cmpc

make it real | TSOFT

Bitbucket

Manual de Uso y Buenas Practicas

Migración a Bitbucket Cloud

Índice

Introducción

En este documento se deja registro del trabajo de implementación del proyecto de migración de los códigos fuentes de los sistemas SIM, CTAC y Reservador de Horarios de CMPC. También se documentará un manual de uso de Git, las buenas practicas del mismo, el flujo de trabajo recomendado propuesto por GitFlow y la administración de la herramienta de gestión de código Git de Atlassian, Bitbucket Cloud.

Definiciones de conceptos o abreviaturas

- **Git:** es una herramienta en la categoría Sistema de control de versiones de una pila tecnológica.
- **Master:** correspondiente al sitio productivo, Es creada por defecto el comando git init.
- **Hotfix:** Parche inmediato en producción.
- **BugFix:** Parche en ambiente QA
- **Release:** Entrega de una nueva versión.
- **Develop:** Desarrollo y correcciones para el próximo release.
- **Feature:** Nueva funcionalidad.

¿Qué es Git?

Git es un sistema de control de versiones distribuido, lo cual significa que su copia local del código es un repositorio de control de versiones completo.

El control de versiones es un sistema que registra los cambios realizados sobre un archivo o conjunto de archivos a lo largo del tiempo de tal manera que es posible recuperar versiones especificas más adelante. Los sistemas de control de versiones han ido evolucionando a lo largo del tiempo y podemos clasificarlos en tres tipos: Sistemas de Control de Versiones Locales, Centralizados y Distribuidos.

En los sistemas de control de versiones distribuidos como Git, se opta por darle a cada desarrollador una copia local de todo el proyecto, se esta manera se construye una red distribuida de repositorios, en la que cada desarrollador puede trabajar de manera aislada, teniendo un mecanismo de resolución de conflictos a la hora de unificar los cambios.

Al no existir un repositorio central, cada desarrollador puede trabajar a su propio ritmo, almacenar los cambios a nivel local y mezclarlos solo cuando se requiera, resolviendo los conflictos si los hubiera. Dado que cada usuario tiene una copia completa del proyecto, el riesgo por una caída del servidor, un repositorio dañado o cualquier otro tipo de perdida de datos es mucho menor que en cualquiera de los otros mecanismos de control de versiones.

¿Qué son las ramas?

En Git, las ramas son parte del proceso de desarrollo diario. Las ramas de Git son un puntero eficaz para las instantáneas de tus cambios. Cuando quieres añadir una nueva función o solucionar un error, independientemente de su tamaño, generas una nueva rama para alojar estos cambios. Esto hace que resulte más complicado que el código inestable se fusione con el código base principal, y te da la oportunidad de limpiar tu historial futuro antes de fusionarlo con la rama principal.

En definitiva, una rama representa una línea independiente de desarrollo. Las ramas sirven como una abstracción de los procesos de cambio, preparación y confirmación. Puedes concebirlas como una forma de solicitar un nuevo directorio de trabajo, un nuevo entorno de ensayo o un nuevo historial de proyecto. Las nuevas confirmaciones se registran en el historial de la rama actual, lo que crea una bifurcación en el historial del proyecto.

Características de Git

- Cada desarrollador tiene una copia completa del proyecto.
- Cada desarrollador trabaja con un repositorio local.
- Crear ramas y mezclarlas es rápido y poco propenso a problemas.
- El concepto de staging area permite seleccionar que cambios versionar según sea conveniente.
- Los desarrolladores pueden tener control de versión de su propia rama, mientras que la rama principal está controlada por el propietario del repositorio.
- Proporciona una mejor auditoría de código.
- Permite flujos de trabajo muy flexibles.

Marco de trabajo

Como se mencionó anteriormente para este proyecto se utilizará Bitbucket Cloud, la herramienta de gestión de código Git de Atlassian. Bitbucket no solo implementa Git, sino que también permite la administrar y organizar los repositorios de Git en proyectos.
Los repositorios contienen el código fuente de los sistemas que se desean versionar, pero el versionamiento de un sistema complejo puede que abarque más de un repositorio. Esto sucede debido a que los repositorios están orientados a versionar el código fuente de un único aplicativo del sistema, es decir que no es apropiado versionar fuentes que se corresponden con más de una tecnología en un mismo repositorio.
Bitbucket Cloud permite agrupar un conjunto de repositorios en proyectos y a su vez los proyectos están contenidos dentro de un área de trabajo o workspace. Para las incumbencias de esta colaboración y teniendo en cuenta las buenas practicas recomendadas, TSoft propone una organización en la cual se implementa un workspace principal en donde se encontrarán los sistemas de CMPC, en donde cada sistema a versionar está representado por la figura de un proyecto de Bitbucket y dentro de los mismos se encuentran repositorios para cada uno de los aplicativos de esos sistemas.

Comandos de Git

A continuación, se listan los comandos más comúnmente utilizados para la creación, manipulación y actualización de repositorios.

Para un repositorio nuevo, crea un directorio nuevo, Abrir y ejecutar, para crear un nuevo repositorio de git.

→ **`git init`**

Crea una copia local del repositorio

→ **`git clone /path/to/repository`**

Servidor remoto, ejecutar

→ **`git clone username@host:/path/to/repository`**

Registrar cambios (añadirlos al Index)

→ **`git add <filename>`**

→ **`git add .`**

** Este es el primer paso en el flujo de trabajo básico.

Hacer commit a estos cambios

→ **`git commit -m "Commit message"`**

**Ahora el archivo este incluido en el HEAD, pero aún no en tu repositorio remoto.

Envío de cambios
Tus cambios están ahora en el HEAD de tu copia local. Para enviar estos cambios a tu repositorio remoto ejecuta.

→ **`git push origin master`**

** Reemplaza master por la rama a la que quieres enviar tus cambios.

Clonar un repositorio ya existente y conectar tu repositorio local a un repositorio remoto.

→ **`git remote add origin <server>`**

Subir cambios al repositorio remoto seleccionado ramas

Crear una nueva rama llamada "feature_x" y cámbiate a ella usando:

→ **git checkout -b feature_x**

Volver a la rama principal

→ **git checkout master**

Borrar rama

→ **git branch -d feature_x**

Una rama nueva no estará disponible para los demás a menos que subas (push) la rama a tu repositorio remoto.

→ **git push origin <branch>**

Actualiza & fusiona
Para actualizar tu repositorio local al commit más nuevo, ejecuta en tu directorio de trabajo para bajar y fusionar los cambios remotos.

→ **git pull**

Para fusionar otra rama a tu rama activa (por ejemplo, master), utiliza:

→ **git merge <branch>**

**En ambos casos git intentará fusionar automáticamente los cambios. Desafortunadamente, no siempre será posible y se podrán producir conflictos. Tú eres responsable de fusionar esos conflictos manualmente al editar los archivos mostrados por git. Después de modificarlos, necesitas marcarlos como fusionados.

→ **git add <filename>**

Revisar antes de fusionar los cambios

→ **git diff <source_branch> <target_branch>**

En caso de que hagas algo mal (lo que seguramente nunca suceda ;) puedes reemplazar cambios locales usando el comando

→ **git checkout -- <filename>**

Este comando reemplaza los cambios en tu directorio de trabajo con el último contenido de HEAD. Los cambios que ya han sido agregados al Index, así como también los nuevos archivos, se mantendrán sin cambio.

Por otro lado, si quieres deshacer todos los cambios locales y commits, puedes traer la última versión del servidor y apuntar a tu copia local principal de esta forma

→ **git fetch origin**
→ **git reset --hard origin/master**

Git vs Bitbucket

Como ya se mencionó anteriormente Git es un sistema de control de versiones distribuido, el cual permite administrar nuestros repositorios, tanto locales como remotos. Esto se realiza a través de los comandos que se presentan en la sección anterior. Git es un proyecto de código abierto maduro y mantenido activamente. Una asombrosa cantidad de proyectos de software dependen de Git para el control de versiones, incluidos proyectos comerciales y de código abierto.

Existen varias alternativas a la hora de elegir un proveedor de Git, dentro de las más conocidas encontramos a GitHub, GitLab y en este caso Bitbucket. Estos proveedores son quienes nos brindan el servicio de infraestructura en la nube para alojar nuestros repositorios remotos, así como también una gran variedad de utilidades adicionales que nos facilitan la utilización de Git. En todas estas soluciones vamos a poder visualizar, a través de la web, los cambios realizados y administrar tanto nuestros repositorios como la organización de los mismos.

En esencia Git es una herramienta de control de versiones distribuida a partir de la cual, mediante comandos (universales a todos los proveedores), podemos gestionar tanto nuestro repositorio local como el remoto. En este proyecto, nuestros repositorios remotos van a estar alojados en Bitbucket Cloud y en la sección de *Administración de Bitbucket* de este documento es explicaran las funcionales adicionales provistas por la herramienta Bitbucket para la administración de los repositorios remotos.

Cabe destacar que el núcleo de funcionalidades de Git se encuentra plasmando en los comandos que se mencionaron en la sección anterior, los mismos son open source y universales a cualquier proveedor de Git. Para poder utilizar estos comandos basta con tener instalado el cliente de Git en el equipo. El mismo debe ser instalado en el equipo de cualquier administrador/desarrollador que desee interactuar con el repositorio y nos permitirá trabajar con repositorios remotos provistos por cualquier proveedor.

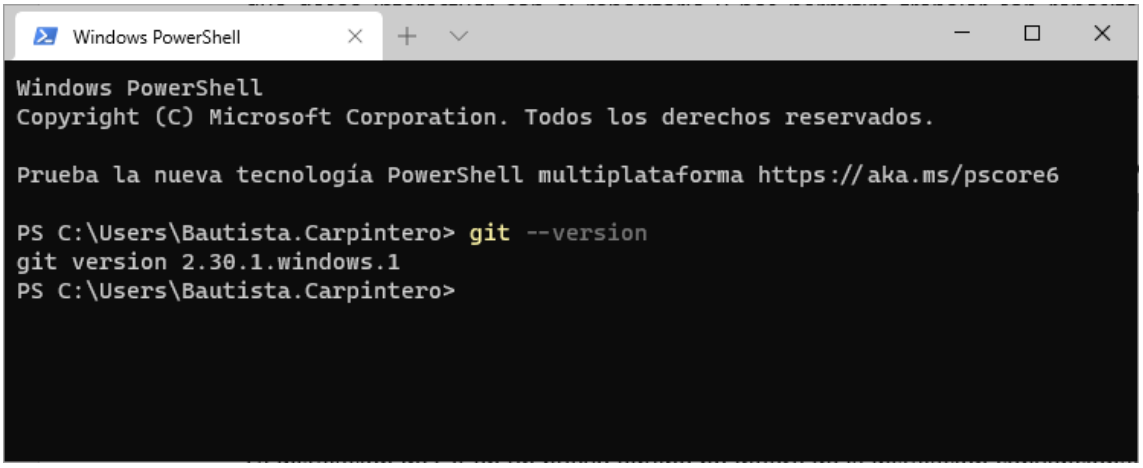
Nota: Si solo se desea descargar el código fuente alojado en el repositorio, esto se puede hacer desde la web de Bitbucket, es decir que nuestro código no está allí encerrado. Pero esto no se recomienda ya que se debe seguir el flujo de trabajo GitFlow y obtener una copia del repositorio remoto mediante el comando **`git clone`**, y de esta forma tener el proyecto listo de forma local para versionar futuros cambios.

Instalación de Git

La instalación de Git en un nuevo equipo no difiere de la instalación convencional de cualquier otro software. Simplemente debemos ingresar a la web oficial de Git y descargar la versión apropiada según nuestro sistema operativo:

https://git-scm.com/downloads

Una vez instalado Git se puede comprobar su instalación y su correcto funcionamiento ejecutando el comando **`git --version`**.



