Una introducción a Darcs

por Esteban Manchado (aka zoso).

Darcs es un sistema de control de versiones (como CVS o SVN) pero distribuido. Daremos una introducción a sus fundamentos y veremos un ejemplo práctico utilizando darcs.

Cuando programas algo o preparas cualquier tipo de escrito, tener un sistema de control de versiones es algo que se agradece enormemente. En su forma más sencilla, equivale a poder seguir la historia de un documento desde que se comienza hasta que lo terminas, pudiendo elegir el momento en el que quieres ver lo que contenia. Una especie de "punto de restauracion" con lo que de esa manera siempre puede «retomar» un documento ANTES de meter la pata ;)

Índice de contenido

Introducción Teórica

Cómo instalar Darcs

- En debian/ubuntu: aptitude install darcs
- En MDV: urpmi darcs (no se necesita el rpm darcs-server)
- Compilado estático (para Linux) en http://www.carpetcode.org/
- Enlaces a binarios para muchos sistemas: http://darcs.net/

Introducción

Básicamente creo[1] que el control de versiones es en cierta manera una herramienta que ayuda a mejorar la calidad de lo que escribimos (sea código u otras cosas).

Descripción de Darcs

Básicamente un sistema de control de versiones (en adelante SCM) es una herramienta que nos permite controlar qué cambios se le hacen a un conjunto de ficheros. Normalmente se aplican a programas, pero no necesariamente.

Si yo estoy *escribiendo* un programa (o sea, tengo los ficheros de texto que son los fuentes), y quiero controlar qué cambios les voy haciendo, necesitaría alguna forma de seguimiento de esos ficheros de texto.

Un SCM me permite hacer eso, de tal forma que puedo saber qué cambios le he hecho a un fichero, qué día y qué persona (si es un equipo de varios), que líneas cambiaron exactamente cada vez, etc.

En los SCM tradicionales tenemos dos conceptos básicos muy importantes: la «Copia de Trabajo» y el «Repositorio». El «Repositorio» es el «*oráculo*» al que le preguntamos sobre la historia de los cambios que hemos hecho.

Una de las cosas que podemos solicitar es que nos dé una copia de la última versión de los ficheros en los que estamos trabajando.

Esa copia es lo que se llama una «Copia de Trabajo». Con esa Copia de Trabajo podemos ir.... pues eso, trabajando :-) cambiando ficheros, renombrándolos, o lo que necesitemos, añadiendo nuevos ficheros, borrando, etc. y en cualquier momento podemos preguntar qué cambios hemos hecho que todavía no están en el repositorio, y cuando estemos contentos podemos mandar esos cambios para que se conviertan en la última versión.

Ahora pasamos a explicar la coletilla de «distribuido». Los SCMs tradicionales son «centralizados»: eso significa que hay, digamos, un solo Repositorio por proyecto y muchas Copias de Trabajo. Lo malo de eso es que si queremos que haya colaboradores del proyecto les tendremos que dar permisos para que manden cambios a nuestro Repositorio, con lo cual tienes que confiar en ellos, claro, y claro, no siempre puedes hacer eso.

1 Opinión mía al respecto (bastante reciente, por cierto) en mi blog.

Entonces, hace unos años, salieron los SCM distribuidos. La idea básica es que por proyecto, haya un repositorio por cada *colaborador*. Uno lo consideras «*central*», probablemente, y tiene la copia «*oficial*» y cualquier persona que quiera colaborar no te tiene que pedir permiso para nada puede crear su propio repositorio, en su máquina que «*sale*» del tuyo y trabajar él en el suyo sin molestarte a ti y cuando tenga varias cosas hechas puede publicar su repositorio «*personal*».

Tenemos, por ejemplo el núcleo de Linux. Hay un Repositorio, que es el de Linus Torvalds, y que es el que la gente entiende que es el repositorio oficial, pero sólo es una convención social. Supongamos entonces que Alan Cox[2] dice «*voy a hacer tal modificación*» y coge y se hace un repositorio en su máquina, que «*salió*» del repositorio oficial de Linus.

