# CI-CD

## Continuous Integration - Continuous Deployment

(Teórica)

tiiMiit by

# Introducción

Bienvenidos al módulo de CI-CD, Continuous Integration and Continuous deployment. Esta es la práctica moderna que los DevOps utilizan para la entrega de nuevas versiones de software y su integración en el entorno de producción de manera ágil y confiable.

Una vez finalizado este módulo serás capaz de:

- Entender el proceso de Integración continua
- Entender el proceso de implementación continua
- Conocer herramientas de mercado para realizar estos trabajos

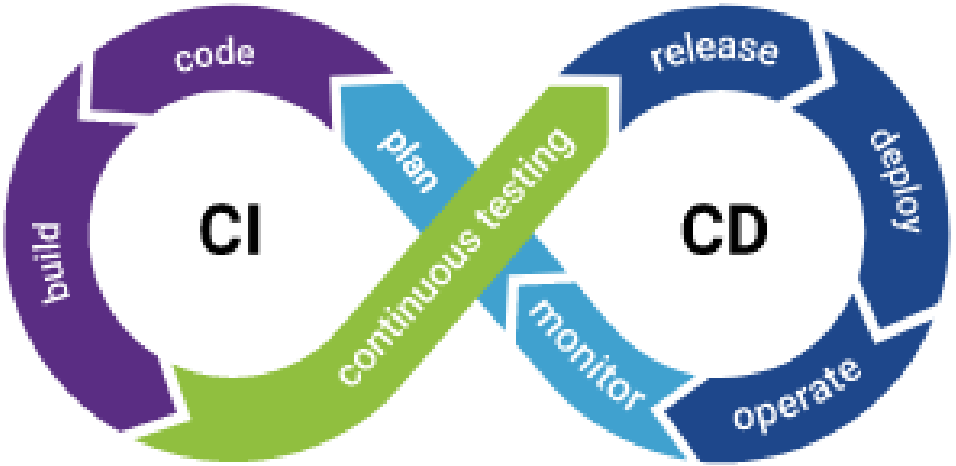
# Tema 1. Concepto de CI-CD

## Objetivo

El objetivo de este tema es presentar CI - CD con el fin de tomar conocimiento de los conceptos principales de estos procesos.

## Definición

CI y CD significan integración continua y entrega continua/implementación continua. En términos muy simples, CI es una práctica moderna de desarrollo de software en la que se realizan cambios de código incrementales con frecuencia y de manera confiable. Los pasos automatizados de compilación y prueba activados por CI garantizan que los cambios de código que se fusionan en el repositorio sean confiables. Luego, el código se entrega de forma rápida y sin problemas como parte del proceso de CD. En el mundo del software, la canalización de CI/CD se refiere a la automatización que permite que los cambios de código incrementales desde los escritorios de los desarrolladores se entreguen de manera rápida y confiable a la producción.




## ¿Por qué es importante CI/CD?

CI/CD permite a las organizaciones enviar software de manera rápida y eficiente. CI/CD facilita un proceso efectivo para llevar los productos al mercado más rápido que nunca, entregando continuamente el código en

producción y asegurando un flujo continuo de nuevas funciones y correcciones de errores a través del método de entrega más eficiente.

## ¿Cuál es la diferencia entre CI y CD?

La integración continua (CI) es una práctica que implica que los desarrolladores realicen pequeños cambios y comprobaciones en su código. Debido a la escala de requisitos y la cantidad de pasos involucrados, este proceso está automatizado para garantizar que los equipos puedan crear, probar y empaquetar sus aplicaciones de manera confiable y repetible. CI ayuda a agilizar los cambios de código, lo que aumenta el tiempo para que los desarrolladores realicen cambios y contribuyan a mejorar el software.

La entrega continua (CD) es la entrega automatizada de código completo a entornos como pruebas y desarrollo. El CD proporciona una forma automatizada y consistente de entregar el código a estos entornos.

La implementación continua es el siguiente paso de la entrega continua. Cada cambio que pasa las pruebas automatizadas se coloca automáticamente en producción, lo que da como resultado muchas implementaciones de producción.