Teniendo Git instalado hay que realizar una primera configuración inicial en la que debemos cargar nuestro nombre y nuestro email vinculado a nuestra cuenta de, en este caso, Bitbucket.

- → **`git config --global user.name "Emma Paris"`**
- → **`git config --global user.email "eparis@atlassian.com"`**

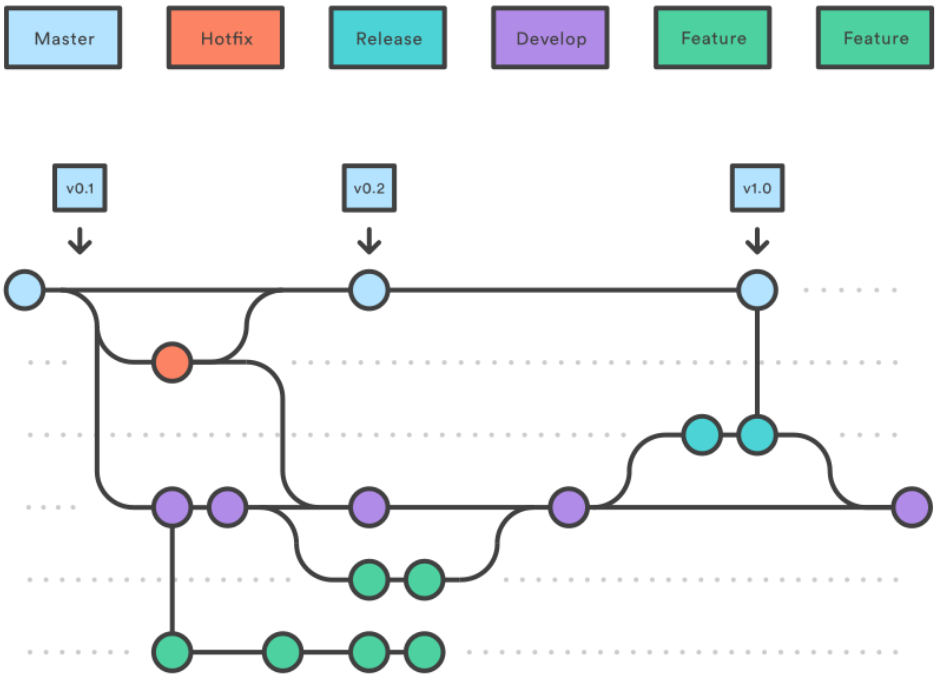
En caso de no realizarse esta configuración Git va a solicitar que la realice cuando intente hacer su primer commit.

Flujo de trabajo GitFlow

Gitflow es un popular flujo de trabajo de Git y sugiere trabajar con las siguientes ramas para asegurar la calidad y eficiencia de salidas a producción de los productos y servicios.

Las ramas son utilizadas para desarrollar funcionalidades aisladas unas de otras. La rama master es la rama "por defecto" cuando creas un repositorio. Crea nuevas ramas durante el desarrollo y fusiónalas a la rama principal cuando termines.

Nombre Rama	Descripción
Master	Correspondiente al sitio productivo, solo publique cuando el release esté listo. Es creada por efecto el comando (git init).
Hotfix	Crear cuando un parche inmediato es necesario en producción.
Release	Entrega de una nueva versión. Se utiliza para preparar y probar antes de publicar en producción.
Develop	Desarrollo y correcciones para el próximo release.
Feature	Se trabaja sobre una nueva funcionalidad fuera de la rama de desarrollo.



Reglas de uso de Git e implementación de GitFlow

En esta sección se presentarán las buenas prácticas para la utilización de Git y de las ramas sugeridas según GitFlow. Como las ramas interactúan en función a los casos de uso cotidianos durante el desarrollo de un proyecto.

Buenas Prácticas de Git

- La nomenclatura para creación de ramas **feature** y **hotfix** debe ser con el **número de ticket** que gatilla la modificación o la resolución de un incidente productivo.
- Las ramas **feature** deben ser creadas a partir de **develop**.
- Sólo integrar en **master** cuando se planifiqué un pase a producción, de lo contrario el código debe estar en **develop**.
- La rama release no admite nuevas funcionalidades, solo corrección de errores.
- Para un hotfix clonar desde **master** y cuando se resuelva el incidente hacer merge de código en **develop** y **master**.
- Definir periodicidad de **push** y responsabilidad de integración de los cambios por parte de la célula.
- La mezcla de código en las ramas principales del proyecto se debe efectuar a través de un **pull request**.
- En los **commits** dividir el comentario en dos partes principales para tener un asunto y un mensaje
- En el asunto del commit añadir palabras claves como ADD, UPDATED, FIXED o DELETED.
- En el mensaje del commit agregar descripciones en inglés por simplicidad del lenguaje.
- El proyecto debe tener un archivo **README.md** en la raíz que cumpla con las siguientes condiciones:
    - Descripción funcional del proyecto.
    - Tecnologías utilizadas en el proyecto.
    - Dependencias del proyecto.
    - Instrucciones para compilar.
    - Instrucciones para desplegar.
    - Otras consideraciones de uso.

Reglas para la utilización de las ramas

La rama master es una rama que está estrechamente controlada y a la cual solo debe llegar código estable correspondiente a producción.

La branch develop es una rama utilizada por todos para construir ramas features, en donde se implementan las nuevas funcionalidades o cambios. Pero no es una rama sobre la cual se deba trabajar.

Ciclo de vida de una Feature Branch

Las features branches, o ramas de implementación de características/funcionalidades, surgen de develop y siempre deben ser unidas a develop. Estas son las ramas que son utilizadas por los desarrolladores para trabajar y realizar sus commit's y push's según ellos requieran.

Los estados que atraviesa una rama feature durante su ciclo de vida son los siguientes:

- Crear una rama feature desde develop con el número de ticket como nombre y se realizan los cambios.
- Hacer fetch regularmente para revisar si hay nuevos cambios en develop que deba incorporar.
    - En caso que los haya realizar git pull para traerlos
- Una vez implementados los cambios realizar un pull request para unificarlos con develop.
- Hacer el merge con develop.

Ciclo de vida de una Release Branch

La rama release es una rama que sale desde develop con el objetivo de lanzar nuevas features a master. Una rama release representa un conjunto nuevo de características a incorporar en producción y debe tener un tiempo de vida corto.

El ciclo de vida de una rama release es el siguiente:

- Se crea una rama release desde develop
- Se corrigen los posibles errores que haya llegado a esta rama.
- Se solicita un pull request (solicitud de unión) desde esta rama de release hacia **master** y también hacia **develop** en caso de que se hayan realizado correcciones.

Ciclo de vida de una Hotfix Branch

Estas ramas surgen para reparar rápidamente errores que hayan llegado a producción, es decir la rama master. Estas ramas son muy similares a las ramas de release con la diferencia de que estas se originan desde master.

Los estados del ciclo de vida de una rama de Hotfix son los siguientes:

- Se genera un incidente productivo y lo toma un desarrollador.
- Este genera una rama Hotfix desde master con el número de ticket.
- El desarrollador resuelve el incidente.
- Se deben publicar los cambios con el arreglo del incidente tanto en la rama master como en develop. Para esto se realizar un pull request a master y también a develop.

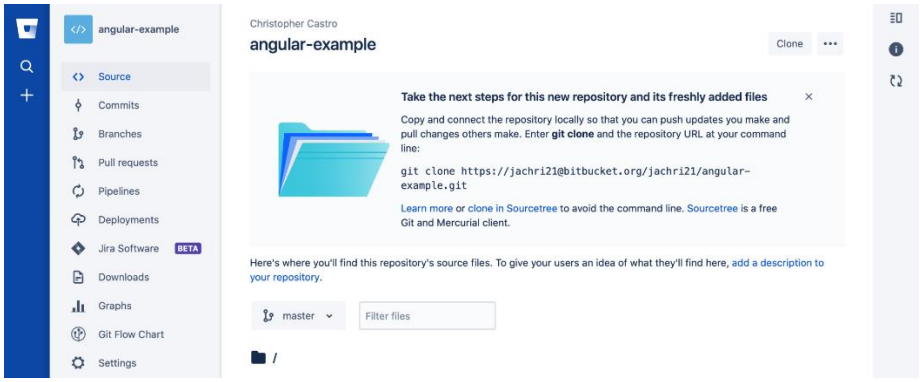
Administración de Bitbucket

En este apartado se expondrán los mecanismos de administración de Bitbucket Cloud para la gestión de roles grupos, buenas prácticas en el uso y manipulación de la herramienta.

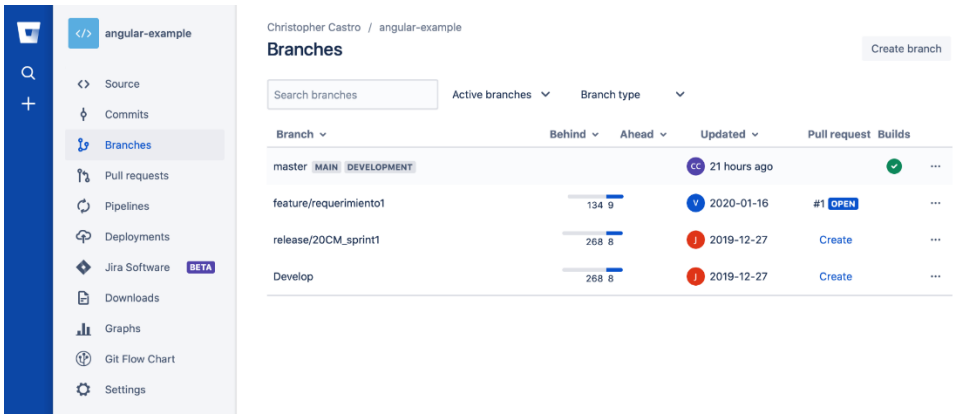
Vista previa

En principio vamos a tomar el ejemplo de un proyecto en angular para ver como Bitbucket Cloud presenta en su interfaz los elementos de Git que se mencionaron en apartados anteriores.

