Introducción Git y GitHub

Hecho por: Daniel y Rodrigo


if lang is None:
raise Exception("Input language could not be determined")
return None
parsedInput = self.parseInputToLanguageModel(inputString, inputLanguage, context = [])
if not parsedInput or not self.model:
return None
context.append(parsedInput) # Add new conversation entry to context
return (self.model.generateLLMOutput(parsedInput), context)
def parseInputToLanguageModel(inputString, inputLanguage, context):
if self.model is None or self.model.language != inputLanguage:
# LLM is not initalised or has wrong language, load LLM
self.model = self.loadAILanguageModelFromDatabase(inputLanguage)
if self.model is None or not self.runModelSelfDiagnosticTests():
raise Exception("AI language model load failed")
return None
self.model.setLLMContext(context) # Put past conversation context into LLM
llmInputParser = self.model.getInputParser()
llmInputParser.parseInput(inputString)

Introducción

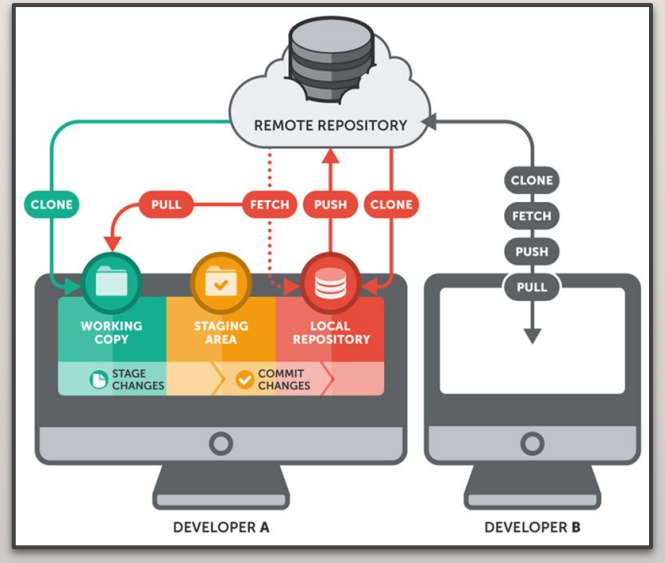
¿Qué vamos a aprender el día de hoy?

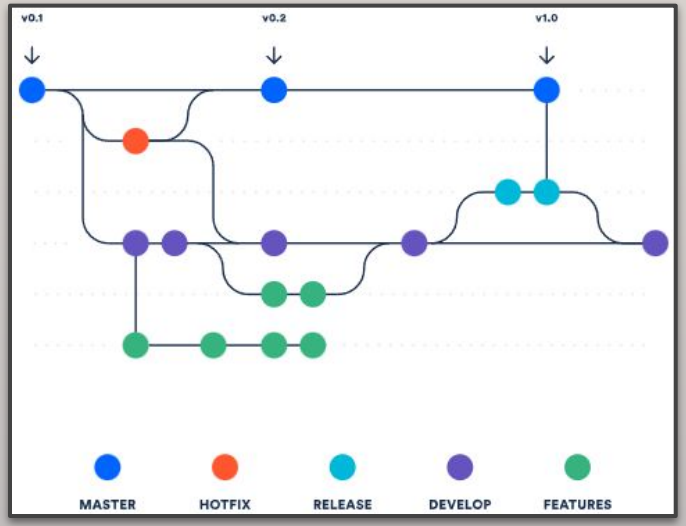
- Sistema de control de versiones - Funcionalidades y Características [1]
  - Git y GitHub - Diferencias [2]
    - Control de versiones - Ventajas y Beneficios [3]
      - Flujo de trabajo - Git y GitHub [4]

