKTOP-R2QSHFB MINGW64 ~/Desktop/IPC2/Clase2
$ git init
Initialized empty Git repository in C:/Users/Monica/Desktop/IPC2/Clase2/.git/
Monica@DESKTOP-R2QSHFB MINGW64 ~/Desktop/IPC2/Clase2 (master)
$ |
```

# CREANDO UN REPOSITORIO

Se agrega al área de trabajo el archivo README.ME a través de **git add**



```
MINGW64:/c/Users/Monica/Desktop/IPC2/Clase2
Monica@DESKTOP-R2QSHFB MINGW64 ~/Desktop/IPC2
$ cd Clase2/

Monica@DESKTOP-R2QSHFB MINGW64 ~/Desktop/IPC2/Clase2
$ touch README.md

Monica@DESKTOP-R2QSHFB MINGW64 ~/Desktop/IPC2/Clase2
$ git init
Initialized empty Git repository in C:/Users/Monica/Desktop/IPC2/Clase2/.git/

Monica@DESKTOP-R2QSHFB MINGW64 ~/Desktop/IPC2/Clase2 (master)
$ git add README.md

Monica@DESKTOP-R2QSHFB MINGW64 ~/Desktop/IPC2/Clase2 (master)
$
```

# CREANDO UN REPOSITORIO

Se agrega un remoto nuevo

```
MINGW64:/c/Users/Monica/Desktop/IPC2/Clase2
Monica@DESKTOP-R2QSHFB MINGW64 ~/Desktop/IPC2
$ cd Clase2/

Monica@DESKTOP-R2QSHFB MINGW64 ~/Desktop/IPC2/Clase2
$ touch README.md

Monica@DESKTOP-R2QSHFB MINGW64 ~/Desktop/IPC2/Clase2
$ git init
Initialized empty Git repository in C:/Users/Monica/Desktop/IPC2/Clase2/.git/

Monica@DESKTOP-R2QSHFB MINGW64 ~/Desktop/IPC2/Clase2 (master)
$ git add README.md

Monica@DESKTOP-R2QSHFB MINGW64 ~/Desktop/IPC2/Clase2 (master)
$ git commit -m "Commit de prueba"
[master (root-commit) 26a82bf] Commit de prueba
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 README.md

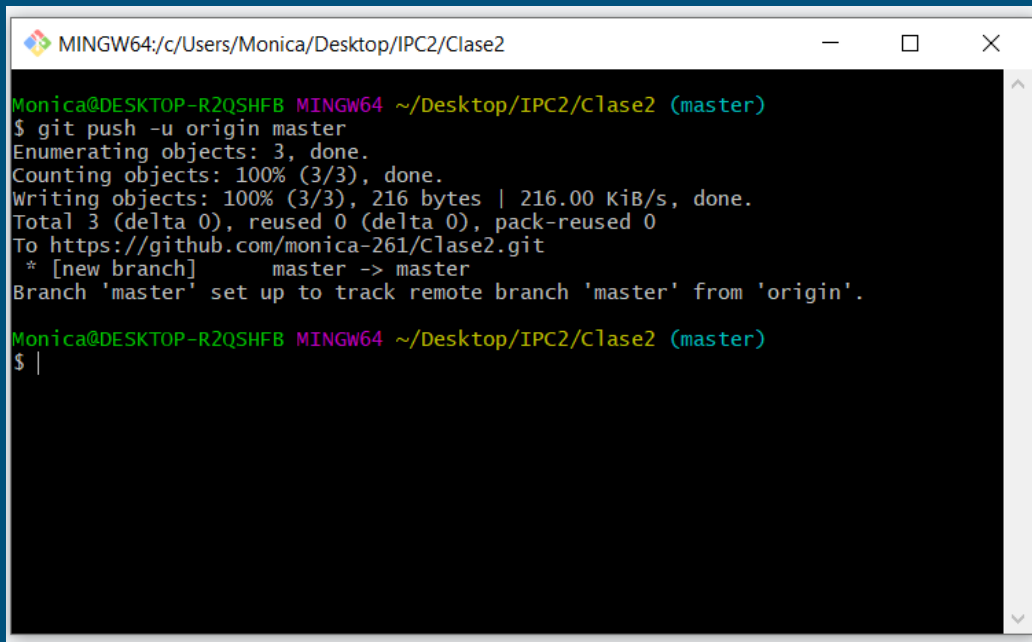
Monica@DESKTOP-R2QSHFB MINGW64 ~/Desktop/IPC2/Clase2 (master)
$ git remote add origin https://github.com/monica-261/Clase2.git

