# Planear actividades de construcción del software de acuerdo con el diseño establecido.

Producción y compilación del contenido digital Inst. Angélica M. Triana
Solo fines académicos

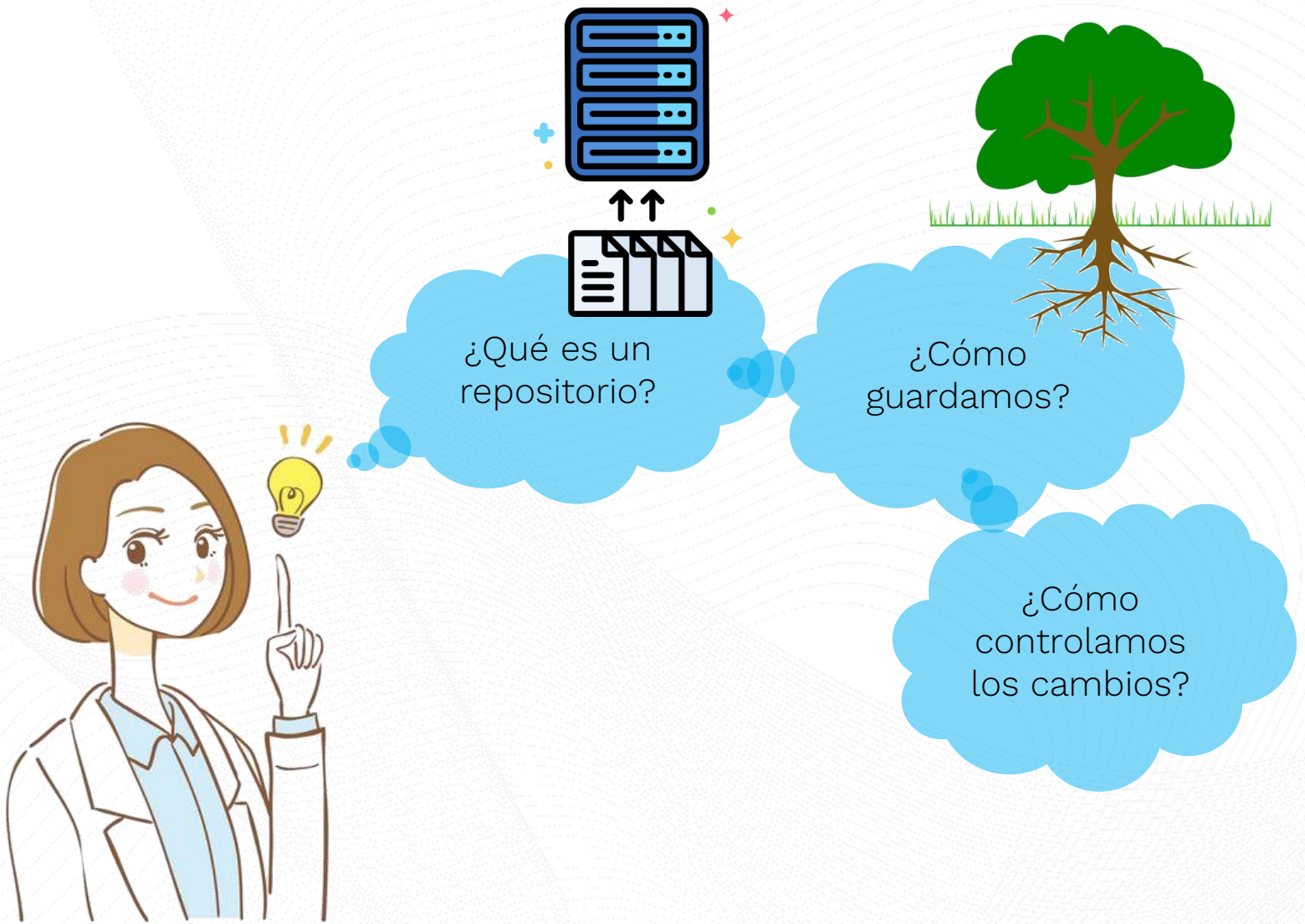
# Contenido

Herramientas de apoyo:
versionamiento, control de cambios
del código, integración continua)

Sistema de control de versiones

Actividades de la sesión

# Reflexión inicial



# ¿Qué es el control de versiones?

El control de versiones, también conocido como "control de código fuente", es la práctica de rastrear y gestionar los cambios en el código de software.