Entonces Alan Cox puede trabajar en su «rama» (en su repositorio personal, con su versión personal) sin molestar a Linus y cuando tenga los cambios terminados puede decírselo a Linus, para que éste coja los cambios que le interesen (todo, o parte) y los incorpore al repositorio «oficial» del núcleo de Linux.

Lo de las versiones va aparte. Normalmente tienes una marca en los fuentes que indica el número de versión. Entonces, tú haces todos los cambios que quieres y cuando estás contento y le quieres llamar 1.2 entonces cambias ese número en los fuentes y haces una marca especial en el repositorio para que, si alguna vez necesitas la versión 1.2, puedas acceder a ella, pero básicamente son independientes el número de cambios que haces, y el número de versión que le das a tu programa.

Como ejemplo práctico, si Linus escribe un kernel y Alan lo modifica, luego cuando termina envia la copia a Linus para que la aproveche, o no. Linus sacaría una nueva version basada en ese cambio.

Estrictamente hablando no hace falta que Alan *envíe* la copia del «*cambio*» a Linus, sólo que la publique en algún sitio, y que Linus lo sepa, claro.

Características principales de Darcs

Darcs tiene cuatro características interesantes que lo distinguen de los SCM tradicionales (como CVS y Subversion) y de los otros SCMs distribuidos (arch, baz). Los puntos que paso a comentar seguidamente se aplican tanto a los SCM distribuidos como a los tradicionales, pues son similares: los distribuidos son como los tradicionales pero más avanzados.

- ### La Interfaz de Darcs

  Es bastante sencilla, y sobre todo... ¡es interactiva! Muy pocos SCMs tienen una interfaz interactiva, de hecho, creo que Darcs es el único que conozco.
  Como ejemplo diré que el Commit generalmente en los demás SCMs, lo único que tiene de interactivo es que pregunta por la descripción del

2 Recuerden que este dialogo es sólo es un supuesto.

cambio (para los que no habían usado SCMs: el «Commit» es la orden para mandar cambios de una Copia de Trabajo a un Repositorio).
Pues bien, en Darcs el Commit (que se llama «record») va preguntando por todos los cambios que has hecho para que los vayas validando uno a uno y sólo manda los que aceptes.
Si uno está completamente seguro y quiere mandar todos los cambios, como en los otros SCMs sólo tiene que hacer:

```
darcs record -a
```

- ## Simplicidad

Todo en Darcs es bastante simple; se basa en un par de conceptos bastante sencillos y que son muy fáciles de entender.

Si alguien ha usado alguna vez arch o baz....

... Darcs es todo lo contrario, es muy potente, pero a la vez muy sencillo (arch/baz son muy potentes, pero complicadísimos)

- ## No tiene servidor

Para usar Darcs no hace falta ni instalar ni configurar ningún servidor. Como no hace falta servidor, con sólo el cliente podemos trabajar, y no necesitamos configurar nada ni estar leyendo manuales para ver cómo se monta la autenticación de nosequé.

- ## Es muy fácil hacer proyectos derivados/ramas personales

Esto «sólo» es posible con los SCMs distribuidos y con arch, p.ej., aunque es bastante complicado (en comparación).

Cuando veamos Darcs en la práctica verán por qué es tan fácil hacer los proyectos derivados, ya que en realidad no hace falta que «des acceso» a ningún colaborador por lo que decía antes de los SCMs distribuidos aunque lo puedes hacer si quieres.

Generalmente los colaboradores «externos» montan su propio repositorio local y trabajan ahí y cuando tienen cambios «terminados», te los mandan. De hecho, lo más común es mandar cambios sueltos por correo, o publicar repositorios enteros por web.

El meollo de la cuestión

Veamos la primera sorpresa de Darcs:

> El concepto básico sobre el que se asienta Darcs es que ***es lo mismo*** una «Copia de Trabajo» que un «Repositorio».
>
> ***zoso oye el sonido de las pelucas cambándose***

¿Qué narices quiere decir esto?

