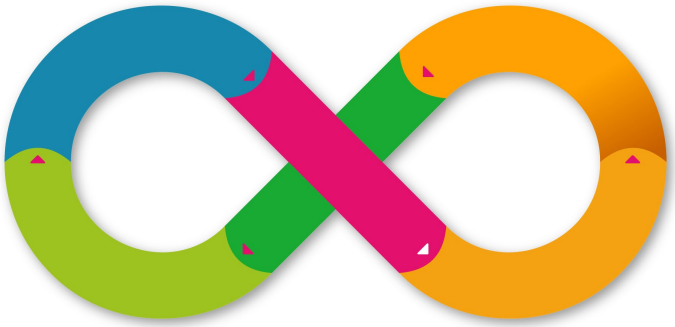


Preparación de Ambiente de Trabajo

Control de Versiones con Git

Sección 2
Módulo 2

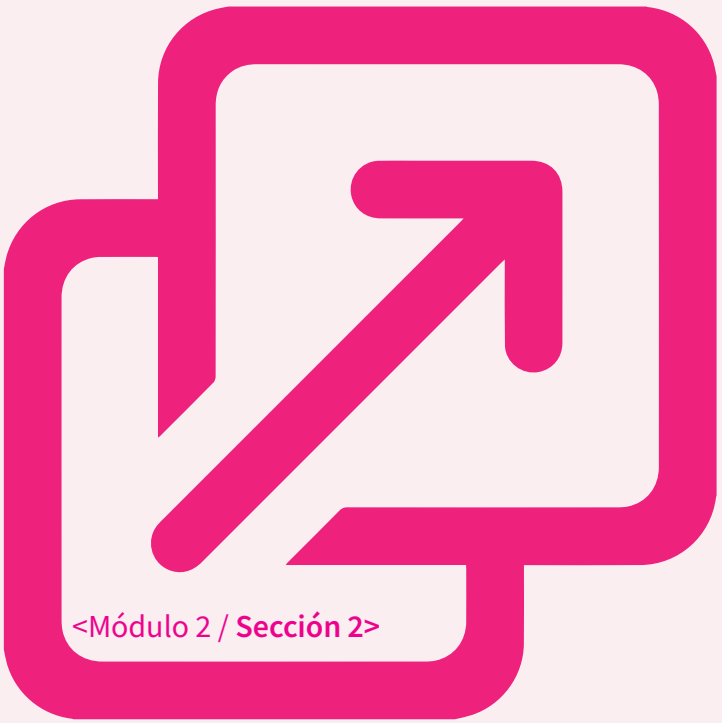
KIBERNUM
IT Academy

Control de versiones en DevOps?

El control de versiones es una práctica esencial en DevOps que permite a los equipos de desarrollo y operaciones rastrear y gestionar cambios en el código fuente, documentación y otros elementos de un proyecto de software. Al utilizar un sistema de control de versiones, es posible revertir fácilmente los cambios, ver quién ha realizado cambios específicos y cuándo se realizaron, y trabajar de manera más colaborativa en el proyecto.

Los sistemas de control de versiones se pueden clasificar en dos categorías: **centralizados y distribuidos.**

Un sistema de control de versiones centralizado (VCS) tiene un repositorio centralizado al que todos los usuarios deben enviar sus cambios. Este enfoque es similar al de una red de árbol, donde todos los cambios deben pasar por un nodo central. Los sistemas de control de versiones centralizados son fáciles de configurar y utilizar, y son adecuados para proyectos pequeños o equipos que no requieren una gran cantidad de colaboración.



A menudo, aproximarse a DevOps puede parecer intimidante, ya que existe una abrumadora cantidad de herramientas y técnicas de toda índole que se utilizan en este proceso. Sin embargo, no todas estas son necesarias; lo importante es saber escoger, y una vez se comprende cómo estas se utilizan en el **ciclo DevOps ,** el desarrollador podrá seleccionar las que se adecuan a las necesidades de su equipo y organización.

