**Trabajo Práctico Nro. 4**

**“Gestión de Configuración”**

**Herramientas de Control de Versiones**

**Subversion:**

Subversion es un servidor de control de versiones centralizado. Está pensado para que varias personas puedan trabajar bajo el mismo proyecto sin interferirse mutuamente.

Contiene un histórico de versiones que facilita la recuperación de codificaciones antiguas en el caso de que fuera necesario hacer uso de código ya modificado. También permite gestionar distintas versiones de la misma aplicación o proyecto, “trunks”, “branches”.

Para trabajar con Subversion, se suelen realizar las implementaciones sobre el equipo local, no sobre el servidor.

El costo de complejidad de la creación de estas ramas o etiquetas es constante, no lineal. Las modificaciones se realizan de forma atómica (o se actualiza el fichero entero o no se actualiza nada). Permite bloquear los ficheros que se quieren modificar, de esta manera sólo una persona al mismo tiempo estará modificando la misma clase y así se evitan problemas de incoherencias. En el caso de que exista algún tipo de incoherencia, permite descargar las modificaciones que hay en el servidor pero mantiene en el directorio local un versionado con las últimas revisiones. De esta forma se puede valorar qué es el código relevante antes de actualizar el servidor. Las revisiones sólo se realizan sobre los cambios o modificaciones, no sobre la totalidad del fichero.

En Subversión, las ramas se crean a partir de un enlace a un número de revisión del código y las tags o etiquetas no son más que ramas a las que no se le añaden cambios.

Los cambios se realizan sobre el servidor, dejando constancia de todos ellos sobre un histórico y son identificados de 3 maneras:

- Usuario.

- Número de revisión.

- Comentarios en “commit”.

**Comandos básicos y forma de trabajo de Subversion**

**svn help: muestra**opciones generales y una lista de órdenes básicas.

**svn import: es una forma rápida de copiar una jerarquía de archivos sin versionar a un repositorio.**

**svn checkout:** para comenzar a usar el repositorio Subversion.

**svn copy:** crea una copia local.

**svn update:** actualiza una copia local.

Para hacer cambios:

**svn add:** agrega algo al repositorio.

**svn delete:** elimina algo del repositorio.

**svn copy:** crea un nuevo item como duplicado de otro element del repositorio.

**svn move:** mueve un fichero o directorio en su copia local de trabajo o en el repositorio.

Para examinar cambios realizados:

**svn status:** ver los cambios

**svn diff:** ver detalles de los cambios.

Posiblemente revertir algunos cambios:

**svn revert:** permite deshacer los cambios hechos en la copia local.

Resolver conflictos (merge con los cambios de otros):

**svn update:** actualiza la copia local.

**svn resolve**: resuelve conflictos en archivos o directorios de la copia local.

Commit de tus cambios:

**svn commit ­m ”Mensaje”:** envía los cambios de la copia local al repositorio.

**Mercurial:**

Mercurial es un servidor de control de versiones distribuido, esto significa que cada desarrollador dispone de una copia completa del repositorio, permitiéndole el manejo de las distintas versiones sin conexión.

Su rendimiento es alto en cuanto a la velocidad de ejecución de comandos y operaciones, y bajo en la gestión de espacio en disco ya que siempre Mercurial está estructurado de tal forma que siempre se añaden objetos al repositorio.

El acceso al repositorio se puede controlar de tres maneras: mediante una cuenta SSH compartida, mediante un servidor apache con soporte para el protocolo HTTPS y scripts .cgi y mediante un grupo de usuarios del sistema de archivos.

A efectos operacionales, Mercurial trata todos los cambios en el código como una serie de branches y merges.

Existen 4 maneras de trabajar con branches en mercurial:

- La manera más sencilla  pero más lenta de realizar branches es creando nuevos clones del repositorio.

- Otra forma es utilizar tags o bookmarks. Esta estrategia permite realizar un seguimiento del desarrollo de manera ligera y rápida.

- La tercera posibilidad es mediante named branches. En este caso se crean ramas de desarrollo independientes. Se considera que para cambios importantes es uno de los métodos más recomendables pero para pequeños cambios  supone demasiado gasto de recursos.

- La última manera de crear branches es la más sencilla y rápida. Simplemente consiste en hacer updates y commits. Con esta estrategia simplemente se establece una rama sin especificar ningún nombre.

Los “merges”, en este caso, no son simétricos, por lo que establecerá la rama más antigua como la principal y a esta unirá la más actual. Para hacer un merge, se utiliza el comando **hg merge**. En ese momento Mercurial recupera los contenidos de cada versión HEAD y los une en el directorio de trabajo, dejando constancia de cada uno de ellos hasta que se realice un commit. En caso de encontrar conflictos, Mercurial busca algún software para hacer merge automáticamente instalado en el sistema ya que no cuenta con ningún módulo para la gestión de estos conflictos. Si no lo encuentra o se requiere la interacción del usuario, intenta lo mismo con programas con entorno visual.

En Mercurial el seguimiento de los cambios se realiza en el repositorio y en un solo archivo, minimizando los costos de acceso a disco ya que no necesita buscar el archivo en cada directorio.

Para identificar revisiones existen 3 criterios que se pueden seguir:

-  Según el número (decimal) de revisión que se corresponde con el orden de los commits en el repositorio local.

- Mediante el changeset ID que identifica la posición de la revisión en cualquier momento con respecto a la historia total del proyecto.

- Según tags que añaden contenido semántico al changeset ID.

Mercurial es multiplataforma y fácilmente portable.

Estos servidores son diametralmente opuestos, ya que están orientados a dos tipos de proyectos distintos. Para proyectos más pequeños y centralizados en cuanto a localización de los desarrolladores, se considera más oportuno el uso de Subversion o svn. Para proyectos más grandes y con un mayor grado de distribución en cuanto a localización de los desarrolladores es más recomendable el uso de Mercurial. Por lo tanto, a la hora de elegir entre un sistema u otro se debe tener bien definido el proyecto.

**Comandos básicos y forma de trabajo de Mercurial**

$ **hg help**: órdenes básicas.

$ **hg init nombre-repositorio**: para crear un repositorio.

$ **hg add**: para agregar archivos a un repositorio.

$ **hg status**: para conocer el estado de los archivos del directorio de trabajo. Los estados pueden ser M (modificado), A (marcado para ser añadido al repositorio), ? ( el archivo no está siendo seguido por Mercurial).

$ **hg commit**: guardar ese conjunto de cambios.

$ **hg log**: bitácora de cambios.

$ **hg update *revisión****:* permite realizar la actualización de un repositorio.

$ **hg diff -r version1 -r versión2**: Se pueden ver las diferencias entre dos versiones.

$ **hg clone repositorio-original repositorio-copiado**: Se puede clonar un repositorio, de manera que sea posible recuperar datos de su “original” para integrar los cambios en uno u otro sentido.

$ **hg push repositorio-destino**: integra en un repositorio remoto los conjuntos de cambios del repositorio actual que no se encuentren en él.

$ **hg pull repositorio-origen**: integra en el repositorio local los cambios adicionales que contenga el repositorio remoto.

$ **hg merge changset**: para reintegrar los cambios de una rama en otra rama. Changset es la revisión que queremos integrar en la rama actual.

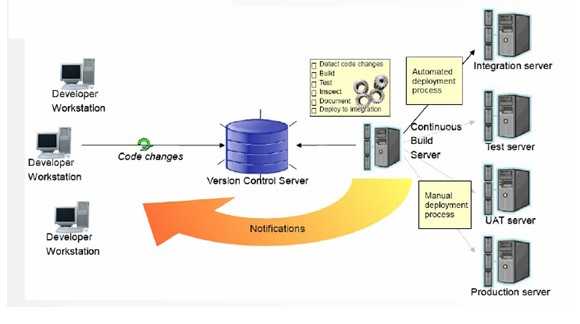
**Herramienta de Integración continua: Apache Gump**

**Apache Gump** es un sistema de integración continua, desarrollado en Python bajo licencia Apache 2.0, que tiene como objetivo construir y probar todos los proyectos Java de código abierto, todas las noches. Su objetivo es asegurarse de que todos los proyectos sean compatibles, tanto a nivel de API y como a las especificaciones de funcionalidad. Está alojado en gump.apache.org , y se ejecuta cada noche en la oficial JVM de Oracle.

Los resultados de la integración son capturados en páginas HTML, con las fallas resaltadas en color rojo, y pueden ser enviadas mediante e-mail a las personas correspondientes. Para ver un ejemplo, ingresar a http://vmgump.apache.org/gump/public/.

Las herramientas de control de versión que admite son Subversion y CVS. De éstas toma la última versión desde el repositorio al momento de la integración.

¿Cómo funciona?



El proceso normal que se sigue en este esquema de trabajo CI es como sigue:

1. Los desarrollares envían sus modificaciones al controlador de versiones (SVN, CVS, etc).
2. El Servidor de Integración Continua monitorea el repositorio buscando cambios y ejecuta automáticamente el build.
3. Una vez finalizado el proceso de build (integración, construcción, pruebas y despliegue), el servidor envía a los responsables un email con el resultado del proceso (feedback del proyecto).
4. El servidor realiza el paso 2 continuamente.

El proceso es simple pero en muchos proyectos no se realiza esto que es tan elemental y que a la larga mejora el proceso de desarrollo y evita problemas.

Las ventajas:

* Los problemas de integración son detectados rápidamente y pueden ser corregidos con la misma celeridad, no hay por qué integrar 1 vez al mes y estar corriendo contra el reloj para detectar problemas.
* Código que rompa el build es detectado automáticamente al tener un feedback automático.
* Las pruebas unitarias corren cada vez que se ejecuta el build lo cual permite que el desarrollador pueda darse cuenta rápidamente de cualquier error en su código.
* Existe un mecanismo de despliegue automatizado donde el código producido puede ir a parar a los distintos servers (development server, testing server, production server).

**Resultados a obtener**

* Control de cambios
* Monitoreo constante
* Integración de procesos
* Retroalimentación
* Gestión de dependencias
* Proceso de desarrollo en base a Métricas
* Finalmente, **disminuir el tiempo de desarrollo requerido para cada proyecto.**

La integración continua impacta en cada fase del proceso de la siguiente forma:

En el ensamble:  
• Generación automática de versiones.  
• Integración silenciosa.  
• Detección automática de errores.  
• Integración continua de versiones.

En pruebas:  
• Detección rápida y temprana de errores.  
• Pruebas de regresión automáticas.  
• Verificación del cubrimiento de las pruebas unitarias.  
• Integración con herramientas de pruebas (unitarias, de carga, stress, funcionales).  
• Trazabilidad con sistemas de bug tracking.

En control de calidad:  
• Monitoreo de estándares de codificación.  
• Monitoreo de malas prácticas.  
• Monitoreo de métricas de calidad.

En despliegue:  
• Automatización de despliegue de aplicaciones en ambiente de desarrollo y pruebas.