APUNTES

1.1.1. DEFINICIÓN DE PROGRAMA INFORMÁTICO

- Conjunto de **instrucciones escritas en un lenguaje de programación** que se ejecutan de manera **secuencial** con el objetivo de **realizar una o más tareas en un sistema**.
- Instrucciones divididas en microinstrucciones que se ejecutan en cada ciclo del procesador.
- ¿Qué datos procesa un programa informático?
    - Navegador web:
        - Ordenes de usuario y del servidor.
    - Videojuego:
        - Ubicación de jugadores, impactos, puntuaciones, etc.
    - Programa ofimático:
        - Texto.
        - Datos numéricos.
        - Imágenes.

1.2.1. DEFINICIÓN DE LENGUAJE DE PROGRAMACIÓN

- Es un conjunto de **instrucciones + operadores + reglas de sintaxis y semánticas.**
- La tendencia actual usar **lenguajes de alto nivel**, que son **más cercanos al lenguaje humano.**

1.2.2.1. LENGUAJES DE PRIMERA GENERACIÓN. EL LENGUAJE MÁQUINA

- El lenguaje que entiende el ordenador directamente, estamos hablando a nivel de procesador (**Código máquina(0 y 1)**).
- Hace posible que el programador pueda utilizar todos los recursos del hardware permitiendo así obtener programas muy eficientes.

1.2.2.2. LENGUAJES DE SEGUNDA GENERACIÓN O LENGUAJE ENSAMBLADOR

- Permiten escribir programas muy optimizados que permiten aprovechar al máximo el hardware.
- Dependen directamente del Hardware donde se ejecutan.
- **Assembler** es el primer lenguaje de programación qué ha hecho servir códigos mnemotécnicos.
- Se utiliza para **programar controladores** (drivers) o aplicaciones que requieran un uso muy eficiente de la velocidad y la memoria.

1.2.2.3. LENGUAJES DE TERCERA GENERACIÓN. LOS LENGUAJES DE ALTO NIVEL

- **Lenguajes que hacen servir palabras y frases** relativamente fáciles de entender para expresar flujos de control, por lo que son fáciles de aprender.
- Se usan para desarrollar grandes aplicaciones.
- **Independientes de la máquina en la que se van a ejecutar.**
- Código sencillo y comprensible (Facilidad de ejecución en distintas arquitecturas o SO)
- Un programa escrito en **alto nivel** será **más lento** que haberlo hecho en un lenguaje de bajo nivel.

1.2.2.4. LENGUAJES DE CUARTA GENERACIÓN O DE PROPÓSITO ESPECÍFICO

- Permiten desarrollar **aplicaciones sofisticadas** en poco tiempo.
- Permiten **muchas acciones con una sola instrucción** (Ejp. Realizar consultas a una base de datos con una instrucción como lo sería un Select en SQL).
- Están **orientados al manejo de bbdd.**

1.2.2.5. LENGUAJES DE QUINTA GENERACIÓN

- Lenguajes específicos para **tratar problemas relacionados con la inteligencia artificial.**

1.2.3. EL NIVEL DE ABSTRACCIÓN EN LOS LENGUAJES DE PROGRAMACIÓN

- El nivel de abstracción de un lenguaje **implica lo alejado que está del código máquina. Cuando más parecido sea a nuestro lenguaje y menos al código máquina será de mayor nivel de lenguaje.**
    - **Bajo nivel** - Sólo hay uno el código máquina ceros y unos. •
    - **Medio nivel** - El lenguaje ensamblador que hace servir instrucciones sencillas para trabajar con datos simples y posiciones de memoria.
    - **Alto nive**l - Todos los demás lenguajes de programación son los que son más cercanos a nuestro lenguaje

1.2.4. LOS LENGUAJES DE PROGRAMACIÓN SEGÚN LA FORMA DE EJECUCIÓN

- **Compilados**
    - Compilados antes de poder ejecutarse.
    - **La compilación es el proceso que consigue que el lenguaje de programación baje de nivel hasta el código máquina y sea capaz de ejecutarse.**

