# Una introducción a la integración, entrega e implementación continuas de software.

En un artículo cuya autoría pertenece a Justin Ellingwood y publicada en el blog de Digital Ocean, en idioma inglés y bajo licencia Creative Commons 4.0 (la cual permite distribuir y adpatar el trabajo allí publicado) pues hoy día nos hemos dado a la tarea de **traducir y analizar** dicha entrada web. Para que queden las cosas claras, primero haremos la traducción y los subtítulos dependientes de ella están claramente indicado en la tabla de contenido que genera automáticamente el excelente complemento para WordPress llamado TablePress hecho Tobias Bäthge. Luego, y a continuación, con el título de "Nuestro análisis" agregaremos y desglosaremos nuestras personalísimas consideraciones acerca del tema. **La imagen que acompaña al tema también está traducida al castellano, siempre amparada según la licencia que rige en el sitio web original de la entrada al blog de Digital Ocean.**

---

# Traducción.

## Introducción.

El desarrollo y distribución de software puede ser un proceso complicado, especialmente a medida que las aplicaciones, los equipos y la infraestructura de implementación crecen en complejidad. A menudo, los desafíos se vuelven más pronunciados a medida que crecen los proyectos. Para desarrollar, probar y distribuir software de manera rápida y consistente, los desarrolladores y las organizaciones han creado tres estrategias relacionadas pero distintas para administrar y automatizar estos procesos.

- La **Integración Continua ("*Continuous integration*")** se centra en la integración del trabajo de los desarrolladores individuales en un repositorio principal varias veces al día para detectar errores de integración de manera temprana y acelerar el desarrollo colaborativo.
- La **Distribución Continua ("*Continuous delivery*")** tiene que ver con la reducción de la fricción en el proceso de implementación o liberación, la automatización de los pasos necesarios para implementar una compilación para que el código se pueda liberar de forma segura en cualquier momento.
- La **Implementación Continua ("*Continuous deployment*")** va incluso un paso más allá al implementarse automáticamente cada vez que se realiza un cambio de código.

En esta guía, discutiremos cada una de estas estrategias, cómo se relacionan entre sí y cómo el

incorporarlas al ciclo de vida de nuestras aplicaciones puede transformar nuestros desarrollos de software y nuestras prácticas de distribución. Para tener una mejor idea de las diferencias entre varios proyectos Integración Continua / Distribución Continua de código abierto, consulte nuestra comparación de herramientas al respecto (en idioma inglés, *N. del T.*).

## ¿Qué es la Integración Continua y por qué es útil?

La **Integración Continua** es una práctica que alienta a los desarrolladores a integrar su código en una rama principal de un repositorio compartido de forma temprana y frecuente. En lugar de crear características de forma aislada e integrarlas al final de un ciclo de desarrollo, cada desarrollador integra el código con el repositorio compartido varias veces a lo largo del día.

La idea es minimizar el costo de la integración haciéndo una prueba de integración temprana. Los desarrolladores pueden descubrir conflictos en los límites entre del código nuevo y del existente desde el principio, lo que hace que los conflictos aún sean relativamente fáciles de reconciliar. Una vez que se resuelve el conflicto, el trabajo puede continuar con la confianza de que el nuevo código respeta los requisitos de la base de código existente.

La integración del código con frecuencia no ofrece, en sí misma, ninguna garantía sobre la calidad del nuevo código o funcionalidad. En muchas organizaciones, la integración es costosa porque los procesos manuales se utilizan para garantizar que el código cumpla con los estándares, no presente errores y no rompa la funcionalidad existente. La integración frecuente puede crear fricción cuando el nivel de automatización no coincide con las medidas de garantía de calidad establecidas.

