

SISTEMA DE CONTROL DE VERSIONES - SCV

IEO. Ph.D. VERA ZASULICH PÉREZ ARIZA

IEO. HERNÁN DARÍO PATARROYO SERRANO

I. INTRODUCCIÓN

¿Qué es un Sistema de Control de Versiones – SCV?

¿Nunca has deseado poder volver a una versión previa de un proyecto o registrar los cambios que has hecho?



Archivo_v1
Archivo_v2
...
Archivo_vN
Archivo_casi_completo
Archivo_final
Archivo_final_final

Permíteme decirte que esto es posible con un **SISTEMA DE CONTROL DE VERSIONES (CONCURRENT VERSIONS SYSTEM)**

Con un SCV se puede mantener un registro de los cambios que has hecho en tus archivos (proyecto) y puedes tener varias versiones de tu proyecto en la misma computadora de forma simultánea. Esto te permite cambiar de una versión a otra de tu proyecto mientras haces revisiones, cambios, pruebas y actualizaciones.



¿Qué es un Sistema de Control de Versiones – SCV?

Un sistema de control de versiones permite:

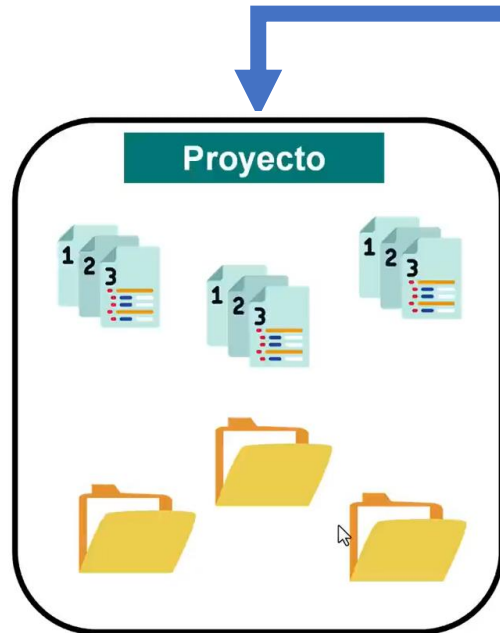
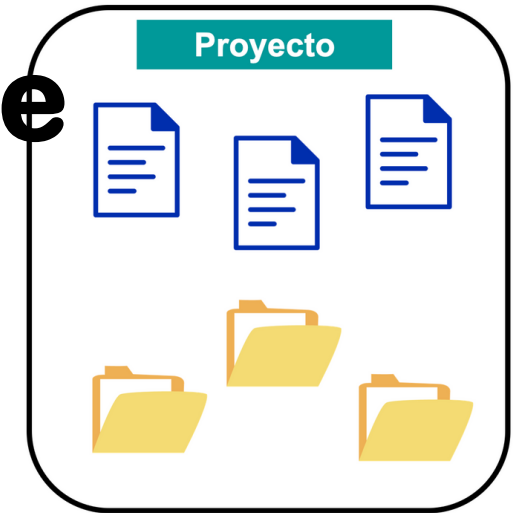
- ✓ Guarda el registro histórico de las modificaciones realizadas a cada archivo y/o carpeta.
- ✓ Añade trazabilidad al desarrollo de software. Por lo tanto, permite identificar cambios entre una versión y otra.
- ✓ Colaboración de varios desarrolladores en el mismo proyecto.
- ✓ Ayudan a gestionar y unir los diferentes archivos del proyecto.
- ✓ Evita que pierdas información que por error se haya modificado o borrado accidentalmente.
- ✓ Es posible revertir o deshacer los cambios realizados en el archivo.



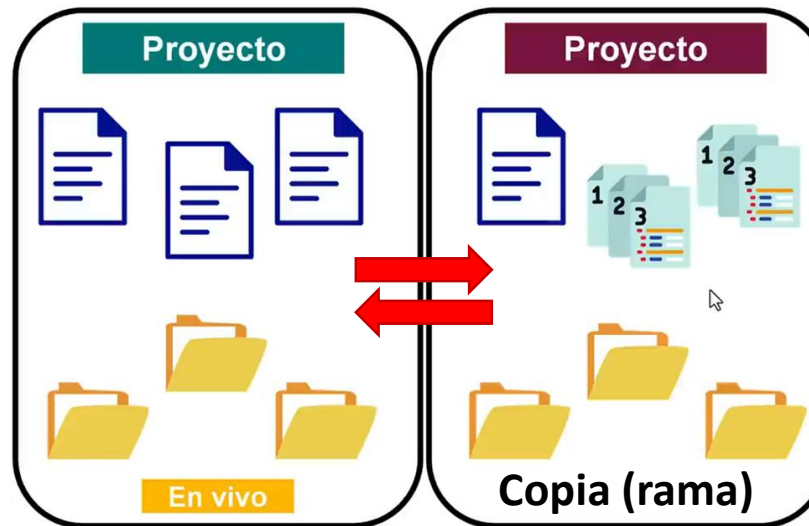
¿Qué es un Sistema de Control de Versiones – SCV?



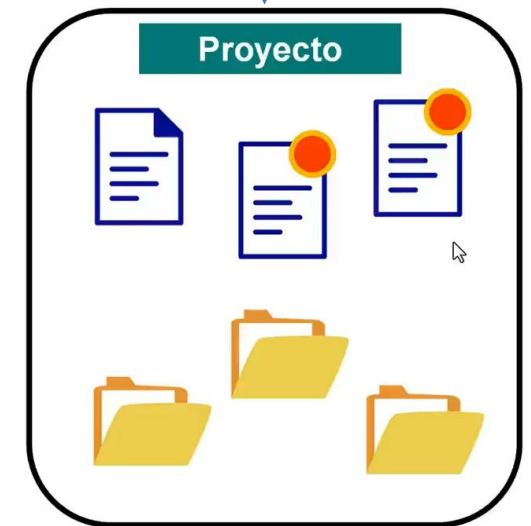
git



Permite rastrear las versiones para cada uno de los archivos, registrando los cambios hechos

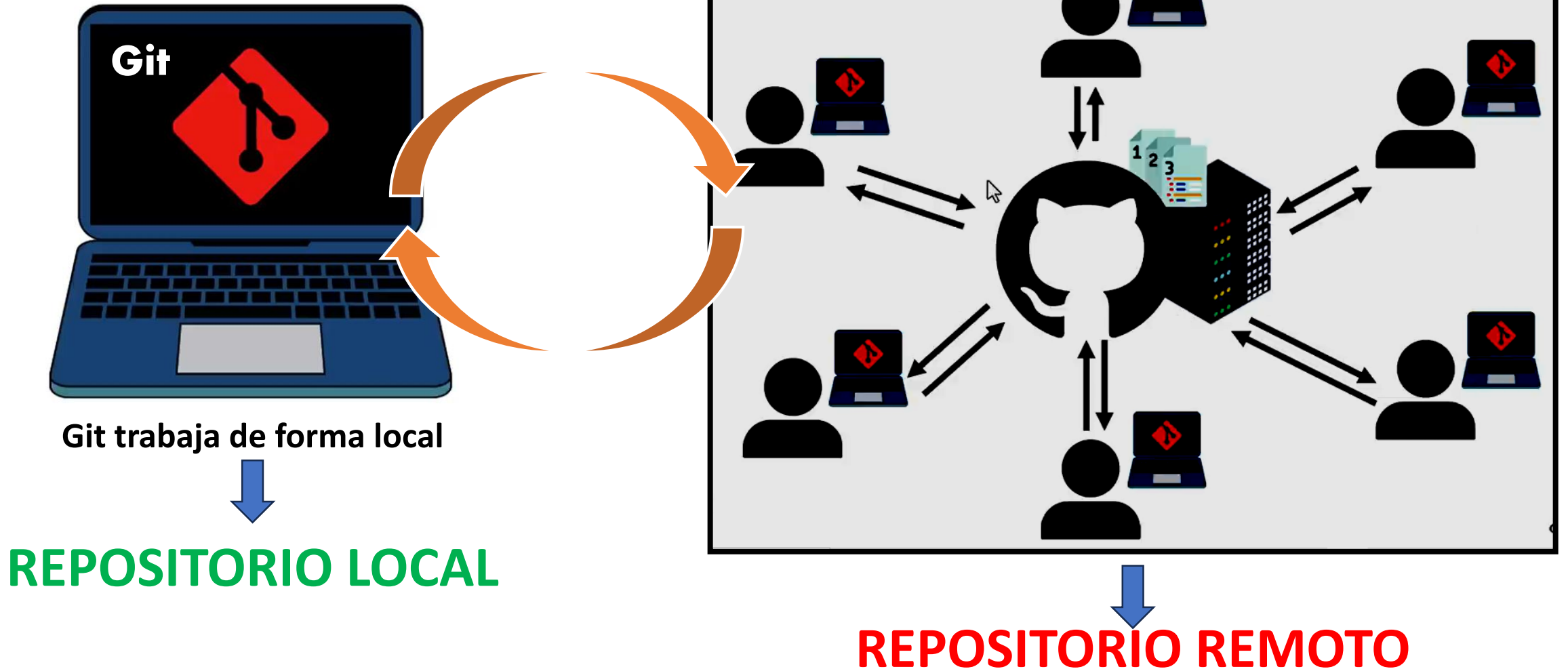


Crear copias en paralelo (ramas) para hacer modificaciones (historial) para luego integrar o fusionar al proyecto en vivo

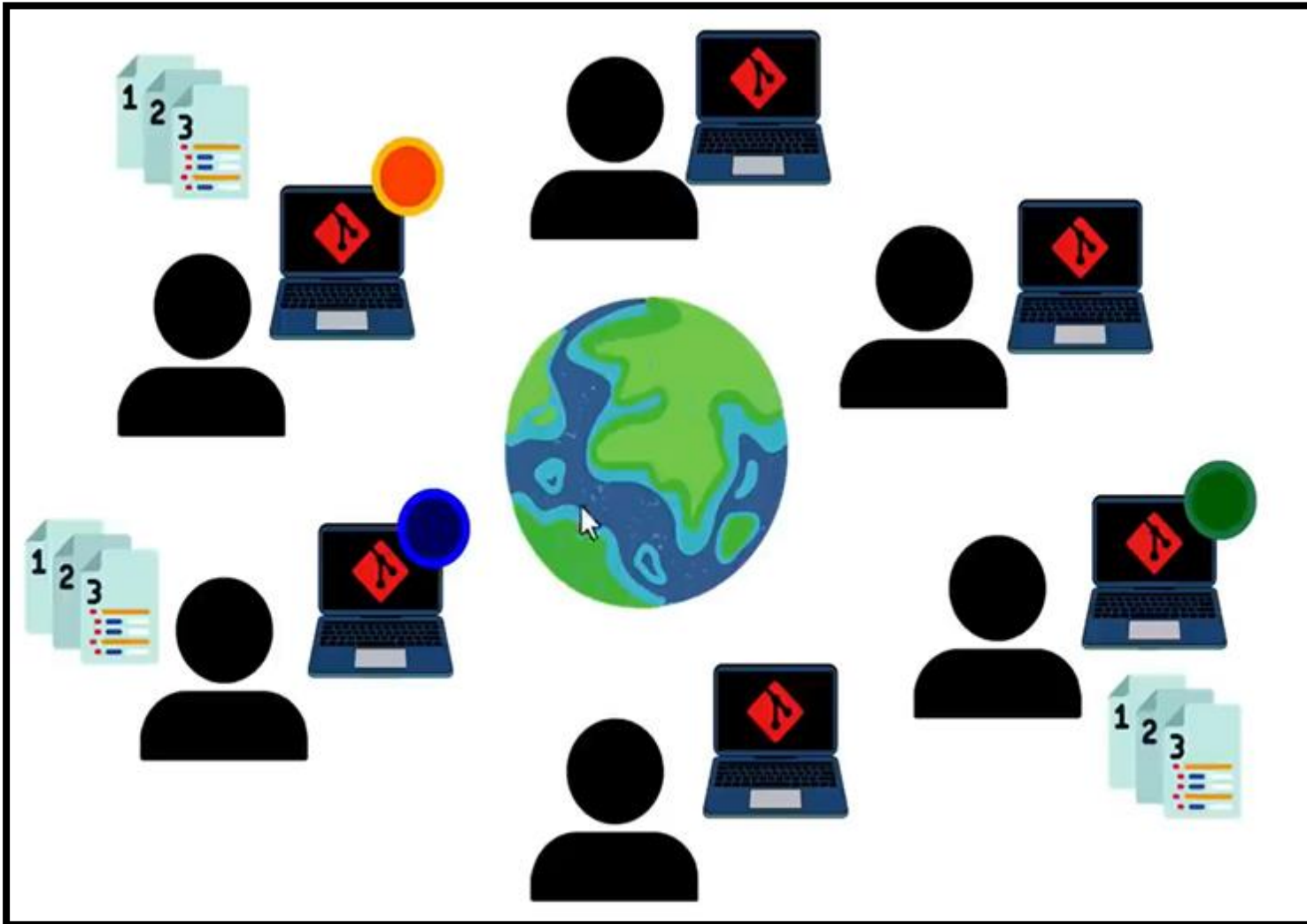


Registrar cambios en archivos específicos sin necesidad de generar una copia de todo el proyecto

¿Qué es un Sistema de Control de Versiones – SCV?



¿Qué es un Sistema de Control de Versiones – SCV?



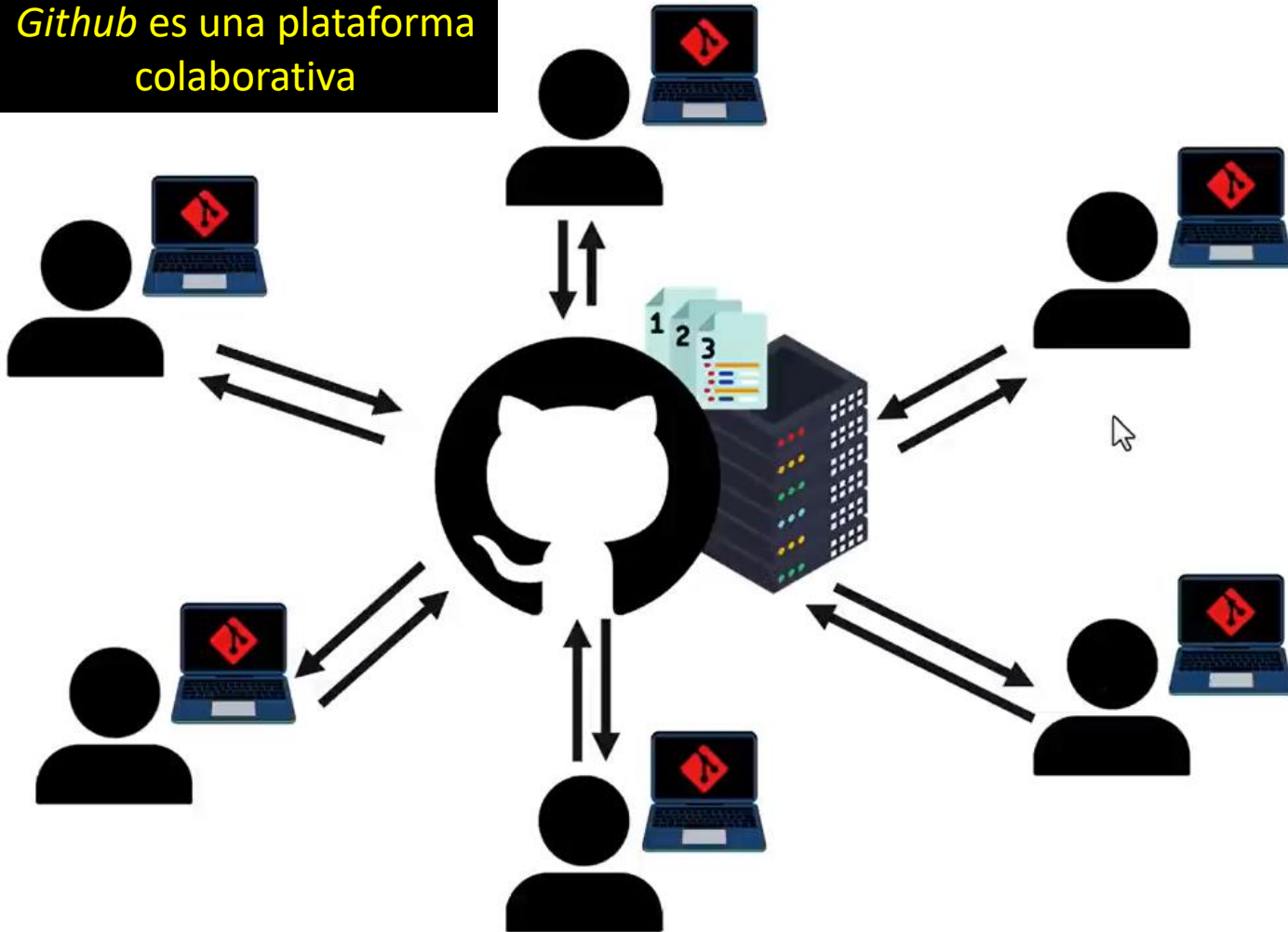
Problemas:

- Cada uno tiene una copia del repositorio inicial.
- Ahora cada programador tiene su “nueva versión”, la cual es local y es una copia aislada.
- Hay incompatibilidad entre los cambios



¿Qué es un Sistema de Control de Versiones – SCV?