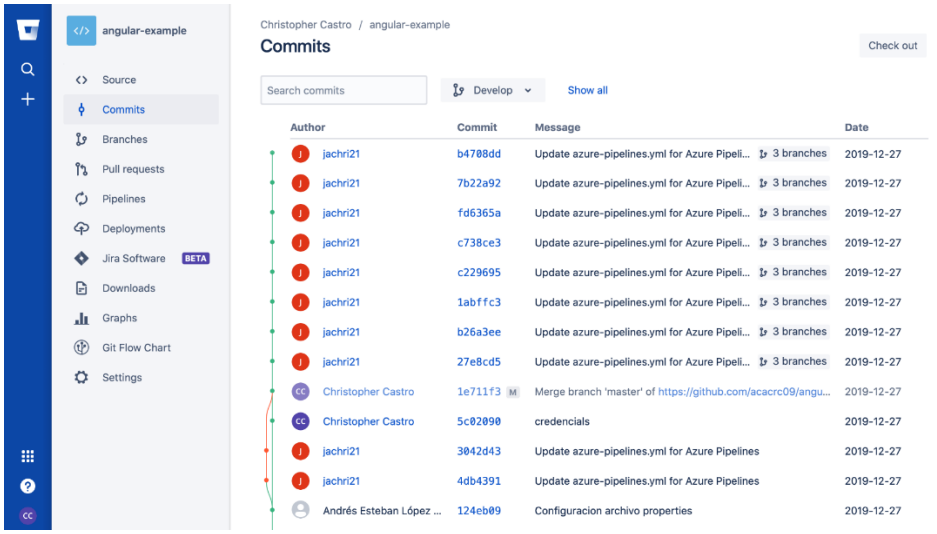
Repositorios



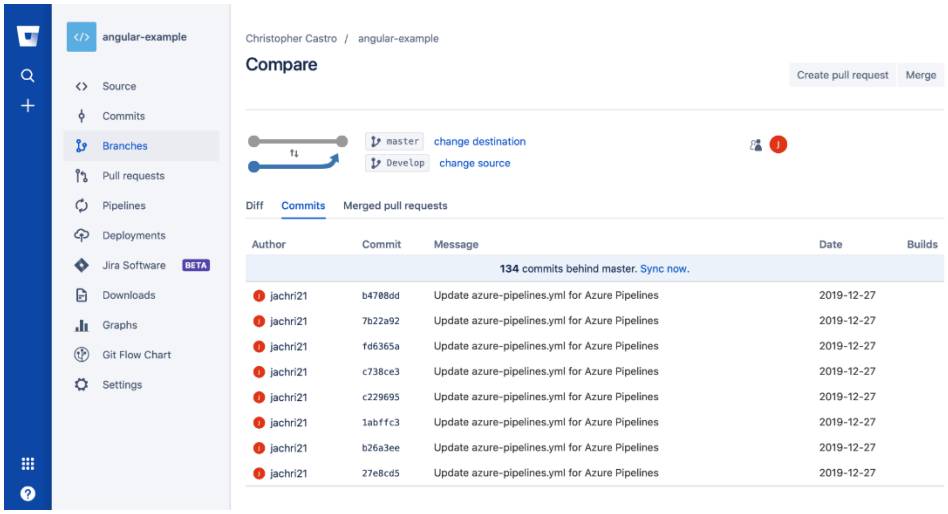
Ramas



Commits

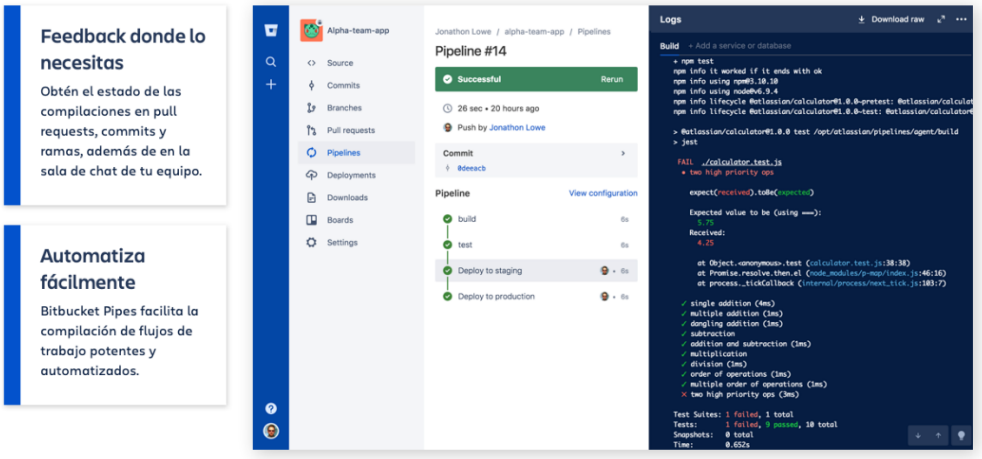


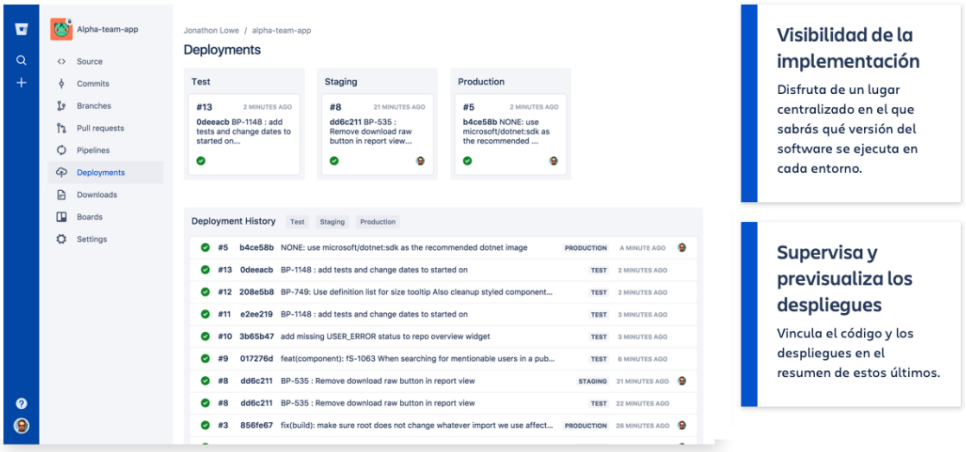
Comparaciones entre ramas



Pipeline

Bitbucket pipelines es un servicio de integración y despliegue continuo, una combinación de integración y entrega continuas para Bitbucket Cloud extremadamente sencilla de configurar, con la que automatizarás el código desde la fase de pruebas hasta la de producción.





Administración de accesos y grupos

Equipos de trabajo: Los equipos de Bitbucket Cloud le permiten administrar múltiples proyectos, repositorios y colaborar con los usuarios. Existen diferentes componentes para un equipo de trabajo, usuarios, grupos y repositorios. La forma en que trabajan los equipos es la siguiente: Todos los equipos tienen grupos de usuarios, los grupos especifican permisos predeterminados para los repositorios de sus equipos y cada equipo posee su repositorio. Se suma la virtud de agregar o invitar usuarios a los grupos del equipo, dándoles permisos de acceso para repositorios.

Grupos de usuarios y configuraciones de acceso: Cada equipo tiene dos grupos de usuarios por defecto, un grupo de usuarios Administradores y un grupo de usuarios Desarrolladores sobre grupos de usuarios. Los grupos del equipo se agregan al repositorio con permisos predeterminados, pudiendo cambiar el acceso de un grupo en cualquier momento.

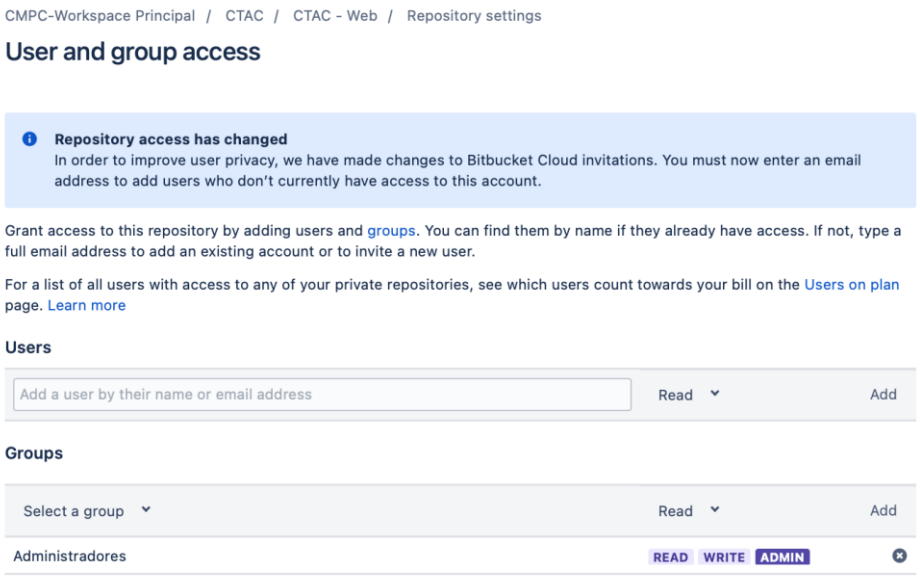
Permisos de Usuarios: Un grupo de usuarios tiene dos tipos diferentes de permisos: acceso predeterminado al repositorio y permisos de actualización del equipo.

Niveles de Acceso	Descripción
Admin	Permite cambiar la configuración del repositorio, cambiar los permisos del usuario y eliminar el repositorio
Write	Permite a los usuarios contribuir al repositorio empujando los cambios directamente
Read	Permite a los usuarios ver, clonar y bifurcar el código del repositorio, pero no enviar los cambios. Permite crear Issue, comentar Issue y editar páginas wiki.

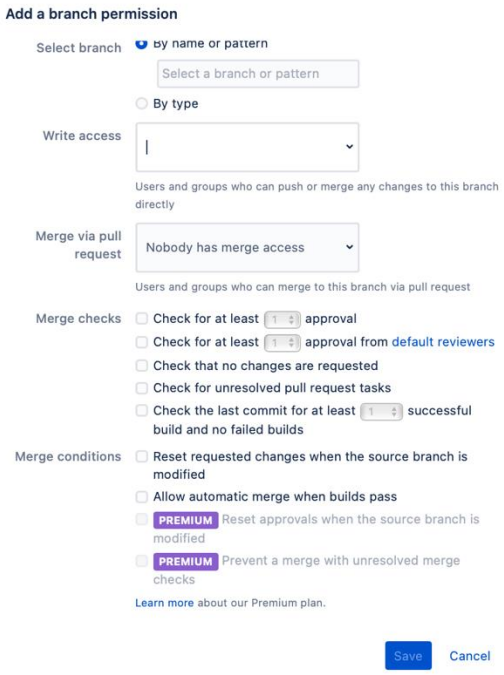
Permisos de grupo: Estos permisos especifican lo que los usuarios del grupo pueden hacer con su cuenta de equipo.

Niveles de Acceso	Descripción
Crear repositorios	Permite a los miembros del grupo crear nuevos repositorios (para el equipo).
Administrar grupo	Permite a los miembros el equipo actualizar la configuración del grupo y la configuración de cualquier repositorio propiedad del grupo.

Permisos de Repositorio: Cada repositorio permite el acceso a usuarios y grupos de forma independiente. Para acceder a esta opción se debe ingresar a la configuración del repositorio en la pestaña User and Group Access.



Permisos de Branch: Estos permisos permiten controlar el empuje (push) y mezcla (merge) de cambios directamente a los usuarios o grupos que se especifiquen. Al seleccionar el Branch puede ser especificado por nombre o por tipo.



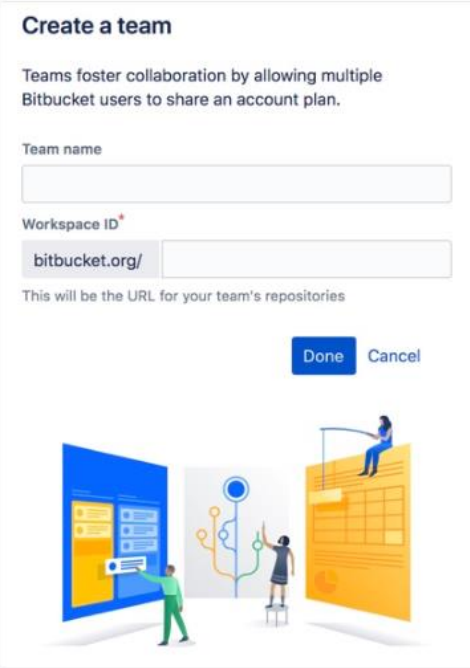
Crear y administrar un equipo

Para crear o unirse a un equipo, es necesario contar con una cuenta individual de Bitbucket. El responsable de crear el equipo automáticamente tiene acceso administrativo. Este usuario también puede especificar el acceso de administrador para otros miembros del equipo, dándoles la capacidad de administrador. Cualquier administrador puede actualizar el plan del equipo y los detalles de la tarjeta de crédito.

Crear un equipo Admin / Developers

Cualquier usuario puede crear un nuevo equipo y se convertirá automáticamente en miembro administrador del equipo.

