



DESPLIEGUE DE APLICACIONES WEB

Unidad 06

**Documentación,
sistemas de control de
versiones y de
integración continua**



Los documentos, elementos gráficos, vídeos, transparencias y otros recursos didácticos incluidos en este contenido pueden contener imprecisiones técnicas o errores tipográficos. Periódicamente se realizan cambios en el contenido. Fomento Ocupacional FOC SL puede realizar en cualquier momento, sin previo aviso, mejoras y/o cambios en el contenido.

Es responsabilidad del usuario el cumplimiento de todas las leyes de derechos de autor aplicables. Ningún elemento de este contenido (documentos, elementos gráficos, vídeos, transparencias y otros recursos didácticos asociados), ni parte de este contenido puede ser reproducida, almacenada o introducida en un sistema de recuperación, ni transmitida de ninguna forma ni por ningún medio (ya sea electrónico, mecánico, por fotocopia, grabación o de otra manera), ni con ningún propósito, sin la previa autorización por escrito de Fomento Ocupacional FOC SL.

Este contenido está protegido por la ley de propiedad intelectual e industrial. Pertenecen a Fomento Ocupacional FOC SL los derechos de autor y los demás derechos de propiedad intelectual e industrial sobre este contenido.

Sin perjuicio de los casos en que la ley aplicable prohíbe la exclusión de la responsabilidad por daños, Fomento Ocupacional FOC SL no se responsabiliza en ningún caso de daños indirectos, sean cuales fueren su naturaleza u origen, que se deriven o de otro modo estén relacionados con el uso de este contenido.

Índice

1. Objetivos	4
2. Introducción	5
3. Herramientas colaborativas para la elaboración y mantenimiento de la documentación	7
4. Instalación, configuración y uso	8
4.1. Javadoc	8
4.1.1. Instalación de Javadoc	9
4.1.2. Documentando con Javadoc.....	11
4.2. PhpDocumentor.....	12
4.2.1. Instalación de phpDocumentor	13
4.2.2. Configuración de phpDocumentor	14
5. Creación y utilización de plantillas	15
6. Instalación, configuración y uso de sistemas de control de versiones	17
6.1. Instalación de git.....	18
6.2. Configuración de git.....	19
7. Operaciones avanzadas	21
7.1. Ayuda de git.....	21
7.2. Instalación de Gitweb	21
7.3. Configurar proyectos sobre git	22
7.4. Gestionar versiones sobre git	23
8. Seguridad de los sistemas de control de versiones	25
9. Instalación, configuración y uso de sistemas de integración continua del código. Monitorización continua de las métricas de calidad de la aplicación	27
9.1. Instalación de sistemas de integración continua del código	27
9.2. Configuración y uso de sistemas de integración continua del código.....	32
9.3. Monitorización continua de las métricas de calidad de la aplicación	36
10. Resumen.....	38
11. Mapa Conceptual.....	39
12. Bibliografía	40

Unidad 06

Documentación, sistemas de control de versiones y de integración continua

1. Objetivos

Después de completar esta unidad, será capaz de:

- Identificar diferentes herramientas de generación de documentación.
- Documentar los componentes software utilizando los generadores específicos de las plataformas.
- Utilizar diferentes formatos para la documentación.
- Utilizar herramientas colaborativas para la elaboración y mantenimiento de la documentación.
- Instalar, configurar y utilizar un sistema de control de versiones.
- Garantizar la accesibilidad y seguridad de la documentación almacenada por el sistema de control de versiones.
- Documentar la instalación, configuración y uso del sistema de control de versiones utilizado.
- Utilizar herramientas para la integración continua del código.

2. Introducción

Durante el desarrollo de una aplicación informática, los desarrolladores o programadores, que la están implementando, generan una documentación del código que se crea, con el objeto de facilitar el mantenimiento de la aplicación y corrección de posibles errores que se produzcan durante el desarrollo y uso del sistema desarrollado. Sin embargo, en la mayoría de los casos, los usuarios o usuarias finales de la aplicación, van a ser distintos a los desarrolladores y desarrolladoras de la aplicación, y posiblemente desconozcan por completo el código de la aplicación que utilizan.

Para facilitar el uso de la aplicación que se crea, es necesario que como parte del proyecto se generen documentos, tanto en formato electrónico como impresos en papel, que permitan a los usuarios y usuarias finales de la aplicación su instalación, configuración, uso, mantenimiento, actualización y eliminación en su caso. Estos documentos que se incorporan a las aplicaciones son los manuales. En la documentación de software, se deben considerar los siguientes estándares:

- La **Norma ISO/IEC 26514:2008** pretende cubrir las necesidades de cualquier persona que utiliza aplicaciones web sobre la forma en que el software puede ayudarle a realizar una tarea. Pretende ayudar a los diseñadores y desarrolladores, ya que define el proceso de catalogación de la documentación del desarrollador. El informe abarca las etapas implicadas en el diseño, especificando, y la producción de documentación para el usuario. Se aplica tanto a la documentación impresa como en pantalla. Recomienda que el desarrollo de la documentación del usuario debe ser parte del desarrollo del producto de software y sigue los mismos procesos como el ciclo de vida del producto.
- El estándar **IEEE 830** tiene como finalidad la integración de los requerimientos del sistema desde la perspectiva del usuario, cliente y desarrollador. Para ello, se encarga de establecer las pautas para identificar y esquematizar los requerimientos de software.
- **PMBOK** es un estándar en la gestión de proyectos desarrollado por el Project Management Institute (PMI). La misma comprende dos grandes secciones, la primera sobre los procesos y contextos de un proyecto, la segunda sobre las áreas de conocimiento específico para la gestión de un proyecto. El estándar **PMBOK** es reconocido internacionalmente (**IEEE Std 1490-2003**) que provee los fundamentos de la gestión de proyectos.
- El estándar **ITIL** (del inglés Information Technology Infrastructure Library), es un marco de trabajo de las buenas prácticas destinadas a facilitar la entrega de servicios de tecnologías de la información. **ITIL** resume un extenso conjunto de procedimientos de gestión ideados para ayudar a las organizaciones a lograr calidad y eficiencia en las operaciones de tecnologías de la información.

Cuando desarrollamos software, normalmente la información sobre la implementación no necesita salir del código, pero, por el contrario, la información de la interfaz conviene pasarla a un documento independiente del código fuente (**manual de uso**). De esta forma, la persona que necesite utilizar una determinada librería de clases o funciones tendrá toda la información necesaria.

El problema con este tipo de documentación es que cada vez que se modifica algo en el código (actualizaciones, corrección de errores, etc...) hay que reflejarlo también en el manual de uso, lo que implica doble trabajo. Lo ideal, por tanto, sería poder **automatizar** de alguna forma este proceso.



Imagen: Documentación

Por otro lado, cuando realizamos un proyecto software, es bastante habitual que vayamos haciendo pruebas, modificando nuestro código fuentes continuamente, añadiendo funcionalidades, etc. Muchas veces, antes de abordar un cambio importante que requiera tocar mucho código, nos puede interesar guardarnos una **versión** de las fuentes que tenemos en ese momento, de forma que guardamos una versión que sabemos que funciona y abordamos, por separado, los cambios.

Si no usamos ningún tipo de herramienta que nos ayude a hacer esto, lo más utilizado es directamente hacer una copia del código fuente en un directorio separado. Pero esta no es la mejor forma. Hay herramientas, los **sistemas de control de versiones**, que nos ayudan a guardar las distintas versiones de las fuentes.

3. Herramientas colaborativas para la elaboración y mantenimiento de la documentación

El **Software colaborativo** o **groupware** es un conjunto de programas informáticos que integran el trabajo en un sólo proyecto con muchos usuarios concurrentes, que se encuentran en diversas estaciones de trabajo, mejorando su rendimiento y contribuyendo a que personas que están localizadas en puntos geográficos diferentes puedan trabajar a la vez, ya sea directamente o de forma anónima, a través de las redes.

Entre las herramientas colaborativas para la gestión y documentación de aplicaciones web, podemos destacar las siguientes.

- **Coefficient**. Es una plataforma de colaboración escalable que se puede ejecutar en contenedores J2EE y aplicaciones web. En la actualidad ofrece herramientas básicas de colaboración como módulos desplegables. También proporciona un motor de flujo de trabajo para ayudar a proyectos de guía a través de su ciclo de desarrollo.
- **DataShare** es la aplicación de red para los clientes que desean compartir datos utilizando un servidor Central. DataShare está escrito en Java, y permite a los clientes compartir datos (enviar y recibir) sin tener que saber nada acerca de los otros clientes.
- **LibreSource** es una plataforma de colaboración versátil. Open Source, modular y altamente personalizable, LibreSource está adaptada para el desarrollo de software colaborativo, trabajo en grupo y publicación en la Web. En un único servidor, LibreSource puede alojar varios proyectos, varios grupos de usuarios, y permitir el acceso a los recursos. Está basado en la tecnología Java / J2EE.
- **MediaWiki** es un proyecto de software de fundación Wikimedia desarrollado en PHP muy popular y extendido por Internet
- **Teambox** es un proyecto de software colaborativo online, que combina las mejores prácticas en productividad con herramientas propias de redes sociales. Este proyecto trata de cubrir la necesidad de una herramienta de administración de proyectos con capacidades de colaboración y fácil de usar. Posee integración con Google Docs, Dropbox e integra fácilmente archivos, calendarios y email.

En los entornos de programación, cada vez es más común emplear generadores de automáticos de documentación a partir del código fuente. Por ejemplo, **Javadoc** es la herramienta estándar en Java mientras **phpDocumentor** es herramienta más utilizadas para PHP. De esta forma, es más fácil realizar el mantenimiento de la documentación.

4. Instalación, configuración y uso

Los entornos de programación modernos, como **NetBeans** o **Eclipse**, aprovechan los comentarios de nuestro código fuente para mostrar información muy útil, sobre todo para terceras personas.

Javadoc es la herramienta estándar para Java, mientras para PHP una de las herramientas más utilizadas es **phpDocumentor**. O bien podemos emplear una alternativa a estas herramientas como **Doxxygen**, que puede documentar código en varios lenguajes de programación.

La principal diferencia entre **Doxxygen** y **Javadoc** o **phpDocumentor** está en que el primero es un programa, mientras las otras herramientas indicadas son una colección de código, es decir, generan la documentación con el mismo lenguaje de programación.



Imagen: Logo de Doxygen

4.1.

Javadoc

Javadoc es una utilidad de Sun Microsystems empleado para generar **API (Application Programming Interface)** en formato **HTML** de un archivo de código fuente **Java**. Es el estándar de la industria para documentar clases de **Java** y la mayoría de los IDE lo emplean para generar automáticamente la documentación.