Github es una plataforma colaborativa



Solución:

- Usar un repositorio remoto.
- Cada uno tiene una copia del repositorio inicial de una ÚNICA FUENTE.
- Permite enviar los cambios desarrollados y obtener los cambios de otros programadores de forma casi inmediata.
- Permite asegurar las copias del repositorio (historial)




¿Qué es Git?

- ❑ **Git** es un software para el control de versiones pensando en la eficiencia, la confiabilidad y compatibilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos.
- ❑ Su propósito es llevar el registro de los cambios en archivos de computadora, incluyendo coordinar el trabajo que varias personas realizan sobre los archivos compartidos en un repositorio de código.
- ❑ Fue diseñado por Linus Torvalds, el 19 de octubre de 2007.



GIT es un **sistema de control de versiones (SCV)** que nos permite administrar y rastrear los cambios que se han hecho en un conjunto de archivos (proyecto).

Git	
 git	
Información general	
Tipo de programa	Control de versiones
Autor	Linus Torvalds
Desarrollador	Junio Hamano, Linus Torvalds
Modelo de desarrollo	Software libre
Lanzamiento inicial	19 de octubre de 2007
Vulnerabilidades	CVE-2022-39253 CVE-2022-39260
Licencia	GNU GPL v2
Información técnica	
Programado en	C, Bourne Shell, Perl ¹
Versiones	
35.1 (info) (19 de enero de 2022 (1 año, 5 meses y 29 días))	

¿Qué es GitHub?

- ❑ **GitHub** es una plataforma de desarrollo colaborativo para alojar proyectos utilizando el sistema de control de versiones Git.
- ❑ Se utiliza principalmente para la creación de código fuente de programas para ordenador.
- ❑ El software sobre el cual opera GitHub fue escrito en Ruby on Rails.



Github es un servicio de *hosting* que permite almacenar los proyectos de desarrollo de software y control de versiones usando Git

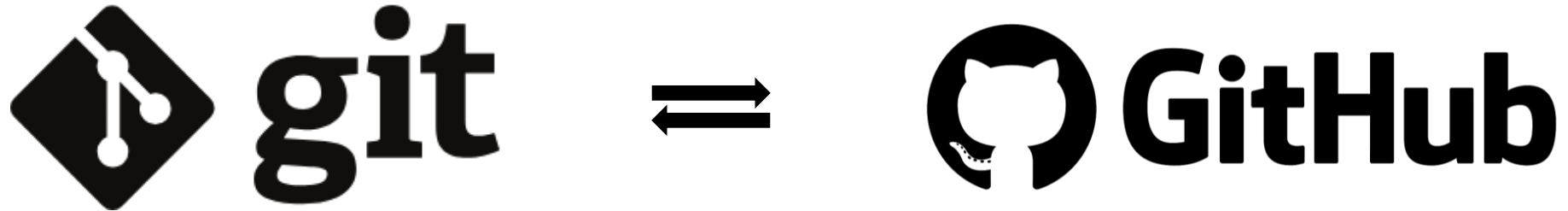
GitHub	
<i>"Build software better, together"; "Where software is built"</i>	
GitHub	
Información general	
Dominio	https://github.com/
Tipo	Sistema de control colaborativo de revisión y desarrollo de software
Comercial	Sí
Registro	Opcional (se requiere para crear y unirse a proyectos)
Idiomas disponibles	Inglés
En español	No
Estado actual	Activo
Gestión	
Desarrollador	Tom Preston-Werner Chris Wanstrath PJ Hyett
Propietario	Microsoft
Operador	Microsoft
Lanzamiento	9 de 2008
Estadísticas	
millones	
(2021)	

¿Por qué usar Git y Github?

- ☐ **Git** es el sistema de control de versiones por excelencia y **Github** es la plataforma de código colaborativo.
- ☐ Ambas son herramientas ampliamente usadas a nivel mundial, tanto para el desarrollo de proyectos personales como profesionales.
- ☐ No importa el tipo del lenguaje de programación y el entorno de desarrollo.
- ☐ Facilita el trabajo colaborativo rápido y sin errores.
- ☐ Se han convertido en las herramientas de excelencia para el desarrollo de software.
- ☐ Permite registrar el histórico de nuestro trabajo.
- ☐ Permite crear y disponer de copias de seguridad de nuestro código.
- ☐ Es un estándar que se debe conocer y dominar.



¿Por qué usar Git y Github?



- ☐ Es un software.
- ☐ Se instala localmente en el sistema.
- ☐ Es una herramienta de línea de comando.
- ☐ Es una herramienta para gestionar diferentes versiones de ediciones realizadas en archivos en un repositorio git.
- ☐ Proporciona funcionalidades como la gestión del código fuente del sistema de control de versiones.

- ☐ Es un servicio.
- ☐ Está alojado en la web.
- ☐ Proporciona una interfaz gráfica.
- ☐ Es un espacio para subir una copia del repositorio Git.
- ☐ Proporciona funcionalidades de Git como SCV, administración de código fuente y agrega algunas de sus propias características.



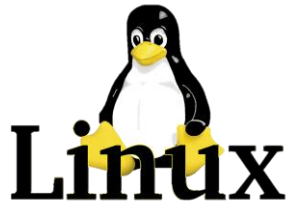
¿Por qué usar Git y Github?

Google

 Microsoft



LinkedIn



The Netflix logo, featuring a large red 'N' above the word 'NETFLIX' in a bold, red, sans-serif font.



II. CONCEPTOS BÁSICOS

Conceptos Básicos

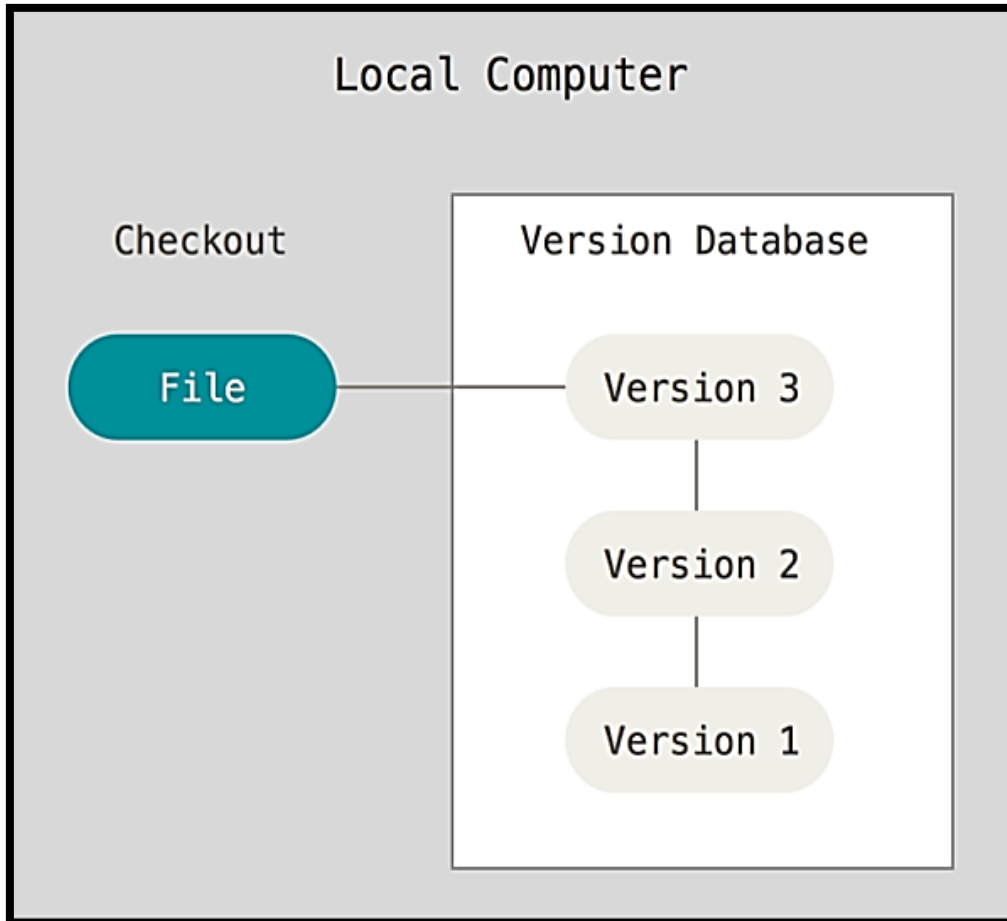
1. SISTEMA DE CONTROL DE VERSIONES (VERSION CONTROL)

Un **Sistema de Control de Versiones** es un sistema que registra los cambios realizados en un archivo o conjunto de archivos a lo largo del tiempo, de modo que se puedan recuperar versiones específicas más adelante.



Conceptos Básicos

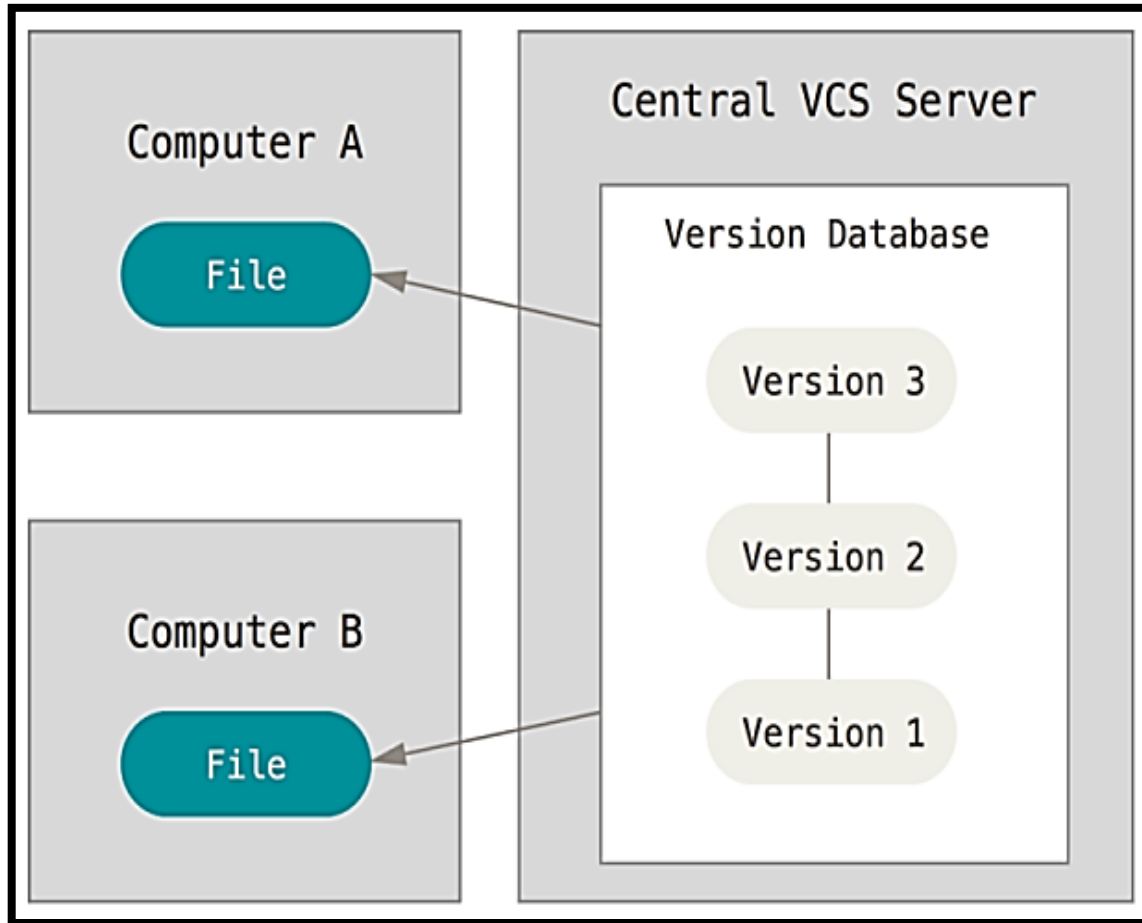
Control de Versión LOCAL (LCVS - Local Version Control System)



- Consiste en crear copias locales de tus archivos en otra carpeta de tu computadora.
- **Se caracteriza por ser fácil de implementar; es decir, copias, pegas y renombas directorios.**
- **Contiene una simple base de datos en la que se llevaba el registro de todos los cambios realizados a los archivos.**
- Una de las herramientas de CVS más popular fue un sistema llamado **RCS**.
- **Desventaja:**
poca o nula trazabilidad que se puede dar a cada archivo y ¿cómo colaborar con desarrolladores?

Conceptos Básicos

Control de Versión CENTRALIZADO (Centralized Version Control System - CVCS)



➤ Tienen un único servidor que contiene todos los archivos versionados y varios clientes que descargan los archivos desde ese lugar central.

➤ Ejemplos: **Subversion** y **Perforce**

➤ *Ventajas:*

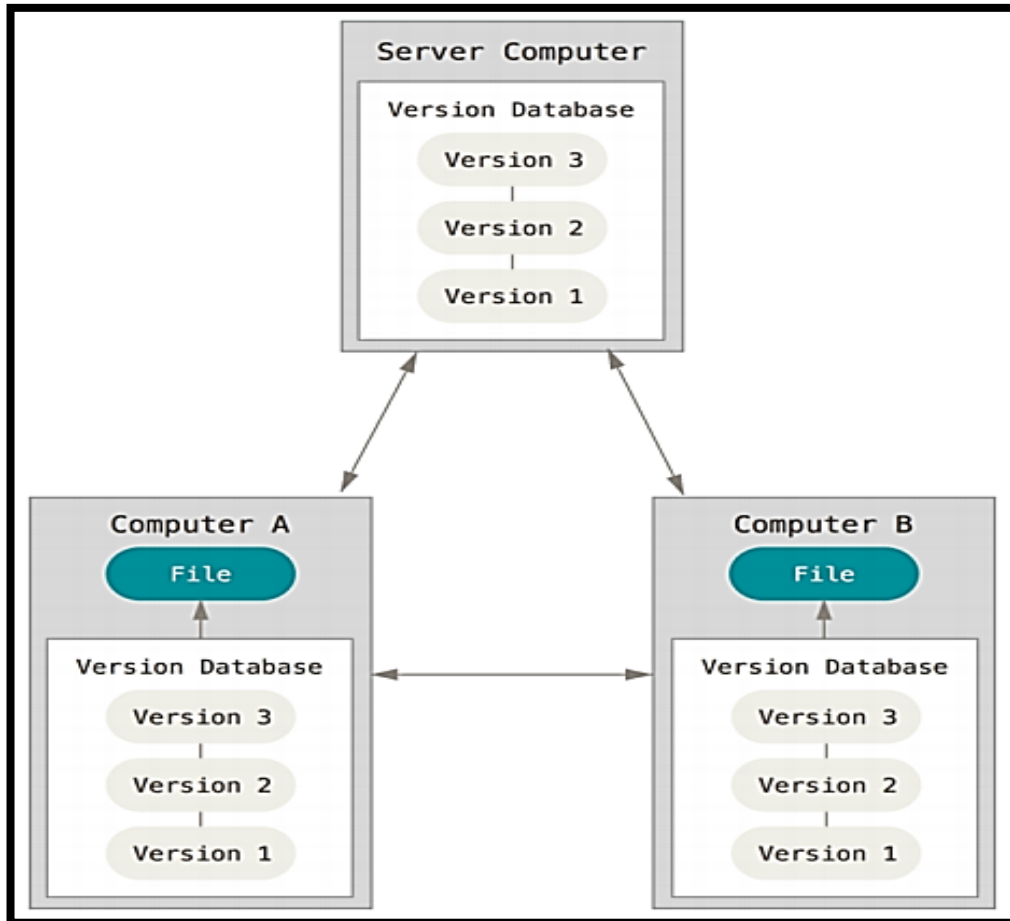
- ✓ todas las personas saben en qué están trabajando los otros colaboradores,
- ✓ los administradores tienen control detallado sobre qué puede hacer cada usuario,
- ✓ es mucho más fácil administrar un CVCS que tener que lidiar con bases de datos locales en cada cliente.

