



# ENTORNOS DE DESARROLLO

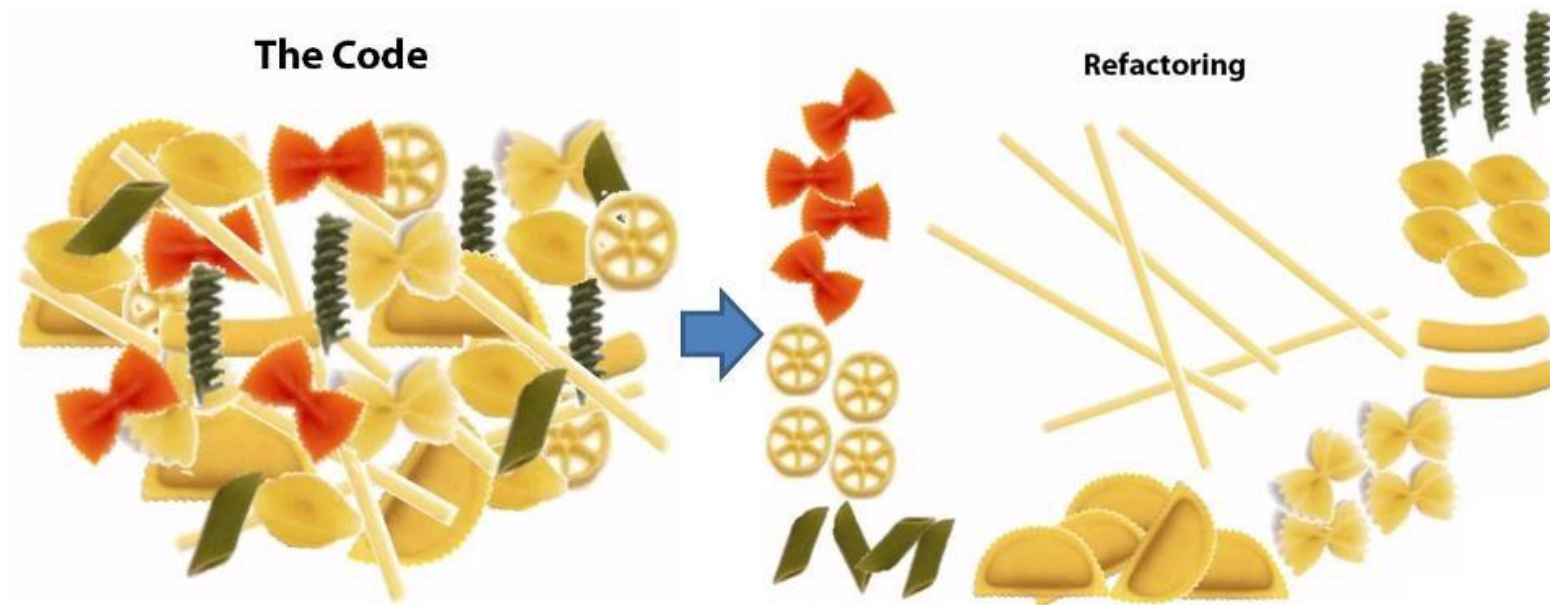
---

TUTORÍA COLECTIVA 11 de Mayo 2021

# Ut6

## Optimización y documentación

---



---

# REFACTORIZACIÓN

# Bad Smells

---

- **Código duplicado** (Duplicated code). Si se detectan bloques de código iguales o muy parecidos en distintas partes del programa.
- **Métodos muy largos** (Long Method). Los métodos de muchas líneas dificultan su comprensión. Un método largo probablemente está realizando distintas tareas, que se podrían dividir en otros métodos. Un método debe hacer solo una cosa, hacerla bien, y que sea la única que haga.
- **Clases muy grandes** (Large class). Una clase debe tener solo una finalidad. Si una clase se usa para distintos problemas tendremos clases con demasiados métodos, atributos e incluso instancias.
- **Lista de parámetros extensa** (Long parameter list). Los métodos deben tener el mínimo número de parámetros posible, siendo 0 lo perfecto. Si un método requiere muchos parámetros puede que sea necesario crear una clase con esa cantidad de datos y pasarle un objeto de la clase como parámetro. Del mismo modo ocurre con el valor de retorno, si necesito devolver más de un dato.

# Bad Smells

---

- **Cambio divergente** (Divergent change). Si una clase necesita ser modificada a menudo y por razones muy distintas, puede que la clase esté realizando demasiadas tareas. Podría ser eliminada y/o dividida.
- **Cirugía a tiros** (Shotgun surgery). Si al modificar una clase, se necesitan modificar otras clases o elementos ajenos a ella para compatibilizar el cambio. Lo opuesto al smell anterior.
- **Envidia de funcionalidad** (Feature Envy). Ocurre cuando una clase usa más métodos de otra clase, o un método usa más datos de otra clase, que de la propia.
- **Legado rechazado** (Refused bequest). Cuando una subclase extiende (hereda) de otra clase, y utiliza pocas características de la superclase, puede que haya un error en la jerarquía de clases.

<https://sourcemaking.com/refactoring>

# Refactorización

---

**Refactorización:** técnica de la ingeniería de software para reestructurar un código fuente, alterando su estructura interna **sin cambiar su comportamiento** externo.

- Crear un código más claro y sencillo, más limpio.
- No arregla errores. No añade funcionalidad. Facilita cambios en el código.

<https://sourcemaking.com/refactoring>

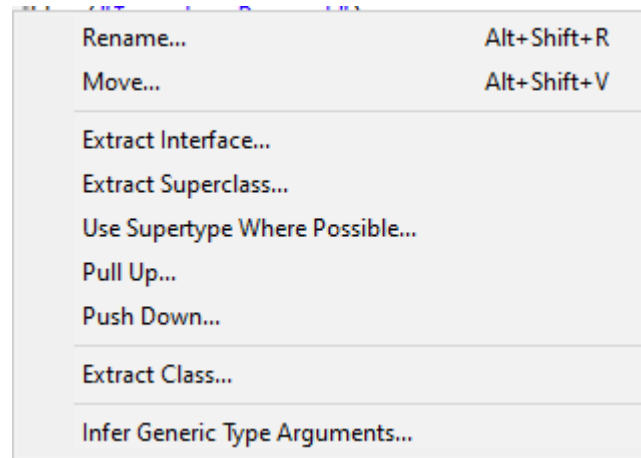
# Refactorización de Eclipse



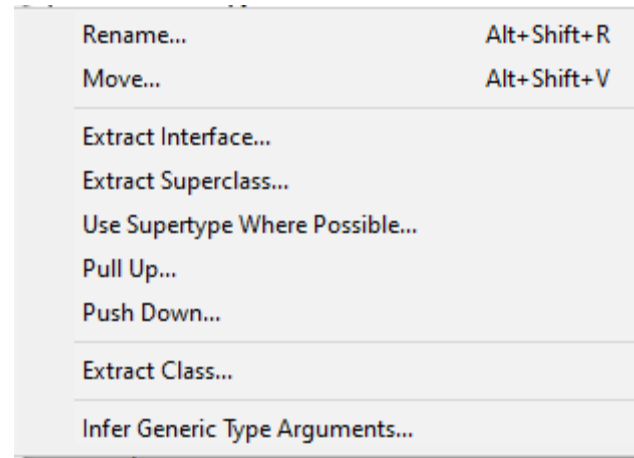
Eclipse tiene distintos métodos de refactorización. Dependiendo de sobre qué mostremos el menú de refactorización, nos ofrecerá unas u otras opciones.

Opción **Refactor** del menú contextual:

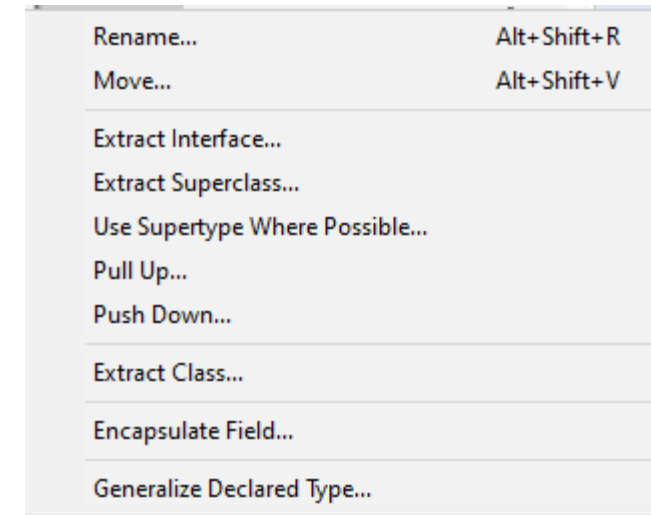
## CLASE



## MÉTODO

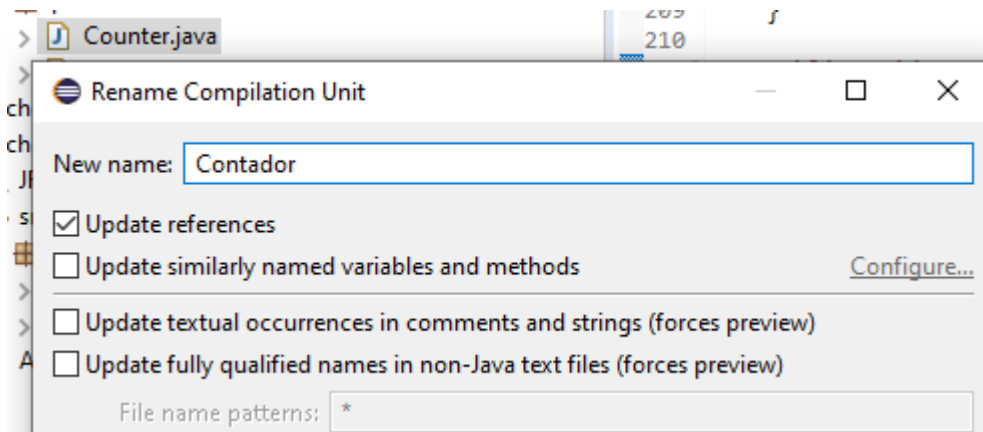


## ATRIBUTO



# Rename (Alt + Shift + R)

- Opción más utilizada. Cambia el nombre de variables, clases, métodos, paquetes, directorios y casi cualquier identificador Java.
- Tras la refactorización, se modifican las referencias a ese identificador.

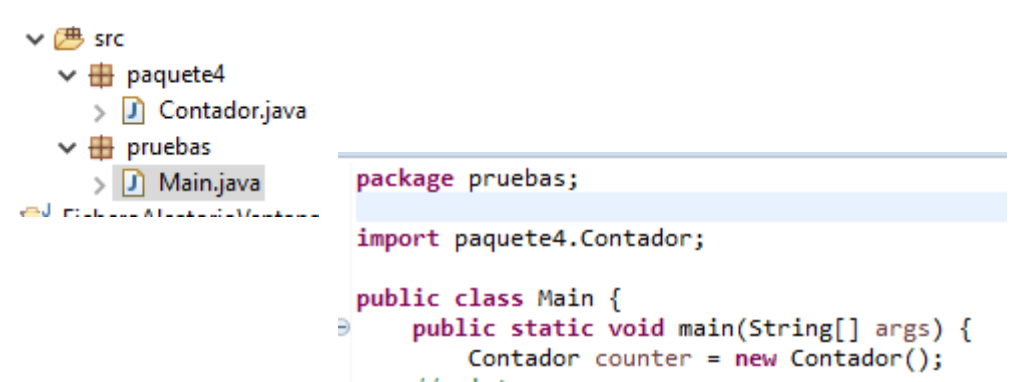
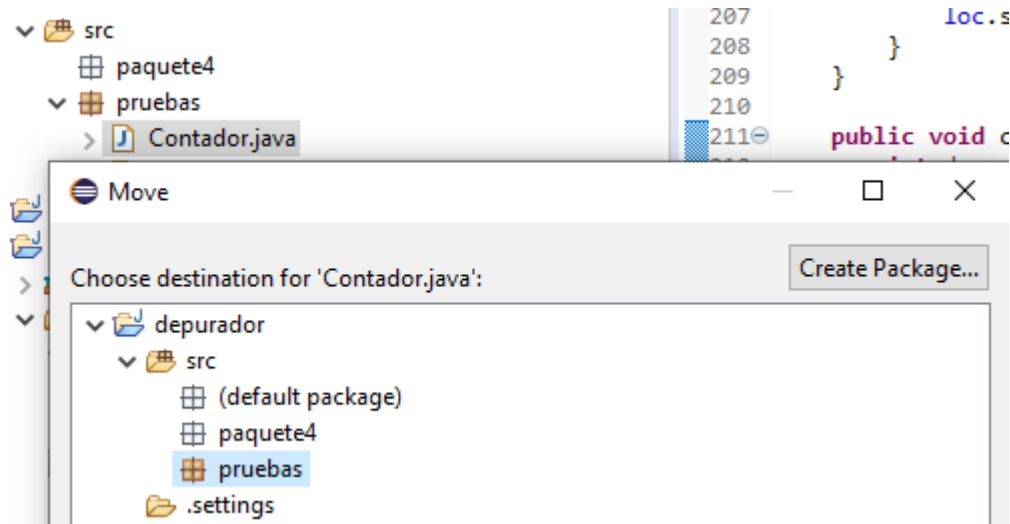


```
public class Contador {  
    private int result = 0;  
    public int getResult() {  
        return result;  
    }  
    ...  
    ...  
    ...  
    public static void main(String[] args) {  
        Contador counter = new Contador();  
        ...  
    }  
}
```



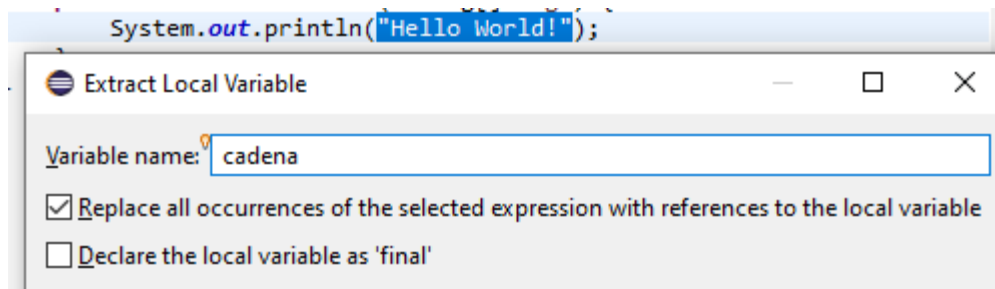
# Move (Alt + Shift + V)

- Mover una clase de un paquete a otro. Se mueve el archivo .java a la carpeta. Se cambian todas las referencias.
- Arrastrar y soltar una clase a un nuevo paquete. Refactorización automática.



# Extract Local Variable (Alt + Shift + L)

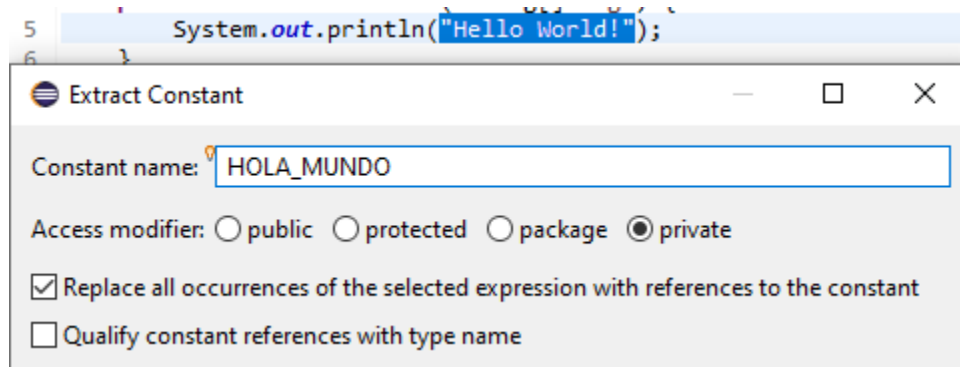
- Convierte un número o cadena literal en una variable de ámbito local.
- Tras la refactorización, cualquier referencia a la expresión en el ámbito local se sustituye por la variable.
- La misma expresión en otro método no se modifica.



```
public class ExtractLocalVariable {  
    public static void main(String[] args) {  
        System.out.println("Hello World!");  
    }  
}  
  
public class ExtractLocalVariable {  
    public static void main(String[] args) {  
        String cadena = "Hello World!";  
        System.out.println(cadena);  
    }  
}
```

# Extract Constant

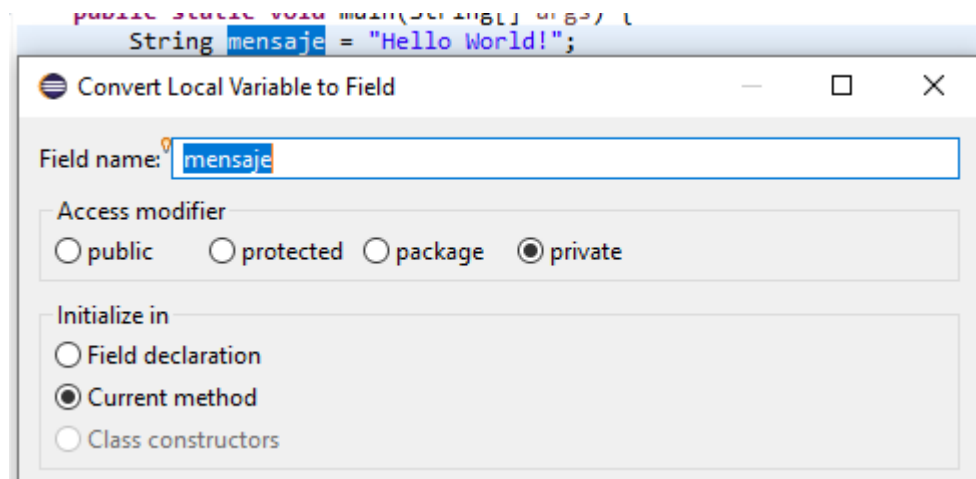
- Convierte un número o cadena literal en una constante.
- Tras la refactorización, todos los usos del literal se sustituyen por esa constante.
- Objetivo: Modificar el valor del literal en un único lugar.



```
public class ExtractConstant {  
    public static void main(String[] args) {  
        System.out.println("Hello World!");  
    }  
}  
  
public class ExtractConstant {  
    private static final String HOLA_MUNDO = "Hello World!";  
  
    public static void main(String[] args) {  
        System.out.println(HOLA_MUNDO);  
    }  
}
```

