Reporte de práctica de laboratorio

Control de versiones (CodeCommit)

Marzo 28 de 2020

Maestría en Sistemas Computacionales

Aplicaciones y Servicios en la Nube

Prof. Mtro. Rodolfo Luthe Ríos

Alumna: Sarahi Partida Ochoa

ms710662@iteso.mx

ITESO

Universidad Jesuita
de Guadalajara

Introducción

Para esta práctica haremos uso de las herramientas de control de versiones; una de ellas es github, donde solo necesitamos crear una cuenta con nuestro correo electrónico, instalar localmente git donde configuraremos dicho correo para luego hacer los commits que necesitemos, los cuales nos ayudarán a tener la verificación de los cambios que le hemos hecho al proyecto que hayamos compartido; luego usaremos CodeComit que es parte de AWS, donde también es otra herramienta para control de versiones; para este servicio también haremos uso de git, cuando generemos nuestro proyecto en CodeCommit, el link que se nos genera se lo pasaremos a git para luego sincronizarlo y poder monitorear los cambios que realicemos ya sea a un archivo o a varios del folder que tengamos compartido.

Marco Teórico

Un control de versiones es un sistema que registra los cambios realizados en un archivo o conjunto de archivos a lo largo del tiempo, de modo que puedas recuperar versiones específicas más adelante. Entre las opciones que tenemos está GitHub el cual crea un entorno que permite almacenar código en un servidor remoto, brinda la capacidad de compartir código con otras personas y facilita que más de una persona agregue, modifique o elimine el código en el mismo archivo y proyecto, manteniendo una fuente de verdad para ese archivo. [1]

Los sistemas de control de versiones (también conocidos como gestión de control de origen o SCM) son software que realiza un seguimiento de cada versión de cada archivo en un proyecto de codificación, una marca de tiempo de cuándo se creó esa versión y el autor de esos cambios. El flujo de trabajo básico de codificación con soporte de sistema de control de versiones es el siguiente:

- Crear un proyecto, generalmente en una carpeta en su computadora.
- Decir al sistema de control de versiones que realice un seguimiento de los cambios del proyecto/carpeta.
- Cada vez que el proyecto esté funcionando, decirle al sistema de control de versiones que lo guardará como la próxima versión.
- Si alguna vez se necesita volver a una versión anterior, se puede solicitar al sistema de control de versiones que vuelva a la versión anterior que se necesite.

Un punto de partida importante para cualquier canalización de CI/CD es un repositorio de control de fuente simple pero funcional. Tradicionalmente, esto se configuraría en uno o más servidores físicos en forma de un repositorio Git o SVN que los desarrolladores usarían para enviar su código y actualizaciones; sin embargo, mantener dichos repositorios de código y escalarlos siempre sería un desafío. AWS CodeCommit es un servicio de control de origen administrado que permite a los desarrolladores almacenar de forma segura su código en la nube de AWS. Ofrece muchas de las características que necesitaría y utilizaría al trabajar con diferentes repositorios de control de código fuente, como ramificaciones, confirmaciones, reversiones y mucho más.[2]

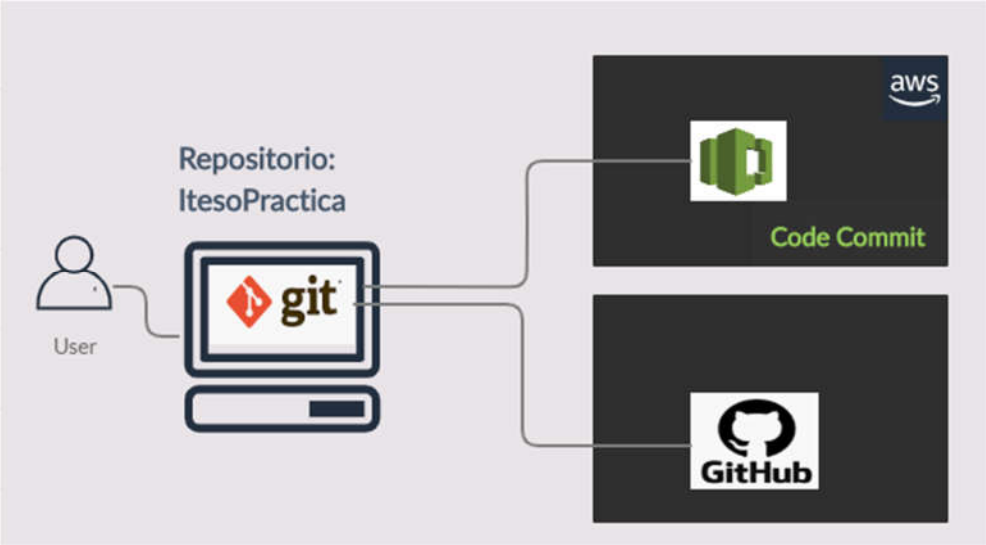
• **AWS CodeBuild:** AWS CodeBuild es un servicio de creación de código que los desarrolladores pueden aprovechar para automatizar sus compilaciones de código fuente, pruebas, ejecuciones y empaque de código para implementaciones. Al igual que sus otros servicios de contraparte en Code Suite, CodeBuild también es administrado completamente por AWS, lo que elimina cualquier sobrecarga administrativa innecesaria, como parchear o escalar el software de creación de código. CodeBuild es altamente extensible y también se integra fácilmente con sus flujos de trabajo de CI/CD existentes.