➤ *Desventaja:*

¿Y si falla en el servidor centralizado?

Conceptos Básicos

Control de Versiones DISTRIBUIDO (Distributed Version Control System - DVCS)



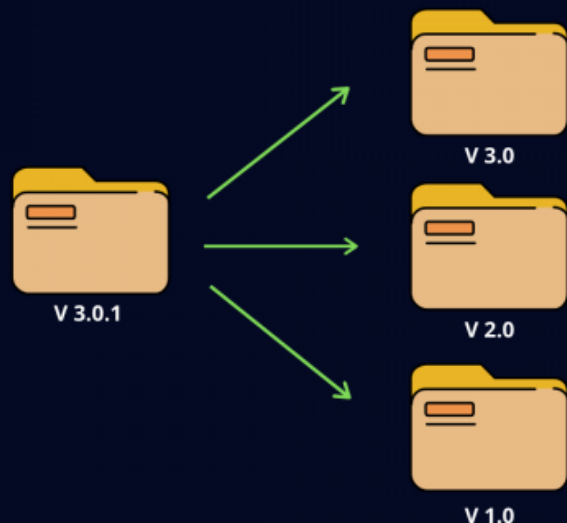
- Ofrecen soluciones a los problemas LVCS y CVCS.
- Ejemplos: **Git**, **Mercurial**, **BitKeeper**, **Bazaar** o **Darcs**.
- *Ventajas:*
 - ✓ Los clientes no solo descargan la última copia instantánea de los archivos, sino que se replican completamente el repositorio.
 - ✓ Cada clon es realmente una copia completa de todos los datos.
 - ✓ Permite colaborar simultáneamente con diferentes grupos de personas en distintas maneras dentro del mismo proyecto.

Conceptos Básicos

Arquitectura de los Sistemas de Control de Versión

Locales

Copias locales del código fuente.



Centralizados

Una copia central en servidor dedicado. (Cliente - Servidor)



Distribuidos.

Cada programador tiene una copia local del código.



Ejemplos de herramientas para hacer control de versiones



Bazaar



mercurial

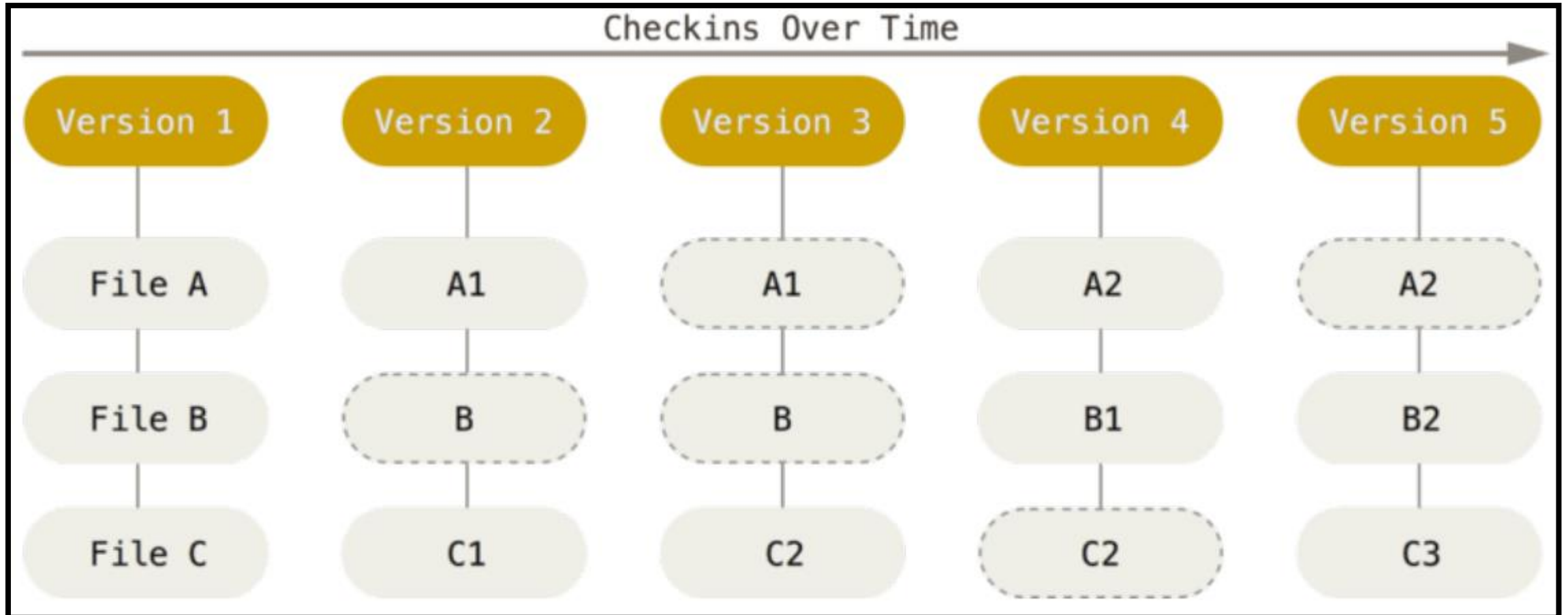
Distribuido



Cliente-Servidor

Conceptos Básicos

Almacenamiento de datos en Git, como instantáneas del proyecto a través del tiempo

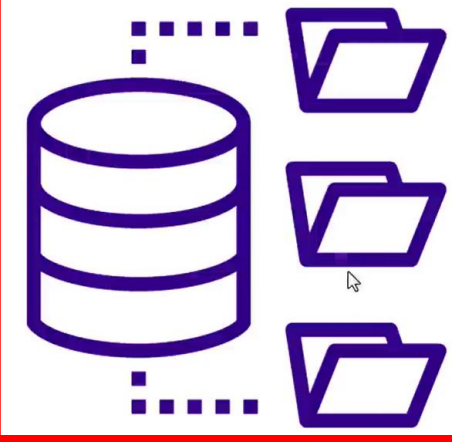


Conceptos Básicos

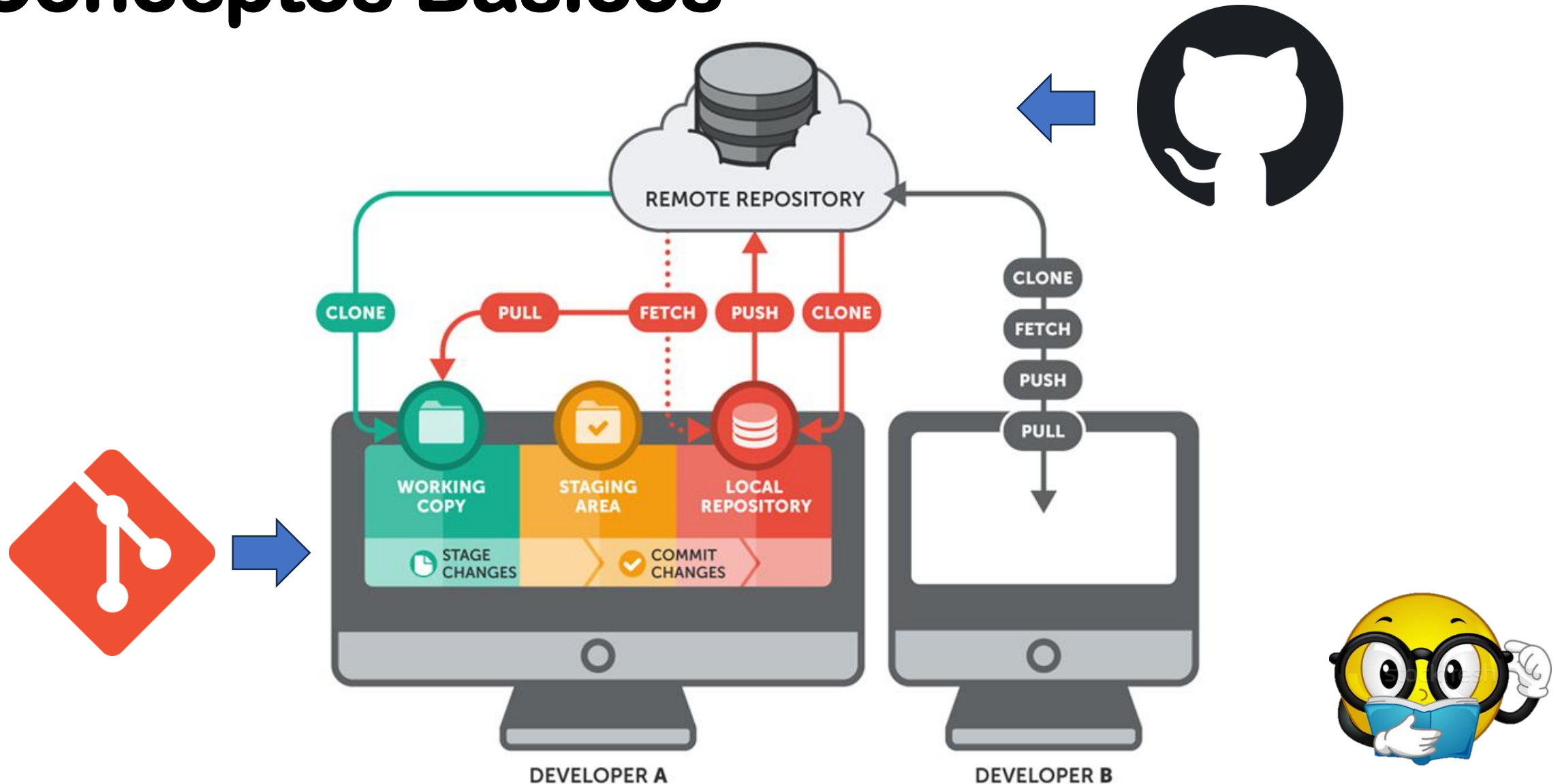
2. REPOSITORIO (REPOSITORY)

- Un Repositorio es una colección de archivos de distintas versiones de un proyecto.
- Es el sistema encargado de administrar los cambios realizados en programas de computadora o conjuntos de archivos.
- Este puede ser local o remoto.

Un **repositorio local** se guarda de forma local en tu computadora (por ejemplo, Git). Mientras un **repositorio remoto** se guarda en los servidores del servicio de hosting que elijas (por ejemplo, GitHub).



Conceptos Básicos



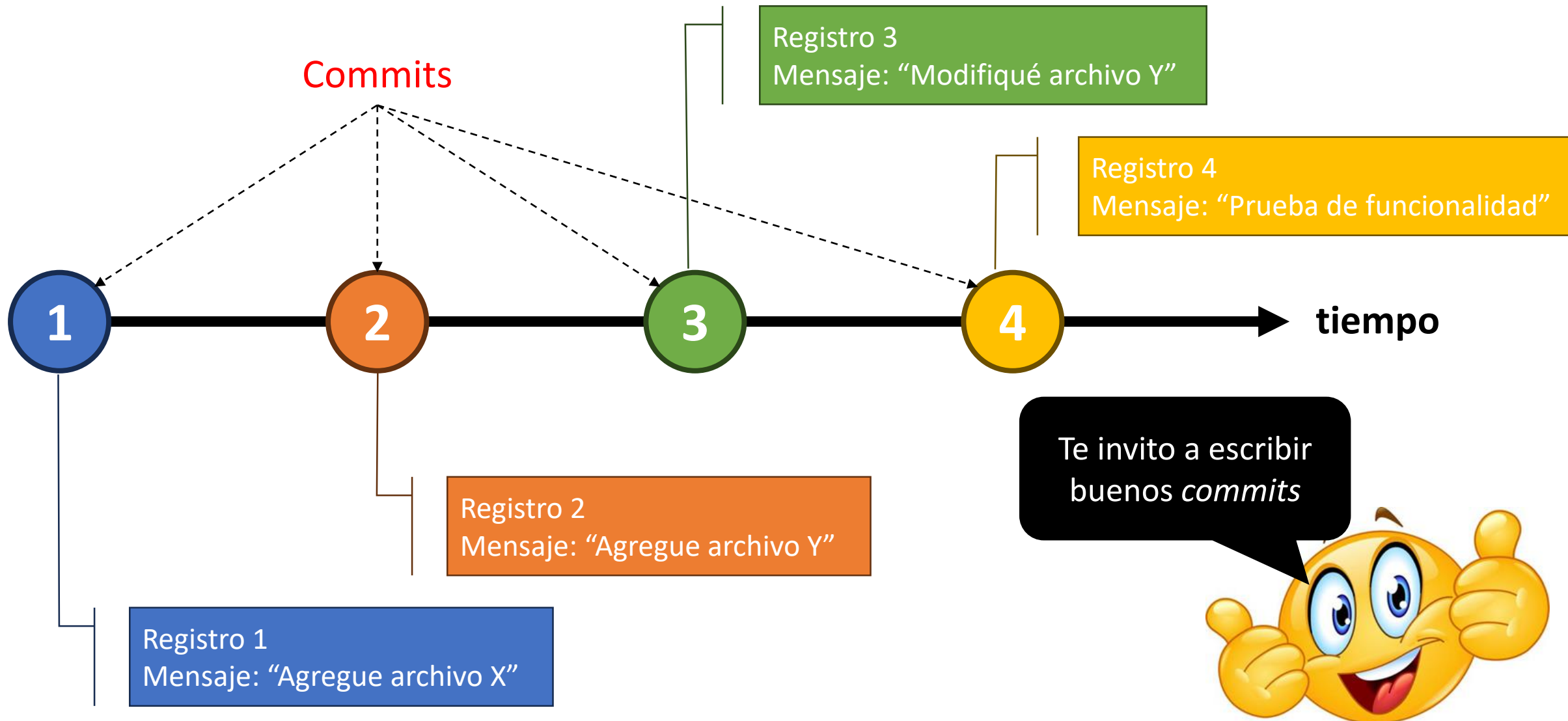
Conceptos Básicos

3. PERPETUAR/COMPROMETERSE/CONFIRMAR (COMMIT)

- Es el sistema encargado de administrar los cambios realizados en programas de computadora o conjuntos de archivos.
- Es el componente básico de la línea del tiempo de un proyecto de Git.
- Es un registro o "foto" del estado de un proyecto en un momento específico, donde se registra los cambios que se realizaron en los archivos en comparación con la versión anterior.



Conceptos Básicos



Conceptos Básicos



4. GIT BASH (GIT BOURNE-AGAIN SHELL)

- Es el sistema encargado de administrar los cambios realizados en programas de computadora o conjuntos de archivos.
- Se instala automática al instalar Git.
- Es una aplicación o herramienta de línea de comandos (inglés CLI = *Command Line Interface*), para entornos de Microsoft Windows, que proporciona una capa de emulación para una experiencia de línea de comandos de Git.
- **Bash** es un acrónimo de **Bourne-Again Shell**. Un **Shell** es una aplicación de terminal que se utiliza para interactuar con un sistema operativo a través de comandos escritos.



III. INSTALACIÓN DE GIT

Instalación de Git



Terminal (comandos)



Interfaz gráfica de usuario- GUI



Control de Versiones - SCV

Mayor información. consultar:
<https://git-scm.com/downloads/guis/>

Instalación de Git (descarga)



1



<https://www.git-scm.com>

2



download

3



4

Download for Windows

[Click here to download](#) the latest (2.41.0) 64-bit version of Git for Windows
released 15 days ago, on 2023-07-13.