Monica@DESKTOP-R2QSHFB MINGW64 ~/Desktop/IPC2/Clase2 (master)
$
```



# CREANDO UN REPOSITORIO

Se hace push a la rama master del repositorio



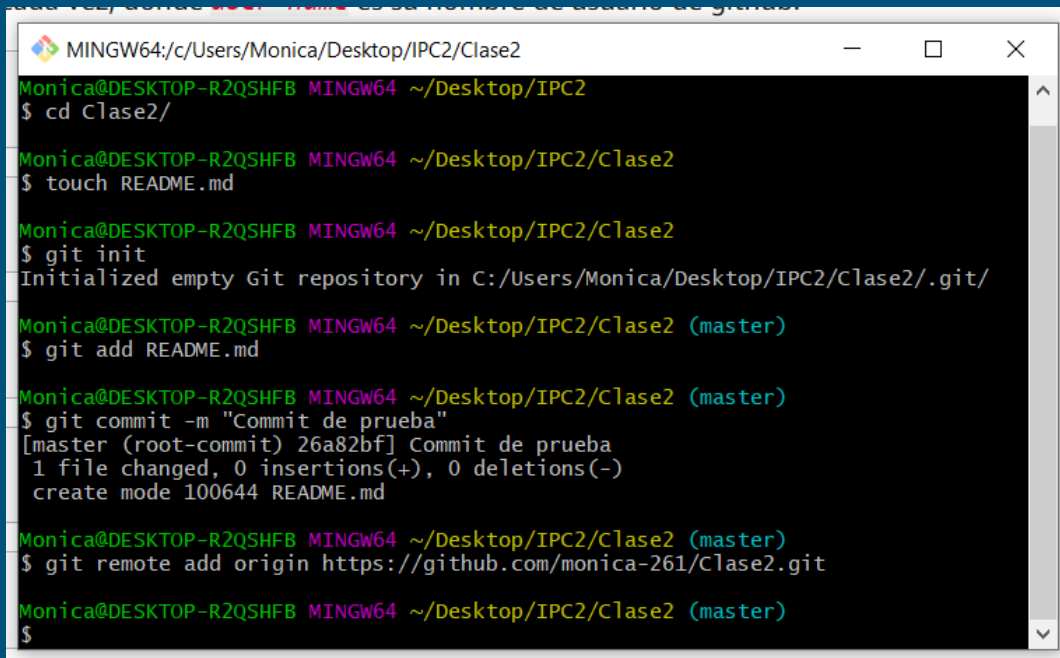
```
MINGW64:/c/Users/Monica/Desktop/IPC2/Clase2

Monica@DESKTOP-R2QSHFB MINGW64 ~/Desktop/IPC2/Clase2 (master)
$ git push -u origin master
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 216 bytes | 216.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/monica-261/Clase2.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.

Monica@DESKTOP-R2QSHFB MINGW64 ~/Desktop/IPC2/Clase2 (master)
$ |
```

# CREANDO UN REPOSITORIO

Se agrega un remoto nuevo



```
MINGW64:/c/Users/Monica/Desktop/IPC2/Clase2
Monica@DESKTOP-R2QSHFB MINGW64 ~/Desktop/IPC2
$ cd Clase2/

Monica@DESKTOP-R2QSHFB MINGW64 ~/Desktop/IPC2/Clase2
$ touch README.md

Monica@DESKTOP-R2QSHFB MINGW64 ~/Desktop/IPC2/Clase2
$ git init
Initialized empty Git repository in C:/Users/Monica/Desktop/IPC2/Clase2/.git/

Monica@DESKTOP-R2QSHFB MINGW64 ~/Desktop/IPC2/Clase2 (master)
$ git add README.md

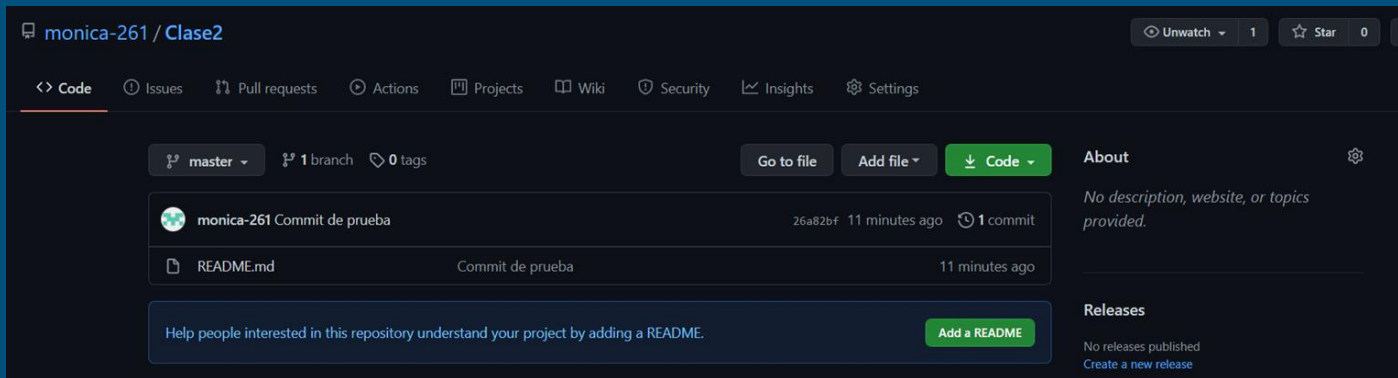
Monica@DESKTOP-R2QSHFB MINGW64 ~/Desktop/IPC2/Clase2 (master)
$ git commit -m "Commit de prueba"
[master (root-commit) 26a82bf] Commit de prueba
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 README.md

Monica@DESKTOP-R2QSHFB MINGW64 ~/Desktop/IPC2/Clase2 (master)
$ git remote add origin https://github.com/monica-261/Clase2.git

