Reporte de práctica de laboratorio

Práctica 10: Control de Versiones (CodeCommit)

Fecha 05/04/2019

Maestría en Sistemas Computacionales

*Aplicaciones y Servicios en la Nube*

Prof. Mtro. Rodolfo Luthe Ríos

Adrián Puente Castillo

[ms723528@iteso.mx](mailto:ms723528@iteso.mx)

# Introducción

En esta práctica se presenta el servicio de CodeCommit de AWS. CodeCommit es un servicio de manejo de versiones administrado, seguro, escalable que aloja repositorios privados de Git. CodeCommit elimina la necesidad de administración adicional de control de versiones de manera local o preocuparnos de escalar nuestra infraestructura. Es posible usar CodeCommit para almacenar cualquier tipo de archivo, desde código fuente hasta archivos binarios. Soporta la funcionalidad estándar de Git, de tal forma que se integra perfectamente con nuestras herramientas basadas en Git existentes.

Con CodeCommit se puede:

Obtener los beneficios de un servicio completamente administrado y alojado por AWS. CodeCommit provee alta disponibilidad y durabilidad del servicio y elimina el overhead de administrar nuestro propio software o hardware.

Almacenar código de manera segura. Los repositorios de CC están encriptados tanto en reposo (rest) como en tránsito (transit).

Los repositorios también permiten solicitudes de extracción (pull), donde esposible revisar y comentar los cambios de los demás antes de fusionar (merege) a las ramas (branches); las notificaciones son enviadas automáticamente por correo a los usuarios para informarlos de extracciones (pulls), comentarios, y más.

Los repositorios pueden escalarse para cumplir con nuestras necesidades de desarrollo de software. El servicio puede manejar repositorios con grandes números de branches, archivos grandes y largas historias de revisión. No hay límite del tamaño de los repositorios o el tipo de archivos que se pueden almacenar.

Los repositorios son almacenados cerca de otros recursos de ambiente productivo de la nube AWS, lo cual ayuda a incrementar la velocidad y frecuencia de nuestros ciclos de desarrollo. Está integrado con IAM y puede ser usado con otros servicios AWS en paralelo con otros repositorios. También se puede migrar a CodeCommit desde cualquier repositorio basado en Git. Por último, CC permite usar comandos de Git al igual que sus propios comandos de API AWS CLI. [2]

# Marco Teórico

Control de Versiones: Es un sistema que permite registrar los cambios a un archivo o conjunto de archivos a lo largo del tiempo de forma que es posible volver a versiones específicas de manera posterior. El sistema funciona en una variedad de aplicaciones y tipos de archivos.

Sistemas de Control de Versiones Locales: En muchos casos el método elegido para controlar versiones de manera local es copiar archivos a un directorio diferente dentro de la misma máquina de trabajo. Este método es común porque es muy simple, pero muy propenso a errores. Es fácil olvidar en cual directorio nos encontramos y accidentalmente escribir sobre el archivo equivocado o copiar archivos sobre los que no debíamos.

Para afrontar este problema, hace tiempo se construyeron sistemas de manejo de versiones (VCS por sus siglas en inglés) que consistían en una base de datos simple que mantenía todos los cambios de archivos bajo control. Una de las herramientas más populares de VCS era una llamada RCS, la cual sigue siendo distribuida con muchas computadoras hoy. RCS funciona manteniendo conjuntos de parches (diferencias entre archivos) en un formato especial en disco. Puede después recrear cómo se veía el archivo en un momento determinado agregando todos los parches.

Sistemas de Control de Versión Centralizado:

El siguiente problema fue tener que colaborar con desarrolladores de manera remota. Para lidiar con este problema, se desarrollaron los sistemas de Control de Versiones Centralizados (CVCSs) por sus siglas en inglés. Estos sistemas tienen un solo servidor que contiene todos los archivos versionados, y un número de clientes que revisan (check-out) los archivos desde una ubicación centralizada. Por muchos años este fue el estándar de control de versiones.

Pese a sus ventajas, este sistema tiene también serias desventajas. La mas obvia es el punto único de falla que representa el servidor centralizado. Los sistemas de VCS locales adolecen del mismo problema: siempre que se almacena la totalidad de la historia de un proyecto en un solo lugar se arriesga uno a perderlo todo.

Sistemas de Control de Versiones Distribuidos.

En un sistema de control de versiones Distribuido (DVCS) como Git, los clientes no solo hacen check-out de la fotografía (snapshot) más reciente, sino que replican el repositorio en su totalidad, incluyendo su historia. De tal forma que, si un servidor muere y estos sistemas colaboraban por medio de este servidor, cualquier repositorio de los clientes puede ser copiado para restaurar el servidor.

Git

La diferencia más importante entre Git y otros DVCSs es que Git no maneja los datos como una serie de deltas de archivos a lo largo del tiempo, sino como una secuencia de snapshots de un sistema de archivos en miniatura. En Git, cada vez que se realiza un commit se guarda el estado de nuestro proyecto, Git toma la fotografía de cómo se ve nuestro disco en dicho momento y almacena una referencia a dicho snapshot. De tal forma que Git considera los datos como un flujo de fotografías.