- **Interpretados**
    - **Se ejecutan línea a línea** es decir interpreta una línea y realiza la acción que está indica una vez realizada pasa a la siguiente línea y así sucesivamente.
    - **No se necesita compilar el programa completo para ejecutarlo.**
    - **Lo ejecuta un intérprete y no el SO.**
        - El **intérprete** es un programa que **traduce el código de alto nivel a lenguaje máquina y lo hace en tiempo real** va traduciendo y ejecutando cada instrucción una tras otra.
- **Virtuales**
    - **Se compila el código fuente para ejecutarlo en una máquina virtual.**

1.2.5. LOS LENGUAJES DE PROGRAMACIÓN SEGÚN EL PARADIGMA DE PROGRAMACIÓN

- El **paradigma de programación** de un lenguaje de programación se basa en:
    - El **método para llevar a cabo los cálculos** en el proceso.
    - La **forma en la que deben estructurarse las tareas** que debe realizar el programa.

- Se diferencian por **la forma de abstraer los elementos del lenguaje de programación**, así como de **los pasos que se deben seguir para llegar a la solución del problema.**

    - **LENGUAJES IMPERATIVOS O ESTRUCTURADOS**
        - **Sentencias imperativas** (Realiza una operación tras otra y modifican los datos de la memoria).
        - Se hace servir la técnica de la **programación estructurada** (un programa grande y complejo se divide y se representa con **secuencias, selecciones, iteraciones,** etc)
        - Se trabaja **dividiendo el programa en módulos** y así conseguir porciones más pequeñas de **código con tareas específicas**, estos módulos también se dividen y **se crean funciones más pequeñas y reutilizables**.
        - Algunos lenguajes de programación que utilizan este paradigma son:
            - Fortran
            - Java
            - Python
            - Ruby y más

    - **ORIENTADO A OBJETOS**
        - **Intentan abstraer conceptos de la vida real y representarlos con objetos.**
            - Un objeto es una combinación de datos y métodos diseñados para interactuar entre objetos.

    - **FUNCIONAL**
        - **Lenguajes basados en modelos matemáticos.**
        - Se encarga de dividir la app en tareas llamadas funciones. Cada función hace una única cosa y luego se van comunicando entre ellas para, juntas, hacer una tarea mayor.

    - **LÓGICA**
        - **Basada en lógica matemática, usa símbolos lógicos (“and”, “or”, “not”) para describir las relaciones lógicas entre los hechos y las reglas.**

- **CONCEPTOS CLAVE DEL TEMA 1:**
    - Los lenguajes de programación nos facilitan la tarea de programar acercándose al lenguaje humano.
    - Un lenguaje de programación está formado por un conjunto de instrucciones más una serie de operadores y unas reglas de sintaxis y semánticas.
    - Existen diferentes tipos y generaciones de los lenguajes de programación.
    - El nivel de abstracción de los lenguajes de programación los clasifica según lo alejado que está del código máquina. (Bajo, medio y alto)
    - Se pueden clasificar los lenguajes de programación según su forma de ejecución.(Compilados, interpretados y virtuales)
    - El paradigma de los lenguajes de programación indica el método de programación y la forma de programar.

1. **Clasificar lenguajes de programación.**

   1.1. Rellenar la siguiente tabla con diferentes lenguajes de programación.

Tipo de lenguaje	Ejemplos de lenguajes
De primera generación	Lenguaje maquina
De segunda generación	Lenguaje ensamblador
De tercera generación*	C, Pascal
De cuarta generación*	PHP, .NET
De quinta generación	Lisp, Prolog

2.1. ELEMENTOS QUE INTERVIENEN EN EL DESARROLLO DE APLICACIONES

2.1.1. LOS ELEMENTOS MÁS IMPORTANTES

- **Código fuente**
  - **Instrucciones que codifican los programadores.**
  - **Debe cumplir con las normas del lenguaje de programación en el que estemos codificando.**

- **Código Objeto**
  - **Resultado de compilar el código fuente, previamente validado a nivel sitáctico y semántico**

- **Código ejecutable**
  - **Código que se puede ejecutar, resultado de enlazar el código objeto con las librerías.**

2.1.2. HERRAMIENTAS IMPLICADAS PARA LA OBTENCIÓN DE CÓDIGO EJECUTABLE

- **Compilador**
  - Traduce el código fuente a lenguaje máquina.
  - Su objetivo es conseguir el programa ejecutable depurado, ya que detecta posibles errores del código fuente.