Monica@DESKTOP-R2QSHFB MINGW64 ~/Desktop/IPC2/Clase2 (master)
$
```

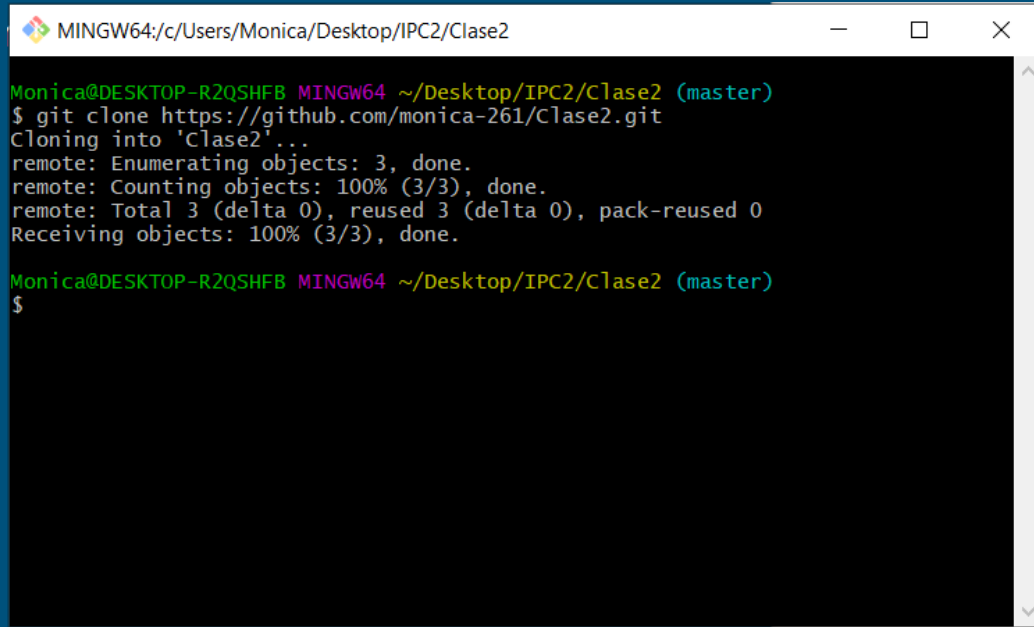
# CREANDO UN REPOSITORIO

El archivo README.md fue exitosamente agregado



# CLONANDO UN REPOSITORIO

Para clonar un repositorio, se usa el comando **git clone**

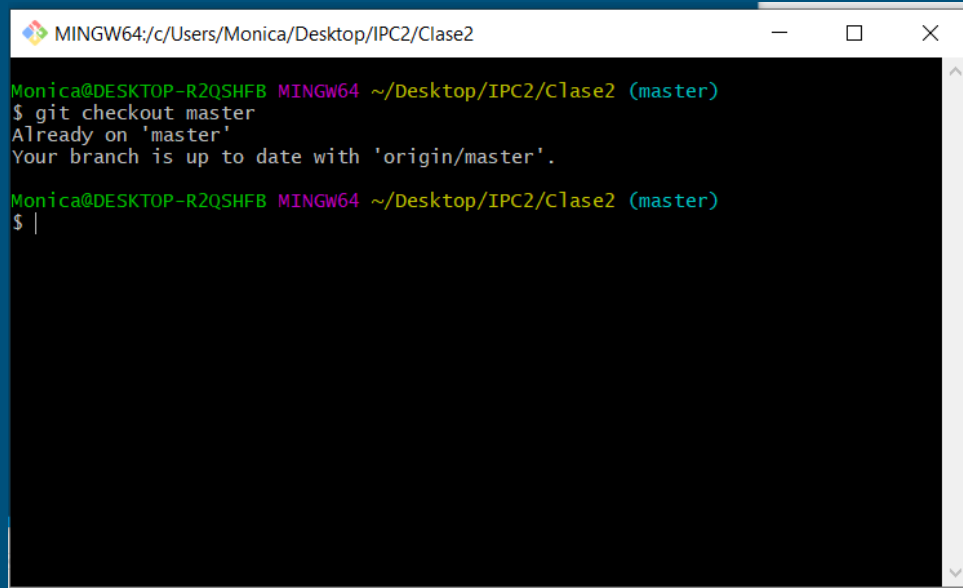
A screenshot of a Windows terminal window titled "MINGW64; c:/Users/Monica/Desktop/IPC2/Clase2". The terminal shows the execution of the "git clone" command to clone a repository from GitHub. The output indicates that the repository was successfully cloned into a directory named "Clase2".

```
MINGW64; c:/Users/Monica/Desktop/IPC2/Clase2
Monica@DESKTOP-R2QSHFB MINGW64 ~/Desktop/IPC2/Clase2 (master)
$ git clone https://github.com/monica-261/Clase2.git
Cloning into 'Clase2'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.

