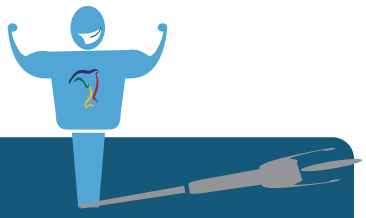
autentia

Avenida de Castilla,1 - Edificio Best Point - Oficina 21B
28830 San Fernando de Henares (Madrid)
tel./fax: +34 91 675 33 06
info@autentia.com - **www.autentia.com**

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

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

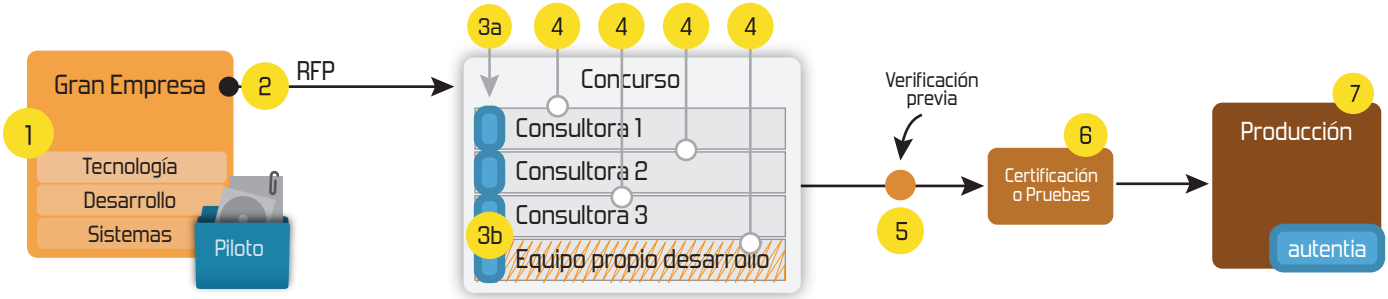
1. Desarrollo de componentes y proyectos a medida



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
- 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
- Tareas programadas (Quartz)
- Gestor documental (Alfresco)
- Inversión de control (Spring)
- BPM (jBPM o Bonita)
- Generación de informes (JasperReport)
- ESB (Open ESB)

Compartimos nuestro conociemiento en:
www.adictosaltrabajo.com

Para más información visítenos en:
www.autentia.com

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 darte ayuda con soporte experto, impartir **cursos presenciales** en tu empresa o para que **realicemos tus proyectos como factoría** (Madrid). Puedes encontrarme en Autentia: Ofrecemos servicios de soporte a desarrollo, factoría y formación

Fecha de publicación del tutorial: 2011-06-27

Trabajando con GIT, introducción al uso de los branch y git-completion.bash

Creación: 26-06-2011

Índice de contenidos

1. Introducción

GIT es un sistema de control de versiones distribuido. Ya hemos hablado de él en otros tutoriales, así que no voy a extenderme, simplemente os recomiendo visitar el tutoriales *Git y cómo trabajar con un repositorio de código distribuido* para saber más al respecto:

En esta ocasión vamos a hablar de como trabajar con los branch de GIT.

Catálogo de servicios Autentia

Últimas Noticias

Alfresco Day 2011
XVII Charla Autentia - Grails - Vídeos y Material
¡¡¡ 15 millones de descargas de tutoriales !!!
XVII Charla Autentia - Grails
Charla en WhyFLOSS en el IE: la ppt

Histórico de NOTICIAS

Últimos Tutoriales

Introducción a Spring Security 3.1
Implementando SSO con CAS: ejemplo práctico
Como desarrollar un plugin para Eclipse
Técnica del Time-Lapse
Incluir Gadgets en Liferay 6.0.5: Cómo añadir Gadgets de forma sencilla

2. Entorno

El tutorial está escrito usando el siguiente entorno:

- Hardware: Portátil MacBook Pro 17' (2.8 GHz Intel i7, 8GB DDR3 SDRAM, 256GB Solid State Drive).
- NVIDIA GeForce GT 330M with 512MB
- Sistema Operativo: Mac OS X Snow Leopard 10.6.8
- GIT 1.7.5.4

3. Branch en Git

Bueno, la primera pregunta sería ¿y qué es un *branch*? La traducción literal sería: *rama*. Es decir, dentro de nuestro sistema de control de versiones podemos ver el histórico de cambios como si de un árbol se tratase. De esta forma podemos ir abriendo ramas que parten bien de la rama principal (master) o de otra rama (branch).