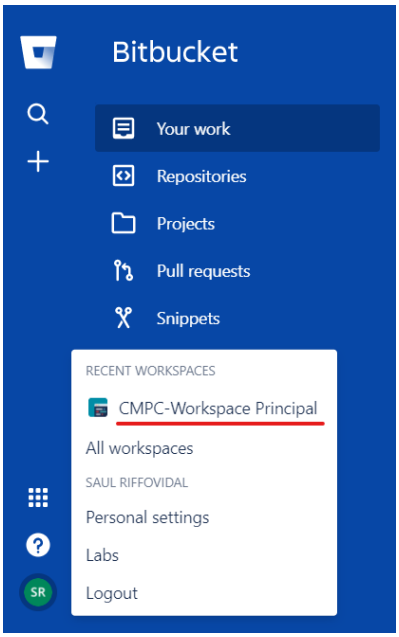
- Haga clic en + en la barra lateral global y seleccione Equipo en Crear un nuevo.
- Ingrese un nombre de equipo. Este nombre es lo que ves en la navegación de los equipos en Bitbucket y lo que ven tus compañeros de equipo en los correos electrónicos de invitaciones.
- Ingrese una ID de espacio de trabajo. Su ID no puede tener espacios ni caracteres especiales, pero los números y las letras mayúsculas están bien. Este ID se convierte en parte de la URL del equipo y en cualquier otro lugar donde haya una etiqueta que identifique al equipo (API, grupos de permisos, OAuth, etc.)

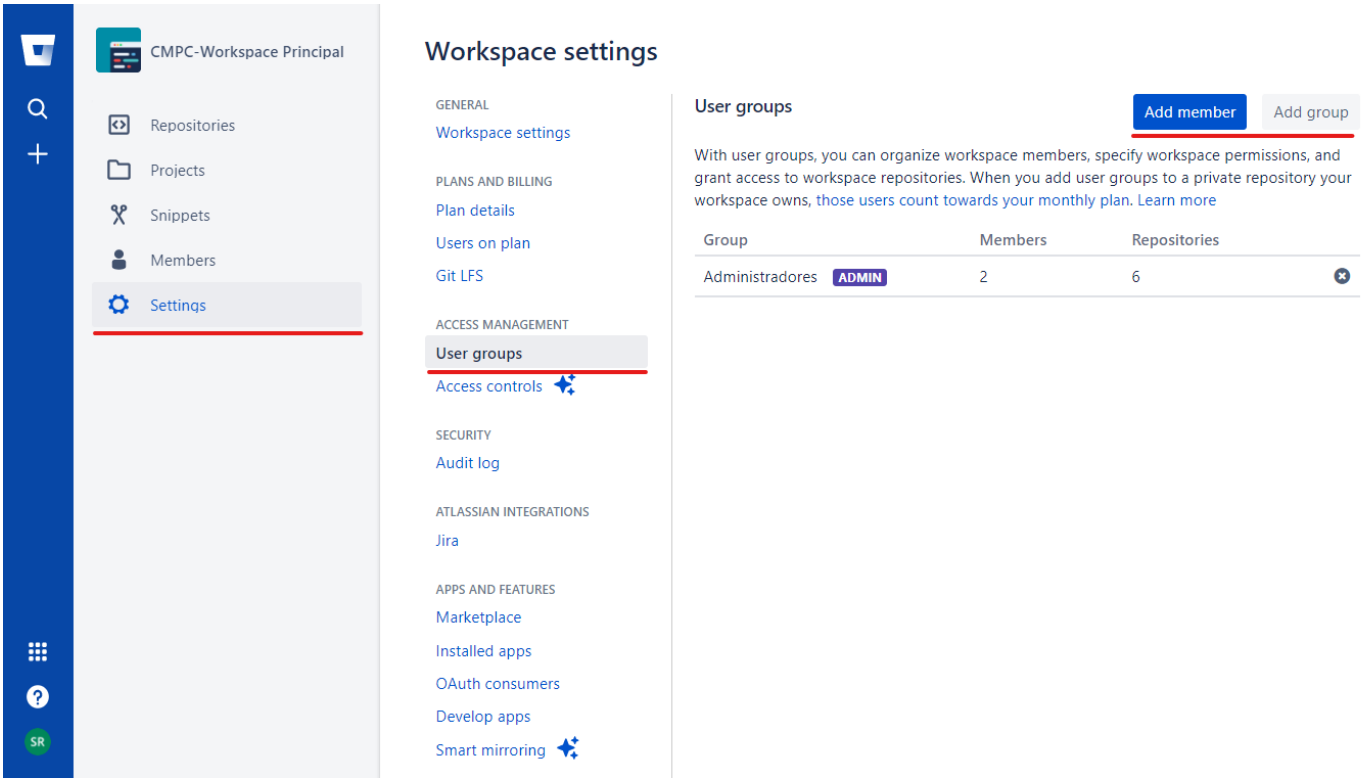


Agrega miembros a tu equipo

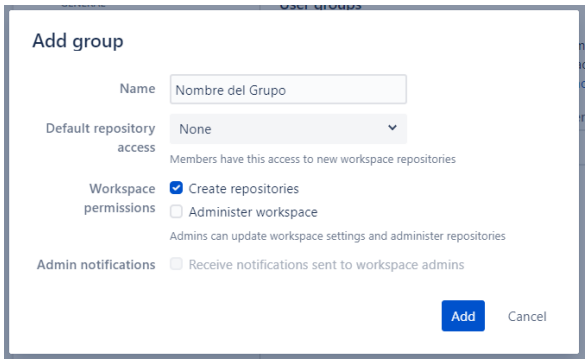
Una vez que tenga un equipo, puede comenzar a agregar otros usuarios como miembros.

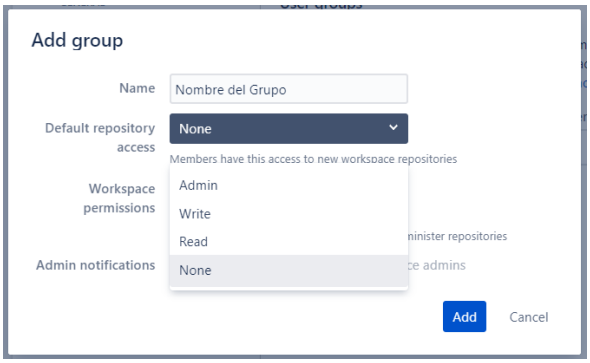
- Desde su avatar en la parte inferior izquierda, seleccione su equipo o haga clic en Ver todos los equipos para obtener una lista completa.
- Haga click en el nombre del workspace que desea configurar y en la siguente pantalla entre a la pestaña de configuraciones (Settings).
- Dentro de la seccion de configuraciones va a encontrar la opcion de grupos de usuarios contenida dentro de 'Access Management'.
- Alli podra agregar miembros a su grupo o equipo e inclusive crear nuevos grupos de usuarios.





Nuevos grupos de usuarios: Cuando crea un grupo, especifica el tipo de permisos que desea otorgar a los miembros del grupo para el equipo y para los repositorios recién creados.





Desde la página Grupos de usuarios, haga clic en Agregar grupo. Actualice los campos en la pantalla y haga clic en Agregar para guardar el grupo.

Niveles de Acceso	Descripción
Nombre	Describe el tipo de usuarios que incluirá este grupo.
Acceso al repositorio predeterminado	Especifica el acceso predeterminado del grupo cuando el equipo tiene un nuevo repositorio: Administrador: permite a los usuarios hacer todo lo que un propietario del repositorio puede hacer: cambiar la configuración del repositorio, actualizar los permisos del usuario y eliminar el repositorio. Escritura: permite a los usuarios contribuir al repositorio mediante la inserción de cambios directamente. Lectura: permite a los usuarios ver, clonar y bifurcar el código del repositorio, pero no enviar los cambios. El acceso de lectura también permite a los usuarios crear problemas, comentar problemas y editar páginas wiki. None: evita que esos usuarios vean algo en el repositorio.
Permisos del equipo Especifica lo que los usuarios del grupo pueden hacer con la cuenta del equipo:	Crear repositorios: permite a los miembros del equipo crear nuevos repositorios para el equipo. Administrar equipo: permite a los miembros del equipo actualizar la configuración del equipo y la configuración de cualquier repositorio propiedad del equipo.
Notificaciones de administrador (Si este grupo tiene acceso de administrador)	Especifica si los usuarios del grupo recibirán notificaciones del equipo. Solo podrá seleccionar esta opción si también seleccionó el permiso Administrar equipo.

Cuando hace clic en Agregar, se abre la página de ese grupo para que pueda agregar usuarios. Ingrese el nombre del usuario o la dirección de correo electrónico y haga clic en Agregar.

Si ingresa la dirección de correo electrónico de un no usuario, esa persona recibe una invitación para crear una cuenta Atlassian y unirse al equipo.

Después de crear un grupo, haga clic en Editar para actualizar los permisos del grupo.

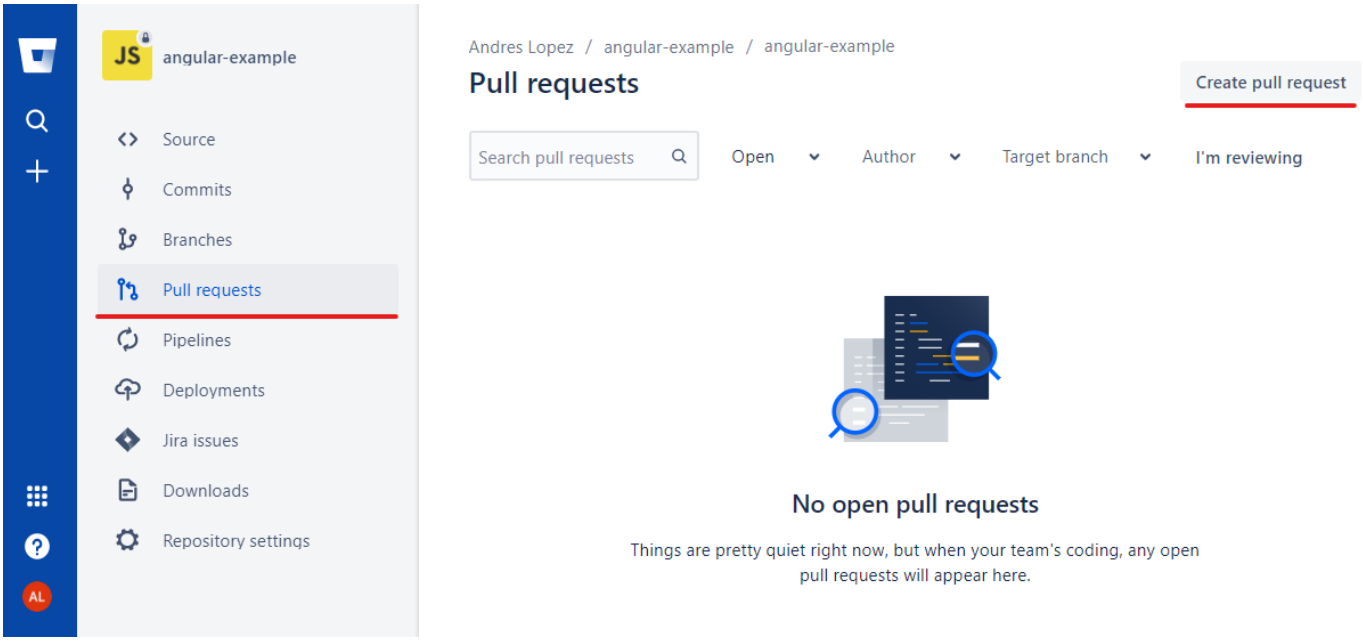
***Nota**: distinguir entre el repositorio y los permisos de equipo puede permitir a los usuarios crear repositorios para el equipo sin darles ningún tipo de permisos de administrador. Para hacer esto, configure los permisos del Equipo para Crear repositorios mientras su acceso al repositorio predeterminado está configurado en Escritura. Los usuarios de este grupo no pueden administrar la cuenta del equipo, pero pueden crear repositorios para el equipo.*