Monica@DESKTOP-R2QSHFB MINGW64 ~/Desktop/IPC2/Clase2 (master)
$
```

# CREANDO UNA RAMA

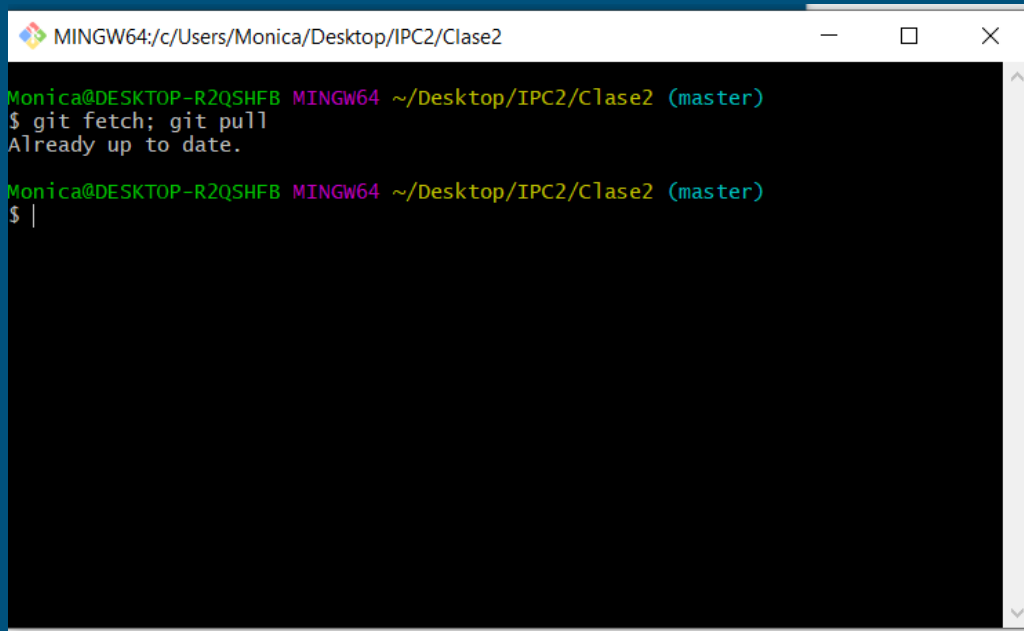
Para crear una rama, primero hay que posicionarse en la rama master, dado sea el caso que ya se esté ahí, se observa un mensaje



```
MINGW64:/c/Users/Monica/Desktop/IPC2/Clase2
Monica@DESKTOP-R2QSHFB MINGW64 ~/Desktop/IPC2/Clase2 (master)
$ git checkout master
Already on 'master'
Your branch is up to date with 'origin/master'.
Monica@DESKTOP-R2QSHFB MINGW64 ~/Desktop/IPC2/Clase2 (master)
$ |
```

# CREANDO UNA RAMA

Actualizamos la rama al 100%

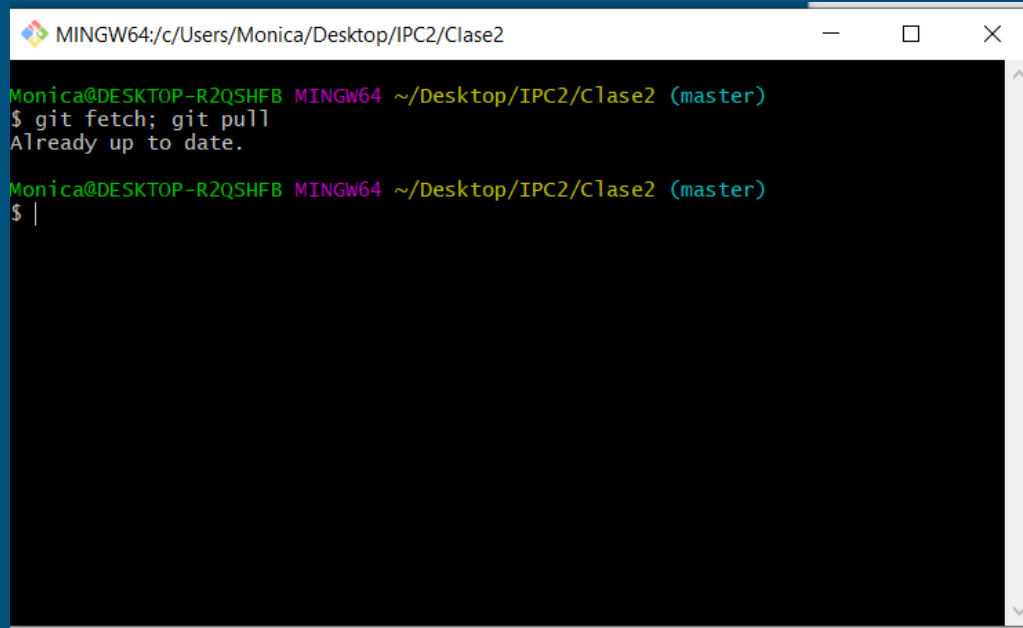
A screenshot of a MINGW64 terminal window. The title bar shows the path 'MINGW64:/c/Users/Monica/Desktop/IPC2/Clase2'. The terminal content shows a user named Monica at a desktop named R2QSHFB, in a directory ~/Desktop/IPC2/Clase2, on the master branch. The user enters the command '\$ git fetch; git pull', and the terminal responds with 'Already up to date.' followed by a new prompt '\$ |'.

```
MINGW64:/c/Users/Monica/Desktop/IPC2/Clase2
Monica@DESKTOP-R2QSHFB MINGW64 ~/Desktop/IPC2/Clase2 (master)
$ git fetch; git pull
Already up to date.