Webgrafía/Bibliografía

Sistema de control de versiones - Funcionalidades y Características

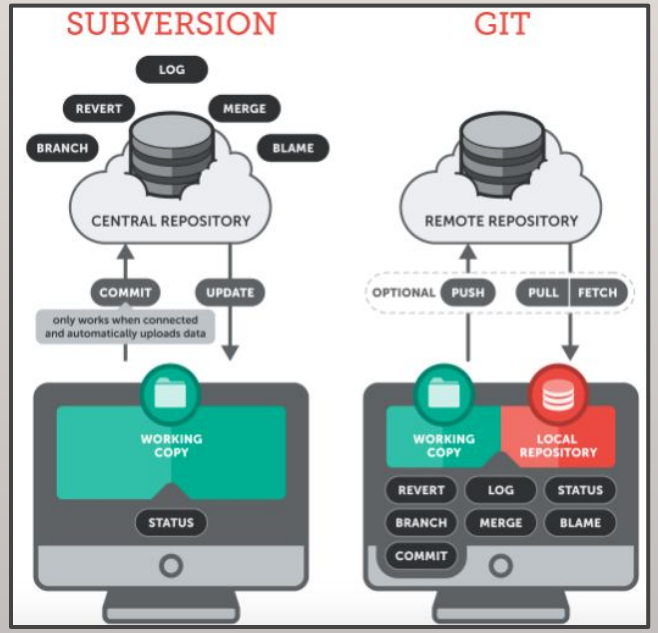
Un sistema de control de versiones es una herramienta que tiene como funciones principales gestionar y monitorizar un archivo o conjunto de estos, proyectos.





También se caracteriza permite crear, añadir, modificar y eliminar archivos y directorios.

Sistema de control de versiones - Funcionalidades y Características



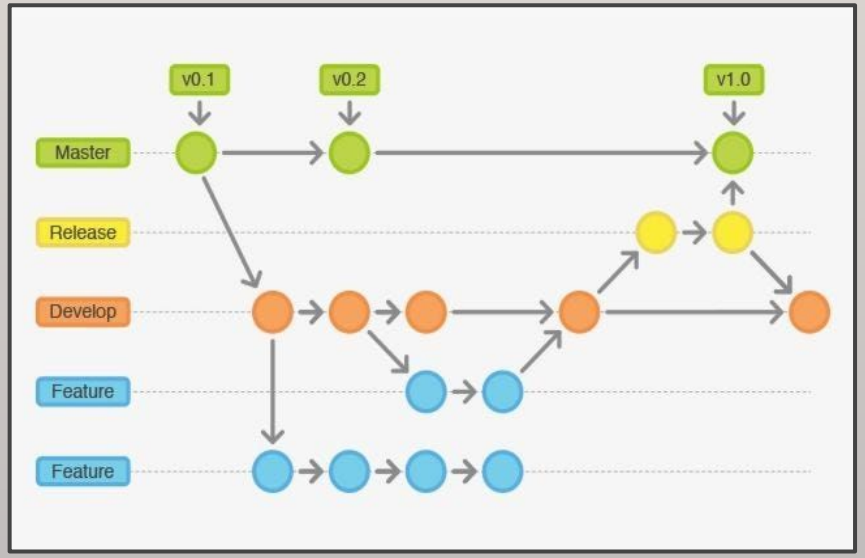
Hay que destacar que una de las características por la que este tipo de sistemas son considerados imprescindibles por los programadores, es la capacidad de trabajo colaborativo.

Sistema de control de versiones - Funcionalidades y Características

Esquema/Resumen visual

Trabajo individual ➡ Commit ➡ Historial de versiones ➡ Compartir con el equipo ➡ Merge de cambios

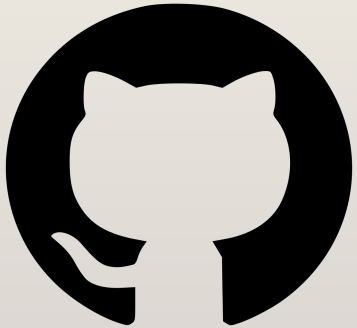
Commit
=
compromiso/confirmación sobre un conjunto de cambios provisionales de forma permanente, es decir, conversión de cambios provisionales a permanentes