- **Máquina virtual**
  - Bytecode es código máquina de bajo nivel que se puede ejecutar en la máquina virtual de java.
  - La máquina interpreta el bytecode y lo adapta al equipo donde se está ejecutando.

2.1.3. ETAPAS DEL PROCESO DE OBTENCIÓN DE CÓDIGO EJECUTABLE

- **Código fuente:**
  - Líneas de texto con los pasos que se deben seguir para ejecutar el programa.

- **Análisis lexicográfico:**

- A partir del código fuente genera una salida compuesta de tokens qué son los componentes léxicos.

- **Análisis sintáctico-semántico:**
  - Se comprueba el texto de entrada en base a una gramática dada, la del lenguaje de programación.

- **Generador de código intermedio:**
  - Es el que transforma el código lenguaje más próximo a la plataforma de ejecución.

- **Optimizador de código:**
  - realiza una serie de transformaciones de mejora del código, se obtiene un código optimizado.

- **Generador de código**:
  - El compilador convierte el programa sintácticamente correcto en una serie de instrucciones que deben ser interpretadas por una máquina. A partir del generador de código se obtiene el código objeto.

- **Enlazador:** programa que a partir del código generado y los recursos necesarios (bibliotecas) quita los recursos que no necesita y enlaza los que necesite al código objeto finalmente genera un código ejecutable.

2.1.4. FASES DE DESARROLLO DE UNA APLICACIÓN

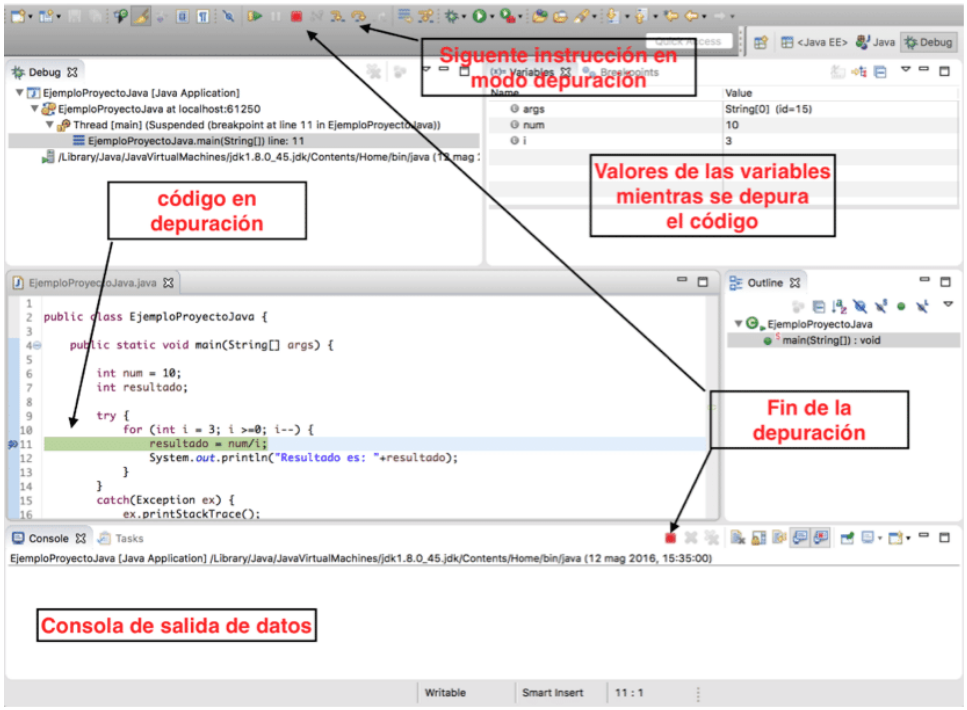
1. **Análisis de requisitos del programa**
   1.1. Se hace un análisis junto con el cliente de lo que necesitamos que haga el programa
2. **Diseño**
   2.1. Determinar cómo funcionará el software de forma global (Diagramas de clases o de comportamiento).
3. **Codificación**
   3.1. Diseño del código.
4. **Pruebas**
   4.1. Se realizan durante y después de la codificación.
   4.2. Verificación de errores y de la labor correcta del software.
5. **Documentación**
   5.1. La documentación es de carácter técnica y está pensada para que sea leída por otros desarrolladores por los clientes usuarios.
6. **Mantenimiento**
   6.1. Se corrigen errores que se detectan cuando el programa ya está implementado y en explotación.
   6.2. Ampliaciones del software o modificaciones también se consideran en el mantenimiento.
7. **Explotación**
   7.1. Se debe preparar el software para su distribución y tener en cuenta las posibles afectaciones que puedan surgir.