5

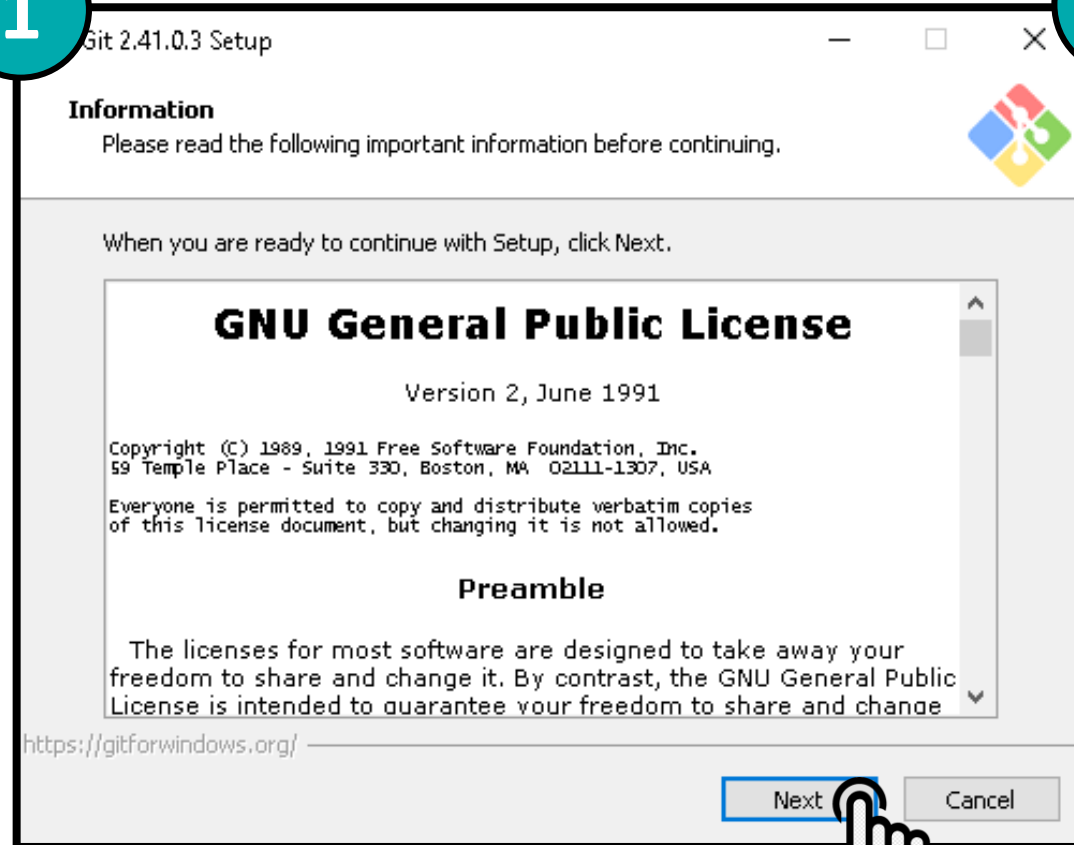
Nombre

 Git-2.41.0.3-Win64-bit_Setup



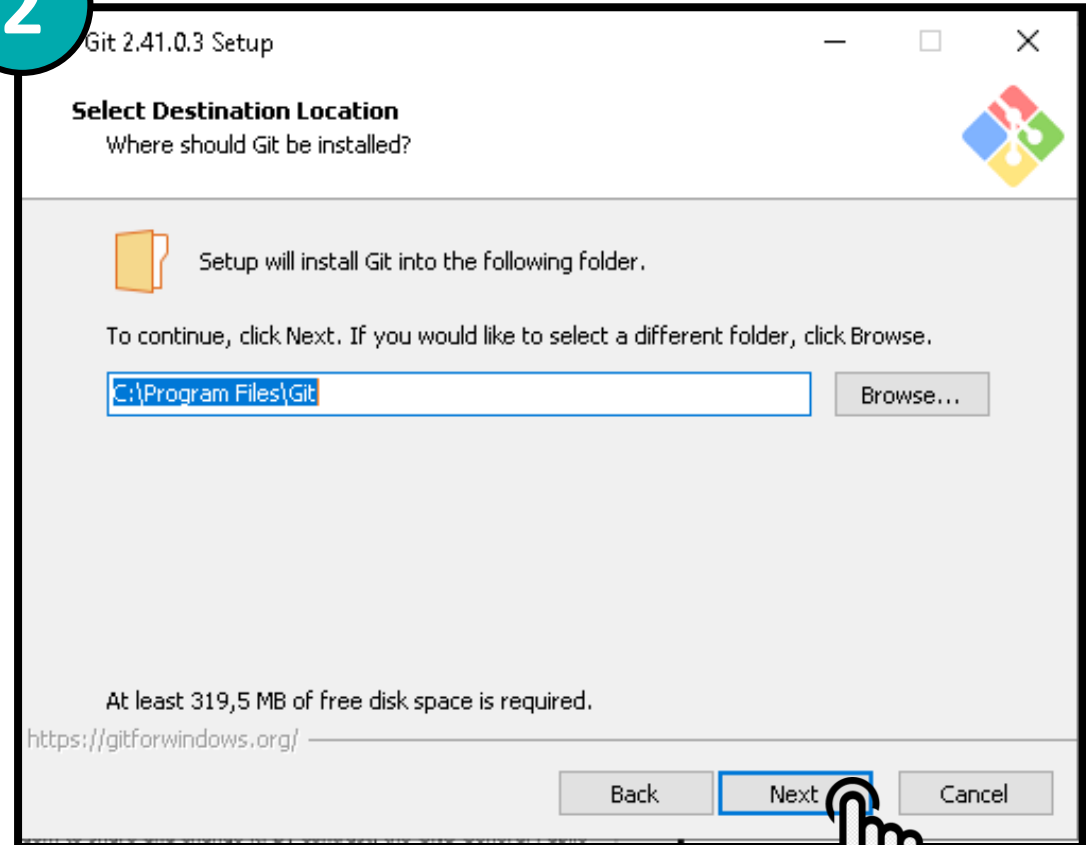
Instalación de Git (i)

1



Lea los términos y condiciones.
Clic en 'Next'

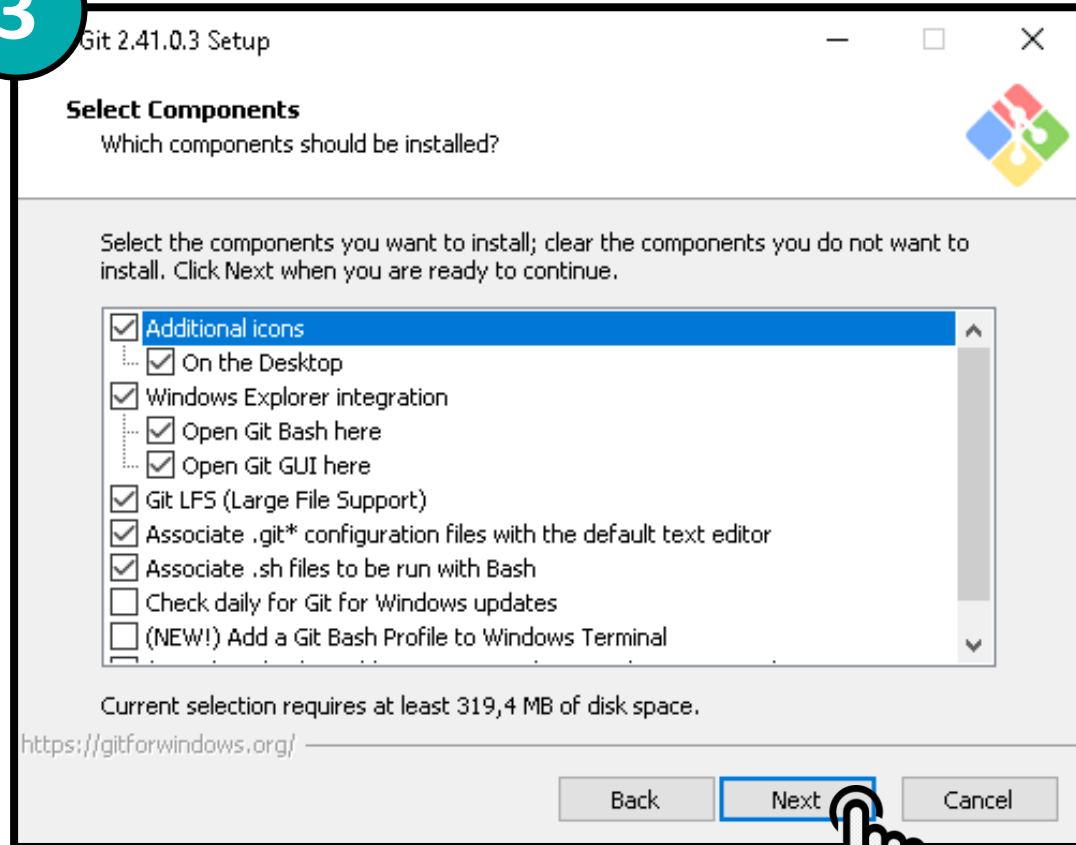
2



Defina la carpeta donde se instalará Git y clic
en 'Next'

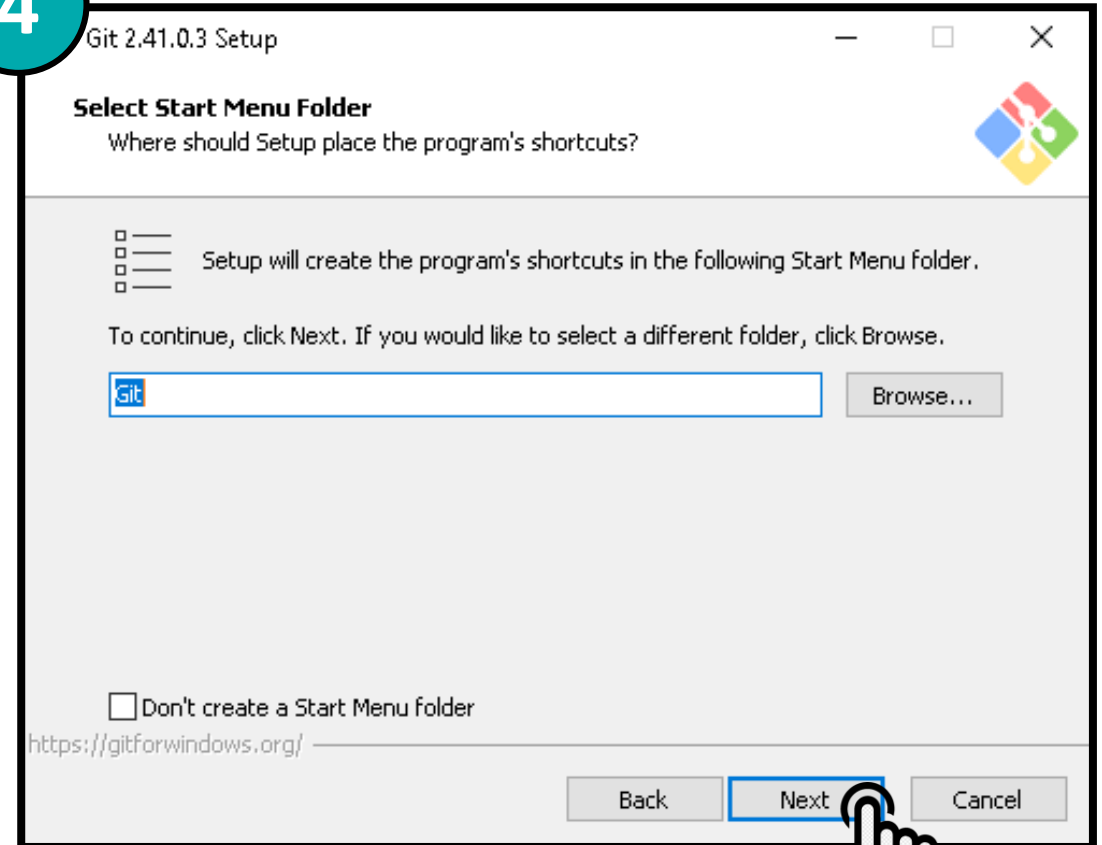
Instalación de Git (ii)

3



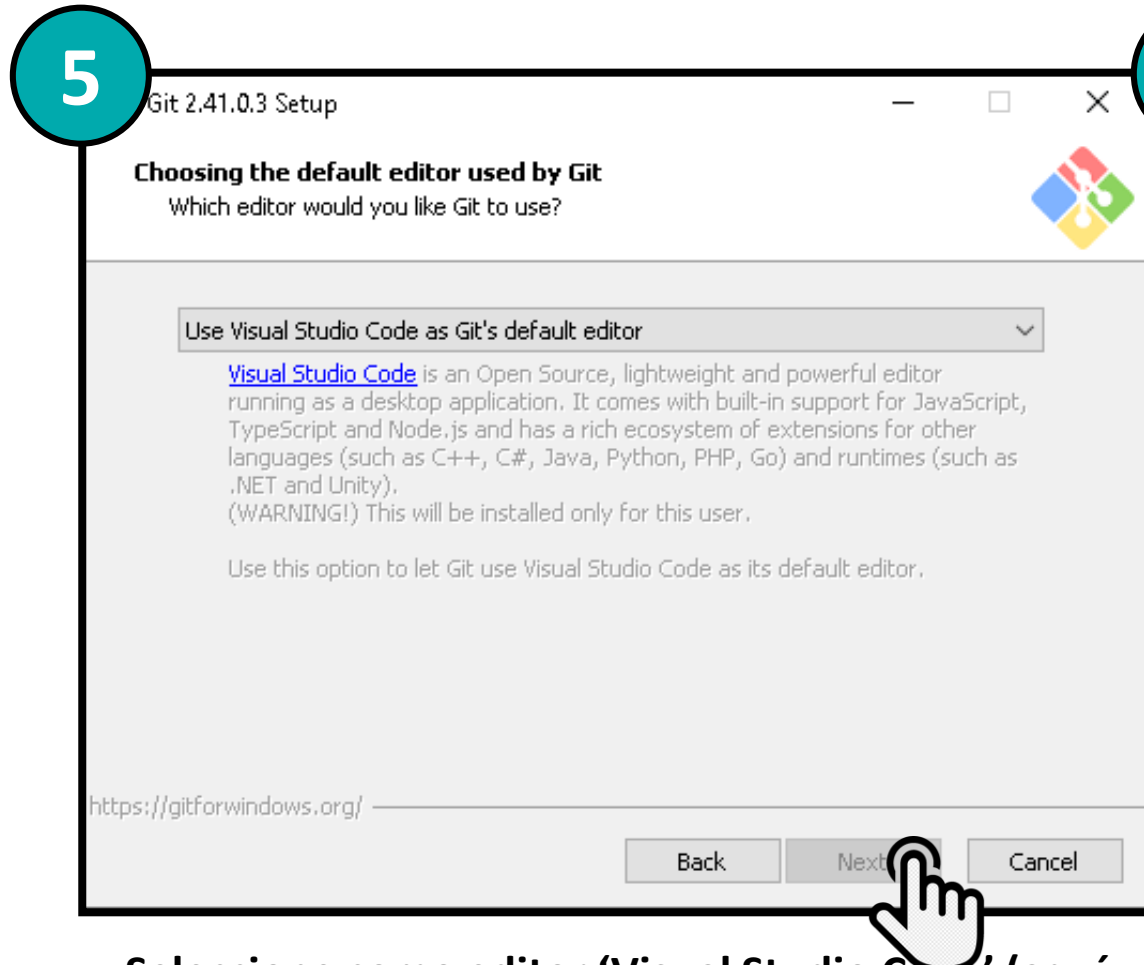
Se sugiere habilitar el ícono en el escritorio y clic en 'Next'

4

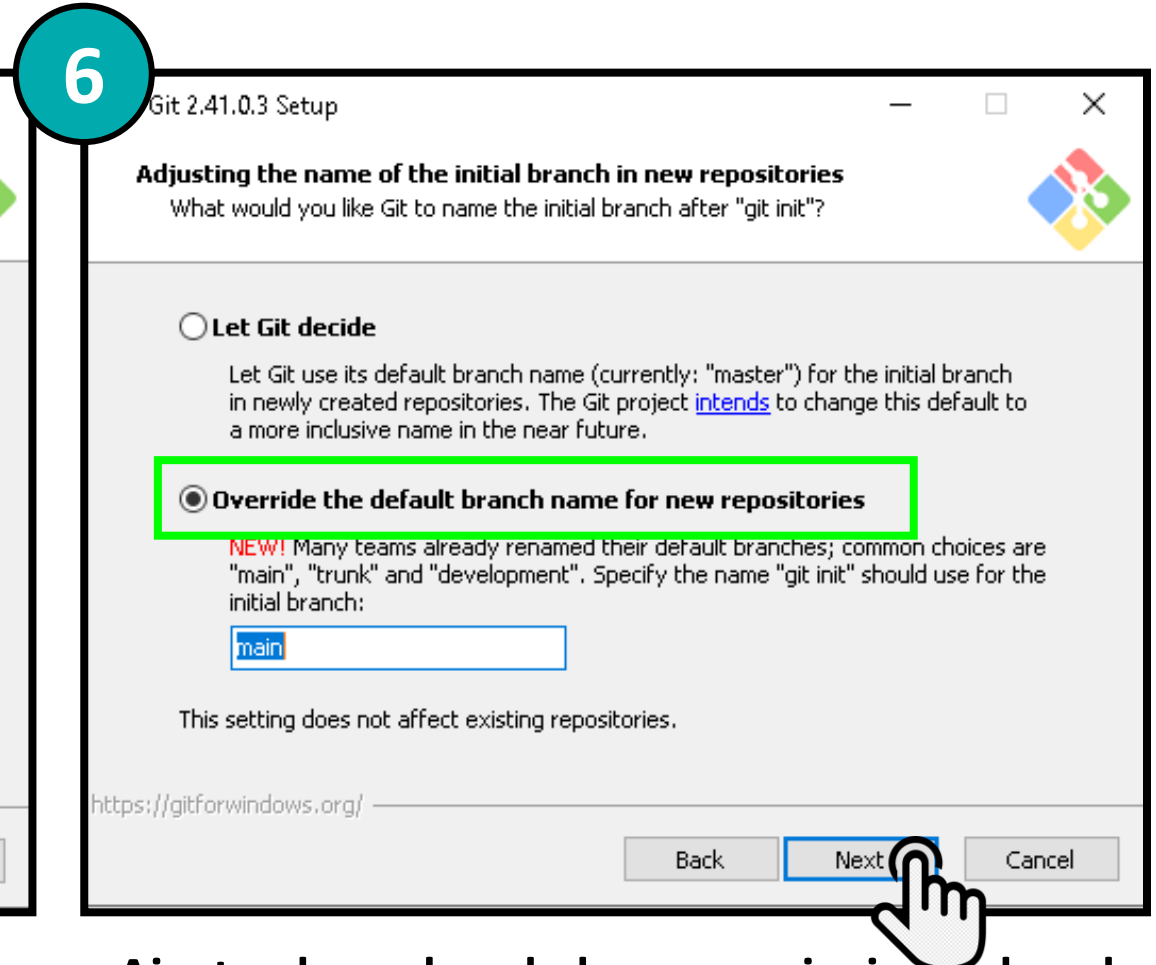


Clic en 'Next'