Muy sencillo: que si alguien tiene un repositorio por ahí (digamos, Linus) y alguien se trae una copia (digamos, Alan), ese alguien automáticamente tiene un repositorio y cuando hace «record» en su copia de trabajo los cambios no se están mandando a Linus, sino que se está quedando en ese directorio que se trajo (la «Copia de Trabajo»/«Repositorio»)

Supongo que esto es lo más sorprendente/difícil de entender

Alguien se preguntará... ¿En que momento se vuelven a "reunir" los dos repositorios, el de Linus y el de Alan? Realmente nunca se «unen»... son independientes.

Linus puede coger cambios de Alan si Alan lo publica en algún sitio, y viceversa. Es decir, comparten cambios entre uno y otro, pero no se «funden».

Pero supongamos que interesa "reunirlos" de nuevo ¿Se puede hacer? No exactamente. Lo que puedes hacer es que todos tengan exactamente los mismos cambios, pero siempre serán independientes porque están en distintas máquinas y tal.

En Darcs un repositorio es un conjunto de «parches» o «cambios», entonces, como los parches los puedes llevar de un repositorio a otro puedes hacer que dos repositorios sean la colección de exactamente los mismos parches. Es perfectamente posible, ya que si Linus se queda con todos los cambios de Alan, será así.

El corolario de esto es que: Copia de Trabajo = Repositorio

Un repositorio, es simplemente un directorio que incluye, además, una copia «limpia» o «visible» de los ficheros que componen el proyecto. Por tanto publicar un repositorio es tan simple como hacer que esté disponible para lectura. De alguna forma, lo que se hace normalmente es publicarlo por web con la opción de Apache de Indexes, para que se puedan ver todos los ficheros y navegar por los directorios.

No sé si me explico ...

A ver, que busco un ejemplo

Esto es un repositorio Darcs publicado:

http://www.chneukirchen.org/repos/package/

Es coger un directorio de la máquina, y ponerlo visible por web.

Lo último que iba a decir es lo de que un repositorio Darcs no es más que un conjunto de parches así que con esto termina la introducción teórica

Introducción Práctica

Primeros pasos...

Para ayudar a explicar el funcionamiento de Darcs, echemosle imaginación al asunto. Imaginemos dos amigos: Linus y Alan. Con ellos simularemos el uso diario con Darcs. Linus será el que comience un proyecto y Alan el que, más adelante, le mande algún que otro parche ;)

Asumamos que nosotros somos Linus ...

Abran una terminal ...

En el directorio raiz del usuario linus (*/home/linus/*) creen un directorio llamado, p.ej., holamundo (¡OJO!, procuren no ponerlo en el desktop...)

```
mkdir holamundo
```

Métanse dentro y escriban "darcs initialize"[3].

```
cd holamundo
darcs initialize
```

¡Ya tienen su primer repositorio Darcs...!, vacío, pero lo tienen :-)

Fíjense en que ahora hay un directorio _darcs dentro.

```
.
`-- _darcs
    |-- current
    |-- inventories
    |-- inventory
    |-- patches
    `-- prefs
        |-- binaries
        |-- boring
        `-- motd

   5 directories, 4 files
```

Digamos que eso es el repositorio estrictamente hablando. Este directorio _darcs *nunca* se toca, excepto alguna que otra vez el directorio prefs, donde se pueden configurar algunas cosas.

3 Como comentario, añadir que el “bash-completion” (si usamos la shell bash y este está activado) tiene completado para darcs. O sea, que si escriben "darcs in<TAB>" rellena a "darcs initialize".

Estando dentro del directorio holamundo, vamos a crear un fichero, llamado por ejemplo, principal.txt, y vamos a añadirle el siguiente texto:

```
  echo “Esto es una prueba para Darcs.” > principal.txt
```

Grábenlo normalmente y ahora hagan "darcs add principal.txt" y luego "darcs record".

```
  darcs add principal.txt
  darcs record
```

Lo primero que aparece es solicitando una dirección de correo electrónico para identificar el repositorio. Si no se la hemos dado, entonces intenta coger de las variables de entorno "DARCS_EMAIL" o "EMAIL". Si tampoco existen, entonces nos la pide, y él la guarda en _darcs/prefs/author [[[[o email... comprobar]]]].

```
  Darcs needs to know what name (conventionally an email
address) to use as the
  patch author, e.g. 'Fred Bloggs <fred@bloggs.invalid>'.
If you provide one
  now it will be stored in the file '_darcs/prefs/author'
and used as a default
  in the future.  To change your preferred author address,
simply delete or edit
  this file.

  What is your email address? linustorvalds@example.com
```

