Índice de contenido

Introducción....1
Configuración Subversion....2
Crear el repositorio del SVN que almacenará el proyecto....2
Establecer la seguridad del repositorio....3
Configuración NetBeans....4
Conexión con el Subversion....4
Tests unitarios con JUnit en NetBeans....6
Configuración del Ant....8
Configuración Hudson....11
Automatización de la construcción....11
Configurar el workspace del Hudson y añadir dependencias externas al Subversion....13
Comprobando que el job está bien configurado....15
Conexión con el SVN....16

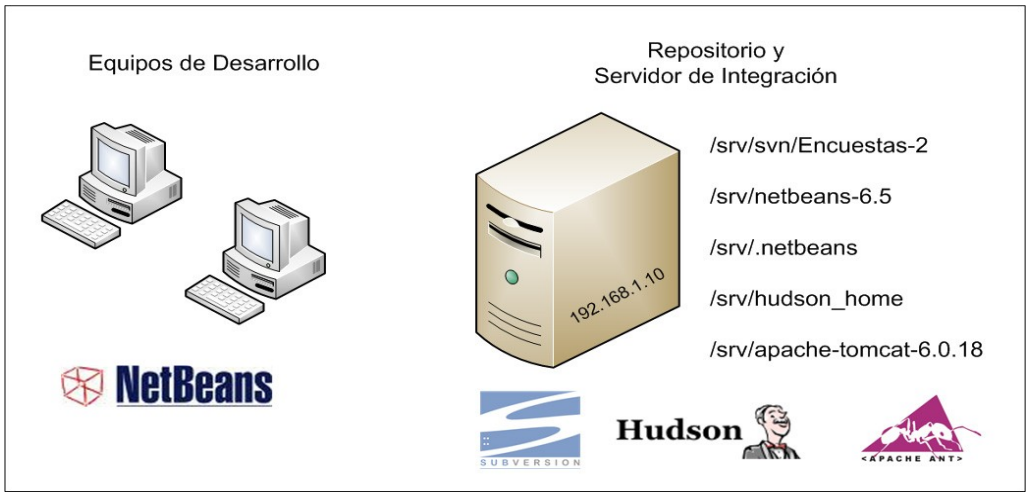
Introducción

Cuando uno lee posts, libros y escucha experiencias de la aproximación ágil al desarrollo del software y empieza a profundizar y a aprender conceptos tales como el TDD (test driven development) y la integración continua, se le despiertan una ganas irrefrenables de empezar a trabajar de este modo. En este post lo que intentaré explicar es cómo montar un entorno sencillo que nos permita tener los cimientos de un sistema ágil que cada uno podrá extender para satisfacer sus propias necesidades. Como se menciona en este post, tres de los pilares de la integración continua son: un repositorio de datos, un servidor de integración continua y una herramienta que permita automatizar la construcción de la aplicación. Yo además añadiría un cuarto: un IDE que se integre bien con el resto del entorno. No es el propósito de este post discutir la utilidad de estos componentes (el post mencionado anteriormente puede ser un buen punto de partida para los recién llegados), sino elegir unas herramientas concretas y describir su configuración e integración. En mi caso he hecho la siguiente elección (que no tiene por qué ser la única ni la mejor):

- repositorio de código: SVN
- servidor de integración continua: Hudson
- herramienta de construcción: ANT
- IDE: NetBeans 6.5

Por simplicidad he distribuido el entorno entre una máquina Linux (Ubuntu 8.10) con IP 192.168.1.10 (dato que es relevante para tareas de configuración que iremos viendo) que actúa simultáneamente como repositorio de datos y de integración continua y un número arbitrario de máquinas de desarrollo (también Ubuntu 8.10) que contienen los IDE's y el entorno con el que trabajan los desarrolladores. En cualquier caso, otras distribuciones tienen exactamente la misma complicación y los pasos requeridos son directamente extrapolables y fácilmente adaptables. Para explicar los diferentes pasos utilizaré un mismo ejemplo ya que considero que hablar con algo concreto suele ser más sencillo, tanto para el que explica como para el que intenta entender, que hacerlo en general o en abstracto. Presupongo la inteligencia de los potenciales lectores para hacer las adaptaciones requeridas en cada punto a sus propias necesidades. No es muy relevante, pero utilizaré un proyecto personal que estoy empezando estos días que sirve para hacer un seguimiento de las valoraciones de mis cursos y que se titula, en un alarde de originalidad, "Encuestas".

En el siguiente diagrama muestro el entorno que acabaremos montando. Las rutas que aparecen en el servidor no tenemos por qué entenderlas ahora pero servirán como referencia a medida que vayamos construyendo la solución.



Configuración Subversion

He elegido Subversion porque es un repositorio más o menos potente que supera algunas de las limitaciones de nuestro querido y viejo CVS y aún a día de hoy es una de las herramientas que, bajo mi humilde opinión, están más extendidas en la industria. No voy a entrar en los detalles de instalación y de configuración del propio Subversion. Para ello ya hay mucha documentación online (por ejemplo aquí). Me centraré en la creación de un repositorio para nuestro proyecto. Es cierto que se puede usar el mismo repositorio para más de un proyecto, pero yo prefiero tenerlo separado (alguno de los motivos aparecerán más adelante, aunque también se puede justificar razonadamente la otra postura).

Crear el repositorio del SVN que almacenará el proyecto

Como decía, presupongo que el servidor que actuará como repositorio de datos tiene instalado el Subversion y sus herramientas administrativas (si no es así, nuestro querido *apt-get* o *Synaptic* vendrán a nuestro rescate). Lo primero que hay que hacer es determinar en qué ruta almacenaré el repositorio. Yo tiendo a almacenar los diferentes repositorios de cada uno de los proyectos bajo una misma raíz y así mantengo el servidor más o menos organizado. La ruta también debería ser vistosa porque luego formará parte de la configuración que utilicen los clientes (como los IDE's) para conectarse con dicho repositorio. Trabajando con Ubuntu, y si hemos instalado el Subversion desde paquetes, se crea un usuario *svn*. Para que todo funcione de manera correcta, deberíamos asegurarnos que el propietario del repositorio es este usuario. Una manera de hacerlo es ejecutar los comandos de administración del Subversion con este usuario y otro es cambiar después el propietario del directorio (y su contenido) con el comando *chown* (chown -fR svn .)/Encuestas-2). En definitiva, para crear el repositorio haremos lo siguiente:

```
svnadmin create /srv/svn/Encuestas-2
```

El directorio */srv/svn* es la raíz de todos mis repositorios y *Encuestas-2* es el que usaré para este proyecto en particular. Si ahora nos situamos en en el ese directorio y hacemos un listado deberíamos ver (más o menos) lo siguiente:

```
ivan@hargon:/srv/svn/Encuestas-2$ ls -alh
total 36K
drwxr-xr-x 7 svn root 4,0K 2009-01-11 10:43 .
drwxr-xr-x 6 svn root 4,0K 2009-01-11 10:43 ..
drwxr-xr-x 2 svn root 4,0K 2009-01-11 10:43 conf
drwxr-xr-x 2 svn root 4,0K 2009-01-11 10:43 dav
drwxr-sr-x 5 svn root 4,0K 2009-01-11 10:43 db
-r--r--r-- 1 svn root    2 2009-01-11 10:43 format
drwxr-xr-x 2 svn root 4,0K 2009-01-11 10:43 hooks
drwxr-xr-x 2 svn root 4,0K 2009-01-11 10:43 locks
-rw-r--r-- 1 svn root  229 2009-01-11 10:43 README.txt
```

Establecer la seguridad del repositorio

Cuando se crea un repositorio en el Subverson es, obviamente, para que los desarrolladores puedan utilizarlo. Por tanto un paso importante es la parte de securización y autorización del mismo: establecer quién podrá acceder y sus credenciales. Subversion tiene diferentes mecanismos de autenticación y de transporte seguro de la información (incluyendo *ssh*). No voy a entrar en los detalles y explicaré cómo hacerlo de la manera más sencilla. Para más información os remito al anterior enlace.

En mi caso, todas las máquinas residen en el interior de mi red de área local, que puede considerarse "zona segura". Un punto importante es que los mecanismos de seguridad (y otros más) se establecen por repositorio (en contraposición a una configuración global para todos ellos) lo que nos permite que convivan repositorios con diferentes configuraciones (este es uno de los motivos por los que prefiero tener un repositorio por proyecto). Lo primero es determinar el modo de autenticación. Para ello habremos de editar el fichero

```
/srv/svn/Encuestas/conf/svnserve.conf
```

para que quede más o menos así:

```
[general]
anon-access = none
auth-access = write
password-db = passwd
realm = My Realm
```

De hecho estas opciones seguramente preexisten en el fichero generado en el repositorio y lo único que habrá que hacer es descomentarlas. También es posible que en el fichero haya otras tantas opciones comentadas que no vamos a necesitar para nuestra configuración básica, así como diferentes comentarios descriptivos. Yo los he obviado por claridad. Con esta configuración lo que estamos haciendo es decirle al Subversion que busque los usuarios y las credenciales en un fichero de texto plano. El siguiente paso, precisamente, consiste en configurar qué usuarios y con qué credenciales podrán acceder al repositorio editando dicho fichero que se encuentra en:

```
/srv/svn/Encuestas/conf/passwd
```

que editaremos para que quede algo parecido a lo siguiente:

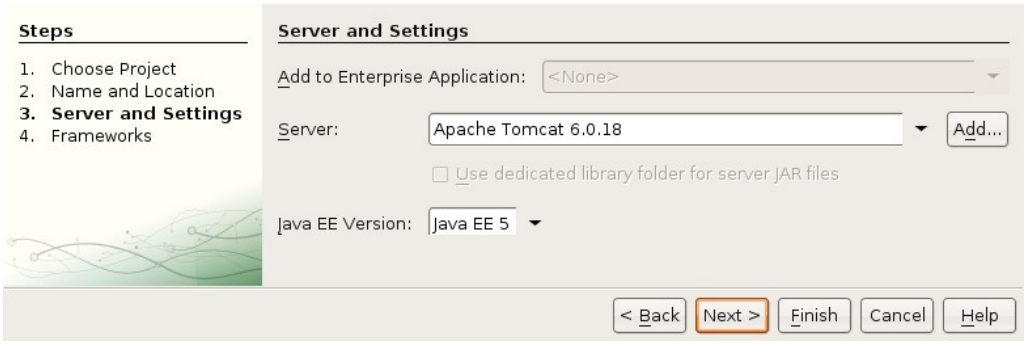
```
[users]
ivan = super-secret-passwd
ana = top-secret-passwd
```

Estos serán los usuarios y passwords que habrá que configurar en los clientes (en nuestro caso usaremos

NetBeans).

Configuración NetBeans

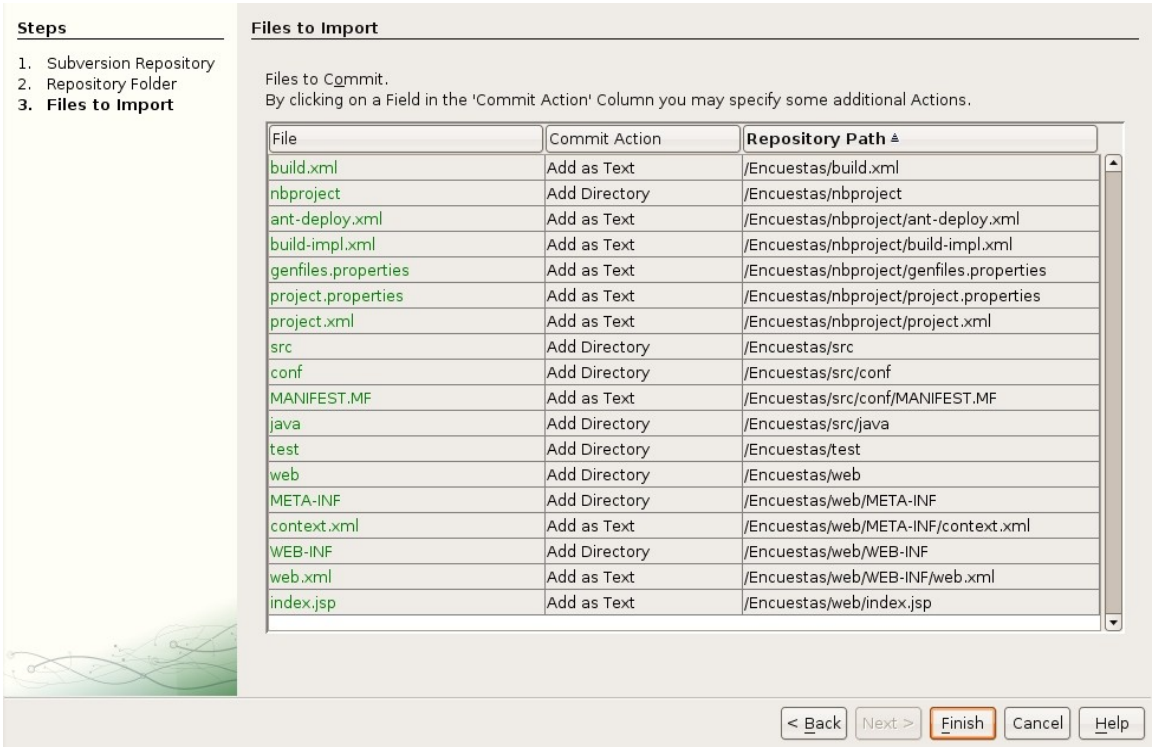
Supongamos que nuestra aplicación de ejemplo *Encuestas* es una aplicación web típica que va a correr encima de un Tomcat 6. Así pues lo primero que haremos es crear un nuevo proyecto web mediante los asistentes del IDE. Asumo que estos pasos los sabréis llevar a cabo pero, en cualquier caso, no son el tema del presente post y lo pasaré muy rápido.



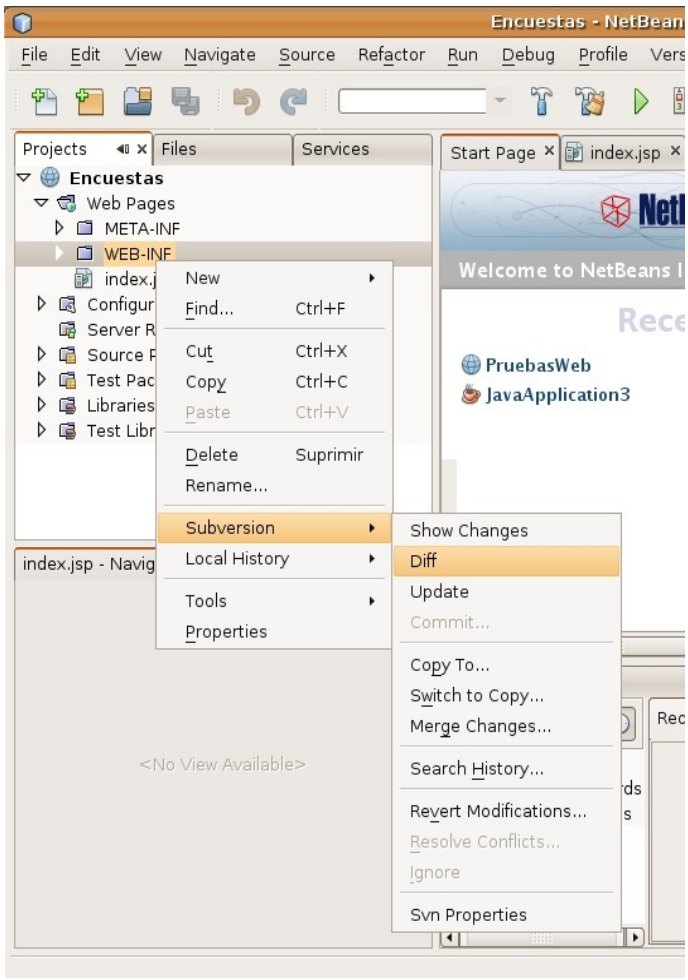
Conexión con el Subversion

Una vez tenemos creado el proyecto web, lo siguiente es enlazarlo con el Subversion. Para ello haremos **click derecho** en el nombre del **proyecto**, y en el menú **versioning** seleccionaremos la opción **Import into Subversion Repository** que nos abrirá un asistente para configurar la conexión. En mi versión de NetBeans (la 6.5) el aspecto que tiene es el que podéis ver en la siguiente captura de pantalla.

Los siguientes pasos nos permiten seleccionar qué ficheros importar al repositorio. En este punto se podría abrir una discusión sobre si sólo subir los ficheros de código y recursos necesarios para el proyecto desde un punto de vista de independencia del IDE utilizado o si bien subir también los ficheros y recursos asociados al IDE. Puede ser una discusión interesante, pero la dejaremos para otro momento. En esta ocasión, para simplificar (y en mi caso es siempre la opción preferida), subiremos todos los recursos tal y como muestra el tercer paso del asistente.



En este momento ya tenemos conectado el IDE con el repositorio. Los pasos e ideas fundamentales son los mismos para otros entornos como Eclipse y CVS. Quizá en otro post puedo explicar cómo configurar alguna de estas otras opciones. Para utilizar el NetBeans como cliente de Subversion tan sólo hay que hacer **click derecho** sobre el recurso correspondiente (un directorio o un fichero) y elegir la opción **Subversion** del menú contextual. La siguiente captura de pantalla lo ilustra.



Tests unitarios con JUnit en NetBeans

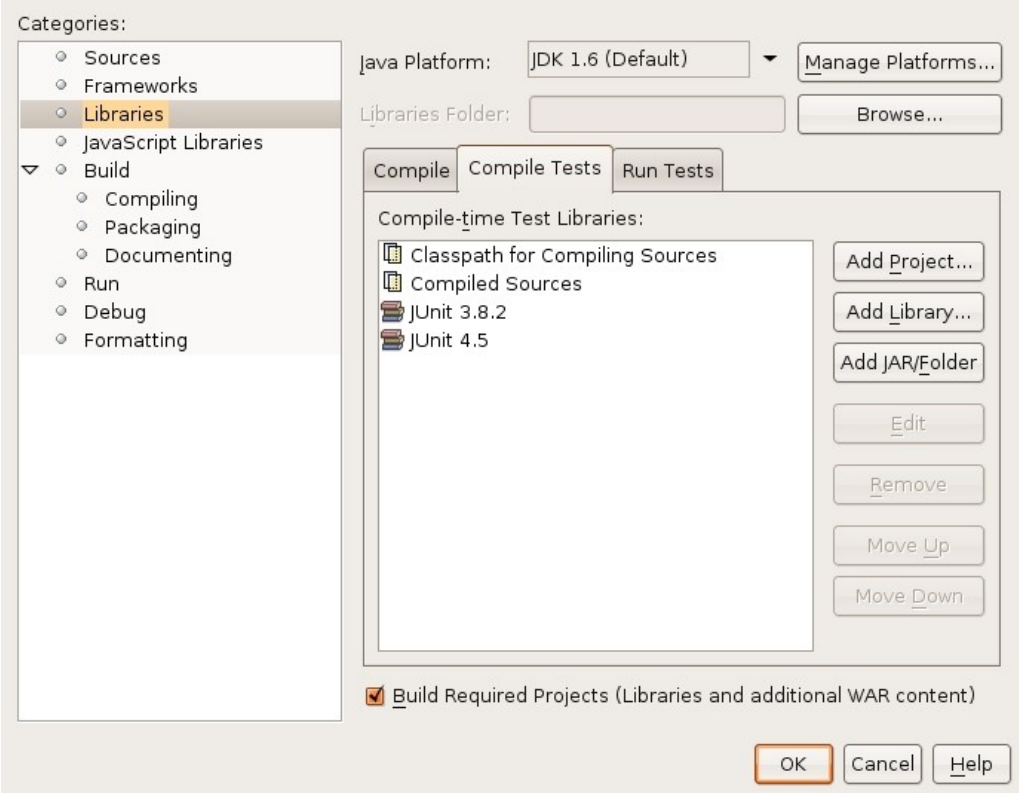
En posts siguientes explicaré cómo integrar en el entorno ágil que estamos montando diferentes herramientas que aumenten nuestra productividad y la calidad de nuestro código (PMD, CheckStyle, FindBugs, herramientas de cobertura...), pero por el momento me centraré en aquellas herramientas mínimas que requiere un entorno ágil que quiera trabajar con TDD: test unitarios. La idea es la siguiente: imaginemos que queremos trabajar con el framework de tests unitarios JUnit. Instalaríamos la herramienta en tres "lugares":

1. Como plug-in del IDE, lo que permite un uso integrado en el entorno de desarrollo del programador, con todas la ventajas de uso que ello conlleva.

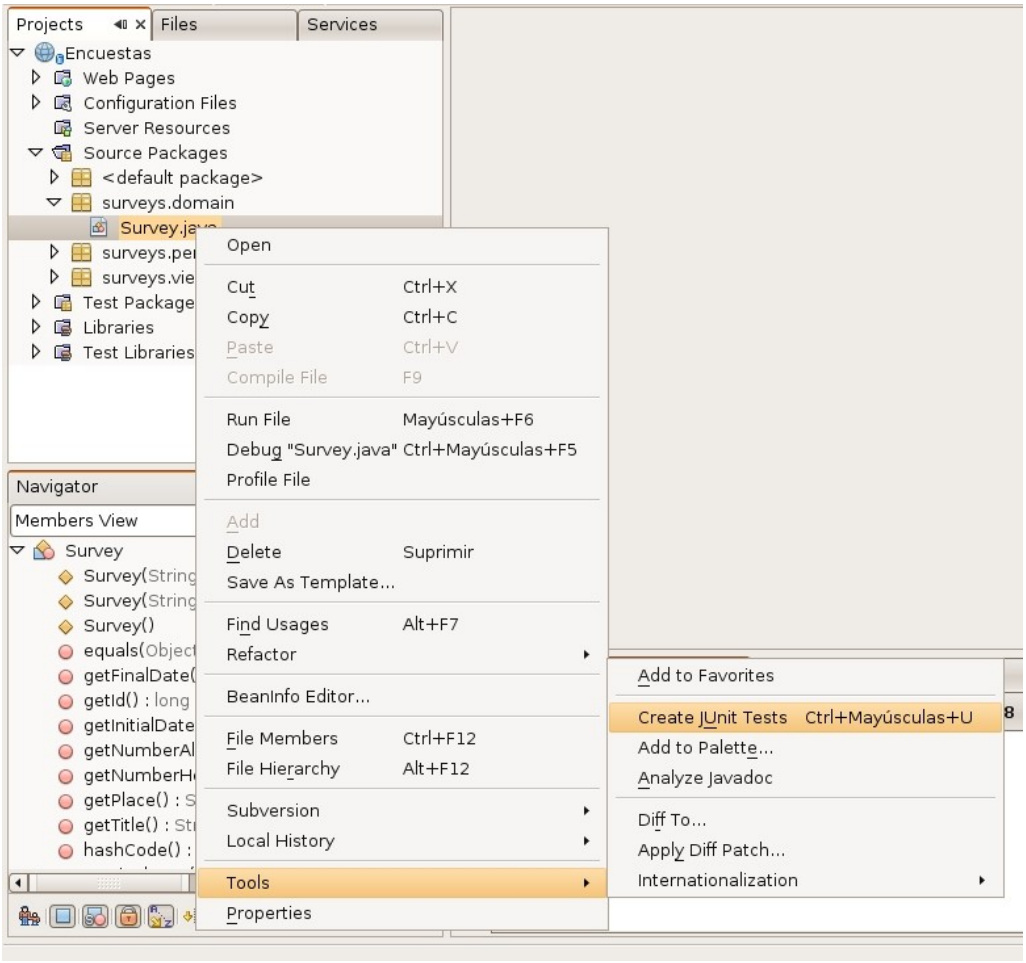
2. Como *target* en el *build.xml* del Ant. Este *target* nos permitirá ejecutar las mismas tareas desde la línea de comandos sin ayuda del IDE pero además, y más importante, permitirá al servidor de integración continua lanzar la tarea y generar los diferentes informes (en el caso de JUnit serán ficheros XML) especificando qué tests han sido superados, cuáles han fallado, etcétera.

3. Como plug-in (que a veces puede venir preinstalado) del servidor de integración continua (en nuestro caso Hudson), de manera que pueda utilizar los informes generados en el *target* del Ant para mostrar información útil y gráfica y determinar si la construcción ha fallado o no entre otras cosas.

En este apartado veremos el primer punto y el resto los iré desarrollando a lo largo del post. También os aconsejo que leáis este magnífico post sobre cómo usar JUnit en NetBeans del propio blog oficial del IDE. El plug-in de JUnit ya está instalado por defecto en la mayoría (si no en todas) las distribuciones de NetBeans como IDE Java. Incluye además las dos versiones que se utilizan hoy día (la familia 3.X y la familia 4.X). Yo personalmente me decanto por la versión 4 porque me gusta poder utilizar anotaciones. Al crear cualquier proyecto Java con los asistentes de NetBeans, éste ya está preconfigurado para utilizar el framework (aunque nosotros podamos ignorarlo). Lo podéis comprobar en dos sitios: por un lado en la ventana **Projects** podemos ver que se ha creado un directorio **Test Packages** y por otro lado si hacemos **click derecho** sobre el nombre del **proyecto**, y abrimos el diálogo **Properties** en la sección **Libraries - Compile Test** veremos que están están agregadas las bibliotecas correspondientes.



La manera de construir un test unitario con NetBeans es muy simple. Simplemente tenemos que **seleccionar** la clase que queremos **testear**, hacer **click derecho**, seleccionar **Tools** y finalmente **Create JUnit Tests** que lanzará un menú modal en el que podremos seleccionar qué código autogenerar en el unit test que se va a crear (en el primero unit test que creemos, el entorno nos preguntará qué versión de JUnit querremos usar y ya la mantendrá para el resto del proyecto).



Una vez codifiquemos el test unitario, lo siguiente es poder ejecutarlo; nada más sencillo: **Run -> Test Project** (o la combinación **ALT + F6**) y el resultado indicando si se pasan o no los tests se mostrará por la consola del IDE.

Configuración del Ant

Supongo que en este punto no tengo que explicar qué es Ant y asumo que conocéis su funcionamiento. Si bien trabajando en local, el IDE puede hacernos transparentes todos aquellos aspectos engorrosos como configurar el CLASSPATH, compilar, generar documentación, correr herramientas de análisis, desplegar en el servidor de aplicaciones, etcétera, cuando tenemos que hacerlo de manera remota, o hacer el despliegue en producción, o automatizar cualquiera de estas tareas dentro de un script, o, lo que más nos interesa en este post, permitir a un servidor de integración continua que construya nuestro proyecto, necesitamos una herramienta independiente del IDE. Ant es una opción que nos permite llevar esto a cabo (otra opción, por ejemplo, sería Maven).

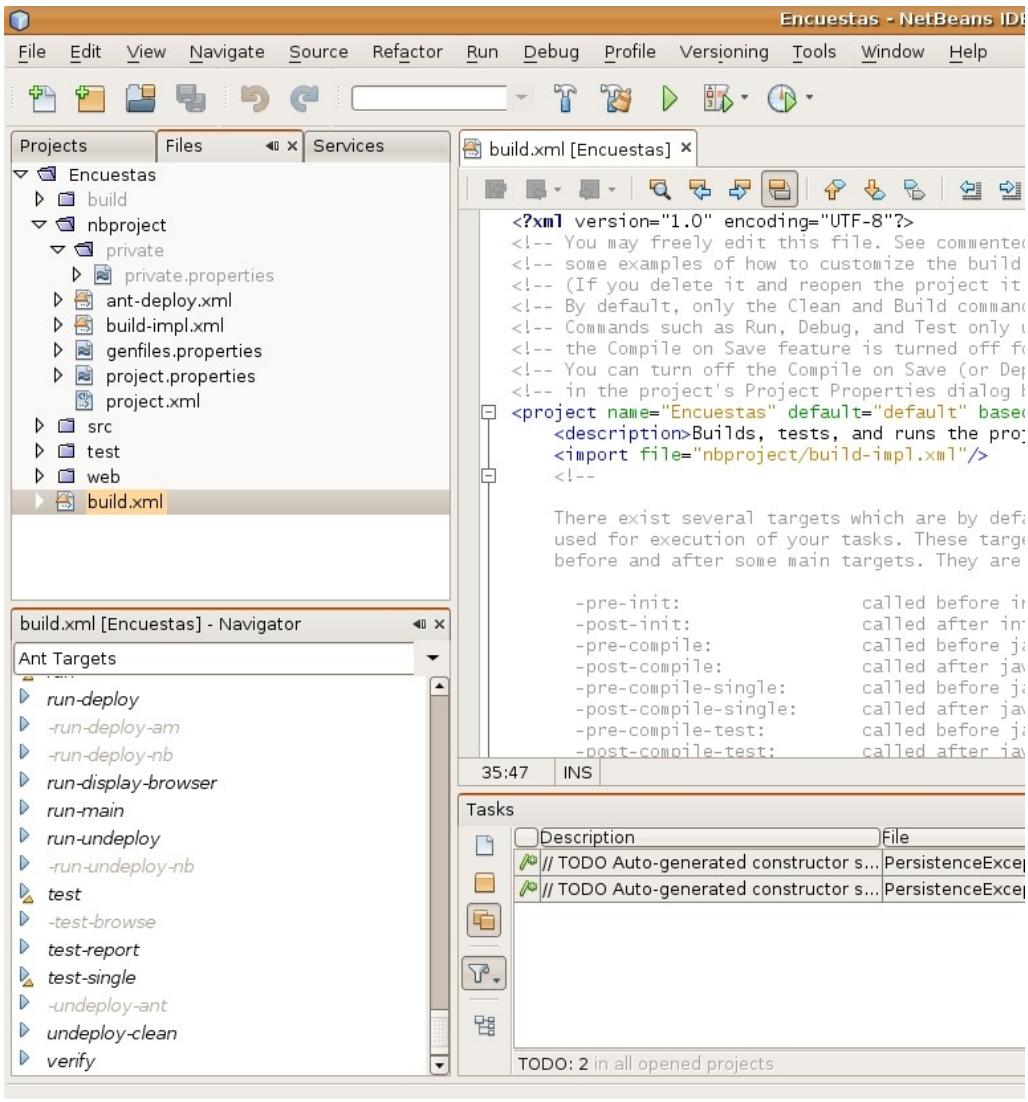
Trabajando con Eclipse, hasta donde llega mi conocimiento, no tenemos más remedio que construir los *build.xml* a mano. De todos modos una vez que se ha hecho uno, si se organiza y parametriza de forma adecuada, el resto de proyectos pueden ir reutilizando una misma “plantilla” con cambios mínimos.

Trabajando con NetBeans podemos tener parte del trabajo resuelto. Ello se debe a que todas las tareas que realiza el IDE (compilar, testear, configurar los diferentes CLASSPATH, desplegar, etc.), lo hace con Ant mediante ficheros *build.xml* (y otros ficheros auxiliares) que construye dinámicamente en función de lo que nosotros configuremos de manera gráfica en la interfaz de NetBeans o a través de cualquiera de sus

asistentes. Nosotros podemos aprovecharnos de estos ficheros generados para utilizarlos desde fuera del IDE para nuestros propios propósitos. La propia estructura de los scripts preveen diferentes *targets* "hook" donde añadir nuestro propio scripting si el generado no es suficiente.

A primera vista esto podría parecer genial y bueno, en gran parte lo es, pero hay que tener en cuenta una serie de consideraciones. La principal es que los scripts generados no son totalmente independientes del IDE ni de la máquina en la que se han generado porque utilizan recursos (bibliotecas y *tasks*) que sólo existen en los directorios de instalación del IDE y algunas de las rutas generadas son absolutas y por tanto dependientes de la máquina en donde se generaron. Por tanto si estos ficheros quieren utilizarse en otro contexto habrá que hacer algunos ajustes a mano. Dependiendo del tipo de proyecto en el que estemos trabajando (no será lo mismo una aplicación JSE que una aplicación J2EE con dependencias de un servidor de aplicaciones) la cantidad de esfuerzo requerida para hacer los scripts portables será mayor o menor. En este post se muestran algunos de los perfiles de aplicaciones y sus posibles problemas y soluciones (se corresponde con una versión de NetBeans algo vieja, pero la mayor parte de lo que se explica todavía aplica a las versiones actuales).

NetBeans genera el *build.xml* en la raíz del proyecto. Si lo abrís podéis ver que básicamente es un script vacío que lo que hace es importar otros tantos ficheros (más o menos dependiendo del tipo de proyecto) que se encuentran en la subcarpeta **nbproject**. NetBeans utiliza esta carpeta como su directorio de trabajo y para contener los diferentes metadatos del proyecto. En principio no deberíamos tocar ninguno de los ficheros contenidos bajo esta jerarquía de directorios a no ser que sepamos exactamente lo que estamos haciendo. Dentro de nbproject tenemos otra carpeta especial: **private**. Deberemos ir con cuidado de no incluir esta carpeta en el repositorio (por defecto el asistente ya lo hace bien) ya que contiene ficheros de parámetros con paths absolutos y dependientes de la máquina de cada desarrollador (o servidor de integración) usados por, entre otras cosas, por los scripts de Ant. Esto será parte de lo que tendremos que replicar a mano en el servidor de integración adaptándolo adecuadamente. (pero ya lo veremos más adelante).



Al abrir el *build.xml* desde el propio NetBeans, la ventana Navigator nos muestra las tareas disponibles. Si queréis detalles, investigad un poco, pero bueno tenemos todos aquellos *targets* que esperaríamos encontrar y que podemos necesitar (y muchos más), por ejemplo:

- clean
- compile
- debug
- dist
- javadoc
- test

Los incrédulos podéis abrir una consola y comprobar como podéis invocar las tareas desde la línea de comandos. Para ello sólo os tenéis que situar en el directorio raíz del proyecto y ejecutar el ant como lo haríais con cualquier otro build.xml.

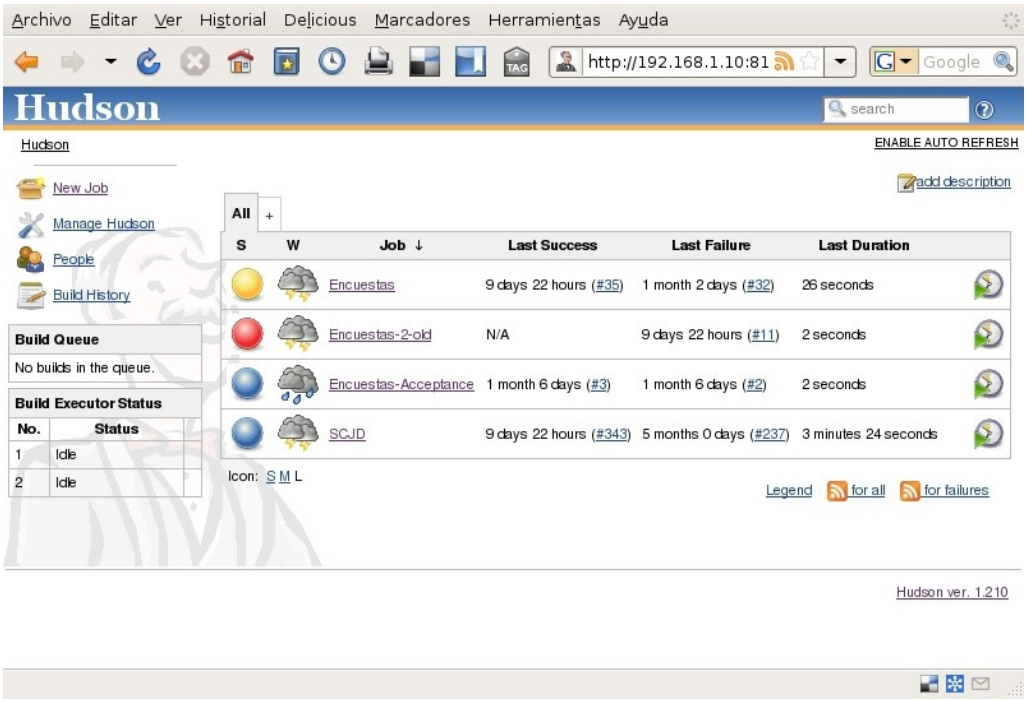
Configuración Hudson

Sólo resta la última pieza del puzle: automatizar la construcción del proyecto desde nuestro servidor de integración continua (Hudson) conectándolo al repositorio. También explicaré como automatizar el hecho que se programe una nueva construcción automática cada vez que hay un *commit* en el repositorio. La instalación y configuración inicial de Hudson es trivial pero está fuera del ámbito de este tutorial (baste decir que básicamente consiste en desplegar un *war* en un servidor de aplicaciones o un contenedor web como Tomcat).

Automatización de la construcción

Hudson automatiza la construcción de proyectos en lo que denomina ***jobs***, así pues un mismo proyecto software puede tener diferentes *jobs*. Por ejemplo, tener configurado un *job* que se ejecute cada vez que se haga un *commit* en el repositorio y que lo único que compruebe es que el proyectos es compilable y que se pasan los test unitarios y tener otro *job* programado para ejecutarse tres veces al día para correr tests de aceptación o de integración más pesados. Nosotros empezaremos por algo muy sencillo, configurando un *job* para que haga lo que explicaba en el primer ejemplo (compilación más tests unitarios).

Hudson tiene una interfaz muy sencilla de utilizar. A diferencia de otros servidores de integración, tanto la creación como la configuración de los *jobs* se hace íntegramente desde la interfaz web.



Para crear nuestro *job* haremos click en **New Job** y se iniciará el asistente de creación. Lo primero que tenemos que hacer es dar un nombre a nuestro *job* que será utilizado tanto en la interfaz e internamente en el servidor de integración. Se puede utilizar cualquier símbolo en el nombre e incluso espacios, pero yo prefiero (manía personal) utilizar nombres con una sola palabra porque cuando luego se quiere navegar "a mano" (mediante la consola) por los *jobs* siempre es más cómodo no tener espacios puesto que el nombre que usemos se corresponderá con el nombre del directorio donde residirá toda la información asociada a dicho *job*. En nuestro ejemplo: **Encuestas-2**. Lo siguiente es elegir el tipo de *job* que estamos construyendo; nosotros construiremos un *job* estándar que es lo que Hudson llama un **Build a free-style software project**. Lo seleccionamos y le damos a **OK** para ir a la siguiente pantalla del asistente. En esta

nueva pantalla tenemos todos los elementos de configuración del *job* que rellenaremos (aquéllos que haga falta) como sigue:

- **Source Code Managment:** Subversion

    - **Repository URL:** svn://192.168.1.10/srv/svn/Encuestas-2/Encuestas
      Aquí hay que tener en cuenta una cosa. La raíz del repositorio es *Encuestas-2* pero configuramos la url hasta *Encuestas*. Si no fuera así, el *workspace* generado para construir este job tendría un subdirectorio *Encuestas* (lo que complicaría la configuración del resto de elementos del *job*)

- **Build Triggers**: permite definir en qué momento Hudson tiene que construir el proyecto. Hay diferentes políticas con las que podéis experimentar. Nuestro propósito para este *job* es que se construya cada vez que hay modificaciones en el repositorio. Para implementar esta política hay dos posibles maneras: o bien Hudson va preguntando al repositorio "¿ha habido cambios en el código?" de forma periódica (lo que se conoce como mecanismo por encuesta y que se corresponde con la opción "poll") o bien que el propio Subversion avise a Hudson cada vez que ha habido un cambio. Esta última aproximación es más eficiente (y a mí parecer más elegante), pero se trata más bien de una configuración del Subversion (que veremos más adelante) y no del Hudson. Así pues no activaremos ningún tipo de *trigger* (lo dejamos tal y como está).

- **Build**: nos permite configurar cómo Hudson debe construir el proyecto. Existen diferentes mecanismos pero nosotros en este post vemos cómo se hace mediante Ant, así que será esta la opción que elegiremos.

    - **Ant version**: dejaremos la "Default" con lo que Hudson utilizará la versión que encuentre en el path (esto está presuponiendo que existe un Ant configurado correctamente en la máquina donde reside Hudson).

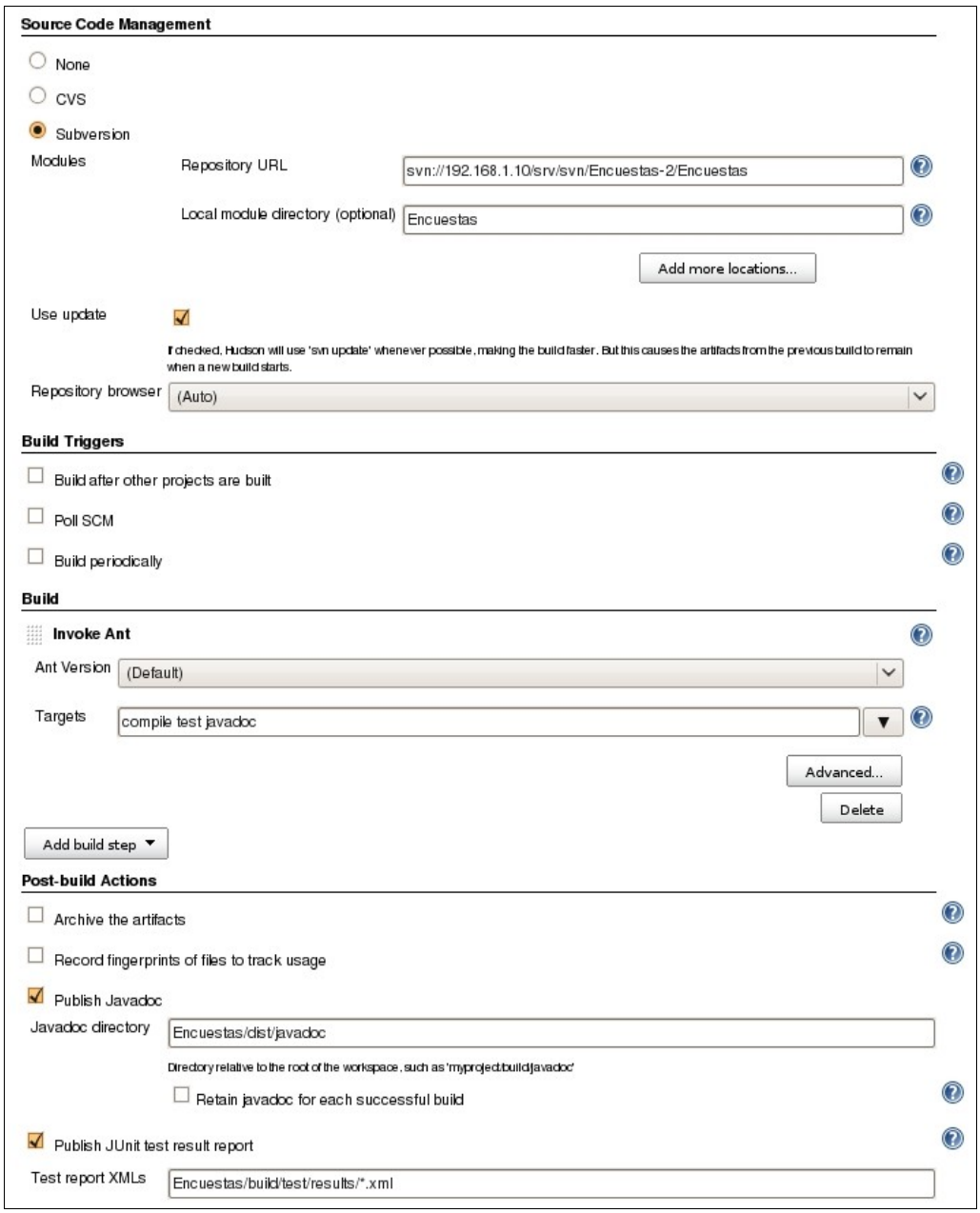
    - **Targets**: compile test javadoc
      Aquí tendremos que especificar los *targets* definidos en el *build.xml* que queremos que Hudson ejecute. Como estamos utilizando el *build.xml* automáticamente generado por NetBeans tendremos que tener en cuenta algunas consideraciones que explico más adelante (como había comentado el *build.xml* generado no es del todo independiente del NetBeans y habrá que poner algún parche para que funcione desde Hudson).

- **Publish Javadoc**: marcamos esta opción para publicar los documentos generados por *javadoc* asociados a nuestro proyecto. Los *javadoc* se generarán porque hemos definido el *target* correspondiente en la sección previa.

    - **Javadoc directory**:Encuestas/dist/javadoc

- **Publish JUnit test result report**: marcamos esta opción para publicar los informes (y gráficas) de los resultados producidos por los diferentes test unitarios creados en el proyecto.

    - **Test report XMLs**: Encuestas/build/test/results/*.xml



Configurar el workspace del Hudson y añadir dependencias externas al Subversion

Hudson almacena todos los ficheros que necesita para su funcionamiento así como los diferentes artefactos para cada uno de los *job* en un directorio raíz conocido como el HUDSON_HOME. No es el propósito de este artículo explicar cómo configurarlo, pero puesto que necesitaremos modificar algunos aspectos de nuestro *job* sí que necesitamos saber cuál es su valor. Para verlo, desde la página principal de Hudson accedemos al menú de configuración y es el primer parámetro que nos encontramos (Hudson Home -> Manage Hudson -> System Configuration -> Home directory). En mi caso es el siguiente

```
HUDSON_HOME = /srv/hudson_home
```

Dentro de este directorio existe el subdirectorio *jobs* que contiene una entrada para cada *job* configurado. Así pues aquí tendremos un subdirectorio *Encuestas-2* (que, como decía, se corresponde con el nombre

que le asignamos al *job* cuando lo configuramos previamente). Dentro del directorio correspondiente a cada job, uno de los subdirectorios es *workspace* (si no existiera, le daríamos a construir el job desde la interfaz del Hudson). Este directorio contiene los ficheros descargados desde el repositorio de código cada vez que se lanza una construcción del job (el código fuente, los ficheros a usar por Ant, las bibliotecas incluidas en el proyecto, etc) y los artefactos generados en la construcción del proyecto. Lo remarco de nuevo: **contiene los ficheros descargados desde el repositorio**. Lo que quiere decir que si el proyecto para ser construido necesita artefactos que existen en los entornos de desarrollo pero que no están incluidos en el repositorio (o sus localizaciones son diferentes a las especificadas en los diferentes scripts de construcción), la construcción inevitablemente fallará. Con las herramientas con las que estamos trabajando habrá que hacer lo siguiente:

- opcionalmente, si es un proyecto J2EE o requiere bibliotecas externas al contenido del repositorios de código, hacerlas accesibles al Hudson,

- añadir algunas otras dependencias de NetBeans que tiene el *build.xml* generado,

- construir el directorio */nbproject/private* y su contenido para solucionar las dependencias.

Nuestro proyecto es un proyecto web que hemos configurado de manera que haga uso de la implementación de Servlets de Tomcat. Ello significa que en nuestro entorno de desarrollo tenemos un Tomcat y por tanto tendremos que replicar su instalación en la máquina donde reside Hudson para que pueda usar dichas bibliotecas. Lo más sencillo es empaquetar el Tomcat instalado en uno de los entornos de desarrollo y desempaquetarlo tal cual en el servidor de integración. En mi caso lo situaré en */srv/apache-tomcat-6.0.18*.

La manera más sencilla de solucionar las dependencias que tienen los scripts del Ant del NetBeans es seguir una aproximación similar a la del caso del servidor de aplicaciones: copiar el propio NetBeans en la máquina donde reside el servidor de integración para que éste tenga acceso directo a todos los recursos. Obviamente es posible usar una aproximación menos “drástica” y copiar sólo aquellos recursos necesarios, pero sin duda copiar todo el entorno es la manera más rápida que hacerlo y la única penalización que obtenemos es ocupar unos cuantos MB más en el disco duro. Personalmente prefiero “malgastar” esos pocos a MB a tener que estar hackeando los scripts para ver qué necesito exactamente o pelearme con posibles y sutiles errores difíciles de trazar. Recapitulando: empaquetaremos el NetBeans de una de las máquinas de desarrollo y en mi caso lo situaré en */srv/netbeans-6.5*. Adicionalmente NetBeans también guarda parte de la configuración en un directorio oculto en el *home* del usuario en un directorio oculto llamado *.netbeans* (esto es útil para que diferentes usuarios en una misma máquina física puedan tener configuraciones diferentes). En este caso no será necesario copiarlo absolutamente todo (un poco más adelante daré los detalles).

A continuación hemos de construir el directorio *private* que contiene los metadatos donde, entre otras cosas, se está configurando parte de la información local a utilizar por Ant y que por ello no se ha incluido en el Subversion. El tipo de proyecto que estamos usando (proyecto web usando Tomcat) sólo genera un fichero en esta carpeta, *build.properties*. Construiremos la carpeta *private* como un subdirectorio de la carpeta *nbproject* y copiaremos el fichero *properties* de uno de los entornos de desarrollo.

Los cambios que aplicaremos (todos relacionados con el cambio de rutas absolutas diferentes en desarrollo y en integración) son los siguientes:

- cambiar la ruta al Tomcat por la que corresponde (sustituir la cadena */home/ivan/apache-tomcat-6.0.18* por la cadena */srv/apache-tomcat-6.0.18*),

- cambiar la ruta del Netbeans (sustituir la cadena */home/ivan/.netbeans-6.5* por la cadena */srv/netbeans-6.5*),

- cambiar las referencias al directorio de configuración del *home* (sustituir la cadena */home/ivan/.netbeans* por */srv/.netbeans*).

La primera línea del *nbproject/private/properties* pasará a referenciar al archivo de configuración del Tomcat a la nueva ubicación en el servidor de integración en la ruta */srv/.netbeans/6.5/tomcat60.properties*. Editaremos el fichero para adaptarlo al nuevo entorno; de hecho realmente la única línea que habremos de modificar es la que define el directorio donde reside el Tomcat porque el resto sólo tienen sentido si queremos arrancar el propio Tomcat (circunstancia que se da en las máquinas de desarrollo pero no, en principio, en el servidor de integración). Tras modificar el fichero tendríamos algo así:

```
tomcat.home=/srv/apache-tomcat-6.0.18
tomcat.password=super-password-de-administracion-de-Tomcat
tomcat.url=http://localhost:8084
tomcat.username=ide
```

Otro fichero referenciado en el *nbproject/private/properties* que también tendemos que modificar será el */srv/.netbeans/6.5/build.properties*. Este fichero gestiona una gran cantidad de propiedades. Afortunadamente con sólo sustituir todas las ocurrencias de la cadena que indica la ruta de la raíz de la instalación del NetBeans por la nueva ruta en el servidor de integración será suficiente (en mi caso, sustituir */home/ivan/netbeans-6.5* por */srv/netbeans-6.5*).

Comprobando que el job está bien configurado

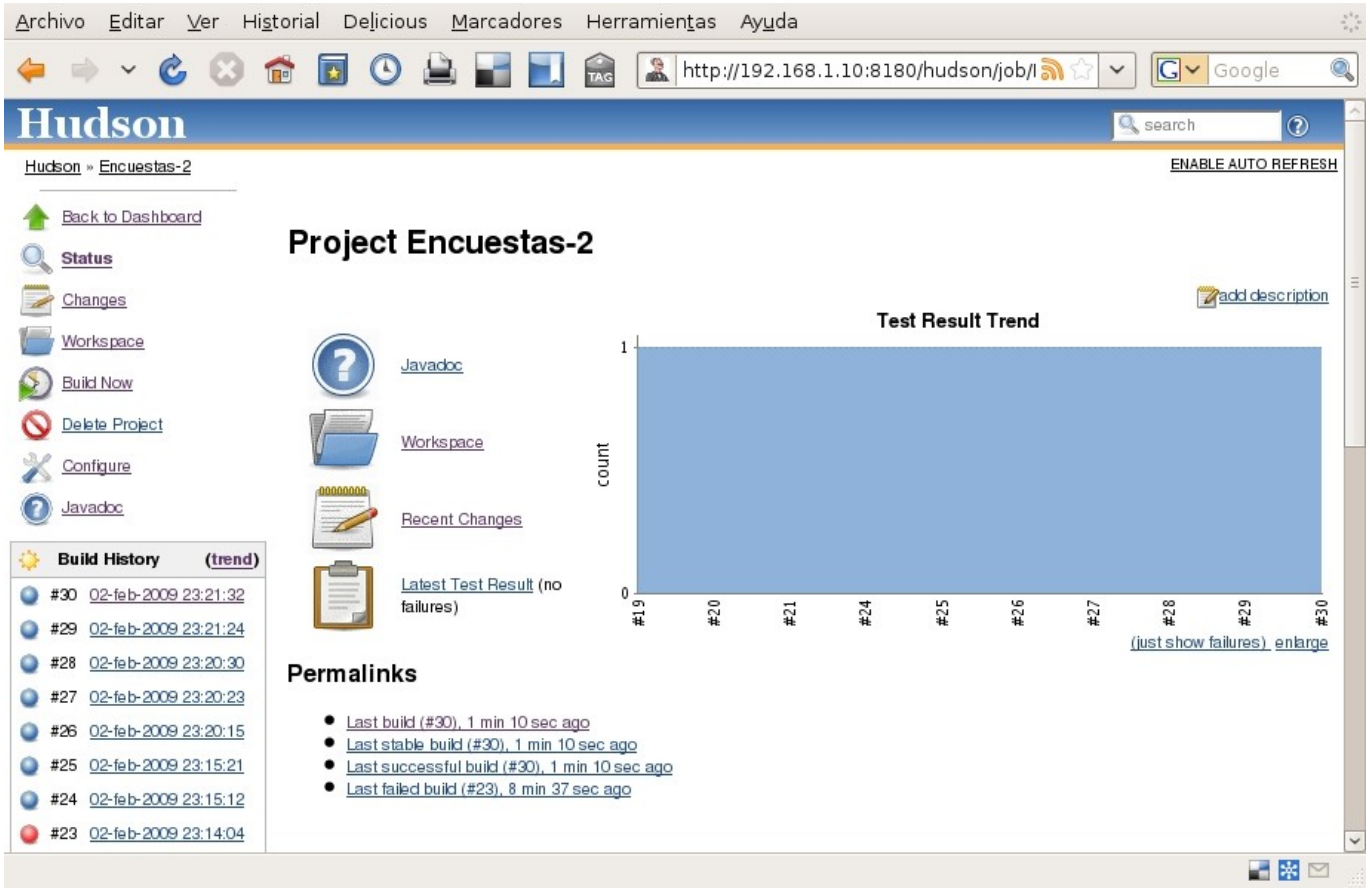
¡Felicidades! El camino ha sido más o menos largo y tedioso (no tanto, ¿verdad?), pero ya hemos casi terminado.

Para asegurarnos que todo se ha configurado correctamente intentaremos construir el proyecto desde la consola del servidor de integración. Para ello nos situaremos en el directorio *Encuestas* dentro del workspace del job que estamos configurando *(/srv/hudson_home/jobs/Encuestas-2/workspace/Encuestas)* que define la raíz del proyecto tal y como está en el Subversion. En ese directorio tenemos el *build.xml* así que podemos invocar al Ant tranquilamente. Una buena idea es probar todos los *targets* que querremos ejecutar desde el Hudson (*compile*, *test* y *javadoc*). Si se ejecutan sin problemas tenemos la garantía que Hudson también podrá hacerlo.

La prueba de fuego será construir el job desde la interfaz del Hudson: **Hudson Home -> Encuestas-2 -> Build Now** lo que programa una ejecución. Al terminar la misma, tendremos una nueva entrada en la **Build History** que tendrá una bullet azul o roja en función de si el proyecto se ha podido construir de forma correcta o de si ha habido algún error.

Los links **Javadoc** y **Latest Test Result** nos permiten acceder respectivamente a la documentación del proyecto y a los informes de testing generados por los diferentes tests unitoarios (Junit).

En este punto no estaría de más hacer una copia de seguridad del directorio *nbproject/private* y todo su contenido. El resto del workspace lo podéis regenerar al momento desde el repositorio, pero los cambios que habéis aplicado a los ficheros contenidos en el *private* no los tenéis en ningún otro sitio. Puesto que a priori, como decía, podemos recuperar (casi) todo el workspace desde el Subversion, en según qué circunstancias podemos decidir borrarlo sin acordarnos de todas las modificaciones que hemos tenido que



hacer manualmente.

Conexión con el SVN

Podríamos dejarlo aquí, pero uno de los requisitos que queríamos configurar es que el *job* se construyera automáticamente cada vez que hiciéramos un *commit* en el Subversion. Para conseguir este comportamiento tenemos que configurar el Subversion para que avise al Hudson cada vez que detecte esta situación.

Una de las características que tiene Hudson es que permite programar la construcción de un *job* haciendo una petición *GET* sobre una *url* asociada al proyecto en cuestión. Siguiendo con nuestro ejemplo dicha *url* sería *http://192.168.1.10:8180/hudson/job/Encuestas-2/build.*

Por otro lado, el Subversion prevee una serie de eventos que pueden ser detectados y asociarles un script. Tenemos una serie de plantillas de los scripts y eventos que se pueden detectar en la carpeta *hooks* del repositorio que estamos utilizando. Puesto que nosotros queremos activar la construcción del proyecto cada vez que hacemos un commit, el script que necesitamos codificar es uno que se llame *post-commit*:

```
svn@hargon:/srv/svn/Encuestas-2/hooks$ ls
post-commit.tmpl          post-unlock.tmpl  pre-revprop-change.tmpl
post-lock.tmpl            pre-commit.tmpl   pre-unlock.tmpl
post-revprop-change.tmpl  pre-lock.tmpl     start-commit.tmpl
svn@hargon:/srv/svn/Encuestas-2/hooks$ cp post-commit.tmpl post-commit
svn@hargon:/srv/svn/Encuestas-2/hooks$ chmod ug+x post-commit
```

El fichero que hemos copiado es una simple plantilla. Ahora la editaremos para que efectivamente haga una petición *GET*. Esto lo podemos hacer de múltiples maneras, aunque una muy fácil es utilizar Perl y algunas de sus bibliotecas (el gestor de paquetes *apt-get* o *Synaptic* nos permitirá instalar todas las

bibliotecas requeridas de una manera sencilla). Una posible implementación de este script es la siguiente:

```
#!/usr/bin/perl
use LWP::Simple;
use HTML::LinkExtor;
my $content = get("http://192.168.1.10:8180/hudson/job/Encuestas-2/build");
```

Comprobemos que el script funciona ejecutándolo desde la consola. Si todo va bien, al abrir el Hudson veremos que se ha lanzado una construcción del job.

El último paso es probar que la integración con el Subversion funciona correctamente. Simplemente tenemos que hacer un *commit* que modifique cualquier fichero y el Hudson debería construir el job.