Instalación de Git (iii)

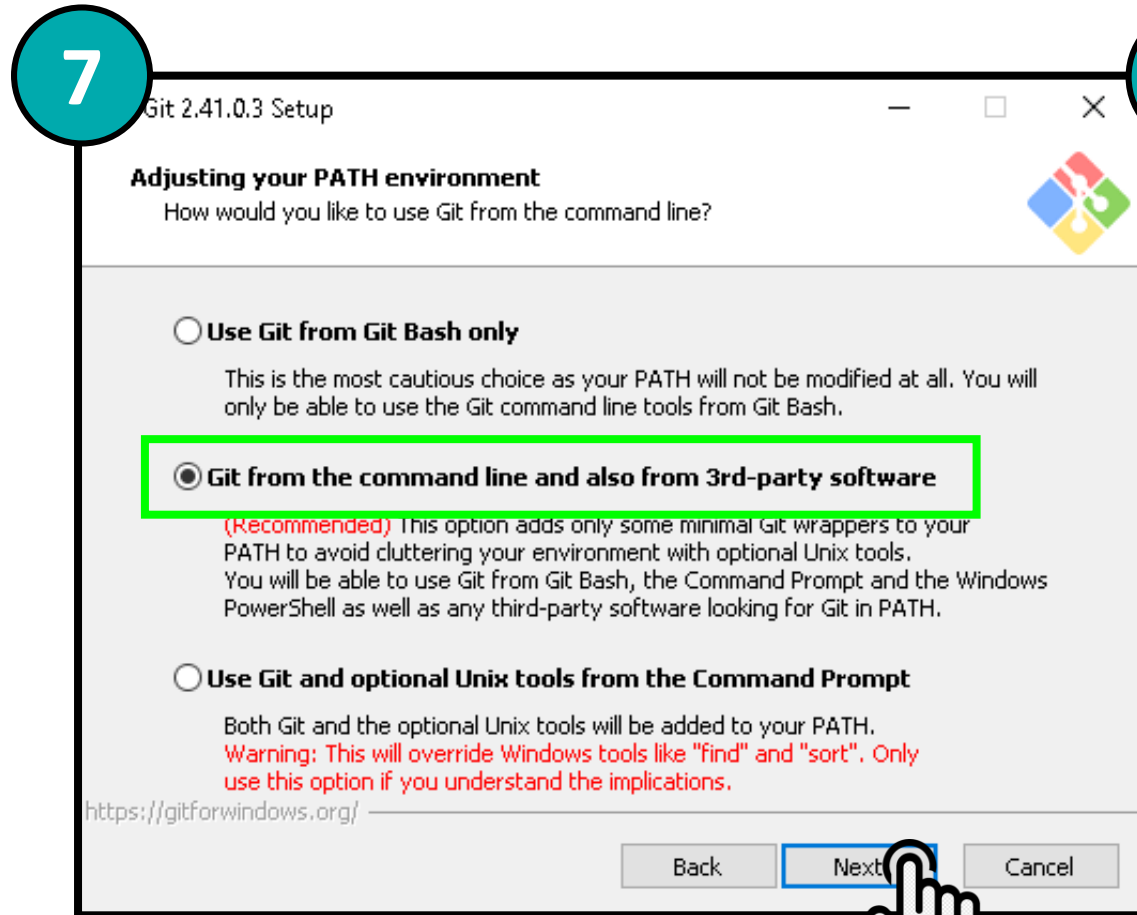


Seleccione como editor 'Visual Studio Code' (aquí se escribirán los *commits*). Clic en 'Next'

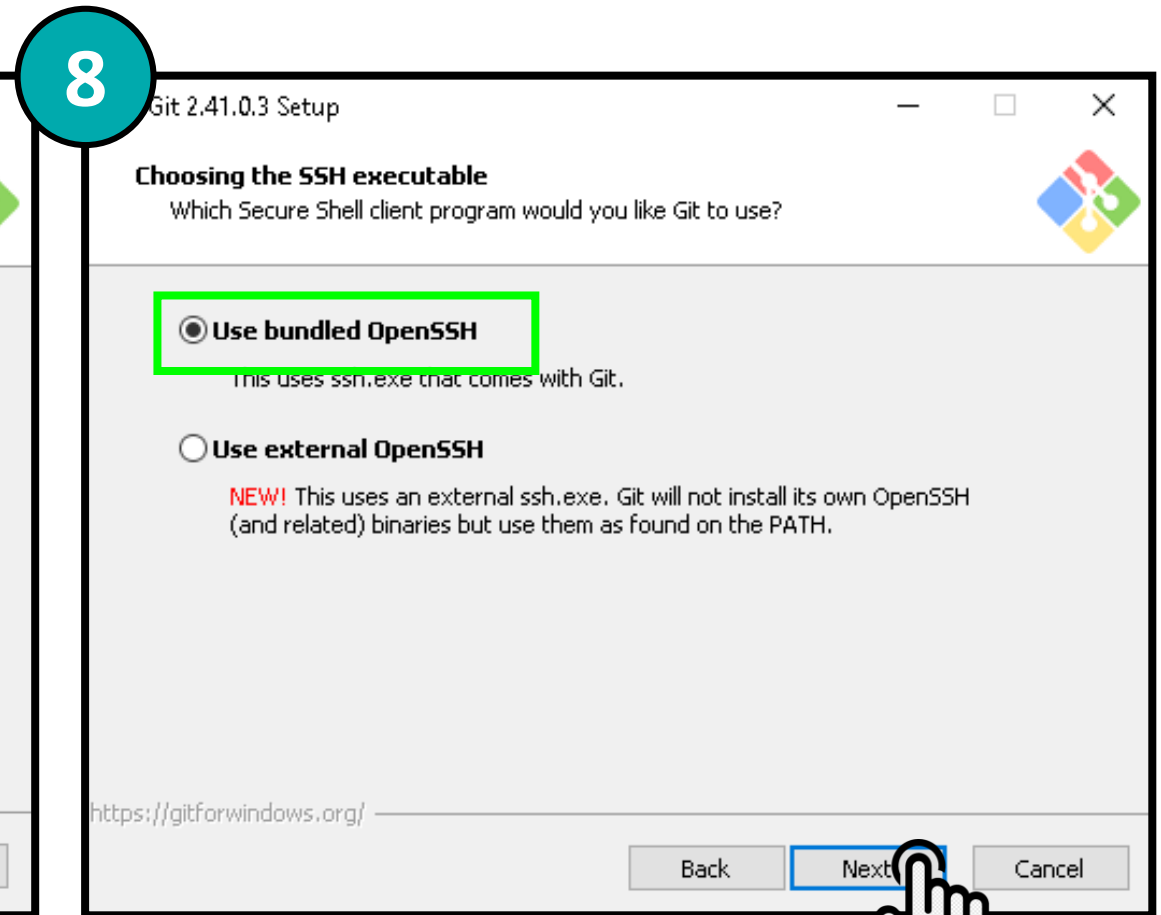


Ajuste el nombre de la rama principal, el cual defecto es 'master' o 'main'. Clic en 'Next'

Instalación de Git (iv)



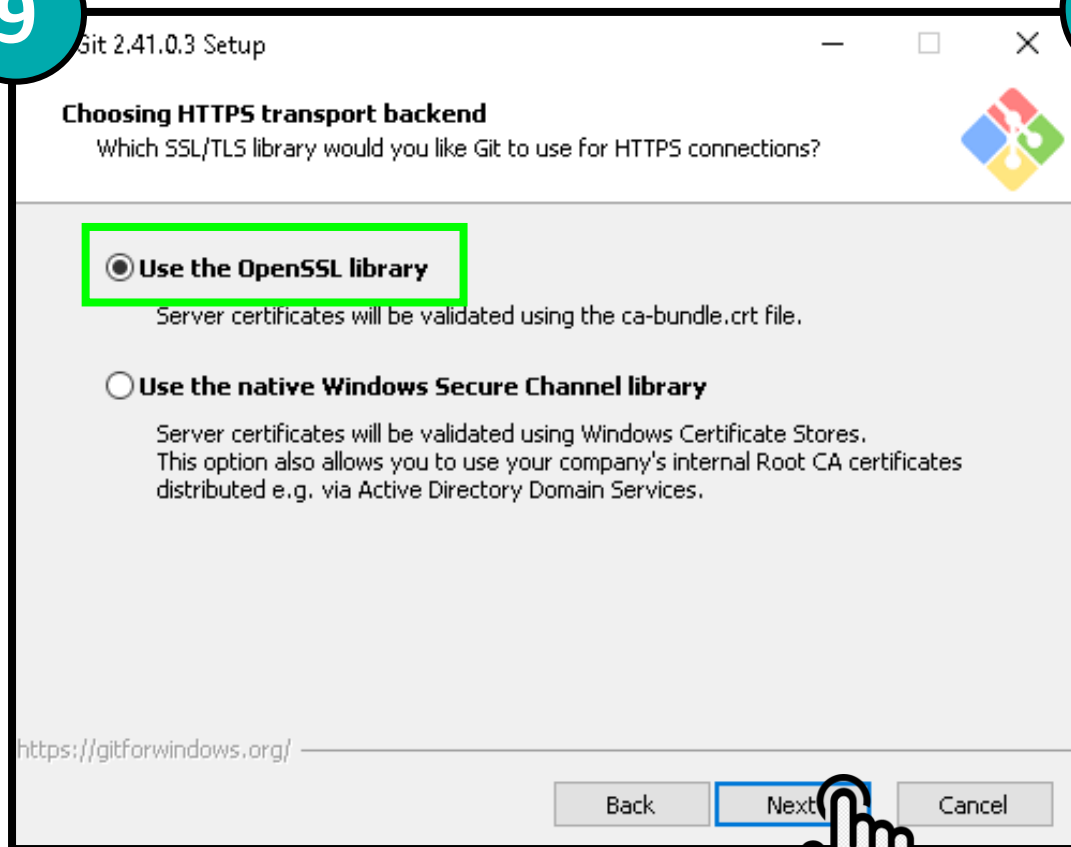
Seleccione la opción recomendada y clic en 'Next'



Ajuste el nombre de la rama principal, el cual defecto es 'master' y clic en 'Next'

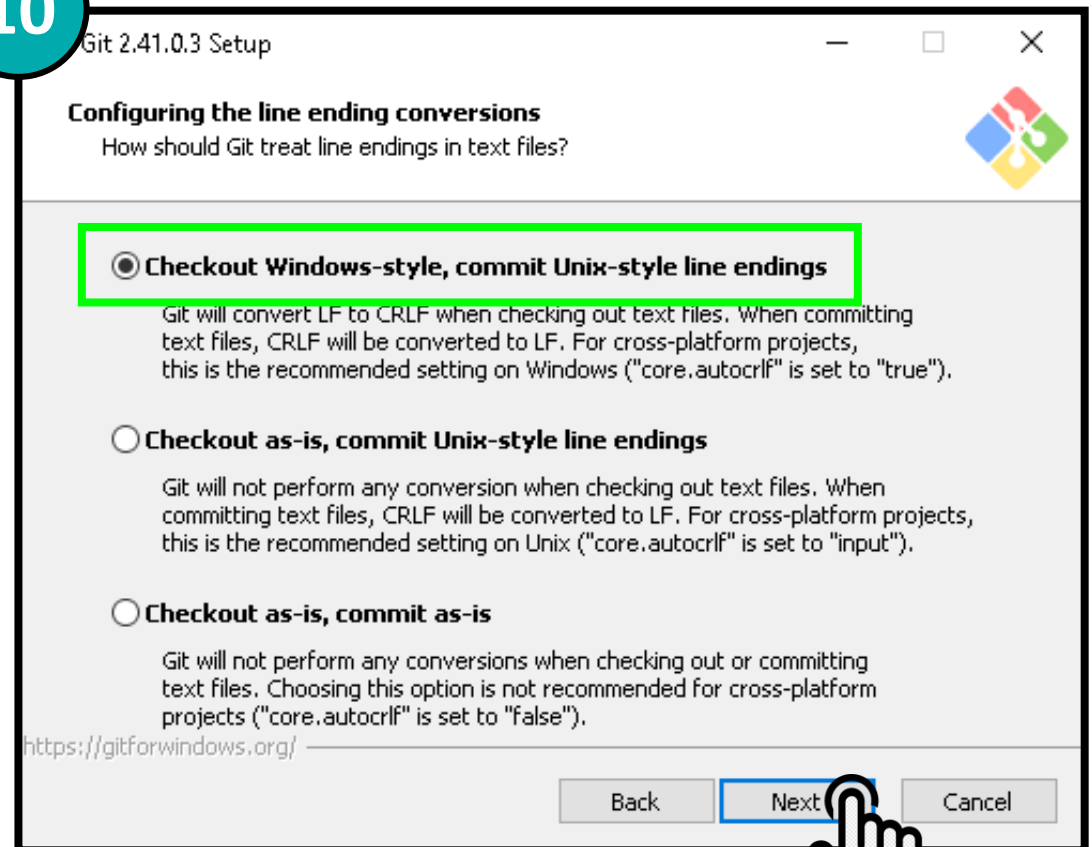
Instalación de Git (v)

9



Seleccione la opción recomendada y clic en
'Next'

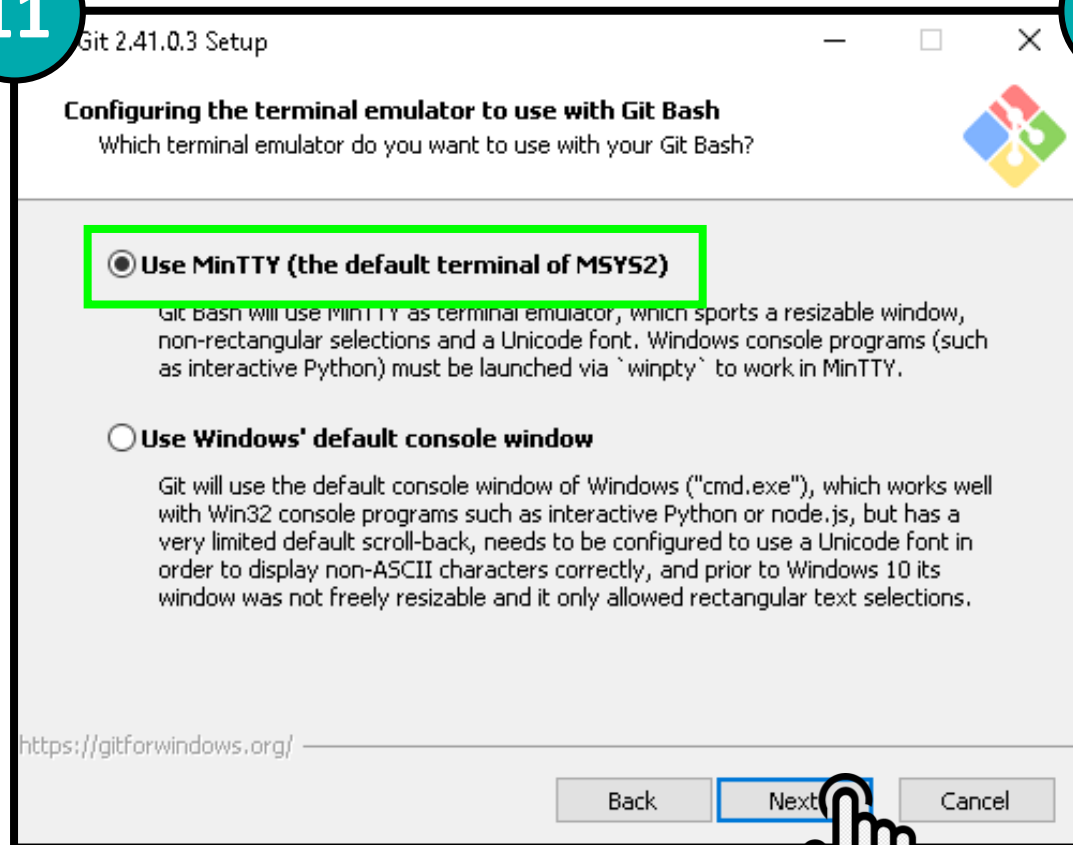
10



Seleccione la opción recomendada y clic en
'Next'