# Convert Local Variable to Field

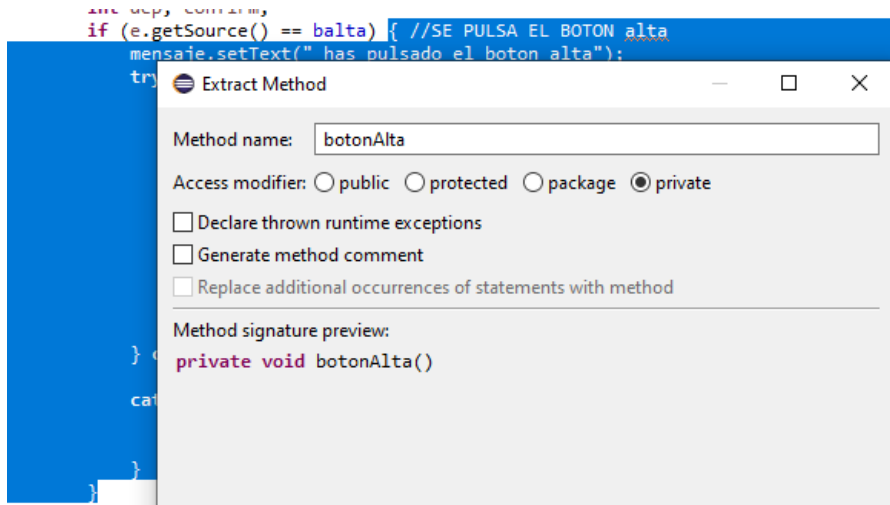
- Convierte una variable local en un atributo privado de la clase.
- Tras la refactorización, todos los usos de la variable local se sustituyen por ese atributo.



```
public class ConvertLocalVariable {  
    public static void main(String[] args) {  
        String mensaje = "Hello World!";  
        System.out.println(mensaje);  
    }  
}  
  
public class ConvertLocalVariable {  
    private static String mensaje;  
  
    public static void main(String[] args) {  
        mensaje = "Hello World!";  
        System.out.println(mensaje);  
    }  
}
```

# Extract Method (Alt + Shift + M)

- Convierte un bloque de código en un método, a partir de un bloque cerrado por {}.
- Eclipse ajustará automáticamente los parámetros y el retorno del método.
- Muy útil cuando detectamos *bad smells* en métodos muy largos o en bloques de código que se repiten.



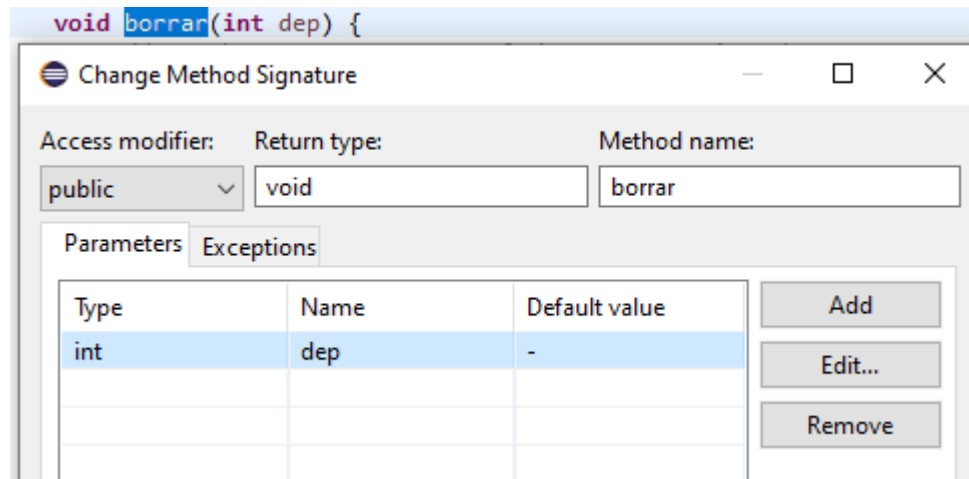
```
if (e.getSource() == balta) { //SE PULSA EL BOTON alta
    mensaje.setText(" has pulsado el boton alta");
    try {
        dep=Integer.parseInt(num.getText());
        if (dep >0)
            if (consultar(dep))
                mensaje.setText(" DEPARTAMENTO EXISTEEE");
            else {
                mensaje.setText(" NUEVO DEPARTAMENTO");
                grabar(dep, nombre.getText(), loc.getText());
                mensaje.setText(" NUEVO DEPARTAMENTO GRABADO");
            }
        else mensaje.setText(" DEPARTAMENTO DEBE SER MAYOR QUE 0");
    } catch (java.lang.NumberFormatException ex) { //controlar el error del Int
        mensaje.setText(" DEPARTAMENTO ERRÓNEO");
    } catch (IOException ex2) {
        mensaje.setText(" ERRORRR EN EL FICHERO. Fichero no existe. (ALTA)");
        // lo creo
    }
}
```

```
if (e.getSource() == balta)
    botonAlta();

private void botonAlta() {
    int dep;
    { //SE PULSA EL BOTON alta
        mensaje.setText(" has pulsado el boton alta");
        try {
            dep=Integer.parseInt(num.getText());
            if (dep >0)
                if (consultar(dep))
                    mensaje.setText(" DEPARTAMENTO EXISTEEE");
                else {
                    mensaje.setText(" NUEVO DEPARTAMENTO");
                    grabar(dep, nombre.getText(), loc.getText());
                    mensaje.setText(" NUEVO DEPARTAMENTO GRABADO");
                }
            else mensaje.setText(" DEPARTAMENTO DEBE SER MAYOR QUE 0");
        } catch (java.lang.NumberFormatException ex) { //controlar el error
            mensaje.setText(" DEPARTAMENTO ERRÓNEO");
        } catch (IOException ex2) {
            mensaje.setText(" ERRORRR EN EL FICHERO. Fichero no existe. (
            // lo creo
        }
    }
}
```

# Change Method Signature

- Nos permitirá cambiar la firma de un método. La firma de un método es única y está compuesta por nombre y parámetros.
- De forma automática se actualizarán todas las dependencias y llamadas al método dentro del proyecto actual.



# Inline (Alt + Shift + I)

---

- Nos permite ajustar una referencia a una variable o método en una sola línea de código.

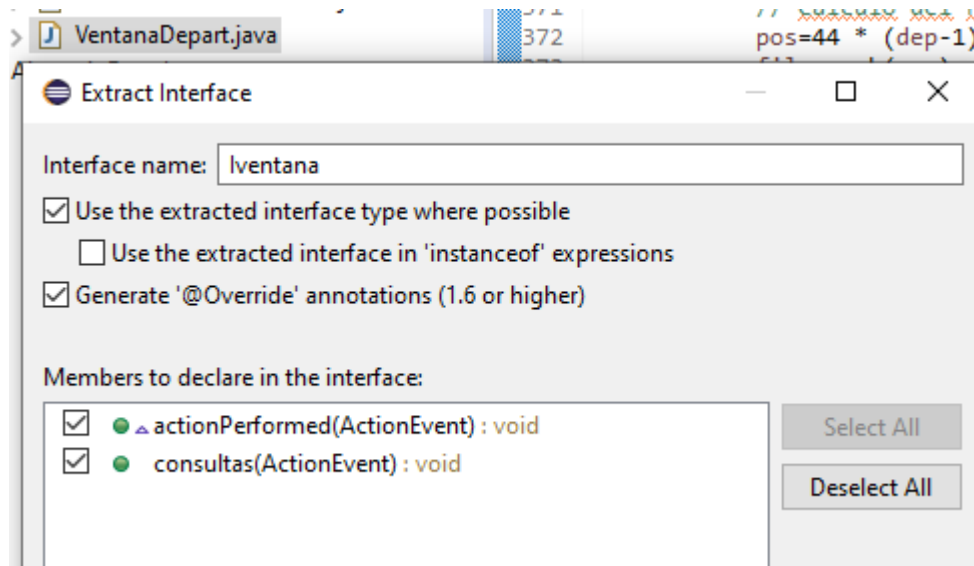
```
public static void main(String[] args) throws FileNotFoundException {  
    File fichero = new File("datos.dat");  
    PrintWriter escritor;  
    escritor = new PrintWriter(fichero);  
    escritor.close();  
}
```



```
public static void main(String[] args) throws FileNotFoundException {  
    PrintWriter escritor = new PrintWriter(new File("datos.dat"));
```

# Extract Interface

- Nos permite seleccionar los métodos de una clase para crear una Interface.
- Una Interface es una plantilla que define los métodos acerca de lo que puede hacer una clase, pero no los desarrolla.