La principal utilidad que tienen los branch es la de organizar nuestro tabajao, por ejemplo:

- para desarrollar una nueva funcionalidad sin afectar al master mientras lo hacemos.
- para hacer un hotfix en una versión que ya ha salido a producción.
- para hacer un branch de producción, otro de pre, otro de testing, ... y así ir promoviendo los cambios de uno a otro.
- para gestionar distintas versiones de un mismo producto: podríamos tener un branch por cada cliente donde está instalado el producto (la verdad es que esto no lo recomiendo porque en cuento tengas más de un cliente la gestión se vuelve un infierno, es mejor tener un proyecto único y módulos de personalización; aquí lo comento simplemente porque he visto a gente trabajar así en la oscuridad cerca de la puerta de Tannhäuser).
- ...

Uno de los usos más comunes, por lo menos para mi, es el de desarrollar las nuevas funcionalidades dentro de un branch, en lugar de hacerlo directamente en el master. La principal ventaja que tiene esto para mi es que mantengo el master "limpio" lo que me permite hacer *pull* en cualquier momento y se que nunca voy a tener conflictos. Una vez hecho el *pull* puedo inspeccionar los cambios que han hecho mis compañeros y hacer *merge* si lo creo oportuno.

Además esto me permite desarrollar la funcionalidad sin "estorbar" a mis compañeros, y una vez esté estable hacer el merge con el master para que compartir los cambios con todo el equipo.

¿Y por qué hablo ahora de los branch cuando no lo había hecho nunca antes? Bueno, la verdad es que antes de usar GIT (cuando usaba CVS o Subversion) no usaba demasiado los branch, sobre todo porque era una tarea lenta y tediosa, ya que se hacía en el servidor. Una de las cosas buenas que tienen los sistemas de control de versiones distribuidos, y en concreto GIT es que las operaciones se hacen en local, siendo muy rápidas (GIT es especialmente rápido, posiblemente el más rápido de todos los sistemas de control de versiones). Por lo que no da pereza alguna trabajar con branch.

Para ver otras lindezas de GIT podéis ver esta página: *Por qué Git es mejor que X*, donde se hace una pequeña comparativa con otros sistemas de control de vesiones.

3.1. Para crear un branch

```
$ git branch nuevo-branch
```

```
$ git checkout nuevo branch
```

Con esto creamos un branch llamado nuevo "nuevo-branch" y con el checkout nos movemos a el (el working directory queda apuntando a este branch para poder trabajar con el).

El nuevo branch se crea a partir del branch en el que nos encontramos, por lo que ahora mismo el *master* y el *nuevo-branch* tendrían los mismos cambios. A partir de este momento podemos trabajar en cada branch de forma independiente.

Un comando equivalente, donde hacemos las dos operaciones (crear y movernos al branch) en una sería:

```
$ git checkout -b nuevo-branch
```

3.2. Para ver los branch que tenemos

```
$ git branch
```

Y obtendremos una salida del estilo:

```
  master
* nuevo-branch
```

Donde el * indica el branch activo (y además si el terminal soporta colores, el nombre del branch activo vemos que aparece en verde).

El listado anterior solo muestra los branch locales, pero también podemos listar los *branch remotos* si

hacemos:

```
$ git branch -a
```

```
* master
  nuevo-branch
  remotes/origin/master
```

Vemos como los branch remotos se pintan en rojo.

También podemos ver el branch en el que estamos usando el comando *status*:

```
$ git status
```

```
# On branch nuevo-branch
nothing to commit (working directory clean)
```

3.3. Para aplicar todos los commits que hemos hecho en nuestro branch sobre otro branch

Lo primero que tenemos que hacer es movernos al brach sobre el que queremos aplicar los cambios:

```
$ git checkout master
```

Ahora que estamos sobre el master vamos a hacer el *merge* con los commits que hemos hecho en el branch *nuevo-branch*. El comando *merge* lo que hace es incorporar los cambios realizados en el branch que estamos indicando (en nuestro caso *nuevo-branch*), sobre el branch actual (en nuestro caso *master*).

```
$ git merge nuevo-branch
```

Ahora en el branch *master* tenemos todos los cambios que hicimos sobre nuestro branch, por lo que podemos hacer un *push* para subirlos al repositorio central (como por ejemplo *GitHub*).