Operaciones Locales: De manera general, no se necesita información remota para trabajar con Git. Por ejemplo, para navegar la historia de un proyecto, Git no necesita ir al servidor para extraer la historia y desplegarla, simplemente la lee de la base de datos local. Esto también significa que hay muy pocas cosas que no se puedan hacer estando offline o fuera de la VPN.

Todo dentro de Git es verificado mediante un checksum antes de ser almacenado y es referenciado por dicho checksum. Por lo que es imposible cambiar algo dentro de un directorio o archivo sin que Git lo sepa.

Tres estados.

Git almacena los archivos en tres estados principales: commited, modificado y staged.

Commited significa que laos datos están almacenados de manera segura en la base de datos local.

Modificado significa que han cambiado, pero no han sido procesados con commit.

Staged significa que los archivos han sido marcados para ser transportados como parte del siguiente commit. [1]

# Diagrama

# 

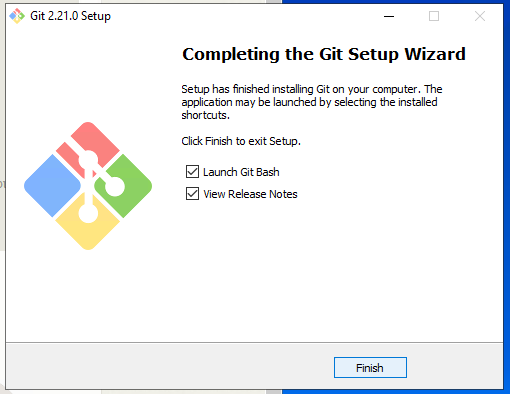
# Desarrollo de la Práctica.

En esta sección se debe presentar de forma cronológica el desarrollo de la práctica. Debe ser tan claro para servir como guía para repetir la práctica. En esta sección se pueden incluir imágenes, diagramas, o fotografía. Pero es importante no abusar o querer representar el desarrollo como una secuencia de fotografías.

## Configurar repositorio local

1. Instalar el cliente de git en una maquina

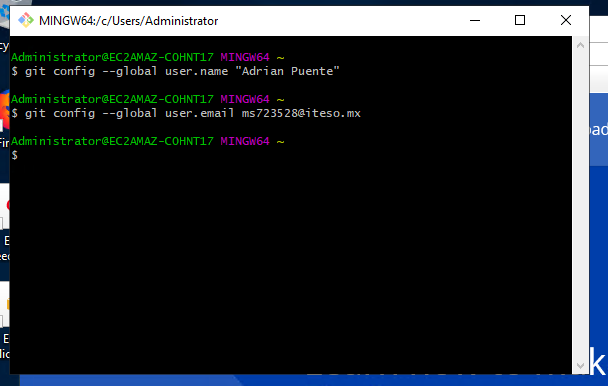
a. <https://git-scm.com>



2. Configurar git con su cuenta de iteso

a. git config --global user.name "John Doe"

b. git config --global user.email [johndoe@example.com](mailto:johndoe@example.com)

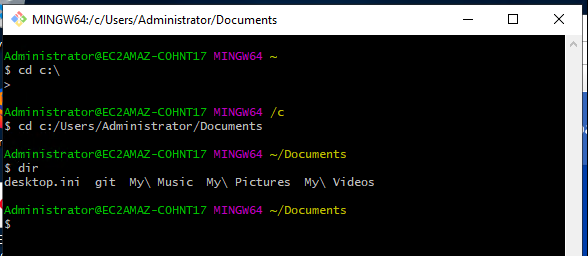


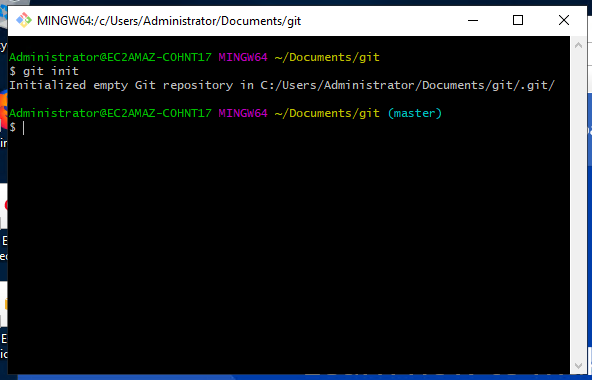
3. Configurar el repositorio

a. Crear en Mis Documentos una carpeta llamada git

b. Desde consola cambiarse a ese directorio (o en Windows click derecho sobre la

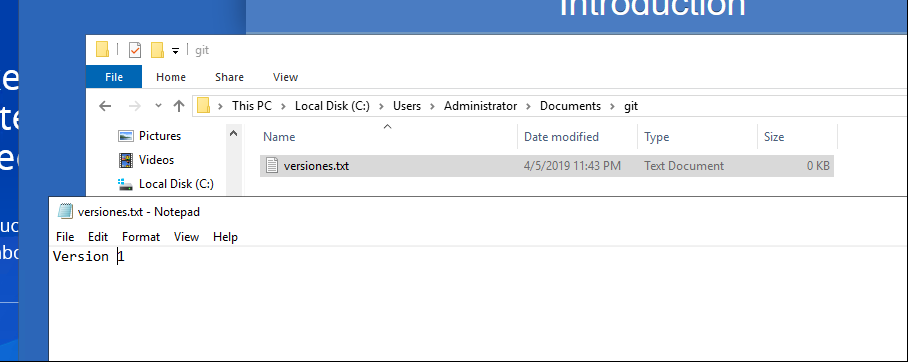
carpeta y “Git Bash here”)



