

# ¿Qué ofrece Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.  
Ese apoyo que siempre quiso tener...

## 1. Desarrollo de componentes y proyectos a medida



## 2. Auditoría de código y recomendaciones de mejora

## 3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



## 4. Cursos de formación (impartidos por desarrolladores en activo)

Spring MVC, JSF-PrimeFaces /RichFaces,  
HTML5, CSS3, JavaScript-jQuery

Gestor portales (Liferay)  
Gestor de contenidos (Alfresco)  
Aplicaciones híbridas

Tareas programadas (Quartz)  
Gestor documental (Alfresco)  
Inversión de control (Spring)

Control de autenticación y  
acceso (Spring Security)  
UDDI  
Web Services  
Rest Services  
Social SSO  
SSO (Cas)

JPA-Hibernate, MyBatis  
Motor de búsqueda empresarial (Solr)  
ETL (Talend)

Dirección de Proyectos Informáticos.  
Metodologías ágiles  
Patrones de diseño  
TDD

BPM (jBPM o Bonita)  
Generación de informes (JasperReport)  
ESB (Open ESB)



adictos al trabajo

Autentia  
ess solutions  
inad por  
idos

E-mail:

Contraseña:

Deseo registrarme **Entrar**  
He olvidado mis datos de acceso

[Inicio](#) [Quiénes somos](#) [Tutoriales](#) [Formación](#) [Comparador de salarios](#) [Nuestro libro](#) [Charlas](#)  
[Más](#)

Estas en:

[Inicio](#) [Tutoriales](#) [Git y cómo trabajar con un repositorio de código distribuido](#)



DESARROLLADO POR:

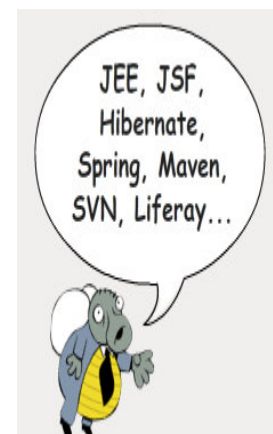
Alejandro Pérez García

**Alejandro es socio fundador de Autentia y nuestro experto en J2EE, Linux y optimización de aplicaciones empresariales.**

**Ingeniero en Informática y Certified ScrumMaster**

Si te gusta lo que ves, puedes contratarle para impartir **cursos presenciales** en tu empresa o para **ayudarte en proyectos** (Madrid). Puedes encontrarme en [Autentia](#): Ofrecemos servicios soporte a desarrollo, factoría y formación

Catálogo de servicios  
Autentia



Últimas Noticias

Nuevo formato de tutoriales podcast con bolígrafo LiveScribe

Estrenamos nuevo aspecto en AdictosAlTrabajo

X Charla Autentia - Primeros pasos con Talend

IX Charla Autentia - Android - Vídeos y Material

Creamos el grupo amigos de AdictosAlTrabajo en Facebook

Histórico de NOTICIAS

Últimos Tutoriales del Autor

Gestión de proyectos ágiles con Pivotal Tracker

Fecha de publicación del tutorial: 2009-02-26

Descargar TUTORIAL



15

**Vota este TUTORIAL** (Tienes que estar registrado para votar.)

Share |

## Git y cómo trabajar con un repositorio de código distribuido

Creación: 09-07-2010

### Índice de contenidos

1. Introducción
2. Entorno
3. Instalación de Git
4. Configuración de Git
5. Empezando a trabajar con Git
  - 5.1. Consiguiendo un repositorio de Git
    - 5.1.1. Creando un repositorio de Git
    - 5.1.2. Clonando un repositorio de Git
  - 5.2. Trabajo habitual (trabajando en local yo solo) con Git
  - 5.3. Trabajo distribuido (trabajando en equipo) con Git
    - 5.3.1. Trabajando sólo con git pull
    - 5.3.2. Trabajando con repositorios públicos de Git (git pull y git push)
  - 5.4. Creando una etiqueta
  - 5.5. Otras formas de colaborar y usar Git
6. Herramientas gráficas
7. Conclusiones

## 1. Introducción a Git

Git es, como dice en su propia página web (<http://git-scm.com/>), un sistema de control de versiones distribuido, de código abierto, y gratuito. Y la palabra clave es "distribuido", y ahora vamos a explicar por qué.