Para abordar esta fricción dentro del proceso de integración, en la práctica, la **Integración Continua** se basa en un conjunto de pruebas sólidos y un sistema automatizado para ejecutar esas pruebas. Cuando un desarrollador combina código en el repositorio principal, los procesos automatizados inician una compilación del nuevo código. Posteriormente, los entornos de prueba se ejecutan contra la nueva compilación para comprobar si se introdujeron problemas de integración. Si falla la fase de compilación o la fase de prueba, se alerta al equipo para que pueda trabajar en la reparación de la compilación.

El objetivo final de la integración continua es hacer que sea un proceso simple y repetible y que forme parte del flujo de trabajo de desarrollo diario a fin de reducir los costos de integración y responder a los defectos de manera temprana. Trabajar para garantizar que el sistema sea robusto, automatizado y rápido, mientras se cultiva una cultura de equipo que fomenta la interacción frecuente y la capacidad de respuesta para generar problemas es fundamental para el éxito de la estrategia.

## ¿Qué es la distribución Continua y por qué es útil?

La **Distribución Continua** es una extensión de la **Integración Continua**. Se centra en la automatización del proceso de entrega de software para que los equipos puedan implementar su código de forma fácil y confiable en cualquier momento. Al asegurar que la base de código siempre se encuentre en un estado desplegable, la distribución del software se convierte en un evento sin complicaciones y sin un ritual complicado. Los equipos pueden estar seguros de que pueden lanzar cuando lo necesiten, sin una coordinación compleja o pruebas de última etapa. Al igual que con la **Integración Continua**, la entrega continua es una práctica que requiere una combinación de mejoras técnicas y organizativas para ser efectiva.

Desde el punto de visto tecnológico, la **Distribución Continua** se apoya fuertemente en los canales de despliegue para automatizar los procesos de prueba e implementación. Una canalización de implementación es un sistema automatizado que ejecuta conjuntos de prueba cada vez más rigurosos contra una compilación como una serie de etapas secuenciales. Esto se inicia cuando la **Integración Continua** ha finalizado una etapa, por lo que una configuración de **Integración Continua** confiable es un requisito previo para implementar la **Distribución Continua**.

En cada etapa, la compilación o bien falla las pruebas (lo que alerta al equipo) o bien pasa las pruebas, lo que da como resultado una promoción automática a la siguiente etapa. A medida que la trabajo se mueve a través de la *cadena de procesos*, las etapas posteriores implementan la construcción en entornos que reflejan el entorno de producción lo más cerca posible. De esta forma, la compilación, el proceso de implementación y el entorno pueden probarse en *tándem*. La *cadena de procesos* termina con una compilación que se puede implementar en producción en cualquier momento y en un solo paso.

Los aspectos organizativos de la **Distribución Continua** fomentan la priorización de la "capacidad de despliegue" como una preocupación principal. Esto tiene un impacto en la forma en que las características se construyen y se enganchan en el resto de la base de código. Se debe pensar en el diseño del código para que las características se puedan implementar de forma segura en la producción en cualquier momento, incluso cuando esté incompleto. **Han surgido varias técnicas para ayudar en esta área.**

La **Distribución Continua** es atractiva porque automatiza los pasos entre verificar el código en el repositorio y decidir si lanzar versiones funcionales bien probadas a su infraestructura de producción. Los pasos que ayudan a afirmar la calidad y corrección del código son automáticos, pero la decisión final sobre qué lanzar se deja en manos de la organización para la máxima flexibilidad.

## ¿Qué es la Implementación Continua y por qué es útil?

La **Implementación Continua** es una extensión de la **Distribución Continua** e implementa automáticamente cada versión que pasa por el ciclo de prueba completo. En lugar de esperar a

que un controlador de acceso humano *(persona, N. del T.)* decida qué y cuándo desplegarlo en la producción, un sistema de **Implementación Continua** instala todo lo que ha atravesado con éxito la canalización de implementación. Tenga en cuenta que, si bien el nuevo código se implementa automáticamente, existen técnicas para activar nuevas características en un momento posterior o para un subconjunto de usuarios. La implementación lleva automáticamente las características y soluciones a los clientes rápidamente, fomenta los cambios más pequeños con un alcance limitado y ayuda a evitar la confusión sobre lo que se implementa actualmente en la producción.