Monica@DESKTOP-R2QSHFB MINGW64 ~/Desktop/IPC2/Clase2 (master)
$ |
```

# CREANDO UNA RAMA

Se procede a actualizar la rama master.

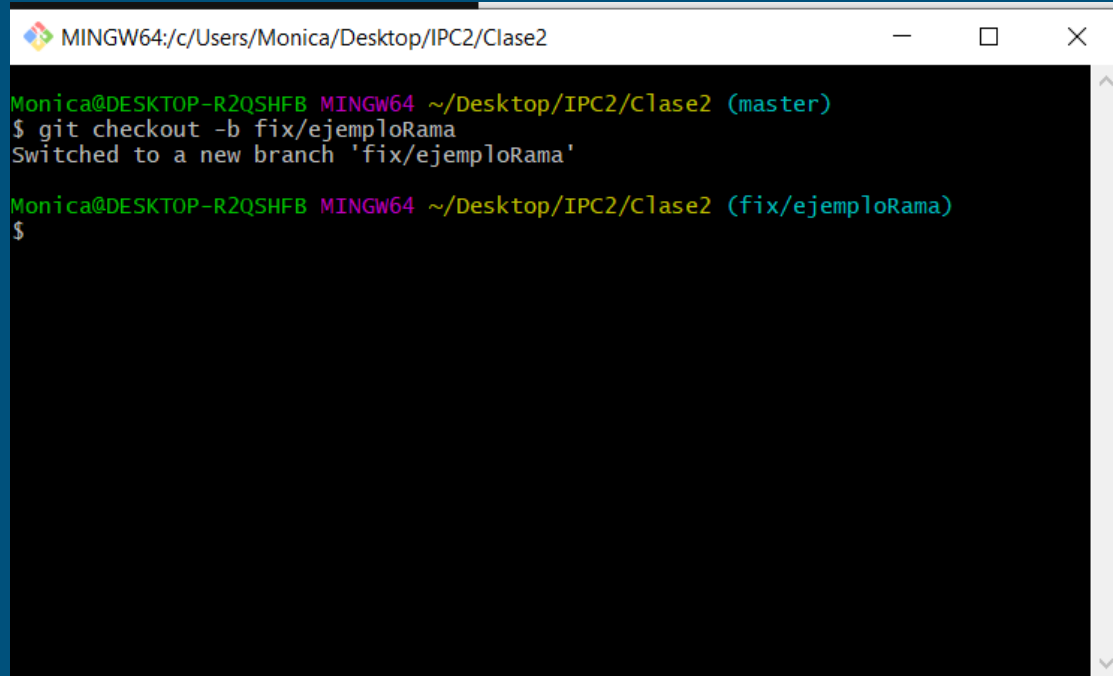


```
MINGW64:/c/Users/Monica/Desktop/IPC2/Clase2
Monica@DESKTOP-R2QSHFB MINGW64 ~/Desktop/IPC2/Clase2 (master)
$ git fetch; git pull
Already up to date.

Monica@DESKTOP-R2QSHFB MINGW64 ~/Desktop/IPC2/Clase2 (master)
$ |
```

# CREANDO UNA RAMA

Se crea una nueva rama a través del comando checkout



```
MINGW64:/c/Users/Monica/Desktop/IPC2/Clase2

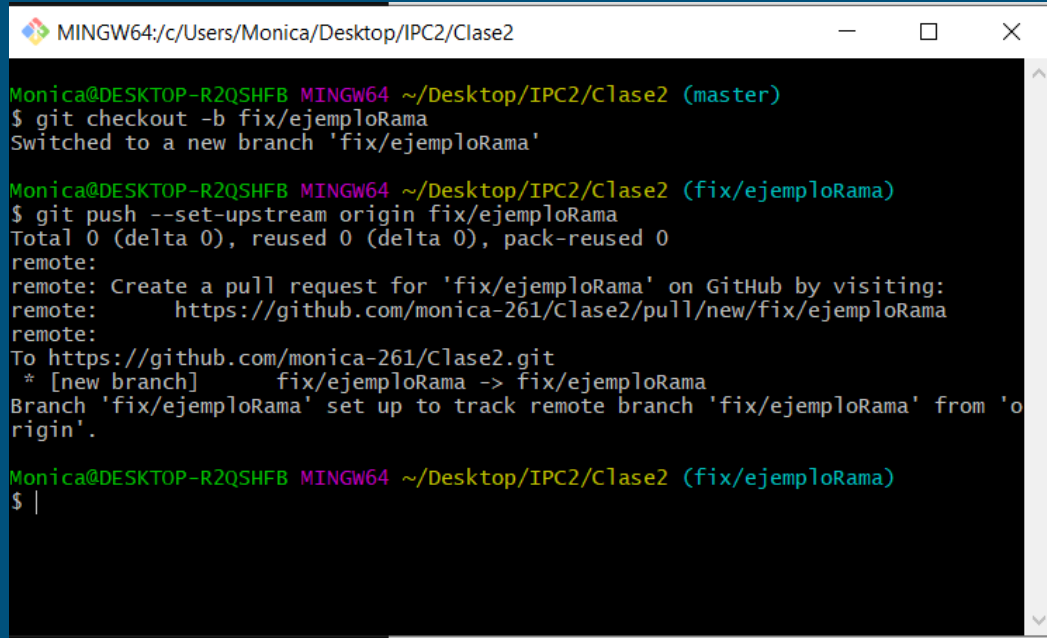
Monica@DESKTOP-R2QSHFB MINGW64 ~/Desktop/IPC2/Clase2 (master)
$ git checkout -b fix/ejemploRama
Switched to a new branch 'fix/ejemploRama'

Monica@DESKTOP-R2QSHFB MINGW64 ~/Desktop/IPC2/Clase2 (fix/ejemploRama)
$
```



# CREANDO UNA RAMA

Se envía la rama al repositorio remoto

A terminal window titled 'MINGW64:/c/Users/Monica/Desktop/IPC2/Clase2' with standard window controls. The terminal shows a sequence of Git commands and their output. The user is in the 'master' branch and creates a new branch 'fix/ejemploRama'. Then, they push the new branch to the remote repository. The output shows the push was successful and provides instructions on how to create a pull request on GitHub.