Ya conocemos otros sistemas de control de versiones como Subversion (<http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=subversion>), pero la diferencia con Git es que en el caso de Subversion, CVS o similares hay un repositorio central con el cual se sincroniza todo el mundo. Este repositorio central está situado en una máquina concreta y es el repositorio que contiene todo el histórico, etiquetas, ramas, ...

En el caso de los sistemas de control de versiones distribuidos, como es el caso de Git, esta idea de repositorio central no existe, y el repositorio está distribuido por todos los participantes. Es decir todos los participantes tienen en local todo el histórico, etiquetas, ramas, ... La gran ventaja de esto es que no necesitas estar conectado a la red para hacer cualquier operación contra el repositorio, por lo que el trabajo es mucho más rápido y con menos dependencias (imaginate haciendo un branch de tu proyecto en mitad de un vuelo a 33.000 pies de altura ;)

Ante esta idea hay algunas preguntas comunes que suelen asaltar nuestra cabeza:

- ¿Si tenemos todo el repositorio en local, no ocupará mucho espacio? Lo normal es que no porque al ser un repositorio distribuido, sólo tendremos las partes que nos interesan. Habitualmente si tenemos un repositorio central, tendremos muchos proyectos, ramas, etiquetas, ... Al tener un repositorio distribuido sólo tendremos en local la parte con la que estamos trabajando (la rama de la que hemos partido).
- ¿Si todo el mundo trabaja en su local, no puede resultar en que distintas ramas diverjan? Efectivamente y sí. Esto puede ocurrir y es natural. En un desarrollo tipo open-source no hay mucho problema (es lo habitual y constantemente están saliendo nuevos "forks" de los productos), en un desarrollo "interno" tampoco hay problema si ocurre esto, porque luego vamos a acabar haciendo merge. Todo al final depende de nuestro proceso de desarrollo, no tanto de la herramienta que usemos; es normal que los desarrolladores abran nuevas ramas constantemente para desarrollar partes del sistema y que luego hagan merge para unir estos cambios sobre una "línea principal".

Ahora que tenemos claro que es un sistema de control de versiones distribuido, vamos ver como trabajar con Git.

## 2. Entorno

El tutorial está escrito usando el siguiente entorno:

- Hardware: Portátil MacBook Pro 17' (2.93 GHz Intel Core 2 Duo, 4GB DDR3 SDRAM, 128GB Solid State Drive).
- NVIDIA GeForce 9400M + 9600M GT with 512MB
- Sistema Operativo: Mac OS X Snow Leopard 10.6.3
- Git 1.7.1

## 3. Instalación de Git


Tenemos dos opciones, podemos descargarnos los fuentes y compilarlos (algo realmente fácil si nuestro sistema es algún sabor de Unix, como Linux o incluso Mac) o buscar una versión compilada.


Para compilar los fuentes es tan fácil como bajarlos de <http://git-scm.com/download>, después descomprimir el archivo, y desde un terminal hacer:


```
$ make prefix=/usr/local/git all doc ; # este comando
lo ejecutamos con nuestro usuario normal

$ sudo make prefix=/usr/local/git install install-doc;
# este comando lo ejecutamos como root, como estoy en
Mac por eso he puesto el sudo delante
```

 Eclipse Helios, la nueva versión 3.6 de Eclipse


 Mac Automator y como renombrar múltiples ficheros

 Prototipado rápido de pantallas (mocks) con Google Drawings

 IAQ (Interesting Asked Questions), SPI ¿qué es, hay que usarlo, o no, cuándo?