Merge
=
Unión de dos o más ramas en una sola

Git y GitHub - Diferencias

GIT

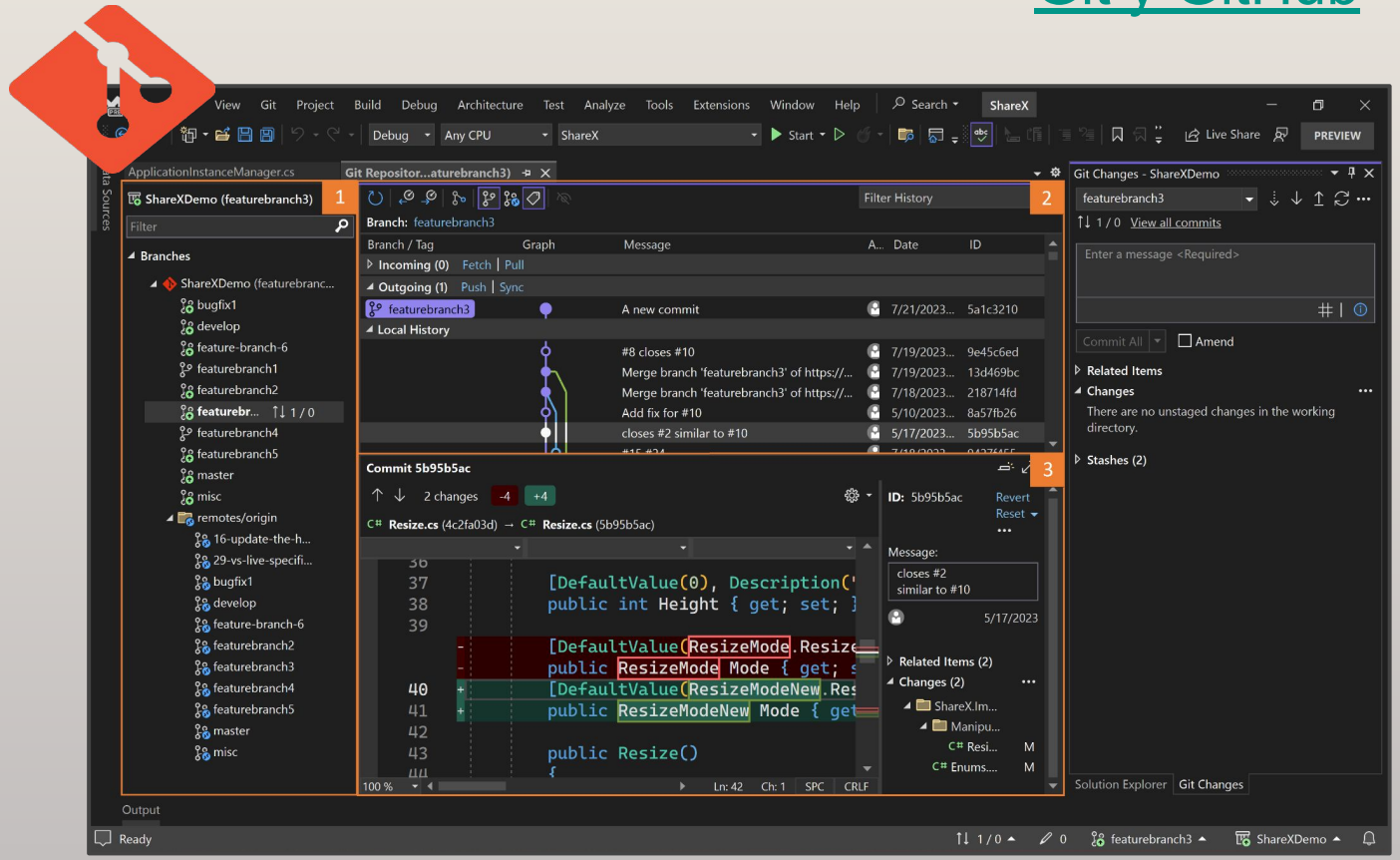


GITHUB

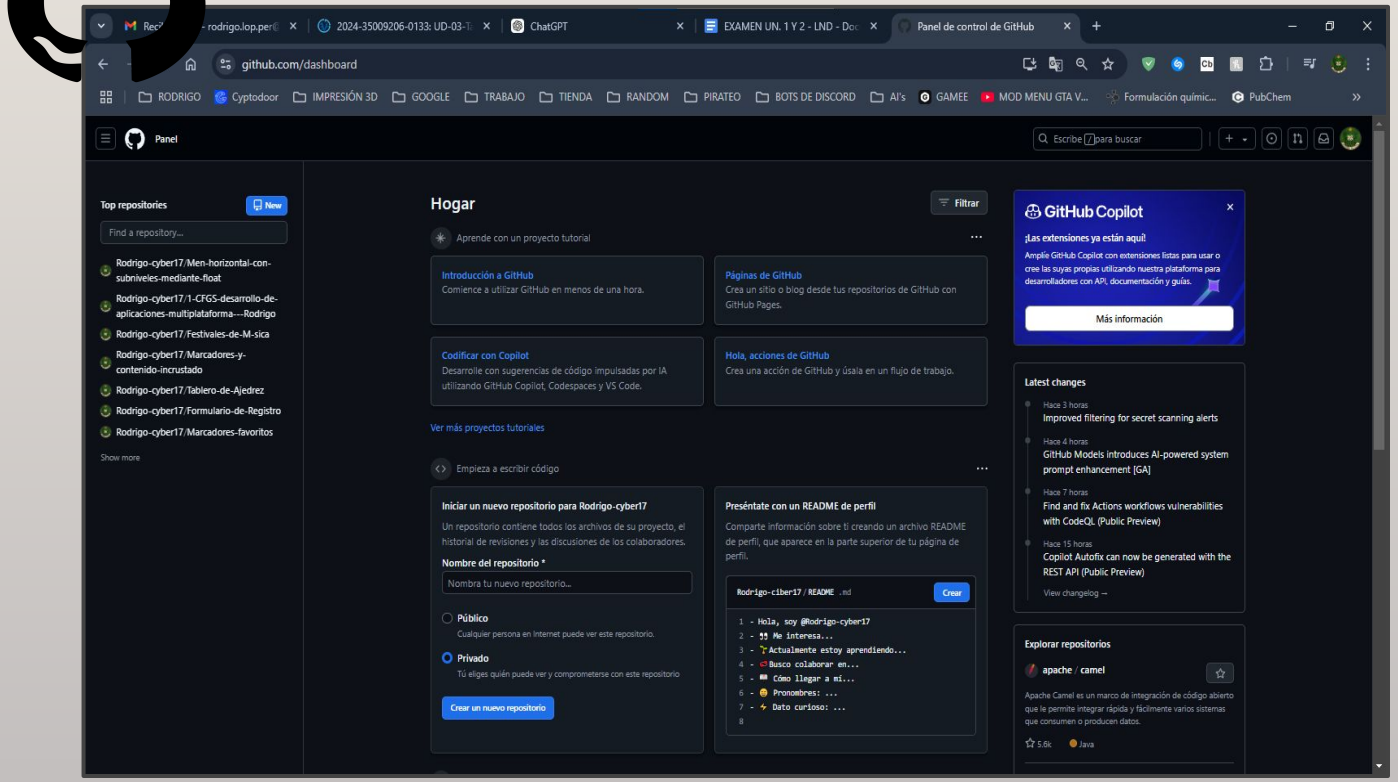
GitHub es una plataforma de desarrollo colaborativo donde se albergan diferentes proyectos utilizando el sistema de control de versiones Git.

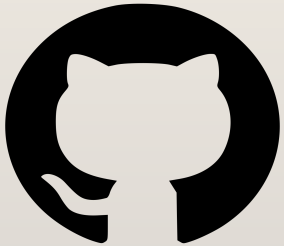
Git es un sistema de control de versiones de código fuente, que, gracias a su diseño eficiente y con capacidad en la contabilidad y computabilidad en el mantenimiento de versiones de aplicaciones hacen posible el trabajo colaborativo y simultáneo de varios programadores en el mismo proyecto.

Git y GitHub - Diferencias



Git y GitHub - Diferencias





Git y GitHub - Diferencias

Resumen

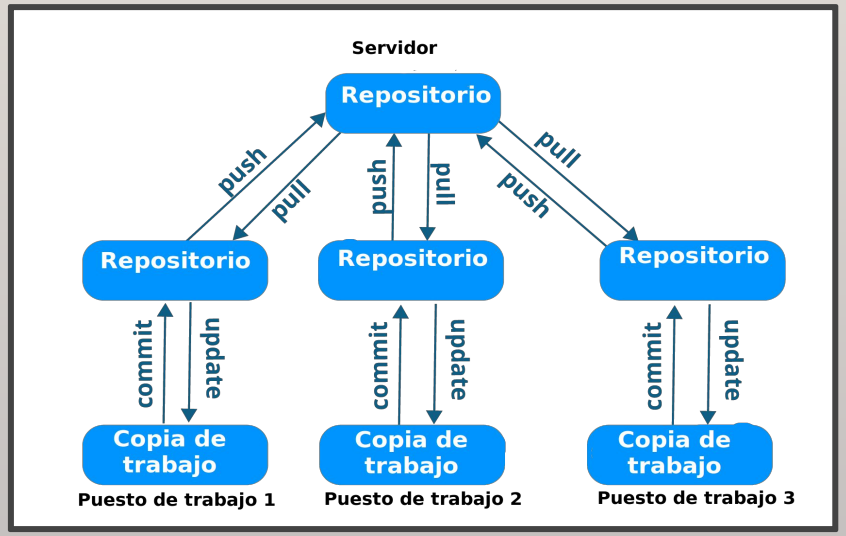
Mientras que Git es la herramienta local encargada de la gestión de versiones y cambios en el proyecto, GitHub es la plataforma en la nube con la que facilitar al programador/es el trabajo en equipo, la colaboración entre equipos y/o miembros de éste y, el alojamiento de repositorios

Esquema visual

Trabajo Local con Git ➡ Sincronización con GitHub (Remoto) ➡ Colaboración con otros desarrolladores

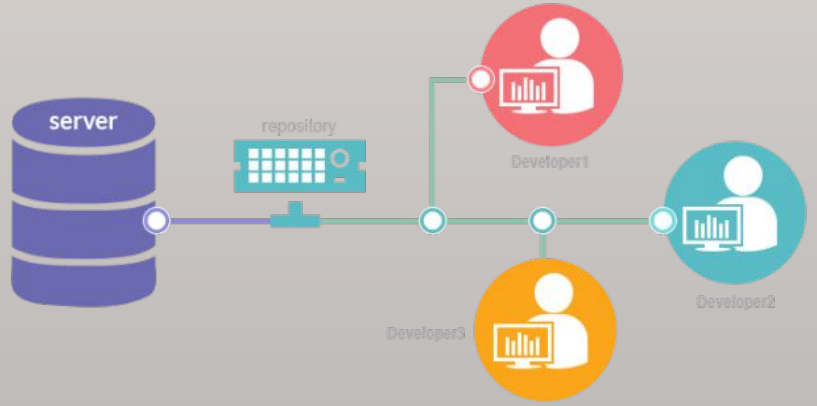
Control de versiones - Ventajas y Beneficios

Un sistema de control de versiones aporta bastantes ventajas al programador a la hora de trabajar. Para ello las dividiremos en 3 puntos clave a mencionar:



- Trabajo en equipo -

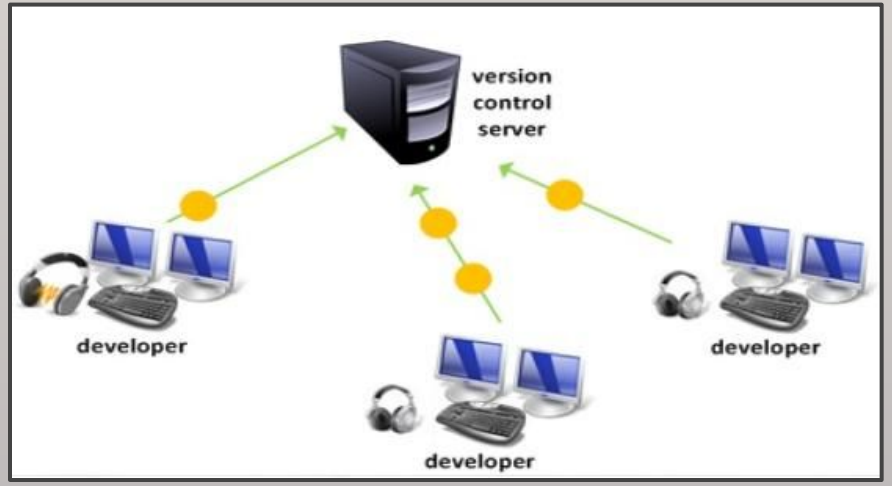
trabajo colaborativo-simultáneo dentro de la gestión de proyectos.

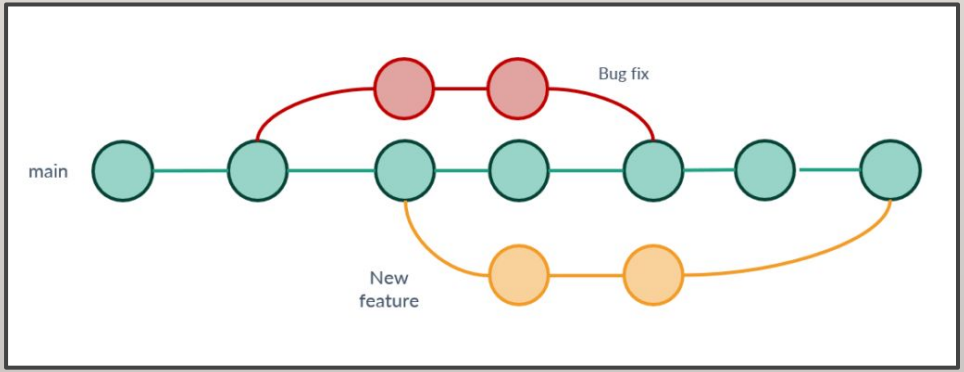


Control de versiones - Ventajas y Beneficios

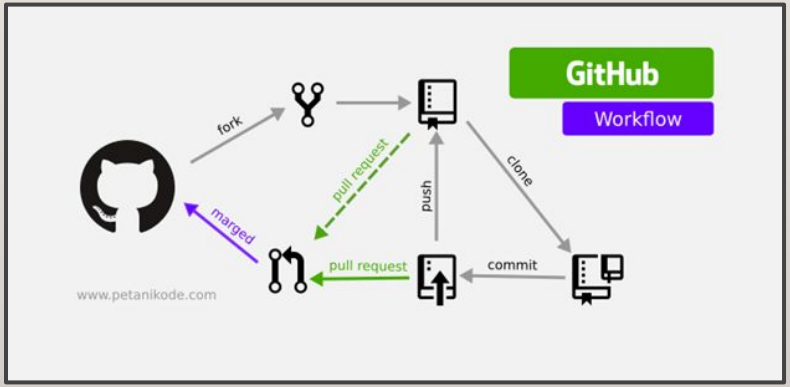
- Historial de cambios -

Hace un guardado sobre el registro completo de las modificaciones, facilitando así el rastreo y la corrección de errores.



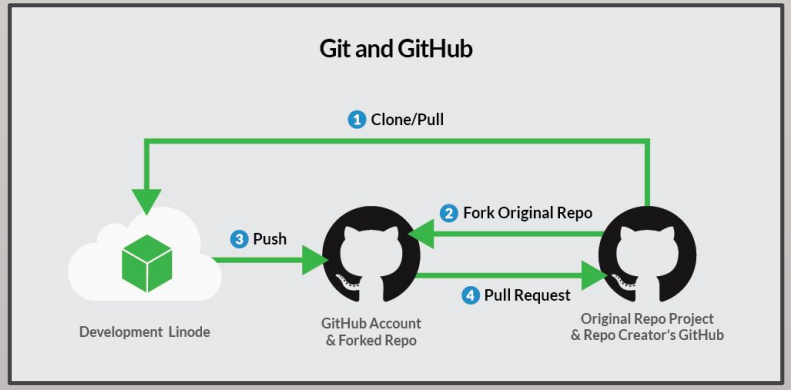


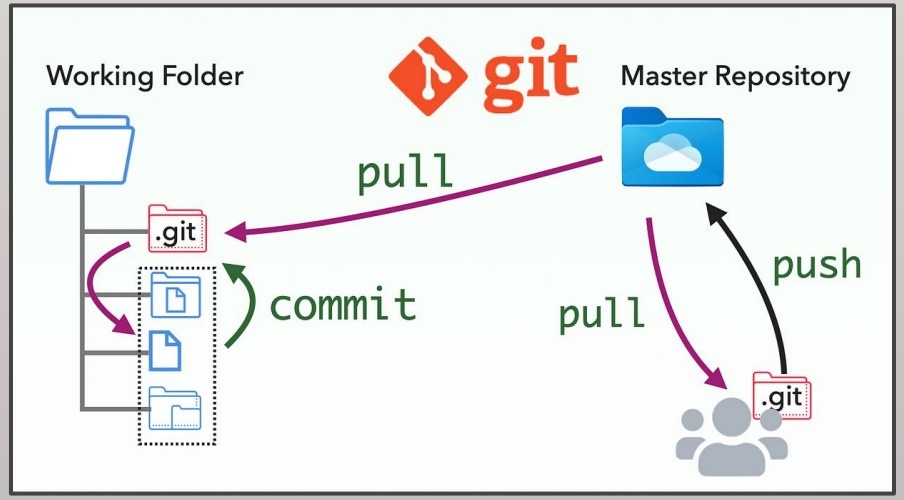
Control de versiones - Ventajas y Beneficios



- Uso de ramas -

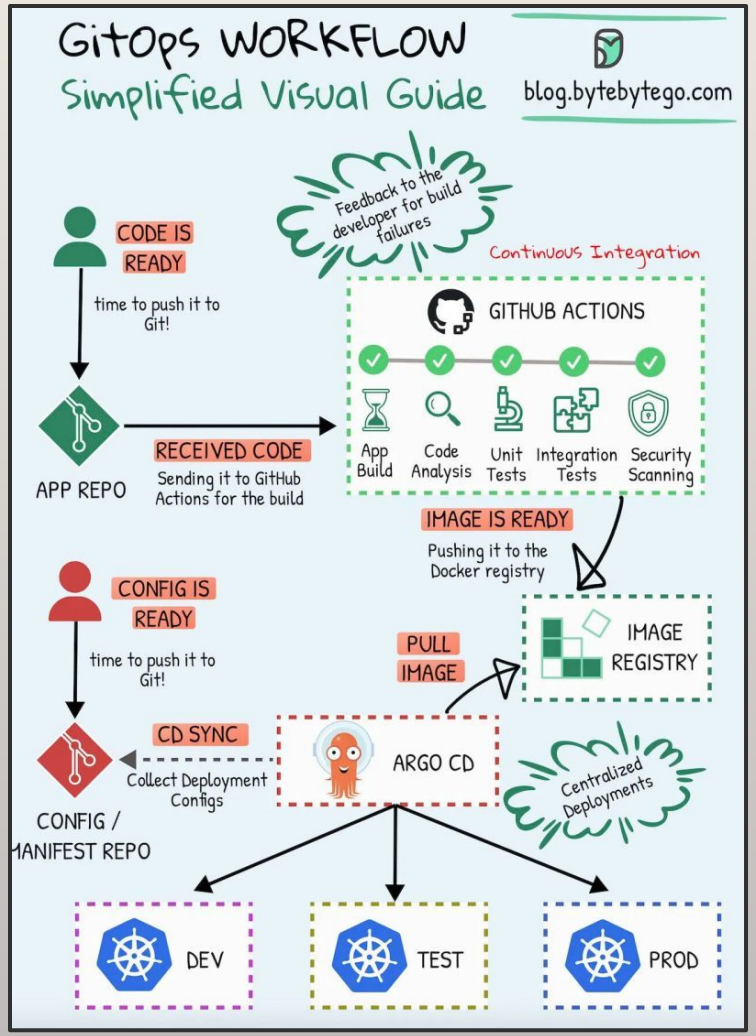
Permite desarrollar nuevas funcionalidades o pruebas sin afectar la versión principal del proyecto.