La implementación continua debe ser el objetivo de la mayoría de las empresas que no están limitadas por requisitos normativos o de otro tipo.

En resumen, CI es un conjunto de prácticas realizadas mientras los desarrolladores escriben código, y CD es un conjunto de prácticas realizadas después de completar el código .

## ¿Cómo se relaciona CI/CD con DevOps?

DevOps es un conjunto de prácticas y herramientas diseñadas para aumentar la capacidad de una organización para entregar aplicaciones y servicios más rápido que los procesos tradicionales de desarrollo de

software. La mayor velocidad de DevOps ayuda a una organización a servir a sus clientes con más éxito y a ser más competitiva en el mercado. En un entorno DevOps, las organizaciones exitosas “integran la seguridad” en todas las fases del ciclo de vida del desarrollo, una práctica llamada DevOps .

La práctica clave de DevSecOps es integrar la seguridad en todos los flujos de trabajo de DevOps. Al realizar actividades de seguridad de manera temprana y consistente a lo largo del ciclo de vida de desarrollo de software (SDLC), las organizaciones pueden asegurarse de detectar vulnerabilidades lo antes posible y estar en mejores condiciones para tomar decisiones informadas sobre riesgos y mitigación. En las prácticas de seguridad más tradicionales, la seguridad no se aborda hasta la etapa de producción, que ya no es compatible con el enfoque DevOps más rápido y ágil. Hoy en día, las herramientas de seguridad deben encajar perfectamente en el flujo de trabajo del desarrollador y en la canalización de CI/CD para seguir el ritmo de DevOps y no ralentizar la velocidad de desarrollo.

La canalización de CI/CD es parte del marco más amplio de DevOps. Para implementar y ejecutar con éxito una canalización de CI/CD, las organizaciones necesitan herramientas para evitar puntos de fricción que ralentizan la integración y la entrega. Los equipos requieren una cadena de herramientas integrada de tecnologías para facilitar los esfuerzos de desarrollo colaborativos y sin obstáculos.

## ¿Qué herramientas se requieren para las canalizaciones de CI/CD?

Uno de los mayores desafíos que enfrentan los equipos de desarrollo que utilizan una canalización de CI/CD es abordar adecuadamente la seguridad. Es fundamental que los equipos incorporen seguridad sin ralentizar sus ciclos de integración y entrega. Mover las pruebas de seguridad a una etapa más temprana del ciclo de vida es uno de los pasos más importantes para lograr este objetivo. Esto es especialmente cierto para las organizaciones de DevOps que confían en las pruebas de seguridad automatizadas para mantenerse al día con la velocidad de entrega.

La implementación de las herramientas adecuadas en el momento adecuado reduce la fricción general de DevOps, aumenta la velocidad de lanzamiento y mejora la calidad y la eficiencia.

## ¿Cuáles son los beneficios de CI/CD?

Las pruebas automatizadas permiten la entrega continua, lo que garantiza la calidad y la seguridad del software y aumenta la rentabilidad del código en producción.

Las canalizaciones de CI/CD permiten un tiempo de comercialización mucho más corto para las características de nuevos productos, creando clientes más felices y reduciendo la tensión en el desarrollo.

El gran aumento en la velocidad general de entrega que permiten las canalizaciones de CI/CD mejora la ventaja competitiva de una organización.

La automatización libera a los miembros del equipo para que se concentren en lo que mejor saben hacer, generando los mejores productos finales.

Las organizaciones con una canalización exitosa de CI/CD pueden atraer grandes talentos. Al alejarse de los métodos tradicionales en cascada , los ingenieros y desarrolladores ya no están empantanados con actividades repetitivas que a menudo dependen en gran medida de la realización de otras tareas.

# Tema 2. ¿Qué es GIT?

## Objetivo

El objetivo de este tema es conocer qué es un sistema de versionamiento de código distribuido y tomar conocimiento de que es GIT y que es GITHUB.

## Que es Git

Git es un sistema de control de versiones distribuido, diseñado por Linus Torvalds. Está pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente.

Git está optimizado para guardar cambios de forma incremental.