Una vez le pongan el email, preguntará por cada parche. En este caso aparecen dos porque el primero es añadir el archivo y ...

```
  addfile ./principal.txt
  Shall I record this patch? (1/2) [ynWsfqadjkc], or ? for
help: y
```

... el siguiente es añadir el contenido.

```
  hunk ./principal.txt 1
  +Esto es una prueba para Darcs.
  Shall I record this patch? (2/2) [ynWsfqadjkc], or ? for
help: y
```

Darcs los considera como cosas diferentes por la forma que tiene de construir los parches. Es una de las (pocas, que yo recuerde) cosas raras que tiene Darcs.

Como hemos visto, hay que darle "y" a cada uno para aceptarlos.

Ahora pregunta el nombre del parche (para entendernos: una descripción). Le pueden poner una descripción cualquiera, por ejemplo: «Versión inicial».

```
What is the patch name? Versión inicial
```

Lo que pregunta luego es si quieres poner una descripción más larga. Si dices que sí y te salta el vi (o el programq que definas[4]) y ahí puedes meter una descripción más larga, para poder los detalles en cada línea y eso. Si le dices que no, se queda con una descripción de una frase/línea y ya está.

```
Do you want to add a long comment? [yn] n
Finished recording patch 'Versión inicial'
```

Con esto hemos terminado la explicación de mandar el primer parche/fichero.

Añadiendo más ficheros

Veamos lo que tenemos. Ejecutamos "*darcs changes*" y aparece:

```
darcs changes
Sat May  6 12:03:16 WEST 2006  linustorvalds@example.com
  * Versi\f3n inicial
```

Si te pasa como a mí, y visualizas mal el acento en la palabra 'Versión', prueba a ejecutar esta orden:

```
export DARCS_DONT_ESCAPE_ISPRINT=1
```

Volvamos a pedir los parches[5] disponibles, a ver si ahora se visualizan correctamente:

```
darcs changes
Sat May  6 12:03:16 WEST 2006  linustorvalds@example.com
  * Versión inicial
```

Una anotación importante: los cambios se llaman «parches» en Darcs.

A continuación vamos a añadir dos ficheros, uno llamado "algo.bak", con un contenido cualquiera, y otro llamado "algo.txt" con este contenido (el contenido de algo.bak da igual):

4 En realidad saltará el programa que tengas definido en una de las siguientes variables globales: EDITOR, VISUAL
5 En la terminología Darcs, a los cambios se les denomina «parches». Cada cambio genera su propio «parche»

```
echo “un contenido cualquiera” > algo.bak
echo “Otro fichero para probar.” > algo.txt
```

Ahora vamos a usar otra orden. Hagan «*darcs whatsnew*». No debería aparecer nada, porque no hemos hecho cambios en los ficheros que «pertenecen al proyecto».

```
darcs whatsnew
No changes!
```

Ahora hagan "*darcs whatsnew -l*" (L minúscula), o bien, en su forma larga "*darcs whatsnew –look-for-adds*":

```
darcs whatsnew --look-for-adds
a ./algo.txt
```

Esta orden significa que busque ficheros que parece que deberían estar dentro del repositorio para ver qué ficheros nuevos hay que todavía no les hemos hecho «*darcs add*».

Fíjense además que ha pasado de mostrarnos el fichero algo.bak. ¿Por qué?. Porque Darcs lo considera un «fichero aburrido», un «*boring file*». Este tipo de ficheros aburridos se pueden configurar en *_darcs/prefs/boring* (aunque normalmente no hace falta tocarlo, porque el contenido inicial es bastante completo).

Darcs intenta ignorar los típicos ficheros temporales y en general poco interesantes que prácticamente nunca queremos meter bajo control de versiones.

Para ficheros binarios, darcs no calcula las diferencias sino que guarda ambas versiones enteras. Lo que quiero decir es que internamente Darcs lo guarda usando ambas versiones enteras o sea, que con ficheros binarios grandes se gasta mucho espacio en disco.