### Últimos Tutoriales

 Introducción a Activiti

 PMBOK (Project Management Body of Knowledge) v4.0


 Gestión de proyectos ágiles con Pivotal Tracker

 Introducción al gestor de base de datos H2

 Exportación de trabajos en Talend

### Últimas ofertas de empleo

2010-06-25

 T. Información - Analista / Programador - BARCELONA.

Si no queremos liarnos podemos descargar un binario ya compilado, en caso de Mac lo podemos encontrar en: <http://code.google.com/p/git-osx-installer/downloads/list?can=3>. El único problema de esto es que puede ser que no encontremos la ultimísima versión, así que dependerá de lo "obsesivos" que seamos para estas cosas ;)

## 4. Configuración de Git

Una vez instalado, lo primero que tenemos que hacer es configurar nuestro nombre y nuestro email para que Git pueda "firmar" nuestros commits. De esta forma estaremos identificando que ese commit, que ese cambio en el código, es nuestro.

```
$ git config --global user.name "Alejandro Perez Garcia"

$ git config --global user.email
"alejandropg@autentia.com"
```

Esta información se guardará en un fichero en nuestro directorio home (por defecto ~/.gitconfig). Mas o menos contendrá esto:

```
[user]
    name = Alejandro Perez Garcia
    email = alejandropg@autentia.com
```

Si quisiéramos sobrescribir estos valores para un proyecto concreto, podemos ejecutar los mismos comandos en el directorio raíz del proyecto, pero sin la opción --global. De esta forma la información se guardará en el fichero .git/config del proyecto.

Otra configuración interesante puede ser:

```
$ git config --global color.status auto

$ git config --global color.branch auto

$ git config --global color.diff auto

$ git config --global color.interactive auto
```

Con esto conseguimos que, si nuestro terminal lo permite, Git nos va a colorear ciertas cosas, con lo que será más fácil leer la información que nos proporcione.

## 5. Empezando a trabajar con Git

### 5.1. Consiguiendo un repositorio de Git

#### 5.1.1. Creando un repositorio de Git

Para crear un repositorio es tan sencillo como, dentro del directorio que queremos gestionar con git, hacer:

```
$ git init
```

Git contestará:

```
Initialized empty Git repository in .git/
```

Esto quiere decir que se ha creado el directorio .git/ donde Git guardará toda la información de control. Esto es otra diferencia con los sistemas de control habituales, como Subversion, que nos dejaban un directorio de control en todos los subdirectorios de nuestro proyecto (el típico .svn/). Lo cual es de agradecer porque "ensucia" menos nuestro trabajo.

Si os dais cuenta como Git es distribuido el repositorio por ahora sólo lo tenemos en nuestro local, y por lo tanto, no ha hecho falta tener conexión de red. De esta forma también podemos usar Git para versionar nuestro trabajo de forma muy sencilla, aunque no pensemos compartirlo con nadie más.

### 5.1.2. Clonando un repositorio de Git

Lo normal cuando trabajamos con sistemas de control de versiones suele ser “descargarnos” un proyecto de un repositorio existente (normalmente los repositorios no los creamos nosotros como en el punto anterior, sino que nos conectamos a ellos).

Para esto es tan fácil como hacer cosas como:

```
git clone git://git.kernel.org/pub/scm/git/git.git
git clone http://www.kernel.org/pub/scm/git/git.git
```

Fijaos como podemos usar un protocolo propio de git (`git://`) que es más óptimo, o podemos hacerlo por HTTP normal (`http://`). Otra de las ventajas de git es que podemos hacer el “trasiego” de información de muchísimas maneras, incluso a través de un email o un pendrive.

## 5.2. Trabajo habitual (trabajando en local yo solo) con Git

Modificaremos fichero, bien porque cambiamos su contenido o bien porque los añadimos nuevos. En ambos casos tendremos que hacer:

```
$ git add file1 file2 file3
```

Es importante destacar que el comando `add` hay que lanzarlo incluso si el fichero ya estaba bajo el “control” de repositorio y sólo hemos cambiado su contenido. Esto es bastante diferente a como suelen trabajar el resto de sistemas de control de versiones. Podríamos decir que el `add` no significa “añadir al repositorio”, sino que significa “añadir al *index* para el próximo commit”. Es decir, con el `add` le estamos indicando que es lo que queremos que versione la próxima vez que hagamos commit.

Podemos ver que es lo que tenemos pendiente de hacer commit con:

```
$ git diff --cached
```

La opción `--cached` indica que queremos ver los cambios con respecto a lo que ya tenemos añadido al *index* y que se tendrá en cuenta en el proximo commit. Si no ponemos esta opción nos indicará todas las cosas que potencialmente podríamos añadir al *index* con un `add`.