Los sistemas de control de versiones son herramientas de software que ayudan a los equipos de software a gestionar los cambios en el código fuente a lo largo del tiempo.

Ayudan a reducir el tiempo de desarrollo y a aumentar las implementaciones exitosas.

El software de control de versiones realiza un seguimiento de todas las modificaciones en el código en un tipo especial de base de datos. Si se comete un error, los desarrolladores pueden ir hacia atrás en el tiempo y comparar las versiones anteriores del código para ayudar a resolver el error, al tiempo que se minimizan las interrupciones para todos los miembros del equipo.

# El inicio...

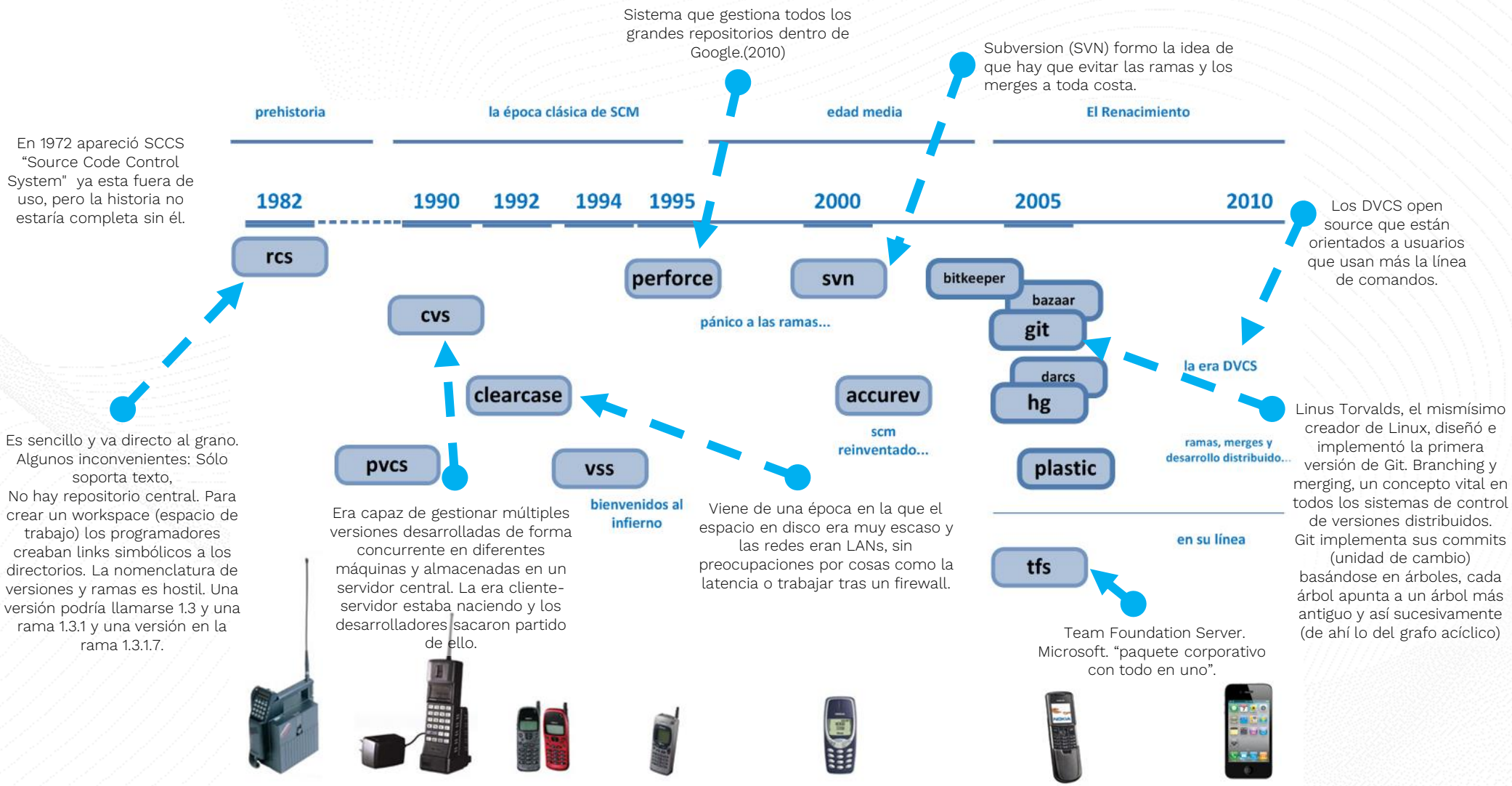
La Gestión de Configuración de Software (Software Configuration Management, SCM, o Source Code Management) ha ido evolucionando lentamente desde los lejanos días en los que era casi una labor manual hasta la actualidad de los sistemas DVCS (Distributed Version Control System).