• **AWS CodeDeploy:** con el código de la aplicación almacenado de forma segura y compilado, el paso final requiere que el código se implemente en una flota de instancias EC2. Usando CodeDeploy, un desarrollador puede automatizar las implementaciones de código en cualquier entorno que se ejecute fuera de las instancias de EC2, así como de los servidores que se ejecutan en un centro de datos local. CodeDeploy esencialmente elimina las complejidades de implementación al permitirle automatizar la entrega de su código en miles de instancias sin tener que pasar por tiempos de inactividad importantes.

• **AWS CodePipeline:** AWS CodePipeline es un servicio completo de CI/CD provisto por AWS que los desarrolladores pueden aprovechar para construir sus pipelines de CI/CD de extremo a extremo, ya sea mediante el uso del conjunto de servicios AWS Code Suite o incluso con otras herramientas populares de terceros, como GitHub, Jenkins, etc. Usando CodePipeline, también puede crear y definir modelos de lanzamiento de software personalizados mediante los cuales su aplicación se actualiza con el último conjunto de actualizaciones, probado y empaquetado para el próximo conjunto iterativo de implementaciones.

Diagrama

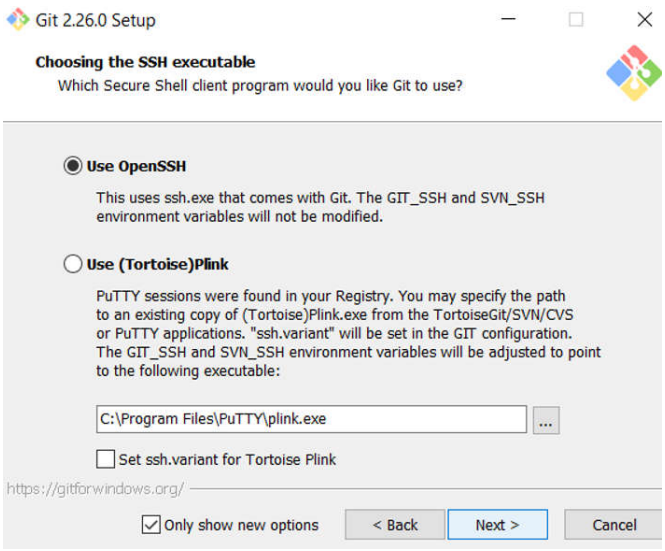
En el diagrama se muestran los dos tipos de control de versiones que se utilizaron para la practicas que fue: CodeCommit y GitHub



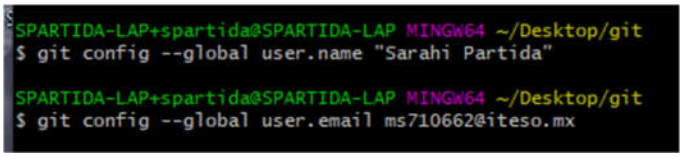
Desarrollo de la Práctica.

Configurar repositorio local

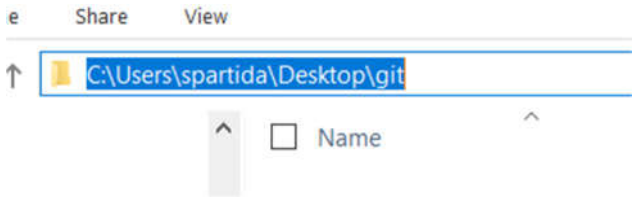
Como primer paso es instalar Git en nuestro ordenador:



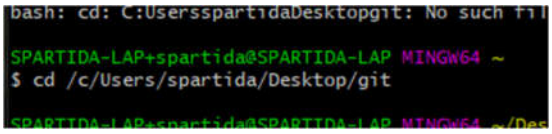
Luego se configuró git con mi cuenta de iteso; dentro de la consola de git es necesario ejecutar los dos comandos de global user.name y user email, como se muestra en la imagen:



Después configuré el repositorio, donde generé una carpeta llamada git en mi Desktop:

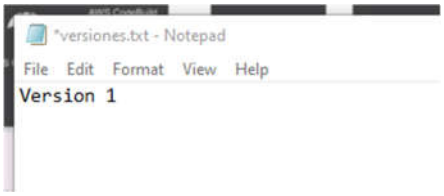


Luego me fui a la ruta de dicha carpeta dentro de la consola de git:



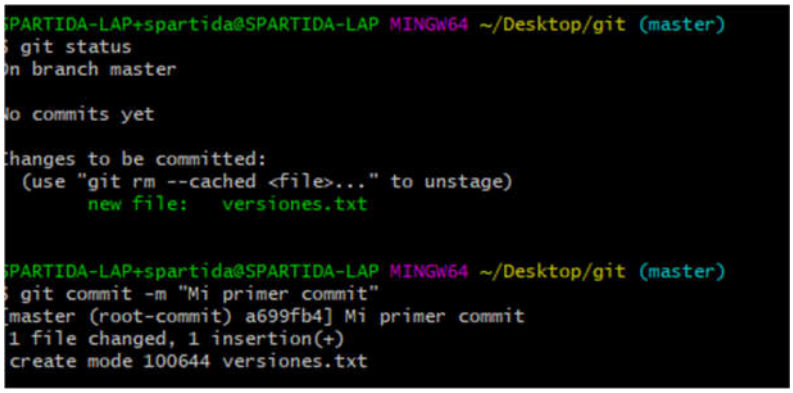
Para agregarlo al ambiente de git este directorio usamos git init:

Es necesario configurar el control de versiones, para esto se creó un archivo llamado versiones.txt con el contenido Version 1:



Se agregó el archivo al seguimiento de git, con el comando “git add .”, el punto significa que agregara lo que este en path actual de la consola.

Si queremos ver el status de nuestro repositorio con un git status nos dice si tenemos archivos por hacer commit o no. Luego hice mi primer commit con un comentario de que era el primero:



Genere un segundo commit donde modifique el archivo txt con otro comentario, se siguieron los pasos anteriores de “git add .” y luego otro commit con su respectivo comentario, donde pude observar que ya tenía dos commits:

```
SPARTIDA-LAP+spartida@SPARTIDA-LAP MINGW64 ~/Desktop/git (master)
$ git add .

SPARTIDA-LAP+spartida@SPARTIDA-LAP MINGW64 ~/Desktop/git (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   versiones.txt
SPARTIDA-LAP+spartida@SPARTIDA-LAP MINGW64 ~/Desktop/git (master)
$ git commit -m "Mi segundo commit"
[master d1400a7] Mi segundo commit
 1 file changed, 2 insertions(+), 1 deletion(-)

SPARTIDA-LAP+spartida@SPARTIDA-LAP MINGW64 ~/Desktop/git (master)
$ git status
On branch master
nothing to commit, working tree clean

SPARTIDA-LAP+spartida@SPARTIDA-LAP MINGW64 ~/Desktop/git (master)
```

Se nos pidió una versión 3 realizando los pasos previos:

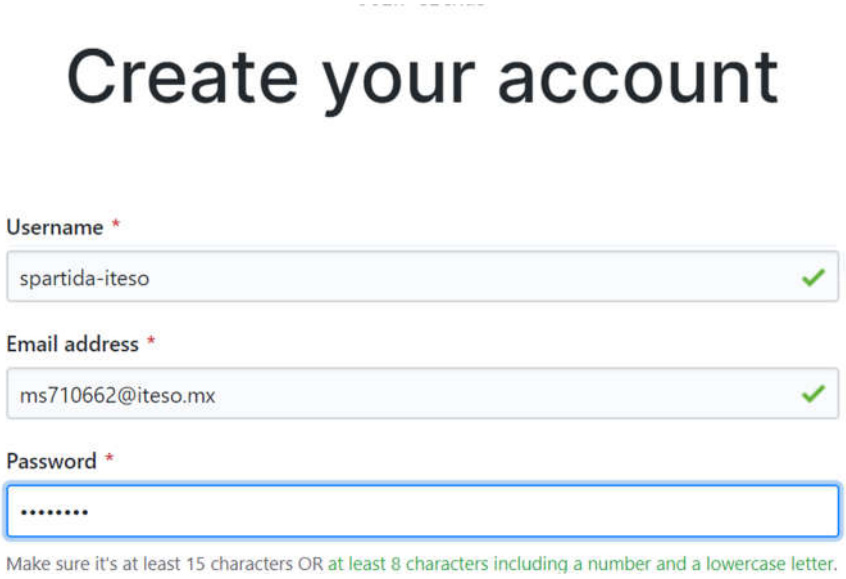
```
SPARTIDA-LAP+spartida@SPARTIDA-LAP MINGW64 ~/Desktop/git (master)
$ git add .

SPARTIDA-LAP+spartida@SPARTIDA-LAP MINGW64 ~/Desktop/git (master)
$ git commit -m "Mi tercer commit"
[master 8755175] Mi tercer commit
 1 file changed, 2 insertions(+), 1 deletion(-)

SPARTIDA-LAP+spartida@SPARTIDA-LAP MINGW64 ~/Desktop/git (master)
```

Para visualizar los commits que se han realizado podemos usar el comando git log:

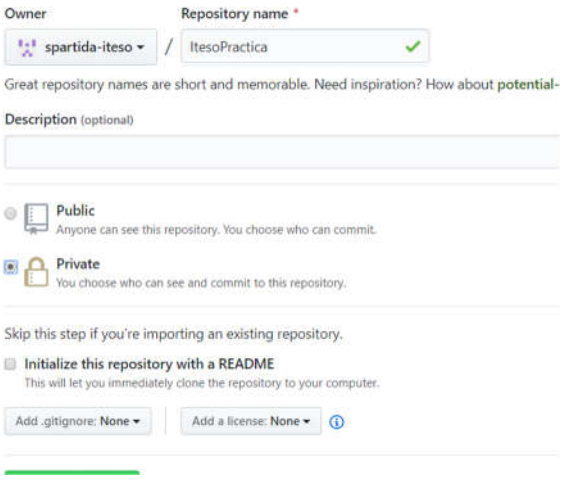
```
commit 8755175d9eeb68dfe700f4794202988c2ad61db4 (HEAD -> master)
Author: Sarahi Partida <ms710662@iteso.mx>
Date:   Sat Mar 28 00:00:41 2020 -0600

    Mi tercer commit

commit d1400a7be2fc0a591f505fa10d8d42b7ffa96460
Author: Sarahi Partida <ms710662@iteso.mx>
Date:   Fri Mar 27 23:59:13 2020 -0600

    Mi segundo commit

commit a699fb4c807a1dc48fc448699875bfce1cde5b3c
Author: Sarahi Partida <ms710662@iteso.mx>
Date:   Fri Mar 27 23:54:20 2020 -0600

    Mi primer commit

SPARTIDA-LAP+spartida@SPARTIDA-LAP MINGW64 ~/Desktop/git (master)
```

Configurar repositorio en GitHub:

Para esta parte se generó una cuenta con mi correo del Iteso en GitHub



Luego de verificar mi cuenta, se configuro mi repositorio como privado y sin seleccionar el "readme" ya que nosotros creamos previamente el archivo txt.:



El link de mi repositorio es:

https://github.com/spartida-iteso/ItesoPractica.git

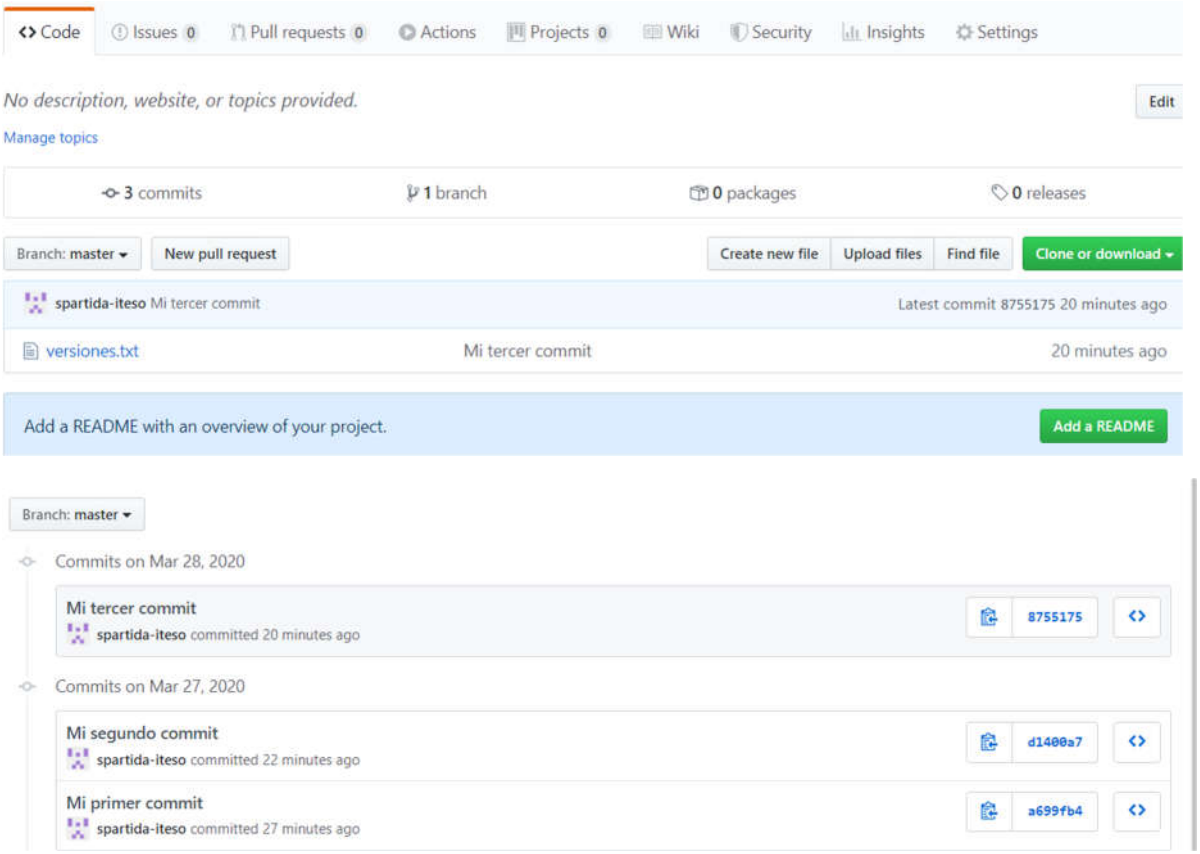
Para hacer la sincronización con mi repositorio local fue necesario correr los siguientes comandos:

-git remote add Hub https://github.com/spartida-iteso/ItesoPractica.git

-git push Hub master (con este comando estamos agregando los archivos que tenemos locales)

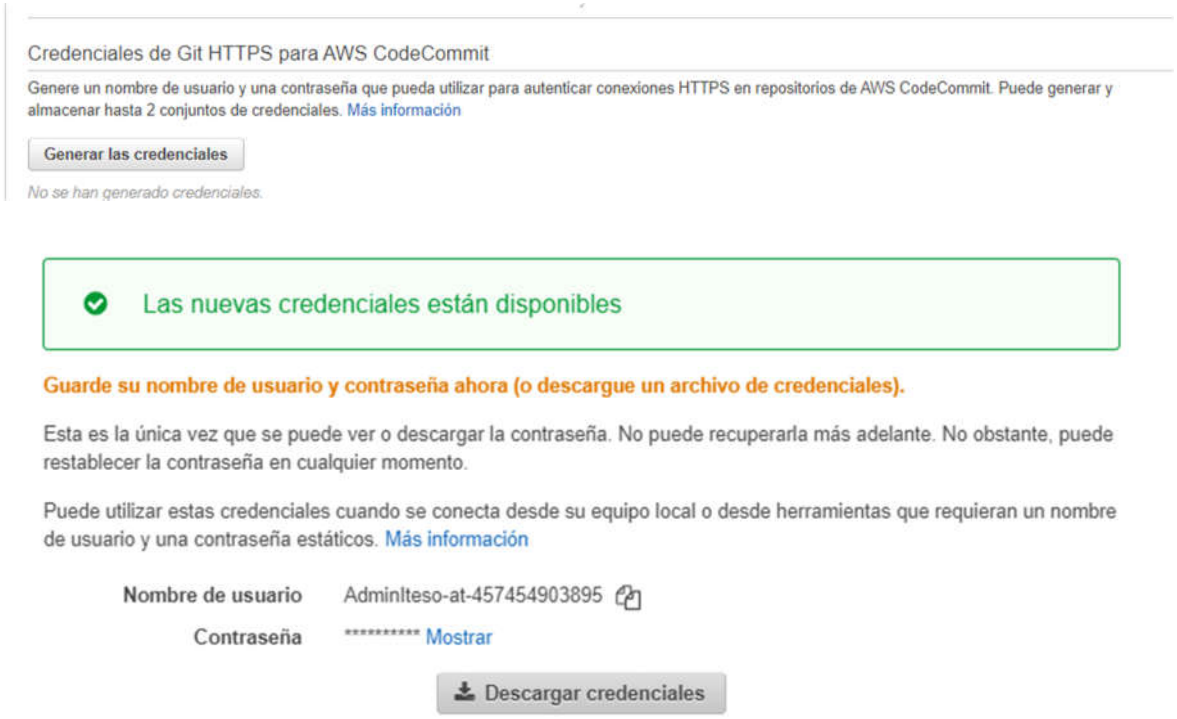
```
SPARTIDA-LAP+spartida@SPARTIDA-LAP MINGW64 ~/Desktop/git (master)
$ git remote add Hub https://github.com/spartida-iteso/ItesoPractica.git

SPARTIDA-LAP+spartida@SPARTIDA-LAP MINGW64 ~/Desktop/git (master)
$ git push Hub master
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 4 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (9/9), 751 bytes | 68.00 KiB/s, done.
Total 9 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/spartida-iteso/ItesoPractica.git
 * [new branch]      master -> master

SPARTIDA-LAP+spartida@SPARTIDA-LAP MINGW64 ~/Desktop/git (master)
$
```

Actualizando la página de github se puede observar que ya tenemos el archivo de versiones con los 3 commits que hice previamente:

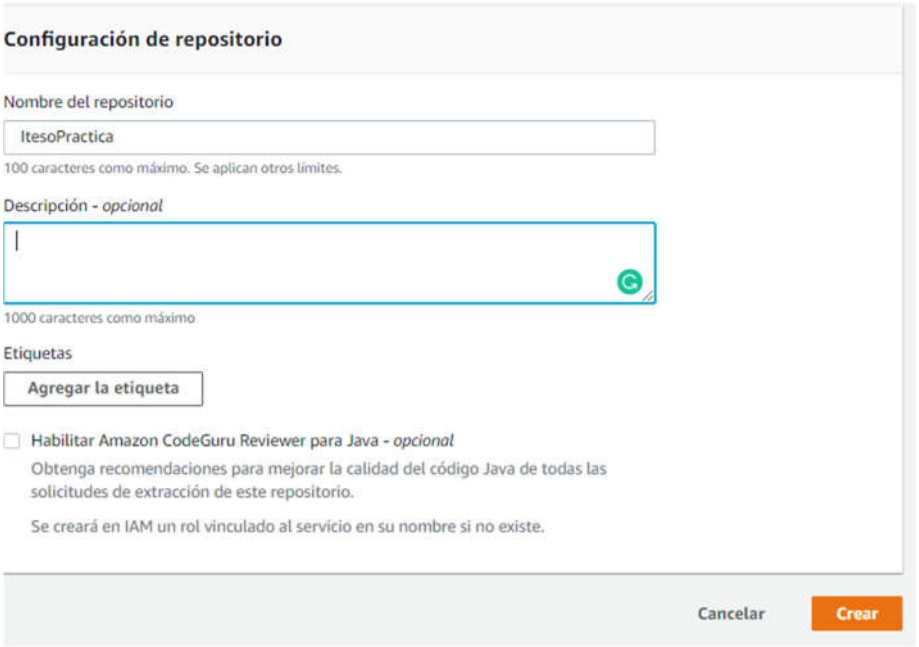


Configurar repositorio en AWS CodeCommit

Para generar las credenciales de CodeCommit es dirigirnos a IAM, luego al usuario que le queremos dar permisos, que en mi caso sería AdminIteso y le damos en generar credenciales git HTTPS CodeCommit



Ya con las credenciales se puede crear el repositorio en CodeCommit:



Luego copie el URL del repositorio y lo agregue a mi repositorio local, tal como se muestra en la imagen:

Paso 3: clonación del repositorio

Clone su repositorio en su equipo local y comience a trabajar en el código. Ejecute el siguiente comando:

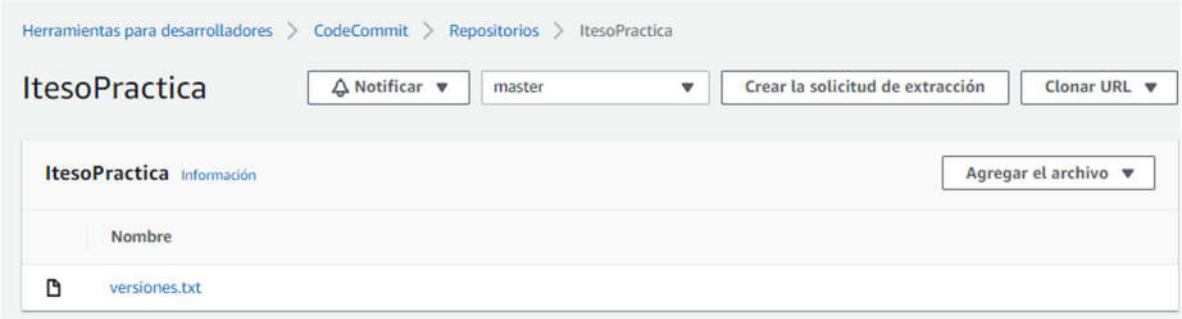
```
git clone https://git-codecommit.us-east-1.amazonaws.com/v1/repos/ItesoPractica
```

Copiar

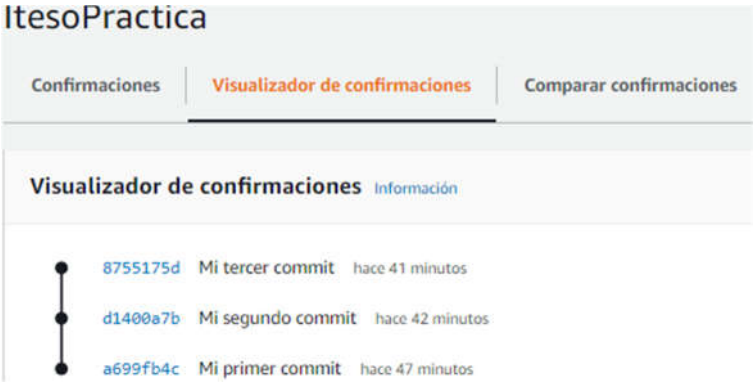
Detalles adicionales

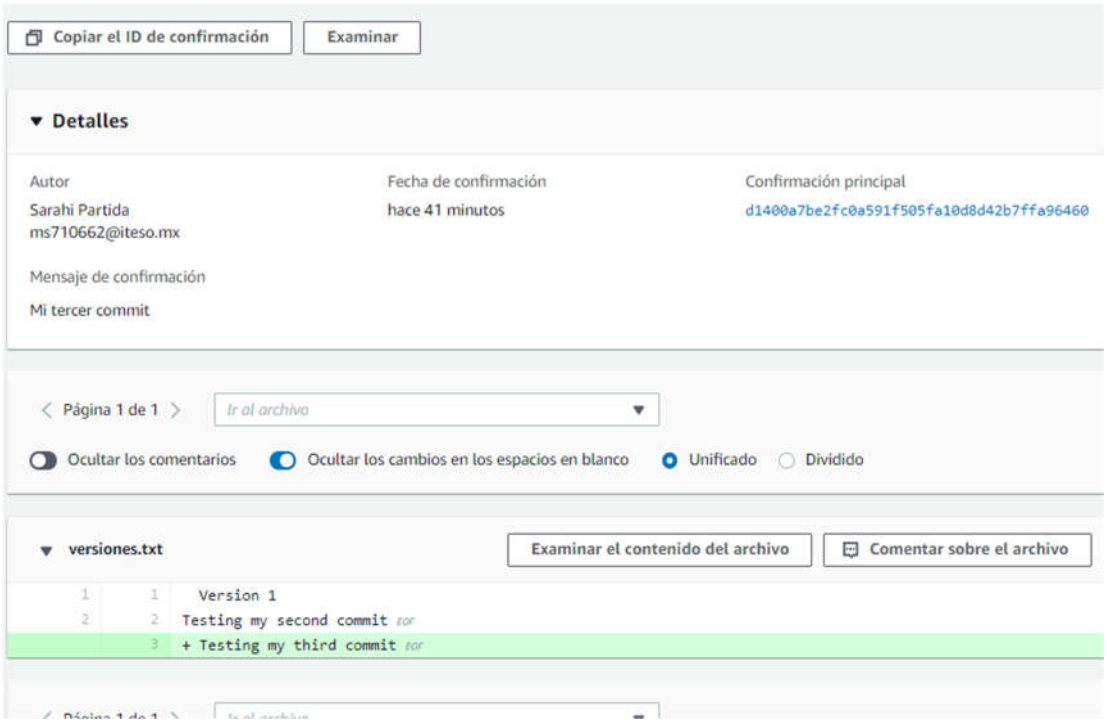
```
SPARTIDA-LAP+spartida@SPARTIDA-LAP MINGW64 ~/Desktop/git (master)
$ git remote add AWS https://git-codecommit.us-east-1.amazonaws.com/v1/repos/ItesoPract
ca

SPARTIDA-LAP+spartida@SPARTIDA-LAP MINGW64 ~/Desktop/git (master)
$ git push AWS master
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 4 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (9/9), 751 bytes | 107.00 KiB/s, done.
Total 9 (delta 0), reused 0 (delta 0), pack-reused 0
To https://git-codecommit.us-east-1.amazonaws.com/v1/repos/ItesoPractica
 * [new branch]      master -> master

SPARTIDA-LAP+spartida@SPARTIDA-LAP MINGW64 ~/Desktop/git (master)
$
```

Si consultamos la página de CodeCommit podemos observar que ya tenemos el archivo de versiontes.txt cargado:



Podemos visualizar los commits:





Controlar las versiones del entregable de la práctica 10

Uno de los requisitos de esta práctica era realizar tracking de cada apartado, haciendo add y commit de cada apartado y mostrar el log de los cambios:

¿Es conveniente el uso de control de versiones para documentos que no son código?

Depende el documento que necesitemos trackear; por ejemplo, para specs es de vital importancia usar un control de versiones ya que los productos de las empresas cambias a cada momento, si existe una modificación del producto, debemos saber cuándo sucedió; ya que si viene un cliente y te menciona de un posible error, poder confirmar si en el tiempo que lo adquirió estaba soportado o se depreco; otra parte es para nosotros que probamos funcionalidades y sale un nuevo reléase, para nosotros agregar nuevas pruebas nos basamos en las fechas que se publicaron dichos cambios o si es algo antiguo si realmente la pena implementar un test case o no.

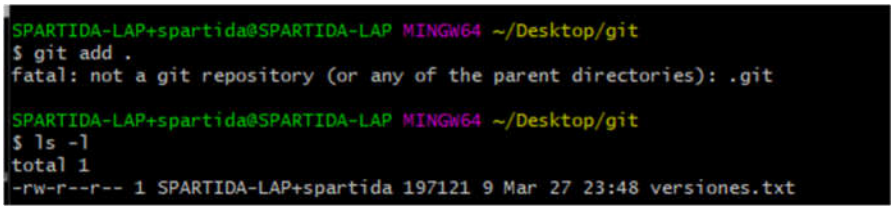
¿Cómo comparas GitHub y CodeCommit?

En funcionalidad podría decir que me sirven para lo mismo, para trackear cambios en mi código/documentos. Sé que muchos devops usan Github ya que es de libre acceso, ya si tenemos un uso más rudo tiene un costo y CodeCommit tiene un costo de 1 dólar por mes con 10GB/mes y 2000 solicitudes por mes las cuales no son acumulables. Aquí dependerá en que plataforma estés trabajando, la seguridad y así se decidirá con que versionamiento se trabajará.

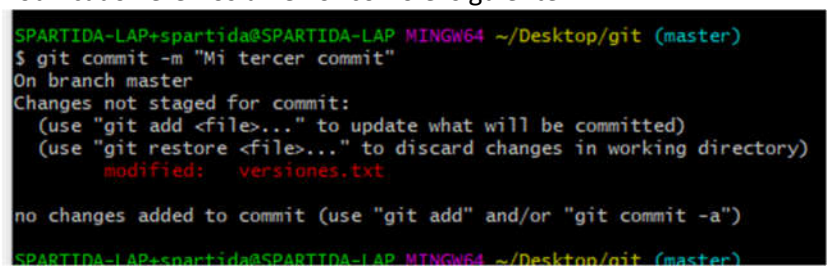
Problemas y Soluciones

A continuación, enlisto los problemas enfrentados:

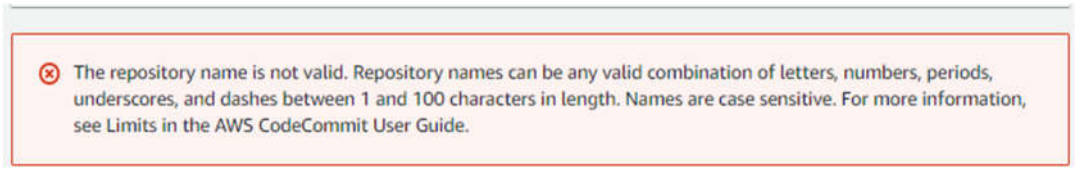
1. Si no inicializamos nuestro repositorio tendremos un error de que no puedes agregar nada:



2. Si no hemos agregado nuestros cambios con git add y le damos git commit a un archivo que ya hemos modificado veremos un error como el siguiente:



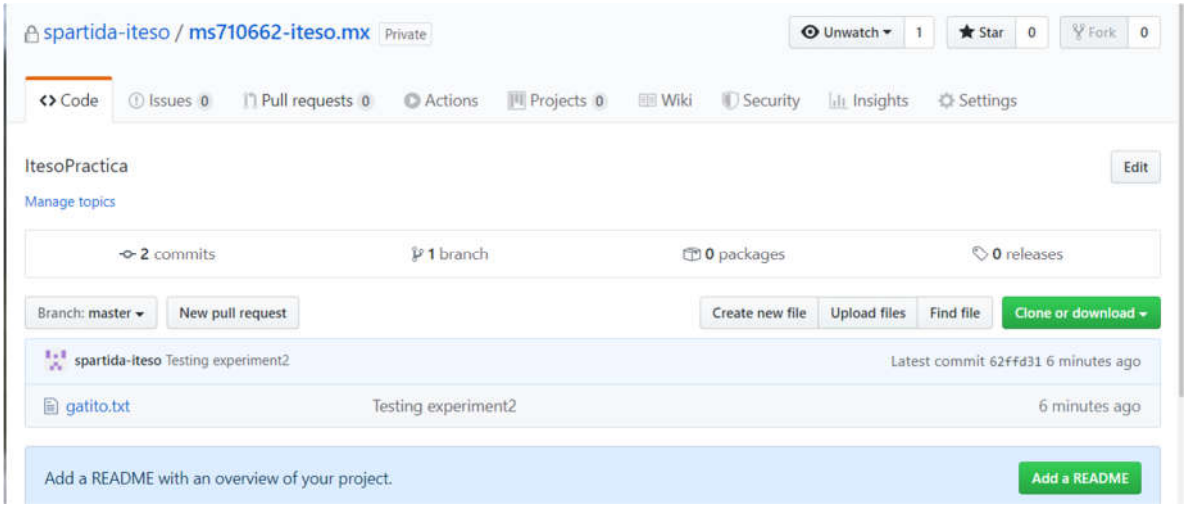
3. Cuando hice copy/paste del nombre de mi repositorio en CodeCommit no me permitía generarlo, al parecer copio un carácter invalido sin darme cuenta, por lo que hice lo escribí el nombre igual y ya me dejo crearlo:



Experimentos y Resultados.

Generé otros repositorios en github y pude ver que podemos tener varios y controlar aparte, en cada nuevo repositorio hacemos los pasos previos con el nombre de la carpeta que queremos versionar.

```
SPARTIDA-LAP+spartida@SPARTIDA-LAP MINGW64 ~/Desktop/testSarahi (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   gatito.txt


SPARTIDA-LAP+spartida@SPARTIDA-LAP MINGW64 ~/Desktop/testSarahi (master)
$ git commit -m "Testing experiment"
[master (root-commit) 2afc872] Testing experiment
 1 file changed, 1 insertion(+)
 create mode 100644 gatito.txt

SPARTIDA-LAP+spartida@SPARTIDA-LAP MINGW64 ~/Desktop/testSarahi (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   gatito.txt

no changes added to commit (use "git add" and/or "git commit -a")

SPARTIDA-LAP+spartida@SPARTIDA-LAP MINGW64 ~/Desktop/testSarahi (master)
$ git commit -m "Testing experiment2"
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   gatito.txt

no changes added to commit (use "git add" and/or "git commit -a")

SPARTIDA-LAP+spartida@SPARTIDA-LAP MINGW64 ~/Desktop/testSarahi (master)
$ git add .

SPARTIDA-LAP+spartida@SPARTIDA-LAP MINGW64 ~/Desktop/testSarahi (master)
$ git commit -m "Testing experiment2"
[master 62ffd31] Testing experiment2
 1 file changed, 3 insertions(+), 1 deletion(-)

SPARTIDA-LAP+spartida@SPARTIDA-LAP MINGW64 ~/Desktop/testSarahi (master)
$ git remote add Hub https://github.com/spartida-iteso/ms710662-iteso.mx.git

SPARTIDA-LAP+spartida@SPARTIDA-LAP MINGW64 ~/Desktop/testSarahi (master)
$ git push Hub master
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (6/6), 472 bytes | 47.00 KiB/s, done.
Total 6 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/spartida-iteso/ms710662-iteso.mx.git
 * [new branch]      master -> master
```



Costo

Los costos acordes a las instancias creadas son los siguientes:

- En Github genere una cuenta gratuita por lo que no me generaría costo.
- CodeCommit me generaría un cobro de 1 dólar por mes si no sobrepaso los limites antes mencionados de 10MB/mes y 2000 solicitudes por mes.

Conclusiones

Si trabajamos en un ambiente de desarrollo es de vital importancia usar un controlador de versiones para evitar que sea un desorden los cambios que se realicen a cierto proyecto. En mi caso hace un par de años utilice git para control de versiones con teamforce la plataforma que se usaba como GitHub. Actualmente utilizo un controlador de versiones interno de Oracle el cual funciona similar a git; contamos con un branch principal que es donde se encuentran los cambios que pasaron por una revisión y fueron aprobados que vendría siendo el producto con las mejoras. Generamos un branch secundario cada desarrollador/tester para generar nuestras pruebas; las cuales se encuentran localmente. Cuando ya estamos seguros que no haremos cambios o que ya está prácticamente listo, es como hacer un commit de nuestros cambios, donde agregamos quienes deben revisar tu código y previamente se debió correr los test cases relacionados a los que estas modificando para evitar que introduzcas nuevos bugs a lo que ya está funcional. Ya después de estas verificaciones tus cambios son agregados al branch principal, donde se genera uno nuevo cada día por lo regular. Si no usáramos estas herramientas los costos de errores humanos seria alto, porque puede pasar que algún cambio sin validar puede eliminar o romper cosas o generar código sin calidad. He visto bugs donde han roto el instalador de la herramienta o funciones core, aun teniendo un controlador de versiones, lo que ha pasado es que no correr las pruebas de regresión de los cambios que hacen y los revisores no se percatan de esta parte.

Bibliografía

[1] “Chapter 1: Understanding the Git in GitHub - GitHub For Dummies.” https://learning.oreilly.com/library/view/github-for-dummies/9781119572671/c01.xhtml (accessed Mar. 27, 2020).

[2] “Transforming Application Development Using the AWS Code Suite - AWS Administration ; The Definitive Guide.” https://learning.oreilly.com/library/view/aws-administration/9781788478793/3b221537-7188-4194-92bf-d2ed87a5fce1.xhtml (accessed Mar. 26, 2020).