Permite contar con un historial, regresar a una versión anterior y agregar funcionalidades.

Lleva un registro de los cambios que otras personas realicen en los archivos.

Git fue diseñado para operar en un entorno Linux. Actualmente, es multiplataforma, es decir, es compatible con Linux, MacOS y Windows. En la máquina local se encuentra Git, se utiliza bajo la terminal o línea de comandos y tiene comandos como merge, pull, add, commit y rebase, entre otros.

## Para qué proyectos sirve Git

Con Git se obtiene una mayor eficiencia usando archivos de texto plano, ya que con archivos binarios no puede guardar solo los cambios, sino que debe volver a grabar el archivo completo ante cada modificación, por mínima que sea, lo que hace que incremente demasiado el tamaño del repositorio.

“Guardar archivos binarios en el repositorio de Git no es una buena práctica, únicamente deberían guardarse archivos pequeños (como logos) que no sufran casi modificaciones durante la vida del proyecto. Los binarios deben guardarse en un CDN”.

## Características de Git

Git almacena la información como un conjunto de archivos.

No existen cambios, corrupción en archivos o cualquier alteración sin que Git lo sepa.

Casi todo en Git es local. Es difícil que se necesiten recursos o información externos, basta con los recursos locales con los que cuenta.

Git cuenta con 3 estados en los que es posible localizar archivos: Staged, Modified y Committed.

## ¿Qué es un sistema de control de versiones?

El SCV o VCS (por sus siglas en inglés) es un sistema que registra los cambios realizados sobre un archivo o conjunto de archivos a lo largo del tiempo, de modo que puedas llevar el historial del ciclo de vida de un proyecto, comparar cambios a lo largo del tiempo, ver quién los realizó o revertir el proyecto entero a un estado anterior.

Cualquier tipo de archivo que se encuentre en un ordenador puede ponerse bajo control de versiones.

## ¿Qué es Github?

Github es una plataforma de desarrollo colaborativo para alojar proyectos utilizando el sistema de control de versiones Git. Se emplea principalmente para la creación de código fuente de programas informáticos.

Puede considerarse a Github como la red social de código para los programadores y en muchos casos es visto como un curriculum vitae, pues aquí se guarda el portafolio de proyectos de programación.

## Características de Github

GitHub permite alojar proyectos en repositorios de forma gratuita y pública, pero tiene una forma de pago para privados.

Puedes compartir fácilmente tus proyectos.

Permite colaborar para mejorar los proyectos de otros y a otros mejorar o aportar a los tuyos.

Ayuda a reducir significativamente los errores humanos, a tener un mejor mantenimiento de distintos entornos y a detectar fallos de una forma más rápida y eficiente.

Es la opción perfecta para poder trabajar en equipo en un mismo proyecto.

Ofrece todas las ventajas del sistema de control de versiones Git, pero también tiene otras herramientas que ayudan a tener un mejor control de los proyectos.

## Git vs GitHub

Al comienzo puede ser tentador pensar que Git y GitHub son lo mismo. Pero en realidad no lo son. En realidad es posible utilizar Git sin GitHub! En definitiva, los dos existen por diferentes razones.

Git es un Sistema de Control de Versiones Distribuido (DVCS) utilizado para guardar diferentes versiones de un archivo (o conjunto de archivos) para que cualquier versión sea recuperable cuando lo desee.

Git también facilita el registro y comparación de diferentes versiones de un archivo. Esto significa que los detalles sobre qué cambió, quién cambió qué, o quién ha iniciado una propuesta, se pueden revisar en cualquier momento.

## ¿Qué significa "distribuido"?

El término "distribuido" significa que cuando le instruyes a Git que comparta el directorio de un proyecto, Git no sólo comparte la última versión del archivo. En cambio, distribuye cada versión que ha registrado para ese proyecto.

Este sistema "distribuido" tiene un marcado contraste con otros sistemas de control de versiones. Ellos sólo comparten cualquier versión individual que un usuario haya explícitamente extraído desde la base de datos central/local.