Iventana.java

VentanaDepart.java

```
public interface Iventana {  
    void actionPerformed(ActionEvent e);  
    void consultas(ActionEvent e);  
}
```



# Extract Superclass

---

- Permite crear una superclase (clase padre) con los métodos y atributos que seleccionamos de una clase concreta.
- Se usa cuando la clase con la que trabajamos podría tener cosas en común con otras clases, las cuales serían subclases de la superclase creada.

```
/**
 * Método para ingresar cantidades en la cuenta. Modifica el saldo.
 *
 * @param cantidad          comentario estilo Javadoc
 * @throws Exception
 */
public void ingresar(double cantidad) throws Exception {
    // el parámetro cantidad debe ser positivo comentario de una línea
    if (cantidad < 0)
        throw new Exception("No se puede ingresar una cantidad negativa");
    saldo = saldo + cantidad;
    /* Este método realiza el ingreso de una cantidad
     * de dinero en la cuenta comentario multilínea
     */
}
```

---

# DOCUMENTACIÓN

# Documentación del código fuente

---

- Por una u otra razón, todo programa que tenga éxito será modificado en el futuro.
- Pensando en esta revisión de código es por lo que es importante que el programa se entienda: para poder repararlo y modificarlo.
- ¿Qué hay que documentar?
  - Hay que añadir explicaciones a **todo lo que no es evidente**.
  - No hay que repetir lo que se hace, sino explicar por qué se hace.

# Documentación del código fuente

---

- ¿De qué se encarga una clase? ¿Un paquete?
- ¿Qué hace un método?
- ¿Cuál es el uso esperado de un método?
- ¿Para qué se usa una variable?
- ¿Cuál es el uso esperado de una variable?
- ¿Qué algoritmo estamos usando? ¿De dónde lo hemos sacado?
- ¿Qué limitaciones tiene el algoritmo? ¿... la implementación?
- ¿Qué se debería mejorar... si hubiera tiempo?

# Tipos de comentarios en Java

## Javadoc

- Comienzan con los caracteres `"/**`, se pueden prolongar a lo largo de varias líneas (que probablemente comiencen con el carácter `"*`) y terminan con los caracteres `"*/`.

## Una línea

- Comienzan con los caracteres `"//` y terminan con la línea

## Tipo C

- Comienzan con los caracteres `"/*`, se pueden prolongar a lo largo de varias líneas (que probablemente comiencen con el carácter `"*`) y terminan con los caracteres `"*/`.

```
/**
 * <h1>Encuentra un promedio de tres numeros!</h1>
 * El programa FindAvg implementa una aplicacion que
 * simplemente calcula el promedio de tres enteros y los imprime
 * a la salida en la pantalla.
 *
 * @author Alex Walton
 * @version 1.0
 * @since 2017-15-12
 */
public class FindAvg {

    // Calculo del reg a leer
    pos=44 * (dep-1);

    /*
    public static void main(String args[]) {
        FindAvg obj = new FindAvg();
        int avg = obj.findAvg(10, 20, 30);
        System.out.println("El Promedio de 10, 20 y 30 es: " + avg);
    }
    */
}
```

# Tipos de comentarios en Java

---

Cada tipo de comentario se debe adaptar a un propósito:

## **javadoc**

- Para generar documentación externa.

## **una línea**

- Para documentar código que no necesitamos que aparezca en la documentación externa. Se usará incluso cuando el comentario ocupe varias líneas, cada una comenzará con "//"

## **tipo C**

- Para eliminar código. Ocurre a menudo que código obsoleto no queremos que desaparezca, sino mantenerlo "por si acaso". Para que no se ejecute, se comenta.

# Documentación con JavaDoc

---

## Clases e Interfaces

Deben usarse al menos las etiquetas:

@author

@version

@author	nombre del autor
@version	identificación de la versión y fecha
@see	referencia a otras clases y métodos

```
/**
 * <h2>Clase Empleado, se utiliza para crear y leer empleados de una BD.</h2>
 *
 * Busca información de javadoc en <a href="http://google.com">GOOGLE</a>
 * @see <a href="http://google.com">Google</a>
 * @author Entornos de Desarrollo 2017
 * @version v1.2017
 * @since 30-9-2015
 */
public class Empleado{
```

# Documentación con JavaDoc

---

## Constructores y métodos

Deben usarse las etiquetas

@param : una por argumento de entrada

@return: si el método no es void

@exception ó @throws: una por tipo de Exception que se puede lanzar. Se pueden usar indistintamente.

```
/**
 * Constructor con 3 parámetros
 * Crea objetos empleado, con nombre, apellido y salario.
 * @param nombre Nombre del empleado
 * @param apellido Apellido del empleado
 * @param salario Salario del empleado
 */
public Empleado(String nombre, String apellido, double salario){
    this.nombre = nombre;
}
```

@param	nombre del parámetro	descripción de su significado y uso
@return		descripción de lo que se devuelve
@exception	nombre de la excepción	excepciones que pueden lanzarse
@throws	nombre de la excepción	excepciones que pueden lanzarse



# Javadoc

Documentación generada

All Classes

Empleado

PACKAGE

CLASS

USE

TREE

DEPRECATED

INDEX

HELP

PREV CLASS

NEXT CLASS

FRAMES

NO FRAMES

SUMMARY: NESTED | FIELD | CONSTR | METHOD

DETAIL: FIELD | CONSTR | METHOD

## Class Empleado

java.lang.Object  
Empleado

```
public class Empleado  
extends java.lang.Object
```

**Clase Empleado, se utiliza para crear y leer empleados de una BD.**

Busca información de javadoc en [GOOGLE](#)

Since:  
30-9-2015

Version:  
v1.2017

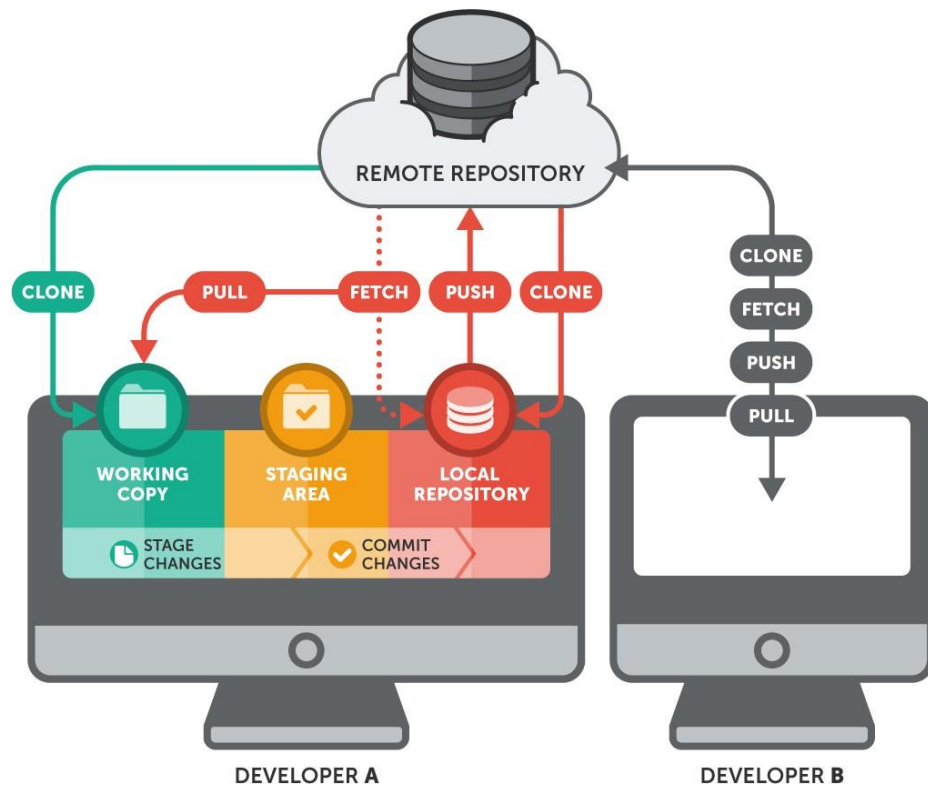
Author:  
Entornos de Desarrollo 2017

See Also:  
[Google](#)

### Field Summary

Fields

Modifier and Type	Field and Description
private java.lang.String	apellido Atributo apellido del empleado



# CONTROL DE VERSIONES

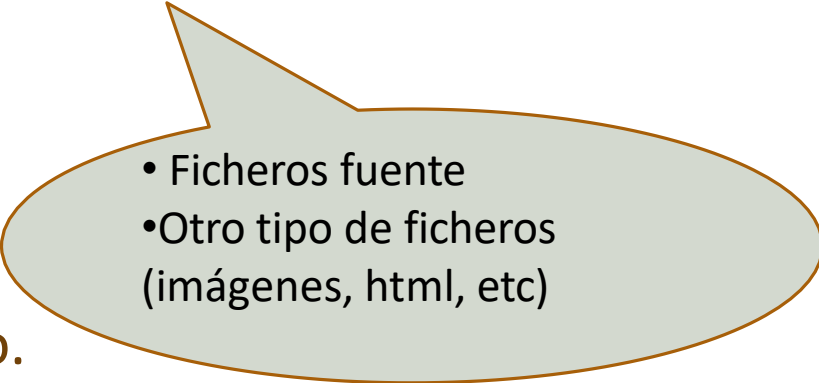
# ¿Qué es un VCS (Sistema de Control de Versiones)?