3.1.1. DEFINICIÓN DE ENTORNO DE DESARROLLO INTEGRADO (IDE)

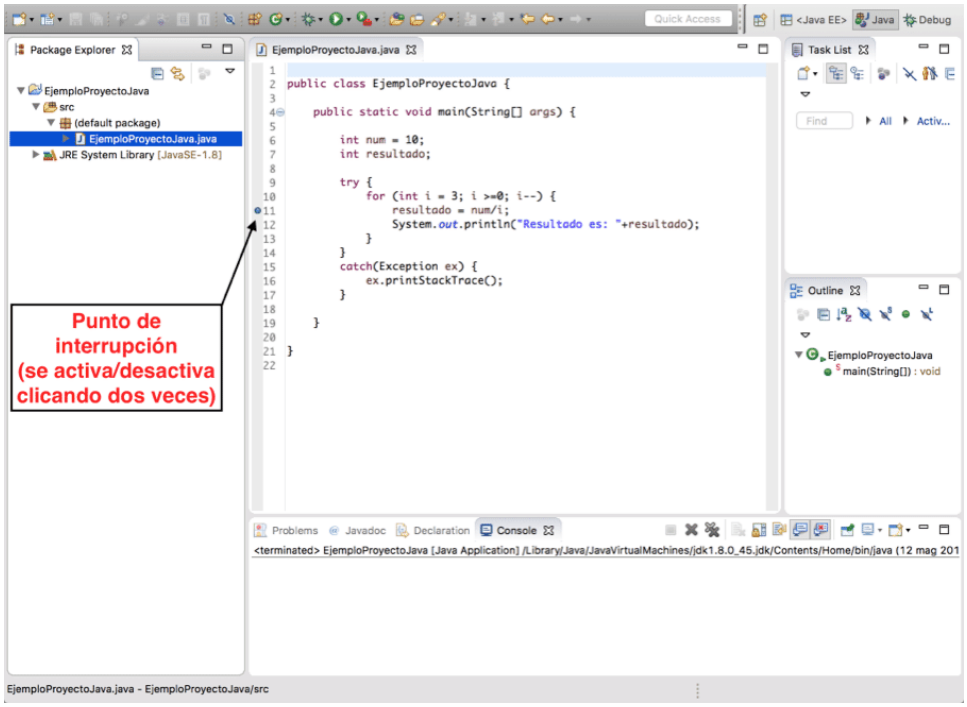
- Es un **programa que integra las herramientas** que necesita un programador para codificar un software de forma cómoda.
- **Incluye un editor de código, un compilador, un intérprete, un depurador y el cliente (para control de versiones).**

3.1.5. LAS DIFERENTES HERRAMIENTAS QUE PODEMOS ENCONTRAR EN UN IDE

- **Compilador**
  - Compila el código fuente de un lenguaje de programación a lenguaje máquina para que pueda ser interpretado por un procesador.
- **Ejecución de forma virtual**
  - El IDE nos permite ejecutar el programa de forma virtual para hacer simulaciones de funcionamiento antes de generar el ejecutable.
  - Para ejecutar de forma virtual nuestro código no debe tener errores de compilación.
- **Depurador**
  - Permite depurar y probar el código fuente del programa y así detectar errores.
  - Ejecutar código línea a línea.
  - Pausar el programa en un momento determinado
  - Manipular los valores de las variables y modificar partes del programa mientras se ejecuta.



- Un Breakpoint es un punto que permite parar la ejecución de un programa a una línea en concreto:
  - Es un punto de interrupción en el código de un programa.
  - Pueden haber varios.
  - Añadir/Quitar un Breakpoint: a través de un click en la línea de código.