En los ficheros de texto, p.ej, no se guardan todas las versiones, sino la inicial, y luego las diferencias con las siguientes, lo que ahorra mucho disco. Pero vamos, esto no nos importa.

Entonces, vamos a ver los cambios que hemos hecho, y a subirlo. Tenemos el nuevo fichero *algo.txt*, pero no está en el repositorio. Por tanto, "*darcs add algo.txt*" y luego "*darcs record*". Lo mismo de antes: responder dos veces «y», y meter una descripción.

```
  darcs add algo.txt
  darcs record
  addfile ./algo.txt
  Shall I record this patch? (1/1) [ynWsfqadjkc], or ? for
help: y
  hunk ./algo.txt 1
  +Otro fichero para probar.
  Shall I record this patch? (2/2) [ynWsfqadjkc], or ? for
help: y
  What is the patch name? Fichero algo.txt
  Do you want to add a long comment? [yn] n
  Finished recording patch 'Fichero algo.txt'
```

Diversión a dos

En este apartado vamos a sacar nuestra segunda personalidad y vamos a hacer una rama de nuestro propio proyecto. Es tan fácil como obtener una copia de nuestro proyecto, en otro directorio. En el ejemplo de repositorio que puse antes, http://www.chneukirchen.org/repos/package/, para obtener una copia haríamos en cualquier directorio (no lo hagan):

```
  darcs get http://www.chneukirchen.org/repos/package/
```

Si una persona hace un get, no queda por algun lado reflejado que hizo ese get, entre otras cosas, porque no se tiene permiso de escritura en ningún sitio. Lo que si puede quedar, son los logs de apache si lo haces por www, los del ftp, etc.

Lo que sí queda y es importante, es, en nuestra copia, una referencia al sitio de donde nos lo trajimos.

Vamos a hacer algo similar. Vamos al directorio anterior, al directorio «padre» de «holamundo», y vamos a hacer un get. Podemos elegir el nombre del directorio donde se trae la copia. Entonces, hagamos exactamente eso. ¿dónde está nuestro proyecto? En un directorio local, por lo tanto podemos hacer (fuera del «holamundo», insisto) "*darcs get holamundo holamundo-rama*", o, mejor, "*darcs get holamundo holamundo-alan*" por Alan Cox (si ya lo hicieron, simplemente renombren el directorio).

```
pwd
~/holamundo
cd ..
darcs get holamundo holamundo-alan
Copying patch 2 of 2... done!
Finished getting.
```

Ahora tenemos otra *CopiadeTrabajo/Repositorio* que «*sale de*» holamundo. Veamos lo que hay en el nuevo Repositorio. Entramos en el nuevo repositorio holamundo-alan y escribimos "*darcs changes*". El contenido debería ser los dos ficheros *.txt que estaban en el otro sitio, pero no el algo.bak.

```
cd holamundo-alan/
darcs changes
Sat May  6 19:36:36 WEST 2006  linustorvalds@example.com
  * Fichero algo.txt

Sat May  6 19:04:31 WEST 2006  linustorvalds@example.com
  * Versión inicial
```

El contenido del directorio holamundo-alan lo veremos mejor asi:

```
holamundo-alan
|-- _darcs
|   |-- checkpoints
|   |-- current
|   |   |-- algo.txt
|   |   `-- principal.txt
|   |-- inventories
|   |-- inventory
|   |-- patches
|   |   |-- 20060506180431-970cd-c9d17f7bca72534571c8e6.gz
|   |   `-- 20060506183636-970cd-ce2616af1a189ac514b80c.gz
|   `-- prefs
|       |-- binaries
|       |-- boring
|       |-- defaultrepo
|       |-- motd
|       `-- repos
|-- algo.txt
`-- principal.txt

    6 directories, 12 files
```

Ahora vamos a volver al primer directorio (el Repositorio original) durante un momento, abramos el fichero *principal.txt* y le vamos a añadir más líneas al final. Por ejemplo:

```
  cd ..
  cd holamundo
  echo -en "\nHay una segunda línea.\n\nUna tercera línea
remataría la jugada." >> principal.txt
```

Volvemos a hacer "darcs record".

```
  darcs record
  hunk ./principal.txt 3
  +Hay una segunda línea.
  +
  +Una tercera línea remataría la jugada.
  Shall I record this patch? (1/1) [ynWsfqadjkc], or ? for
help: y
  What is the patch name? Añadí dos líneas de ejemplo
  Do you want to add a long comment? [yn] n
  Finished recording patch 'Añadí dos líneas de ejemplo'
```

Ahora aparecerá sólo un parche porque el fichero ya existía de antes (y no lo estamos añadiendo aquí), le decimos que sí, y rellenamos la descripción.