---

Sistema que registra los cambios realizados en un archivo o conjunto de archivos a lo largo del tiempo.

En cualquier momento se puede :

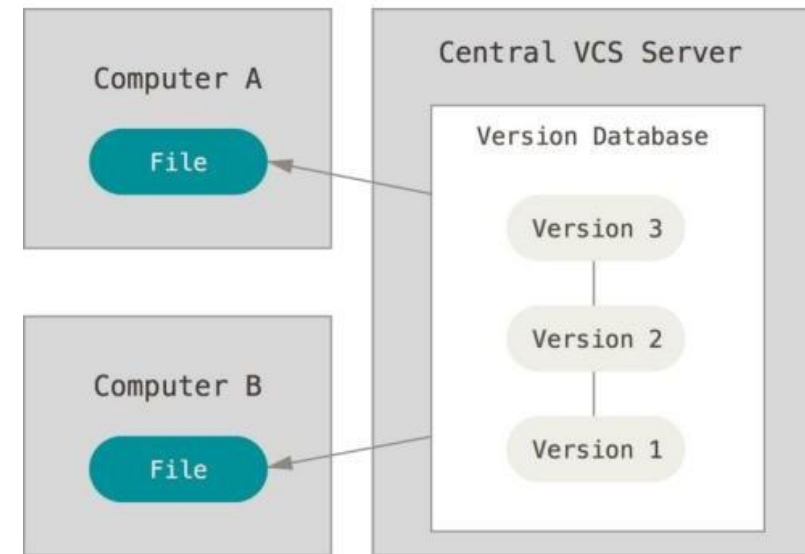
- Volver a una versión anterior.
- Ver los cambios en un determinado archivo.
- Ver quién ha hecho los cambios.

- 
- Ficheros fuente
  - Otro tipo de ficheros (imágenes, html, etc)

# VCS Centralizados (CVCS)

---

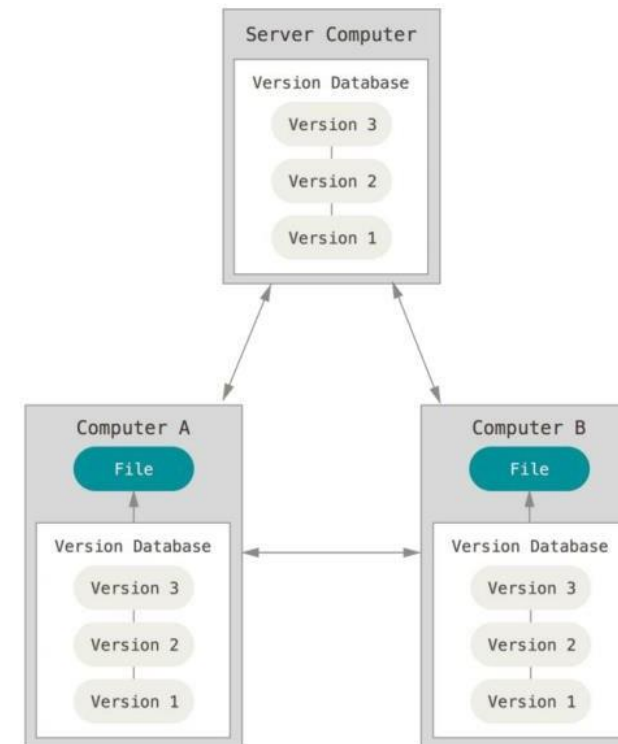
- Un único servidor contiene todos los archivos versionados.
- Varios clientes descargan estos archivos.
- Ejemplos: CVS, Subversion, Perforce



# VCS Distribuidos (DVCS)

---

- Los clientes replican completamente el repositorio.
- Ejemplos: Git, Mercurial, Bazaar, Darcs.

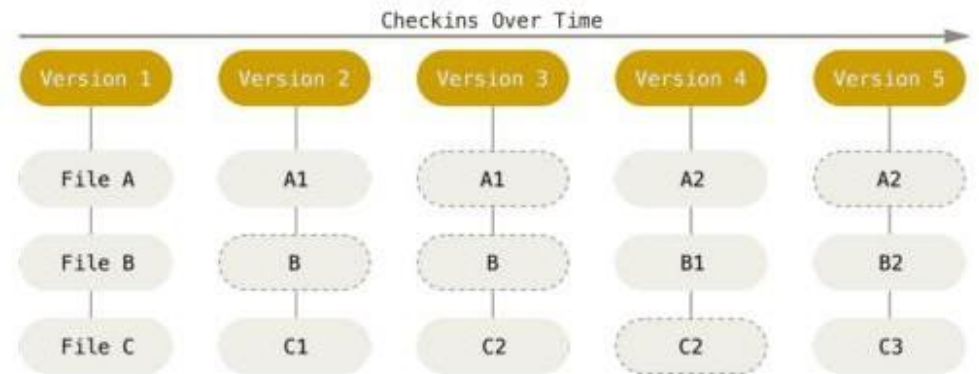


# Fundamentos **git**

---

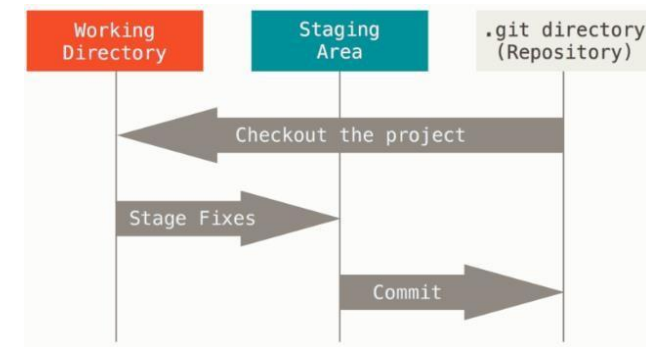
# Copias instantáneas

- Git maneja sus datos como un conjunto de instantáneas.
- Cada vez que se confirma un cambio, toma una foto del aspecto de todos los archivos y guarda una referencia a esa copia instantánea.
- Si los archivos no se han modificado, Git almacena un enlace al archivo anterior idéntico que ya tiene almacenado.



# Secciones de un proyecto Git

- **Directorio de trabajo (WORKING DIRECTORY):** es una copia de una versión del proyecto. Serán los archivos a usar y modificar.
- **Área de preparación (STAGING AREA):** almacena información acerca de lo que va a ir en la próxima confirmación.
- **Repositorio:** (directorio .git) Almacena la base de datos de los objetos del proyecto. Es lo que se copia cuando se clona un repositorio desde otro ordenador.





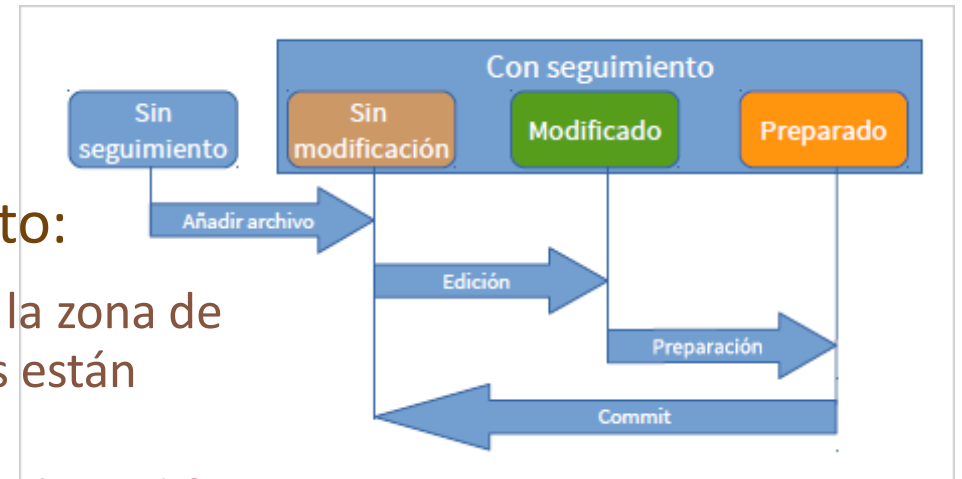
# Estados de un archivo

Cada archivo del directorio de trabajo, puede estar

- sin seguimiento
- bajo seguimiento por parte de Git.