3.4. Otras operaciones con los branch

Con lo que hemos visto hasta ahora tendríamos cubierto el ciclo básico de trabajo (crear, trabajar, unir). Pero podemos hacer muchas más operaciones, por ejemplo, para ver las diferencias entre el branch *master* y el *nuevo-branch*, suponiendo que estamos en el *master*, podemos hacer:

```
$ git branch diff nuevo-branch
```

O podemos borrar un branch si ya no lo vamos a usar más:

```
$ git branch -d nuevo-branch
```

4. GIT y git-completion.bash

En el punto anterior hemos visto como podemos trabajar con branch. Ahora si imaginamos que tenemos tres o cuatro brach, podemos pensar que el trabajo puede ser un poco tedioso si constantemente tenemos que hacer `$ git branch` o `$ git status` para saber en que branch estamos y no meter la pata.

No hay que preocupares porque tenemos una extensión del bash shell que nos va a ayudar en el trabajo con los branch y en todo el trabajo con GIT en general. Esta se denomina **git-completion.bash**, y se encuentra en el *directorio contrib* de la propia distribución de GIT.

Esta extensión se trata de un simple script de bash que nos va a permitir, entre otras cosas, auto-completar los comandos de GIT así como los nombres de los branch; y lo que es más interesante, nos va a permitir modificar el prompt del sistema de forma que si estamos en un directorio gestionado por GIT vamos a ver en el propio prompt el branch activo!!!

Para instalar git-completion.bash basta con modificar nuestro **~/.bash_profile** y añadir las líneas:

```
source /usr/local/git/contrib/completion/git-completion.bash

export PS1='\u@\h:\w\[\033[32m\]$(__git_ps1 " (%s)")\[\033[0m\]$ '

export GIT_PS1_SHOWDIRTYSTATE=true
export GIT_PS1_SHOWSTASHSTATE=true
export GIT_PS1_SHOWUNTRACKEDFILES=true
export GIT_PS1_SHOWUPSTREAM="auto"
```

- **`source`** - carga el script. Si os fijáis en la ruta lo estamos cargando directamente de la instalación de GIT. También podéis haceros una copia en vuestro home como .git-completion.bash, pero yo prefiero cargarlo del directorio de la instalación para que cuando actualice el GIT se me actualice también el script.

- **`export PS1`** - define el prompt del sistema de forma que si estamos en un directorio gestionado por GIT nos aparecerá en color verde el nombre del branch activo.

```
alex@cell2:~/src/sandbox/branch-tutorial (nuevo-branch)$ git branch
  master
* nuevo-branch
alex@cell2:~/src/sandbox/branch-tutorial (nuevo-branch)$
```

- **el resto de `export`** - son opcionales, activándolos todos conseguimos ver en el promt más información como por ejemplo que tenemos archivos sin añadir al git, o que hay diferencias respecto al origen del branch, ...

5. GIT y ~/.gitconfig

`~/.gitconfig` es el fichero de configuración de GIT. Aquí os dejo un pequeño ejemplo que puede ser útil para terminar con este tutorial.

```
[user]
    name = tu nombre para identificarte en el repositorio
    email = tu correo para identificarte en el repositorio

[core]
    autocrlf = input

[color]
    ui = true

[alias]
    st = status
    ci = commit
    co = checkout
    br = branch
    lg = log --graph -10
    ll = log --graph -25 --pretty=format:'%Cgreen%h %Creset%ad %Cblue%an %Creset%s'
    in = fetch --dry-run
    out = push --dry-run

[diff]
    tool = opendiff

[difftool]
    prompt = false

[merge]
    conflictstyle = diff3
```

Podemos destacar el **uso de alias** para abreviar las operaciones más comunes, y la opción **`merge.conflictstyle = diff3`**. Esta opción puede resultar muy útil ya que cuando hagamos un merge y existan conflictos GIT nos va a mostrar el ancestro común a los dos branch (más información en *Reducing merge headaches: git meets diff3*).

6. Conclusiones

He probado varias herramientas visuales para trabajar con GIT, como *SmartGit* (de las visuales la mejor) o *GitX*, y la verdad es que al final donde más cómodo me encuentro y más productivo soy es en la línea de comandos. Estos se debe principalmente a que casi siempre trabajo con un número muy reducido de comandos y el **histórico del bash** es perfecto para esto. Y además el trabajo con los branch es infinitamente más rápido y cómodo gracias a **git-completion.bash** (con las herramientas visuales tienes que hacer veinte click para una sencilla operación de cambio de branch :P).

Como siempre os digo tenéis que conocer las herramientas que manejáis. Muchas veces basta con leerse el manual y sino siempre os intentaremos ayudar desde adictosaltrabajo.

Y recordar, **the command line power** !!!

7. Sobre el autor

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

Socio fundador de Autentia (Desarrollo de software, Consultoría, Formación)

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)

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

Comentarios