Otra forma de ver los cambios pendientes es con:

```
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#   modified:   file1
#   modified:   file2
#   modified:   file3
#
```

Cuando ya no queramos hacer más cambios y queramos hacer el commit (la versión en el repositorio), haremos:

```
$ git commit
```

Para automatizar el comando `add`, y no tener que hacerlo de cada fichero que hemos modificado, podemos ejecutar:

```
$ git commit -a
```

Pero ojo, que sólo se hará el `add` automáticamente de los ficheros modificados que ya existían en el repositorio; de los ficheros nuevos siempre tendremos que hacer el `add`.

## 5.3. Trabajo distribuido (trabajando en equipo) con Git

El trabajo distribuido implica que estamos trabajando con más personas, donde cada una tiene su propio repositorio, con sus propios branches, ...

En este entorno empezaremos trabajando con un clone de forma que tengamos nuestra propia copia en local.

## Git y cómo trabajar con un repositorio de código distribuido

A partir de aquí editaremos ficheros y haremos commit tantas veces como queramos. Simplemente hay que tener en cuenta que todos estos commits los estamos haciendo sobre nuestro repositorio local.

La cuestión es como sincronizamos estos cambios que tenemos en nuestro local con el repositorio original del cual hicimos el clone. Pues bien, tenemos dos herramientas básicas:

- `git pull` - "Tira" de un repositorio los cambios hacia nuestro repositorio. Nos estamos bajando las actualizaciones.
- `git push` - "Empuja" nuestros cambios hacia otro repositorio. Es como si estuviéramos subiendo hacia ese repositorio los cambios que hemos hecho en local.

Vamos a ver algunos modelos de trabajo. Pero no debéis quedaros sólo con esto porque Git, como el resto de sistemas de control de versiones distribuidos, permite un enorme abanico de nuevos modelos de trabajo, al romper "las cadenas" de tener un único repositorio central.

### 5.3.1. Trabajando sólo con git pull

Imaginemos que tenemos a María y a Carlos. Carlos puede empezar a trabajar a partir del código de María con:

```
$ git clone ssh://mariahost/home/maria/project
```

Ahora Carlos hace varios cambios en su repositorio, cuando cree el que el código está listo puede avisar a María para que esta haga un pull del código de Carlos. De esta forma María ejecutará algo como:

```
$ git pull ssh://carloshost/home/carlos/project
```

Lo que está haciendo María es "bajarse" el código de Carlos y luego hacer un merge en su propio repositorio. Evidentemente si hay conflictos, tendrá que ser María la que los resuelva.

Si ahora María ha hecho más cambios sobre el código, y Carlos quisiera actualizarse, podría hacer:

```
$ git pull ssh://mariahost/home/maria/project
```

De forma que Carlos se estaría bajando el código del repositorio de María y estaría haciendo un merge con su propio repositorio. Vemos así como el `pull` puede funcionar en ambos sentidos.

Esta forma de trabajar puede estar bien para equipos pequeños y con mucha comunicación. El problema que tiene es que el que hace el `pull` es el encargado de resolver los conflictos, así que si queremos tener una "copia maestra" del repositorio, el que mantenga esta copia tendrá que resolver los conflictos de todos los participantes.

### 5.3.2. Trabajando con repositorios públicos de Git (git pull y git push)

En este punto vamos a ver otra forma de trabajar que puede ser más conveniente si queremos mantener esa "copia maestra".

Todo empieza igual que siempre, haciendo un `clone`. Lo único es que esta vez lo podríamos hacer de la "copia maestra".

```
$ git clone ssh://mainhost/var/autentia/project
```

Ahora, igual que antes, iremos haciendo modificaciones y commits sobre nuestro repositorio local. Llegará un momento en el cual querremos "subir" nuestros cambios. Para ello podemos hacer:

```
$ git push ssh://mainhost/var/autentia/project master
```

Con esto estamos subiendo todos los cambios de nuestro repositorio local al repositorio remoto. "master" indica que los cambios se subirán al branch "master" que es el branch por defecto.

Podría pasar que no pudiéramos hacer el `push` directamente porque el código del repositorio remoto a cambiado. En este caso somos nosotros mismos los que nos tendremos que encargar de resolver los conflictos, normalmente haciendo primero un `pull` para actualizarnos, resolviendo los conflictos en local, y luego volviendo a hacer un `push` para volver a subir los cambios.