Este ciclo de implementación completamente automatizado puede ser una fuente de ansiedad para las organizaciones preocupadas por renunciar al control de su sistema de automatización de lo que se libera o publica. La compensación ofrecida por las implementaciones automatizadas a veces se considera demasiado peligrosa para la recompensa que proporcionan.

Otros grupos aprovechan la promesa de la publicación automática como un método para garantizar que siempre se sigan las mejores prácticas y para extender el proceso de prueba a un entorno de producción limitado. Sin una verificación manual final antes de implementar un fragmento de código, los desarrolladores deben asumir la responsabilidad de garantizar que su código esté bien diseñado y que los bancos de pruebas estén actualizados. Esto colapsa la decisión de qué y cuándo combinar con el repositorio principal y qué y cuándo liberar a la producción en un único punto que existe firmemente en las manos del equipo de desarrollo.

La implementación continua también permite a las organizaciones beneficiarse de la retroalimentación temprana constante. Las características pueden ponerse inmediatamente a disposición de los usuarios y los defectos o las implementaciones inútiles se pueden detectar antes de que el equipo dedique un gran esfuerzo en una dirección improductiva. Recibir comentarios rápidos de que una característica no es útil permite al equipo cambiar de enfoque en lugar de sumergir más energía en un área con un impacto mínimo.

## Conceptos clave y prácticas clave para los Procesos Continuos.

Si bien la integración, entrega e implementación continuas varían en el alcance de su participación, existen algunos conceptos y prácticas que son fundamentales para el éxito de cada uno.

### Cambios pequeños e iterativos.

Una de las prácticas más importantes al adoptar la integración continua es alentar los cambios pequeños. Los desarrolladores deben practicar el dividir el trabajo más grande en pequeños pedazos y combinarlos al repositorio apenas estén listos. Las técnicas especiales, como la rama por abstracción y las marcas de características (ver a continuación) ayudan a proteger la funcionalidad de la rama principal de los cambios en el código de progreso.

Los pequeños cambios minimizan la posibilidad y el impacto de los problemas de integración. Al integrarlos al repositorio en la rama correspondiente, en la etapa más temprana posible, y luego continuamente a lo largo del desarrollo, el costo de la integración disminuye y el trabajo no relacionado se sincroniza regularmente.

### Desarrollo basado en tronco.

Con el desarrollo basado en troncos, el trabajo se realiza en la rama principal del repositorio o se fusiona en el repositorio compartido a intervalos frecuentes. Las ramas con característica de vida corta son permisibles siempre que representen pequeños cambios y se fusionen lo antes posible.

La idea detrás del desarrollo basado en tronco es evitar grandes compromisos que violan el concepto de pequeños cambios iterativos discutidos anteriormente. El código está disponible para los pares de manera temprana, para que los conflictos se puedan resolver cuando su alcance es pequeño.

Las versiones se realizan desde la rama principal o desde una rama de publicación creada desde la línea troncal específicamente para ese propósito. No se produce ningún desarrollo en las ramas de publicación para mantener así el enfoque en la rama principal como la única fuente de verdad.

### Mantener rápidas las fases de construcción y prueba.

Cada uno de los procesos se basa en construcciones *(compilaciones, N. del T.)* y pruebas automatizadas para validar la corrección. Debido a que los pasos de compilación y prueba deben realizarse con frecuencia, es esencial que estos procesos se simplifiquen para minimizar el tiempo dedicado a estos pasos.

Los aumentos en el tiempo de construcción deberían tratarse como un problema importante porque el impacto se complica por el hecho de que cada compromiso inicia una construcción. Debido a que los procesos continuos obligan a los desarrolladores a comprometerse con estas actividades a diario, la reducción de la fricción en estas áreas es una actividad que vale la pena.