Algunos ejemplos de herramientas centralizadas son:

- CSV (current versión system)
- Subversion (SVN)
- Perforce Helix

Un sistema de control de versiones distribuido (DVCS), por otro lado, no depende de un repositorio centralizado. En su lugar, cada usuario tiene una copia completa del repositorio y puede realizar commits y fusiones de manera local. Este enfoque es más adecuado para proyectos más grandes o equipos que necesitan una mayor flexibilidad y colaboración. Los sistemas de control de versiones distribuidos son un poco más complejos de configurar y utilizar, pero ofrecen un mayor grado de libertad y flexibilidad.

Algunos ejemplos de herramientas distribuidas son:

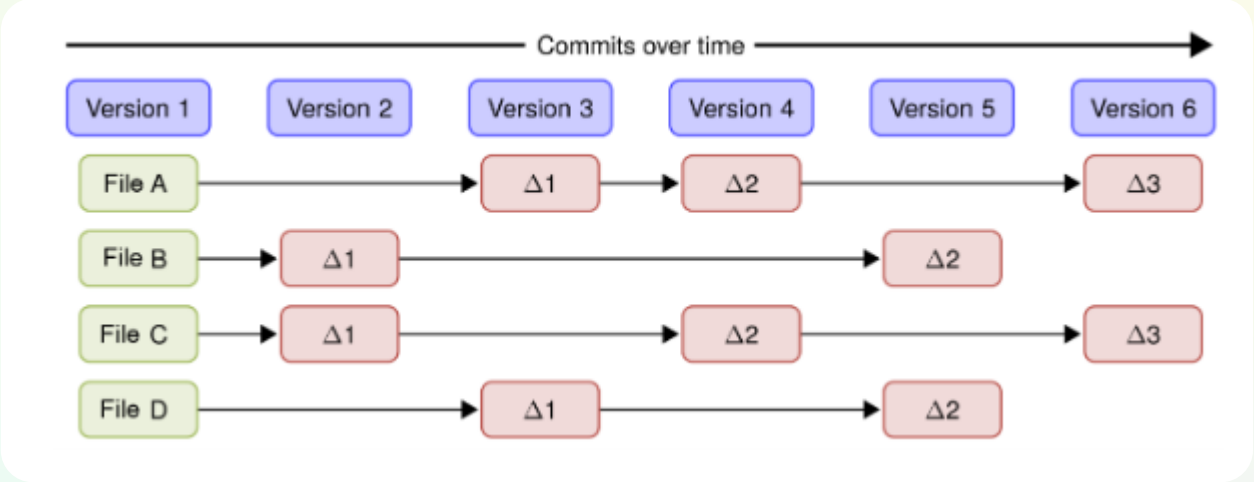
- Bazaar
- Mercurial
- Git

Herramientas de **apoyo**

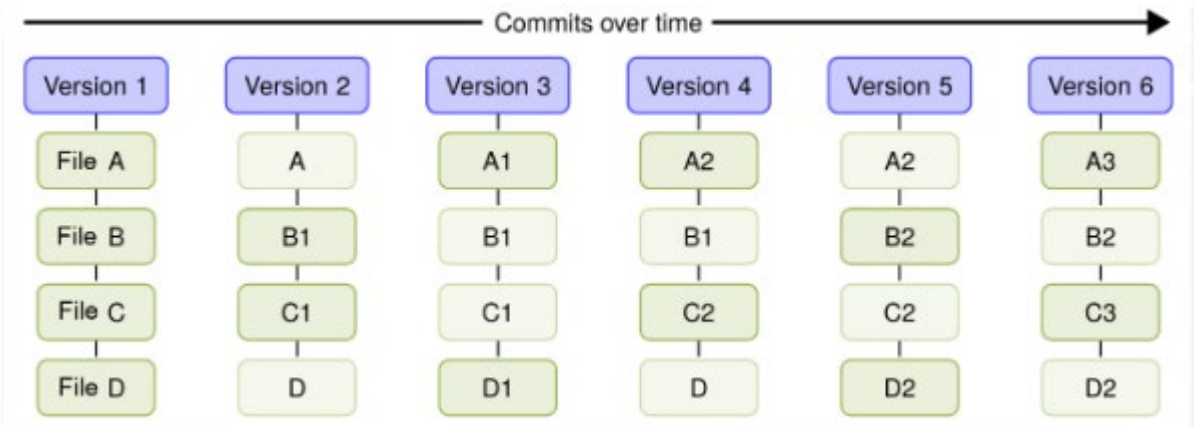
para la fase de Operaciones (opérate)

Primero entenderemos como Git referencia los objetos que registra y cómo la historia de estos se registra para que ud. Pueda tener una buena comprensión de lo que sucede a bajo nivel al trabajar con Git.