Bueno, entonces "distribuido" significa distribuir todas – no solo algunas seleccionadas – versiones de los archivos del proyecto que Git haya registrado. Pero qué es exactamente un sistema de control de versiones?

## ¿Qué es un Sistema de Control de Versiones?

Un Sistema de Control de Versiones (VCS) se refiere al método utilizado para guardar las versiones de un archivo para referencia futura.

De manera intuitiva muchas personas ya utilizan control de versiones en sus proyectos al renombrar las distintas versiones de un mismo archivo de varias formas como blogScript.js, blogScript_v2.js, blogScript_v3.js, blogScript_final.js, blogScript_definite_final.js, etcétera. Pero esta forma de abordarlo es propensa a errores e inefectivo para proyectos grupales.

Además, con esta forma de abordarlo, rastrear qué cambió, quién lo cambió y porqué se cambió, es un esfuerzo tedioso. Esto resalta la importancia de un sistema de control de versiones confiable y colaborativo como Git.

Sin embargo, para obtener lo mejor de Git, es importante entender cómo Git administra tus archivos.

## Estados de los archivos en Git

En Git hay tres etapas primarias (condiciones) en las cuales un archivo puede estar: estado Modified, estado Staged, o estado Committed.

Estado Modified

Un archivo en el estado modificado es un archivo revisado – pero no acometido (sin registrar).

En otras palabras, archivos en el estado modificado son archivos que has modificado pero no le has instruido explícitamente a Git que controle.

Estado Staged

Archivos en la etapa preparada son archivos modificados que han sido seleccionados – en su estado (versión) actual – y están siendo preparados para ser guardados (acometidos) al repositorio .git durante la próxima instantánea de confirmación.

Una vez que el archivo está preparado implica que has explícitamente autorizado a Git que controle la versión de ese archivo.

Estado Committed

Archivos en el estado confirmado son archivos que se guardaron en el repositorio .git exitosamente.

Por lo tanto un archivo confirmado es un archivo en el cual has registrado su versión preparada en el directorio (carpeta) Git.

El estado de un archivo determina la ubicación donde Git lo colocará.

## Ubicación de archivos

Existen tres lugares principales donde pueden residir las versiones de un archivo cuando se hace control de versiones con Git: el directorio de trabajo, el sector de preparación, o el directorio Git.

Directorio de trabajo

El directorio de trabajo es una carpeta local para los archivos de un proyecto. Esto significa que cualquier carpeta creada en cualquier lugar en un sistema es un directorio de trabajo. Los archivos en el estado modificado residen dentro del directorio de trabajo.

El directorio de trabajo es distinto al directorio .git. Es decir, tu creas un directorio de trabajo mientras que Git crea un directorio .git.

Zona de preparación

La zona de preparación – técnicamente llamado “index” en lenguaje Git – es un archivo normalmente ubicado en el directorio .git, que guarda información sobre archivos próximos a ser acometidos en el directorio .git.

- Los archivos en la etapa de preparación residen en la zona de preparación.

Directorio Git

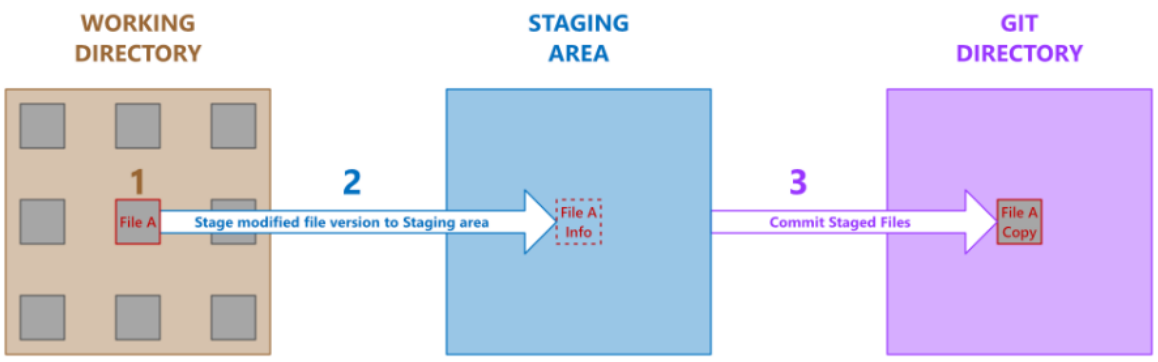
El directorio.git es la carpeta (también llamada "repositorio") que Git crea dentro del directorio de trabajo que le has instruido para realizar un seguimiento.