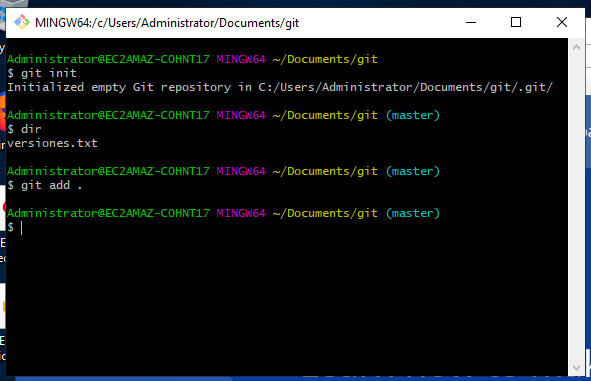
1. git init
2. 

4. Control de versiones

a. Crear en el directorio git un archivo: versiones.txt con el contenido: Versión 1

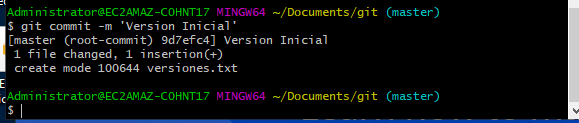


b. Añadir el archivo al seguimiento de git, desde la consola en la carpeta: git add .

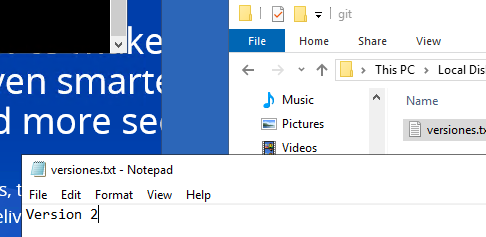


i. Es importante el “.” para incluir todos los archivos en el tracking

c. Hacer un commit de los cambios: git commit –poner mensaje versión inicial



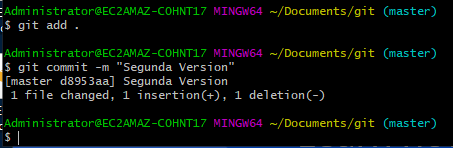
d. Editar el archivo y cambiar el contenido a: Versión 2



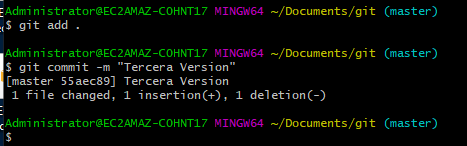
i. Hacer add y commit de los cambios:

1. git add .

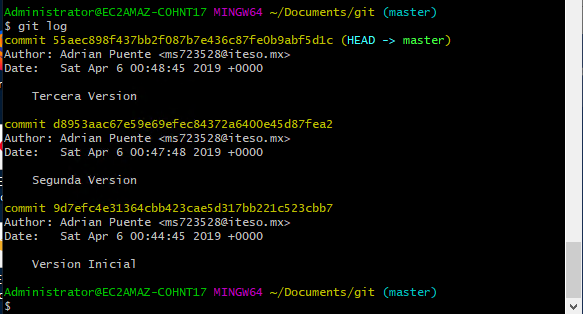
2. git commit -m “segunda version”