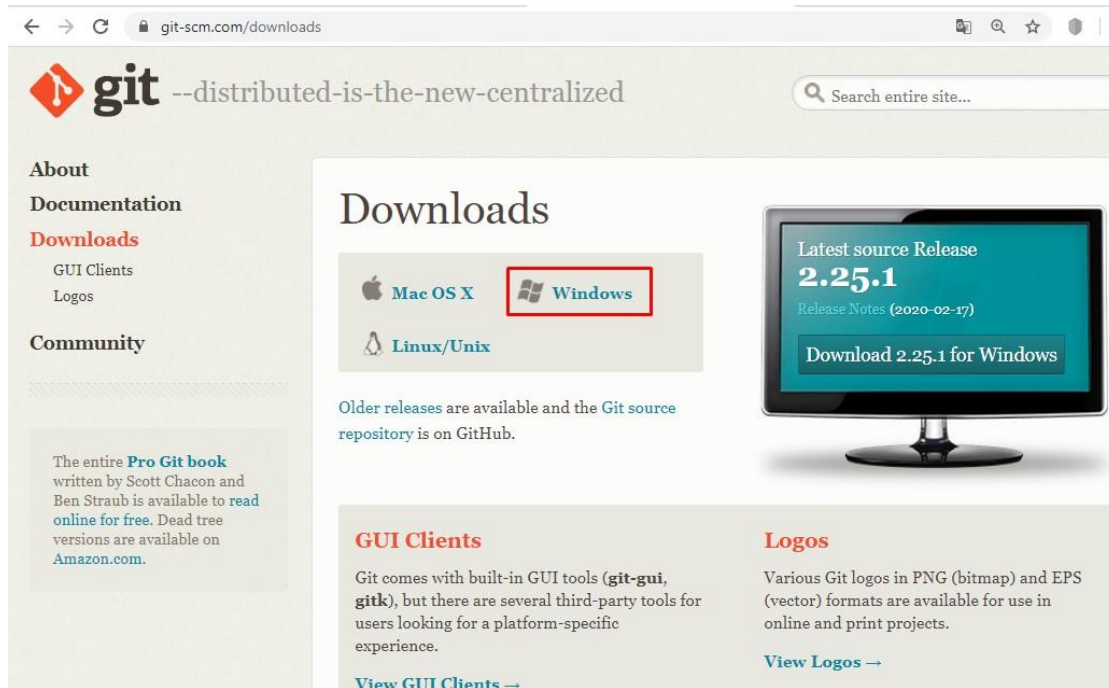
Posibles estados de un archivo bajo seguimiento:

- **Sin modificación** (committed): mismo contenido en la zona de trabajo, zona de preparación y repositorio. Los datos están almacenados de manera segura en el repositorio.
- **Modificado** (modified): el contenido en la zona de trabajo difiere del que está en la zona de preparación y el repositorio.
- **Preparado** (staged): el contenido en la zona de trabajo coincide con el de la zona de preparación pero difiere del repositorio.



# Instalación

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



## Downloading Git



### Your download is starting...

You are downloading the latest (**2.25.1**) 64-bit version of Git for Windows. This is the most recent [maintained build](#). It was released **7 days ago**, on 2020-02-19.

[Click here to download manually](#), if your download hasn't started.

### Other Git for Windows downloads

Git for Windows Setup

**32-bit Git for Windows Setup.**

**64-bit Git for Windows Setup.**

Git for Windows Portable ("thumbdrive edition")

**32-bit Git for Windows Portable.**

**64-bit Git for Windows Portable.**

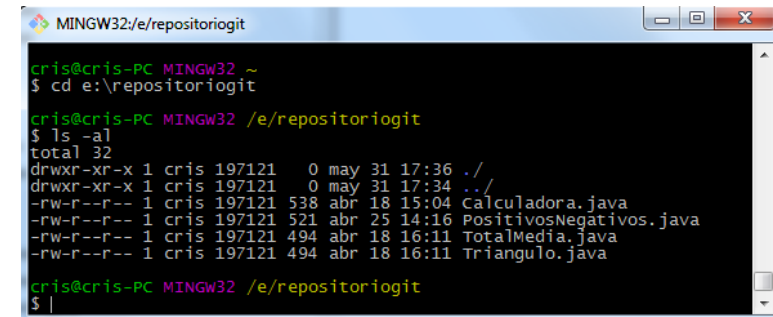
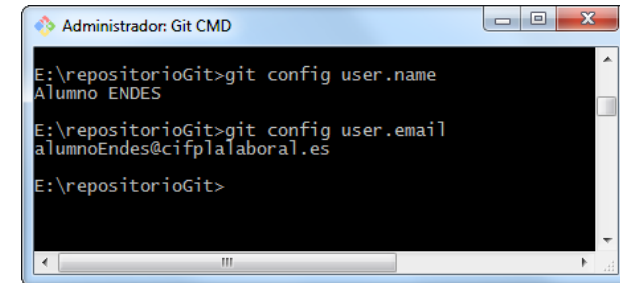
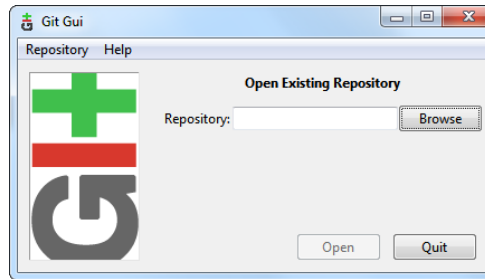
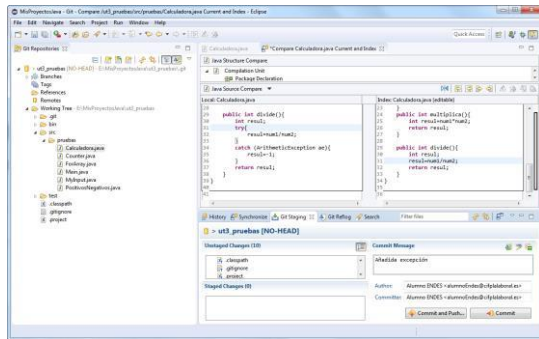
The current source code release is version **2.25.1**. If you want the newer version, you can build it from [the source code](#).

Escogemos el archivo a descargar, dependiendo del sistema operativo utilizado por nuestro ordenador

# Cómo usar Git

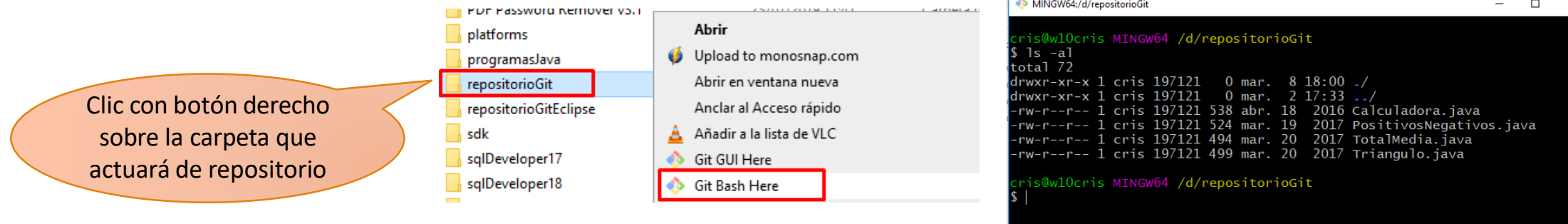
Existen muchas formas de utilizarlo:

- La línea de comandos.(CMD, Bash,..)
- Interfaces gráficas de usuario GUI.
- Un IDE como Eclipse.



# Nosotros utilizaremos Git Bash

Consola de comandos que nos permite trabajar con comandos LINUX.



- git es un comando de UNIX/Linux
- git init      git es un meta-comando, donde el primer parámetro (init) es la operación solicitada
- git --versión

# Configurando Git por primera vez

---

Establecer nombre de usuario y dirección de correo electrónico.