Cuando sea posible, ejecutar diferentes secciones del conjunto de pruebas en paralelo puede ayudar a mover la construcción a través de la *cadena de procesos* de manera más rápida. También se debe tener cuidado para asegurarse de que la proporción de cada tipo de prueba tenga sentido. Las pruebas unitarias suelen ser muy rápidas y tienen una sobrecarga de mantenimiento mínima. En contraste, el sistema automatizado o las pruebas de aceptación a menudo son complejas y propensas a la rotura. Para dar cuenta de esto, a menudo es una buena idea confiar en gran medida en las pruebas unitarias, realizar un buen número de pruebas de integración y luego retroceder en la cantidad de pruebas para realizarlas de manera posterior y más complejas.

### Consistencia a lo largo de la *cadena de procesos* de implementación.

Debido a que se supone que las implementaciones de entrega continua o implementación evalúen la solidez de la versión, es esencial mantener la coherencia durante cada paso del proceso: la construcción en sí misma, los entornos de implementación y el proceso de implementación en sí. Veamos:

- **El código debe construirse una vez al comienzo de la *cadena de procesos*:** el software resultante se debe almacenar y acceder a los procesos posteriores sin reconstrucción. Al usar el mismo artefacto exacto en cada fase, puede estar seguro de que no está presentando incoherencias como resultado de diferentes herramientas de compilación.
- **Los entornos de implementación deben ser coherentes:** un sistema de gestión de configuración puede controlar los diversos entornos y los cambios medioambientales pueden pasar a través de la propia ruta de implementación para garantizar la corrección y la coherencia. Se deben aprovisionar entornos de implementación limpios en cada ciclo de prueba para evitar que las condiciones heredadas comprometan la integridad de las pruebas. Los entornos de ensayo deben coincidir lo más posible con el entorno de producción para reducir los factores desconocidos presentes cuando se promueve la compilación.
- **Se deben usar procesos consistentes para implementar la construcción en cada entorno:** cada implementación debe ser automática y cada implementación debe usar las mismas herramientas y procedimientos centralizados. Las implementaciones *ad-hoc* deben eliminarse a favor de implementar solo con las herramientas de canalización.

### Desacoplar implementación y lanzamiento.

Separar la implementación del código desde su lanzamiento a los usuarios es una parte extremadamente poderosa de la entrega e implementación continua. El código se puede implementar en la producción sin activarlo inicialmente o hacerlo accesible a los usuarios. Luego, la organización decide cuándo lanzar nuevas funcionalidades o características independientes de la implementación.

Esto brinda a las organizaciones una gran flexibilidad al separar las decisiones comerciales de los procesos técnicos. Si el código ya está en los servidores, la implementación ya no es una parte delicada del proceso de lanzamiento, lo que minimiza el número de personas y la cantidad de trabajo involucrado en el momento del lanzamiento.

Hay una serie de técnicas que ayudan a los equipos a implementar el código responsable de una característica sin liberarla. Los indicadores de características configuran la lógica condicional para verificar si se debe ejecutar el código según el valor de una variable de entorno. La *Rama por Abstracción* permite a los desarrolladores reemplazar implementaciones al colocar una capa de

abstracción entre los consumidores de recursos y los proveedores. Una planificación cuidadosa para incorporar estas técnicas le brinda la capacidad de desacoplar estos dos procesos.

## Tipos de pruebas.

### Pruebas de humo.

Las pruebas de humo son un tipo especial de controles iniciales diseñados para asegurar una funcionalidad muy básica, así como algunas implementaciones básicas y suposiciones ambientales. Las pruebas de humo generalmente se llevan a cabo al comienzo de cada ciclo de prueba como una verificación de cordura antes de ejecutar un conjunto de pruebas más completo.

La idea detrás de este tipo de prueba es ayudar a atrapar grandes señales de alto en una implementación y llamar la atención sobre problemas que podrían indicar que no es posible realizar más pruebas o que no valen la pena. Las pruebas de humo no son muy extensas, pero deben ser extremadamente rápidas. Si un cambio no pasa una prueba de humo, es una señal temprana de que las afirmaciones básicas se rompieron y que no debe dedicar más tiempo a las pruebas hasta que se resuelva el problema.

Las pruebas de humo específicas del contexto pueden emplearse al comienzo de cualquier nueva prueba de fase para afirmar que se cumplen los supuestos y requisitos básicos. Por ejemplo, las pruebas de humo pueden usarse tanto antes de las pruebas de integración como de implementación en servidores de etapas, pero las condiciones que se probarán variarán en cada caso.

### Pruebas unitarias.

Las pruebas unitarias son responsables de probar elementos individuales del código de una manera aislada y altamente específica. La funcionalidad de las funciones y clases individuales se prueban por sí mismas. Todas las dependencias externas se reemplazan por implementaciones secundarias o simuladas para centrar la prueba completamente en el código en cuestión.

Las pruebas unitarias son esenciales para probar la exactitud de los componentes del código individual para la coherencia interna y la corrección antes de que se coloquen en contextos más complejos. El alcance limitado de las pruebas y la eliminación de dependencias hace que sea más fácil buscar la causa de cualquier defecto. También es el mejor momento para probar una variedad de entradas y ramas de código que podrían ser difíciles de encontrar más adelante. A menudo, después de cualquier prueba de humo, las pruebas unitarias son las primeras pruebas que se realizan cuando se realizan cambios.

Las pruebas unitarias las ejecutan normalmente los desarrolladores individuales en su propia estación de trabajo antes de enviar los cambios. Sin embargo, los servidores de integración continua casi siempre vuelven a ejecutar estas pruebas como protección segura antes de

comenzar las pruebas de integración.

**Pruebas de Integración.**

Después de las pruebas unitarias, las pruebas de integración se realizan agrupando los componentes y probándolos como un conjunto. Mientras que las pruebas unitarias validan la funcionalidad del código de forma aislada, las pruebas de integración aseguran que los componentes cooperen cuando interactúan entre sí. Este tipo de prueba tiene la oportunidad de detectar una clase completamente diferente de errores que se exponen a través de la interacción entre los componentes.

Normalmente, las pruebas de integración se realizan automáticamente cuando el código se registra en un repositorio compartido. Un servidor de integración continua verifica el código, realiza los pasos de compilación necesarios (por lo general, realiza una prueba rápida de detección de humo para asegurarse de que la construcción fue exitosa) y luego ejecuta las pruebas de unidad y de integración. Los módulos se conectan en diferentes combinaciones y se prueban.

Las pruebas de integración son importantes para el trabajo compartido porque protegen la salud del proyecto. Los cambios deben demostrar que no rompen la funcionalidad existente y que interactúan con otros códigos como se esperaba. Un objetivo secundario de las pruebas de integración es verificar que los cambios se puedan implementar en un entorno limpio. Este es frecuentemente el primer ciclo de prueba que se realiza fuera de las máquinas del desarrollador, por lo que también se pueden descubrir dependencias de software y ambientales desconocidas durante este proceso. Por lo general, también es la primera vez que se prueba el nuevo código con bibliotecas, servicios y datos externos reales.

**Pruebas de sistema.**

Una vez que se realizan las pruebas de integración, puede comenzar otro nivel de prueba llamado prueba del sistema. En muchos sentidos, las pruebas del sistema actúan como una extensión de las pruebas de integración. El objetivo de las pruebas del sistema es asegurarse de que los grupos de componentes funcionen correctamente como un todo cohesivo.