Proceso de aceptación de cambios

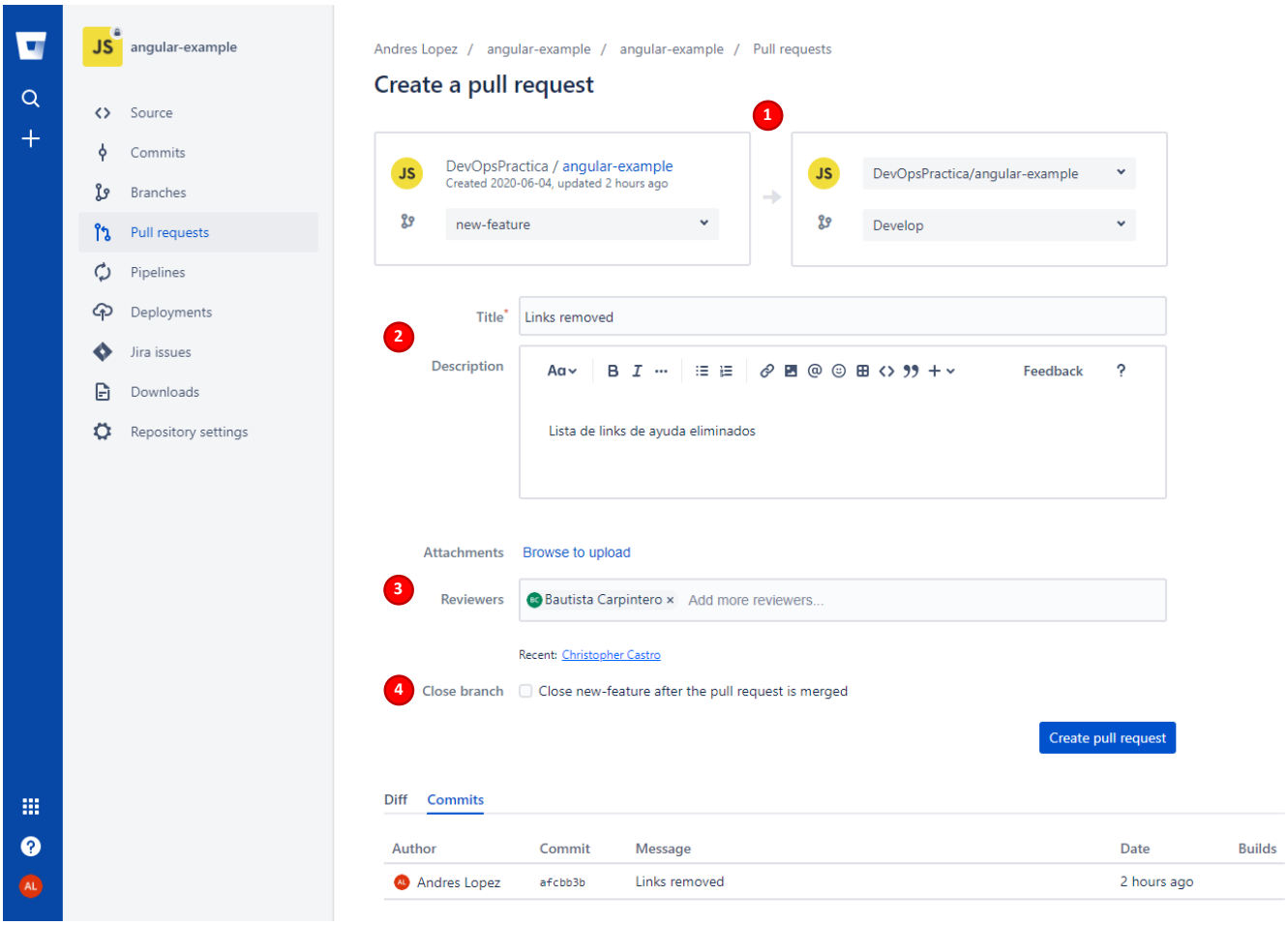
Cuando un desarrollador realiza la implementación de una nueva feature o funcionalidad debe hacerlo sobre una nueva rama a partir de develop. Una vez finalizada la implementación el desarrollador debe incorporar los cambios que haya realizado sobre su rama en la rama develop. Para esto se debe iniciar un proceso llamado Pull Request en el cual se solicita permiso para realizar el merge (unificación) de la rama actual con la rama de destino deseada, en este caso de las ramas **new-feature** a **develop**.

Este pull request será realizado por el desarrollador y revisado por un administrador, el cual puede ver commit por commit y archivo por archivo que líneas de código cambiaron, agregar comentarios y solicitar cambios, en caso que se requieran, antes de aprobar la unificación del nuevo desarrollo.

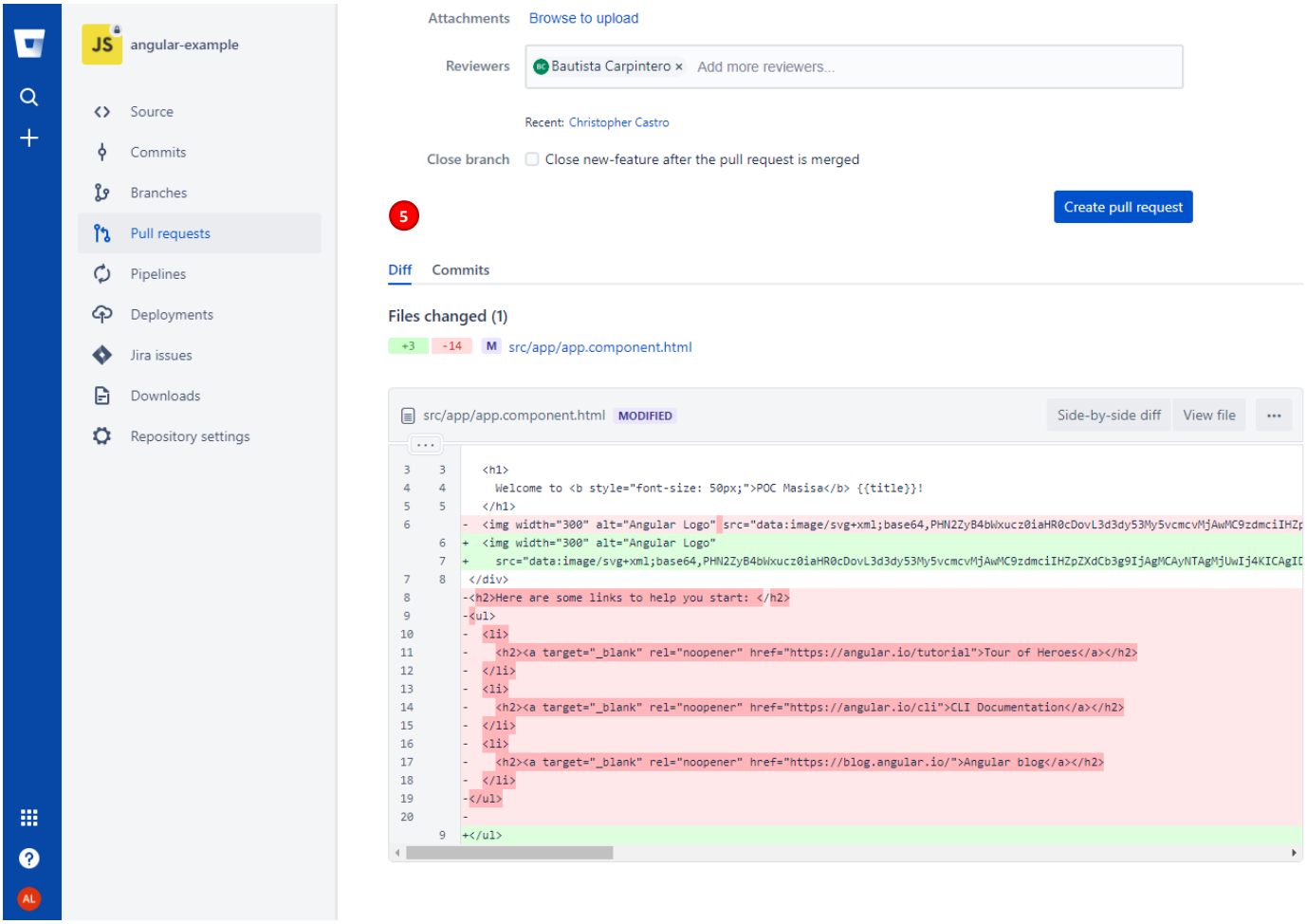
Para solicitar un pull request es necesario ir al repositorio en Bitbucket. Entrar a la sección de "Pull request", y luego hacer click en el botón "Create pull request".



En la pantalla de creación del pull request se dan las opciones para seleccionar desde que rama se quiere hacer la unión y hacia que rama se la realiza (1). Se agrega un título y descripción para el pull request (2). También se debe especificar quien va a ser el responsable de revisar y autorizar el pull request (3). La lista de los posibles administradores responsables, **reviewers**, se encuentra dentro de las configuraciones del repositorio y allí se pueden agregar y eliminar reviewers al proyecto. También se puede configurar, si así se lo desea, la eliminación automática de la rama sobre la cual se está trabajando en la feature (4), luego de que se apruebe y se concrete el merge o unión.

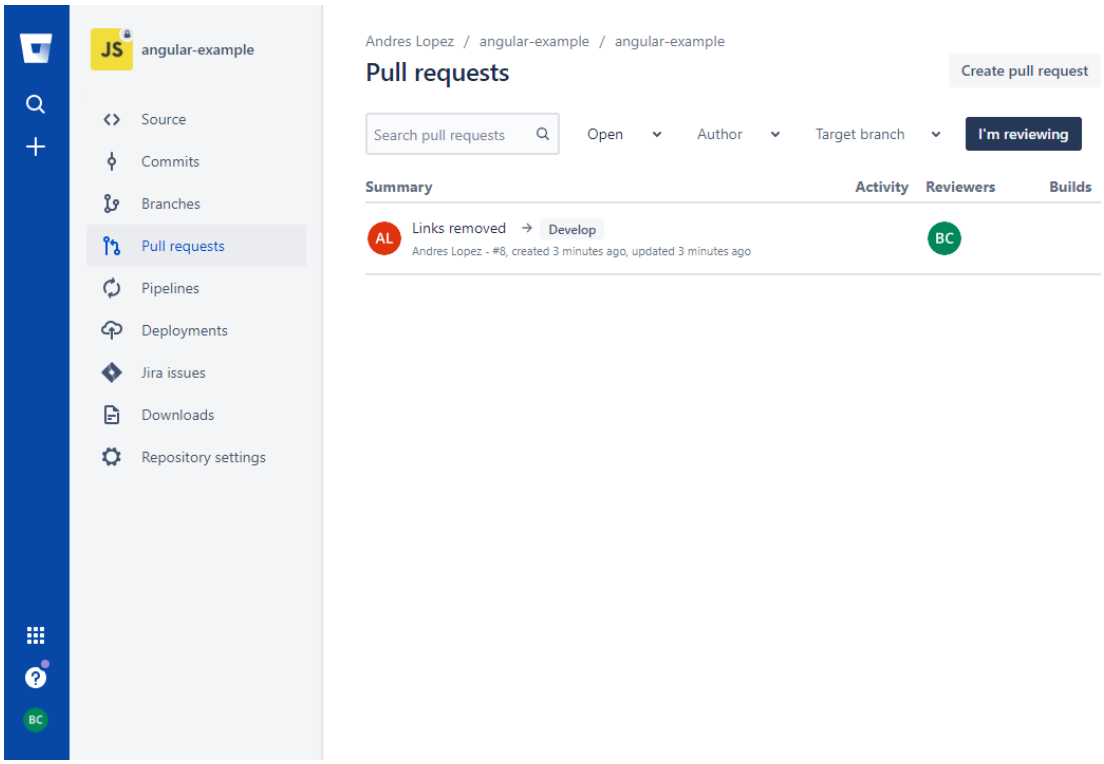


En la imagen que se presenta a continuación se puede observar que al momento de crear un pull request, también se puede visualizar los cambios realizados, mediante la opción “Diff” (5). Aquí se muestra la diferencia, línea a línea, entre el código que está en la rama new-feature y el contenido en develop. En este ejemplo en particular solo se habían realizado cambios en un único archivo, pero en caso de que varios archivos hubiesen sufrido cambios, se disponen de las opciones para navegar entre los archivos viendo las modificaciones. Las líneas rojas son las líneas de código eliminadas y las verdes son las líneas nuevas o modificadas.