- **Control de versiones**
    - Registro histórico de las tareas hechas o versiones del código fuente.
- **Refactorización**
    - Técnica que permite reestructurar el código fuente mejorarlo sin alterar la funcionalidad.
- **Documentación**
    - Generar la documentación del código que lo vamos construyendo como por ejemplo pasa con JavaDoc.
- **Gestor de proyectos**
    - Genera de forma automática todas las dependencias de nuestro código fuente y las localiza.
    - Por ejemplo, cuando generamos una nueva clase en Eclipse, se generará dentro de un proyecto, el gestor se asegurará que quede debidamente enlazado haciendo las llamadas e imports pertinentes.
- **Editor de texto**

4.1.1.2. HERRAMIENTAS PARA HACER LAS PRUEBAS Y DETECTAR ERRORES

- **Depurador**
    - Permite ver el proceso del programa paso a paso.
        - Deja establecer puntos de control (Breakpoints), incluso podemos interrumpir la ejecución del programa. Modificar valores y forzar situaciones para ver otros comportamientos del programa.
    - Modo depurador
        - Resume: Continúa la ejecución del programa hasta el siguiente break point
        - Terminate: Finaliza la ejecución

        - Step into: ejecuta la linea actual
        - Step over: Ejecuta la línea actual y pasa a la siguiente sin entrar en los métodos.
        - Step return: Ejecuta hasta que encuentre un return.
    - Herramientas para la depuración
        - BreakPoint: Detiene la ejecución del programa cuando se alcanza uno.
        - Puntos de seguimiento: BreakPoints que no detienen el programa (Utiles para hacer seguimiento a los valores de variables).
        - Analizador de código: Ayuda a identificar errores en tiempo real.
        - Errores: Aspa de color rojo, no se puede seguir con la compilación si no se corrige.
        - Warnings: Advertencias, no son errores que nos impidan compilar.
    - Errores más comunes que detecta el analizador de código
        - Error de declaración de una variable.
            - Cuando se usa una variable que no se ha declarado.
        - Error de tipo de variable
            - Cuando se mezcla el uso de variables con tipos diferentes.
        - Error de importación
            - Cuando no se ha importado de la librería una función.

- **4.2.2. ¿QUÉ SON LOS CASOS DE PRUEBAS?**
    - Condiciones para determinar si l aplicación funciona correctamente mientras más casos de prueba haya más cerca estaremos de que haya un programa robusto.
        - Pruebas de nivel atómico: funciones y bucles.
        - Integración de las funciones.

- **4.2.3. TIPOS DE PRUEBAS HASTA ABAJO**
    - Pruebas unitarias: Verificar el correcto funcionamiento de una unidad de código.
        - Pruebas de caja blanca: Examinan la lógica interna.
            - Método del camino básico.
                - Para trazar caminos de ejecución
            - Complejidad ciclomática.
            - Caminos independientes.
        - Pruebas de caja negra: Pruebas funcionales, centradas en la entrada y salida, no se verifica el código, la idea es obtener los resultados esperados de los valores de salida.
            - Método de partición equivalente
                - Divide y separa los campos de entrada según el tipo de datos y las restricciones que el tipo impone a estos datos.
                - Se establecen un rango de valores (límites).
    - Pruebas de integración
        - Identifican errores en la comunicación entre funciones o componentes.
        - Posteriores a las pruebas unitarias probando funciones en conjunto
    - Pruebas de sistema
        - Verifican el correcto funcionamiento del sistema completo.
        - Incluyen aspectos como rendimiento, seguridad, usabilidad e instalación.
    - Pruebas de carga

        - Rendimiento del software con una demanda elevada de peticiones.
    - Pruebas de estrés
        - Se simula un escenario de situaciones extremas con tal de ver el comportamiento ante estos escenarios.
    - Pruebas de seguridad
        - Realizadas cuando hay diferentes niveles de permisos en la aplicación.
    - Pruebas de aceptación
        - Validar la expectativas del cliente y de los usuarios.
            - Pruebas alfa: se hacen al final del desarrollo con el usuario final acompañado de los programadores o de alguien relacionado en el desarrollo. El cliente y el desarrollador toman nota juntos de los aspectos a corregir o mejorar.

            - Pruebas beta: A diferencia de las pruebas alfa, estas pruebas pretenden que las haga el usuario final en un entorno no controlado. El cliente prueba el software solo y toma nota de los aspectos a corregir o mejorar, después se los enviará al equipo de desarrollo para que lo mejore.