**La prehistoria:** Hubo un tiempo en el que las versiones se guardaban a mano.

Para muchos de ese tiempo no eran los 80 sino hace mucho menos, cuando se guarda las prácticas con nombres como practica.zip, practica-version2.zip, practica-buena.zip, practica-buena-de-verdad.zip.

**Primera generación de los Sistemas de Control de Versiones:** Los primeros Sistemas de Control de Versiones funcionaban únicamente de forma local, estaban orientados a archivos y se basaban en bloqueos para evitar que un archivo fuera editado por 2 desarrolladores de forma simultánea. Estos sistemas están diseñados para su ejecución en plataformas UNIX.

# La historia de control de versiones



# Artículo: Versionado de Software desde el punto de vista del usuario

El versionado de software pretende establecer una comunicación entre la empresa y los usuarios. Como usuarios, lo que podemos obtener del modelo de versionado de software utilizado por una empresa en particular para un producto de software específico es información:

- Información sobre cómo enfrenta esta empresa el proceso de desarrollo de su producto de software.
- Información sobre cómo gestiona las mejoras, correcciones y cambios que experimenta el software, y finalmente
- Información sobre la frecuencia en que habrá de liberar los productos, llámense versiones o actualizaciones.

Una vez obtenida esta información, ¿qué podemos hacer con ella?

Podemos ejecutar, entre otras actividades como la planificación de respaldos de los sistemas, planificación de descarga y ejecución de actualizaciones, y la definición de los procesos asociados a la recuperación de un estado estable en el caso de un fallo.

# El problema del versionado

Lamentablemente, no es infrecuente que algunos usuarios de un software dado ignoren cómo interpretar la secuencia de números y letras que solemos llamar versión y a partir de allí pierden toda la información asociada.

En realidad puede ser difícil comprender un esquema de versionado de software, ya que no existe un modelo universal que rija para todas las empresas. Por el contrario lo que sí que existe son múltiples esquemas de versionado con diferentes preceptos y características.

Además, también existe el agravante de que las empresas de software se sienten en la libertad de adaptar los modelos de versionado a su propia filosofía, introduciendo conceptos como edición, fecha de compilación, fase de desarrollo, parches, etc., generándose más confusión.

Una de las confusiones más recurrentes es aquella que se centra en la **diferencia entre versión, release y actualización**:

# Diferencias entre versión, release y actualización

Cuando una empresa de software introduce cambios o mejoras sobre su producto está generando diferentes versiones del software. Muchas de estas versiones no llegan a ser lanzadas al mercado, sino que quedan como parte del trabajo interno de la empresa.

Cuando la empresa produce una versión que desea liberar se genera entonces una versión release que suele incluir todos los programas involucrados, además de la documentación y materiales de soporte apropiados.

Por lo tanto, todos los releases son versiones, pero no todas las versiones se convierten en releases.

Razones para generar un nuevo reléase (actualización):

- Correcciones de errores en el código.
- Optimizaciones del código.
- Mejoras en los niveles de seguridad.
- Introducción de nuevas funciones pequeñas, en términos de agregación y no de sustitución.

Release o actualizaciones a cambios menores o mejoras dentro de una misma versión.