e. Hacer lo mismo para la Versión 3

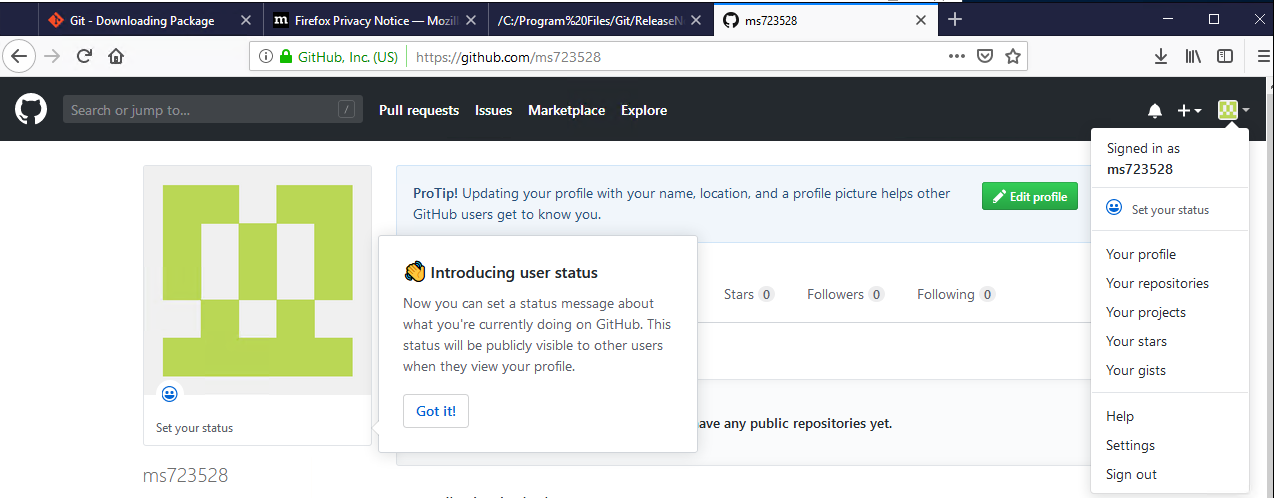


f. Consultar historial de vesiones: git log

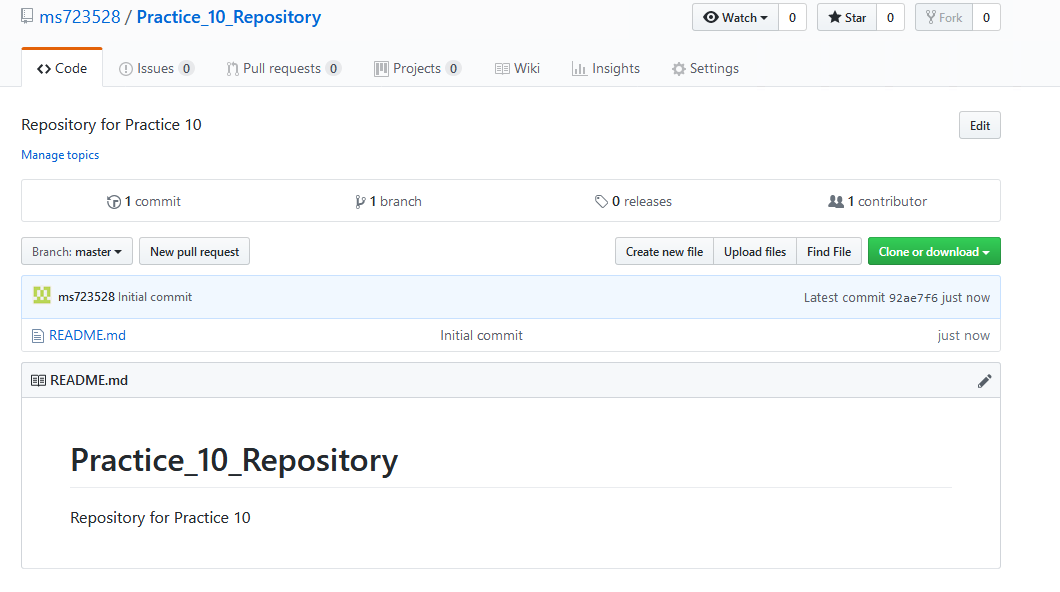


## Configurar repositorio en GitHub

1. Crear una cuenta con su correo de iteso en GitHub



2. Crear un repositorio en GitHub

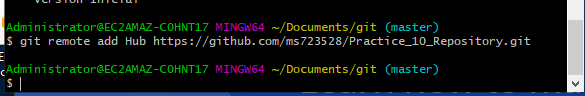


3. Consultar el URL del repositorio

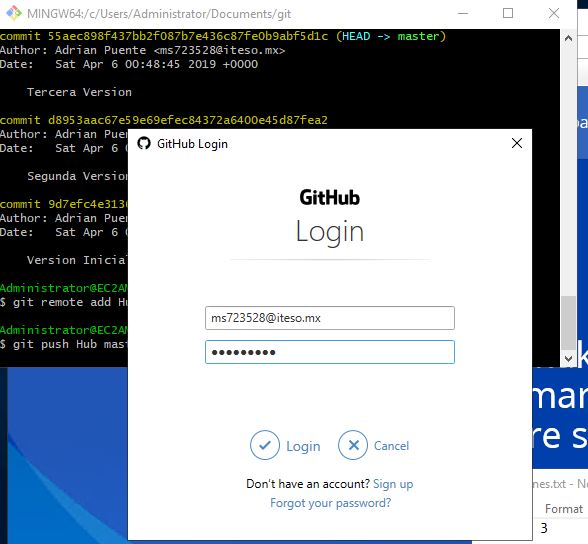
<https://github.com/ms723528/Practice_10_Repository.git>

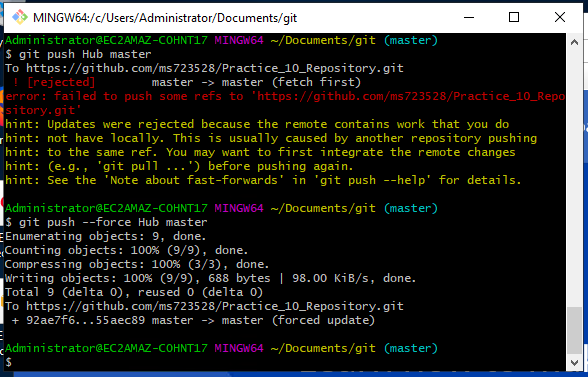
4. Añadir al repositorio local el repostorio de GitHub, desde la carpeta de git:

a. git remote add Hub <URL>



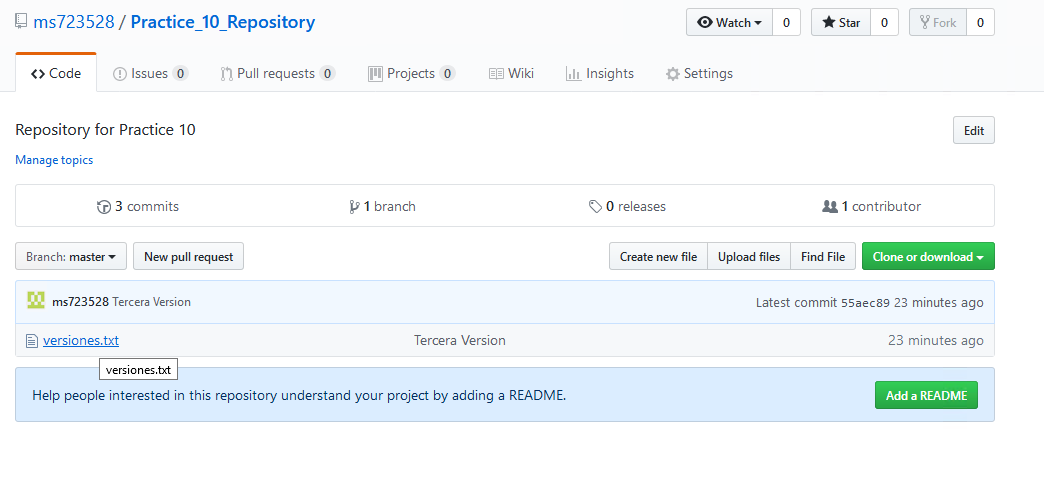
b. git push Hub master





5. Refrescar la pantalla del repositorio en GitHub ¿Qué cambio observan?

**Archivo *versiones.txt* está en el repositorio remoto.**



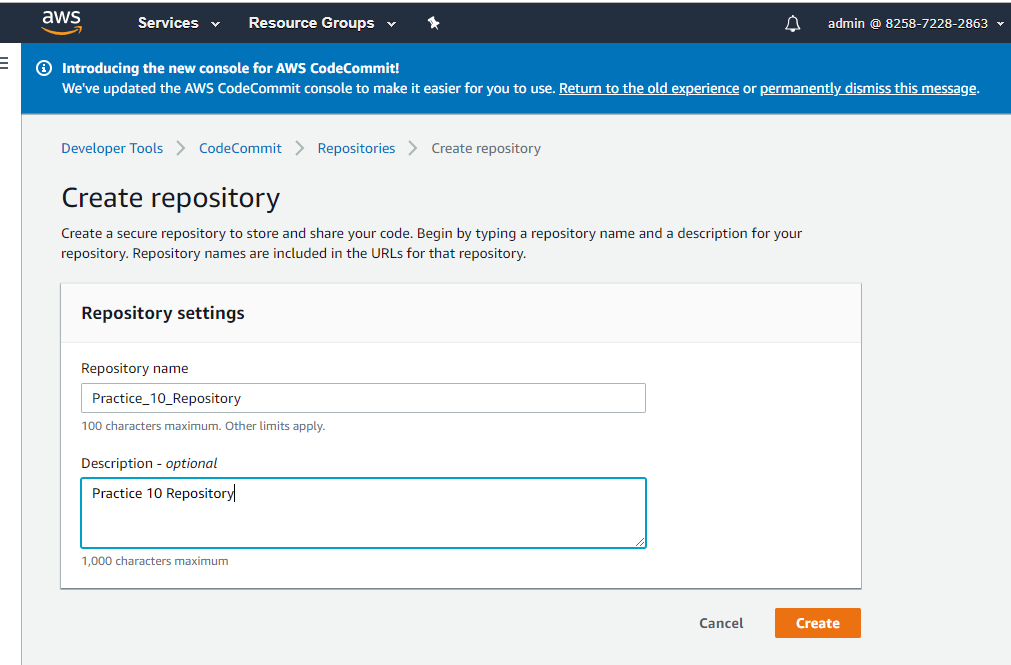
## Configurar repositorio en AWS CodeCommit