## 5.4. Creando una etiqueta

Tan fácil como hacer:

```
$git tag version-1.0 1b2e1d63ff
```

Donde 1b2e1d63ff es el hash del commit que queremos etiquetar (en Subversion estamos acostumbrados a hablar de número de revisión, siendo este un número consecutivo que identifica un snapshot concreto del repositorio; en Git el equivalente es este hash, de forma que cada commit tiene un hash propio que lo identifica de forma inequívoca).

Podríamos decir que esta es la forma ligera de poner etiquetas, ya que más bien lo que estamos haciendo es una referencia a un commit concreto (como si fuera una especie de branch de sólo lectura), pero no estamos añadiendo un objeto nuevo. Si queremos crear un objeto tag lo podemos hacer con:

```
$ git tag -a version-1.0 1b2e1d63ff
```

Como ahora el tag es un objeto, podemos asociarle un comentario, e incluso firmarlo digitalmente.

## 5.5. Otras formas de colaborar y usar Git

Nos estamos dejando en el tintero muchas otras características de Git como:

- Crear repositorios públicos y privados
- El trabajo con branches que es bastante fácil y dinámico.
- Los parches que se pueden convertir en una potente herramienta para compartir cambios.
- El “stashing” que nos permite guardar el estado actual, por ejemplo para resolver un bug puntual, y luego recuperar el estado que teníamos, gestión del historico.
- Git hooks, ...

Pero ya sería extender el tutorial demasiado, así que lo mejor es ponerse a estudiar, practicar y probar.

## 6. Herramientas gráficas

Dentro del propio Git podemos encontrar:

- `gitk` - Para explorar el repositorio, ver los cambios, ...
- `git gui` - Para hacer las tareas habituales (commits, branches, etc) con una interfaz gráfica.

Si esto se nos queda corto, tenemos a nuestra disposición un plugin de Eclipse: EGit (<http://www.eclipse.org/egit/>).

Y si esto tampoco nos vale, en <https://git.wiki.kernel.org/index.php/InterfacesFrontendsAndTools> podemos encontrar multitud de herramientas.

## 7. Conclusiones

Sólo hemos empezado a rascar en la superficie de las funcionalidades y posibilidades que ofrecen sistemas de control de versiones distribuidos como Git, así que os recomiendo que echéis un vistazo a la documentación, porque veréis que queda muchísimo por descubrir; en especial:

- el libro que podéis encontrar online en su propia web (<http://book.git-scm.com/>),
- el manual de referencia (<http://www.kernel.org/pub/software/scm/git/docs/>),
- y la tarjeta de “trucos” (<http://zrusin.blogspot.com/2007/09/git-cheat-sheet.html>).

Como resumen podríamos decir que Git está especialmente indicado cuando:



## Git y cómo trabajar con un repositorio de código 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 branches 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).
- ...

pero en general tendréis que ser vosotros los que veáis si os resulta útil o no. Lo que si os puedo decir es que se puede hacer todo lo que hacíamos con los sistemas de control de versiones centralizados, y más ;)

## 8. Sobre el autor

Alejandro Pérez García, Ingeniero en Informática (especialidad de Ingeniería del Software) y Certified ScrumMaster

Socio fundador de Autentia (Formación, Consultoría, Desarrollo de sistemas transaccionales)

<mailto:alejandropg@autentia.com>

Autentia Real Business Solutions S.L. - "Soporte a Desarrollo"

<http://www.autentia.com>

### Anímate y coméntanos lo que pienses sobre este **TUTORIAL**:

Puedes opinar o comentar cualquier sugerencia que quieras comunicarnos sobre este tutorial; con tu ayuda, podemos ofrecerte un mejor servicio.

Enviar comentario

(Sólo para usuarios registrados)

» **Regístrate** y accede a esta y otras ventajas «

**Autor**

**Mensaje de usuario registrado**



SOME RIGHTS RESERVED

Esta obra está licenciada bajo [licencia Creative Commons de Reconocimiento-No comercial-Sin obras derivadas 2.5](#)