*El desarrollador debe firmar todos los commits que crea en la historia de un proyecto porque git es un software de colaboración. Por ello debe configurar sus credenciales antes de utilizar git.*

- Usuario: `git config --global user.name usuario`

```
cris@w10cris MINGW64 /d/repositorioGit  
$ git config --global user.name "Alumno ENDES"
```

- Email: `git config --global user.email email@dominio.es`

```
cris@w10cris MINGW64 /d/repositorioGit  
$ git config --global user.email "alumnoEndes@cifplalaboral.es"
```

# Configurando Git por primera vez

---

## Comprobar los valores de configuración

- Consultar todas las opciones configuradas: `git config -list`

```
user.name=Alumno ENDES  
user.email=alumnoEndes@cifplalaboral.es  
core.repositoryformatversion=0
```

- Consultar el valor de una opción: `git config user.name`

```
$ git config user.name  
Alumno ENDES
```

- Obtener ayuda sobre un comando

`git help config`

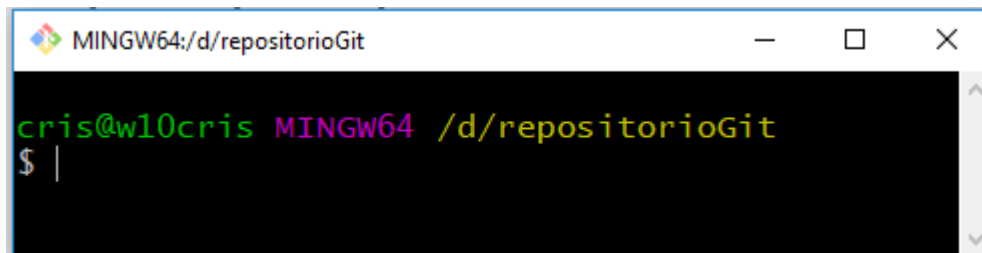
`git config --help`

```
git config [<file-option>] [type] [--show-origin] [-z|--null] name [value [value_regex]]  
git config [<file-option>] [type] --add name value  
git config [<file-option>] [type] --replace-all name value [value_regex]  
git config [<file-option>] [type] [--show-origin] [-z|--null] --get name [value_regex]  
git config [<file-option>] [type] [--show-origin] [-z|--null] --get-all name [value_regex]  
git config [<file-option>] [type] [--show-origin] [-z|--null] [--name-only] --get-regexp name_regex  
[value_regex]  
git config [<file-option>] [type] [-z|--null] --get-urlmatch name URL  
git config [<file-option>] --unset name [value_regex]  
git config [<file-option>] --unset-all name [value_regex]  
git config [<file-option>] --rename-section old_name new_name  
git config [<file-option>] --remove-section name  
git config [<file-option>] [--show-origin] [-z|--null] [--name-only] -l | --list  
git config [<file-option>] --get-color name [default]  
git config [<file-option>] --get-colorbool name [stdout-is-tty]  
git config [<file-option>] -e | --edit
```

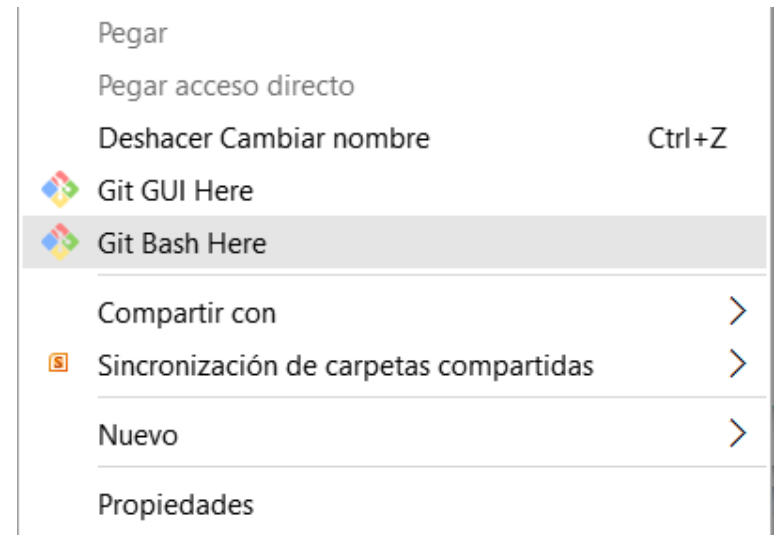
# Trabajo en LOCAL

# Inicializar repositorio en carpeta existente

- Seleccionamos la carpeta que queremos utilizar como repositorio de Git.
- Desde el menú contextual (botón derecho), elegimos Git Bash Here
- Se abre la carpeta del repositorio en la consola.



```
MINGW64:/d/repositorioGit
cris@w10cris MINGW64 /d/repositorioGit
$ |
```





# Inicializar repositorio en carpeta existente



- Crear repositorio: `git init`

Esto crea el subdirectorio `.git`

La rama principal se llama master

```
cris@w10cris MINGW64 /d/repositorioGit
$ git init
Initialized empty Git repository in D:/repositorioGit/.git/
cris@w10cris MINGW64 /d/repositorioGit (master)
```

- Comprobar estado: `git status`

Todavía no hay ningún fichero en el repositorio que esté bajo seguimiento.

Ficheros no rastreados,  
sin seguimiento  
(untracked)

```
cris@w10cris MINGW64 /d/repositorioGit (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    Calculadora.java
    PositivosNegativos.java
    TotalMedia.java
    Triangulo.java

nothing added to commit but untracked files present (use "git add" to track)
cris@w10cris MINGW64 /d/repositorioGit (master)
$
```

# Empezar a rastrear archivos

- Añadir todos los archivos: `git add .`  
O uno a uno `git add Calculadora.java`  
O los de tipo java `git add *.java`
- Confirmar cambios: `git commit -m "mensaje"`

```
cris@w10cris MINGW64 /d/repositorioGit (master)
$ git commit -m "Añadimos ficheros java de prueba"
[master (root-commit) 42fcec2] Añadimos ficheros java de prueba
4 files changed, 89 insertions(+)
 create mode 100644 Calculadora.java
 create mode 100644 PositivosNegativos.java
 create mode 100644 TotalMedia.java
 create mode 100644 Triangulo.java

cris@w10cris MINGW64 /d/repositorioGit (master)
```

```
cris@w10cris MINGW64 /d/repositorioGit (master)
$ git add .

cris@w10cris MINGW64 /d/repositorioGit (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   Calculadora.java
        new file:   PositivosNegativos.java
        new file:   TotalMedia.java
        new file:   Triangulo.java

cris@w10cris MINGW64 /d/repositorioGit (master)
$
```

# Información sobre los cambios confirmados

Información sobre los commits realizados (cambios confirmados): `git log`

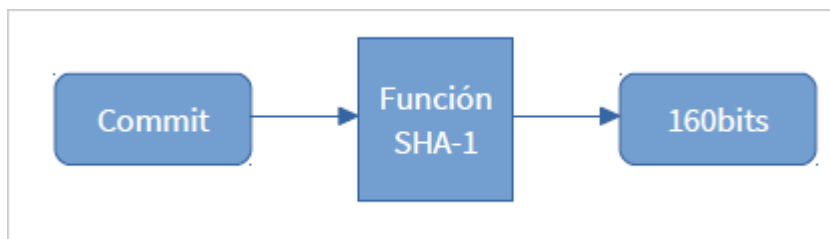
Quién y cuándo se hizo el commit

```
cris@w10cris MINGW64 /d/repositorioGit (master)
$ git log
commit 42fcec27ef87e94f1e3d1eabbce677b9b254bbab (HEAD -> master)
Author: Alumno ENDES <alumnoEndes@cifplalaboral.es>
Date: Thu Mar 8 18:30:03 2018 +0100

    Añadimos ficheros java de prueba

cris@w10cris MINGW64 /d/repositorioGit (master)
```

Comentario



Función criptográfica (**Secure Hash Algorithm**), que genera una cadena de 40 caracteres hexadecimales (20bytes) que identifica cualquier cambio en un fichero.

`42fcec27ef87e94f1e3d1eabbce677b9b254bbab`

Versión corta (7 primeros caracteres)

`42fcec2`

# Realizar cambios

- Hacemos algunas modificaciones en uno de los ficheros, en el directorio de trabajo.