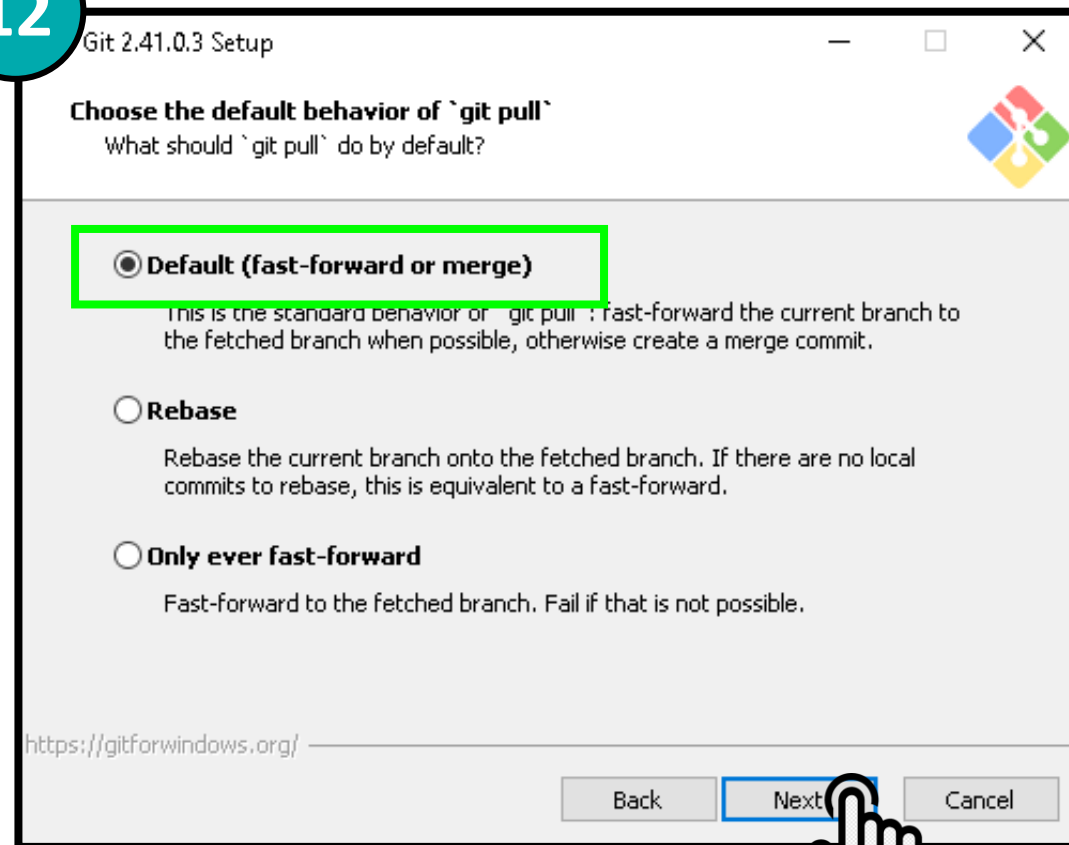
Instalación de Git (vi)

11



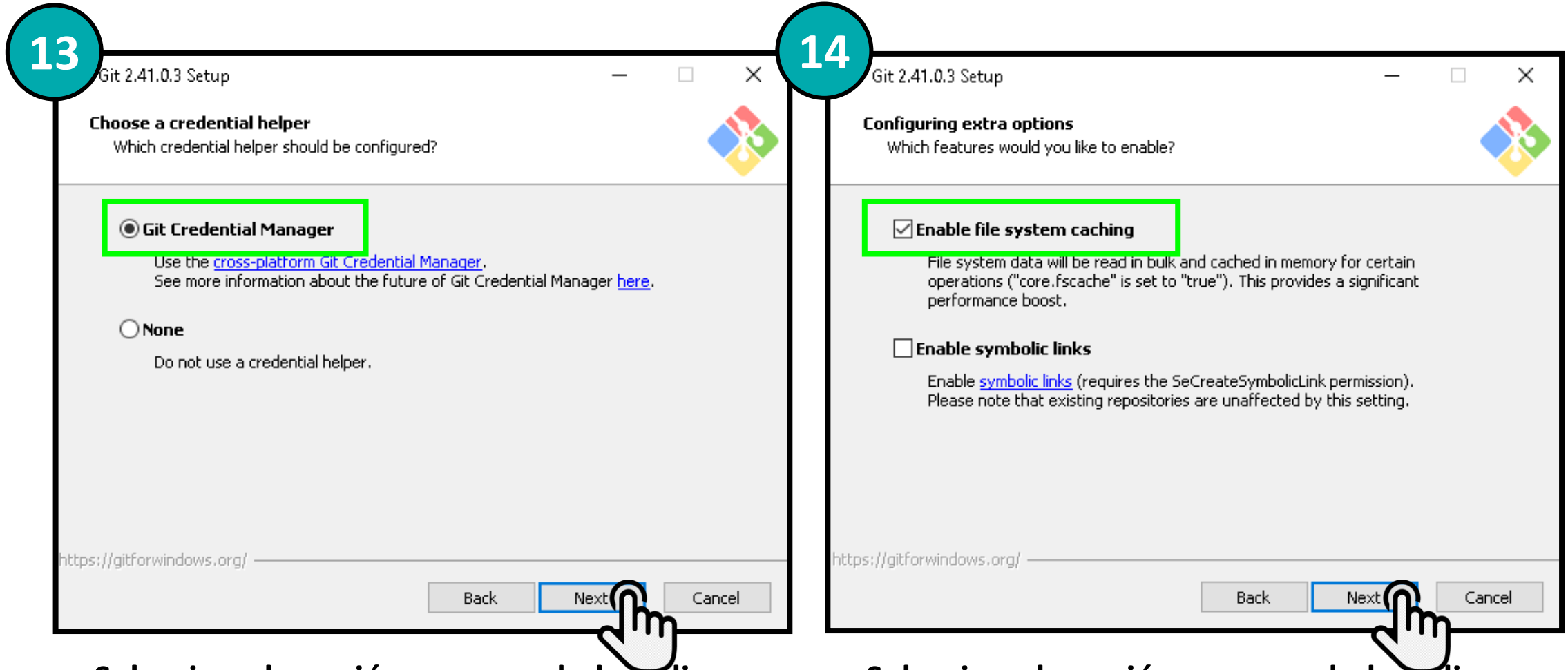
Seleccione la opción recomendada y clic en
'Next'

12



Seleccione la opción recomendada y clic en
'Next'

Instalación de Git (vii)

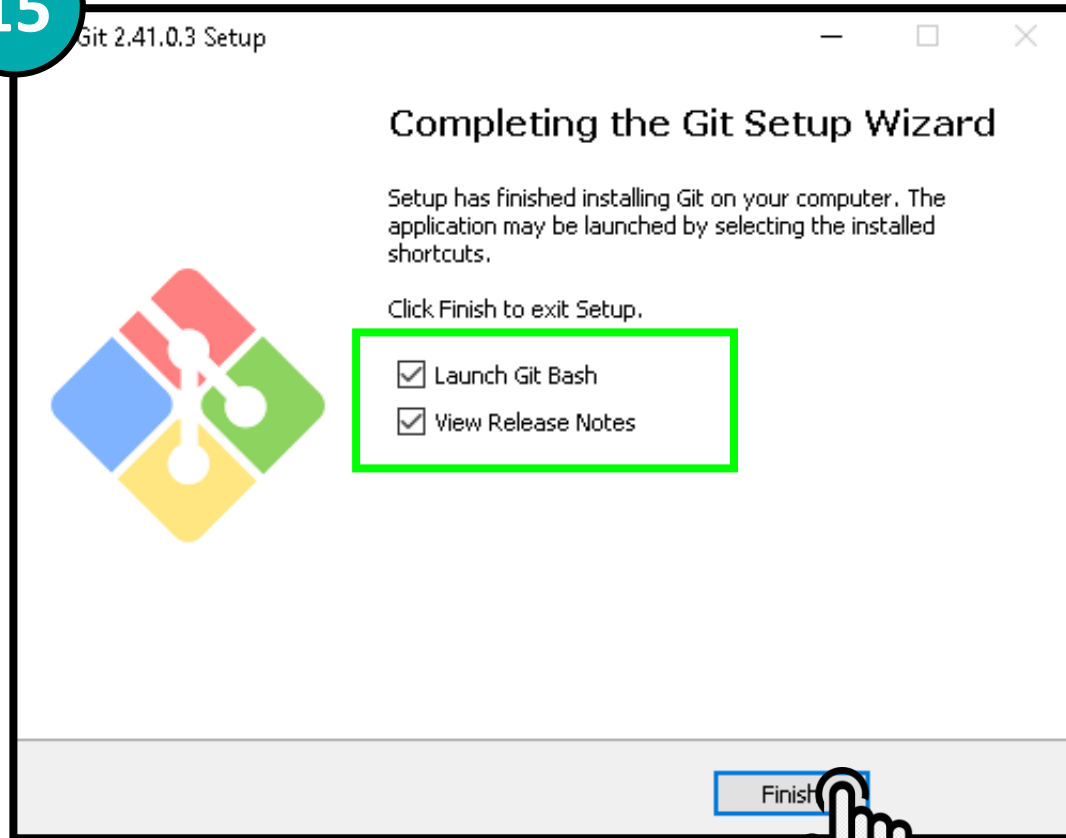


Seleccione la opción recomendada y clic en
'Next'

Seleccione la opción recomendada y clic en
'Next'

Instalación de Git (viii)

15



Seleccione las dos opciones y clic en 'Finish'

Ten presente que Git ya viene instalado en los sistemas operativos Linux y Mac OS; luego, en estos OS sería una actualización.



Mayor información:
<https://git-scm.com/book/en/v2/Getting-Started-Installing-Git>

Bienvenido a Git



```
USUARIO@DESKTOP-5B7BAH4 MINGW64 ~
```

\$ |

Usuario de tu computadora

Prompt

Indica en que directorio se está dentro
nuestro sistema de archivos
'~' indica que se está en el directorio raíz

Identificador o nombre del dispositivo

GitBash es una herramienta de línea de comandos (inglés CLI = *Command Line Interface*) que permite la ejecución, por escrito, de los comandos de Git.



Ctrl + Scroll = Zoom In/Out

IV. COMANDOS BÁSICOS



**Elige
una
pregunta**

Cuestionario 1

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16



Comando Básicos (resumen)

Comando	Comentario
<code>pwd</code>	Muestra el directorio de trabajo actual.
<code>mkdir DIRECTORY</code>	Permite crear un nuevo directorio o carpeta en el sistema de archivos.
<code>touch FILE</code>	Se utiliza para crear archivos vacíos y cambiar marcas de tiempo de archivos o carpetas
<code>cd DIRECTORY</code>	Permite cambiar el directorio en el cual nos encontramos trabajando para ir a otro según sea la necesidad
<code>cd ..</code>	Permite regresar al directorio o carpeta anterior.

Comando Básicos (resumen)

Comando	Comentario
cd o cd ~	Nos lleva a la ruta raíz del usuario
cd /D/	Permite pasar al disco D:\
cd -	Nos lleva directamente al último directorio visitado
ls -l	Permite ver todos los archivos como una lista en donde incluye el usuario, grupo, permisos sobre el archivo, tamaño, fecha y hora de creación (formato largo).

Comando Básicos (resumen)

Comando	Comentario
ls -R	Muestra el contenido de todos los subdirectorios de forma recursiva (árbol), es decir, se muestra el contenido de todos los directorios y subdirectorios de manera descendente.
rm -r DIRECTORY	Permite eliminar la carpeta y los archivos dentro de ella de forma recursiva.
rm -d DIRECTORY	Borra directorios vacíos
rmdir DIRECTORY	Permite eliminar permanentemente un directorio o subdirectorio <u>vacío</u>

Comando Básicos (resumen)

Comando	Comentario
history	Permite ver los últimos comandos que se han ejecutado y un número especial con el que podemos volver a repetir el comando. El sistema listará hasta 500 comandos ejecutados.
history -c	Borra toda la lista del historial
clear	Se utiliza para limpiar la pantalla de la terminal, eliminando todo el contenido anterior y dejando la pantalla en blanco. Es importante mencionar que este comando no elimina el historial de comandos anteriores.

Comando Básicos (resumen)

Comando	Comentario
date	Permite visualizar la fecha y hora del sistema
exit	Permite salir de la terminal, cerrando su ventana



`ls ..`

RESPUESTA

`pwd`

RESPUESTA

`ls`

RESPUESTA

`exit`

RESPUESTA



Usando el comando 'mkdir' seguido de los nombres de los directorios que quiero crear separados por espacio

RESPUESTA

Lista los archivos y carpetas del directorio actual

RESPUESTA

clear

No cambia el directorio en el que se encuentra

RESPUESTA

Digitando el comando seguido de '--help'.
Hay algunas excepciones

RESPUESTA



```
rm -r [Nombre_de_Directorio]
```

RESPUESTA

Git Bash

RESPUESTA

La ruta absoluta representa la ruta completa del recurso, parte del directorio raíz hasta llegar al archivo concreto que se está buscando.
Por su parte, la ruta relativa representa solo una parte de la ruta, ya que en ella se tiene en cuenta el directorio actual desde el que se está trabajando.

RESPUESTA

```
rmdir
```

la condición es que la carpeta debe estar vacía

RESPUESTA



Con la combinación Ctrl + a,
que es equivalente a la tecla Inicio

RESPUESTA

date

RESPUESTA

Que distingue instrucciones escritas en mayúsculas
de minúsculas. Y Git Bash no es case sensitive

RESPUESTA

16) Si se desea crear el directorio `dir11` dentro

`mkdir -p dir1/dir11`

RESPUESTA

V. CONFIGURACIÓN DE GIT Y CREACIÓN DE UN REPOSITORIO

Configurar usuario y correo electrónico

Comando	Comentario
\$ git config --global user.name "NOMBRE_APELLIDO"	Permite definir o modificar el nombre del usuario
\$ git config user.name	Muestra el nombre usuario actual
\$ git config --global user.email "E-MAIL_INSTITUCIONAL"	Permite definir o modificar el correo electrónico
\$ git config user.email	Muestra el correo electrónico actual
\$ git config --list	Muestra el resumen de la configuración

Crear un repositorio

Comando	Comentario
\$ git init	Convierte a la carpeta que alojará el proyecto en un repositorio (,git)
\$ git config --global init.defaultBranch main	Permite definir el nombre de la rama principal ('master' → 'main')
\$ git status	Permite revisar el estado del repositorio (área vs. estado del archivo)

VI. LAS ÁREAS DE GIT Y COMMITTS

Las tres áreas de Git



Los estados de los archivos



¿Qué es un *commit*?