La mayoría de los productores de software basados en su modelo de versionado de software llaman versión a aquellos productos que surgen de cambios radicales en el sistema.

Consenso que justifica una nueva versión:

- Cambios significativos en la estructura de la base de datos.
- Cambios en la plataforma como cambios de lenguajes e incluso un cambio en la idea general que sustenta el producto.
- Introducir nuevas funciones significativas del sistema.
- Cambios en la estructura de comunicaciones o en los protocolos utilizados.

# ¿Cómo se deciden las versiones del software? (parte 1)

**Versiones por número:** Algo común es realizar el manejo de versiones mediante 3 números: X.Y.Z y cada uno indica una cosa diferente.

- El primero (X) se le conoce como versión mayor y nos indica la versión principal del software. Ejemplo: 1.0.0, 3.0.0
- El segundo (Y) se le conoce como versión menor y nos indica nuevas funcionalidades. Ejemplo: 1.2.0, 3.3.0
- El tercero (Z) se le conoce como revisión y nos indica que se hizo una revisión del código por algún fallo. Ejemplo: 1.2.2, 3.3.4

Una pregunta importante: ¿cómo sabemos cuando cambiarlos y cuál cambiar?

- Versión mayor (major) o X, cuando agreguemos nuevas funcionalidades importantes, puede ser como un nuevo modulo o característica clave para la funcionalidad.
- Versión menor (minor)  o Y, cuando hacemos correcciones menores, cuando arreglamos un error y se agregan funcionalidades que no son cruciales para el proyecto.
- Revisión (patch) o Z, cada vez que se corrigen fallas, estética, entregamos el proyecto.

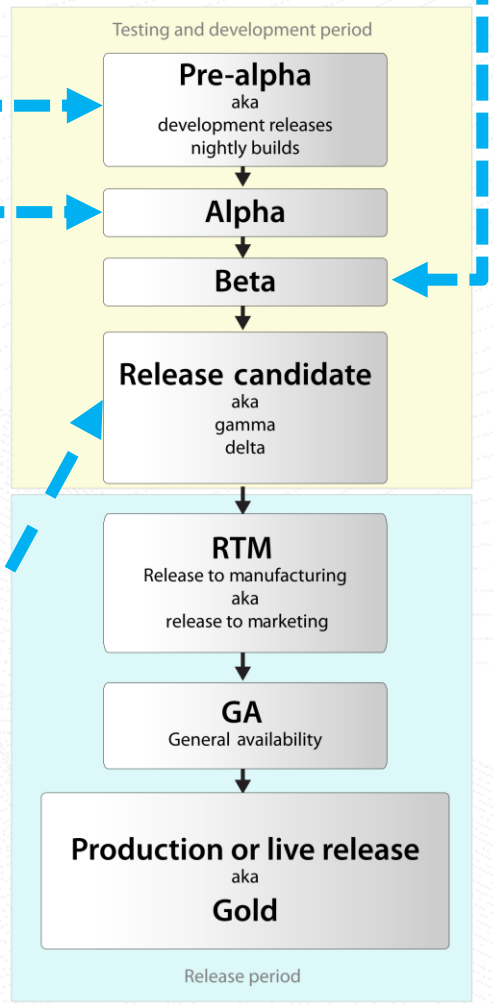
# ¿Cómo se deciden las versiones del software? (parte 2)

## Versiones por estabilidad:

Pre-alfa se refiere a todas las actividades realizadas durante el proyecto de software antes de las pruebas formales.

Alpha es una versión inestable que es muy probable que tenga muchas opciones que mejorar, pero queremos que sea probada para encontrar errores y poder poner a prueba funcionalidades, en la mayoría de los casos podemos decir que esta casi listo el producto. Ejemplo: 1.0Alpha, 1.0a1,1.0a2.

RC (Release Candidate), que es el último toque fino del software antes de salir y después de pasar por Beta. Ejemplo: 3.0-RC o también 3.0-RC1.