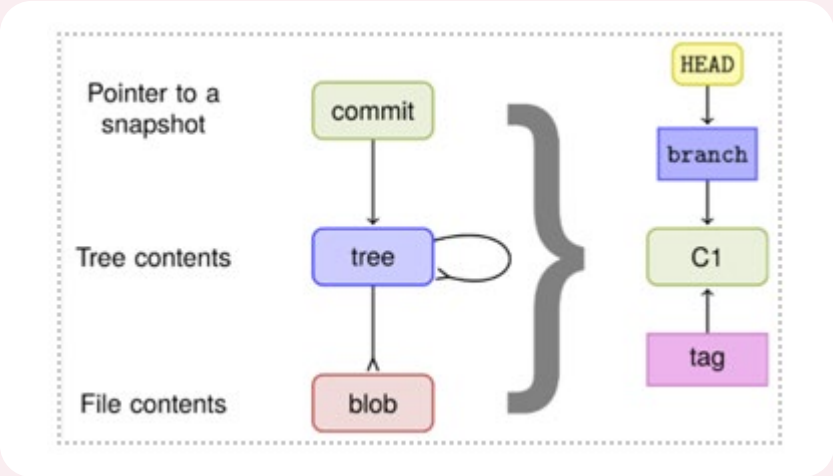
El modelo de datos de Git es diferente de otros VCSs en cómo maneja la data. Tradicionalmente, un VCS almacena la data en un archivo inicial, seguido de una serie de parches para cada nueva versión del archivo.



Git es diferente: registra una captura de todos los archivos monitoreados y sus rutas relativas a la raíz del repositorio. Cada commit registra completamente el estado completo del árbol. Si el archivo no se ha modificado, Git no lo almacena de nuevo, y en su lugar almacena un enlace al archivo ya registrado.



Esto hace a Git diferente de otros VCSs y más adelante explicaremos más acerca de las ventajas de este modelo. Podemos resumir el modelo de datos con el siguiente diagrama.



Existen **4** objetos

principales en este modelo:

Commit: Este objeto apunta a la raíz del árbol, que a su vez apunta a los subárboles y archivos. Se considera una instantánea del repositorio en un momento concreto.

Blob: son los archivos que componen el repositorio.

Tags: Las etiquetas son marcadores que se pueden utilizar para marcar un punto específico en la historia del repositorio. Las etiquetas se pueden utilizar para marcar versiones importantes del software, como lanzamientos o puntos de partida para ramas de desarrollo.

Tree: Un árbol (tree) es un objeto que representa un directorio de archivos. Cada árbol tiene un conjunto de entradas.

Cada objeto en Git es identificado a través de un código Hash generado a través de la función SHA-1 (Secure Hash Algorithm 1) y tiene 40 caracteres (representado en 160 bits), pero al referenciar un objeto basta usar los 7 primeros caracteres de este código, en lugar del código completo.

Además, existen dos tipos de punteros importantes:

Un branch (rama) es un puntero móvil que siempre apunta al último commit realizado en la rama. Cuando se realiza un commit, el branch se mueve automáticamente para apuntar al nuevo commit. Los branches se utilizan a menudo para representar diferentes versiones o ramas de desarrollo de un proyecto. Por ejemplo, se podría tener una rama “master” que represente la versión estable del proyecto y una rama “develop” que represente la versión en desarrollo.

HEAD es un puntero especial que siempre apunta al commit al que se está actualmente “posicionado”. Cuando se realiza un commit, HEAD se actualiza automáticamente para apuntar al nuevo commit. HEAD también se puede utilizar para cambiar entre ramas.

Iniciar en **Git**

Para empezar a utilizar Git, primero necesitarás instalarlo en tu máquina. Es importante verificar que no se encuentre instalado previamente, ya que algunos sistemas operativos lo incluyen en sus distribuciones oficiales (como es el caso de MacOs y algunas distribuciones de Linux).