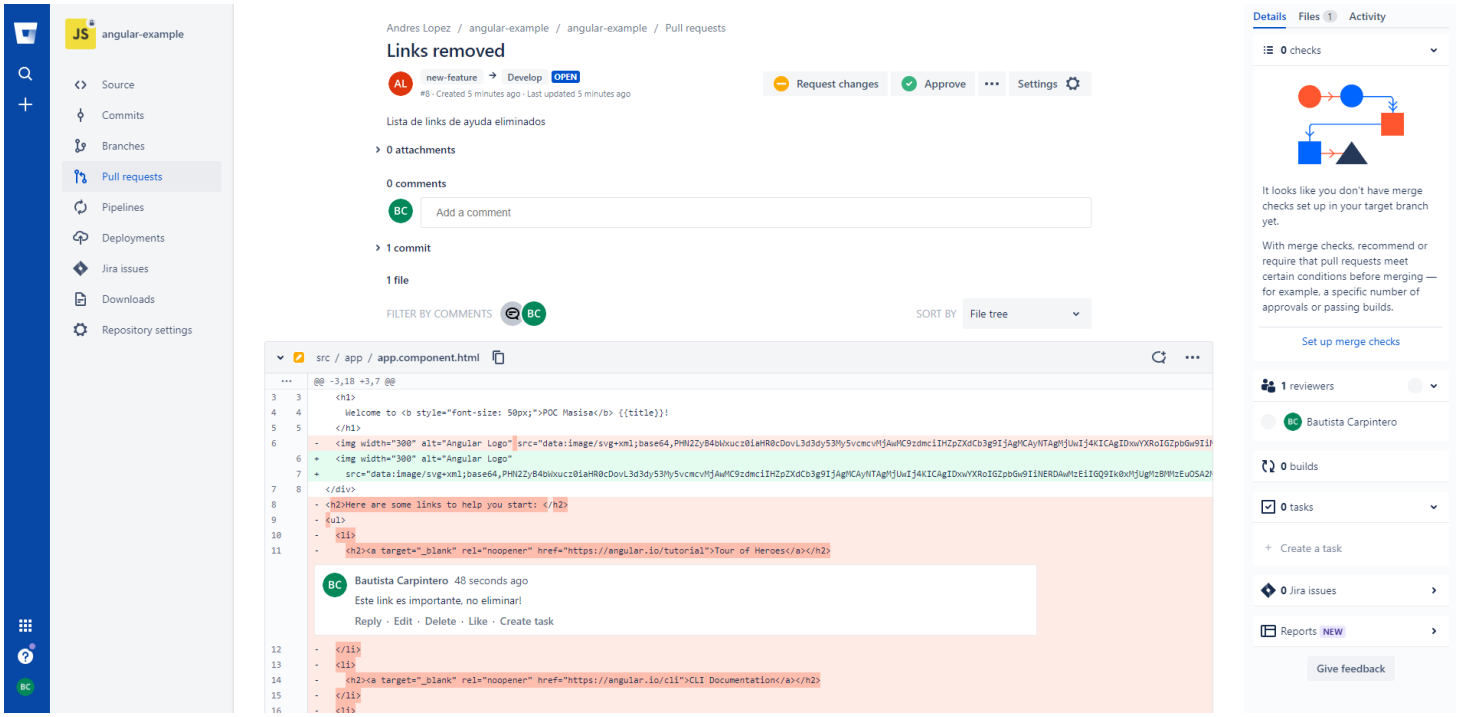
Finalmente, solo hay que hacer click en el botón de “Create pull request” y la solicitud será enviada y el/los revisores o reviewers serán notificados mediante una notificación en la herramienta y también vía mail.

Desde el punto de vista del reviewer, simplemente debemos ingresar a la sección de “Pull requests” del repositorio y veremos la lista de peticiones para revisar y autorizar. Simplemente entramos dentro de la solicitud para revisarla.



Cuando se ingrese a la solicitud el reviewer podrá ver el detalle completo de los cambios realizados, podrá agregar comentarios en el código, específicamente en la línea de código de interés. Y si fuese necesario solicitar cambios.

Mediante la opción “Request changes” realizamos la solicitud de cambios y allí se nos solicitara la descripción de los cambios pretendidos. Quedará abierto el pull request, entre las ramas **new-feature** y **develop** en este caso. Cuando se hagan los push con los correspondientes a los cambios solicitados, el desarrollador puede dar por cerrada esa solicitud de cambios, para que luego el reviewer los vuelva a revisar, los autorice y realice la mezcla de los cambios a *develop*.



Documentación oficial de Bitbucket

Para más información o dudas particulares acerca de Bitbucket se deja a continuación la web a los manuales, guías y documentación oficial de Bitbucket:

- Guías: https://bitbucket.org/product/guides
- Soporte: https://support.atlassian.com/bitbucket-cloud/
- Documentación: https://support.atlassian.com/bitbucket-cloud/resources/

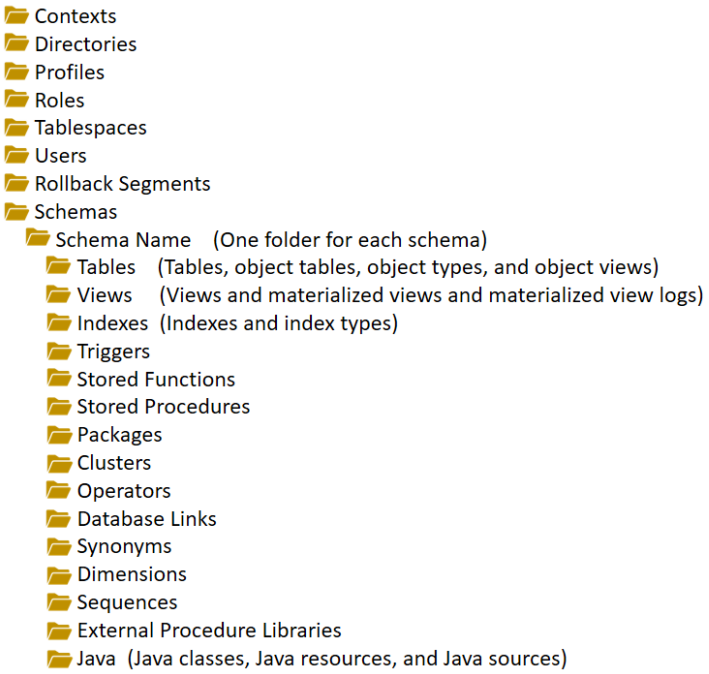
Migración de proyectos piloto

En esta colaboración se implementó la migración de los códigos fuentes de ciertos sistemas seleccionados por CMPC. Cada sistema se implementó como un proyecto de Bitbucket Cloud en cual posee un repositorio por cada aplicativo contenido en el mismo.

Jerarquía de carpetas para artefactos de Base de Datos

Dado que en estos sistemas parte de la lógica de negocios se encuentra implementada en las bases de datos, también resulta de interés versionar los códigos fuentes de los artefactos SQL. Para esto se propuso una jerarquía de carpetas para organizar estos repositorios, en las cuales se volcarán los archivos de código SQL según corresponda.

Esta jerarquía se presenta a continuación:



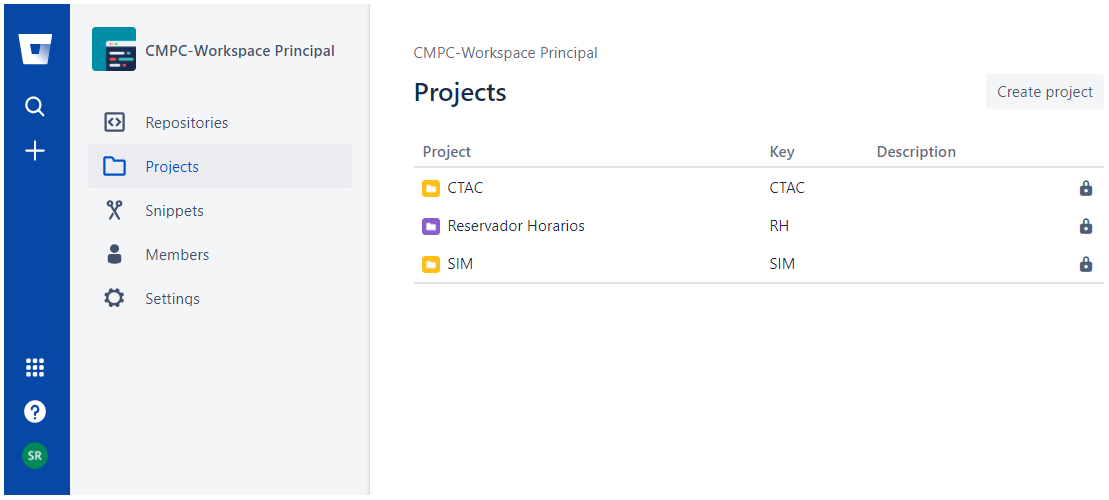
Sistemas seleccionados para la migración

Los sistemas seleccionados para la migración fueron el Sistema SIM (SAM y SGM), Sistema CTAC y Sistema de Reservador de Horarios. En donde el sistema SIM cuenta con tres repositorios, uno para base de datos (aquí encontramos dos esquemas, uno para SAM y otro para SGM), otro para la implementación de la los reportes y formularios de Oracle y el ultimo un aplicativo web desarrollado en .net. El sistema CTAC cuanta con un repositorio para la base de datos y otro para una web también en .net. Y por último el Reservador de Horarios cuenta solo con un repositorio para un aplicativo web .net ya que utiliza el esquema de bases de datos del sistema SAM.