```
MINGW64:/c/Users/Monica/Desktop/IPC2/Clase2

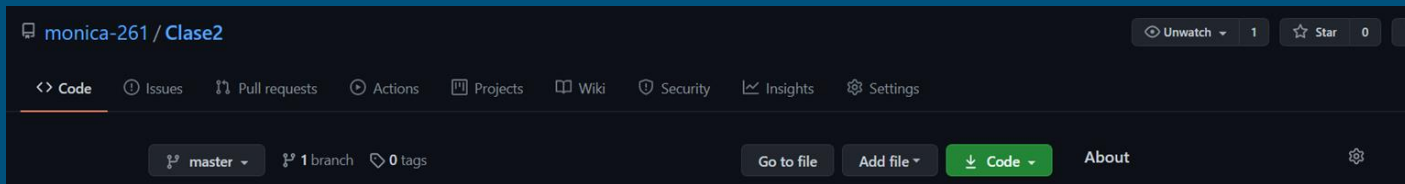
Monica@DESKTOP-R2QSHFB MINGW64 ~/Desktop/IPC2/Clase2 (master)
$ git checkout -b fix/ejemploRama
Switched to a new branch 'fix/ejemploRama'

Monica@DESKTOP-R2QSHFB MINGW64 ~/Desktop/IPC2/Clase2 (fix/ejemploRama)
$ git push --set-upstream origin fix/ejemploRama
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'fix/ejemploRama' on GitHub by visiting:
remote:   https://github.com/monica-261/Clase2/pull/new/fix/ejemploRama
remote:
To https://github.com/monica-261/Clase2.git
 * [new branch]      fix/ejemploRama -> fix/ejemploRama
Branch 'fix/ejemploRama' set up to track remote branch 'fix/ejemploRama' from 'origin'.

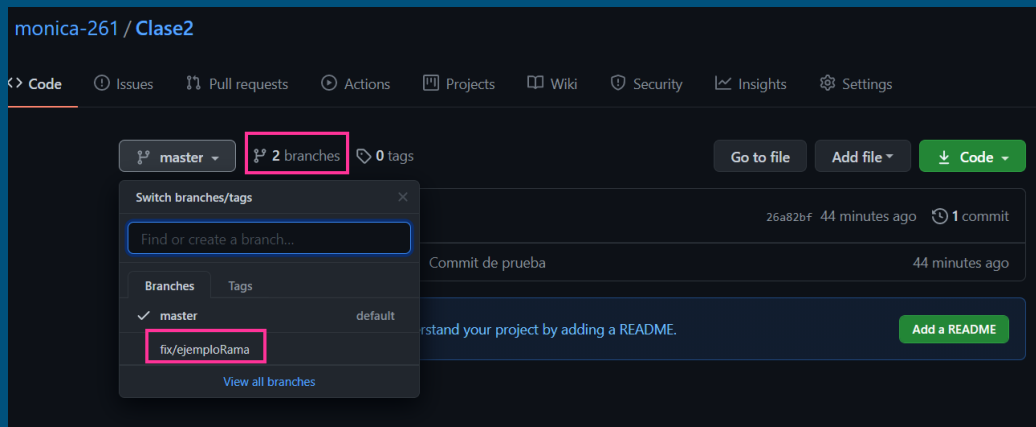
Monica@DESKTOP-R2QSHFB MINGW64 ~/Desktop/IPC2/Clase2 (fix/ejemploRama)
$ |
```

# CREANDO UNA RAMA

Se observa que al inicio, el repositorio contaba únicamente con una branch



Mientras que, tras realizar el pull de la branch, el repositorio ya cuenta con dos y se observa el nombre que se le indico en la creación



# ¿Qué es XML?

---

# XML

---

Es un metalenguaje que fue diseñado para estructurar, almacenar e intercambiar datos entre aplicaciones. Es un estándar ya que es extensible y puede ser utilizado independientemente de la plataforma.

Es el acrónimo de Extensible Markup Language. Es un lenguaje de marcado que define un conjunto de reglas para la codificación de documentos.

Es un metalenguaje que permite definir lenguajes de marcas desarrollado por el World Wide Web Consortium (W3C) utilizado para almacenar datos en forma legible.

# ¿Para qué sirve?

---

Brindar una estructura a la información que se encuentra en la web, para que la información pueda ser:

- Almacenada
- Procesada
- Visualizada
- Manipulada

Por aplicaciones, páginas o lo que lo requiera

# Partes de un XML

```
<?xml version="1.0" encoding="WINDOWS-1252" standalone="yes"?>
```

1

```
<empleados>
  <empleado>
    <empleado_id>120</empleado_id>
    <apellido>Gonzales</apellido>
    <salario>1800</salario>
  </empleado>
  <empleado>
    <empleado_id>121</empleado_id>
    <apellido>Garcia</apellido>
    <salario>1600</salario>
  </empleado>
</empleados>
```

2