La finalidad de **Javadoc** es evitar que la documentación de una aplicación quede obsoleta durante su desarrollo, pero no se dispone del tiempo suficiente para mantener la documentación al día.

Sólo es necesario seguir una serie de reglas a la hora de generar los comentarios en el código y podrán obtener una buena documentación simplemente usando esta herramienta si se mantienen actualizados los comentarios cuando se cambie el código.

```

    /* Método para ingresar cantidades en la cuenta. Modifica el saldo.
     * Este método va a ser probado con Junit
     */
    /**
     * Comentarios estilo Javadoc
     *
     * @param cantidad
     * @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
         * la cuenta. Comentario multilínea
        */
    }

```

Imagen: Comentarios al código

La documentación a ser utilizada por **Javadoc** se escribe en comentarios que comienzan con **/**** y que terminan con ***/**, comenzando cada línea del comentario por ***** a su vez, dentro de estos comentarios se puede escribir código **HTML** y operadores generalmente precedidos por **@** y sensibles a mayúsculas y minúsculas interpretados por **Javadoc**. Existen dos tipos de etiquetas:

- **Etiquetas de bloque**: sólo se pueden utilizar en la sección de etiquetas que sigue a la descripción principal. Son de la forma: **@etiqueta**
- **Etiquetas inline**: se pueden utilizar tanto en la descripción principal como en la sección de etiquetas. Son de la forma: **{@tag}**, es decir, se escriben entre los símbolos de llaves.

Básicamente **Javadoc** es un programa, que recoge los comentarios que se colocan en el código con marcas especiales y construye un archivo **HTML** con clases, métodos y la documentación que corresponde.

4.1.1. Instalación de Javadoc

Previamente a la instalación de Javadoc, tendremos en cuenta que estamos realizando la programación Java desde una herramienta IDE como puede ser **Eclipse** o **NetBeans**; sin duda son los dos entornos de desarrollo integrados que más han crecido en los últimos tiempos. Su comunidad de desarrolladores sirve como pilar para su crecimiento y evolución constante.

El avance de las nuevas tecnologías, nuevos lenguajes y metodologías en el desarrollo del software hacen que lo nuevo quede viejo en poco tiempo. Esto presiona a los programadores a trabajar de manera más intensa agregando nuevas funcionalidades y perfeccionando sus productos en una competencia por ser el mejor IDE.

En el caso de querer realizar la instalación de **NetBeans**, accedemos a la página de **NetBeans** para realizar la descarga.

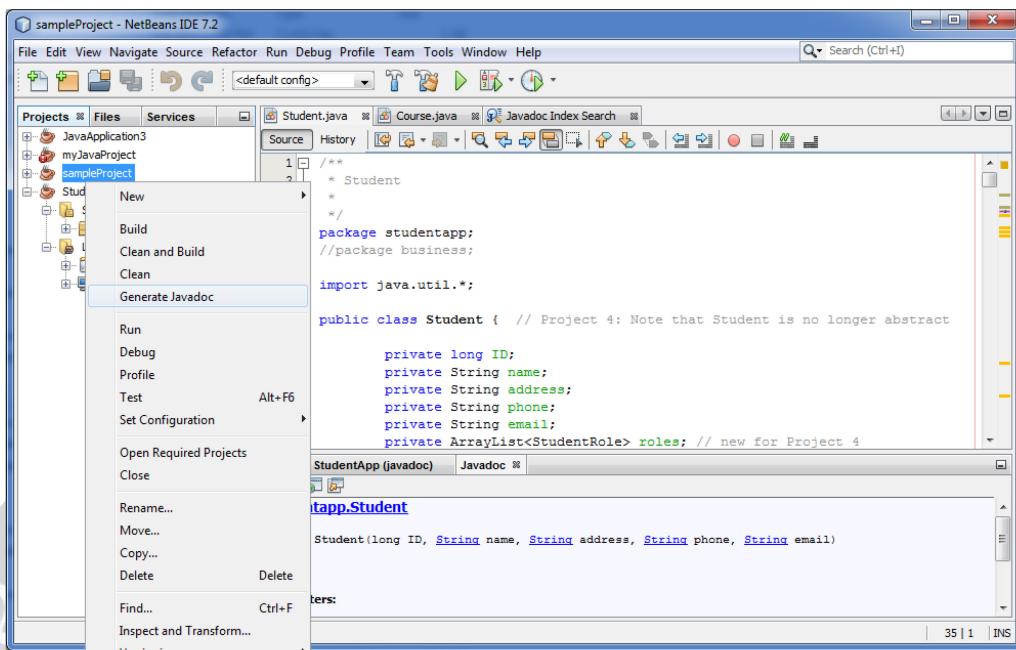


Imagen: Uso de Javadoc en NetBeans

Tanto **Eclipse** como **NetBeans** disponen entre sus opciones la de generar **javadoc** y mediante diversas ventanas que ofrecen se pueden seleccionar las opciones para **javadoc**.

Ahora si accedemos a la opción de menú "**Project**" observamos una opción en la que podemos seleccionar "**Generate Javadoc...**" donde nos permite seleccionar el proyecto del que generar la documentación y también la ruta de la carpeta donde generarla; una vez establecidos dichos parámetros se genera la documentación, que podremos consultar accediendo desde un navegador a los documentos html generados; existe un "index.html" desde el que podremos iniciar la navegación e ir accediendo a la documentación generada.

La mayor parte de los entornos de desarrollo incluyen un botón para llamar a **javadoc** así como opciones de configuración; no obstante, siempre se puede ir al directorio donde se instaló el JDK y ejecutar **javadoc** directamente sobre el código fuente Java con el siguiente comando **sudo javadoc ejemplo.java**

4.1.2. Documentando con Javadoc

Los comentarios **Javadoc** están destinados a describir clases y métodos empleando un formato común, que permite a los comentarios escritos por un programador ser legibles por otro. Para ello los comentarios **Javadoc** deben incluir unos indicadores especiales, que comienzan siempre por '@' y se suelen colocar al comienzo de línea salvo la descripción de la clase o del método que no va precedida de ningún indicador.

```
/**
 * Una clase para empezar a programar en Java
 * el típico ejemplo de HolaMundo
 * @version 1.2.0, 24/07/11
 * @author Programador Java
 */
public class holamundo {
 /**
 * Muestra el mensaje de Hola Mundo
 */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        System.out.println("Hola mundo!");
    }
}
```

Imagen: Ejemplo de documentación

Es importante observar que los indicadores no son obligatorios, es decir, en un método sin parámetros no se incluye obviamente el indicador **@param**. También puede darse que un comentario incluya un indicador más de una vez, es decir, varios indicadores **@param** porque el método tiene varios parámetros.

Tabla: Etiquetas más comunes en Javadoc

Campo	Descripción
@author nombreDelAutor	Indica quién escribió el código al que se refiere el comentario. Si son varias personas se escriben los nombres separados por comas o se repite el indicador, según se prefiera. Es normal incluir este indicador en el comentario de la clase y no repetirlo para cada método, a no ser que algún método haya sido escrito por otra persona.
@version númeroVersión	Si se quiere indicar la versión. Normalmente se usa para clases, pero en ocasiones también para métodos.
@param nombreParámetro	Para describir un parámetro de un método.
@return descripción	Describe el valor de salida de un método.
@see nombre	Cuando el trozo de código comentado se encuentra relacionada con otra clase o método, cuyo nombre se indica en nombre.
@deprecated descripción	Indica que el método (es más raro encontrarlos para una clase) ya no se usa y se ha sustituido.

4.2. PhpDocumentor

phpDocumentor es una herramienta de documentación basada en el estándar establecido por **JavaDoc** que permiten generar documentación de forma automática a partir del código fuente de varias formas:

- **Desde línea de comandos** (php CLI - Command Line Interpreter).
- **Desde interfaz web (incluida en la distribución)**.
- **Desde código.** Como **phpDocumentor** está desarrollado en PHP, podemos incluir su funcionalidad dentro de scripts propios.

En todo caso, es necesario especificar los siguientes parámetros:

1. El **directorio** en el que se encuentra **nuestro código**. PhpDocumentor se encargará luego de recorrer los subdirectorios de forma automática.
2. Opcionalmente los **paquetes (@package)** que deseamos documentar, lista de ficheros incluidos y/o excluidos y otras opciones interesantes para personalizar la documentación.
3. El directorio en el que se generará la **documentación**.
4. Si la documentación va a ser pública (sólo interfaz) o interna (en este caso aparecerán los bloques **private** y los comentarios **@internal**).
5. El formato de salida de la documentación.

Finalmente tenemos que podemos generar varios formatos de salida: **HTML**, **PDF** y **XML (DocBook)**.

phpDocumentor distribuye la documentación en bloques "DocBlock". Estos bloques siempre se colocan justo antes del elemento al que documentan y su formato es:

```
/***
 * Descripción breve (una línea)
 *
 * Descripción extensa. Todas las líneas que
 * sean necesarias
 * Todas las líneas comienzan con *
<- Esta línea es ignorada
*
* Este DocBlock documenta la función suma()
*/
function suma()
{
...
}
```

Imagen: Ejemplo de cabecera phpDocumentor

Los elementos que pueden ser documentados son define, function, class, class vars, include/require/include_once/require_once y global variables. También se puede incluir documentación global a nivel de fichero y clase mediante la marca **@package**.

Dentro de cada **DockBlock** se pueden incluir marcas que serán interpretadas por **phpDocumentor** con un significado especial, dichas marcas pueden ser las siguientes:

- **@access**: Si @access es '**private**' no se genera documentación para el elemento (a menos que se indique explícitamente). Muy interesante si sólo se desea generar documentación sobre la interfaz (métodos públicos) pero no sobre la implementación (métodos privados).
- **@author**: Autor del código.
- **@copyright**: Información sobre derechos.
- **@deprecated**: Para indicar que el elemento no debería utilizarse, ya que en futuras versiones podría no estar disponible.
- **@example**: Permite especificar la ruta hasta un fichero con código PHP. phpDocumentor se encarga de mostrar el código resaltado (syntax-highlighted).
- **@ignore**: Evita que phpDocumentor documente un determinado elemento.
- **@internal**: Para incluir información que no debería aparecer en la documentación pública, pero sí puede estar disponible como documentación interna para desarrolladores.
- **@link**: Para incluir un enlace (<http://...>) a un determinado recurso.
- **@see**: Se utiliza para crear enlaces internos (enlaces a la documentación de un elemento).
- **@since**: Permite indicar que el elemento está disponible desde una determinada versión del paquete o distribución.
- **@version**: Versión actual del elemento.

Existen otras marcas que solamente se pueden utilizar en bloques de determinados elementos:

- **@global**: Para especificar el uso de variables globales dentro de una función.
- **@param**: Para documentar parámetros que recibe una función.
- **@return**: Valor devuelto por una función.