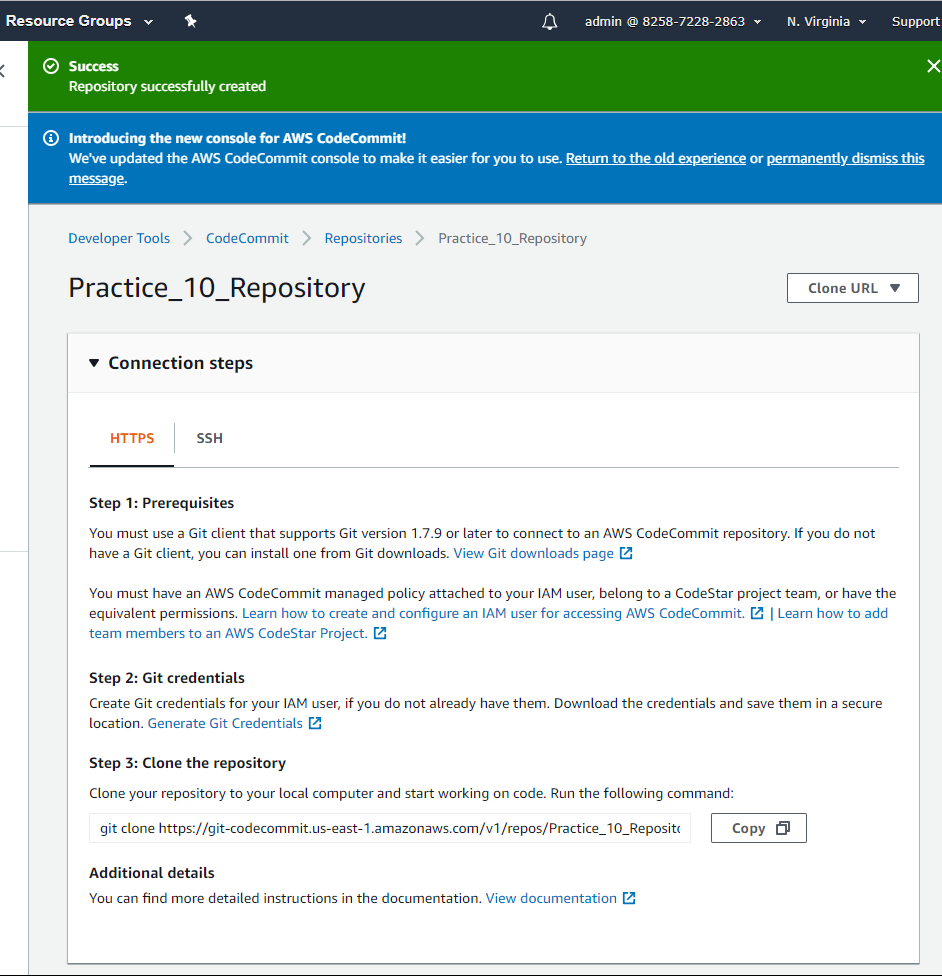
1. Crear credenciales para CodeCommit

a. Desde la consola de IAM y seleccionar el usuario deseado

b. En Security Credentials generar para: HTTPS para CodeCommit

2. Desde la consola de CodeCommit crear un repositorio



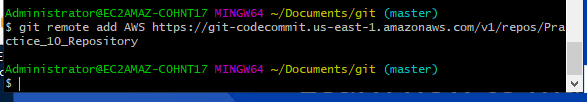


3. Consultar el URL del repositorio

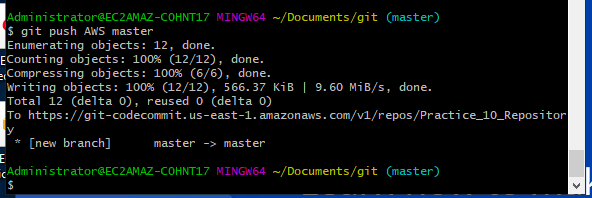
<https://git-codecommit.us-east-1.amazonaws.com/v1/repos/Practice_10_Repository>

4. Añadir al repositorio local el repostorio de CodeCommit, desde la carpeta de git:

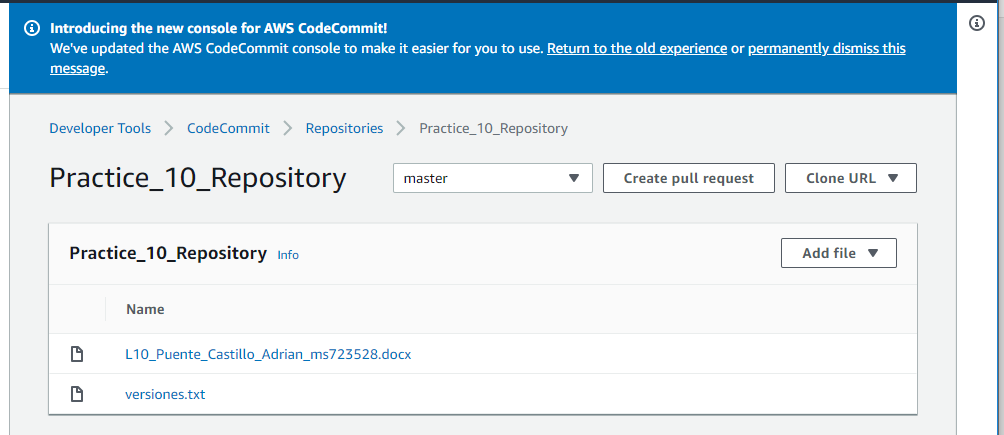
a. git remote add AWS <URL>



b. git push AWS máster

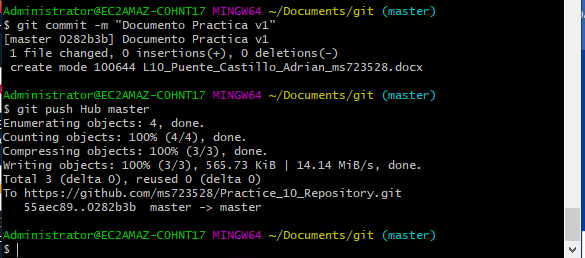


5. Consultar el repositorio de CodeCommit



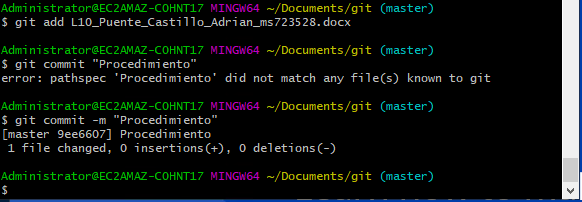
## Controlar las versiones del entregable de la práctica 10

