Proceso de Gestión de Versiones y Despliegue de Software de IT&S en VUCE

IT-SIS-03

Process Owner: Desarrollo de Sistemas / Alejandro Gonzalez Calderón

Registro de Cambios

Versión	Descripción	Autor	Área	Fecha de creación	Revisado por
1.0	Versión inicial	Paula Sajoux	Dirección IT&S	7/3/19	Alejandro Gonzalez Calderón.
1.1	Versión con correcciones de reuniones con Leandro Gorriz y Alejandro Gonzalez Calderon	Paula Sajoux	Dirección IT&S	12/6/19	Alejandro Gonzalez Calderón
1.2	Aprobación	Paula Sajoux	Direcciom IT&S	20/7/19	Alejandro Gonzalez Calderón

Contenido

Objetivo

El presente documento tiene como finalidad detallar el proceso de Gestión de Versiones y Despliegues software de la Dirección de Sistemas y Tecnología de VUCE.

Alcance

Aplica para todos los desarrollos de software de la Dirección de Sistemas y Tecnología de VUCE.

Definiciones generales

Conceptos Básicos:

Versión

Una versión es un conjunto nuevo o modificado de Configurations Items (CIs) que han sido validados y aprobados para su implementación en el entorno de producción. Las especificaciones funcionales y técnicas de una versión están determinadas en la solicitud de cambio correspondiente.

En el caso de un servicio nuevo o modificado se deben tener en cuenta todos los CIs que forman el servicio (infraestructura, hardware, software, aplicaciones, documentación, conocimiento, etc.) para diseñar, construir y configurar la versión.

Paquete de Diseño

El Paquete de Diseño del Servicio (Service Design Package, SDP) contiene toda la información del servicio registrada en el Catálogo de Servicios, incluyendo los requisitos que éste debe cumplir (SLAs, OLAs, SLRs, etc.).

En el contexto del proceso Gestión de Versiones y Despliegues, el paquete de diseño, es la especificación técnica y funcional que va a llegar al proceso con el detalle de lo que se requiere implementar en las versiones.

Tipos de despliegue

Las opciones más frecuentes para el despliegue de las versiones son:

- "**Big Bang**" versus planteamiento de fases: Un despliegue de tipo “Big Bang” ofrece el servicio, nuevo o modificado, para todos los usuarios al mismo tiempo, mientras que un despliegue por fases ofrece la versión a parte del total de usuarios en cada fase.
- **"Push" y "Pull":** Un planteamiento “Push” hace que el componente de servicio se despliegue desde el “centro” hasta las ubicaciones deseadas, mientras que un planteamiento “Pull” ofrece la nueva versión a los usuarios en un punto central, desde el cual puedan descargarla en sus ubicaciones cuando deseen.

- **Automatizado o manual:** La automatización ayudara a asegurar la repetibilidad y consistencia. Es posible que no siempre cuente con el tiempo requerido para proporcionar un mecanismo automatizado bien diseñado y eficiente o que no sea viable siempre. Si se utilizara un mecanismo manual, es importante monitorizar y medir el impacto de muchas actividades manuales repetidas ya que es probable que sean ineficientes y propensas de errores. Demasiadas actividades manuales ralentizaran al equipo encargado de la versión y crearan problemas de recursos o capacidad que afecten a los niveles de servicio.

Gestión de Configuración de Software (SCM)

Se encarga y controla:

- La elaboración de código fuente por varios desarrolladores simultáneamente.
- El seguimiento del estado de las fases del desarrollo de software (versiones) y sus cambios (control de versiones).
- La conducción de la integración de las partes del software en un solo producto de software.

Para poder realizar esto se apoya en herramientas de versionado.

El control de versiones se realiza, tanto para el código fuente como de las configuraciones de la aplicación.

Items bajo control de Configuración

Los elementos que son alcanzados por estas pautas establecidas en la configuración son los siguientes:

1. Fuentes que forman parte de la solución de software.
2. Configuraciones que forman parte de la solución de software.
3. Documentación sobre las decisiones de diseño de las aplicaciones va en Fileserver.

Sistema de Control de Versiones

Un sistema de control de versiones debe proporcionar:

- Mecanismo de almacenamiento de los elementos que deba gestionar (ej. archivos de texto, imágenes, documentación).
- Posibilidad de realizar cambios sobre los elementos almacenados (ej. modificaciones parciales, añadir, borrar, renombrar o mover elementos).
- Registro histórico de las acciones realizadas con cada elemento o conjunto de elementos (normalmente pudiendo volver o extraer un estado anterior del producto).

Actualmente se utilizan: GitLab y Bitbucket.

Descripción del uso del versionador:

Se utilizan básicamente 5 ramas o branches: Master, Desarrollo, QA, Features, y Hotfix.

- Master: es la rama principal. Contiene el repositorio de lo que se encuentra en producción, por lo que debe estar siempre estable.
- Desarrollo: Es una rama sacada del master. Es una rama de integración. Luego que se realice la integración y se corrijan los errores (en caso de haber alguno), es decir que la rama se encuentre estable, se puede hacer un merge de desarrollo sobre la rama QA.
- Features: cada nueva funcionalidad se debe realizar en una rama nueva, especifica para esa funcionalidad. Se debe sacar de la rama Desarrollo. Una vez que la funcionalidad esté desarrollada, se hace un merge sobre la rama de Desarrollo, donde se integrará con el resto de las funcionalidades.
- QA: es una rama sacada del desarrollo que fue probado y que pasa a QA para que el PO o usuario final realice la prueba de aceptación.
- Hotfix: son bugs que surgen en producción, por lo que se deben arreglar e implementar en producción de forma urgente. Por eso son ramas sacadas de Master. Una vez corregido el error, se debe hacer un merge sobre el Master. Luego, para que no quede desactualizada la versión en el resto de los ambientes, se debe hacer un merge sobre QA y luego sobre Desarrollo.

Gestión de Versiones y Despliegue de Software

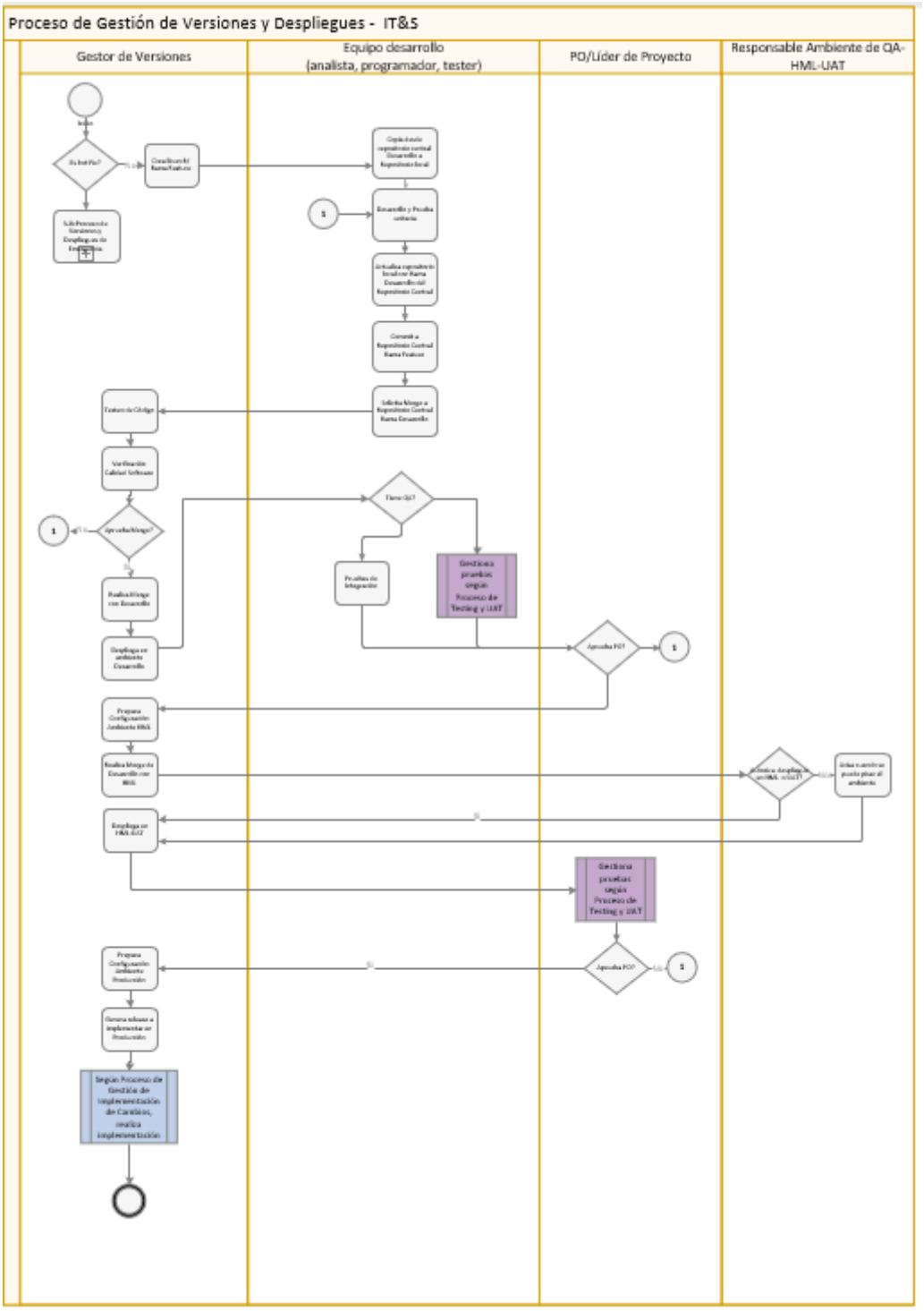
La Gestión de Versiones y Despliegues debe colaborar estrechamente con la Gestión de Cambios y de Configuraciones para asegurar que toda la información relativa a las nuevas versiones se integra adecuadamente en la Configuration Manager Database (CMDB) de forma que ésta se halle correctamente actualizada y ofrezca una imagen real de la configuración de la infraestructura Sistemas y Tecnología.

Actores / Roles

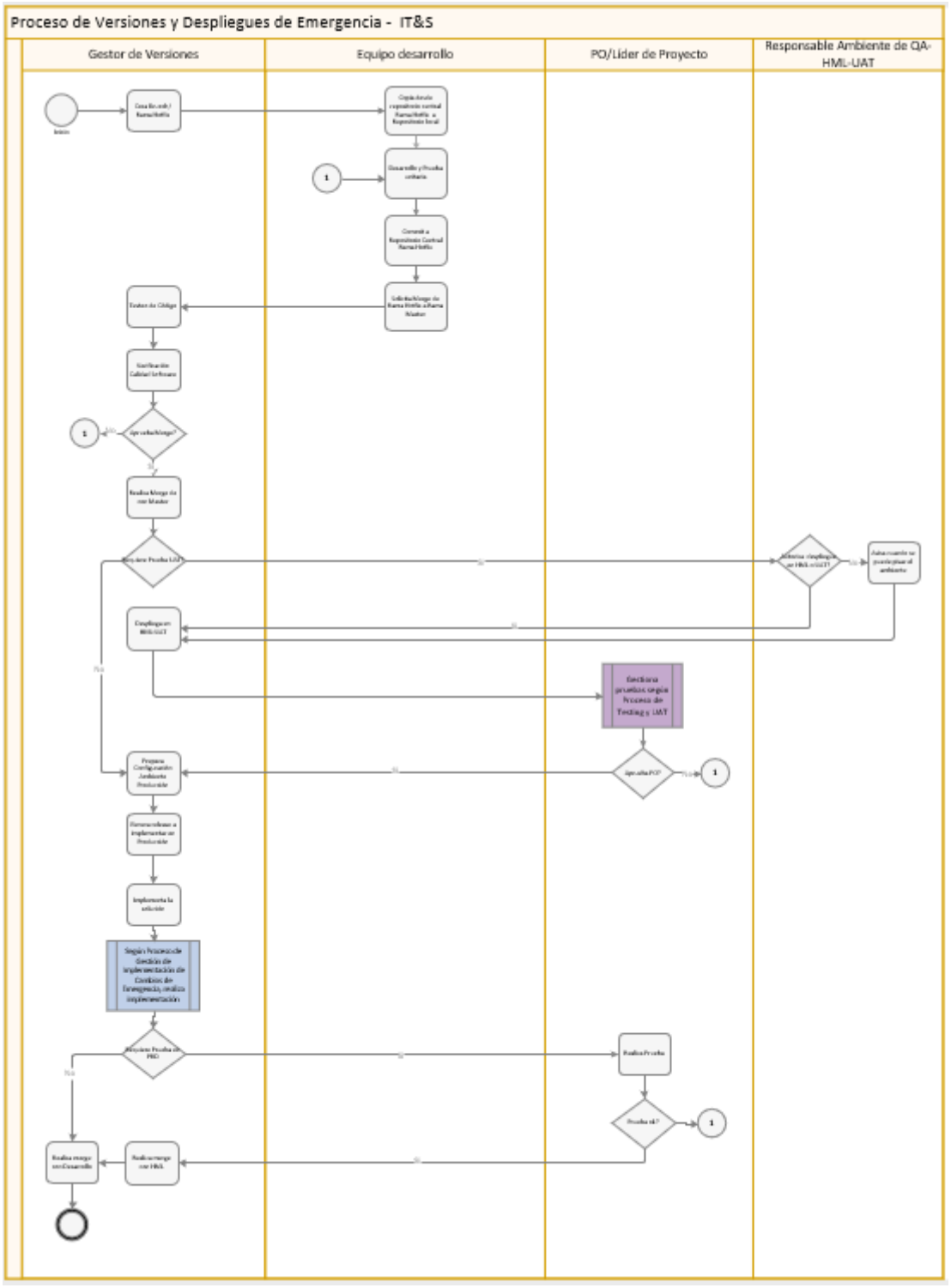
Esta sección define los roles y las responsabilidades sobre las actividades de la gestión de versiones y configuración de la misma:

Rol	Nombre	Responsabilidad
TM	Teammember	• Generar los diferentes ítems de Configuración (IC) que están controlados bajos las pautas de configuración de este proceso. • Acatar las pautas de gestión de versiones definida.
LT	Líder Técnico	• Validar la Calidad Técnica de los IC correspondientes. • Analizar y establecer la factibilidad a nivel técnico de cada liberación. • Velar por la integridad de la solución desde el punto de vista de su configuración. • Referente que da soporte y lineamientos al equipo sobre las características y lineamientos técnicos de las soluciones que el equipo desarrolla.
Gestor de Versiones	Gestor de Versiones	• Asegurar que las pautas establecidas para la configuración y versionado se cumplan. • Realizar revisiones o auditorias para asegurar la adherencia al plan. • Participar en la toma de decisión para liberar versiones. • Determinar riesgos de configuración de las soluciones. • Brindar soporte al Líder Técnico sobre las necesidades y herramientas que requiere el equipo. • Resolver los pedidos de merge de los desarrolladores, crear las ramas de las nuevas funcionalidades (features), migrar a QA y dejar disponible a Operaciones la versión a desplegar en Producción.

Flujo del Proceso

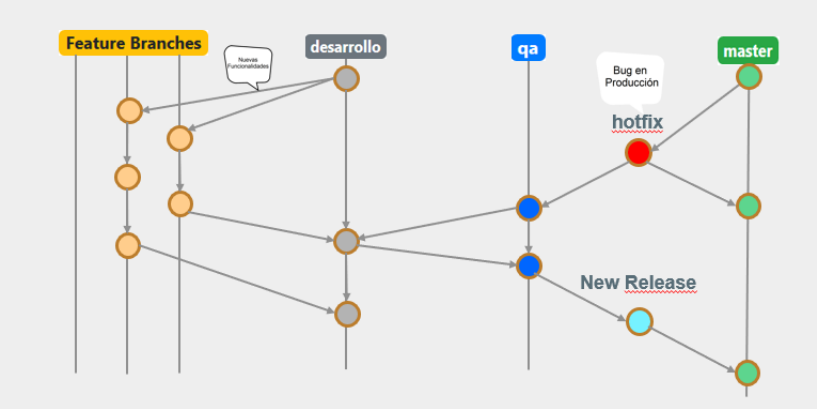


Flujo de subproceso de Versiones y Despliegues de Emergencia



Detalle de tareas de la Gestión de Versiones y Despliegue de Software

Gráfico:



1. Es Hot Fix

Si se trata de un Hot Fix continúa según subproceso "*Proceso de Gestión de Versiones y Despliegues de Emergencias*".

2. Crea de un Branch Rama Feature

El Gestor de Versiones crea una rama para la nueva funcionalidad (feature) en el proyecto a desarrollar desde el Desarrollo. Por ejemplo: CIVUCE-R001.

3. Copia desde Repositorio Central Rama Desarrollo a repositorio local

Para comenzar a trabajar el desarrollador realiza una clonación del repositorio desde la rama Desarrollo a su repositorio local.

4. Desarrollo y Prueba unitaria

El desarrollador realiza el desarrollo en forma local (Feature Branch), junto con la prueba unitaria.

5. Actualiza repositorio local con Rama Desarrollo del Repositorio Central

Una vez desarrollado el código, y antes de commitear el desarrollador debe actualizar el código local con los cambios remotos (rama desarrollo), si los hubiera. En caso de haber conflictos

(misma línea de uno o varios archivos modificados) resolverlos y probar que el código siga cumpliendo los criterios de aceptación en forma local.

6. Commit al Repositorio rama Feature

Una vez que el código está listo para ser subido al repositorio remoto, seguir estos 2 pasos:

a. Commit local - git commit -m “mensaje de commit” (commit al repositorio local): esto generará un nuevo commit en el repositorio local. El mensaje de commit deberá constar del siguiente formato:

“[ID STORY JIRA] - funcionalidad agregada”

```
Por ejemplo: git commit -m “[PROY] - Crear abm de clientes – se agrego el boton eliminar”

Nota: ID STORY JIRA puede ser cambiado por Identificador de tarea de Trello o de la herramienta
de soporte al desarrollo utilizada.
```

b. Subir el código al repositorio remoto - git push.

Nota: el desarrollador podría incluir observaciones que deben ser ejecutadas junto con el merge, como ser: ejecución de scripts de base de datos, configuración de variables de entorno, etc.

7. Solicita Merge a Repositorio Central Rama Desarrollo

Una vez que el desarrollador finaliza su desarrollo con pruebas exitosas, solicita el merge de su código generado con el repositorio desarrollo al Gestor de Versiones (según se explica el Anexo I - Cómo solicitar Merge en GitLab).

8. Testeo de Código

Durante el push de lo desarrollado se realizan Test automáticos de código con Gitlab.

9. Verificación de la Calidad de Software

El Gestor de versiones verifica la calidad de Software mediante SonarQube que sugiere mejoras con niveles de criticidad. El nivel crítico y bloqueante debe ser resuelto antes de pasar al siguiente entorno (ambiente de desarrollo). El Gestor de versiones podrá solicitar al desarrollador que se solucionen más niveles de considerarlo necesario.

10. Aprueba Merge

El Gestor de Versiones aprueba el Merge si la calidad de software es la requerida. Si lo aprueba continúa con la actividad siguiente *"Realiza el Merge con rama Desarrollo"*, sino sigue con la actividad *"Desarrollo y Prueba unitaria".*

11. Realiza Merge con Desarrollo

El Gestor de Versiones realiza el merge de la rama feature con la rama Desarrollo y resuelve los conflictos si los hubiera.

12. Despliega en ambiente de Desarrollo

Se realiza el despliegue en forma automática por medio de alguna de las siguientes herramientas: Bash o GitLab.

13. Verifica si tiene QA

Si tiene equipo de QA continúa con el *"Proceso de Testing y UAT"*. Si no tiene equipo de QA continúa con la actividad *"Prueba de Integración"*.

14. Pruebas de integración

El desarrollador realiza las pruebas de integración en el ambiente de Desarrollo.

15. Aprueba PO

Luego de las pruebas de integración, tanto del equipo de desarrollo como el equipo de QA si correspondiera, el PO aprueba o no el desarrollo. Si las pruebas no son exitosas el PO vuelve las tarjetas al desarrollador y continúa con la actividad *"Desarrollo y Prueba unitaria"*.

Si las pruebas son exitosas da el OK, adelanta las tarjetas de las User Stories y avisa a desarrollo para que avance. Continúa con la actividad siguiente.

16. Prepara la configuración del ambiente HML

Luego de la aprobación del PO, el Gestor de Versiones analiza las configuraciones a incluir en el deploy del sprint y genera el paquete a desplegar.

17. Realiza merge de desarrollo con HML/QA

El Gestor de Versiones realiza el Merge y soluciona los conflictos si los hubiera.

18. Autoriza despliegue en HML/UAT

El PO analiza si es el momento oportuno para pisar el ambiente QA con esta nueva versión, verifica si el entorno está congelado. Esa verificación la realiza con el *Responsable del ambiente de Homologación*. Sino es el momento espera que el ambiente de QA esté listo para una nueva versión (coordina la fecha en que el ambiente se puede pisar).

19. Avisa cuando se puede pisar el ambiente

El *Responsable del ambiente de Homologación*. Avisa al Gestor de Versiones que puede pisar el ambiente y continúa con la actividad de despliegue.

20. Despliega en Ambiente HML-UAT

El Gestor de Versiones realiza el despliegue en el ambiente HML.

21. Pruebas HML

El PO realiza las pruebas según el *"Proceso de Testing y UAT"*.

22. Aprueba PO?

Cuando el PO aprueba el sprint desarrollado le avisa al Gestor de Versiones. El Gestor de Versiones genera la documentación necesaria para solicitar la implementación en el ambiente de Producción.

La aprobación puede ser total (todas las funcionalidades dentro del sprint) o parcial siempre que no sean funcionalidades bloqueantes.

Si el PO no aprueba el sprint completo vuelve a la actividad *"Desarrollo y Prueba unitaria"*.

23. Prepara configuración del ambiente Producción

Luego de la aprobación del PO, el Gestor de Versiones analiza las configuraciones a incluir en el despliegue del sprint y genera el paquete a desplegar.

24. Genera release a implementar en Producción

El Gestor de Versiones genera el reléase a implementar:

- Se realiza un merge de HML con el master, y de ser necesario, se genera una nueva rama con el reléase completo para su implementación.

En el master se pone un tag con la versión implementada y sus funcionalidades.

25. Despliega en Producción

El Gestor de Versiones implementa según el *"Proceso de Gestión de Implementación de Cambios"*.

Detalle de tareas de Gestión de Versiones y Despliegues de Emergencia

1. Creación de un Branch por hotfix

El Gestor de Versiones crea una rama para el hotfix a desarrollar sacando el código fuente de Master.

2. Copia local del repositorio hotfix

Para comenzar a trabajar el desarrollador realiza una clonación del repositorio hotfix a su repositorio local.

3. Desarrollo y Prueba unitaria

El desarrollador realiza el desarrollo en forma local, junto con la prueba unitaria.

4. Commit al Repositorio central rama Hotfix

Una vez que el código está listo para ser subido al repositorio remoto, seguir estos 2 pasos:

a. Commit local - git commit -m “mensaje de commit”(commit al repositorio local) : esto generará un nuevo commit en el repositorio local. El mensaje de commit deberá constar del siguiente formato:

“[ID incidente error] – error xxxxx solucionado”

```
Por ejemplo: git commit -m “[PROY] - Crear abm de clientes – se agrego el boton eliminar”

Nota: ID Incidente Error puede ser cambiado por Identificador de tarea de Trello o de la herramienta de soporte al desarrollo utilizada.
```

b. Subir el código al repositorio remoto - git push.

Nota: el desarrollador podría incluir observaciones que deben ser ejecutadas junto con el merge, como ser: ejecución de scripts de base de datos, configuración de variables de entorno, etc.

5. Solicita merge de rama Hotfix a Rama Master

El Gestor de Versiones aprueba el merge y soluciona los conflictos si los hubiera.

6. Testeo de Código

Durante el push de lo desarrollado se realizan Test automáticos de código con Gitlab.

7. Verificación de la Calidad de Software

El Gestor de versiones verifica la calidad de Software mediante SonarQube que sugiere mejoras con niveles de criticidad. El nivel crítico y bloqueante debe ser resuelto antes de pasar al siguiente entorno (ambiente de desarrollo). El Gestor de versiones podrá solicitar al desarrollador que se solucionen más niveles de considerarlo necesario.

8. Aprueba Merge

El Gestor de Versiones aprueba el Marge si la calidad de software es la requerida.

Si lo aprueba continúa con la actividad “*Realiza el Merge con Master*”, sino sigue con la actividad *“Desarrollo y Prueba Unitaria”*.

9. Realiza Merge con Master

El Gestor de Versiones realiza el merge de la rama hotfix con la rama Master y resuelve los conflictos si los hubiera.

10. Requiere Prueba UAT?

Si sólo se cambió un texto realiza las pruebas en el ambiente productivo (PRD) directamente y la siguiente actividad es *"Prepara Configuración Ambiente Producción*".

Sino debe realizar las pruebas en el ambiente de HML/UAT y la siguiente actividad es *"Autoriza despliegue en HML/UAT"*.

11. Autoriza despliegue en HML/UAT

El PO analiza si es el momento oportuno para pisar el ambiente HML con esta nueva versión, verifica si el entorno está congelado. Esa verificación la realiza con el *Responsable del ambiente de Homologación*. Sino es el momento espera que el ambiente de HML esté listo para una nueva versión (coordina la fecha en que el ambiente se puede pisar).

12. Despliega en Ambiente HML/UAT

El Gestor de Versiones realiza el despliegue en el ambiente HML.

13. Pruebas UAT

El PO realiza las pruebas según el *"Proceso de Testing y UAT"*.

14. Aprueba PO

Si el PO aprueba la versión continúa con la actividad *"Prepara configuración del ambiente Producción"*, sino continua con la actividad *"Desarrollo y Prueba unitaria"*.

15. Prepara configuración del ambiente Producción

El Gestor de Versiones analiza las configuraciones a incluir en el despliegue del sprint y genera el paquete a desplegar.

16. Genera release a implementar en Producción

El Gestor de Versiones genera el reléase a implementar. En el master se pone un tag con la versión implementada y sus funcionalidades.

17. Implementa la solución del hotfix

El Gestor de Versiones gestiona la implementación por medio de un cambio de Emergencias según el *"Proceso de Gestión de Implementación de Cambios de Emergencia"* para la implementación del fix.

18. Verifica si requiere pruebas en Producción

De no haberse realizado las pruebas en el ambiente HML/UAT se realiza la prueba en Producción. Continúa con la actividad *"Realiza Pruebas".*

De no requerir la prueba en producción continúa con la actividad "*Realiza merge con Desarrollo*".

19. Realiza Pruebas

El PO realiza las pruebas de la funcionalidad en producción.

20. Verifica si las pruebas fueron exitosas

Si las pruebas fueron exitosas continua con la actividad *"Realiza merge con HML"*.

Si las pruebas no fueron exitosas realiza rollback y continua con la actividad *"Desarrollo y Prueba unitaria"*.

21. Realiza merge con HML

Realiza el merge con HML. Luego continúa con la actividad *"Realiza merge con Desarrollo"*.

22. Realiza merge con Desarrollo

Realiza merge con desarrollo dejando actualizados todos los repositorios con el hotfix desarrollado.

Versionado de Configuraciones

Las configuraciones se encuentran en un repositorio aparte, diferenciadas por ambiente: desarrollo, QA y PRD (producción).

Tips a tener en cuenta en los comentarios para Commit

- El mensaje debería describir a grandes rasgos el cambio que se está implementando, para dar una idea (en caso que hubiese) al reviewer la funcionalidad que se modificó y cuál es el objetivo que debe cumplir el commit.

  "[TICKET-1] - Mejorar Performance home page - se reemplazó los 5 for anidados por una función de java"

- En caso de que sea un commit que afecte a varios módulos de la aplicación, se debería agregar ese detalle.

  "[TICKET-1] - Actualizar tasa de impuesto - Afecta a los modulos de facturacion y cobranza"

- Para eliminar redundancia se sugiere no poner comentarios sobre temas no relacionados con funcionalidad o que no brinden información sobre ese commit, tales como:

  "se agrego un archivo nuevo","codigo actualizado","se eliminaron 3 archivos."

- Más de una funcionalidad por commit es una mala práctica, cuando se introducen uno o más fixes y/o nueva funcionalidad y este hace que la aplicación no se comporte como debería, no sabremos cual de todas fue la que causó el problema. Se recomienda dividirla en 2 o más commits para poder identificarla.
- Responder a la pregunta de: "Por qué es necesario este cambio?
- No asumir que el revisor de código entiende de qué se trata el commit.

Release Management

Es el proceso responsable de planear, programar y controlar los movimientos de las releases en los distintos entornos.

El objetivo del Release Management es asegurar la integridad de los entornos y que los correctos artefactos han sido desplegados.

Todos los artefactos liberados deberán ser identificados con un número de versión (Release version number).

Trazabilidad con el número de versión

El número de versión de release nos ayudará a identificar los cambios y/o funcionalidades que están incluida en el software que será implementado. Los números iterarán de acuerdo al tipo de funcionalidad liberada.

Es necesario lograr la trazabilidad entre:

- El código de la aplicación que generó el binario luego de la compilación.
- Las funcionalidades que son liberadas con el binario.

Nomenclatura propuesta:

V[X].[Y].[Z]

● X: Es un release que involucra grandes cambios de funcionalidad.

● Y: Se agregan funcionalidades nuevas al reléase.

● Z: Release que se libera para aplicar correcciones a un problema detectado en algún ambiente.

La nomenclatura debe ser de letra V y 3 dígitos, comenzando de izquierda a derecha.

Por ejemplo:

Nuestra primera versión será: **v1.0.0**

Si se encontró un bug en algún ambiente y se corrigió, esta será: v1.0.1 (itera Z).

Si la versión v1.0.1 génerо algún que otro bug, la próxima será: v1.0.2 (itera Z).

Próximo sprint será el release v1.1.0 (itera Y).

Matriz RACI

En esta matriz se asignan los roles que un recurso debe ejecutar, para cada actividad dada.

A continuación, se describe la representación de cada una de las letras asignadas.

R - Responsible (responsable de la ejecución)

Alguien que desempeña una tarea determinada. Para cada tarea en un proceso ITIL existe normalmente un rol ITIL responsable de su ejecución.

A - Accountable (responsable del proceso en conjunto)

Alguien que asume la responsabilidad conjunta final por la correcta y completa ejecución de un proceso y que recibe las informaciones de los responsables de la ejecución del proceso. Normalmente, el Responsable de Proceso asume la responsabilidad conjunta de un proceso y para cada proceso existe un Responsable de Proceso.

C - Consulted (consultado)

Alguien que no está implicado directamente en la ejecución de un proceso pero que brinda algún tipo de input para el proceso y/o al cual se pide su consejo y opinión.

I - Informed (a informar)

Alguien que recibe las salidas (outputs) de un proceso o a quien se informa de los avances del proceso.

Gestión de Versiones y Despliegues

TAREAS	Entradas	Salidas	ROLES Y RESPONSABILIDADES			
			Gestor de Versiones	Desarrollo de Sistemas	PO / Líder de Proyecto	Resp. Ambiente QA
Verifica si es Hotfix	Información del evolutivo / correctivo	Información del evolutivo validada. Información del correctivo validado. Gestión de Versiones y Despliegues de Emergencias cuando es hotfix.	I	R		
Crea Brunch / Rama Feature	Información del evolutivo validada.	Nueva rama feature creada.	R	I		
Copia desde repositorio central Desarrollo a Repositorio local	Repositorio de fuentes central rama Desarrollo	Repositorio local con fuentes central rama desarrollo		R		
Desarrollo y Prueba unitaria	Repositorio local con fuentes central rama desarrollo. Historias de Usuario a desarrollar	Historias de Usuario desarrolladas. Prueba unitaria exitosa. Código fuente actualizado en Repositorio Local.		R		
Actualiza repositorio local con Rama Desarrollo del Repositorio Central	Código fuente actualizado en repositorio Local. Código fuente actualizado con Rama Desarrollo del repositorio Central.	Código fuente actualizado en repositorio local con el repositorio Central		R		
Commit a Repositorio Central Rama Feature	Código fuente actualizado	Código fuente commiteado a Repositorio Central Rama Feature		R		
Solicita Merge a Repositorio Central Rama Desarrollo	Código fuente commiteado a Repositorio Central Rama Feature. Solicitud de Merge a Repositorio Central Rama Desarrollo.	Código fuente commiteado a Repositorio Central Rama Feature. Solicitud de Merge a Repositorio Central Rama Desarrollo.	I	R		
Testeo de Código	Código Fuente commiteado	Código fuente con test automáticos realizados.	R	I		
Verificación Calidad Software	Código Fuente testeado	Código fuente Verificado. Informe de mejoras del código.	R	I		
Aprueba Merge	Código fuente con test automáticos realizados. Código fuente Verificado. Informe de mejoras del código.	Aprobación del Merge. Informe de mejoras a realizar.	R	I		
Realiza Merge con Rama Desarrollo	Código fuente con test automáticos realizados. Código fuente Verificado.	Código fuente actualizado en Rama Desarrollo.	R			
Despliegue en Ambiente Desarrollo	Código fuente actualizado en Rama Desarrollo.	Código desplegado en Ambiente Desarrollo.	R	I		
Verifica si tiene QA	QA	Si tiene QA invoca al *"Proceso de Testing y UAT"*. Si no tiene QA continúa con la actividad de "Pruebas de Integración".		R		
Pruebas de Integración	Casos de Prueba de Integración.	Casos de Prueba de integración ejecutados.		R		
Aprueba PO	Casos de Prueba de integración ejecutados.	Aprobación del PO y avance de US. Rechazo del PO.			R	
Prepara la configuración del ambiente de HML	Configuración ambiente desarrollo	Configuración a realizar en HML	R			
Realiza Merge de Rama Desarrollo con HML	Código Fuente verificado rama desarrollo Código Fuente rama HML.	Código fuente actualizado en Rama HML.	R			
Autoriza Despliegue en HML/UAT	Código fuente actualizado en Rama HML.	Autorización despliegue en HML	R			C
Avisa cuando se puede pisar el ambiente	Autorización solicitada	Autoriza a pisar HML	I			R
Despliega en Ambiente HML-UAT	Código fuente actualizado en Rama HML. Autorización despliegue en HML.	Despliegue en HML Invoca al *"Proceso de Testing y UAT"* para las Pruebas	R	I	I	I
Aprueba PO	Informe de pruebas ejecutadas.	Aprobación de funcionalidades.	I	I	R	
Prepara configuración ambiente Producción	Configuración realizada en ambiente HML	Configuraciones a realizar en PRD	R			
Genera release a implementar en Producción	Código fuente actualizado en Rama HML.	Despliega en Producción Según Proceso de Gestión de Implementación de Cambios, realiza implementación	R		I	

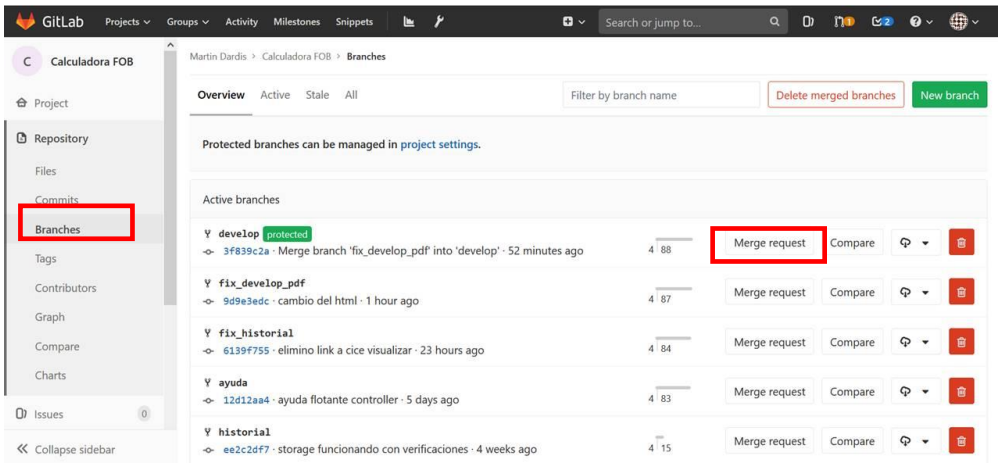
Gestión de Versiones y Despliegues de Emergencia

TAREAS	Entradas	Salidas	ROLES Y RESPONSABILIDADES			
			Gestor de Versiones	Desarrollo de Sistemas	PO / Lider de Proyecto	Resp. Ambiente QA
Crea Brunch/Rama Hotfix	Información del evolutivo validada.	Nueva rama hotfix creada.	R	I		
Copia desde repositorio central Rama Hotfix a Repositorio local	Repositorio de fuentes central rama Hotfix	Repositorio local con fuentes central hotfix		R		
Desarrollo y Prueba unitaria	Repositorio local con fuentes central rama desarrollo. Historias de Usuario a desarrollar	Historias de Usuario desarrolladas. Prueba unitaria exitosa. Código fuente actualizado en Repositorio Local.		R		
Commit a Repositorio Central Rama Hotfix	Código fuente actualizado	Código fuente commiteado a Repositorio Central Rama Hotfix		R		
Solicita Merge de Rama Hotfix a Rama Master	Código fuente commiteado a Repositorio Central Rama HotFix. Solicitud de Merge a Repositorio Central Rama HotFix	Código fuente commiteado a Repositorio Central Rama HotFix. Solicitud de Merge a Repositorio Central Rama HotFix	I	R		
Testeo de Código	Código Fuente commiteado	Código fuente con test automáticos realizados.	R	I		
Verificación Calidad Software	Código Fuente testeado	Código fuente Verificado. Informe de mejoras del código.	R	I		
Aprueba Merge	Código fuente con test automáticos realizados. Código fuente Verificado. Informe de mejoras del código.	Aprobación/Desaprobación del Merge. Informe de mejoras a realizar.	R	I		
Realiza Merge de con Master	Código fuente con test automáticos realizados. Código fuente Verificado.	Código fuente actualizado en Rama Master	R			
Requiere prueba UAT	Código fuenta actualizado en Rama Master	Si requiere Prueba en UAT continua con la actividad "Autoriza Despliegue en HML/QA/UAT". Si no requiere Prueba en UAT continúa con actividad "Prepara Configuración Ambiente Producción"	R		I	
Autoriza Despliegue en HML/UAT	Código fuente actualizado en Rama HML.	Autorización despliegue en HML Solicitud de espera para pisar ambiente HML	R			C
Avisa cuando se puede pisar el ambiente	Solicitud de espera para pisar ambiente HML	Autorización a pisar HML				R
Despliega en Ambiente HML	Código fuente actualizado en Rama HML. Autorización despliegue en HML.	Despliega en HML Invoca al *"Proceso de Testing y UAT"* para las Pruebas	R	I	I	I
Aprueba PO	Informe de pruebas ejecutadas.	Aprobación de hotfix			R	
Prepara configuración ambiente Producción	Lista de configuraciones a realizar	Configuraciones verificadas a realizar en PRD	R			
Genera release a implementar en Producción	Código fuente actualizado en Rama HML.	Despliega en Producción Según Proceso de Gestión de Implementación de Cambios	R			
Realiza Prueba en PRD	Necesidad de Pruebas en PRD con sus casos de prueba	Pruebas realizadas Si las pruebas OK continúa con actividad "Realiza Merge con HML". Si las pruebas NO OK continúa con la actividad "Desarrollo y Prueba unitaria".		I	R	
Realiza merge con HML	Código fuente actualizado en Rama Master.	Código fuente actualizado en Rama HML Continúa con actividad "Realiza merge con Desarrollo"	R	I		
Realiza merge con Desarrollo	Código fuente actualizado en Rama HML	Código fuente actualizado en Rama Desarrollo	R	I		

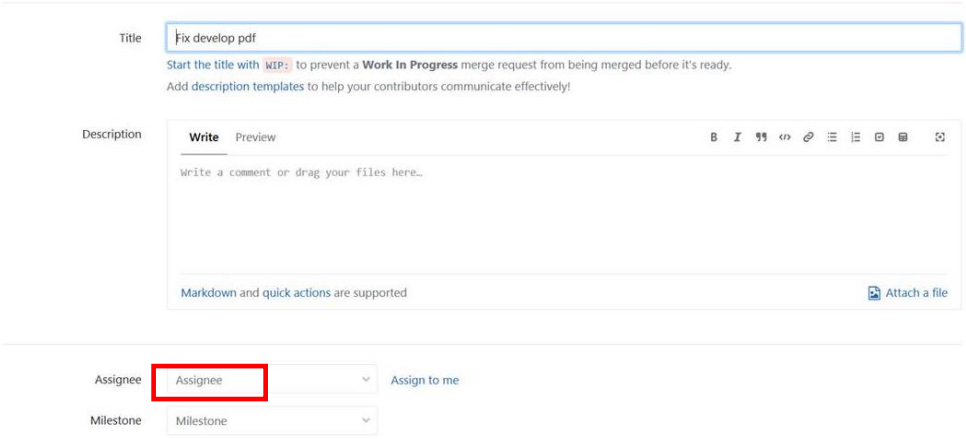
Anexo I – Guías de paso a paso en las herramientas

1. Cómo solicitar Merge en GitLab

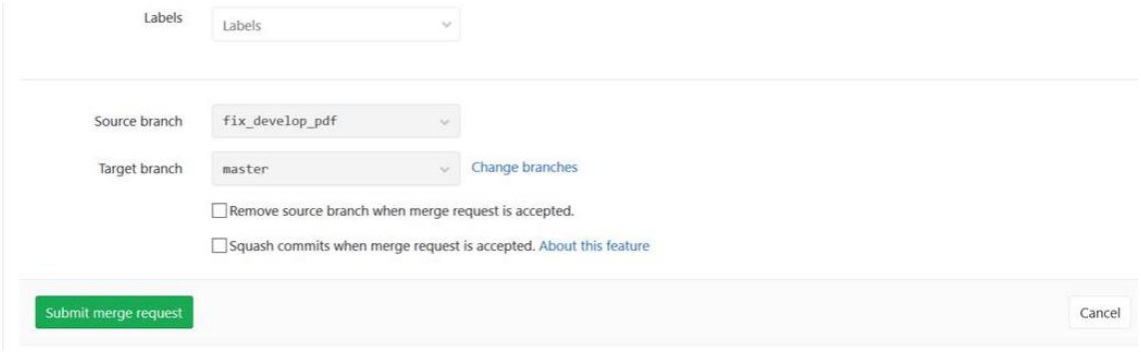
1.1. Ingresar a GitLab y posicionarse en el Branch
1.2. Seleccionar “Merge Request”.



1.3. Completar los campos requeridos:



Donde dice “*Assignee*” poner al Gestor de Versiones (Leandro Gorriz).



Una vez completados todos los campos presionar el botón "*Submit merge request*".

2. Como etiquetar versión en GIT - Procedimiento manual:

El objetivo es identificar el número de versión del commit a ser desplegada en los ambientes.

Una vez identificado se crea una "Etiqueta" (git tag) con el objetivo de establecer una nomenclatura a dicha versión (ver punto anterior Release Version), dicha etiqueta creará una versión del código que no podrá ser modificada.

Para crear un tag en git se debe utilizar el siguiente comando, suponiendo que la versión del siguiente release es la v1.0.0:

git tag v1.0.0

git push origin v1.0.0

Con git push origin <nombre-tag>, estaremos subiendo el tag al repositorio central, en este paso estaremos congelando el código bajo la etiqueta v1.0.0.

Una vez creado se deberá generar el entregable también llamado "artefacto" y ser subido a artifactory (Servidor de repositorios de binarios), para luego ser utilizado en el proceso de deployment.

3. Cómo crear la versión en Jira:

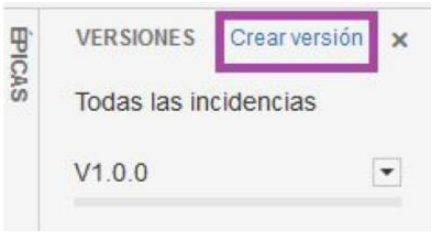
Una versión se puede crear desde 2 secciones diferentes:

- Backlog.
- Lanzamientos

1.1 Backlog:

Seleccionar Backlog, luego ir a la solapa VERSIONES.

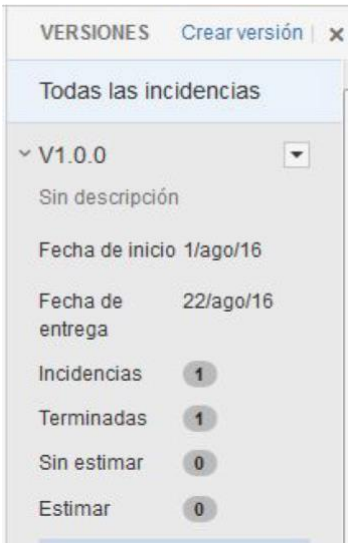
En el menú de versiones que se muestra, pasar el mouse sobre el título el panel, en el mismo se muestra al lado del título un link para Crear versión.



Al presionar sobre el mismo se muestra el siguiente formulario:

Completar el mismo con los datos correspondientes y presionar en botón Crear.

En este ejemplo se creó la versión V1.0.0, la misma se mostrará en el listado de versiones sobre el mismo panel.



Para cerrar el panel, presionar sobre la " x " que se muestra al lado del link Crear versión.

Nota: un release se puede crear en el momento que se desee, no es obligatorio crearla al inicio del proyecto; pero tener en cuenta que, de no ser así, luego se deberán actualizar todas las incidencias agregando la versión a la que corresponde.

1.2 Lanzamientos

Seleccione Lanzamientos, y luego a la derecha está el botón “Crear versión”.

Anexo II - Glosario

CI – Configuration Item

CMDB – Configuration Manager Database

Commit

Es la confirmación del cambio en la rama local.

Repositorio

El **repositorio** es el lugar en el que se almacenan los fuentes, configuraciones y archivos varios actualizados e históricos de cambios.

Versión o Revisión o Release

Una **revisión** es una versión determinada de la información que se gestiona. Hay sistemas que identifican las revisiones con un contador (Ej. subversión).

Rotular (*"tag"*)

Darle a alguna versión de cada uno de los archivos del módulo en desarrollo en un momento preciso un nombre común ("etiqueta" o "rótulo") para asegurarse de reencontrar ese estado de desarrollo posteriormente bajo ese nombre. En la práctica se rotula a todos los archivos en un momento determinado. Para eso el módulo se "congela" durante el rotulado para imponer una versión coherente. Pero bajo ciertas circunstancias puede ser necesario utilizar versiones de algunos archivos que no coinciden temporalmente con las de los otros archivos del módulo.

Los tags permiten identificar de forma fácil versiones importantes en el proyecto. Por ejemplo, se suelen usar tags para identificar el contenido de las versiones publicadas del proyecto.

En algunos sistemas se considera un tag como una rama en la que los archivos no evolucionan, están congelados.

Branch o rama

Un módulo puede ser ***branched*** o **bifurcado** en un instante de tiempo de forma que, desde ese momento en adelante se tienen dos copias (ramas) que evolucionan de forma independiente siguiendo su propia línea de desarrollo. El módulo tiene entonces 2 (o más) "ramas". La ventaja es que se puede hacer un "merge" de las modificaciones de ambas ramas, posibilitando la creación de "ramas de prueba" que contengan código para evaluación, si se decide que las modificaciones realizadas en la "rama de prueba" sean preservadas, se hace un "merge" con la rama principal. Son motivos habituales para la creación de ramas la creación de nuevas funcionalidades o la corrección de errores.

Integración o fusión (*"merge"*)

Una **integración** o **fusión** une dos conjuntos de cambios sobre un archivo o un conjunto de archivos en una revisión unificada de dicho archivo o archivos.

- Esto puede suceder cuando un usuario, trabajando en esos archivos, **actualiza** su copia local con los cambios realizados, y añadidos al repositorio, por otros usuarios. Análogamente, este mismo proceso puede ocurrir en el repositorio cuando un usuario intenta ***check-in*** sus cambios.
- Puede suceder después de que el código haya sido ***branched***, y un problema anterior al *branching* sea arreglado en una rama, y se necesite incorporar dicho arreglo en la otra.
- Puede suceder después de que los archivos hayan sido ***branched***, desarrollados de forma independiente por un tiempo, y que entonces se haya requerido que fueran fundidos de nuevo en un único *trunk* unificado.