La carpeta .git también es donde Git guarda las bases de datos de objetos y metadatos de los archivos que le hayas instruido realizar un monitoreo.

- El directorio.git es la vida de Git – es el ítem copiado cuando se clona un repositorio desde otra computadora (o desde una plataforma en línea como GitHub).
- Los archivos en el estado acometido residen en el directorio Git.

## El flujo de trabajo básico de Git

Trabajar con el Sistema de Control de Versiones Git se ve algo así:




1. Diagrama básico del flujo de trabajo de Git

2. Modificar archivos en el directorio de trabajo.
3. Observe que cualquier archivo que modifiques se convierte en un archivo en el estado modificado.
4. Prepara selectivamente los archivos que quieras confirmar al directorio .git.
5. Observe que cualquier archivo que prepares (agregues) a la zona de preparación se convierte en un archivo en el estado preparado.
6. También tenga en cuenta que los archivos preparados todavía no están en la base de datos .git.
7. Preparar significa que la información sobre el archivo preparado se incluye en un archivo (llamado "index") en el repositorio .git.
8. Confirma el/los archivos que has preparado en el directorio .git. Esto es, guardar de manera permanente una instantánea de los archivos preparados en la base de datos .git.
9. Observe que cualquier versión del archivo que confirmes al directorio .git se convierte en un archivo en el estado confirmado.

## Lo esencial hasta ahora

En resumidas cuentas todo lo discutido hasta el momento es que Git es un sistema de control de versiones genial para versionado, administración y distribución de archivos.

Pero espera un segundo, si Git nos ayuda en la administración y distribución eficaz de distintas versiones de los archivos de un proyecto, ¿cuál es el propósito de GitHub?

## Considerando todo

Git y GitHub son dos entidades diferentes que te ayudan a administrar y alojar archivos. En otras palabras, Git sirve para controlar las versiones de los archivos mientras que GitHub es una plataforma para alojar tus repositorios Git.

# Tema 3. ¿Qué es Jenkins?

## Objetivo

Conocer esta herramienta soporte para trabajos CI-CD

## Introducción

La integración continua es una práctica de desarrollo software donde los miembros del equipo integran su trabajo frecuentemente (como mínimo una vez al día, aunque normalmente se realizan múltiples integraciones diarias).

Cada integración se verifica compilando el código fuente y obteniendo un ejecutable (a esto se le llama build, y debe hacerse de forma automatizada). Además también se pasan las pruebas y métricas de calidad para detectar los errores tan pronto como sea posible.

Es muy recomendable hacer builds periódicamente y comprobar que funcionen correctamente, para conseguir un producto final más fiable, con menos fallos en la producción.

Al integrar frecuentemente el código, y con la ayuda de herramientas como Jenkins, puedes saber el estado del software en todo momento. Sabes qué funciona, qué no y qué errores hay.

También puedes monitorizar la calidad del código y su cobertura de pruebas. La integración continua incluso puede ayudarte a reducir la deuda técnica (aquí puedes saber más sobre la deuda técnica) y mantener los costes bajos.

¿Cómo conseguir todo esto? Para esto existe la herramienta Jenkins!

## Jenkins

Jenkins es un servidor de integración continua, gratuito, open-source y actualmente uno de los más empleados para esta función. Además es muy fácil de utilizar. Lo puedes descargar desde aquí.

Esta herramienta, proviene de otra similar llamada Hudson, ideada por Kohsuke Kawaguchi, que trabajaba en Sun. Unos años después de que Oracle comprara Sun, la comunidad de Hudson decidió renombrar el proyecto a Jenkins, migrar el código a Github y continuar el trabajo desde ahí. No obstante, Oracle ha seguido desde entonces manteniendo y trabajando en Hudson.