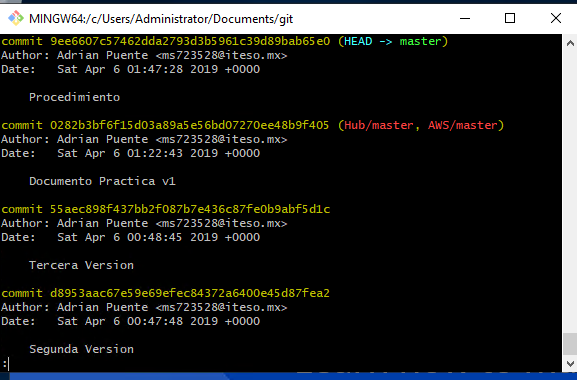
1. Crear el documento del reporte, y añadirlo para tracking

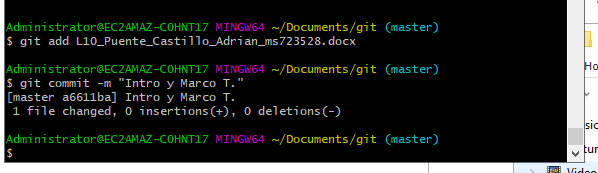


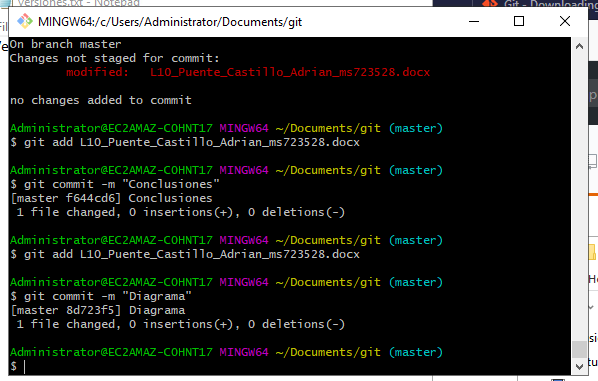
2. Hacer add y commit al terminar cada sección del reporte -Marco, Diagrama, etc- con el

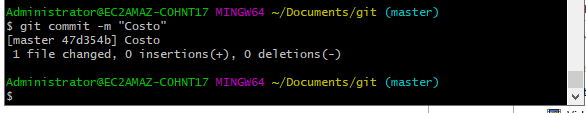
comentario correspondiente



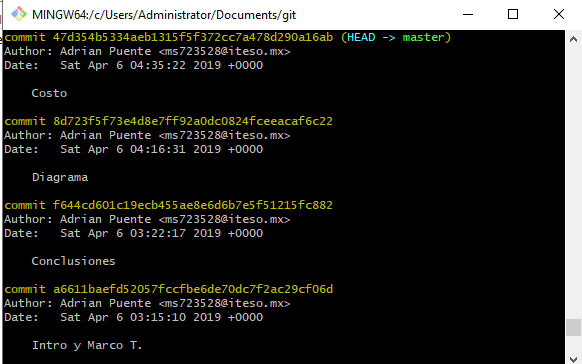








3. Mostrar log de los cambios al documento



4. Subir PDF del reporte a Moodle, compartir el URL del documento en CodeCommit

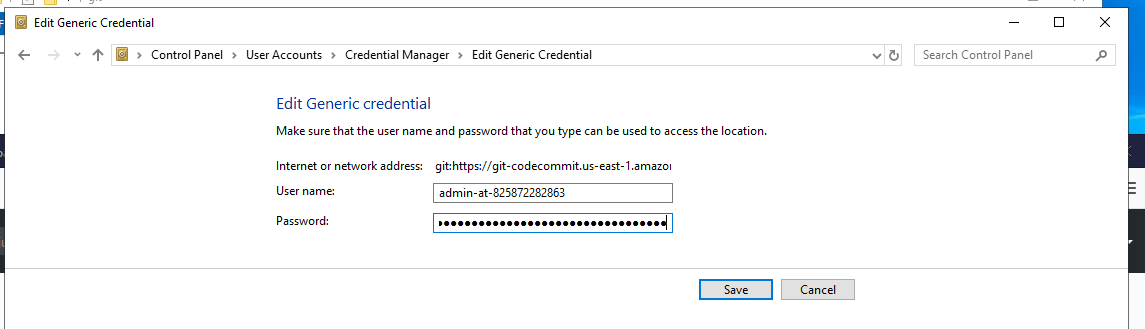
<https://git-codecommit.us-east-1.amazonaws.com/v1/repos/Practice_10_Repository>

<https://github.com/ms723528/Practice_10_Repository.git>

# Problemas y Soluciones

Se deberá enumerar todos los problemas que se encontraron durante el desarrollo de la práctica y como se resolvieron.