Imagen: Interacción de phpDocumentor

4.2.1. Instalación de **phpDocumentor**

El método recomendado actualmente requiere tener Docker instalado e instalar la imagen creada para **phpDocumentor** de Docker con todas las dependencias preinstaladas con el comando `docker run --rm -v "$(pwd)":/data" "phpdoc/phpdoc:3"`

También se recomienda crear un alias para este comando introduciendo el comando `alias phpdoc="docker run --rm -v $(pwd):/data phpdoc/phpdoc:3"`

Después de hacer eso, simplemente puede invocar el comando **phpdoc** desde cualquier ubicación.

4.2.2. Configuración de **phpDocumentor**

Una vez hemos instalado **phpDocumentor** se puede trabajar con él de dos modos para generar automáticamente la documentación de nuestros proyectos PHP; se puede trabajar desde línea de comandos, empleando la siguiente sintaxis:

```
# phpdoc -o [formato_de_la_documentacion_generada] -d  
[carpeta_donde_estan_los_proyectos_php] -t  
[carpeta_donde_se_almacenan_los_archivos_de_documentacion]
```

Por ejemplo, genere la documentación en formato HTML (también es posible en formato PDF, CHM) de los proyectos de la carpeta /var/www y se almacene dicha documentación en la carpeta /var/www/docs/ el comando que introduciremos en la consola de comandos será **sudo phpdoc -o HTML:frames:phpedit -d /var/www/ -t /var/www/docs/**

Aunque existen un gran número de parámetros para adaptar el formato de la documentación que phpdoc genera, podemos obtener información sobre los formatos compatibles ejecutando el comando **sudo phpdoc -h**



5. Creación y utilización de plantillas

Una plantilla es un modelo que se puede utilizar para crear otros documentos. En **javadoc** son **sugerencias de código** asociadas a palabras clave una plantilla y se compone de:

- un **nombre**
- una **descripción**
- un **contexto** en función del lenguaje
- un **pattern**, que es el código de la plantilla.

La documentación del API de Java ha sido creada de este modo. Esto hace que el trabajo de documentar el código de nuevas clases Java sea trivial. Las plantillas se hallan definidas en **Preferences > Java > Editor > Templates** en **Eclipse** y en **Herramientas - Opciones - Editor - Plantillas de código** en **NetBeans**.

Lo más interesante es que nosotros podemos crearnos nuestras propias plantillas, además de modificar las existentes. Para ello no tenemos más que añadir una nueva desde la opción de "**Templates**", asignarle un nombre, descripción y elegir el código que queremos que se muestre al seleccionar la misma.

Es aconsejable examinar todas, ya que pueden ahorrar mucho trabajo ya que muchas de ellas utilizan nombres similares a las construcciones Java que **encapsulan** (try,for, while, if,...).

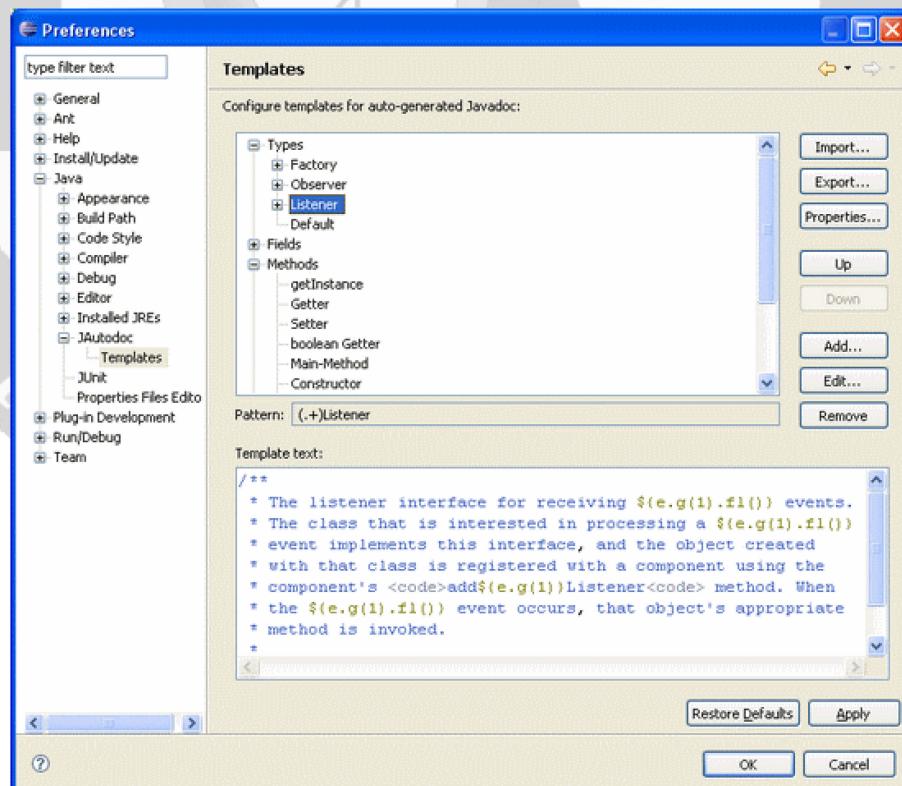


Imagen: Plantillas en NetBeans

Si se usan ciertas convenciones en el **código fuente Java** (como comenzar un comentario con `/**` y terminarlo con `*/`), **javadoc** puede fácilmente generar páginas HTML con el contenido de esos comentarios, que pueden visualizarse en cualquier navegador.

Dentro del código de la plantilla podemos usar texto fijo o una serie de variables predefinidas, por ejemplo:

- `${cursor}`: posición en la que se establecerá el cursor de texto tras desplegar el código de la plantilla.
- `${enclosing_type}`: tipo de la clase en la que nos encontramos.
- `${enclosing_method}`: nombre del método en el que nos encontramos.
- `${year}`: año en curso.
- `${time}`: hora en curso.



6. Instalación, configuración y uso de sistemas de control de versiones

Cuando se empieza el desarrollo de una aplicación, lo normal es que este desarrollo no se haga por un único desarrollador, sino que en él intervengan varios, los cuales se encargarán de partes distintas en el desarrollo.

Para poder elaborar un software de calidad, debemos de utilizar algún mecanismo que nos permita coordinar a todos los miembros que forman parte de este desarrollo. Una buena solución es el uso de un **sistema de control de versiones**.

Un sistema de control de versiones es un software, centralizado o no, que nos permite ir controlando los cambios que sufre la aplicación por cada miembro del equipo de desarrollo. Se utiliza para almacenar las diferentes versiones por las que pasa un proyecto durante su desarrollo.

En un sistema centralizado de control tenemos un servidor común donde se encuentra tanto el código fuente de la versión actual como de las distintas versiones, mientras en un sistema no centralizado de control no es necesario poseer un servidor común, sino que se pueden enviar y recibir actualizaciones de cada uno de los miembros del equipo de forma directa.

La forma de funcionar un sistema de control de versiones es muy sencilla. Cada miembro del equipo trabaja sobre la tarea que le han sido asignadas, y cuando hace algún tipo de cambio, este le llega al resto del equipo para que procedan a la actualización del código.



Imagen: Diferentes versiones de una aplicación

Git es un sistema rápido de control de versiones, está escrito en C y se ha hecho popular sobre todo a raíz de ser el elegido para el kernel de linux. Desde su nacimiento en 2005, Git ha evolucionado y madurado para ser fácil de usar y, aun así, conservar estas cualidades iniciales. Es tremadamente rápido, muy eficiente con grandes proyectos, y tiene un increíble sistema de ramificación (**branching**) para desarrollo no lineal.

La principal diferencia entre **Git** y cualquier otro **VCS (Subversion, etc)** es cómo Git modela sus datos. Conceptualmente, la mayoría de los demás sistemas almacenan la información como una lista de cambios en los archivos. Estos sistemas (CVS, **Subversion**, **Perforce**, **Bazaar**, etc.) modelan la información que almacenan como un conjunto de archivos y las modificaciones hechas sobre cada uno de ellos a lo largo del tiempo.



Imagen: Logo de git

Git no modela ni almacena sus datos de este modo, modela sus datos más como un conjunto de instantáneas de un mini sistema de archivos. Cada vez que se confirma un cambio, o se guarda el estado de tu proyecto en Git, él básicamente hace una "foto" del aspecto de todos tus archivos en ese momento, y guarda una referencia a esa instantánea. Para ser eficiente, si los archivos no se han modificado, Git no almacena el archivo de nuevo, sólo un enlace al archivo anterior idéntico que ya tiene almacenado.

6.1. Instalación de git

Git está instalado por defecto en Ubuntu 22.04 y podemos verificarlo usando el comando `git --version`

Sin embargo, si se desea disponer de la última versión de este software instalado para incluir útiles mejoras en la interfaz de usuario, es necesario instalar previamente unas librerías de las que git depende empleando el comando `sudo apt install libbz-dev libssl-dev libcurl4-gnutls-dev libexpat1-dev gettext cmake gcc`

A continuación, descargamos desde sitio web del proyecto git, la versión en tarball que se desea instalar. Por ejemplo, para descargar la versión 2.9.5 ejecutaremos `sudo curl -o git.tar.gz https://mirrors.edge.kernel.org/pub/software/scm/git/git-2.9.5.tar.gz && mv -f git.tar.gz /tmp`

Una vez descargado, descomprimiremos ejecutando la siguiente secuencia de comandos:

- `cd /tmp`
- `tar -zxf git.tar.gz`
- `cd git-*`

Una vez dentro del directorio con la fuente de git compilaremos la aplicación localmente ejecutando el comando `sudo make prefix=/usr/local all`

Tras esto, instalaremos la aplicación con el comando `sudo make prefix=/usr/local install`

Finalmente actualizaremos las variables del sistema preparamos el entorno para su instalación ejecutando el comando **exec bash**

Si el proceso de instalación se ha realizado correctamente, podremos validar con el comando **git --version** que hemos actualizado la aplicación a la versión que hemos descargado

6.2. Configuración de git

Git trae una herramienta, **git config**, que permite obtener y establecer variables de configuración que controlan el aspecto y funcionamiento de git. Esta herramienta la emplearemos tras instalar git, ya que lo primero que se debe hacer es establecer el nombre de usuario y dirección de correo electrónico. Esto es importante porque las confirmaciones de cambios (**commits**) en git usan esta información, y es introducida de manera inmutable en los **commits** que el usuario va a enviar.

Por tanto, ejecutaremos el siguiente comando para establecer el nombre de usuario **git config --global user.name "alumno"** y a continuación el siguiente comando para establecer su cuenta de correo electrónico **git config --global user.email alumno@example.com**

Solamente se necesita hacer esto una vez si se especifica la opción **--global**, ya que Git siempre usará esta información para todo lo que se haga en ese sistema. En el caso de querer sobrescribir esta información con otro nombre o dirección de correo para proyectos específicos, puedes ejecutar el mismo comando sin la opción **--global** cuando estemos en el proyecto concreto.