## ¿Qué papel juega Jenkins dentro del proceso de integración continua?

La base de Jenkins son las tareas, donde indicamos qué es lo que hay que hacer en un build. Por ejemplo, podríamos programar una tarea en la que se compruebe el repositorio de control de versiones cada cierto tiempo, y cuando un desarrollador quiera subir su código al control de versiones, este se compile y se ejecuten las pruebas.

Si el resultado no es el esperado o hay algún error, Jenkins notificará al desarrollador, al equipo de QA, por email o cualquier otro medio, para que lo solucione. Si el build es correcto, podremos indicar a Jenkins que intente integrar el código y subirlo al repositorio de control de versiones.

Pero la cosa no queda ahí. Una de las cosas buenas que tiene Jenkins es que además de poder ayudarte a integrar el código periódicamente, puede actuar como herramienta que sirva de enlace en todo el proceso de desarrollo.

Desde Jenkins podrás indicar que se lancen métricas de calidad y visualizar los resultados dentro de la misma herramienta. También podrás ver el resultado de los tests, generar y visualizar la documentación del proyecto o incluso pasar una versión estable del software al entorno de QA para ser probado, a pre-producción o producción.

## Cosas a tener en cuenta para llevar a cabo la integración continua con Jenkins

En primer lugar, es imprescindible tener un repositorio de control de versiones (mercurial, git, svn, plastic, etc). Todo lo necesario para realizar el build debe estar allí (código, scripts de test, librerías de terceros...).

Sin esto, no podremos sacar el máximo partido a Jenkins, ya que uno de sus puntos fuertes es que es capaz de monitorizar el control de versiones y actuar ante cualquier cambio.

Además, para que la integración continua funcione, es imprescindible que el equipo está mentalizado y comprometido.

Por ejemplo (y puede que de tanto repetirlo se haga pesado, pero créeme, hay veces que esto no se sigue. Fíjate en este post, que cuenta una experiencia real lo importante que es el control de versiones ) todo el código debe subirse al control de versiones. Los desarrolladores deben subir su trabajo periódicamente al repositorio. Además los proyectos tienen que tener un proceso de build automático, y si un build falla, lo más prioritario debe ser arreglarlo.

Aún así, y sin dudarlo, con todo lo que nos aporta esta buena práctica, merece la pena llevarla a cabo. Y merece la pena utilizar Jenkins.

# Cierre

En este documento hemos presentado el concepto de CI-CD. Este concepto diseñado hace unos años ha permitido reducir costos operativos en las puestas en producción de nuevas versiones, dando robustez al control del proceso, desde el control de versionamiento de código y desde el control y agilidad de nuevas versiones en los ambientes de producción.

# Referencias

https://sentrio.io/blog/que-es-jenkins/#:~:text=Jenkins%20es%20un%20servidor%20open,nuevas%20versiones%20a%20los%20usuarios

https://kryptonsolid.com/que-es-jenkins-y-como-funciona/

https://www.udemy.com/course/introduction-to-continuous-integration-and-continuous-delivery/?utm_source=adwords&utm_medium=udemyads&utm_campaign=LongTail_la.EN_cc.ROW&utm_content=deal4584&utm_term=_._ag_77879424134_._ad_535397245863_._kw__._de_c_._dm__._pl__._ti_dsa-1007766171312_._li_20009_._pd__._&matchtype=&gclid=CjwKCAjwv4SaBhBPEiwA9YzZvEt1DqS63STwJOBOxtz1zRw1wUz1GZD9FVEjw17mpJsiDjfE3n189hoCiyUQAvD_BwE

https://www.jetbrains.com/space/?source=google&medium=cpc&campaign=14889552915&term=continuous%20integration%20deployment&gclid=CjwKCAjwv4SaBhBPEiwA9YzZvDWB01s4wgqAobh7YGo4uir3WvWqN7vX5PX-l4SMQ1HXKH4p23_v5xoClxUQAvD_BwE

https://www.redhat.com/es/topics/devops/what-is-ci-cd

tiimiit by