**Alcance**

Establecer los mecanismo, flujos y herramientas necesarias para la administración de las diversas versiones de cada uno de los desarrollos creados por ABS.

**Introducción**

Dentro de un equipo ágil cada componente del desarrollo (código, documentos, diseños, etc.) son de **intelecto colectivo** por lo cual surge la necesidad de mantener cada componente en un lugar accesible para todos los miembros del equipo, lo cual comienza a volverse cuello de botella si no sabemos manejarlo ocasionando problemas como por ejemplo un miembro del equipo trabajando en una versión no actualizada, por lo que el uso de un software para el control de versiones se vuelve indispensable.

Una versión de un producto, es el estado en el que se encuentra dicho producto en un momento dado de su desarrollo. Se llama control de versiones a la gestión de los diversos cambios que se realizan sobre los elementos de algún producto. Los sistemas de control de versiones facilitan la administración de las distintas versiones de cada producto desarrollado, así como las posibles especializaciones realizadas (por ejemplo, para algún cliente específico).

El control de versiones se realiza principalmente en la industria informática para controlar las distintas versiones del código fuente. Sin embargo, los mismos conceptos son aplicables a otros ámbitos como documentos, imágenes, sitios web, etcétera.

Aunque un sistema de control de versiones puede realizarse de forma manual, es muy aconsejable disponer de herramientas que faciliten esta gestión ***(CVS, Subversion, SourceSafe, ClearCase, Darcs, Bazaar , Plastic SCM, Git, Mercurial, etc.)***

**Control de versiones**

**1.1 Características**

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)

**1.2 Clasificación**

La principal clasificación que se puede establecer está basada en el almacenamiento del código:

* ***Centralizados***: Existe un repositorio centralizado de todo el código, del cual es responsable un único usuario (o conjunto de ellos). Se facilitan las tareas administrativas a cambio de reducir flexibilidad, pues todas las decisiones fuertes (como crear una nueva rama) necesitan la aprobación del responsable. Algunos ejemplos son ***CVS*** y ***Subversion***.
* ***Distribuidos***: Cada usuario tiene su propio repositorio. No es necesario tomar decisiones centralizadamente. Los distintos repositorios pueden intercambiar y mezclar revisiones entre ellos. Ejemplos: ***Git*** y ***Mercurial***.

**Ventajas de sistemas distribuidos**

* Necesita menos veces estar conectado a la red para hacer operaciones. Esto produce una mayor autonomía y una mayor rapidez.
* Aunque se caiga el repositorio remoto la gente puede seguir trabajando
* Al hacer los distintos repositorio una réplica local de la información de los repositorios remotos a los que se conectan, la información está muy replicada y por tanto el sistema tiene menos problemas en recuperarse si por ejemplo se quema la máquina que tiene el repositorio remoto.
* Permite mantener repositorios centrales más limpios en el sentido de que un usuario puede decidir que ciertos cambios realizados por él en el repositorio local, no son relevantes para el resto de usuarios y por tanto no permite que esa información sea accesible de forma pública. Por ejemplo es muy útil se pueden tener versiones inestables o en proceso de codificación o también tags propios del usuario.
* El servidor remoto requiere menos recursos que los que necesitaría un servidor centralizado ya que gran parte del trabajo lo realizan los repositorios locales.
* Al ser los sistemas distribuidos más recientes que los sistemas centralizados, y al tener más flexibilidad por tener un repositorio local y otro/s remotos, estos sistemas han sido diseñados para hacer fácil el uso de ramas (creación, evolución y fusión) y poder aprovechar al máximo su potencial. Por ejemplo se pueden crear ramas en el repositorio remoto para corregir errores o crear funcionalidades nuevas.

**1.3 Funcionamiento**

Todos los sistemas de control de versiones se basan en disponer de un repositorio, que es el conjunto de información gestionada por el sistema. Este repositorio contiene el historial de versiones de todos los elementos gestionados.

Cada uno de los usuarios puede crearse una copia local duplicando el contenido del repositorio para permitir su uso. Es posible duplicar la última versión o cualquier versión almacenada en el historial. Este proceso se suele conocer como check out, para modificar la copia local existen dos semánticas básicas:

* ***Exclusivos***: Para poder realizar un cambio es necesario marcar en el repositorio el elemento que se desea modificar y el sistema se encargará de impedir que otro usuario pueda modificar dicho elemento.
* ***Colaborativos***: Cada usuario se descarga la copia, la modifica, y el sistema automáticamente combina las diversas modificaciones. El principal problema es la posible aparición de conflictos que deban ser solucionados manualmente o las posibles inconsistencias que surjan al modificar el mismo fichero por varias personas no coordinadas. Además, esta semántica no es apropiada para ficheros binarios.

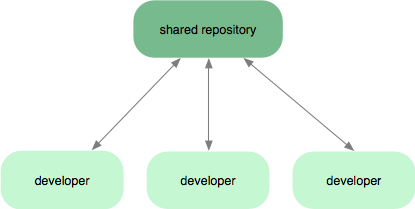
**1.4 Flujos de trabajo**

El flujo de trabajo de un sistema de control de versiones indica cómo se relacionan los distintos usuarios para colaborar entre sí en la consecución de los objetivos del proyecto

En los sistemas de control de versiones centralizados el diseño del sistema restringe la forma en que los distintos usuarios colaboran entre sí. Sin embargo en los sistemas de control de versiones distribuidos hay mucha más flexibilidad en la forma de colaborar ya que cada desarrollador puede tanto contribuir como recibir contribuciones. Hay distintos flujos de trabajo, donde cada uno se adapta mejor a cierto tipo de proyectos.

### 1.4.1 Flujo de trabajo centralizado

En el flujo de trabajo centralizado cada desarrollador es un nodo de trabajo. Por otro lado hay un repositorio remoto central que funciona a modo de punto de sincronización. Todos los nodos de trabajo operan en pie de igualdad sobre el repositorio remoto central

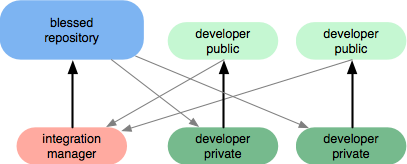


### 1.4.2 Flujo con un gestor de integraciones

En el flujo de trabajo con un Gestor de Integraciones en el que cada desarrollador tiene acceso de escritura a un repositorio propio público y acceso de lectura a los repositorios públicos de todos los demás usuarios. Por otro lado hay un repositorio canónico, representante 'oficial' del proyecto.

Para contribuir en estos proyectos cada desarrollador crea su propio clon público del repositorio canónico y envía sus cambios (realizados en un repositorio privado) a él. Para 'subir' sus cambios al repositorio canónico cada desarrollador tiene que realizar una petición a la persona gestora del mismo.

La principal ventaja de esta forma de trabajar es que puedes continuar trabajando, y la persona gestora del repositorio canónico podrá recuperar tus cambios en cualquier momento. Las personas colaboradoras no tienen por qué esperar a que sus cambio sean incorporados al proyecto, cada cual puede trabajar a su propio ritmo

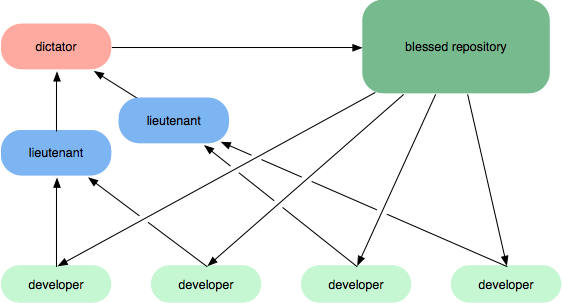


### 1.4.3 Flujo de trabajo con Dictador y Tenientes

El flujo de trabajo con Dictador y Tenientes es una ampliación del flujo de trabajo con gestor de integraciones y que es utilizado en proyectos muy grandes, cientos de colaboradores, como el kernel de Linux.

Hay una serie de gestores de integración que se encargan de partes concretas del repositorio, a los que se denominan tenientes. Todos los tenientes rinden cuentas a un gestor de integración; conocido como el dictador. El dictador integra todos los aportes de los tenientes publicando el trabajo en un repositorio de referencia del que recuperan todos los colaboradores

Este sistema de trabajo permite al líder del grupo (el dictador) delegar gran parte del trabajo en los tenientes, relegando su trabajo en recolectar el fruto de múltiples puntos de trabajo.



### 1.5 Uso de ramas

Las ramas, en un sistema de control de versiones, constituyen una potente herramienta que flexibiliza la forma en la que los colaboradores cooperan en el proyecto.

### 1.5.1 Ramas de largo recorrido

En algunos proyectos se tiene varias ramas siempre abiertas, que indican diversos grados de estabilidad del contenido. Por ejemplo es frecuente mantener en la rama 'master' únicamente lo que es totalmente estable. Luego se tienen otras ramas que revelan distintos grados de estabilidad. Por ejemplo podríamos tener un rama 'beta' (versión beta) y otra 'alfa' (versión alfa), en las que se va trabajando. Cuando se alcanza cierto grado de estabilidad superior a la rama en la que se está entonces se realiza una fusión con rama de estabilidad superior.

### 1.5.2 Ramas puntuales

Las ramas puntuales, también llamadas ramas de soporte, son ramas que se crean de forma puntual para realizar una funcionalidad muy concreta. Por ejemplo añadir una nueva característica se las llama ***feature branch*** ó corregir un fallo concreto se las llama ***hotfix branch***.

Este tipo de ramas permiten trabajar centrándonos exclusivamente en el desarrollo de una característica concreta y cuando esta esté concluida se fusiona con una de las ramas de largo recorrido. La fusión sólo se realiza cuando se está 'seguro' de que esa característica está correctamente implementada, en lugar de fusionar en el orden que se van desarrollando las cosas. Esto permite por un lado tener un historial de las distintas versiones que se han tenido hasta conseguir la funcionalidad. Por otro lado permiten que el historial de las ramas de largo recorrido no sea 'ensuciados' con distintas modificaciones relativas a una funcionalidad concreta.

El uso de este tipo de ramas permiten más flexibilidad a la hora de probar posibles soluciones. Sólo se fusiona con ramas de largo recorrido una vez que estamos seguros de elegir la solución mejor.

Un tipo de ramas de este tipo que tienen un funcionamiento especial son las llamada ***release branch***. Este tipo de ramas se crear para dar soporte a la preparación de una nueva versión de producción. Permiten tener bajo control el contenido de la versión y poder cierto mantenimiento sobre ella (añadirle pequeñas mejoras y corrección de errores).

Es frecuente y una buena práctica utilizar en el nombre de la rama un prefijo que indique el tipo de rama de la que se trata. Por ejemplo podría usar '***feature-***' para ramas de nueva funcionalidad, ***'hotfix-***' para ramas que arreglan errores, y '***release-***' para ramas de release.

**Elección de herramienta para el control de versiones**

Debido a las condiciones del proyecto se opta por elegir una herramienta de control de versiones del tipo distribuido, de tal forma que se pueda dar de manera flexible la incorporación de diversos equipos tanto internos como externo, debido a que la incorporación de sus códigos puedan darse de manera controlada al ser un clon de repositorio y no depender de un control de versiones centralizado. La herramienta concreta a utilizar dentro del proyecto SIIF es ***Git*** la cual resulta de la experiencia del diseñador de Linux (Linus Torvalds) manteniendo una enorme cantidad de código distribuida y gestionada por mucha gente, que incide en numerosos detalles de rendimiento, y de la necesidad de rapidez en una primera implementación.

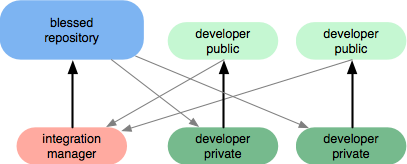
Cuando utilizar un control de versiones distribuido

* No disponemos de conexión continua o la conexión es muy lenta.
* Para equipos muy grandes, con poca comunicación o en distintas localizaciones geográficas.
* Cuando miembros del equipo tienen que viajar mucho y no pueden dejar el desarrollo.
* Cuando queremos que cualquier persona colabore con el proyecto, sin necesidad de que pertenezca a la organización o tenga permisos de acceso (en estos casos para subir los se suele hacer a base de parches).
* Cuando los cambios sobre el código son muy grandes (con Git es muy fácil trabajar con branchs y merge).
* Para trabajar con nuevos modelos descentralizados (se podría hacer, por ejemplo, una jerarquía de repositorios donde los cambios fueran subiendo de las hojas del árbol a la raíz en base de aprobaciones y revisiones por los nodos intermedios).



**Elección de Flujo de trabajo**

### La elección de flujo de trabajo es *"Flujo con un gestor de integraciones"*, actualmente dentro de los grupos de desarrollo existe un miembro el cual es denominado lider del equipo, el cual deberá fungir como gestor de integraciones, este flujo podría cambiar de forma flexible al de *"Flujo de trabajo con Dictador y Tenientes"*, en cuanto otros equipos internos externos sean integrados al producto.



**Uso de ramas**

A continuación se muestra el modelo de branching a seguir, en el se hace uso de ambos tipos de ramas "**ramas de largo recorrido**" las cuales se pueden identificar en la imagen como **"develop"** y **"master"**, dejando claro que dentro de la rama master solo será alojado codigo100% productivo, las ramas puntuales a utilizar serán **"release-branch", "feature-branch" y "hotfix-branch"**.

**Ramas de largo recorrido**

* **develop.** Esta rama representa la constante evolución del producto es la rama de desarrollo constante, siempre debe ser creada a partir del master.
* **master.** Esta rama siempre contiene el código fuente libre de errores, siempre que se necesite realizar un deployment en ambiente de producción debe realizarse a partir de esta rama, no crea de ningún lado pero solo puede ser fusionada con las ramas de "release" y "hotfix".
* **hotfix.** Esta rama es creada cuando se desea atender un error surgido en un entorno productivo es creado a partir de master, al resolver el problema y estar seguro de que el bug fue resuelto debe ser fusionado a las ramas de "develop" y "master".
* **release.** Esta rama puede ser creada a partir de "develop", surge por la customización de un cliente en particular, al terminar dicho desarrollo,

**Modelo de branching**