En lugar de centrarse en las interfaces entre los componentes, las pruebas del sistema generalmente evalúan la funcionalidad externa de una pieza completa de software. Este conjunto de pruebas ignora las partes constitutivas para medir el software compuesto como una entidad unificada. Debido a esta distinción, las pruebas del sistema generalmente se enfocan en interfaces accesibles por el usuario -o de manera externa-.

**Prueba de aceptación.**

Las pruebas de aceptación son uno de los últimos tipos de pruebas que se realizan en el software

antes de la entrega. La prueba de aceptación se usa para determinar si una pieza de software satisface todos los requisitos desde la perspectiva del negocio o del usuario. Estas pruebas a veces se compilan contra la especificación original y, a menudo, prueban las interfaces para la funcionalidad esperada y para la usabilidad.

La prueba de aceptación a menudo es una fase más complicada que puede extenderse más allá del lanzamiento del software. Las pruebas de aceptación automatizada se pueden utilizar para garantizar que se cumplan los requisitos tecnológicos del diseño, pero la verificación manual también suele desempeñar un papel.

Con frecuencia, las pruebas de aceptación comienzan implementando la compilación en un entorno intermedio que refleja el sistema de producción. A partir de aquí, los conjuntos de pruebas automatizadas se pueden ejecutar y los usuarios internos pueden acceder al sistema para verificar si funciona de la manera que lo necesitan. Después de lanzar u ofrecer acceso beta a los clientes, se realizan más pruebas de aceptación evaluando cómo funciona el software con un uso real y recopilando comentarios de los usuarios.

## Terminología adicional.

Si bien hemos discutido algunas de las ideas más generales anteriores, hay muchos conceptos relacionados con los que puede encontrarse a medida que aprende sobre integración, entrega e implementación continuas. Vamos a definir algunos otros términos que probablemente deseas ver:

- **Implementaciones azul-verde**: es una estrategia para probar código en un entorno de producción e implementar código con un tiempo de inactividad mínimo. Se mantienen dos conjuntos de entornos con capacidad de producción y el código se implementa en el conjunto inactivo donde se pueden realizar las pruebas. Cuando esté listo para lanzar, el tráfico de producción se enrutará a los servidores con el nuevo código, haciendo que los cambios estén disponibles al instante.
- **Rama por Abstracción:** es un método para realizar operaciones de refactorización importantes en un proyecto activo sin ramas de desarrollo de larga duración en el repositorio de código fuente, en la cual las prácticas de integración continua se desalientan. Se construye y despliega una capa de abstracción entre los consumidores y la implementación existente para que la nueva implementación se pueda construir detrás de la abstracción en paralelo.
- **Compilación (como sustantivo)**: una compilación es una versión específica del software creado a partir del código fuente. Dependiendo del lenguaje de programación, este podría ser un código compilado o un conjunto consistente de código interpretado.
- **Versiones Canarias:** las Versiones Canarias son una estrategia para lanzar cambios a un subconjunto limitado de usuarios. La idea es asegurarse de que todo funcione correctamente con las cargas de trabajo de producción mientras se minimiza el impacto si hay problemas.

- **Lanzamiento Oculto**: es la práctica de implementar código en producción que recibe tráfico de producción *pero no afecta la experiencia del usuario*. Se implementan los nuevos cambios junto con las implementaciones existentes y el mismo tráfico a menudo se enruta a ambos lugares para realizar pruebas. La implementación anterior aún está conectada a la interfaz del usuario, pero detrás de escena, el nuevo código se puede evaluar para ver si es correcto usando las solicitudes reales de los usuarios en el entorno de producción.
- **Implementación de *cadena de procesos*:** es un conjunto de componentes que mueve el software a través de pruebas cada vez más rigurosas y escenarios de implementación para evaluar su preparación para el lanzamiento. La canalización generalmente finaliza desplegándose automáticamente a la producción o brindando la opción de hacerlo manualmente.
- **Característica Condicionada o Alterna:** son una técnica de despliegue de nuevas funciones detrás de la lógica condicional que determina si se ejecuta o no en función del valor de una variable de entorno. **El nuevo código se puede implementar en la producción sin que se active al establecer la bandera de forma adecuada.** Para liberar el software, se cambia el valor de la variable del entorno, lo que provoca que se active la nueva ruta del código. Los **Características Condicionadas o Alternas** a menudo contienen lógica que permite que subconjuntos de usuarios accedan a la nueva característica, creando un mecanismo para desplegar gradualmente el nuevo código.
- **Promoción:** en el contexto de procesos continuos, promover significa mover una compilación de software a la próxima etapa de prueba.
- **Pruebas de remojo:** las pruebas de remojo implican pruebas de software bajo producción significativa o carga similar a la producción durante un período de tiempo prolongado.