Antes de realizar
nuestro primer
commit, veamos qué es



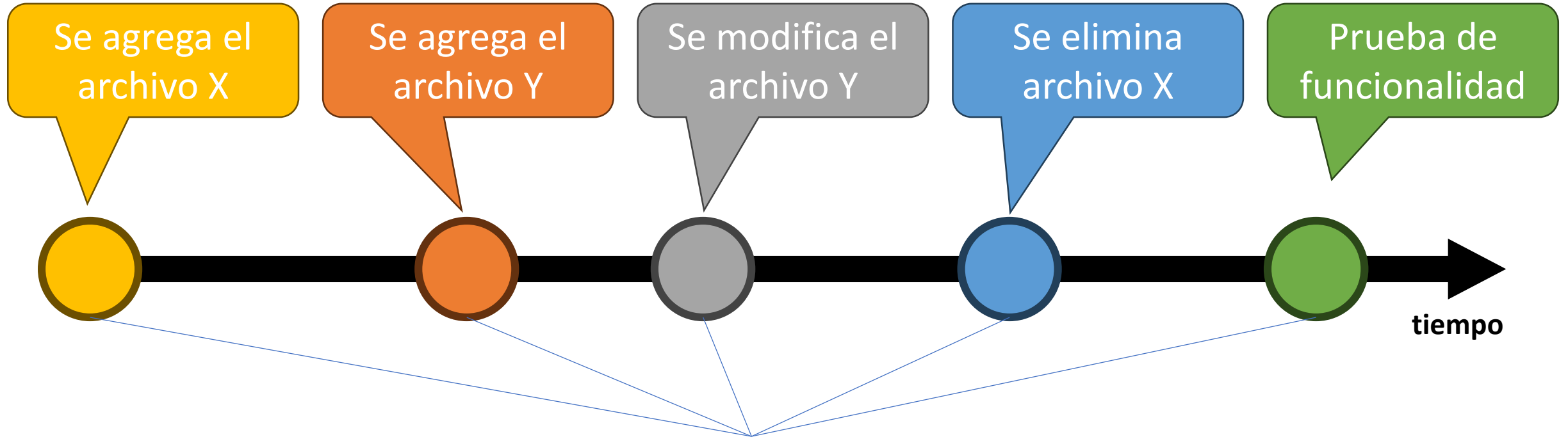
Es el componente básico de la línea del tiempo de un proyecto en Git

Se puede definir como el registro o "foto" del estado de un proyecto en un momento específico

Es donde se registra los cambios que se realizaron en los archivos en comparación con la versión anterior

Commit (Confirmar) significa, en el control de versiones, actualizar el repositorio para los cambios realizados en una copia clonada del proyecto. Después de confirmar, los otros desarrolladores deben actualizar la copia local para obtener el código nuevo y realizar los cambios.

¿Qué es un *commit*?



commits

Por ahora son *commits* básicos.
Ten presente que cada *commit* debe ilustrar o describir los cambios realizados en el proyecto



Cómo escribir un buen mensaje de *commit*

1. Escribir buenos mensajes de *commit* es muy importante, tanto para las personas que trabajan contigo como para ti, ya que te servirán para saber exactamente qué ha pasado y qué cambios se han realizado en cada momento dentro de un proyecto.

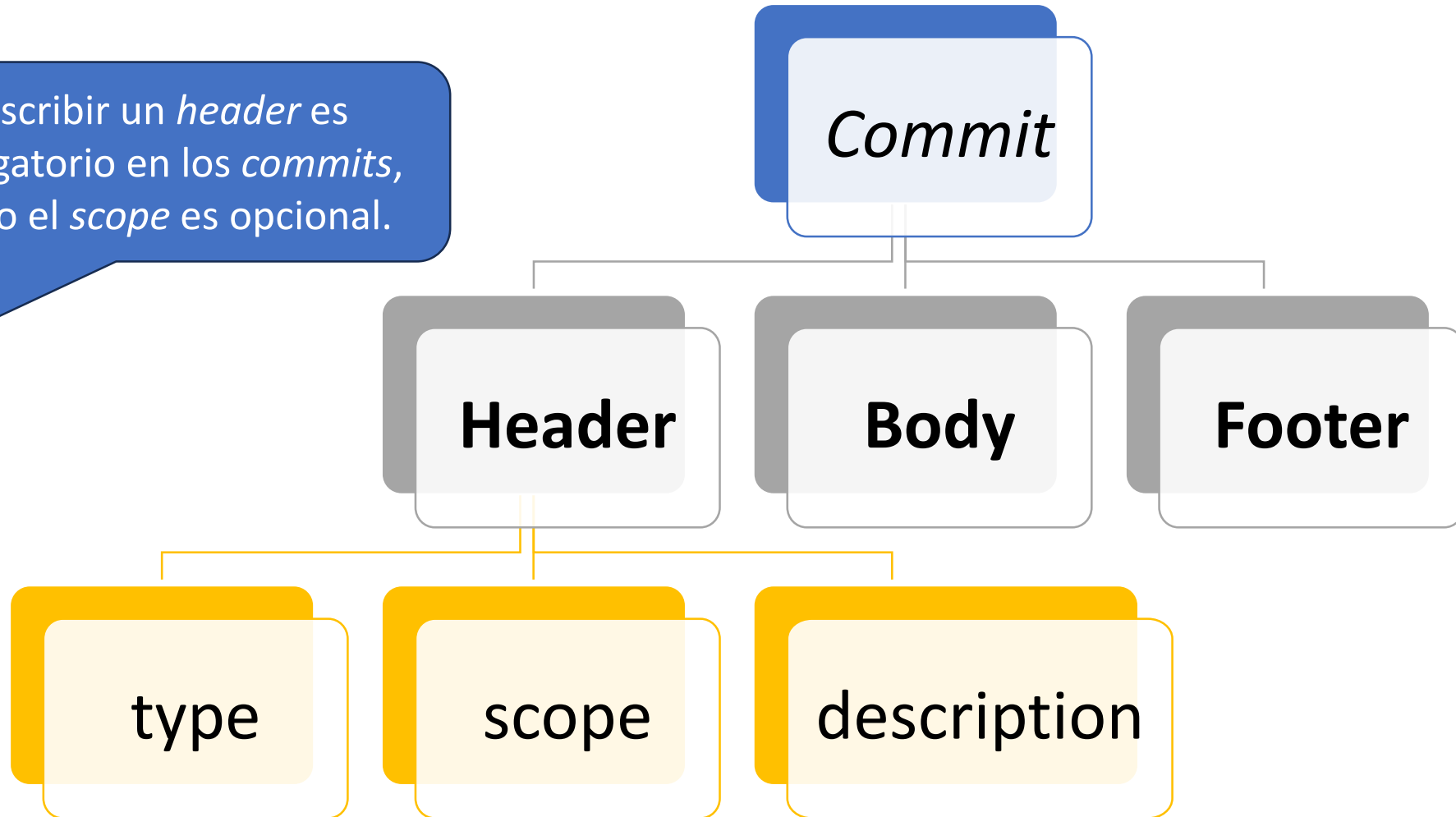
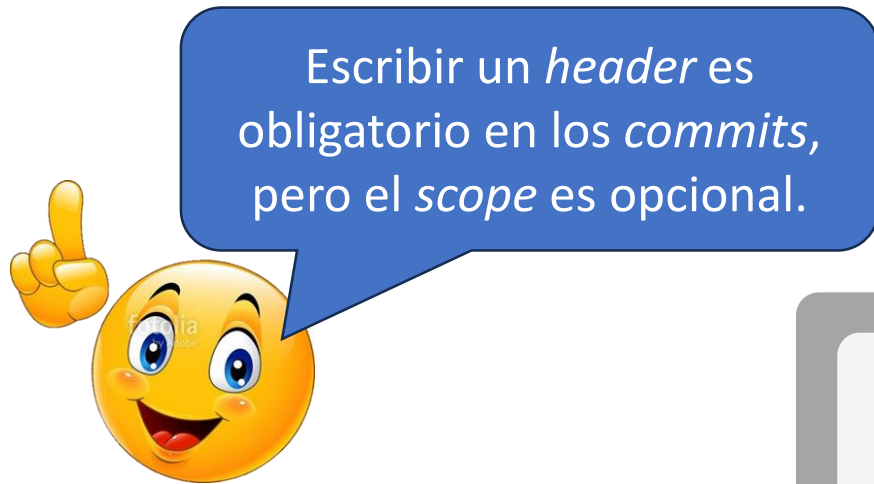
2. Las buenas prácticas a la hora de programar y mantener código también aplican a cómo escribimos mensajes de *commit*, incluso siendo proyectos personales.

3. Un mensaje de *commit* bien escrito es la mejor manera de proporcionar contexto sobre un cambio al resto de desarrolladores.

4. Podríamos pensar que en el diferencial de código se tiene toda la información necesaria para saber qué ha cambiado; pero ¿cómo respondemos al por qué de dicho cambio? Ahí es donde entra en juego el mensaje en el *commit*.

Conventional
Commits

Cómo escribir un buen mensaje de *commit*- estructura



Cómo escribir un buen mensaje de *commit*- estructura

Header

<type> <scope (optional)>: <description>

<Línea en blanco>

<body (optional)>

<Línea en blanco>

<footer (optional)>

Estructura:



Cómo escribir un buen mensaje de *commit*- estructura

Type:

Hace referencia a grandes rasgos sobre los cambios que se están guardando en ese *commit*. Por ejemplo, si las modificaciones que se hicieron fueron relacionadas a agregar una nueva característica o función, en cuyo caso se usaría el tipo 'feat' (ver lista).

Scope:

Se refiere a la parte del código en la que se está realizando un cambio, es decir, a qué clase o archivo afecta, si es para algún navegador específico, si es sobre una herramienta que se ha añadido, etc.

Type

Se debe
escribir uno de
los siguientes:



Feat:

Se ha agregado una nueva funcionalidad que no estaba.

Fix:

Se implementa o arregla una solución de un error.

Docs:

Se realizan cambios en la documentación.

Style:

Cambios que no afectan al funcionamiento del código, p. ejm. eliminar espacios en blanco, formatear código, añadir un nuevo salto de línea, etc.

Refactor:

Se ha realizado un cambio en el código que no añade funcionalidad ni arregla un error manteniendo la funcionalidad exterior.

Perf:

Un cambio en el código que mejora el rendimiento del código.

Test:

Cuando se añade nuevas pruebas o se arreglan las ya existentes.

Chore:

Cambios en el proceso de compilado o en el uso de herramientas externas (higiene al código).

Release:

Cómo escribir un buen mensaje de *commit*- estructura

Description or Subject

Esta es la parte que realmente detalla, resumidamente, lo que se ha cambiado dentro del *scope*.

Se debe aplicar las siguientes reglas:

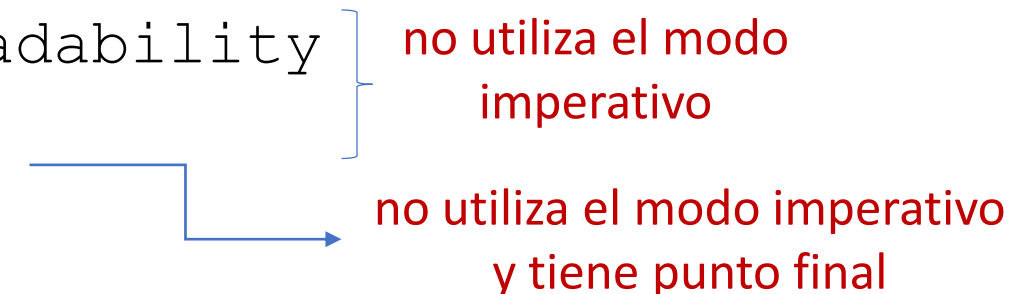
- ☐ Utilizar el modo imperativo
- ☐ No poner mayúscula en la primera letra
- ☐ No añadir un punto (.) al final de la frase
- ☐ Evitar escribir más de 50 caracteres

Cómo escribir un buen mensaje de *commit*- estructura

Algunos ejemplos de un *description* bien escritos:

- `update deployment documentation`
- `remove deprecated method`
- `merge pull request #312 from branch/name`
- `release version 1.2.3`

Algunos ejemplos de un *description* mal escritos:

- `refactored method to improve readability`
 - `more changes to class X`
 - `I have released a new version.`
- 
- no utiliza el modo imperativo
- no utiliza el modo imperativo y tiene punto final

Cómo escribir un buen mensaje de *commit*- estructura

Body:

- ✓ Sirve para explicar qué cambios se han hecho y el por qué se han hecho.
- ✓ Expande el resumen con información adicional.
- ✓ Usa líneas de texto de 72 caracteres o menos.
- ✓ No todos los *commits* son lo suficientemente complejos como para tener que escribir un *body* y a veces basta con escribir un buen *header*.

Cómo escribir un buen mensaje de *commit*- estructura

Footer:

- ✓ Es opcional.
- ✓ Suele utilizarse para añadir enlaces a algún rastreador de problemas (issue tracker) que se está utilizando como Jira, Redmine o Clubhouse.
- ✓ Puedes incluir referencias a problemas o enlaces a documentación o mencionar a colaboradores.

Cómo escribir un buen mensaje de *commit*

Agregar animaciones de ataque para el personaje principal

feat: Implement attack animations for main character

This commit introduces a set of attack animations for the main player character. The animations are seamlessly integrated into the game, enhancing the overall combat experience for players.

Resolver bug en la lógica de colisiones en el nivel 3

fix: Fix collision logic bug in level 3

Address a bug related to collision detection in the third game level. This fix ensures that player interactions with game elements in level 3 are accurately detected and processed.

Ejemplos:



Configuración del editor de texto en Git

¿Cómo puedo asociar un editor de texto diferente a VS Code para la redacción del mensaje de un commit?



Para VS Code:

```
git config --global core.editor "code -wait"
```

Para Sublime Text:

```
git config --global core.editor "'C:/Program Files (x86)/sublime text 3/subl.exe' -w"
```

Para Notepad++:

```
git config --global core.editor "'C:/Program Files (x86)/Notepad++/notepad++.exe' -multiInst -notabbar -nosession -noPlugin"
```



Mayor información. consultar:

<https://docs.github.com/es/get-started/getting-started-with-git/associating-text-editors-with-git>

Crear un commit

Comando	Comentario
\$ git add NOMBRE_ARCHIVO	Permite llevar el archivo especificado al área de preparación.
\$ git add .	Se indica que todos los cambios serán considerados para el siguiente commit.
Alternativa 1: \$ git rm --cached <NOMBRE_ARCHIVO> Alternativa 2: \$ git restore --staged <NOMBRE_ARCHIVO>	Permite revertir la preparación de un archivo específico, es decir, el archivo ya no se está rastreando y se devuelve al área de trabajo.
\$ git restore <NOMBRE_ARCHIVO>	Permite deshacer cambios en un archivo que se encuentra en el área local.