Para verificar si Git está instalado en su máquina, utilice el siguiente comando en la terminal de comandos:

```
git --version
```

Para este curso se requiere ut ilizar la versión 2.0.0 o superior de Git. Si no tiene Git instalado, puede descargar la última versión desde el siguiente enlace y seguir las instrucciones de instalación para su sistema operativo:

https://git-scm.com/downloads

Una vez que hayas instalado Git, necesitarás configurar tu nombre de usuario y dirección de correo electrónico. Esta información se utilizará para identificarte como autor de los commits que realices:

```
git config --global user.name “Tu nombre”
git config --global user.email “tu@correo.com”
```

El comando git config se utiliza para establecer configuraciones de Git en tu máquina. Con este comando, puedes configurar opciones globales que se aplicarán a todos sus proyectos Git, o bien, opciones específicas de un proyecto en particular.

Con la opción --global, estamos estableciendo estas configuraciones de forma global. Si quiere establecer configuraciones específicas para un proyecto en particular, puede omitir la opción –global, pero en este caso, para no repetir las configuraciones, es mejor hacerlo de manera local.

Para trabajar de forma efectiva en Git, es muy importante poder navegar en los directorios desde la consola de navegación. Los comandos de navegación por consola pueden diferir dependiendo del sistema operativo, si usted no sabe cómo recorrer directorios y acceder a archivos en su sistema, por favor consultar el material complementario del curso, ya que esto es fundamental.

Crea un cuenta **GitHub**

Para el desarrollo de este curso, si no la tiene, deberá crear una cuenta en GitHub.com, el repositorio online que se utilizará para las practicas del curso.

GitHub es una plataforma de alojamiento de proyectos y colaboración en línea basada en Git. Es utilizada ampliamente por desarrolladores de software para almacenar y controlar versiones de proyectos, y también es una gran herramienta para colaborar con otros desarrolladores.

Con GitHub, puedes crear repositorios para almacenar tus proyectos y trabajar en ellos de forma colaborativa con otros desarrolladores. También puedes utilizar GitHub para realizar seguimiento de problemas y discutir cambios con tu equipo de desarrollo. Además, GitHub proporciona una gran cantidad de recursos para aprender y colaborar con otros desarrolladores, como guías y tutoriales en línea, comunidades y eventos.

Una vez hayan creado su cuenta en GitHub, se recomienda configurar una conexión SSH al equipo desde el cual están trabajando para facilitar el acceso y flujo de trabajo. Esto a veces puede ser imposible por restricciones de VPN establecidas por instituciones como son los bancos, u otras, que tienen altos estándares de seguridad ya que manejar información sensible.

Configurar túnel **SSH en GitHub**

El término “túnel SSH” puede sonar intimidante, pero un túnel SSH es una conexión segura y encriptada que se establece a través de una red pública, como Internet, utilizando el protocolo SSH (Secure Shell). Un túnel SSH permite a los usuarios conectarse a un servidor remoto y acceder a sus recursos de forma segura, sin tener que preocuparse por la seguridad de la red pública.

En primer lugar, se requiere generar un par de claves SSH utilizando el agente SSH de su sistema operativo. Se trata de dos claves, una pública y otra privada, y pueden utilizarse varios algoritmos para generarlas. Si no está familiarizado con el concepto de pares de llaves pública-privada en criptografía, puede consultar el material complementario de este documento.

Para generar su clave SSH siga los pasos en este artículo facilitado por GitHub, donde encontrará las instrucciones para realizar este proceso de acuerdo a su sistema operativo:

https://docs.github.com/en/free-pro-team@latest/github/authenticating-to-github/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent

Una vez haya generado su par de claves, preste atención a la ruta en que estas se guardan en su computador, diríjase a la carpeta, y vera dos archivos; uno sin extensión y uno con extensión .pub. La extensión .pub deriva de la palabra “public” esta es la llave pública que puede ser compartida como método de autenticación con distintos servicios web, tales como GitHub y algunas APIs. A veces, este tipo de archivos se abre automáticamente con Microsoft Publisher, debido que estos archivos tienen la misma extensión, pero para leer el contenido de la llave pública, debe ser abierto con un editor de texto plano (como el bloc de notas o cualquier IDE).