Git extrae los ajustes de configuración de una lista reducida de archivos. Cada nivel de configuración en el gráfico a continuación puede sobrescribir los ajustes de configuración en el nivel superior. Por ejemplo, la configuración global de git config sobrescribe la configuración del sistema de git config, pero no la configuración local de git config.

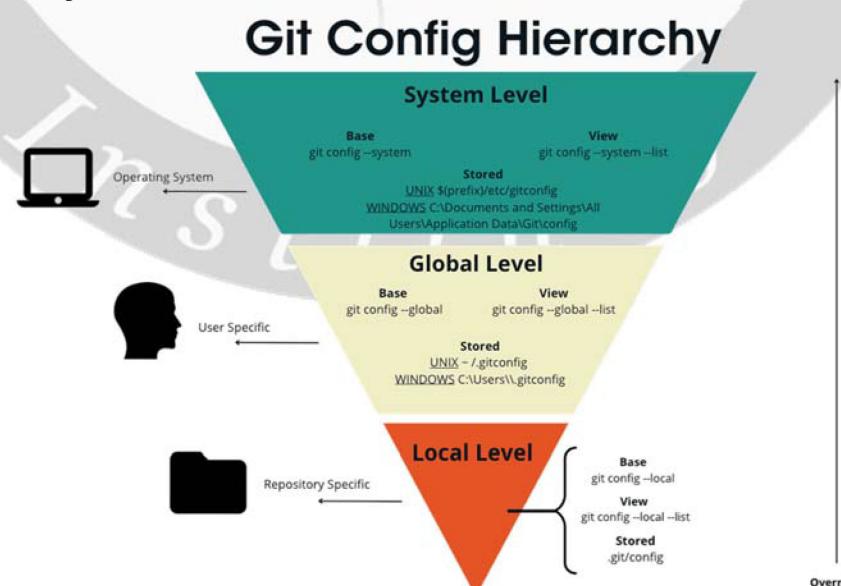


Imagen: Jerarquía de git config

Entre las opciones más comunes que tenemos con git config está elegir el editor de texto por defecto que se utilizará cuando git necesite que introduzcamos un mensaje. Si no se indica nada, git usa el editor por defecto del sistema que, generalmente, es Vi o Vim. En el caso de querer usar otro editor de texto, como emacs, ejecutaremos el comando `git config --global core.editor emacs`

Otra opción que puede ser interesante es configurar la herramienta de diferencias por defecto, usada para resolver conflictos de unión (`merge`). Git acepta `kdiff3`, `tkdiff`, `meld`, `xxdiff`, `emerge`, `vimdiff`, `gvimdiff`, `ecmerge` y `opendiff` como herramientas válidas. Por ejemplo, si deseamos usar vimdiff emplearíamos el comando `git config --global merge.tool vimdiff`

En cualquier momento podremos comprobar la configuración que tenemos mediante el comando `git config --list`



7. Operaciones avanzadas

Una vez que tenemos la aplicación **git** instalada, se recomienda validar la instalación de los manuales de ayuda para poder comprender las opciones disponibles de configuración.

Una vez configurado, hay que comprender en **git** como manejar desde consola los repositorios y el estado de los archivos, pues aquí es donde reside el verdadero sentido del control de versiones. Sin embargo, si nuestros usuarios van a realizar un uso muy básico del mismo, se recomienda emplear **Gitweb** como herramienta web para realizar el primer contacto con este tipo de aplicaciones.

7.1. Ayuda de git

Cuando necesitemos ayuda utilizando comandos de git tenemos tres modos de conseguir ver su página del manual (**manpage**):

- **git help <comando>**
- **git <comando> --help**
- **man git-<comando>**

La mayoría de las opciones que permiten configurar las preferencias personales de trabajo están en el lado cliente. Por ejemplo, para consultar una lista completa con todas las opciones contempladas en la versión instalada de git, se puede emplear el comando **git config -- help**

7.2. Instalación de Gitweb

Debido a la importancia que actualmente poseen las interfaces web, existe un entorno web de git, gitweb, que ofrece un aspecto más intuitivo y cómodo para el usuario.

Partimos de que en nuestra máquina en ya tiene instalado el servidor Apache Web, así que instalamos Gitweb con el comando **sudo apt-get install gitweb**.

Tras instalar la aplicación, creamos los directorios para estructurar nuestro modo de trabajo con Gitweb con el comando **sudo mkdir /home/usuario/gitweb**

Una vez creado el directorio de trabajo, lo añadiremos al archivo de configuración **/etc/gitweb.conf** que presenta el siguiente estado inicial de configuración:

```
# path to git projects (<project>.git)
$projectroot = "/var/lib/git";

# directory to use for temp files
$git_temp = "/tmp";

# target of the home link on top of all pages
#$home_link = $my_uri || "/";

# html text to include at home page
#$home_text = "indextext.html";

# file with project list; by default, simply scan the projectroot dir.
#$projects_list = $projectroot;
```

```

# stylesheet to use
#@stylesheets = ("static/gitweb.css");

# javascript code for gitweb
#$javascript = "static/gitweb.js";

# logo to use
#$logo = "static/git-logo.png";

```

Imagen: Contenido del archivo /etc/gitweb.conf

En nuestro caso introduciremos en la variable **\$projectroot** el directorio que hemos creado para gitweb.

Tras configurar Gitweb, podemos configurar el vhost de Gitweb en **/etc/apache2/conf-available/gitweb.conf** de la siguiente forma:

```

Alias /gitweb /usr/share/gitweb

<Directory /usr/share/gitweb>
    Options +FollowSymLinks +ExecCGI
    AddHandler cgi-script .cgi

    # Disponer aquí los permisos de acceso que considere
    Require ip 127.0.0.1 10.0.0.0/24
</Directory>

```

Si observas la configuración del vhost, se requiere emplear el módulo rewrite, si no lo hemos habilitado en nuestro servidor apache usaremos el **comando sudo a2enmod cgi rewrite**

Tras aplicar todos estos cambios reiniciamos el servidor apache con el comando **sudo systemctl restart apache2** y para acceder a la aplicación lo haremos en la dirección <http://localhost/gitweb>

7.3. Configurar proyectos sobre git

Para trabajar con un proyecto en git, lo primero que debemos hacer es crear la carpeta del mismo ejecutando la siguiente secuencia de comandos:

- **cd /var/lib/git/**
- **mkdir proyecto.git**
- **cd proyecto.git**

Una vez creada la carpeta, iniciamos un repositorio para nuestro nuevo proyecto y lo configuramos de acuerdo a nuestras necesidades con el comando **sudo git init** que nos devuelve un mensaje similar a "**Initialized empty Git repository in /home/foc/proyecto/.git/**"

El siguiente paso es configurar el proyecto en el directorio .git ejecutando los siguientes comandos:

- **sudo echo "Una breve descripción del proyecto" > .git/description**
- **sudo git config --global user.name "Tu nombre"**
- **sudo git config --global user.email "tu@correo.com"**

Una vez creados estos documentos nuevos en nuestro proyecto, ejecutamos el siguiente comando para que los archivos presentes en la carpeta del proyecto sean añadidos al repositorio **sudo git add .**

Una vez incorporados los ficheros nuevos realizamos la subida de los mismos al repositorio con el comando **sudo git commit -a**

Por último, podemos dar permisos de escritura a un usuario que no sea root, de tal manera que con dicho usuario se puedan hacer cambios remotos en el repositorio empleando la siguiente secuencia de comandos:

- **# adduser usuariogit**
- **# passwd usuariogit**
- **# chown -Rv usuariogit:usuariogit /var/cache/git/proyecto.git**

7.4. Gestionar versiones sobre git

Para acceder al repositorio podemos hacerlo de la manera convencional, basta con ejecutar el comando **sudo git clone git://servidor/proyecto.git proyecto** En el caso de ya tener una copia de un proyecto usando git clone, podemos actualizar a la última versión con **git pull**

El comando **git commit** sólo seguirá la pista de los archivos que estaban presentes la primera vez que se ejecutó **git add**. Si son añadidos nuevos archivos o subdirectorios, debe indicarse con el siguiente comando **git add ARCHIVOSNUEVOS**

De manera similar, si queremos eliminar del repositorio determinados archivos emplearemos el comando **git rm ARCHIVOSVIEJOS**

Renombrar un archivo es lo mismo que eliminar el nombre anterior y agregar el nuevo. Para ello podemos emplear el comando **git mv ARCHIVOIEJO ARCHIVONUEVO**

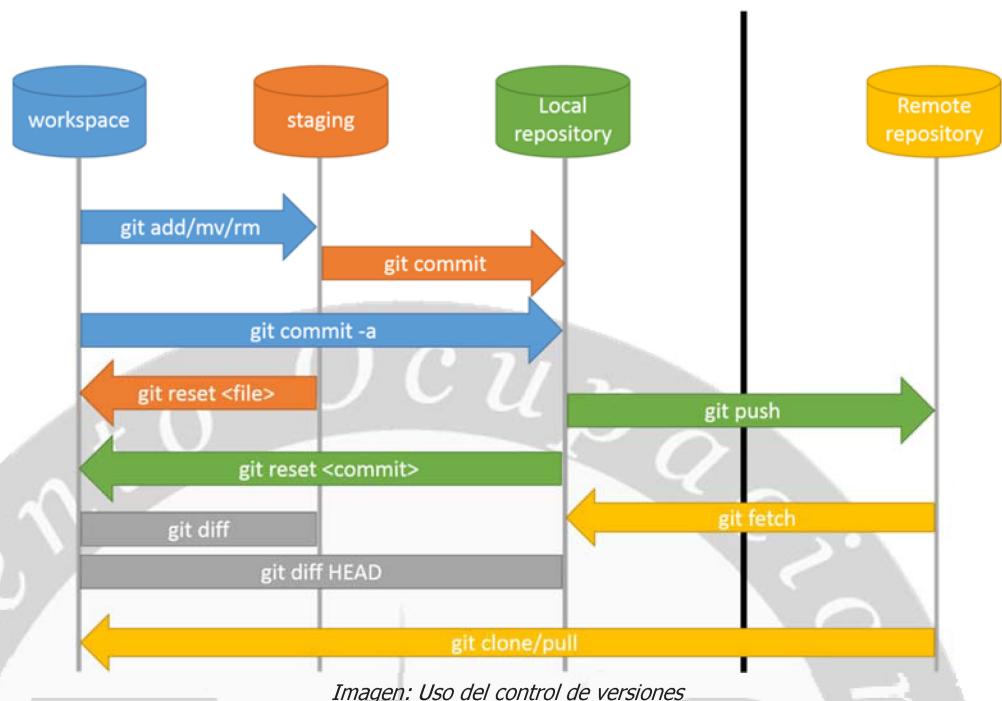
Algunas veces es interesante ir hacia atrás y borrar todos los cambios a partir de cierto punto porque, a lo mejor, estaban todos mal. Para ello utilizaremos el comando **git log** Ese comando muestra una lista de **commits** recientes, y sus hashes SHA1.

