Reporte de práctica de laboratorio
Práctica 10 – Control de Versiones

25 de Marzo de 2020

Maestría en Informática Aplicada
Aplicaciones y Servicios en la Nube
Prof. Mtro. Rodolfo Luthe Ríos

Fernando Ulises Mérito del Campo
mi679752@iteso.mx

ITESO
Universidad Jesuita
de Guadalajara

Introducción

Los objetivos de la presente práctica son los siguientes:

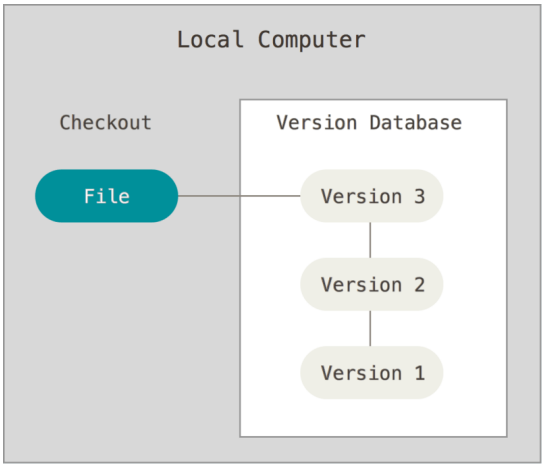
- Utilizar un cliente de control de versiones
- Configurar servicios de control de versiones centralizados
- Controlar las versiones de un documento

Marco Teórico

¿Qué es un control de versiones, y por qué debería importarte? 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 [1].

Dicho sistema te permite regresar a versiones anteriores de tus archivos, regresar a una versión anterior del proyecto completo, comparar cambios a lo largo del tiempo, ver quién modificó por última vez algo que pueda estar causando problemas, ver quién introdujo un problema y cuándo, y mucho más. Usar un VCS también significa generalmente que si arruinas o pierdes archivos, será posible recuperarlos fácilmente. Adicionalmente, obtendrás todos estos beneficios a un costo muy bajo.

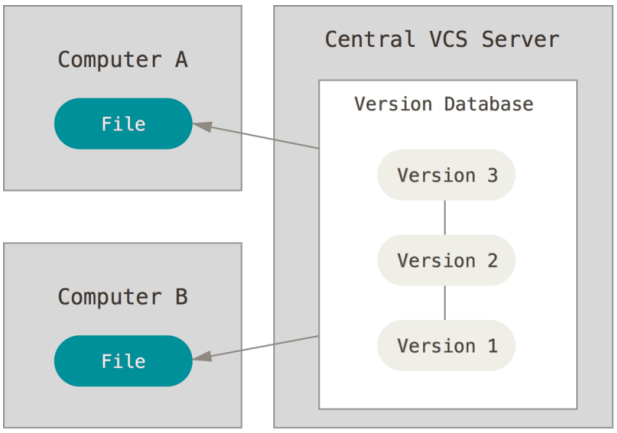
Un método de control de versiones, usado por muchas personas, es copiar los archivos a otro directorio (quizás indicando la fecha y hora en que lo hicieron, si son ingeniosos). Este método es muy común porque es muy sencillo, pero también es tremendamente propenso a errores. Es fácil olvidar en qué directorio te encuentras y guardar accidentalmente en el archivo equivocado o sobrescribir archivos que no querías.
Para afrontar este problema los programadores desarrollaron hace tiempo VCS locales que contenían una simple base de datos, en la que se llevaba el registro de todos los cambios realizados a los archivos [1].



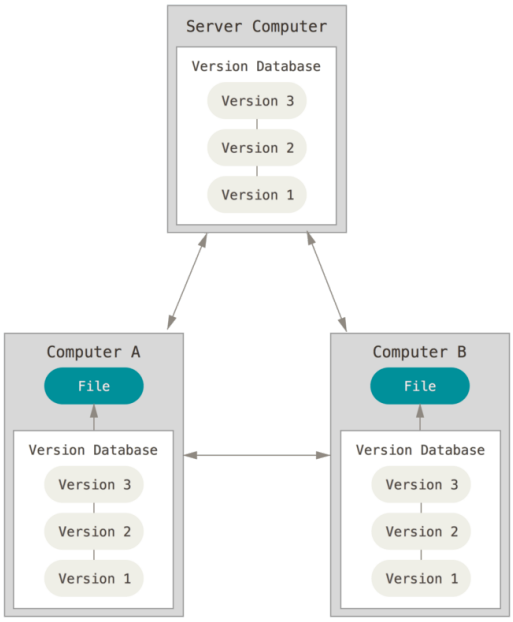
Una de las herramientas de control de versiones más popular fue un sistema llamado RCS, que todavía podemos encontrar en muchas de las computadoras actuales. Incluso el famoso sistema operativo Mac OS X incluye el comando rcs cuando instalas las herramientas de desarrollo. Esta herramienta funciona guardando conjuntos de parches (es decir, las diferencias entre archivos) en un formato especial en disco, y es capaz de recrear cómo era un archivo en cualquier momento a partir de dichos parches.

El siguiente gran problema con el que se encuentran las personas es que necesitan colaborar con desarrolladores en otros sistemas. Los sistemas de Control de Versiones Centralizados (CVCS por sus siglas en inglés) fueron desarrollados para solucionar este problema. Estos sistemas, como CVS, Subversion y Perforce, tienen un único servidor que contiene todos los archivos versionados

y varios clientes que descargan los archivos desde ese lugar central. Este ha sido el estándar para el control de versiones por muchos años.



Esta configuración ofrece muchas ventajas, especialmente frente a VCS locales. Por ejemplo, todas las personas saben hasta cierto punto en qué están trabajando los otros colaboradores del proyecto. Los administradores tienen control detallado sobre qué puede hacer cada usuario, y es mucho más fácil administrar un CVCS que tener que lidiar con bases de datos locales en cada cliente.
Sin embargo, esta configuración también tiene serias desventajas. La más obvia es el punto único de fallo que representa el servidor centralizado. Si ese servidor se cae durante una hora, entonces durante esa hora nadie podrá colaborar o guardar cambios en archivos en los que hayan estado trabajando. Si el disco duro en el que se encuentra la base de datos central se corrompe, y no se han realizado copias de seguridad adecuadamente, se perderá toda la información del proyecto, con excepción de las copias instantáneas que las personas tengan en sus máquinas locales. Los VCS locales sufren de este mismo problema: Cuando tienes toda la historia del proyecto en un mismo lugar, te arriesgas a perderlo todo [1].



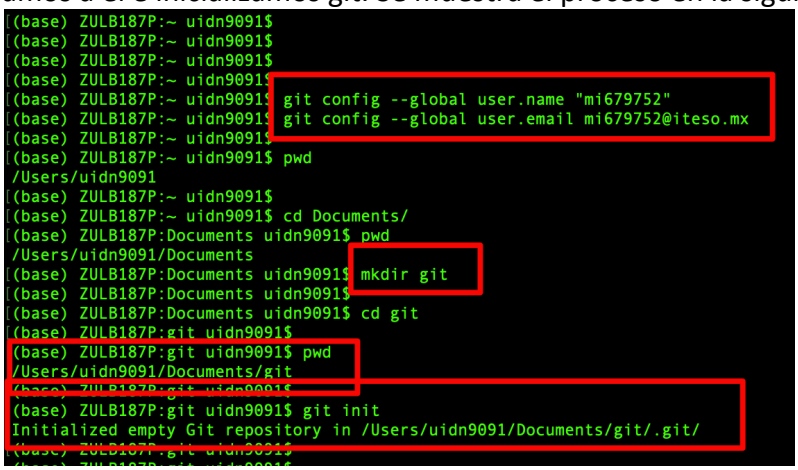
Los sistemas de Control de Versiones Distribuidos (DVCS por sus siglas en inglés) ofrecen soluciones para los problemas que han sido mencionados. En un DVCS (como Git, Mercurial, Bazaar o Darcs), los clientes no solo descargan la última copia instantánea de los archivos, sino que se replica completamente el repositorio. De esta manera, si un servidor deja de funcionar y estos sistemas estaban colaborando a través de él, cualquiera de los repositorios disponibles en los clientes puede ser copiado al servidor con el fin de restaurarlo. Cada clon es realmente una copia completa de todos los datos.

Además, muchos de estos sistemas se encargan de manejar numerosos repositorios remotos con los cuales pueden trabajar, de tal forma que puedes colaborar simultáneamente con diferentes grupos de personas en distintas maneras dentro del mismo proyecto. Esto permite establecer varios flujos de trabajo que no son posibles en sistemas centralizados, como pueden ser los modelos jerárquicos.

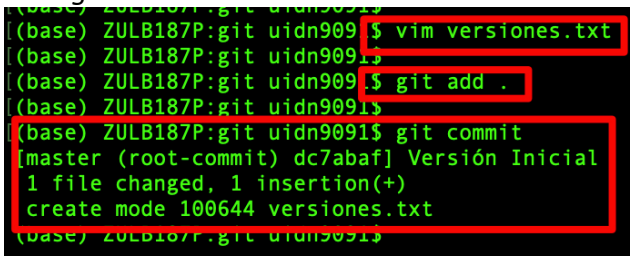
Desarrollo de la Práctica.

Configurar repositorio local

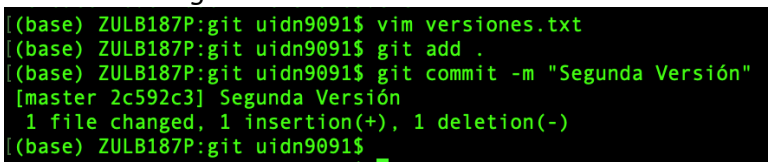
1. Se descarga e instala git desde el sitio “git-scm.com”
2. Se configura git con la cuenta de iteso, se crea en el folder de “Documents” el directorio “git”, entramos a el e inicializamos git. Se muestra el proceso en la siguiente imagen:



3. Para poder comprobar el control de versiones, se crea un archivo llamado versiones.txt con el contenido “Versión 1”
4. Para añadir el seguimiento de git a todo el contenido del folder, se corre el comando “*git add .*”
5. Se hace un commit de los cambios poniendo como mensaje “versión inicial”, esto corriendo el comando “*git commit*”



6. Se edita el archivo “versiones.txt” y se cambia su contenido a “Versión 2”, corriendo después de ello los siguientes comandos:
    a. *“git add .”*
    b. *Git commit -m “Segunda Versión”*



7. Se realiza lo mismo con “Versión 3”

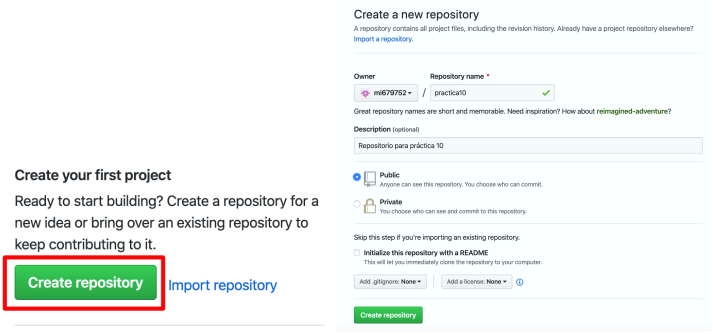
```
(base) ZULB187P:git uidn9091$
(base) ZULB187P:git uidn9091$ vim versiones.txt
(base) ZULB187P:git uidn9091$
(base) ZULB187P:git uidn9091$
(base) ZULB187P:git uidn9091$ git add .
(base) ZULB187P:git uidn9091$
(base) ZULB187P:git uidn9091$ git commit -m "tercera versión"
 [master 3d16d91] tercera versión
  1 file changed, 1 insertion(+), 1 deletion(-)
(base) ZULB187P:git uidn9091$
```

8. Ahora, se consulta el historial de versiones corriendo el comando *"git log"*

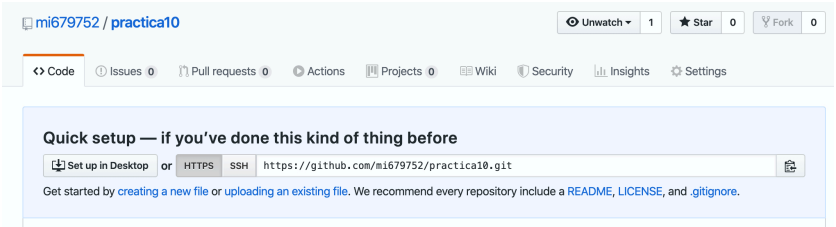
```
(base) ZULB187P:git uidn9091$ git log
commit 3d16d9171144969f228620fc0134aa16d617873c (HEAD -> master)
Author: mi679752 <mi679752@iteso.mx>
Date:   Wed Mar 25 20:17:51 2020 -0600

    tercera versión

commit 2c592c3c90964fa324caf10ef23b409ccae07475
Author: mi679752 <mi679752@iteso.mx>
Date:   Wed Mar 25 20:16:47 2020 -0600

    Segunda Versión

commit dc7abafc886fb9d26f65908be74822a06bc3d151
Author: mi679752 <mi679752@iteso.mx>
Date:   Wed Mar 25 20:13:21 2020 -0600

    Versión Inicial
(base) ZULB187P:git uidn9091$
```

Configurar Repositorio en GitHub

1. Se ingresa a github.com y se crea una cuenta nueva, usando el nombre de usuario y correo del iteso
2. Se crea un nuevo repositorio en GitHub

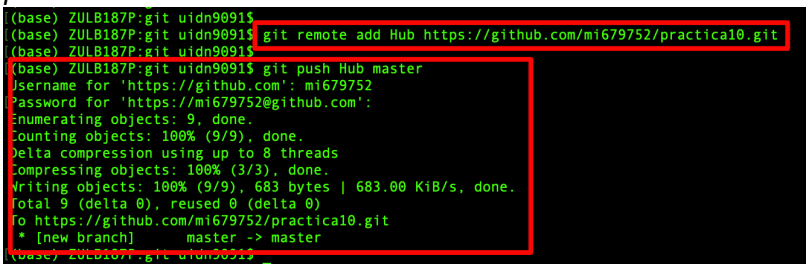


3. Se consulta y obtiene el URL del repositorio recién creado:

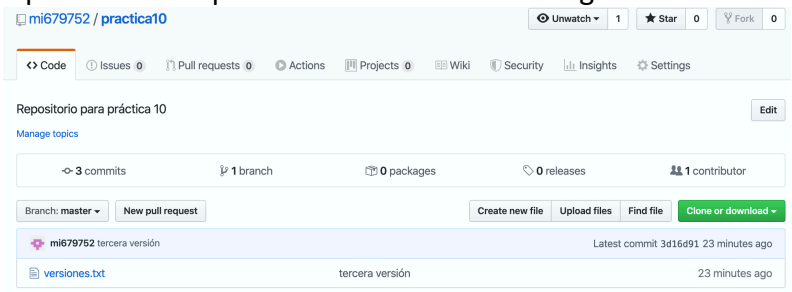


4. Se añade al repositorio local el repositorio de github recién creado, desde la carpeta creada(git), corriendo los siguientes comandos:
    a. *"git remote add Hub https://github.com/mi679752/practica10.git"*

b. *“git push Hub master”*



5. Se refresca la pantalla del repositorio en GitHub con el siguiente resultado



Configurar repositorio en AWS CodeCommit

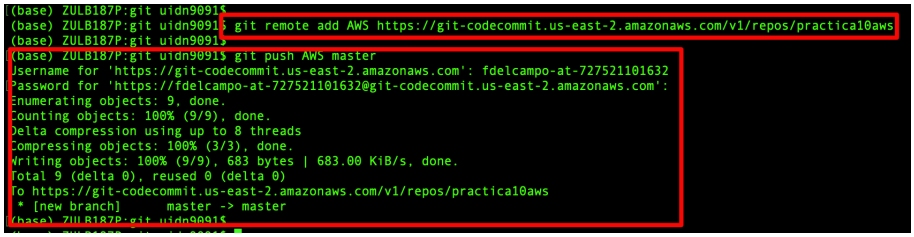
1. Se crean credenciales para CodeCommit, entrando a la consola de AWS, desde el dashboard de IAM:
   a. se selecciona el usuario deseado
   b. en “credenciales de seguridad”, se da clic en “generar las credenciales” en el apartado de “Credenciales de Git HTTPS para AWS CodeCommit”

2. Se accede al dashboard de “CodeCommit” y se crea un nuevo repositorio, dando clic en “Crear el repositorio”

3. Se consulta la URL del repositorio:
    a. Se da clic en “Clonar URL” y se selecciona “clonar HTTPS”, esto genera la URL la cual podemos utilizar:

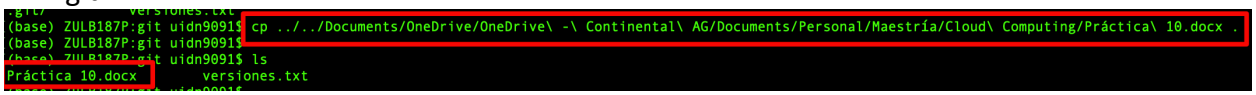
4. Se añade el repo recién creado en CodeCommit al repositorio local desde la carpeta creada para tal fin (git), corriendo los siguientes comandos:
    a. *“git remote add AWS codecommit::us-east-2://practica10aws”*
    b. *“Git push AWS master”*

    Se tienen los siguientes resultados:

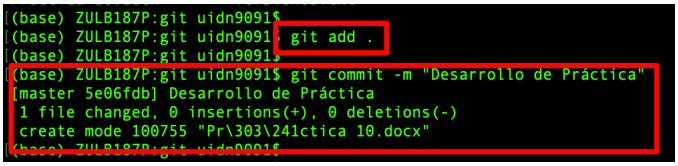


Controlar las versiones del entregable de la práctica 10

1. Se copia este documento inicial que cuenta solo con el desarrollo de la práctica al folder “git”



2. Se corren los comandos “add” y “commit” de cada avance para reflejar las versiones del documento



3. Se muestran los logs de cada avance, en este caso de la primera versión
    a. Desarrollo de práctica

```
(base) ZULB187P:git uidn9091$ git log
commit 5e06fdb8b8d47b8b2b901cfe23c96bcca4856db0 (HEAD -> master)
Author: mi679752 <mi679752@iteso.mx>
Date:   Wed Mar 25 21:12:07 2020 -0600

    Desarrollo de Práctica

commit 3d16d9171144969f228620fc0134aa16d617873c (Hub/master, AWS/master)
Author: mi679752 <mi679752@iteso.mx>
Date:   Wed Mar 25 20:17:51 2020 -0600

    tercera versión

commit 2c592c3c90964fa324caf10ef23b409ccae07475
Author: mi679752 <mi679752@iteso.mx>
Date:   Wed Mar 25 20:16:47 2020 -0600

    Segunda Versión

commit dc7abafc886fb9d26f65908be74822a06bc3d151
Author: mi679752 <mi679752@iteso.mx>
Date:   Wed Mar 25 20:13:21 2020 -0600

    Versión Inicial
(base) ZULB187P:git uidn9091$
```

b. Documento con formato

```
(base) ZULB187P:git uidn9091$ git log
commit ee0ab2a4ba3eb7a948afd0fbd892402e34457af9 (HEAD -> master, Hub/master, AWS/master)
Author: mi679752 <mi679752@iteso.mx>
Date:   Wed Mar 25 21:30:33 2020 -0600

    Documento con formato

commit 5672c927c6217ff54747da3dee3c44573a7b64c9
Author: mi679752 <mi679752@iteso.mx>
Date:   Wed Mar 25 21:20:33 2020 -0600

    Desarrollo de Práctica

commit 5e06fdb8b8d47b8b2b901cfe23c96bcca4856db0
Author: mi679752 <mi679752@iteso.mx>
Date:   Wed Mar 25 21:12:07 2020 -0600
```

c. Introducción y Marco Teórico

```
(base) ZULB187P:git uidn9091$ git log
commit 37cee633a3204ab50f2c866e42a20fcf6f82851b (HEAD -> master, Hub/master, AWS/master)
Author: mi679752 <mi679752@iteso.mx>
Date:   Wed Mar 25 21:49:33 2020 -0600

    Introducción y Marco Teórico

commit ee0ab2a4ba3eb7a948afd0fbd892402e34457af9
Author: mi679752 <mi679752@iteso.mx>
Date:   Wed Mar 25 21:30:33 2020 -0600

    Documento con formato
```

d. Conclusiones

```
(base) ZULB187P:git uidn9091$ git log
commit 7f71a417cc66ba93cb3924c0f5faca44381bb3eb (HEAD -> master, Hub/master, AWS/master)
Author: mi679752 <mi679752@iteso.mx>
Date:   Wed Mar 25 21:57:08 2020 -0600

    Conclusiones

commit 37cee633a3204ab50f2c866e42a20fcf6f82851b
Author: mi679752 <mi679752@iteso.mx>
Date:   Wed Mar 25 21:49:33 2020 -0600

    Introducción y Marco Teórico
```

e. Problemas, soluciones y bibliografía

```
(base) ZULB187P:git uidn9091$ git log
commit 0da79c7d1343d67e77dbbb59470ad61c96fec79c (HEAD -> master, Hub/master, AWS/master)
Author: mi679752 <mi679752@iteso.mx>
Date:   Wed Mar 25 22:04:24 2020 -0600

    Problemas, soluciones y Bibliografía

commit 7f71a417cc66ba93cb3924c0f5faca44381bb3eb
Author: mi679752 <mi679752@iteso.mx>
Date:   Wed Mar 25 21:57:08 2020 -0600

    Conclusiones
```

f. Diagrama de arquitectura

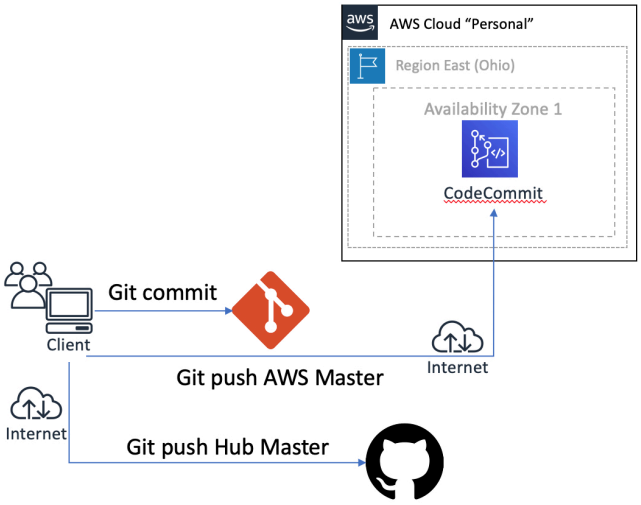
```
(base) ZULB187P:git uidn9091$ git log
commit 249bce27b71735d5a9d6fe35a1b941e579438bc6 (HEAD -> master, Hub/master, AWS/master)
Author: mi679752 <mi679752@iteso.mx>
Date:   Thu Mar 26 22:25:27 2020 -0600

    Arquitectura

commit 0da79c7d1343d67e77dbbb59470ad61c96fec79c
Author: mi679752 <mi679752@iteso.mx>
Date:   Wed Mar 25 22:04:24 2020 -0600

    Problemas, soluciones y Bibliografía

commit 7f71a417cc66ba93cb3924c0f5faca44381bb3eb
Author: mi679752 <mi679752@iteso.mx>
Date:   Wed Mar 25 21:57:08 2020 -0600

    Conclusiones
```

4. Se sube el reporte en formato PDF con el link del mismo en CodeCommit

Problemas y Soluciones

- Después de hacer los “commits” no veía nada en GitHub ni AWS, olvidé hacer los push respectivos
- No podía echar a andar el push del repositorio de CodeCommit, esto dado que no tenía el URL correcto, al darle “clone URL”->”https” me dio la URL del repo, la cual usé y con eso bastó.

Arquitectura



Conclusiones

Me pareció una práctica sumamente didáctica; había escuchado mucho de git pero nunca lo había usado. En mi equipo de trabajo lo usamos mucho pero yo no, dado que no desarrollo más. Me parece sumamente útil y respondiendo la pregunta en el cuaderno de prácticas, veo sumamente útil para control de versiones no solo de código fuente sino para otros tipos de documentos y tener siempre el control. Herramientas como sharepoint lo hacen posible, sin embargo no todas las empresas pueden pagar dicho licenciamiento.

Con respecto a la pregunta de las diferencias entre CodeCommit y GitHub, vi más completa y madura a esta última, ambas corren en la nube y no requieren que el usuario tenga infraestructura local para correrlas.

Bibliografía

[1] GitHub, «Sobre el Control de Versiones,» Git-SCM, 2020. [En línea]. Available: https://git-scm.com/book/es/v2/Inicio---Sobre-el-Control-de-Versiones-Acerca-del-Control-de-Versiones. [Último acceso: 2020].