Ahora vamos a hacer una de las virguerías de Darcs. Supongamos que nos hemos equivocado, porque se nos quedó algo atrás..., pues ahora vamos a deshacer el envío del cambio con "*darcs unrecord*". Aparecerá la lista de cambios, al primero le decimos que sí y luego escribimos «d» que es deshacer los que hemos elegido (sólo uno) y el resto, no.

```
  darcs unrecord

  Sat May  6 23:36:40 WEST 2006  linustorvalds@example.com
    * Añadí dos líneas de ejemplo
  Shall I unrecord this patch? (1/3) [ynWvpxqadjk], or ? for
help: y

  Sat May  6 19:36:36 WEST 2006  linustorvalds@example.com
    * Fichero algo.txt
  Shall I unrecord this patch? (2/3) [ynWvpxqadjk], or ? for
help: d
  Finished unrecording.
```

Si ahora hacemos "darcs diff" o "darcs whatsnew", veremos de nuevo los cambios como si nunca los hubiéramos mandado. Veamoslo:

```
  darcs diff
  diff -rN old-holamundo/principal.txt new-
holamundo/principal.txt
  1a2,5
  >
  > Hay una segunda línea.
  >
  > Una tercera línea remataría la jugada.
  \ No hay ningún carácter de nueva línea al final del
fichero
```

```
  darcs whatsnew
  {
  hunk ./principal.txt 3
  +Hay una segunda línea.
  +
  +Una tercera línea remataría la jugada.
  }
```

Ahora los volvemos a mandar para que esté todo como antes. Eso es "darcs record -a" y un título cualquiera.

```
  darcs record -a
  What is the patch name? Un título cualquiera
  Do you want to add a long comment? [yn] n
  Finished recording patch 'Un título cualquiera'
```

Ahora vamos al Repositorio nuevo, al de holamundo-alan, y ponemos "*darcs pull*".

```
  darcs pull
  Pulling from "/tmp/holamundo"...

  Sat May  6 23:52:21 WEST 2006  linustorvalds@example.com
    * Un título cualquiera
  Shall I pull this patch? (1/1) [ynWvpxqadjk], or ? for
help: y
  Finished pulling and applying.
```

Fíjense que aparece la lista de parches (sólo uno) que hay de nuevos en ... ¡el Repositorio original!, que no tenemos en el holamundo-alan porque son posteriores.

Le decimos que sí... y ya lo tenemos en el Repositorio holamundo-alan.

"*darcs changes*" ya nos lo muestra.

```
darcs changes
Sat May  6 23:52:21 WEST 2006  linustorvalds@example.com
  * Un título cualquiera

Sat May  6 19:36:36 WEST 2006  linustorvalds@example.com
  * Fichero algo.txt

Sat May  6 19:04:31 WEST 2006  linustorvalds@example.com
  * Versión inicial
```

Tienes un e-mail

Recuerden, del principio, que uno de los datos que nos pedía Darcs al ejecutar “darcs initialize”, era un email.

Supongamos que somos Alan... Si no tenemos ninguna forma sencilla de publicar nuestro repositorio, o si simplemente queremos mandar un cambio concreto a Linus, en ese caso Darcs puede mandar parches por correo de una manera muy sencilla.

Volvamos a suponer que somos *Linus* para poder profundizar en esto. Vamos a distinguir entre dos direcciones email que se pueden configurar con darcs..., pero primero, aseguremonos de situarnos dentro de nuestro directorio *holamundo*.