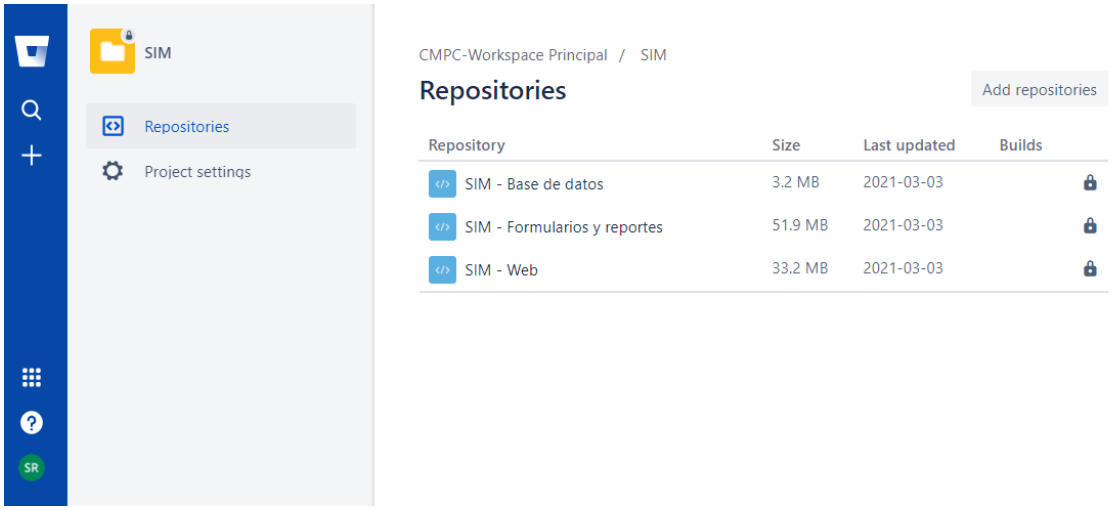
Implementación en Bitbucket

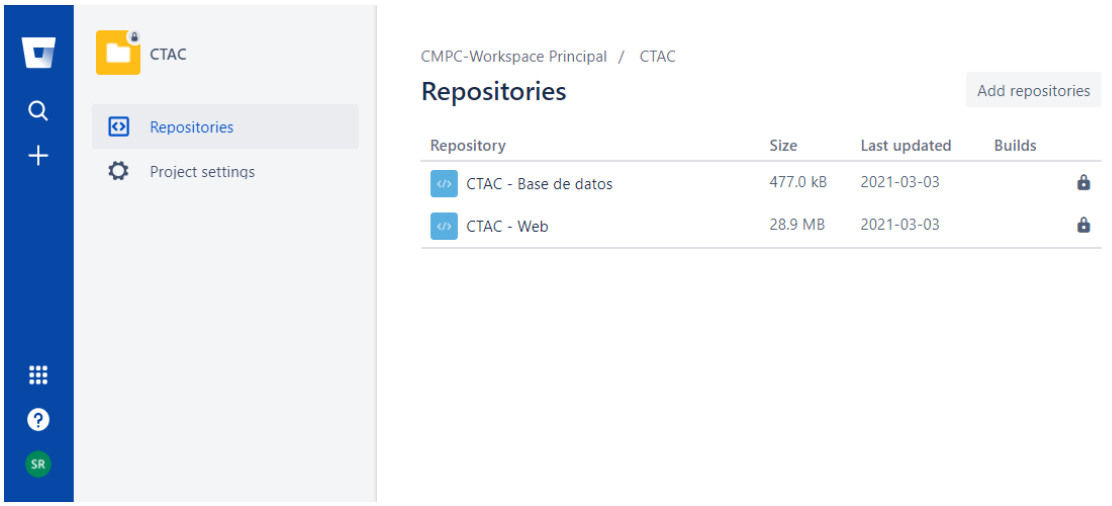
Para el proceso de implementación en Bitbucket se hizo la limpieza correspondiente de los códigos fuentes de los sistemas y sus aplicativos, así como también la generación de las carpetas para el versionado de los SQL. Recibimos estos fuentes a través de un servidor de CMPC al cual se nos dio acceso a través de una VPN. Finalmente se hizo la primar carga de los fuentes en los repositorios correspondientes y la generación de la rama develop a partir de master, de modo tal que los repositorios estén listos para trabajar según el flujo de trabajo propuesto.

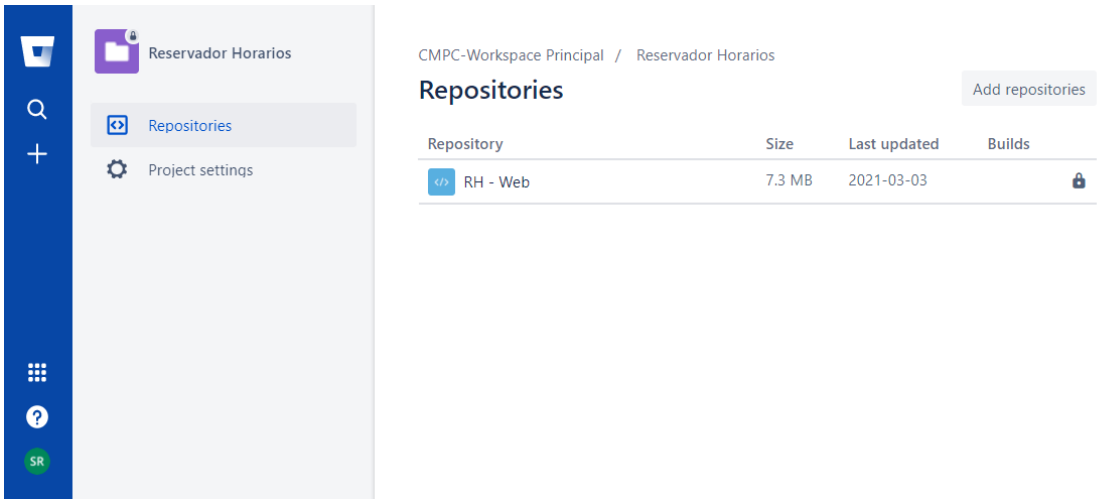
Finalmente, si se explora el workspace principal se encuentran los tres proyectos correspondientes a los sistemas de la siguiente manera:



Si se exploran los proyectos vemos los repositorios por los que están compuestos:



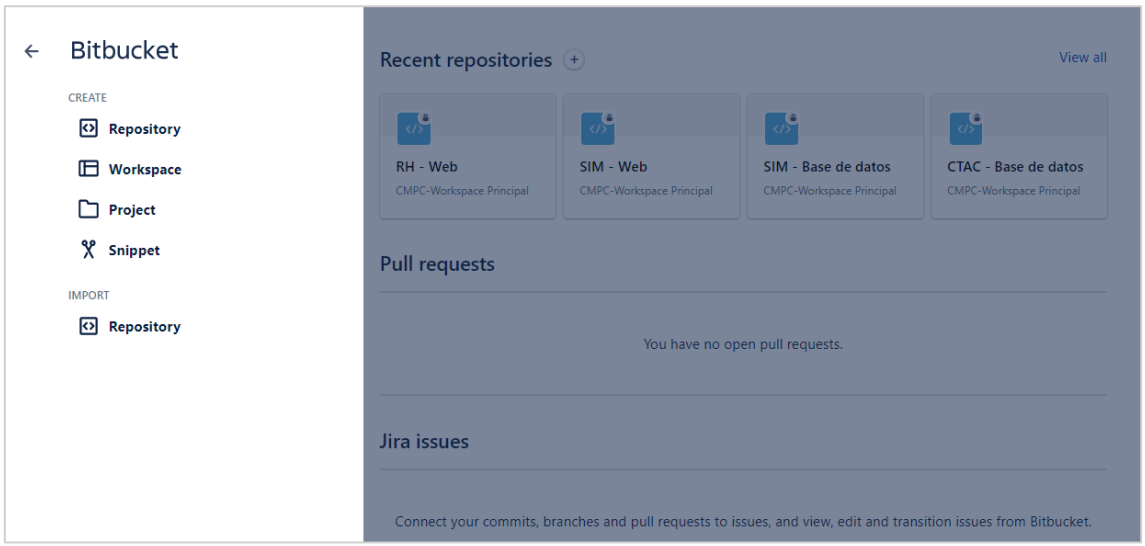




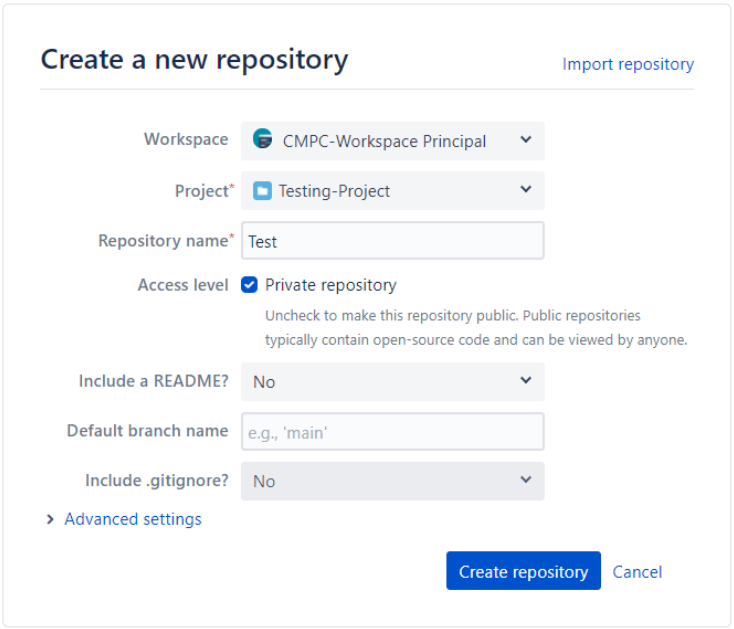
Creación de repositorio y primera inicialización

En este apartado se va a indicar como realizar la creación de un repositorio con su rama master, cargar el primer versionado de código, publicar esos cambios y a partir de la rama master crear la rama develop.

Para crear un repositorio simplemente hay que hacer click en el + que se encuentra en la barra de herramientas azul de la izquierda y nos va a dar las opciones para crear un repositorio, proyecto, workspace o snippet.

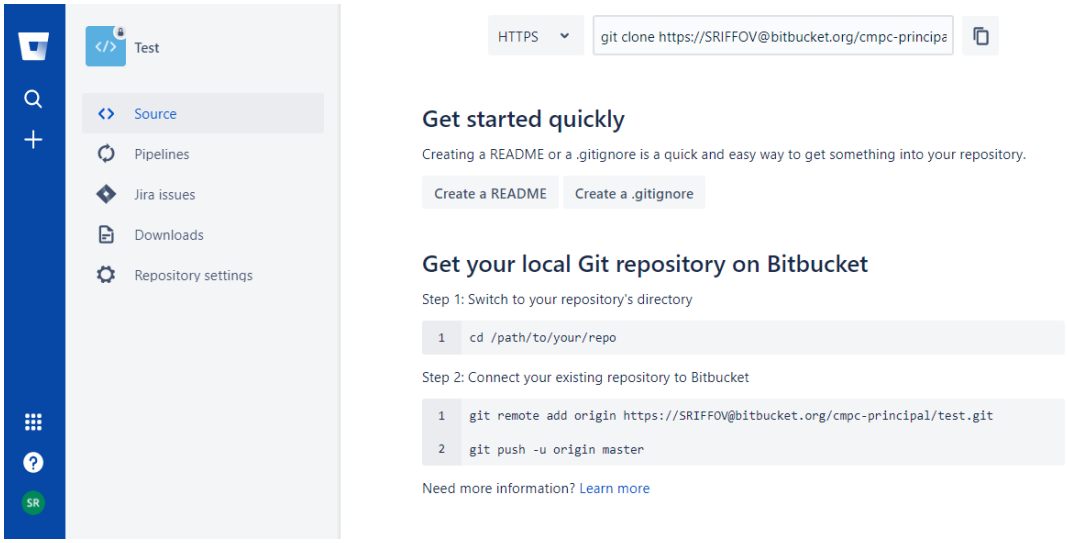


Damos en la opción de repositorio y nos va a llevar al siguiente dialogo en donde podremos configurar las siguientes opciones:

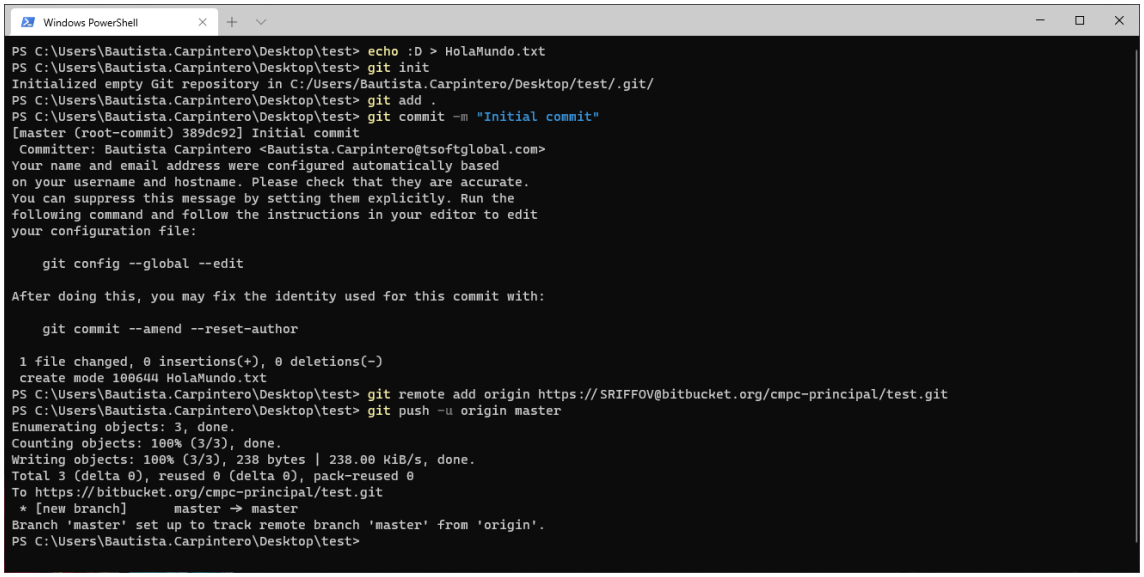


Prestar atención a que en la imagen anterior se negaron las opciones de inclusión de los archivos README y .gitignore. Esto se debe a que, al generar estos archivos durante la construcción del repositorio, automáticamente se va a generar la rama master y eso nos puede traer errores luego, cuando se desee hacer la primer cargar de los códigos fuentes del sistema que se esté migrando a Bitbucket.