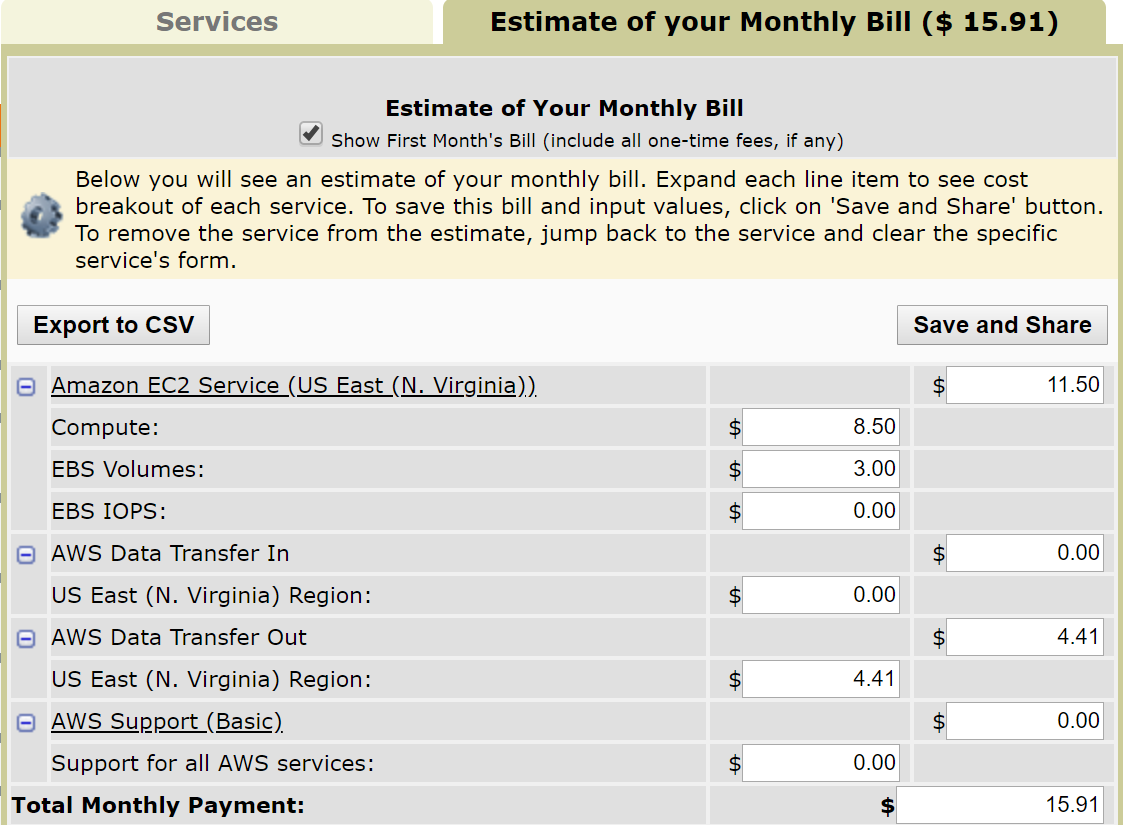
**Se introdujo una contraseña incorrecta para conexión a la cuenta de CodeCommit. Se corrigió en el Credential Manager de Windows**



# Experimentos y Resultados.

# Costo

La solución consta de sólo una instancia Bastión t2.micro (8.50 USD) con un volumen de 30 GiB (3.00 USD) sobre la que se instaló git y se creó un repositorio local y con ella nos conectamos a 2 repositorios remotos: uno en git-hub con la cuenta de correo de ITESO y un segundo repositorio de CodeCommit en AWS. Se incluye también el costo de 50 GiB de datos de subida y bajada (4.41 USD) y un repositorio de CodeCommit. Según la documentación CodeCommit está incluido sin costo para los primeros 5 developers en la cuenta de AWS, por lo que la única cuenta usada no tiene costo. El almacenamiento se cobra a razón de $0.06 GiB por mes. Agregando todo suma 15.91 USD por mes:



<https://calculator.s3.amazonaws.com/index.html#r=IAD&s=EC2&key=calc-86DDB534-0394-4C7D-8AF7-C39BA0F40AC3>

# Conclusiones

El servicio de CodeCommit permite agregar control de versiones a nuestros proyectos dentro de AWS que pueden ser integrados a otros servicios de AWS para hacer ciclos rápidos de deployment a ambientes tanto de pruebas como productivos. Esta funcionalidad permite agregar un nivel de madurez a nuestros procesos y flexibilidad a la operación de estos.

# Bibliografia

[2]

“What Is AWS CodeCommit? - AWS CodeCommit.” [Online]. Available: <https://docs.aws.amazon.com/codecommit/latest/userguide/welcome.html>. [Accessed: 05-Apr-2019].

[1]

“Git - Book.” [Online]. Available: <https://git-scm.com/book/en/v2>. [Accessed: 05-Apr-2019].