## Conclusión.

En esta guía, introdujimos la **Integración Continua**, la **Distribución Continua** y la **Implementación Continua**, y discutimos cómo se pueden usar para crear y lanzar software bien probado de forma segura y rápida. Estos procesos aprovechan una amplia automatización y fomentan el intercambio constante de códigos para corregir los defectos de manera temprana. Si bien las técnicas, los procesos y las herramientas necesarios para implementar estas soluciones representan un desafío significativo, los beneficios de un sistema bien diseñado y utilizado adecuadamente pueden ser enormes.

# Fin de la traducción.

---

## Versionado de software y Liberación Continua

La ingeniería de software es una rama de estudio muy compleja por lo que es necesaria su simplificación para explicarla en un solo artículo web. Esencialmente, como todo en computación,

se comienza desde cero y no desde uno, como dicta el sentido común. Por ello al llegar a la versión 1.0.0 es que en realidad deviene el nacimiento de un software, es estable y listo para el público.

Siguiendo con la simplificación, en la década de los años 1980, cuando la computación personal se masificó y llegó a nuestros hogares gracias a la aplicación "matadora" como lo fueron las hojas de cálculo (primero Visicalc y luego Lotus 1-2-3), la manera natural de distribuir las aplicaciones era por medio de discos flexibles (primero de 8" que confesamos no haberlos utilizado, luego los de 5¼" y comenzando los 90 los de tamaño 3½). Para aquella época teníamos muy presente el versionado porque rotulábamos y catalogábamos muy cuidadosamente nuestras herramientas y las manteníamos en buen estado: no había de otra, el internet era demasiado lento en aquella época y descargar 5 megabytes en aquel tiempo tomaba mucho, muchísimo tiempo (sin contar que eran pocos los sitios web en línea, las pa?inas web eran más bien informativas).

En lo años 1990 llegó a nuestras vidas los discos compactos ("*compact disc*" o CD por su denominación en idioma inglés) con sus "abultados" 700 megabytes y para los años 2000 los discos de vídeo digital *("digital video disc" o DVD)* para simplificarnos el trabajo de instalación de software. Hoy día, curiosamente, el formato Blue-ray con sus 50 gigabytes en doble capa pasa mayormente por desapercibida ¿por qué? Veamos.

## Numeración

La numeración o versionado de software contiene tres números separados por puntos y de izquierda a derecha el primero cambia cuando se han hecho modificaciones grandes e incluso la modificación de la estructura de datos o el soporte para nuevos motores de bases de datos. Generalmente los incrementos son de uno en uno pero se dan casos en que el período de prueba previo al lanzamiento de la versión se pueden hallar excepciones e incluso errores que no se pueden pasar por alto y entonces no se libera la versión, produciendo así el salto. Otras empresas pues simplemente gustan de los números pares solamente y hasta hay los cabalísticos que saltan el número trece. La psicología es una rama de la ciencia muy fértil en su estudio, nosotros los seres humanos hasta llegamos a ser muy caprichosos, desde nuestro punto de vista.