```
<?gifPlayer size="100,300" ?>
```

3

- 1.- Prologo.
- 2.- Elementos o Elemento Raíz
- 3.- Epilogo (opcional)

# Partes de un XML

---

Prólogo: En este se especifica que el documento en cuestión es XML, se define la versión y la codificación de caracteres que se utilizó en el archivo.

Elementos: Se tiene una etiqueta o tag inicial (<ejemplo>) y una final (</ejemplo>).

Epílogo: Datos adicionales, como encriptación. Generalmente los tokens son agregados en este apartado.

# Estructura de una Etiqueta

---





# Características

---

- Extensible: Cada persona puede crear las etiquetas que desee según el tipo de documento que se quiera crear. En otras palabras cada usuario podrá crear su propio lenguaje.
- Estándar abierto: XML es SGML, lo que significa que no es necesario saber programar, además de que existen muchas herramientas para su manipulación.

# Características

---

- Eficiente: La información debe ser transmitida una única vez para su manipulación
- Libre: Nadie tiene una patente sobre él, ya que ha sido definido como un estándar internacional.
- Manejable: Tiene métodos para declarar y reforzar las estructuras que son utilizadas actualmente.

# Ventajas

---

- Simplifica el intercambio de datos: Dada la gran cantidad de aplicaciones que se tiene en la web, los desarrolladores pueden utilizar el mismo estándar para comunicar aplicaciones que a simple vista serían incompatibles.

Los archivos XML se almacenan en forma de texto, lo que facilita la expansión y actualización de los mismos.

# Ventajas

---

- Aumenta la disponibilidad de datos: El acceso a los archivos XML pueden darse de diferentes lugares, no únicamente desde HTML.

Con XML se tiene la opción de que los datos estén disponibles para todos los tipos de máquinas de lectura.

# Ventajas

---

- Entendible: La estructura del archivo
- Estandarizado: Está regulado internacionalmente, con lo que ya se tiene un formato definido para trabajar.
- Presentación: Los datos pueden ser mostrados por medio de HTML

# Aplicaciones

---

- Estructura del HTML,
- PDF está guardado en XML
- Microsoft Office guarda los datos como XML
- SOAP (Simple Object Access Protocol)
- Web Services de instituciones financieras
- Correos electrónicos

# Lectura

---

# Árbol

---

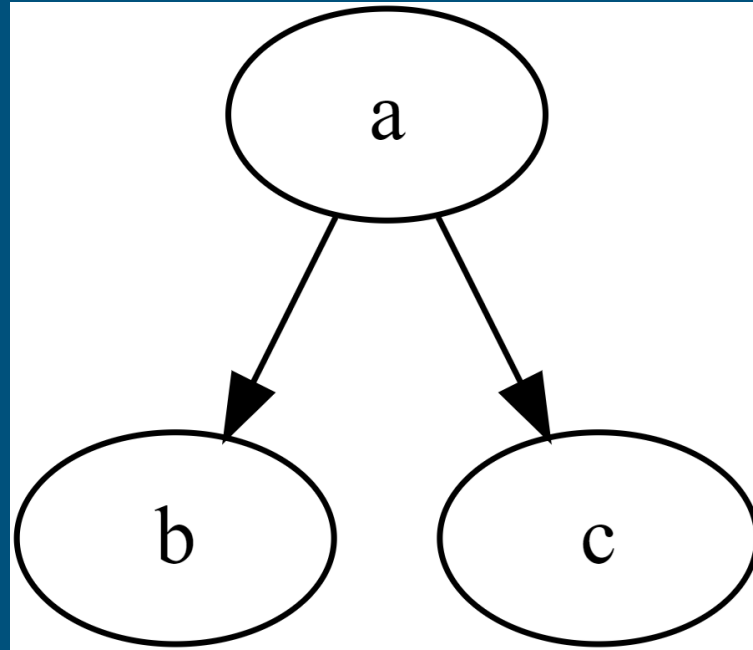
Un árbol es una estructura jerárquica o secuencial formada por nodos, los cuales pueden ser de dos tipos: los nodos hijos y los nodos padre.

- Un padre puede tener muchos hijos
- Un hijo puede tener sus propios hijos, además de tener hermanos



Ejemplo:

---

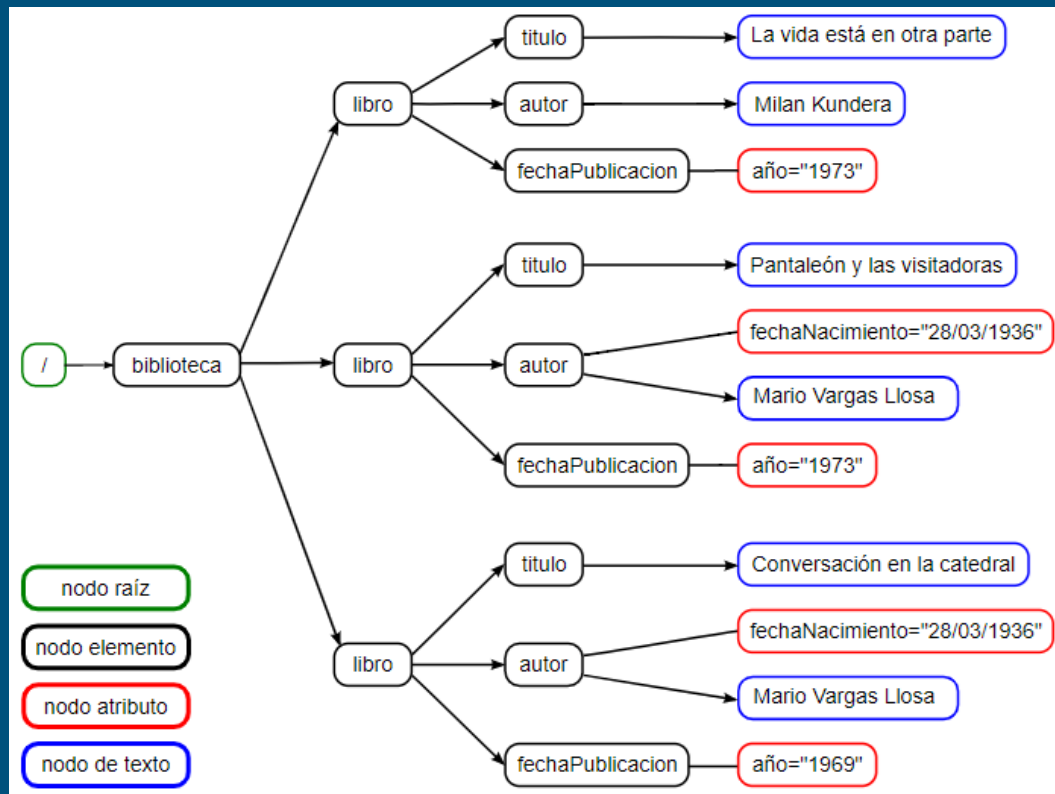


# Lectura de XML

---