Si escribimos el comando **git reset --hard SHA1_HASH** recuperamos el estado de un commit dado y se borran para siempre cualquier recuerdo de commits más nuevos.

Otras veces es necesario saltar a un estado anterior temporalmente. En ese caso hay que escribir **git checkout SHA1_HASH** donde nos lleva atrás en el tiempo, sin tocar los commits más nuevos. Para volver a nuestro estado actual, emplearemos el comando **git checkout master**

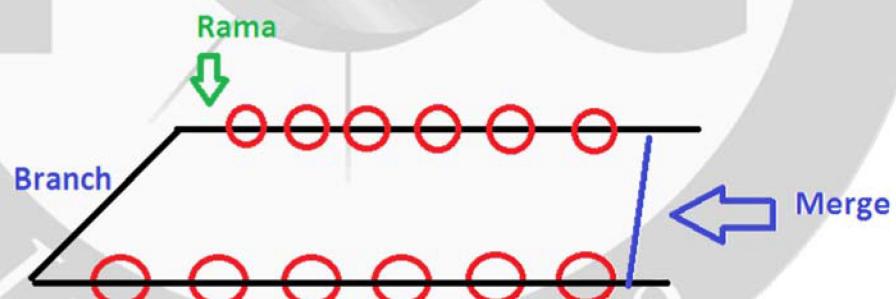
También existe la posibilidad de restaurar sólo archivos o directorios en particular, para ello escribiremos el comando **git checkout SHA1_HASH algun.archivo otro.archivo**

Esta forma de checkout puede sobrescribir archivos sin avisar. Para prevenir accidentes, es recomendable hacer commit antes de ejecutar cualquier comando de checkout.



En el caso de que queramos averiguar qué cambios hicimos desde el último commit ejecutaremos **git diff**

Otra de las ventajas de Git reside en la facilidad de crear nuevas ramas de trabajo, llamadas **branch**, donde probar nuevas características y hacer cambios complejos sin que afecte a la rama de trabajo principal. Luego, el proceso de fusión (**merge**), o la vuelta a un estado anterior son igual de fácil.



8. Seguridad de los sistemas de control de versiones

Git se encuadra en la categoría de los sistemas de gestión de código fuente distribuida. Con Git, cada directorio de trabajo local es un repositorio completo no dependiente, de acceso a un servidor o a la red. El acceso a cada **repositorio** se realiza a través del protocolo de Git, montado sobre **ssh**, o usando **HTTP**, aunque no es necesario ningún servidor web para poder publicar el repositorio en la red.

En el flujo de trabajo que hemos dibujado hasta el momento, los desarrolladores no subían directamente cambios al repositorio público, sino que era el responsable del mismo quien aceptaba los cambios y los incorporaba después de revisarlos. Sin embargo, **Git** también soporta que los desarrolladores puedan subir sus modificaciones directamente a un repositorio centralizado al más puro estilo **CVS** o **Subversion**.

Para que un repositorio público pueda ser utilizado de esta forma, es necesario permitir la ejecución del comando **push**. deberemos crear el repositorio público y habilitar alguno de los siguientes accesos:

- **Acceso directo** por sistema de ficheros con permisos de escritura para el usuario sobre el directorio del repositorio.
- **Acceso remoto vía SSH** (consultar man git-shell) y permisos de escritura para el usuario sobre el directorio del repositorio.
- **Acceso HTTP** con WebDav debidamente configurado.
- **Acceso mediante protocolo Git** con el servicio "receive-pack" activado en el git daemon.

El repositorio es el lugar en el que se almacenan los datos actualizados e históricos de cambios, a menudo en un servidor. A veces se le denomina **depósito** o **depot**. Puede ser un sistema de archivos en un disco duro, un banco de datos, etc.

El **sistema de control de versiones** mantiene un **registro** de las **dependencias** entre estos cambios y cuando llega el momento de realizar un lanzamiento, un conjunto particular de cambios es aprobado para ese lanzamiento.

- En el repositorio hay que poner los **ficheros de texto** (no muy grandes). No se deben poner los ficheros binarios (ejecutables) grandes, ya que suelen causar problemas de rendimiento (aparte de esto, no tiene sentido, ya que se regeneran a partir del código fuente).
- Los sistemas de control de versiones suelen permitir **agrupar los ficheros** y directorios bajo su control en proyectos o módulos
- El **código de una aplicación Web** por ejemplo puede ser un único módulo (todos los ficheros HTML, código fuente, plantillas CSS, ficheros de documentación, ...)
- Al actualizar la copia de trabajo se puede pedir una copia de todo un módulo
- Para una aplicación Web, el proyecto o módulo contendrá ficheros de código fuente, ficheros HTML, CSS, imágenes...

Versión 1 Versión 2 Versión 3 Versión 4 Versión 5 Versión 6



Imagen: Diferentes versiones

Cualquier otro fichero que se vaya a modificar alguna vez también:

- **Documentación propia** o de APIs externas, diagramas, indicaciones de instalación, ...
- **Ficheros de configuración**: ficheros de configuración del servidor Web (en Apache httpd.conf por ejemplo), de la base de datos, del framework, etc.
- Ficheros para **construir** la **aplicación**.



9. Instalación, configuración y uso de sistemas de integración continua del código. Monitorización continua de las métricas de calidad de la aplicación

La **integración continua (CI)** es una práctica de desarrollo de software, en la cual los programadores suben su código a un repositorio central donde automáticamente pasan las pruebas métricas y de calidad. Esta técnica se suele realizar regularmente para detectar fallos cuanto antes y así mantener el código siempre actualizado.

La integración continua constituye una parte fundamental del **enfoque DevOps** para la creación y lanzamiento de software, que promueve la colaboración, la automatización y los ciclos cortos de retroalimentación.

La práctica de la integración continua comienza con la confirmación de los cambios en un sistema de control de versiones/fuentes con regularidad, para que todos construyan sobre la misma base. Cada confirmación desencadena una compilación y una serie de pruebas automatizadas para verificar el comportamiento y asegurar que el cambio no ha estropeado nada. Aunque la integración continua es beneficiosa por sí misma, también es el primer paso hacia la implementación de un **proceso continuo de integración y despliegue (CI/CD)**.

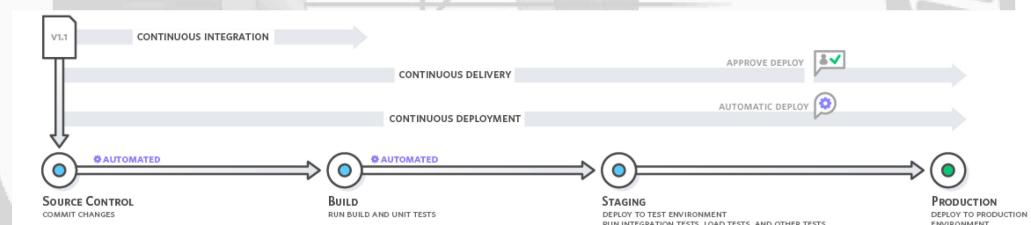


Imagen: CI en AWS © 2019 Amazon Web Services, Inc. o sus empresas afiliadas.

AWS ofrece varias maneras de realizar la CI, pero de forma nativa provee del servicio **AWS CodePipeline**, que le permite crear un flujo de trabajo que compila código en **AWS CodeBuild** cada vez que confirme un cambio. Sin embargo, si nuestras necesidades de integración continua son más modestas, podemos recurrir a soluciones como **Codeship**, **Circle CI**, **Jenkins CI** o **Travis CI** entre las más populares.

9.1. Instalación de sistemas de integración continua del código

Jenkins CI no es más que un sistema desplegado en un servidor que nos ayuda en la tarea de hacer integración continua y programar tareas automáticas cuando ocurra una determinada acción. A este tipo de servicios se los conoce como **CI/CD (Continuous Integration/Continuous Deploy)**



Imagen: Logotipo de Jenkins CI

Jenkins CI se integra a la perfección con Git para poder ejecutar tareas y tiene la posibilidad de instalar muchos plugins para extender su funcionalidad.

Para instalar Jenkins CI agregaremos las claves del repositorio oficial al sistema con el comando `curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo tee /usr/share/keyrings/jenkins-keyring.asc > /dev/null`

A continuación, agregaremos una entrada de repositorio apt para **Jenkins CI** con el siguiente comando `echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] https://pkg.jenkins.io/debian-stable binary/ | sudo tee /etc/apt/sources.list.d/jenkins.list > /dev/null`

Actualizaremos los paquetes locales, instalaremos previamente los paquetes necesarios para el funcionamiento de Jenkins y finalmente la propia aplicación con la siguiente secuencia de comandos:

- `sudo apt-get update`
- `sudo apt-get install fontconfig openjdk-11-jre`
- `sudo apt-get install jenkins`

Una vez finalizada la instalación, puede habilitar el servicio de Jenkins para que se inicie en el arranque con el comando `sudo systemctl enable jenkins` También es posible iniciar el servicio Jenkins con el comando `sudo systemctl start jenkins` o validar el estado del servicio Jenkins con el comando `sudo systemctl status jenkins`

Tras ejecutar Jenkins nos dirigimos a la ruta en local donde lo hayamos configurado, por defecto esta es `http://localhost:8080/` y nos aparecerá una imagen tal que así:

Getting Started

Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log ([not sure where to find it?](#)) and this file on the server:

`/var/jenkins_home/secrets/initialAdminPassword`

Please copy the password from either location and paste it below.