Una vez que este creado el repositorio remoto, este va a estar vacío y nos a presentar las opciones para conectarse y vincularlo al repositorio local, donde esta nuestro código fuente.



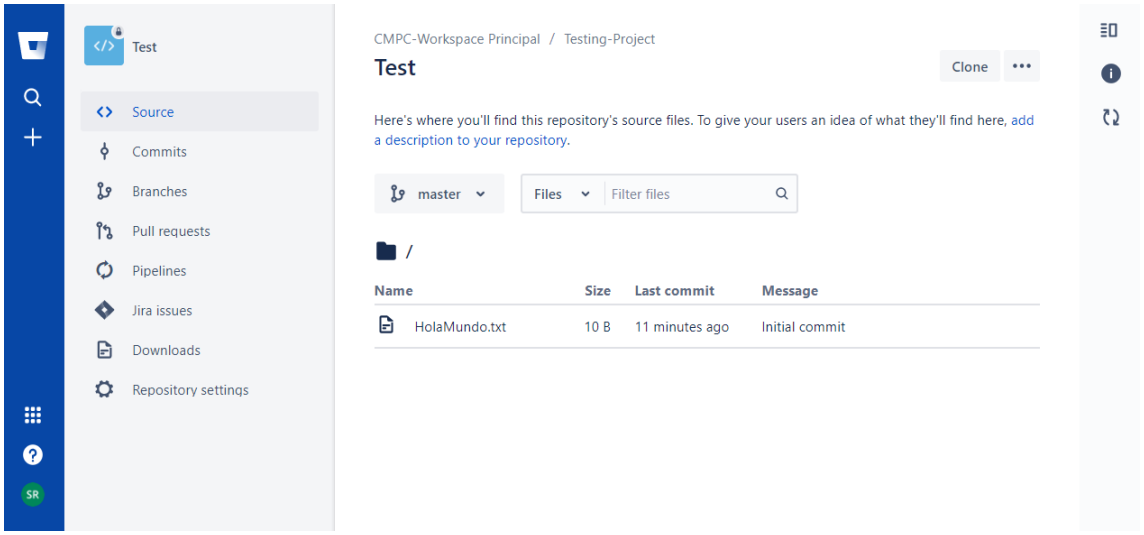
Pero antes de esto es necesario inicializar un repositorio local, para esto debemos tener instalado git en nuestro equipo, ir hacia el directorio de nuestros fuentes y ejecutar el comando '**`git init`**'. Una vez inicializado el repositorio hay que agregar todos los archivos ejecutando el comando '**`git add .`** '. Finalmente se hace el primer versionamiento del código en el repositorio local con **`git commit -m "Initial commit"`** para después si ejecutar los dos comandos sugeridos por Bitbucket, **`remote`** para vincular los repositorios y **`push`** para publicar el primer versionamiento o commit.



En la imagen anterior se muestra un ejemplo comenzando por una carpeta vacía llamada ‘test’ que representa nuestro directorio de códigos fuentes. Al estar vacía es necesario crear un archivo para versionar, por lo tanto, creamos un ‘HolaMundo.txt’. Este paso no va a ser necesario en un caso de uso real en donde ya se dispongan de los códigos fuentes en esa carpeta. Luego se ejecuta la secuencia de comandos de git mencionada anteriormente: init -> add -> commit -> remote -> push.

Nota: dado que el repositorio es privado, en caso que el usuario no se haya autenticado anteriormente, se va a solicitar usuario y clave de Bitbucket.

Una vez realizado el push, los cambios ya están publicados en el repositorio remoto y por lo tanto si refrescamos la web del repositorio en Bitbucket Cloud deberíamos ver nuestro archivo ‘HolaMundo.txt’.

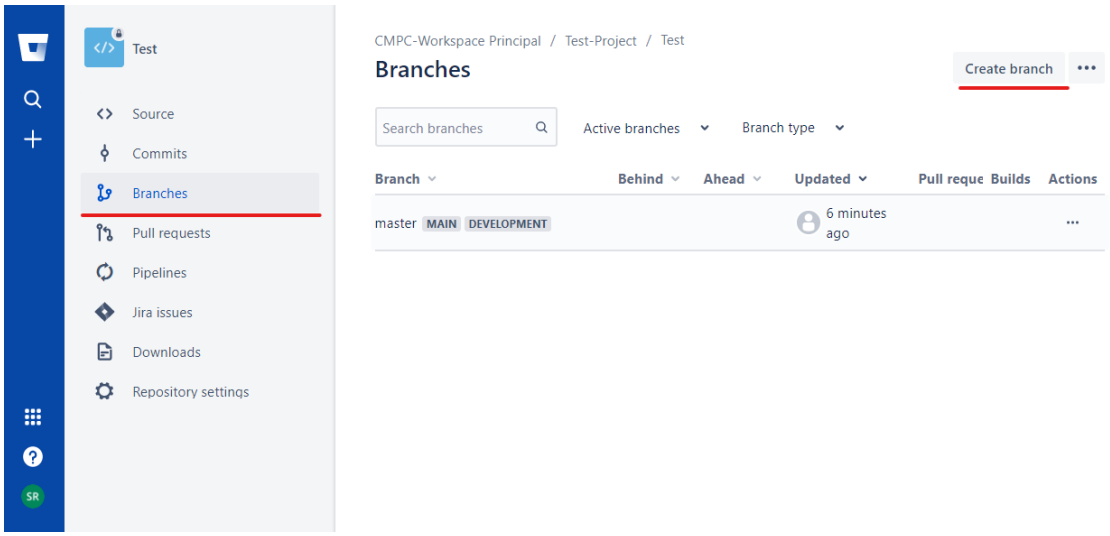


Creación de rama develop a partir de master

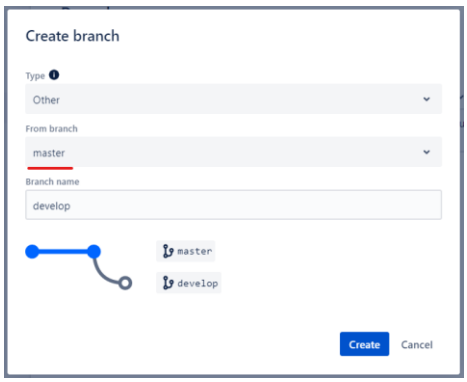
Ahora que ya tenemos el repositorio inicializado solo debemos crear la rama de desarrollo develop, a partir de la rama master. Esto lo podemos hacer de dos maneras, la primera es realizarlo desde el repositorio remoto, utilizando las opciones de Bitbucket. O la otra opción es realizarlo en el repositorio local utilizando los comandos de git, para luego publicar la rama al repositorio remoto.

Desde repositorio remoto

Para la primera opción simplemente tenemos que ir al repositorio remoto y entrar en la opción de ramas o branches, luego hacer click en crear rama o create branch:



Luego se nos presentara un dialogo a partir del cual le pondremos nombre a nuestra rama e indicaremos a partir de que rama preexistente la crearemos, en este caso develop a partir de master:



Click en Create y la rama ya está lista para su uso.

Desde repositorio local

Para la segunda opción, crearla en el repositorio local y después publicarla, ejecutamos la siguiente secuencia de comandos:

Debemos esta parados en la rama master o en la rama a partir de la cual queremos partir. Para esto:

→ **`git checkout master`**

Luego siempre es recomendable traerse los cambios que puedan llegar a haber en el repositorio remoto:

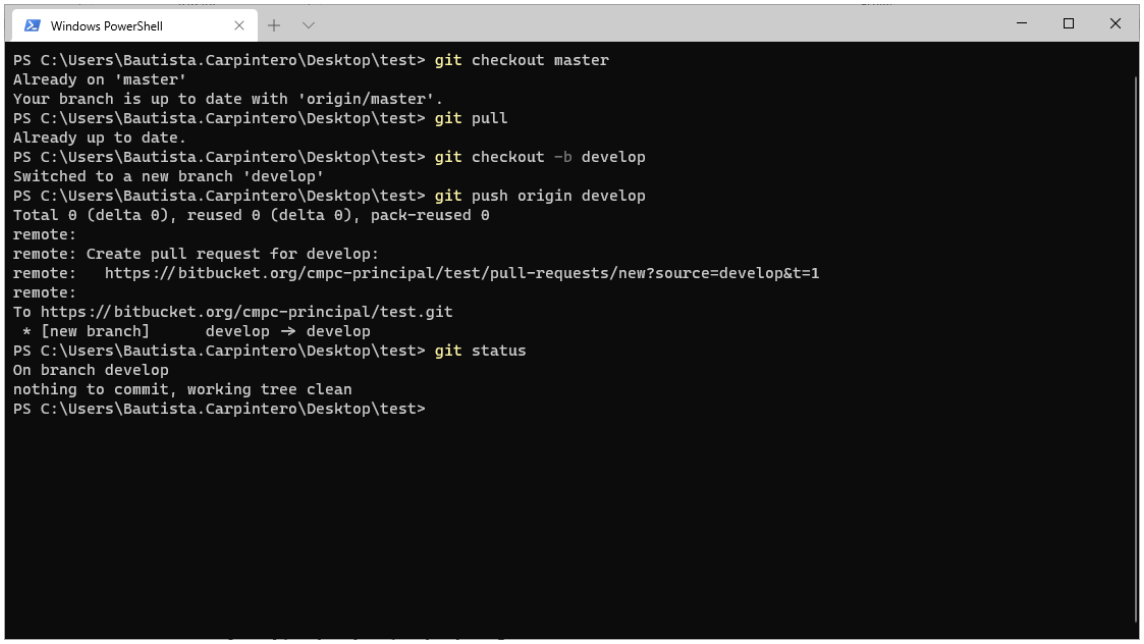
→ **`git pull`**

Creamos la nueva rama develop, luego de esto vamos a quedar posicionados esta:

→ **`git checkout -b develop`**

Publicamos la rama en el repositorio remoto:

→ **`git push origin develop`**



Luego podemos verificar en el repositorio remoto que la rama develop se haya creado exitosamente: