1. Diferencia Fundamental (Concepto Clave)

Es muy común confundirlos, así que ten clara esta distinción para el examen:

- **Git** es el **software/motor** instalado en tu ordenador (local). Es una herramienta de línea de comandos para el control de versiones distribuido. [04:11]
- **GitHub** es la **plataforma/servicio en la nube** (web). Es una página web que aloja los repositorios de Git y añade funciones sociales y de colaboración (como una red social para código). [05:03]

2. ¿Qué es Git y por qué es especial?

- **Sistema Distribuido:** A diferencia de sistemas antiguos donde todo estaba en un servidor central, en Git **cada desarrollador tiene una copia completa** del historial del proyecto en su máquina. Esto permite trabajar sin conexión (offline). [02:46]
- **Snapshots (Instantáneas):** Git no guarda las diferencias de cada archivo, sino que toma una "foto" (snapshot) completa del estado del código en cada *commit*. [03:34]
- **Origen:** Fue creado por **Linus Torvalds** (creador de Linux) en 2005 por frustración con las herramientas existentes mientras desarrollaba el kernel de Linux. [02:21]

3. Las 3 Áreas de Trabajo de Git (¡Muy Importante!)

El video explica el flujo de vida de un archivo en tres estados:

1. **Working Directory (Área de trabajo):** La carpeta de tu ordenador donde estás editando los archivos actualmente. [04:11]
2. **Staging Area (Área de preparación):** Una zona intermedia donde seleccionas qué cambios quieres incluir en la próxima "foto" o guardado. [04:19]
3. **Repository (Repositorio):** Donde se almacena el historial permanente de cambios (la base de datos de Git). [04:26]

4. Comandos Esenciales para el Examen

Apréndete este flujo de trabajo básico que menciona el video:

- `git init`: Crea un nuevo repositorio (crea la carpeta oculta `.git`). [08:42]
- `git add [archivo]`: Pasa cambios del *Working Directory* al *Staging Area*. [09:03]
- `git commit -m "mensaje"`: Pasa cambios del *Staging Area* al *Repositorio* (crea la instantánea/punto en la historia). [09:11]
- `git push`: Sube tus cambios locales a un repositorio remoto (como GitHub). [09:26]
- `git pull`: Descarga los cambios de otros colaboradores desde el remoto a tu local. [09:33]

5. Ramas (Branches)

- **Concepto:** Son copias aisladas del código para probar cosas nuevas sin romper la versión principal (llamada *main* por convención). [09:42]

- **Comandos:**
    - `git branch [nombre]`: Crea una rama.
    - `git checkout [nombre]`: Cambia a esa rama.
    - `git merge`: Fusiona una rama con otra (ej. traer tus cambios a la rama principal). [10:10]

6. Funcionalidades exclusivas de GitHub (No son de Git)

Estas herramientas **no** existen en Git solo, las aporta la web de GitHub:

- **Pull Requests (PR):** Es una petición para fusionar tu código. Permite que otros revisen tu código, comenten y aprueben antes de mezclarlo. Es estándar en la industria. [06:56]
- **Forking:** Copiar un repositorio ajeno a tu cuenta para experimentar sin afectar al original. [07:45]
- **Issues:** Sistema para reportar errores o discutir tareas. [08:09]
- **GitHub Actions:** Automatización (ej. pasar tests automáticamente). [08:17]

1. ¿Qué es un diagrama UML?

- **Definición:** Es la representación visual del *Lenguaje de Modelado Unificado* (Unified Modeling Language). Sirve para visualizar, especificar, construir y documentar los artefactos de un sistema de software.
- **Propósito:** No es código de programación, sino un "mapa" que muestra la arquitectura, el diseño y la estructura de un sistema complejo. Ayuda a entender cómo interactúan los componentes antes de empezar a programar.
- **Origen:** Creado en los años 90 para estandarizar la ingeniería de software y adoptado como estándar ISO en 2005.

2. Tipos de Diagramas UML

La web menciona que existen 14 tipos, pero se dividen principalmente en dos categorías:

- **Diagramas Estructurales (Estructura Estática):** Muestran las cosas que "existen" en el sistema.
    - *Ejemplo clave:* **Diagrama de Clases** (el más común, muestra clases, atributos y métodos) y **Diagrama de Componentes**.
- **Diagramas de Comportamiento (Dinámicos):** Muestran lo que "ocurre" en el sistema y cómo interactúan los objetos.
    - *Ejemplo clave:* **Diagrama de Casos de Uso** (muestra cómo interactúa el usuario final con el sistema).

3. Ventajas de usarlos

- **Comunicación:** Facilita que tanto perfiles técnicos (programadores) como no técnicos (clientes, gerentes) entiendan el flujo del sistema.
- **Estandarización:** Al ser un lenguaje universal, cualquier desarrollador profesional puede entender el diagrama.

- **Flexibilidad:** Aunque nació para software orientado a objetos, se usa también para modelado de procesos de negocio.

1. El Concepto Principal

El desarrollo de software **no es magia ni solo "picar código"**, es un oficio estructurado. Es una disciplina que busca aplicar métodos para dominar la complejidad. El objetivo final no es solo que el programa funcione, sino que sea fiable, eficiente y seguro.

2. El Ciclo de Vida del Software (El Mapa)

Es la hoja de ruta para no perderse. Se compone de estas fases secuenciales:

- **Planificación:** ¿Merece la pena el proyecto?
- **Análisis (CRÍTICA):** Es la fase más importante. Define **QUÉ** debe hacer el programa (requisitos). Si te equivocas aquí, todo lo que construyas después estará mal.
- **Diseño:** Define el **CÓMO** (traza la ruta, esquemas).
- **Codificación:** Escribir el código en sí.
- **Pruebas:** Asegurar que funciona correctamente.
- **Mantenimiento:** Cuidar y mejorar el software una vez entregado.

3. Estrategias de Desarrollo (La Estrategia)

Existen dos grandes enfoques para recorrer el ciclo de vida:

- **Modelos Clásicos (Predictivos):** Como el modelo en **Cascada**. Son rígidos y secuenciales. Se planifica todo al principio. Ideales si los requisitos no van a cambiar.
- **Metodologías Ágiles:** Como **Scrum**. Son flexibles e iterativas (ciclos cortos). Se adaptan al cambio constante.
    - *Manifiesto Ágil:* Valora más a las personas que a los procesos, el software funcionando sobre la documentación, la colaboración con el cliente y la respuesta ante el cambio.

4. Herramientas: Lenguajes y Paradigmas

Una buena analogía para el examen: **El Lenguaje es el balón, el Paradigma son las reglas del juego**.

- **Paradigma:** Define cómo se estructura y se piensa el código (ej. Orientado a Objetos vs Funcional).
- **Tipos de Lenguajes según comprobación de tipos:**
    - *Estáticos/Fuertes (Java):* Comprueban todo antes de ejecutar. Más seguros, previenen errores.
    - *Dinámicos/Débiles (JavaScript):* Más flexibles y rápidos de escribir, pero con más riesgo de errores en ejecución.
- **Nivel de Abstracción:**

    - *Bajo Nivel (Ensamblador):* Difícil, rápido, cercano a la máquina.
    - *Alto Nivel (Python, Java):* Cercano al lenguaje humano, fácil de usar, portable.

5. El Equipo de Trabajo (Roles)

El desarrollo es un deporte de equipo con roles definidos:

- **Arquitecto de Software:** Tiene la visión global, diseña el esqueleto y cómo se conectan las piezas.
- **Analista de Sistemas:** El "traductor". Habla con el cliente y convierte sus necesidades en requisitos técnicos claros.
- **Programador/Desarrollador:** El constructor. Convierte el diseño y requisitos en código.
- **QA / Testers:** Intentan "romper" el software para encontrar fallos antes que el usuario final.