- **5.1.1.1. QUÉ ES EL REFACTORING**
    - Pequeños cambios en el código para que sea más visible(legibilidad), más flexible(adaptar a otro código), más fácil y más modificable(cambios).
    - Se realiza en código que ya funciona.
    - Se realiza para tener un código funcional, fácil de mantener, reutilizar y de entender.
    - Ventajas
        - Evitar problemas derivados de los cambios de los mantenimientos posteriores
        - Simplificar el diseño
        - Ayuda a entenderlo mejor
        - Será más fácil detectar errores
        - Agiliza la programación

    - Inconvenientes
        - No es fácil de aplicar si se tiene poca experiencia
        - Si se llega a un exceso de querer optimizar el código de una forma obsesiva
        - Dedicar mucho tiempo a la refactorización
        - Posible impacto de la refactorizacion al resto del software

- **5.1.1.3. LOS MALOS OLORES (BAD SMELLS)**
    - Código mal hecho, que se puede mejorar.
        - Método largo: Dividir en funciones o metodos más cortos.
        - Clases largas: Clases más pequeñas.
        - Listas de parámetros largas: Métodos con pocos parametros de entrada.

- **5.2.2. Los comentarios**
    - Frases cortas que pretenden aportar información sobre una parte del código.

- **5.3. CONTROL DE VERSIONES**
  - Herramienta de ayuda al desarrollo de software que se encarga de ir almacenando el estado del código fuente en momentos determinados.
  - Para que sirve:
    - **Compartir** archivos de diferentes programadores.
    - **Bloquear** archivos que se están editando.
    - **Fusionar** archivos con diferentes cambios.

  - Que funcionalidades tiene:
    - **Comparar** cambios en el código fuente.
    - **Coordinar** las tareas entre diferentes programadores
    - **Guardar versiones** anteriores del código fuente.
    - **Seguimiento de los cambios** realizados en el código: con un historial de cambios realizados en el código fuente pudiendo conocer el momento del cambio y el autor.
    - **Restaurar** a una versión de código anterior.
    - **Control de los usuarios**.
    - **Crear ramas (forks)** del proyecto que permiten desarrollar varias versiones de un mismo programa a la vez.
    -
- **5.3.1. PARTES DE UN SISTEMA DE CONTROL DE VERSIONES**
  - **Repositorio**: es donde se almacenan los datos actualizados e históricos de cambios. Normalmente es un servidor.
  - **Módulo:** conjunto de directorios y/o archivos dentro del repositorio que pertenecen a un proyecto común.
  - **Revisión:** es una versión determinada de la información que se almacena.
  - **Etiqueta (Tag):** darle a cada uno de los ficheros del módulo en desarrollo en un momento preciso un nombre común para asegurarse de reencontrar ese estado de desarrollo posteriormente bajo ese nombre.
  - **Rama (branch)**: cuando se genera un duplicado del código fuente sobre el que se va a trabajar dentro de los directorios y/o archivos dentro del repositorio.
  - **Trunk:** rama principal de código fuente.
  - **Merge:** operación de fusión de diferentes ramas.
  - **Commit:** operación de confirmación de cambios en el sistema de control de versiones.
  - **Changeset:** conjunto de cambios que hace un usuario y sobre los que se realiza una operación “Commit” de manera simultánea y que son identificados mediante un número único en el sistema de control de versiones.
- **5.3.2. TIPOS DE SISTEMAS DE CONTROL DE VERSIONES**
  - Locales: Copia de los archivos a otro directorio en el mismo equipo. (Propenso a errores)
    - Fácil olvidar en qué directorio te encuentras.

        - Guardar accidentalmente en el archivo equivocado.
        - Sobrescribir archivos que no querías.
    - Centralizados
        - Único servidor que contiene todos los archivos versionados y varios clientes que descargan los archivos desde ese lugar central. El responsable es un único usuario. Esto hace que todas las acciones importantes necesiten la aprobación del responsable.
    - Distribuidos
        - Cada usuario tiene su propio repositorio en local. Los distintos repositorios se mezclan entre ellos de forma sincronizada para mantener la coherencia.

- **5.3.3. OPERACIONES QUE SE PUEDEN HACER EN UN SISTEMA DE VERSIONES**
    - **Commit** (subir): sube una copia de los cambios hechos en local que se integra sobre el repositorio.
    - **CheckOut** (bajar o desplegar): cuando crea una copia de trabajo local desde el repositorio.
    - **Update** (actualización): cuando se integran los cambios que se han hecho en el repositor en la copia de trabajo local.