Beta una versión mas estable que Alpha en la que contamos con el producto en su totalidad, y se desea realizar pruebas de rendimiento, usabilidad y funcionamiento de algunos módulos para ver cómo funciona bajo un ambiente no tan controlado. Aquí aparece el nombre de Beta Tester que escuchamos en el mundo del software. Ejemplo: 2.0Beta, 2.0b, 2.0b1

La versión de disponibilidad general (también llamada dorada) de un producto es su versión final. Normalmente es casi idéntica a la versión candidata final, con sólo correcciones de última hora. Esta versión es considerada muy estable y relativamente libre de errores con una calidad adecuada para una distribución amplia y usada por usuarios finales. Microsoft y otros usan el término distribución a fabricantes (RTM o release to manufacturing) para referirse a esta versión.

# ¿Cómo se deciden las versiones del software? (parte 3)

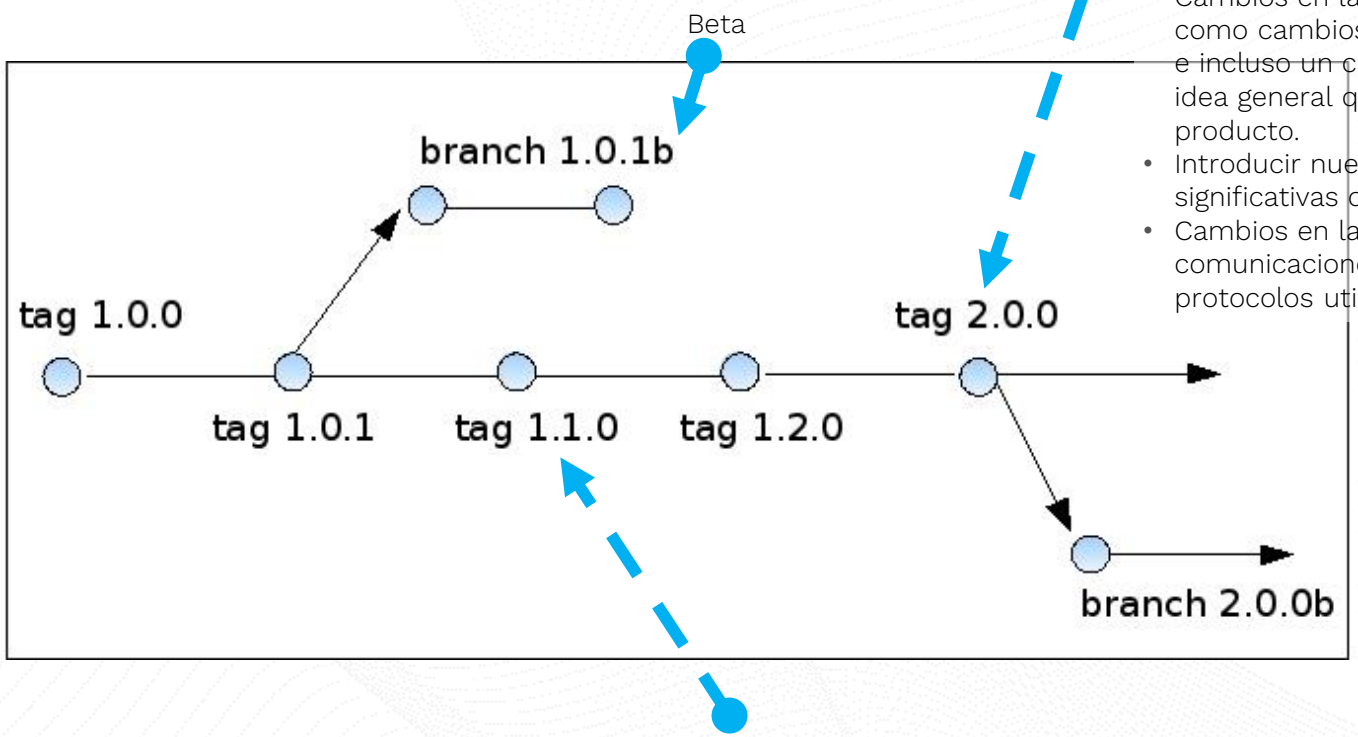
Existe software que necesita considerar los casos de los parches y las fechas de lanzamientos.