Crear un commit

Comando	Comentario
\$ git log	Muestra el historial de commits
\$ git log --oneline	Muestra todos los commits de forma resumida
\$ git log -p	Permite visualizar el commit y los cambios específicos realizados entre las versiones del archivo.
\$ git commit -m "MENSAJE_COMMIT"	Permite realizar un commit desde la consola de comandos. El archivo está en el repositorio

Crear un commit

Comando	Comentario
\$ git commit	Permite realizar un commit utilizando el editor configurado, por defecto es Visual Studio Code. Se debe completar salvando (Ctrl+S) y cerrando la subventana
\$ git config --global core.editor "code --wait"	Permite definir a Visual Studio Code como editor

Modificar y reversar un commit

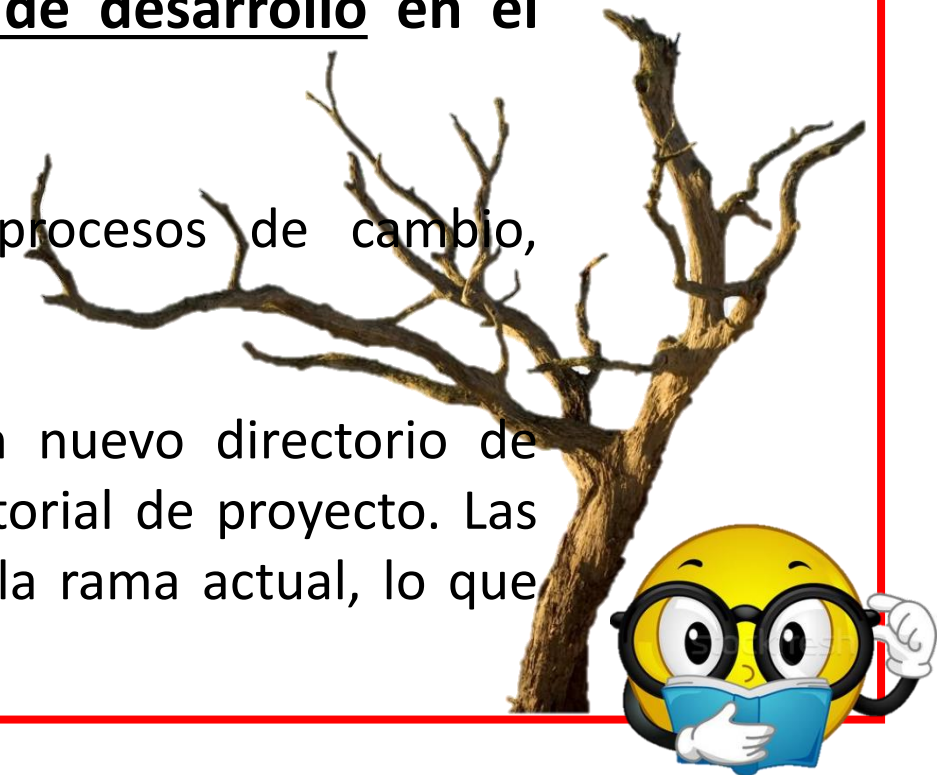
Comando	Comentario
\$ git commit --amend	Permite modificar el mensaje del último commit (edición). Nota: debe ejecutarse en el repositorio local antes de compartir (update)
\$ git reset --soft HEAD~1	Elimina el registro del último commit sin afectar el contenido del archivo

VII. LAS RAMAS (BRANCHES) EN GIT

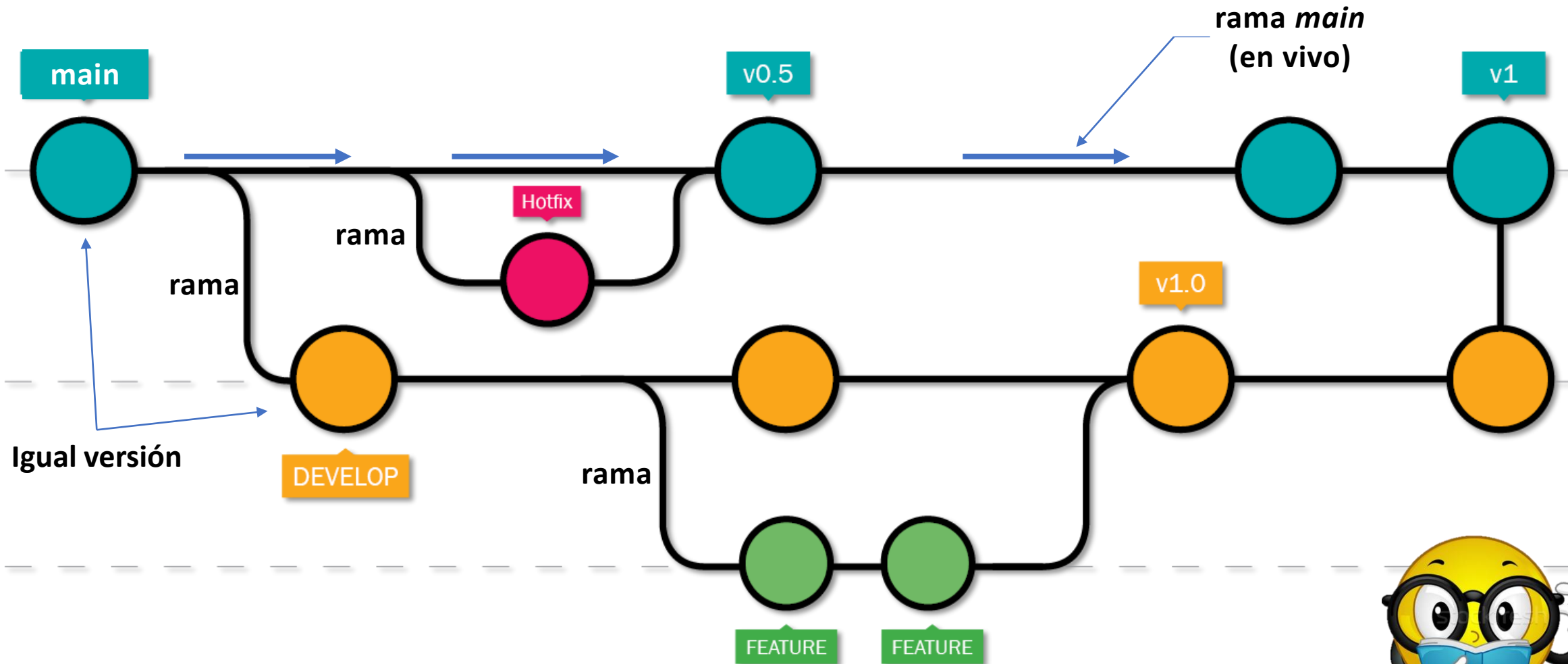
Ramas en Git

RAMA (*BRANCH*)

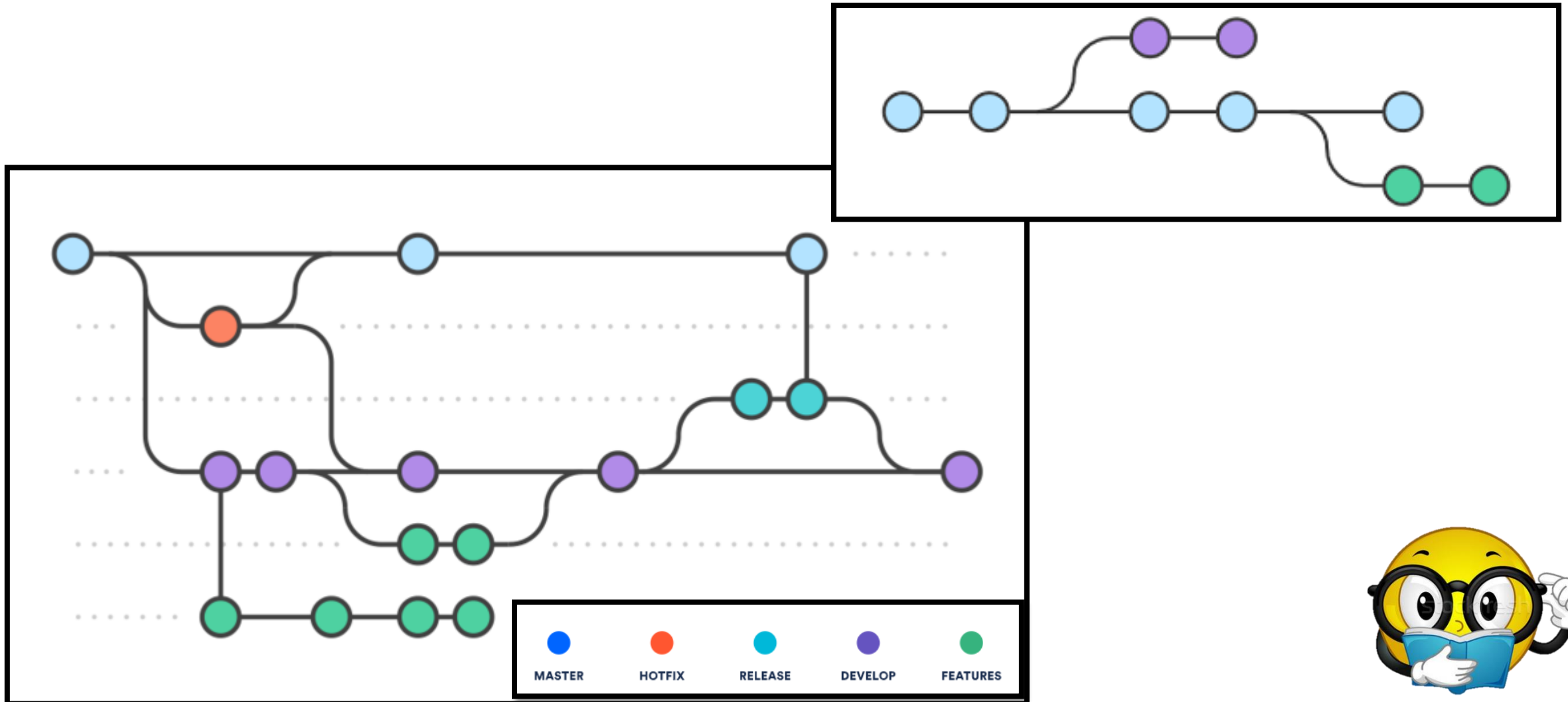
- ❑ Una RAMA en Git es una línea independiente de desarrollo en el repositorio.
- ✓ Las ramas sirven como una abstracción de los procesos de cambio, preparación y confirmación.
- ✓ Puedes concebirlas como una forma de solicitar un nuevo directorio de trabajo, un nuevo entorno de ensayo o un nuevo historial de proyecto. Las nuevas confirmaciones se registran en el historial de la rama actual, lo que crea una bifurcación en el historial del proyecto.



Ramas en Git



Ramas en Git



Crear una rama y cambiar entre ellas

Comando	Comentario
\$ git branch NOMBRE_DE_LA_RAMA	Permite crea una rama nueva
\$ git branch	Muestra las ramas existentes. Nota: '*' indica la rama en la que se está
\$ git checkout NOMBRE_DE_LA_RAMA	Permite ir a una rama especificada

Crear una rama y cambiar entre ellas

Comando	Comentario
Alternativa 1: \$ git checkout -b NOMBRE_DE_LA_RAMA	Permite crea una nueva rama desde la rama donde se encuentra y se ubica en ella.
Alternativa 2: \$ git switch -c NOMBRE_DE_LA_RAMA	Esta instrucción realiza las acciones de los comandos: git branch NOMBRE_DE_LA_RAMA y git checkout NOMBRE_DE_LA_RAMA

Cambiar nombre y eliminar una rama

Comando	Comentario
\$ git branch -m NUEVO_NOMBRE_RAMA	Permite cambiar el nombre de la rama. Nota: hay que estar en la rama a la cual se le quiere modificar el nombre
\$ git branch -m NOMBRE_DE_LA_RAMA NUEVO_NOMBRE_RAMA	Permite cambiar el nombre de la rama sin necesidad de estar ubicado en la rama a modificar.
\$ git branch -d NOMBRE_DE_LA_RAMA	Permite eliminar la rama especificada en el repositorio local. Se debe ejecutar desde una rama diferente a la que se quiere eliminar, puede ser 'main' (sugerido)

Fusionar/combinar ramas

Comando	Comentario
\$ git merge NOMBRE_RAMA	Permite realizar la combinación de la rama indicada en NOMBRE_RAMA, resaltando que se debe estar en la rama que recibe.
\$ git merge --continue	Cuando se está el proceso de validación de conflictos, al cerrar la subventana posterior a la validación del mensaje de commit se requiere dar continuidad a la ejecución de los cambios por parte del editor.

Resumen de comandos...

Comando Básicos (resumen)

Comando	Comentario
<code>pwd</code>	Muestra el directorio de trabajo actual.
<code>mkdir DIRECTORY</code>	Permite crear un nuevo directorio o carpeta en el sistema de archivos.
<code>touch FILE</code>	Se utiliza para crear archivos vacíos y cambiar marcas de tiempo de archivos o carpetas
<code>cd DIRECTORY</code>	Permite cambiar el directorio en el cual nos encontramos trabajando para ir a otro según sea la necesidad
<code>cd ..</code>	Permite regresar al directorio o carpeta anterior.

Comando Básicos (resumen)

Comando	Comentario
cd o cd ~	Nos lleva a la ruta raíz del usuario
cd /D/	Permite pasar al disco D:\
cd -	Nos lleva directamente al último directorio visitado
ls -l	Permite ver todos los archivos como una lista en donde incluye el usuario, grupo, permisos sobre el archivo, tamaño, fecha y hora de creación (formato largo).

Comando Básicos (resumen)

Comando	Comentario
ls -R	Muestra el contenido de todos los subdirectorios de forma recursiva (árbol), es decir, se muestra el contenido de todos los directorios y subdirectorios de manera descendente.
rm -r DIRECTORY	Permite eliminar la carpeta y los archivos dentro de ella de forma recursiva.
rm -d DIRECTORY	Borra directorios vacíos
rmdir DIRECTORY	Permite eliminar permanentemente un directorio o subdirectorio <u>vacío</u>

Comando Básicos (resumen)

Comando	Comentario
history	Permite ver los últimos comandos que se han ejecutado y un número especial con el que podemos volver a repetir el comando. El sistema listará hasta 500 comandos ejecutados.
history -c	Borra toda la lista del historial
clear	Se utiliza para limpiar la pantalla de la terminal, eliminando todo el contenido anterior y dejando la pantalla en blanco. Es importante mencionar que este comando no elimina el historial de comandos anteriores.

Comando Básicos (resumen)

Comando	Comentario
date	Permite visualizar la fecha y hora del sistema
exit	Permite salir de la terminal, cerrando su ventana

Configurar usuario y correo electrónico

Comando	Comentario
\$ git config --global user.name "NOMBRE_APELLIDO"	Permite definir o modificar el nombre del usuario
\$ git config user.name	Muestra el nombre usuario actual
\$ git config --global user.email "E-MAIL_INSTITUCIONAL"	Permite definir o modificar el correo electrónico
\$ git config user.email	Muestra el correo electrónico actual
\$ git config --list	Muestra el resumen de la configuración

Crear un repositorio

Comando	Comentario
\$ git init	Convierte a la carpeta que alojará el proyecto en un repositorio (,git)
\$ git config --global init.defaultBranch main	Permite definir el nombre de la rama principal ('master' → 'main')
\$ git status	Permite revisar el estado del repositorio (área vs. estado del archivo)

Crear un commit

Comando	Comentario
\$ git add NOMBRE_ARCHIVO	Permite llevar el archivo especificado al área de preparación.
\$ git add .	Se indica que todos los cambios serán considerados para el siguiente commit.
Alternativa 1: \$ git rm --cached <NOMBRE_ARCHIVO> Alternativa 2: \$ git restore --staged <NOMBRE_ARCHIVO>	Permite revertir la preparación de un archivo específico, es decir, el archivo ya no se está rastreando y se devuelve al área de trabajo.
\$ git restore <NOMBRE_ARCHIVO>	Permite deshacer cambios en un archivo que se encuentra en el área local.

Crear un commit

Comando	Comentario
\$ git log	Muestra el historial de commits
\$ git log --oneline	Muestra todos los commits de forma resumida
\$ git log -p	Permite visualizar el commit y los cambios específicos realizados entre las versiones del archivo.
\$ git commit -m "MENSAJE_COMMIT"	Permite realizar un commit desde la consola de comandos. El archivo está en el repositorio

Crear un commit

Comando	Comentario
\$ git commit	Permite realizar un commit utilizando el editor configurado, por defecto es Visual Studio Code. Se debe completar salvando (Ctrl+S) y cerrando la subventana
\$ git config --global core.editor "code --wait"	Permite definir a Visual Studio Code como editor

Modificar y reversar un commit

Comando	Comentario
\$ git commit --amend	Permite modificar el mensaje del último commit (edición). Nota: debe ejecutarse en el repositorio local antes de compartir (update)
\$ git reset --soft HEAD~1	Elimina el registro del último commit sin afectar el contenido del archivo

Crear una rama y cambiar entre ellas

Comando	Comentario
\$ git branch NOMBRE_DE_LA_RAMA	Permite crea una rama nueva
\$ git branch	Muestra las ramas existentes. Nota: '*' indica la rama en la que se está
\$ git checkout NOMBRE_DE_LA_RAMA	Permite ir a una rama especificada

Crear una rama y cambiar entre ellas

Comando	Comentario
Alternativa 1: \$ git checkout -b NOMBRE_DE_LA_RAMA	Permite crea una nueva rama desde la rama donde se encuentra y se ubica en ella.
Alternativa 2: \$ git switch -c NOMBRE_DE_LA_RAMA	Esta instrucción realiza las acciones de los comandos: git branch NOMBRE_DE_LA_RAMA y git checkout NOMBRE_DE_LA_RAMA

Cambiar nombre y eliminar una rama

Comando	Comentario
\$ git branch -m NUEVO_NOMBRE_RAMA	Permite cambiar el nombre de la rama. Nota: hay que estar en la rama a la cual se le quiere modificar el nombre
\$ git branch -m NOMBRE_DE_LA_RAMA NUEVO_NOMBRE_RAMA	Permite cambiar el nombre de la rama sin necesidad de estar ubicado en la rama a modificar.
\$ git branch -d NOMBRE_DE_LA_RAMA	Permite eliminar la rama especificada en el repositorio local. Se debe ejecutar desde una rama diferente a la que se quiere eliminar, puede ser 'main' (sugerido)

Fusionar/combinar ramas

Comando	Comentario
\$ git merge NOMBRE_RAMA	Permite realizar la combinación de la rama indicada en NOMBRE_RAMA, resaltando que se debe estar en la rama que recibe.
\$ git merge --continue	Cuando se está el proceso de validación de conflictos, al cerrar la subventana posterior a la validación del mensaje de commit se requiere dar continuidad a la ejecución de los cambios por parte del editor.