```
<?xml version="1.0" encoding="UTF-8"?>
<biblioteca>
  <libro>
    <titulo>La vida está en otra parte</titulo>
    <autor>Milan Kundera</autor>
    <fechaPublicacion año="1973"/>
  </libro>
  <libro>
    <titulo>Pantaleón y las visitadoras</titulo>
    <autor fechaNacimiento="28/03/1936">Mario Vargas Llosa</autor>
    <fechaPublicacion año="1973"/>
  </libro>
  <libro>
    <titulo>Conversación en la catedral</titulo>
    <autor fechaNacimiento="28/03/1936">Mario Vargas Llosa</autor>
    <fechaPublicacion año="1969"/>
  </libro>
</biblioteca>
```

# Árbol resultante



# Escritura

---

Class3
attr : float

Class

Attribute

```
<xs:element name="Class3" type="Class3"/>
<xs:complexType name="Class3">
  <xs:all>
    <xs:element name="Class3.attr" type="xs:float" />
  </xs:all>
</xs:complexType>
```

# Procesamiento con el modelo XPath

---

# XPATH

---

Es un lenguaje que se utiliza para acceder a elementos o atributos de un documento XML

# XPATH

---

```
<?xml version="1.0" encoding="UTF-8"?>

<bookstore>

  <book>
    <title lang="en">Harry Potter</title>
    <price>29.99</price>
  </book>

  <book>
    <title lang="en">Learning XML</title>
    <price>39.95</price>
  </book>

</bookstore>
```

# XPATH - nodeName

---

El comando nodeName se utiliza para seleccionar todos los nodos que tengan en el nombre de la etiqueta el que se indique.

Ej:

bookstore

```
<bookstore>

<book>
  <title lang="en">Harry Potter</title>
  <price>29.99</price>
</book>

<book>
  <title lang="en">Learning XML</title>
  <price>39.95</price>
</book>

</bookstore>
```



# XPATH - Root ( / )

---

El comando root se utiliza para seleccionar el elemento raíz con el nombre especificado.

Ej:

/bookstore

```
<bookstore>

<book>
  <title lang="en">Harry Potter</title>
  <price>29.99</price>
</book>

<book>
  <title lang="en">Learning XML</title>
  <price>39.95</price>
</book>

</bookstore>
```

# XPATH - Node ( // )

---

El comando node se utiliza para seleccionar todos los nodos con el nombre indicado.

Ej:

//book

```
<book>
  <title lang="en">Harry Potter</title>
  <price>29.99</price>
</book>
```

```
<book>
  <title lang="en">Learning XML</title>
  <price>39.95</price>
</book>
```

# XPATH - Atributo ( @ )

---

El comando atributo se utiliza para seleccionar todos los atributos con el nombre indicado.

Ej:

@lang



```
lang="en">
```

# XPATH - Combinación NodeName + Root

---

bookstore/book

```
<book>
  <title lang="en">Harry Potter</title>
  <price>29.99</price>
</book>

<book>
  <title lang="en">Learning XML</title>
  <price>39.95</price>
</book>
```

# XPATH - Combinación nodeName + Node

---

bookstore//book

Aclaración, en caso de que la etiqueta book tenga otro hijo book, devolvería 3 nodos.

```
<book>
  <title lang="en">Harry Potter</title>
  <price>29.99</price>
</book>

<book>
  <title lang="en">Learning XML</title>
  <price>39.95</price>
</book>
```

# XPATH - Combinación Node + Atributo

---

//@lang

```
lang="en">  
lang="en">
```

# ¿Dudas?

---