Abra este archivo, en el encontrará algo parecido a lo siguiente:

```
ssh-rsa
AAAB3NzaC1yc2EAAAABIwAAAQEAklOUpkDHrfHY17SbrmTIpNLTGK9T-
jom/BWDSUGPl+nafzlHDTYW7hdI4yZ5ew18JH4JW9jbhUFrviQzM7xlELEV-
f4h9lFX5QVkbPppSwg0cda3Pbv7kOdJ/MTyBlWXFCR+HAo3FXRitBqxiX1nKhX-
pHAZsMciLq8V6RjsNAQwdsdMFvSlVK/7XAt3FaoJoAsncM1Q9x5+3V0Ww68/
eIFmb1zuUFljQJKprrX88XypNDvjYNby6vw/Pb0rwert/EnmZ+AW4OZPnTPI89ZPmVM-
LuayrD2cE86Z/il8b+gw3r3+1nKatmIkjn2so1d01QraTlMqVSsbxNrRFi9wrf+M7Q==
schacon@mylaptop.local
```

Al inicio del archivo se indica el algoritmo utilizado para generar la clave (en este caso rsa, pero puede diferir si usted utilizó otra opción). Copie este texto al portapapeles.

Una vez generada la clave (asegúrese de añadirla al agente ssh), debe agregar está a la configuración de GitHub siguiendo los siguientes pasos. Puede encontrar las instrucciones de este proceso en el siguiente enlace:

https://docs.github.com/en/authentication/connecting-to-github-with-ssh/adding-a-new-ssh-key-to-your-github-account

Cuando deba configurar su clave SSH en el gestor SSH en GitHub, en el panel mostrado en la imagen, tenga en cuenta lo siguiente:

- El titulo debe identificar el equipo en el que usted esta trabajando, ya que se debe configurar solo una clave SSH por dispositivo. Ej: pc personal, pc trabajo, etc…
- Debe asegurarse de que el tipo de clave sea: Authentication Key
- Copie todo el contenido del archivo de la clave pública .pub en el campo “Key”
- Una vez añadida la clave no puede modificarse, si por alguna razón comete un error, elimínela y cree una nueva.

Finalmente, debe testear que su conexión haya quedado establecida correctamente entre el dispositivo y GitHub, y esto puede hacerse ejecutando el siguiente comando:

```
git --version
```

No debe modificar el correo, debe usarse el indicado. Si la conexión se estableció correctamente, verá un mensaje de éxito que dirá que se ha establecido la conexión, pero GitHub no provee Shell Access.

Con todo lo anterior, usted se encuentra preparado para comenzar a estudiar los comandos de Git y trabajar con GitHub.

Si ha tenido problemas en alguno de estos pasos serán resueltas en el tiempo de actividades en clase con su profesor.

Lecturas complementarias

¿Qué es el Hashing?
Actualizado el 22 de Julio de 2022
https://keepcoding.io/blog/que-es-el-hashing/

Pro Git book 2nd Edition: 10.2 Git Internals - Git Objects
https://git-scm.com/book/en/v2/Git-Internals-Git-Objects

¿Qué es la criptografía asimétrica y cómo funciona?
Actualizado el 12 de mayo de 2020
https://protecciondatos-lopd.com/empresas/criptografia-asimetrica/

Uso de la consola de comandos:

Windows:
Guía de la consola de Windows: https://docs.microsoft.com/es-es/windows-server/administration/windows-commands/windows-commands

Tutorial básico de la consola de Windows: https://www.digitalcitizen.life/command-prompt-how-use-basic-commands

macOS:
Guía de la Terminal de macOS: https://support.apple.com/es-es/guide/terminal/welcome/mac

Tutorial básico de la Terminal de macOS: https://www.macworld.co.uk/feature/mac-software/how-use-terminal-on-mac-3608274/

Linux:
Guía de la terminal de Linux: https://www.linux.com/training-tutorials/first-5-things-do-after-installing-linux/

Tutorial básico de la terminal de Linux: https://www.howtogeek.com/140679/beginner-geek-how-to-start-using-the-linux-terminal/