**Versión de parche:** En el caso de los parches podemos agregar un dígito para señalar el parche, ya teníamos algo así: X.Y.Z y ahora tendríamos algo así: X.Y.Z.P así que P sería el número del parche. Ejemplo: 1.2.5.2, 02.03.03.01

**Versión por fecha:** Este tiene muchas variaciones, se puede tener diferente orden del año, mes y día. Ejemplo: 1.2.3.1543 donde 15 es el año 2015, 4 es el mes y 3 el día, se podrían tener diferentes acomodos y formatos: 1.2.3.4315 o 1.2.3.201543, 1.2.3.1534

Conclusión: No existe una regla o estándar oficial para normar las versiones de software, existen diferentes propuestas que podemos aplicar.

# Ejemplo de las versiones

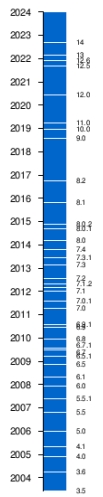



- Cambios significativos en la estructura de la base de datos.
- Cambios en la plataforma como cambios de lenguajes e incluso un cambio en la idea general que sustenta el producto.
- Introducir nuevas funciones significativas del sistema.
- Cambios en la estructura de comunicaciones o en los protocolos utilizados.

(1) Es la versión principal del sw.
(1) Nuevas funcionalidades
(0) Revisión código (fallo)

- Android 4.0 Ice Cream Sandwich
- Android 4.1 Jelly Bean
- Android 4.4 KitKat
- Android 5.0 Lollipop
- Android 6.0 Marshmallow
- Android 7.0 Nougat
- Android 8.0 Oreo
- Android 9 Pie
- Android 10 Quince Tart (nombre interno)
- Android 11 Red Velvet Cake (nombre interno)
- Android 12 Snow Cone (nombre interno)
- Android 13 Tiramisu (nombre interno)

Versiones [ editar ]



Versiones NetBeans[3]

| Versión | Fecha de lanzamiento |
|---|---|
| Apache Netbeans 14 | 9 de junio de 2022 |
| Apache Netbeans 13 | 4 de marzo de 2022 |
| Apache Netbeans 12.6 | 29 de noviembre de 2021 |
| Apache Netbeans 12.5 | 13 de septiembre de 2021 |
| Apache Netbeans 12.3 | 3 de marzo de 2021 |
| Apache Netbeans 12.2 | 7 de diciembre de 2020 |
| Apache Netbeans 12.0 | 4 de junio de 2020 |
| Apache Netbeans 11.3 | 2019 |
| Apache Netbeans 11.2 | 2019 |
| Apache Netbeans 11.1 | 2019 |
| Apache Netbeans 11.0 | 4 de abril de 2019 |
| Apache Netbeans 10.0 | 27 de diciembre de 2018 |
| Apache Netbeans 9.0 | 29 de julio de 2018 |
| NetBeans 8.2 | 3 de octubre de 2016 |
| NetBeans 8.1 | 4 de noviembre de 2015 |
| NetBeans 8.0.1 | 5 de octubre de 2014 |
| NetBeans 7.4 | 15 de octubre de 2013 |