El segundo número, de manera similar al anterior en sus incrementos, se utiliza generalmente para incluir cambios menores y pueden incluir correciones. Algunos desarrolladores y desarrolladoras llegan hasta el tercer número para correcciones. En los tres casos los ceros a la izquierda son ignorados lo que ocasiona ciertos problemas de ordenado al llegar a la decena, así puede producir confusión el evaluar que la versión 12.1.10, a pesar de ser menor que la 12.10.7, nuestros ordenadores insistirán en que es al contrario. Veamos los UNICODE de cada caracter para ilustrar el ejemplo:

```
12.1.10: 49-50-46-49-46-49-48  12.10.7: 49-50-46-49-48-46-55
```

A esta altura tal vez piensen que este artículo es una tontería... pues va a ser que no pues estos valores de versiones **son importantes para determinar si el instalador de versión que vamos a implantar es superior al que tenemos instalado,** de lo contrario se negará a ser registrado en el ordenador o dispositivo. Es recomendable entonces siempre utilizar los puntos como separadores y utilizarlos como delimitadores de campo utilizando librerías que lo extraigan en vez de estar analizando el contar caracteres como espacios fijos... ¡pensad que hay numeraciones que llegan hasta las centenas!

Muchos entornos de programación ofrecen cambiar la numeración cada vez que compilamos, es decir, convertimos del código fuente a lenguaje de máquina -unos y ceros-; todo esto queda a gusto de cada equipo de desarrollo.

### Sentido práctico

Como vemos el sistema anterior tiene sus bemoles, así que mucha gente se decide por lo práctico y útil con respecto a la cuarta dimensión para el universo: el tiempo (en este caso medido en relación a acontecimientos en la historia de la humanidad). Por ello hay quienes numeran simplemente con la fecha en que liberan la versión, así si hoy es uno de febrero de dos mil dieciocho y compilamos pues la versión sería 2018.2.1, sin embargo esto tampoco releva el detalle del listado por pantalla de los ficheros en los repositorios y la comparación de versiones para tomar decisiones de instalación. Acá entonces tomamos el tema de los repositorios de software.

## Liberación continua de software

Con la masificación del internet, su globalización y su consecuente disminución en el coste de alojamiento web debido al abaratamiento del hardware con el paso de los años y el aumento de la oferta de empresas dedicadas a cosntruir granjas de servidores, se comenzó a utilizar los repositorios de software en línea más y más. No era que no existieran antes, sino que estaban limitados a las redes de área local y aislados del mundo exterior. Así, cuando estaba lista una versión, pues se compilaba y empaquetaba para guardarlos en diskettes o discos ópticos y su posterior distribución física con los consecuentes onconvenientes de costo en tiempo y materia prima (medios magnéticos u ópticos).

Quienes resolvieron esto de manera muy práctica fueron los programadores y programadoras de software libre para el kernel Linux ya que los mismos medios de distribución pueden ser utilizados como repositorios locales, siempre y cuando así se especifique al instalar un servidor como Debian, como caso.

Mejor aún es instalar, los programas adecuados y formalmente crear un repositorio local capaz de dar soporte a varias *distros* de GNU/Linux: así un solo ordenador estaría expuesto al internet solo descargaremos las actualizaciones de los sistemas operativos una sola vez y de allí distribuirlos en nuestra red de área local. Un punto a tomar en cuenta es el asunto de la seguridad, por ello las

empresas que dan soporte a estos sistemas operativos recomiendan instalar una clave mediante GPG , un herramienta de cifrado y firmas digitales hecho también en software libre y que garantiza que los datos que estamos bajando a nuestros repositorios privados y/o nuestros ordenadores son copia fiel y exacta de quienes mantienen la distribución del sistema operativo.

El inconveniente con este esquema es que hasta que los desarrolladores de la *distro* no hayan probado y aprobado los paquetes de software así que no tendremos la última versión estable. Por esta razón se publican repositorios completamente dedicados a un software en particular.