El nombre y dirección de email que está en ***_darcs/prefs/author*** es la que figurará como autor en nuestros parches, o sea, nuestro nombre y la dirección más bonita que tengamos y que queremos que los demás vean. Si el fichero ***_darcs/prefs/author*** no existiera, lo crearíamos así:

```
cd ..
cd holamundo
echo “linustorvalds@example.com” > _darcs/prefs/author
```

El fichero ***_darcs/prefs/email*** contiene la direccion a la que los demás nos enviarán nuestros parches, que no tiene porque ser igual.

```
echo “I am Linus Torvalds <linustorvalds@example.com>” >
_darcs/prefs/email
```

Practiquemos. Nos ponemos primero en el papel de Alan, y probemos a hacer un cambio, por ejemplo, abriendo el fichero *principal.txt*, modificamos algo y haciendo el “*darcs record*” correspondiente.

```
  cd ..
  cd holamundo-alan
  echo -ne "\nAñadimos alguna línea más\n" >> principal.txt
  lcabrera@master:/tmp/practica-darcs/holamundo-alan$ darcs
record
  Darcs needs to know what name (conventionally an email
address) to use as the
  patch author, e.g. 'Fred Bloggs <fred@bloggs.invalid>'.
If you provide one
  now it will be stored in the file '_darcs/prefs/author'
and used as a default
  in the future.  To change your preferred author address,
simply delete or edit
  this file.

  What is your email address? alancox@example.com
  hunk ./principal.txt 6
  +Añadimos alguna línea más
  +
  Shall I record this patch? (1/1) [ynWsfqadjkc], or ? for
help: y
  What is the patch name? Primer parche de Alan
  Do you want to add a long comment? [yn] n
  Finished recording patch 'Primer parche de Alan'
```

Antes de seguir, destaquemos una cosa muy importante: cuando creamos la copia del Repositorio original, no se nos pidió para ello una nueva dirección de correo. Sin embargo, ahora, cuando vamos a colaborar con un primer parche, el sistema SI nos ha solicitado nuestra dirección de correo. Este detalle es importante :)

Bien... ¿Y ahora qué? Muy fácil. Para mandar el parche, hacemos simplemente "*darcs send*" sin más, dentro desde nuestra carpeta de trabajo.

```
  darcs send
  Creating patch to "/home/linus/holamundo"...
  Patch bundle will be sent to: I am Linus Torvalds
<linustorvalds@example.com>

  Mon May  8 00:58:35 WEST 2006  alancox@example.com
    * Primer parche de Alan
  Shall I send this patch? (1/1) [ynWvpxqadjk], or ? for
help: y
  Successfully sent patch bundle to: I am Linus Torvalds
<linustorvalds@example.com>.
```

Darcs comprobará las diferencias entre ambos Repositorios y llegará a la conclusión de que Alan tiene un parche que todavía no ha enviado a Linus. Si Alan tuviera más parches hechos, Darcs se daría cuenta. Por lo tanto, le decimos que sí a los parches que queramos enviar.

Darcs pide la dirección de email de destino solo en caso de que el repositorio original no tenga bien configurado el *_darcs/prefs/email*.

El envío del correo implica tener bien configurado un servidor smtp en la máquina (postfix, exim, nullmailer o ssmtp o similares).

En caso de que no tengas ninguno, sino un programa de email en modo grafico, o un webmail, el procedimiento ya no es automático: necesitamos generar un fichero de parche y adjuntarlo manualmente. Este fichero se genera con:

```
darcs send -o nombre-fichero-con-el-parche
```

Bueno, en cualquier caso, de una manera u otra, terminamos con un fichero adjunto en un correo, en el buzón de Linus. Las cabeceras dirán algo así como:

```
To: I am Linus Torvalds <linustorvalds@example.com>
From: alancox@example.com
X-Mail-Originator: Darcs Version Control System
X-Darcs-Version: 1.0.6 (release)
DarcsURL: /home/linus/holamundo
Subject: darcs patch: Primer parche de Alan
```

Fíjense que Darcs es tan inteligente como para:

1. Saber que “holamundo-alan” viene de “holamundo”
2. Consultar la preferencia «email» de ese repositorio
3. Enviar el parche a esa dirección de correo

Bien. Ahora cambiamos de personalidad de nuevo y ahora nos creemos Linus Torvalds :-P

Recibimos un mensaje muy mono que dice que tenemos un parche.