Administrator password



[Continue](#)

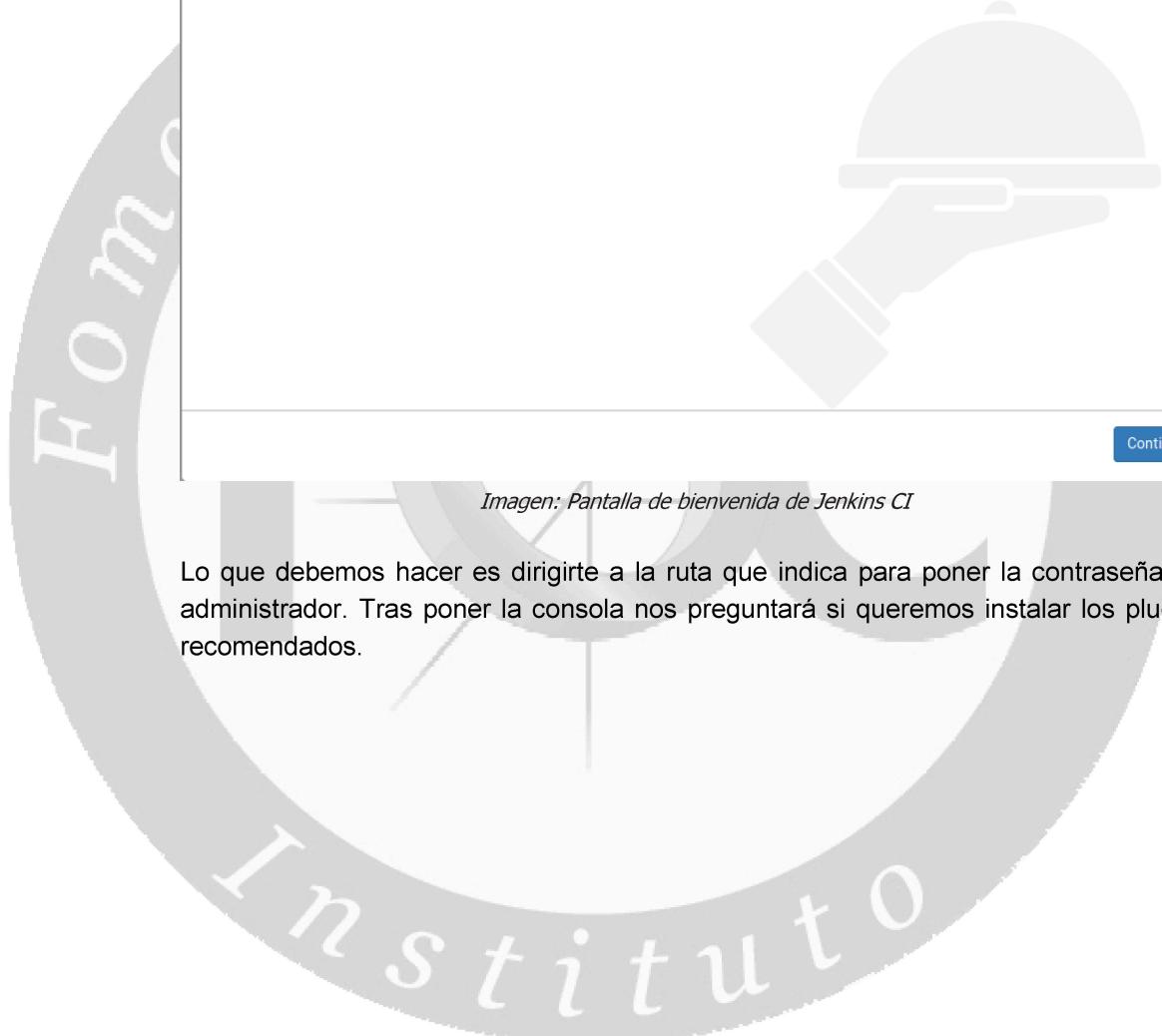


Imagen: Pantalla de bienvenida de Jenkins CI

Lo que debemos hacer es dirigirte a la ruta que indica para poner la contraseña del administrador. Tras poner la consola nos preguntará si queremos instalar los plugins recomendados.

Getting Started



Bienvenido a Jenkins

Plugins extend Jenkins with additional features to support many different needs.

Install suggested plugins

Install plugins the Jenkins community finds most useful.

Select plugins to install

Select and install plugins most suitable for your needs.

Jenkins 2.60.3

Imagen: Pantalla de instalación de plugins en Jenkins CI

Si elegimos manualmente, podremos seleccionar varias opciones ordenadas en categorías junto a su descripción. Los que aparecer marcados por defecto son los recomendados por el propio **Jenkins CI**.

Getting Started

Organization and Administration

Build Features

Build Tools

Build Analysis and Reporting

Pipelines and Continuous Delivery

Source Code Management

Distributed Builds

User Management and Security

Notifications and Publishing

All | None | Suggested Selected (20/57) x

Note that the full list of plugins is not shown here. Additional plugins can be installed in the Plugin Manager once the initial setup is complete. See the [Wiki](#) for more information.

Organization and Administration (2/3)

- Dashboard View ↗** Customizable dashboard that can present various views of job information. 16 ↵
- Folders ↗** This plugin allows users to create "folders" to organize jobs. Users can define custom taxonomies (like by project type, organization type etc). Folders are nestable and you can define views within folders. Maintained by CloudBees, Inc. 1 ↵
- OWASP Markup Formatter ↗** Uses the [OWASP Java HTML Sanitizer](#) to allow safe-seeming HTML markup to be entered in project descriptions and the like. 7 ↵

Build Features (4/10)

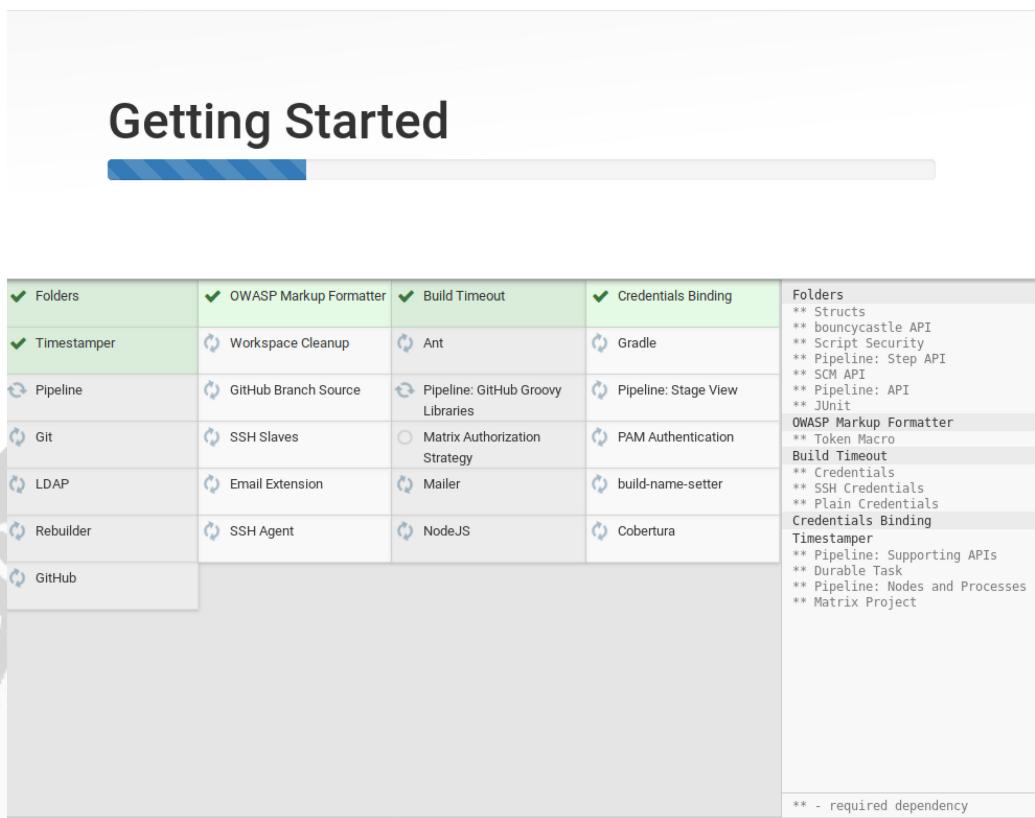
- build-name-setter ↗** This plug-in sets the display name of a build to something other than #1, #2, #3, ... 9 ↵
- Build Timeout ↗** This plugin allows builds to be automatically terminated after the specified amount of time has elapsed. 4 ↵
- Config File Provider ↗** Ability to provide configuration files (e.g. settings.xml for maven, XML, groovy, custom files,...) loaded through the UI which will be copied to the job workspace. 6 ↵
- Credentials Binding ↗** Allows credentials to be bound to environment variables for use from miscellaneous build steps. 6 ↵
- embeddable-build-status ↗** This plugin adds the embeddable build status badge to Jenkins so that you can easily hyperlink/show your 1 ↵

Jenkins 2.60.3 Back Install

Imagen: Pantalla de instalación manual de plugins en Jenkins CI

Tras seleccionar los plugins que deseemos, si pulsamos en install se instalarán Jenkins y los plugins que hemos seleccionado. Si posteriormente quieres añadir plugins nuevos lo puedes hacer desde el panel de control.

Getting Started



Jenkins 2.60.3

Imagen: Pantalla de instalación de Jenkins

Finalmente, el asistente de **Jenkins CI** pedirá un usuario para usar con Jenkins, aunque podemos continuar como admin.

9.2. Configuración y uso de sistemas de integración continua del código

Tras instalar **Jenkins CI** y acceder a la aplicación, nos aparecerá la pantalla principal de la aplicación.



Imagen: Pantalla principal de Jenkins CI

Jenkins CI se basa en tareas, trabajos o un conjunto de instrucciones que podemos programar para que ocurran con una determinada acción. Para crear una tarea en Jenkins pulsaremos sobre el enlace **Crear nueva tarea** y nos mostrará un asistente donde introduciremos el nombre a la tarea. A continuación, seleccionaré **Crear proyecto estilo libre** para configurarlo.

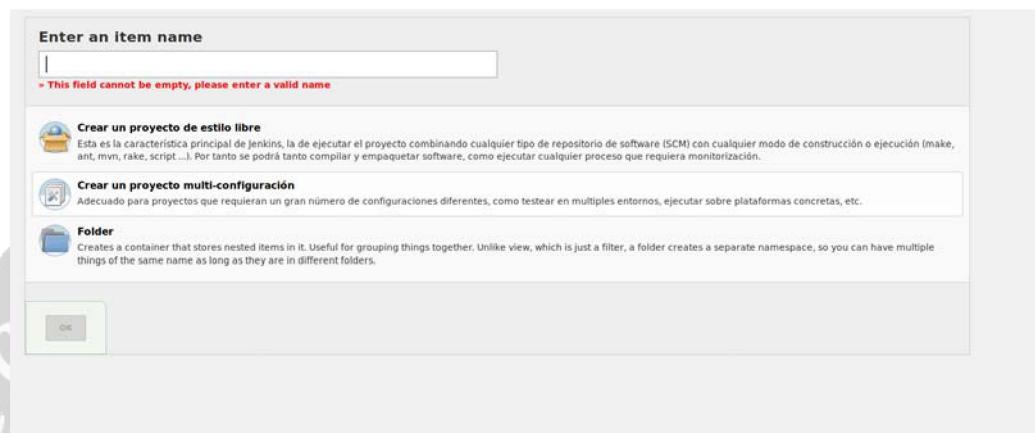


Imagen: Pantalla asistente Crear nueva tarea de Jenkins CI

Entre las opciones desplegadas, que dependerán de los plugings instalados, encontrarás **Origen del código**