Por ejemplo, modificamos el fichero PositivosNegativos.java

- Comprobamos estado: `git status`
- Mostramos los cambios realizados, pero aún no preparados: `git diff`
  - Líneas Añadidas: **en verde** y comienzan por +
  - Líneas eliminadas: **en rojo** y comienzan por -

```
cris@w10cris MINGW64 /d/repositorioGit (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will
   (use "git checkout -- <file>..." to discard ch

        modified:   PositivosNegativos.java

no changes added to commit (use "git add" and/or "git commit -a")

cris@w10cris MINGW64 /d/repositorioGit (master)
$ git diff
diff --git a/PositivosNegativos.java b/PositivosNegativos.java
index 39c712c..a25d3ff 100644
--- a/PositivosNegativos.java
+++ b/PositivosNegativos.java
@@ -1,6 +1,7 @@
 package laboral.entornos.pruebas;

 public class PositivosNegativos {
+    //m<39>todo ejecutable
     public static void main(String[] args) {
         int num, positivos = 0, negativos = 0;
         do {
```

Modificado respecto  
al commit anterior

# Registrar cambios

- Preparamos archivo modificado: `git add PositivosNegativos.java`
- Confirmamos cambios: `git commit -m "Añadimos comentarios"`

```
cris@w10cris MINGW64 /d/repositorioGit (master)
$ git add PositivosNegativos.java

cris@w10cris MINGW64 /d/repositorioGit (master)
```

```
cris@w10cris MINGW64 /d/repositorioGit (master)
$ git commit -m "Reallizamos segundo commit"
[master 346f536] Reallizamos segundo commit
1 file changed, 1 insertion(+)

cris@w10cris MINGW64 /d/repositorioGit (master)
```

- Mostramos histórico (commit): `git log`  
`git log --oneline`

```
cris@w10cris MINGW64 /d/repositorioGit (master)
$ git log --oneline
346f536 (HEAD -> master) Reallizamos segundo commit
42fcec2 Añadimos ficheros java de prueba
```

```
cris@w10cris MINGW64 /d/repositorioGit (master)
$ git log
commit 346f536b05706fe9c97f8926a922d39f3dbdfd7e (HEAD -> master)
Author: Alumno ENDES <alumnoEndes@cifplalaboral.es>
Date: Thu Mar 8 18:55:21 2018 +0100

    Reallizamos segundo commit

commit 42fcec27ef87e94f1e3d1eabbce677b9b254bbab
Author: Alumno ENDES <alumnoEndes@cifplalaboral.es>
Date: Thu Mar 8 18:30:03 2018 +0100

    Añadimos ficheros java de prueba

cris@w10cris MINGW64 /d/repositorioGit (master)
```

# Deshacer cambios

- Deshacer archivo preparado:  
`git reset HEAD Triangulo.java`

Por ejemplo cuando tras modificar dos archivos quiero confirmarlos como cambios separados, pero ejecuto `git add .` y preparo ambos por error.

Extraigo fichero de la zona de preparación

```
cris@w10cris MINGW64 /d/repositorioGit (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   TotalMedia.java
        modified:   Triangulo.java

no changes added to commit (use "git add" and/or "git commit -a")

cris@w10cris MINGW64 /d/repositorioGit (master)
$ git add .

cris@w10cris MINGW64 /d/repositorioGit (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   TotalMedia.java
        modified:   Triangulo.java

cris@w10cris MINGW64 /d/repositorioGit (master)
$ git reset HEAD Triangulo.java
Unstaged changes after reset:
M       Triangulo.java

cris@w10cris MINGW64 /d/repositorioGit (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   TotalMedia.java

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   Triangulo.java
```

Ambos ficheros están preparados

# Deshacer cambios en el directorio de trabajo

- Deshacer cambios:

`git checkout --Triangulo.java`

Esto devuelve al fichero al estado en que estaba en la última confirmación, deshaciendo todos los cambios realizados a partir de la misma.

**Peligro!** Los cambios se pierden.

```
cris@w10cris MINGW64 /d/repositorioGit (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    modified:   TotalMedia.java

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   Triangulo.java

cris@w10cris MINGW64 /d/repositorioGit (master)
$ git checkout -- Triangulo.java

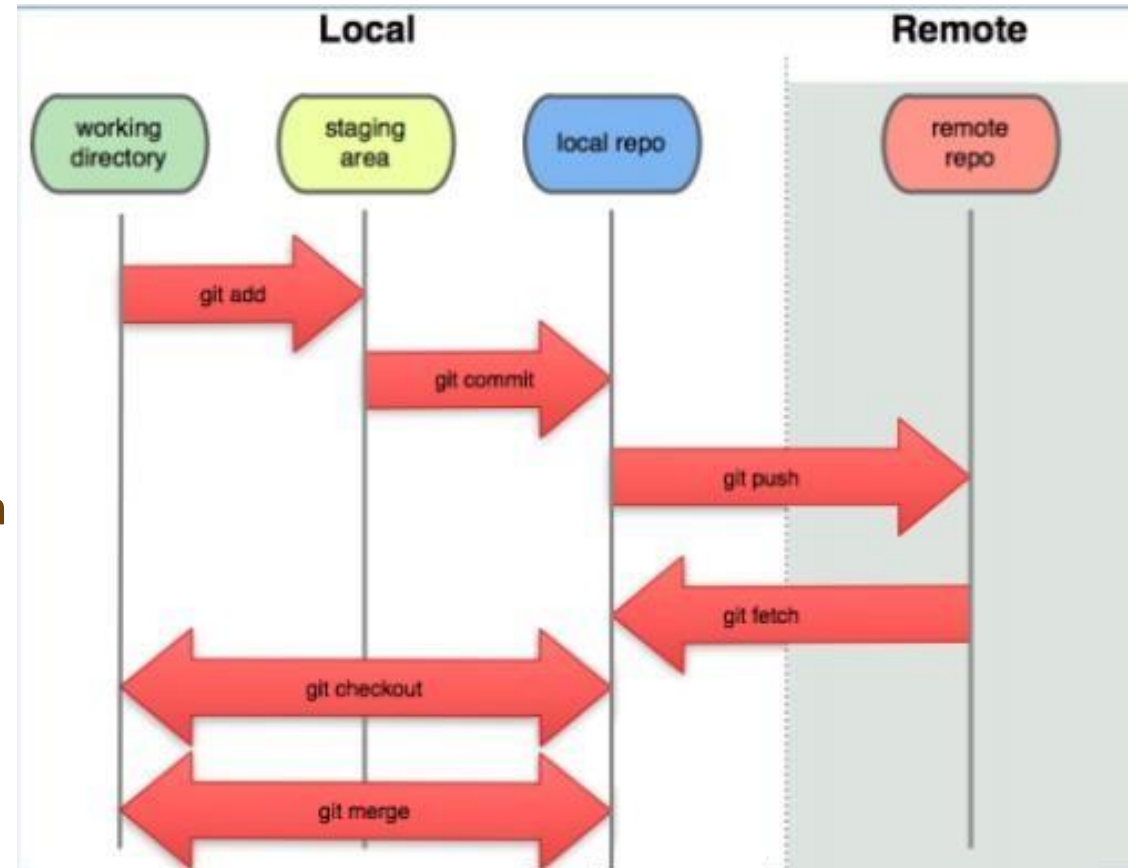
cris@w10cris MINGW64 /d/repositorioGit (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    modified:   TotalMedia.java
```

# Flujo de trabajo básico

- Modificar una serie de archivos en el directorio de trabajo (working directory).
- Preparar los archivos, añadiéndolos al área de preparación (Staging area):  
`git add`
- Confirmar los cambios: tomar los archivos tal y como están en el área de preparación y almacenar esa instantánea de forma permanente en el directorio Git (Local repo):

`git commit -m "mensaje"`





# Trabajo con REMOTOS

---

VERSIONES DE PROYECTOS HOSPEDADAS EN INTERNET.

PUEDES TENER PERMISOS DE SÓLO LECTURA O DE LECTURA/ESCRITURA.

# Clonar un repositorio existente

---

- Obtener una copia de un repositorio Git existente: `git clone [url]`

Crea un directorio con el nombre del directorio del proyecto, descarga toda la información del proyecto y saca una copia de trabajo de la última versión.

<https://github.com/libgit2/libgit2>

```
MINGW64: d/repositorioGit/libgit2
cris@w10cris MINGW64 /d/repositorioGit (master)
$ git clone https://github.com/libgit2/libgit2
Cloning into 'libgit2'...
remote: Counting objects: 79354, done.
remote: Compressing objects: 100% (22600/22600), done.
remote: Total 79354 (delta 55336), reused 79335 (delta 55317), pack-reused 0
Receiving objects: 100% (79354/79354), 36.70 MiB | 21.65 MiB/s, done.
Resolving deltas: 100% (55336/55336), done.
Checking out files: 100% (5564/5564), done.

cris@w10cris MINGW64 /d/repositorioGit (master)
$ cd libgit2

cris@w10cris MINGW64 /d/repositorioGit/libgit2 (master)
$ ls -la
total 299
drwxr-xr-x 1 cris 197121  0 abr.  8 21:07 ./
drwxr-xr-x 1 cris 197121  0 abr.  8 21:07 ../
```

# Trabajo con remotos

---

- Ver los remotos configurados: `git remote -v`

```
cris@w10cris MINGW64 /d/repositorioGit/libgit2 (master)
$ git remote -v
origin https://github.com/libgit2/libgit2 (fetch)
origin https://github.com/libgit2/libgit2 (push)
```

Nombre del repositorio  
(por defecto al clonar)

- Obtener datos de proyectos remotos: `git fetch origin`

Traerá todos los datos que aún no tengo en mi repositorio remoto.

la rama por defecto

- Enviar datos a un proyecto remoto: `git push origin master`

Tengo que tener permisos de escritura... Enviaría todos los commits realizados al servidor.