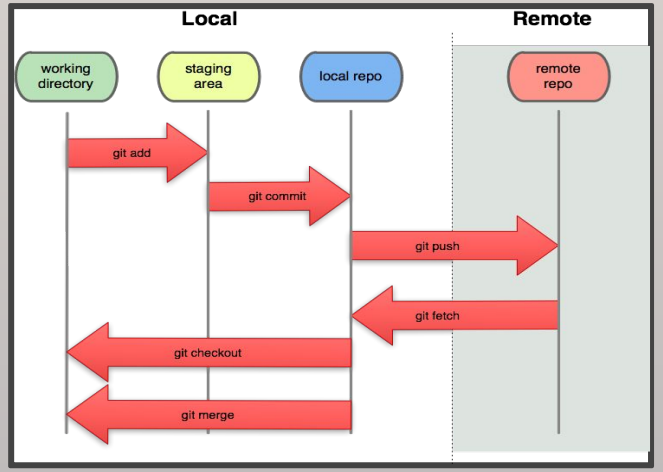




Flujo de trabajo - Git y GitHub

Podemos entonces entender un flujo de trabajo en materia de Git y GitHub mediante lo que denominamos como 6 pilares fundamentales, estos los dividiremos en 2 grupos de 3. Donde los 3 primeros serán sobre la creación y modificación inicial en el desarrollo; mientras que, los otros 3 reflejarán los últimos pasos, el almacenamiento en remoto y el trabajo compartido.





Flujo de trabajo - Git y GitHub

1. Trabajar localmente: Se genera una copia donde cada persona puede realizar cambios sin alterar el proyecto o la versión original de este.

2. Guardar cambios con commits: Se registran los cambios importantes en el historial del proyecto.

3. Ramas para desarrollo paralelo: Se crean ramas separadas para trabajar en nuevas características sin afectar la versión principal. Este punto puede verse semejante o similar al primero.

Flujo de trabajo - Git y GitHub

5. Subir y sincronizar cambios: Los cambios se envían al repositorio remoto en GitHub para ser almacenados y compartirlos.

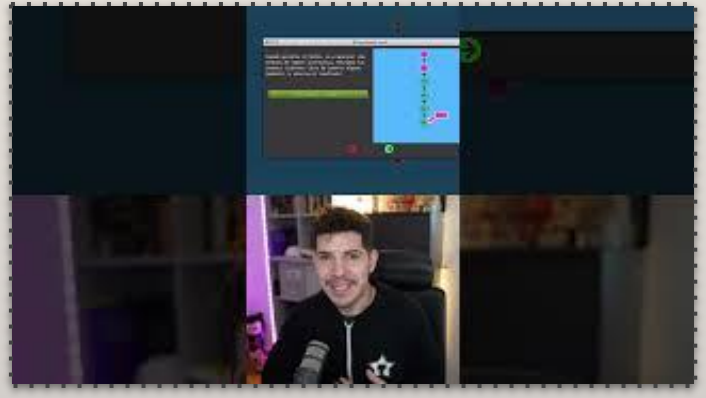
6. Integrar cambios: Se revisan y fusionan los cambios realizados por los colaboradores para obtener un resultado final o previo a este.

7. Mantener un historial: Se obtiene acceso a todas las versiones del proyecto a fines de recuperar cambios base o modificaciones eliminadas sobre versiones más antiguas a la del punto anterior.

Resumen del tema

¿Pero qué es lo que hemos aprendido con esto?

Resumen del tema



Webgrafía/Bibliografía

- [1]
  Wikipedia - Control de versiones
  Bitbucket - Software de control de versiones para equipos profesionales
- [2]
  GIT- Git
  Wikipedia - Git
  GITHUB- Github
  Wikipedia - GitHub
  GitHub - Acerca de GitHub y Git
- [3]
  Atlassian - Qué es el control de versiones
  GitLab - ¿Qué es el control de versiones?
  KeepCoding - ¿Por qué usar sistemas de control de versiones de software?
- [4]

Enlace repositorio GitHub

- https://github.com/Rodrigo-cyber17/Presentaci-n-ETS.git