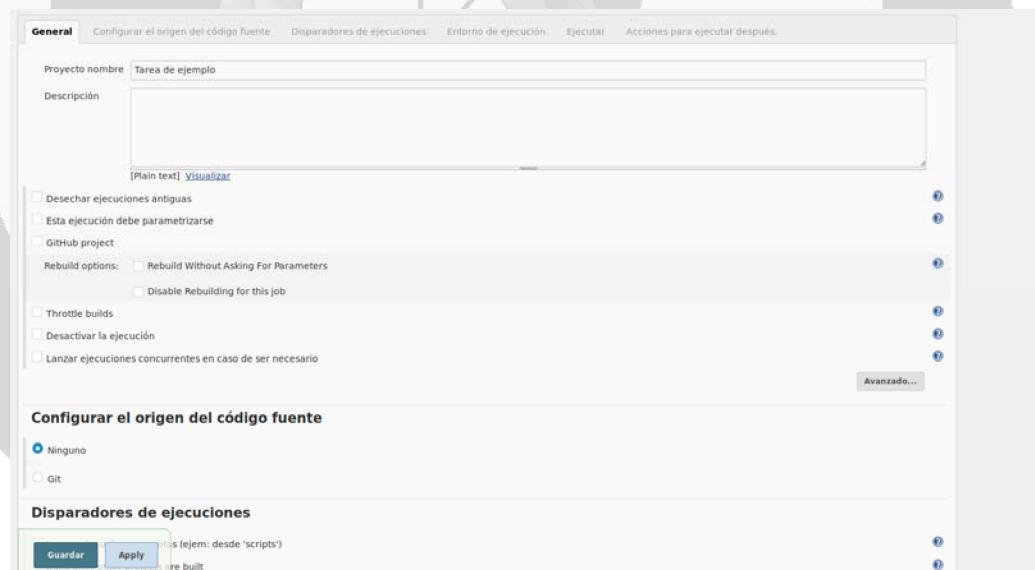


Imagen: Pantalla asistente General de Crear nueva tarea de Jenkins CI

Por ejemplo, supongamos que tenemos un repositorio remoto, en ese caso seleccionaremos git, pegaremos la dirección del repositorio de Github y seleccionaremos el origen de la rama, credenciales y qué disparador vamos a querer.

Las ejecuciones pueden ser desde scripts, cuando se haga otro build, periódicamente (en cuyo caso hay que indicar cada cuánto tiempo), disparadores para gitscm y periódicamente consultando el repositorio.

En este caso vamos a escoger consultar repositorio e introducimos H/15** para que se ejecute si hay un cambio en el repositorio cada 15 minutos. El sistema de programación de tareas es igual al empleado por el gestor de tareas del sistema crontab

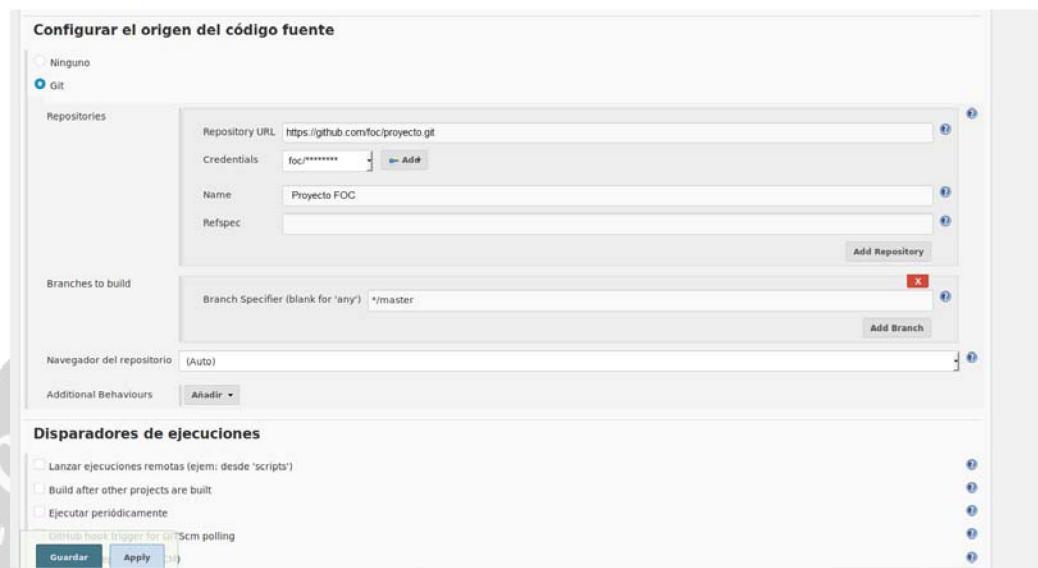


Imagen: Selección del código fuente

Al final encontramos dónde queremos que se ejecuten las tareas y qué queremos ejecutar antes y después. Desde las opciones de entorno de ejecución podemos configurar cómo se van a ejecutar los build, si vamos a poner ficheros de configuración, por ejemplo, o si vamos a indicar rutas para los PATH de node y npm, o de si vamos a darle un nombre específico a la ejecución (si tenemos instalado el plugin), etc.

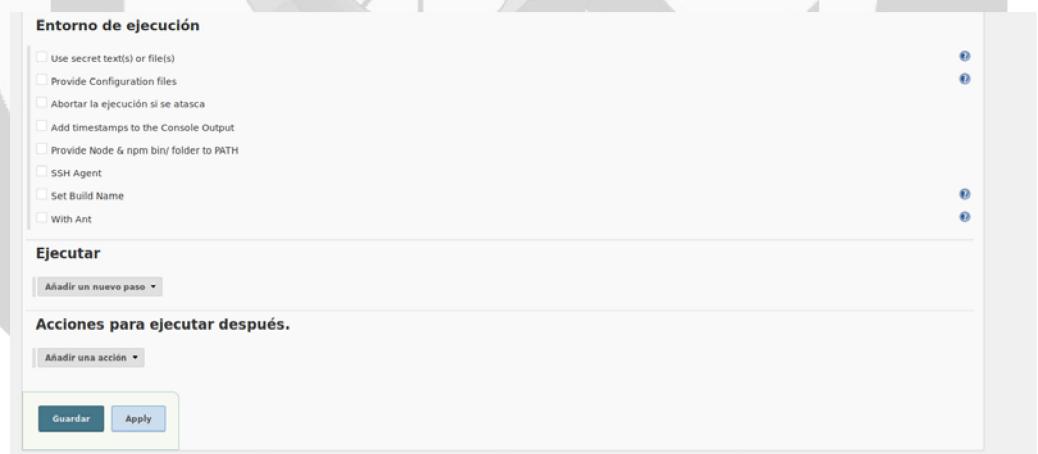


Imagen: Entorno de ejecución

Para decidir qué queremos ejecutar hay muchas opciones dependiendo del tipo de proyecto que vayamos a ejecutar, pueden ser comandos de línea de comandos, comandos nodejs, comandos de windows, etc. También se pueden encadenar distintos pasos por ejemplo ejecutar un script de shell y luego uno de windows.

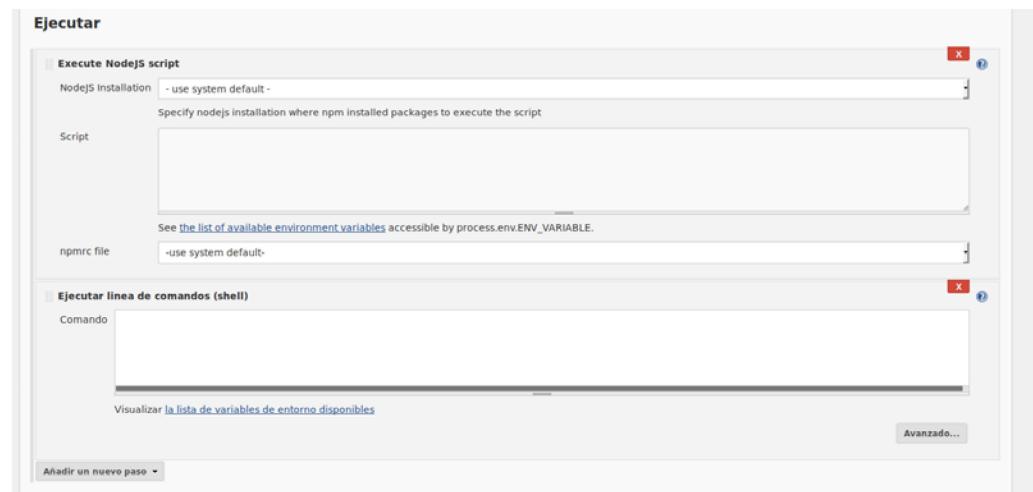


Imagen: Selección de lo que queremos ejecutar

Por último, hay que indicar que hacemos al finalizar la tarea: si notificamos a usuarios, guardar logs, imprimir resultados, hacer commit, publicar cobertura etc.

Tras crear la tarea tendrás acceso al panel de control de la tarea, desde aquí podrás seleccionar si lanzarla manualmente, ver los logs, las builds anteriores, volver a configurar la tarea, etc.

Imagen: Tarea reportando un error

Una vez ejecutada una tarea, si se ha producido algún error, aparecerá un ícono del último build en rojo indicando que algo ha fallado. Si hacemos clic sobre el build, podemos ver su estado y la salida de consola para ver exactamente qué ha fallado.

9.3. Monitorización continua de las métricas de calidad de la aplicación

Las métricas son una herramienta esencial para mejorar el rendimiento del sistema: ayudan a identificar dónde se puede añadir valor y ofrecen una base de referencia para medir el impacto de cualquier mejora efectuada.

Al medir y monitorizar la velocidad de las actividades, la calidad de nuestro software y el grado de uso de la automatización, puede identificar áreas de mejora y confirmar si los cambios realizados han surtido un efecto positivo.