# Actividad de la sesión

# Actividad de la sesión

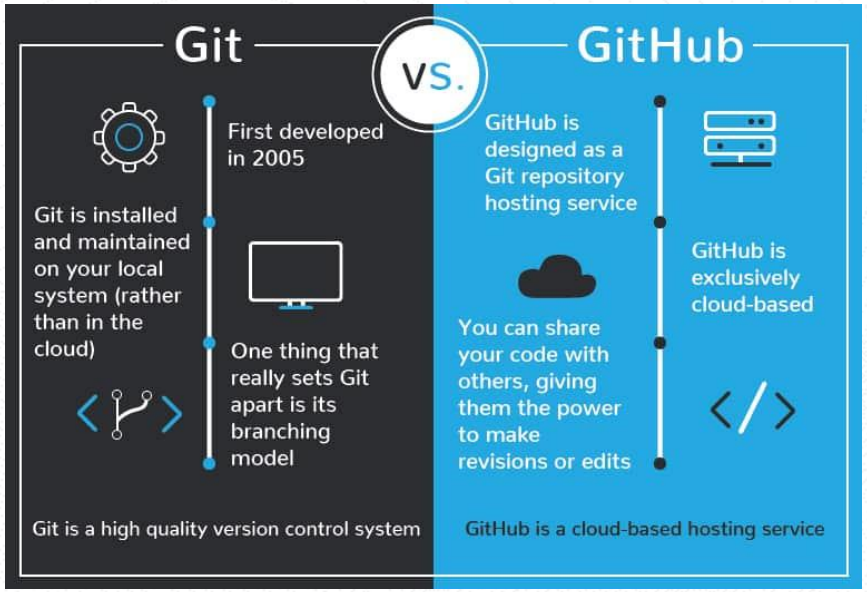
De manera individual,

- Realice la siguiente lectura: https://medium.com/@jointdeveloper/sistemas-de-control-de-versiones-qu%C3%A9-son-y-por-qu%C3%A9-amarlos-24b6957e716e
- Consulte el siguiente vídeo: https://es.coursera.org/lecture/git-espanol/conceptos-basicos-control-de-versiones-CRlQH?utm_source=link&utm_medium=page_share&utm_content=vlp&utm_campaign=top_button

Posteriormente, conteste la encuesta sobre los temas tratados en los enlaces sugeridos en :

pollev.com/angelicamari519

# Reflexión inicial

# Sistema de control de versiones - Git

# Sistema de control de versiones - Git

Git es un sistema de control de versiones que nació en el año 2005 producto de las necesidades que afrontaba la comunidad del kernel de Linux de encontrar un mecanismo que les permitiera soportar las actualizaciones y mantenimientos que requería este gran proyecto.

Según Git (2021), algunos de los objetivos trazados que debía cumplir este nuevo mecanismo incluían: velocidad, diseño sencillo, soporte para múltiples desarrollos no lineales, debía ser un sistema totalmente distribuido y debía ser capaz de manejar eficientemente proyectos de gran envergadura.

Git, a diferencia de otros sistemas de control de versiones ofrece la posibilidad de trabajo aun cuando no se tiene conectividad al servidor central, ya que localmente se posee toda la información requerida incluso si se desea hacer procesos de recuperación a versiones anteriores.

# Sistema de control de versiones - Funcionamiento

Git hace un proceso de comprobación antes de realizar el almacenamiento de datos mediante unos procesos algorítmicos con códigos criptográficos SHA-1, que son calculados a partir del contenido y estructura de directorio del proyecto, lo cual hace imposible cambiar el contenido de algún archivo o de la estructura de directorios sin que Git lo detecte.

Debido a que Git solo añade información a la base de datos con cualquier acción realizada, es difícil llegar a un estado en el que este sistema control de versiones no pueda hacer un proceso de recuperación o que se elimine información de alguna manera una vez exista una confirmación en los cambios.

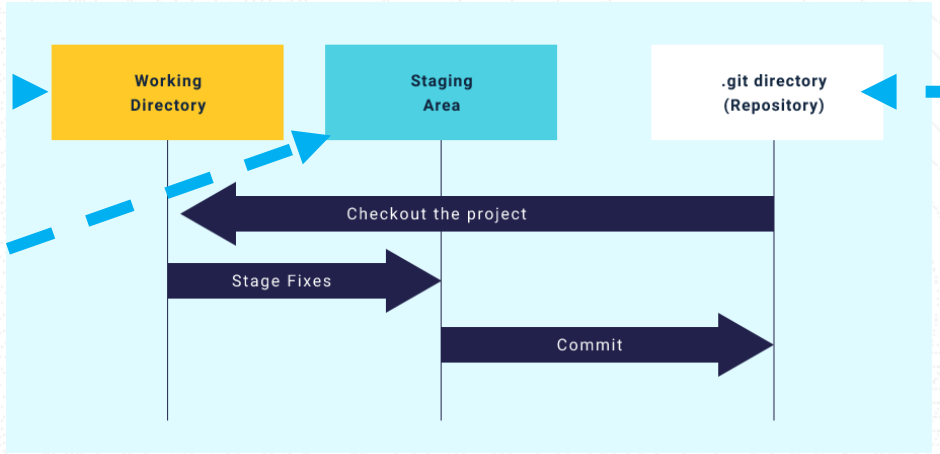
Según Git (2021), el sistema de control de versiones **Git maneja tres estados principales para cada uno de los archivos de un proyecto, los cuales son**:

# Sistema de control de versiones - Mecánica

- **Confirmado (committed):** Indica que los datos se encuentran almacenados de forma segura en la base de datos local. Se considera una versión concreta.
- **Preparado (staged):** Indica un archivo modificado que ha sido marcado en su versión actual para una próxima confirmación.
- **Modificado (modified):** Indica un que el archivo ha sufrido alguna modificación, pero esta aún no se ha preparado ni confirmado en la base de datos.

El directorio de trabajo que es donde se desarrolla las labores de construcción del proyecto y corresponde a una copia de la versión del proyecto.

El área de preparación es un archivo que almacena información de lo que va en la próxima confirmación.




El directorio Git es donde se almacenan los metadatos y base de datos de objetos para el proyecto. Este directorio Git es lo que se copia en el proceso de clonación del repositorio desde otro equipo de trabajo.

# Sistema de control de versiones - Flujo de trabajo

Un flujo normal de acciones para trabajar con Git sería el siguiente:

1) Realizar acciones de construcción del proyecto sobre el directorio de trabajo.
2) Realizar la preparación de los archivos, los cuales se marcaron para ser añadidos al área de preparación.
3) Confirmar los cambios, lo cual toma los archivos del área de preparación y los almacena como una copia instantánea permanente en el directorio Git.

Git se puede instalar en cualquier tipo y distribución de sistema operativo e incluso se puede integrar con diferentes IDE para conexión con varios repositorios locales y en la nube.

Instalación de git: https://git-scm.com/book/es/v2/Inicio---Sobre-el-Control-de-Versiones-Instalaci%C3%B3n-de-Git

# Plataformas para implementar integración continua

En la actualidad existe gran cantidad de plataformas con servicios en la nube en los que se pueden implementar procesos de integración continua, los cuales, generalmente, ofrecen la posibilidad de crear repositorios públicos y privados, se integran con sistemas de control de versiones, permiten el registro de grupos de trabajo y otras posibilidades más. Entre los más utilizados en la industria que se basan en sistemas de control de versiones en Git, se encuentran los descritos a continuación:

- GitHub
- GitLab
- BitBucket

Se podrá conocer sobre las diferentes versiones gratuitas con sus funcionalidades básicas, las cuales son accesibles simplemente con la creación de una cuenta de usuario.

# Actividad de la sesión

# Actividad de la sesión

De manera individual,

Compare las siguientes herramientas:

- GitHub: Conoce GitHub un portal creado para alojar códigos de las aplicaciones. Página oficial: https://Github.com
- GitLab: Conoce sobre GitLab, un servicio web que permite controlar las versiones y desarrollos de software colaborativo basado en Git. Página oficial: https://about.Gitlab.com
- BitBucket: Conoce a Bitbucket un servicio de alojamiento basado en web para los proyectos que utilizan sistema de control de versiones Mercurial y Git. Página oficial: https://bitbucket.org

Al finalizar, se socializará el tema en la sesión.

# G R A C I A S

Línea de atención al ciudadano: 01 8000 910270
Línea de atención al empresario: 01 8000 910682

@SENAComunica

www.sena.edu.co

# Referencias


Cantoral, C. (2017). ¿Que es una guía de código? Recuperado de: https://codigofacilito.com/articulos/guia_codigo

Bose, S. (2021). Coding Standards and Best Practices To Follow. Recuperado de https://www.browserstack.com/guide/coding-standards-best-practices#:~:text=Practices%20To%20Follow-,What%20are%20Coding%20Standards%3F,sophisticated%20and%20highly%20functional%20code.

Kumar, S. (2019). Software coding and testing. Recuperado de https://www.slideshare.net/SandeepKumarNayak1/software-coding-and-testing