Las siguientes cuatro métricas han sido identificadas por **DevOps Research and Assessment (DORA)** como **métricas de alto nivel** que indican con precisión el rendimiento de una organización en el contexto del desarrollo de software:

- **Plazo de entrega:** El enfoque adoptado por DORA consiste en medir el tiempo que transcurre desde que se confirma el código hasta su implementación, lo que le permite centrarse únicamente en las etapas que se encuentran dentro del ámbito de su proceso de CI/CD. Un plazo de entrega prolongado significa que no se están presentando cambios de código a los usuarios con regularidad y, por lo tanto, no está disfrutando de las ventajas de la retroalimentación por parte del cliente para perfeccionar lo que se está creando.
- **Frecuencia de implementación:** Es el registro del número de veces que utiliza su proceso de CI/CD para implementar a producción. La frecuencia de implementación fue seleccionada por DORA como un proxy para el tamaño de lotes, ya que una alta frecuencia de implementación implica menos cambios por implementación. Implementar un pequeño número de cambios con frecuencia reduce el riesgo asociado al lanzamiento (porque hay menos variables que pueden combinarse para arrojar resultados inesperados), y proporciona retroalimentación más pronto.
- **Tasa de fallos por cambios:** Se refiere a la proporción de los cambios implementados en producción que dan lugar a un fallo, como una interrupción o un error que requiere una reversión o un hotfix. La ventaja de esta métrica es que sitúa las implementaciones fallidas en el contexto del volumen de cambios realizados. Una baja tasa de fallos en los cambios debería darle confianza en su proceso; indica que las primeras etapas del proceso están haciendo su trabajo y detectando la mayoría de los defectos antes de que su código se implemente en producción.
- **Tiempo medio de recuperación:** El tiempo medio de recuperación o resolución (MTTR) mide el tiempo que se tarda en solucionar un fallo de producción. Un MTTR bajo requiere una supervisión proactiva de su sistema en producción para alertarle de los problemas a medida que surgen, y la capacidad de revertir los cambios o implementar una solución rápidamente a través del proceso.

Además de estas medidas de alto nivel, hay una serie de **métricas operativas** que puede utilizar para comprender mejor el rendimiento de su proceso e identificar las áreas en las que podría mejorarlo. Entre las más comunes podemos destacar las siguientes:

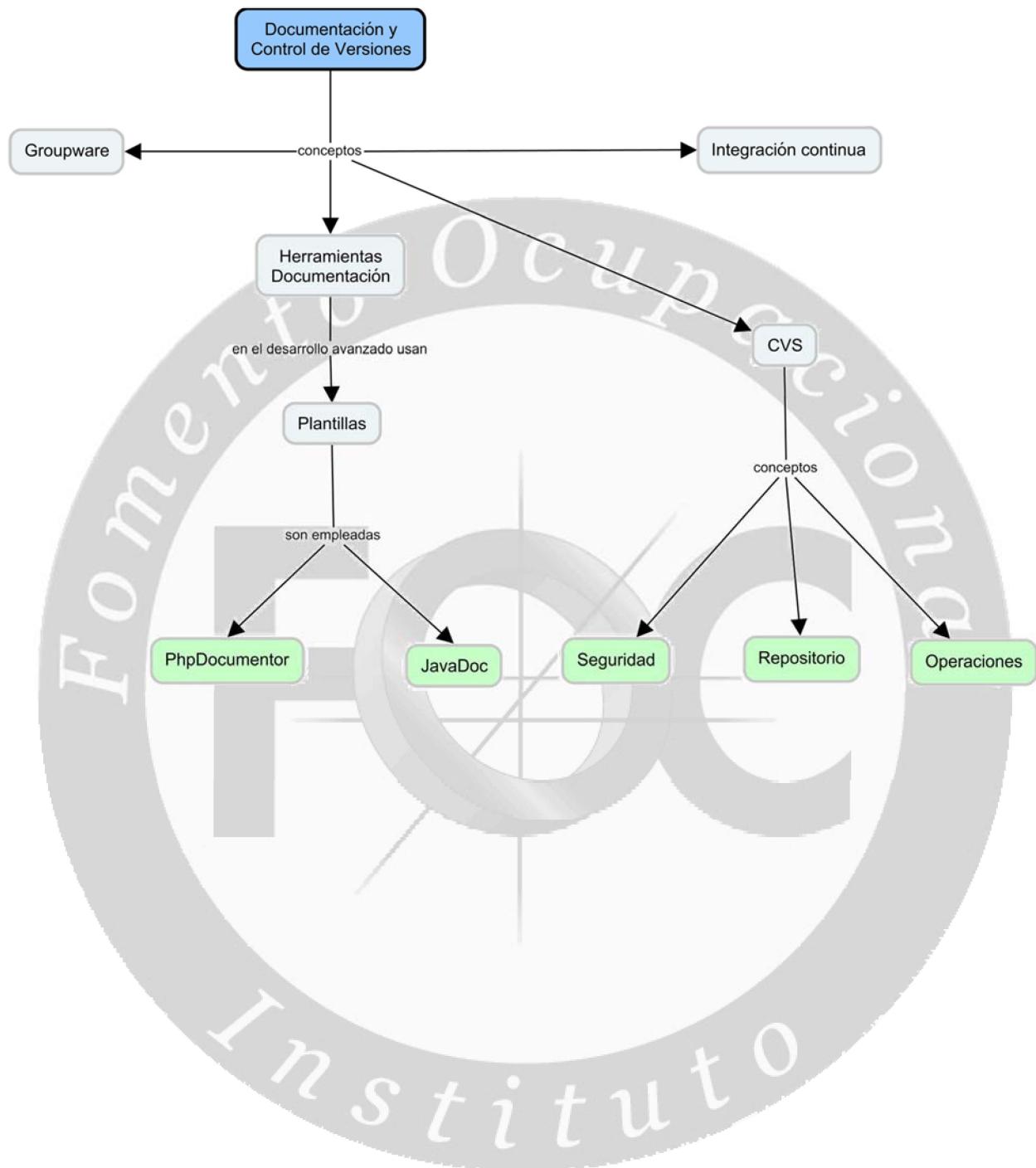
- **Cobertura de código:** calcula la proporción de su código que cubren las pruebas de unidades. Merece la pena controlar esta métrica para asegurarse de que mantiene una cobertura de pruebas adecuada a medida que escribe más código.
- **Duración de la compilación:** mide el tiempo que se tarda en completar las distintas etapas del proceso automatizado. Observar el tiempo invertido en cada etapa del proceso es útil para detectar los puntos débiles o los cuellos de botella que podrían estar ralentizando el tiempo total que se tarda en obtener el feedback de las pruebas o en implementar para los usuarios.
- **Tasa de superación de pruebas:** es el porcentaje de casos de prueba que se han superado con éxito en una compilación determinada. Mientras tenga un nivel razonable de pruebas automatizadas, esta es una buena indicación de la calidad de cada compilación. Puede utilizar esta métrica para comprender la frecuencia con la que los cambios en el código dan lugar a pruebas fallidas.
- **Tiempo de resolución de pruebas:** es el tiempo que transcurre entre que una compilación informa de que una prueba ha fallado y la misma prueba se supera en una compilación posterior. Esta métrica le indica la rapidez con la que es capaz de responder a los problemas identificados en el proceso.
- **Implementaciones fallidas:** Se utiliza para calcular la tasa de fallos por cambios. La supervisión de la proporción de fallos sobre el número total de implementaciones ayuda a medir su rendimiento con respecto a los acuerdos de nivel de servicio. Sin embargo, hay que tener en cuenta que un objetivo de cero (o muy pocas) implementaciones fallidas no es necesariamente realista, y puede animar a los equipos a dar prioridad a la seguridad. Si se hace así, los plazos de entrega son más largos y las implementaciones más grandes, ya que los cambios se agrupan, lo que aumenta la probabilidad de que se produzcan fallos en la producción (puesto que se unen más variables) y hace que sean más difíciles de solucionar (ya que hay más cambios que revisar).
- **Recuento de defectos:** se refiere al número de tickets abiertos en su backlog clasificados como errores. También puede desglosarse por incidencias encontradas en pruebas o en fase de puesta en escena, y por incidencias encontradas en producción.
- **Tamaño de la implementación:** se mide por el número de puntos de historia incluidos en una compilación o lanzamiento y puede utilizarse para controlar el tamaño de los lotes dentro de un equipo concreto. Mantener las implementaciones reducidas muestra que su equipo confirma con frecuencia, con todos los beneficios que ello conlleva. Sin embargo, como las estimaciones de las historias no son comparables entre los equipos de desarrollo, esta métrica no debería utilizarse para medir el tamaño total de la implementación.

10. Resumen

En esta unidad se ha aprendido lo siguiente:

- Una vez que hemos realizado la instalación y configuración de nuestros servidores web, aplicaciones y ftp, y hemos publicado nuestra aplicación web, es necesario establecer que debemos documentar, ya que la documentación es un proceso vital en el ciclo de vida del software. Para ello, hemos conocido que son las herramientas de **generación de documentación**, como se instala y configuran dos de ellas (phpdocumentor y Javadoc), que se usan en el desarrollo de aplicaciones web.
- Hemos conocido el concepto de documentación, y una vez presentadas las herramientas de documentación en PHP y en Java, pasamos a ver cómo se **instala, configura** y usa **phpdocumentor** y como se instala, configura y usa **Javadoc**.
- Hemos aprendido a cómo crear y utilizar **plantillas** para realizar la **documentación**.
- Se han presentado las **herramientas colaborativas**, como una de los mecanismos más utilizados actualmente, para el desarrollo de aplicaciones web por parte de un grupo de desarrollo de software. Se han indicado algunas de las más populares.
- Dado que, en cualquier proceso de desarrollo de software, se producen una ingente cantidad de material, se modifican constantemente los archivos, es necesario gestionar todo este proceso de forma fácil y efectiva. Por este motivo, se ha estudiado qué son y para que se utilizan los **sistemas de control de versiones**.
- En nuestro caso, hemos optado por trabajar con una de las herramientas de gestión de versiones más robusta y de una amplia difusión como es **GIT**.
- Se ha estudiado la instalación, configuración y uso del sistema de gestión de versiones **GIT**.
- Las herramientas de gestión de versiones, hemos aprendido que son los **repositorios**, como núcleo central donde se almacena las diferentes versiones de un documento.
- Hemos estudiado la **documentación** que se realiza sobre el sistema de gestión de versiones.
- Hemos visto varios sistemas de integración continua y entre ellos hemos aprendido a instalar y configurar **Jenkins CI** para realizar tareas simples.
- Hemos reconocido las distintas monitorizaciones que se deben dar en los procesos CI.

11. Mapa Conceptual



12. Bibliografía

Desarrollo de Aplicaciones Web

Mateu, Carles.

UOC. 2012.

Desarrollo de Aplicaciones Web II

Moseley, Ralph.

Anaya, 2007.

Documentación oficial de Amazon Web Services

Amazon Web Services Inc.

Disponible en: <<https://docs.aws.amazon.com/>> [Accedido el 29 de junio 2022].

Documentación oficial de Apache HTTP Server Version 2.4.

The Apache Software Foundation.

Disponible en: <<https://httpd.apache.org/docs/2.4/>> [Accedido el 29 de junio 2022].

Documentación de git

Linux Torvalds.

Disponible en: <<https://git-scm.com/docs>> [Accedido el 29 de junio 2022].

Documentación de Jenkins

Jenkins CI.

Disponible en: <<https://www.jenkins.io/doc/book/>> [Accedido el 29 de junio 2022].

Servidores para Internet con Apache HttpServer

Egea, Francisco Javier.

Eidos 2008

ISBN: 978-84-88457-19-7