Bien, entonces, como Linus, ¿qué hacemos con el fichero?

Lo primero será guardar el parche en algún sitio:

```
Guardar en archivo: /tmp/primer-parche-de-alan.dpatch
```

Luego, aplicaremos ese parche

```
darcs apply /tmp/primer-parche-de-alan.dpatch
Finished applying...
```

Nos aplica el parche en nuestro repositorio local, o sea tú ejecutas el "*darcs apply*" dentro del Repositorio que quieras.

¡¡y yatá!! ¿A que es guay? :-P

¡¡¡Con esto hemos acabado este manual!!!, aunque, por supuesto, Darks tiene un montón de cosas más pero este es su uso básico.

Para ver más documentacion de Darcs y mas opciones de trabajo, en http://darcs.net/ está el manual enlazado y un Wiki con un montón de información.

FAQ

> *¿Hay que tener alguna precacución especial con ficheros binarios?*

> *Dentro del repositorio, en _darcs/prefs/binaries, se listan las extensiones que el sistema interpreta como binarios.*

> *¿Funciona con los open documents y esas cosas ?*

> *Los editores de texto tienen su propio sistema de versiones para un documento. Un odt es un xml comprimido con zip, usease, que se podria tratar desde un SMC externo sin muchos problemas. Por servir, sirve, aunque los SCMs son _mucho_ más útiles cuando manejas texto sin formato (del que puedes ver en el vi).*

> *Darcs record no pilla to los caracteres (+y esto la 3\c2\ba muahaha :P):*

> *Realmente están grabados sin problemas, pero los muestra mal. Es necesario hacer lo siguiente para que los muestre bien:*
>
> *export DARCS_DONT_ESCAPE_ISPRINT=1*

> *Si Alan borra un archivo sin querer.... debe hacer:*

> *darcs revert [nombre del archivo]*

¿Qué es una «copia de trabajo», un «repositorio», «mandar cambios» («commit») y «actualizar una copia» («update»)?

- ***Copia de Trabajo:*** *es el documento en el que vamos realizando los cambios*
- ***Repositorio:*** *Almacen donde se va depositando el registro de cambios de cada documento.*
- ***Commit*** *(mandar cambios) [record]: En otros sistemas de control de versiones, es el acto de enviar los cambios en un documento al repositorio. En darcs se le conoce como 'record'.*
- ***Update*** *(actualizar la copia): Es la acción que actualiza nuestra Copia de Trabajo con las últimas aportaciones que existan en el Repositorio. En otras palabras, es dejar la Copa de Trabajo 'a la última'.*

¿Si dos usuarios abren y cambian el mismo fichero a la vez hay problemas?

Realmente será problema del vi o del Emacs, es decir, del editor de texto, el darse cuenta de que dos a la vez intentan abrir el mismo fichero. Para Darcs sólo importa el contenido que tengan los ficheros al hacer darcs record y similares. O sea, que el darcs es individual y cada usuario tiene que tener su darcs. Es lo normal que cada uno pueda trabajar a su rollo, digamos sobre todo por lo fácil que es trabajar con repos independientes y luego compartir parches.

Enlaces, Créditos y Documentación adicional.

Darcs, Página principal del Proyecto Dars: http://darcs.net/

Manual de Dars: http://darcs.net/manual/node4.html#SECTION00410000000000000000

SCM: http://en.wikipedia.org/wiki/Software_Configuration_Management

CVS, Concurrent Versions System,cvs: http://www.nongnu.org/cvs/

Subversion, svn: http://subversion.tigris.org/

GNU Arch Revision Control System, arch: http://gnuarch.org/revc/index.html

Linus Torvalds, Linus: http://es.wikipedia.org/wiki/Linus_Torvalds

Alan Cox, Alan: http://es.wikipedia.org/wiki/Alan_Cox

Créditos:

Ponencia impartida por: Esteban Manchado (aka zoso)

Asistentes: llorens, amd77, lcabrera, Mr-Petah (formateo), El ornitorrinco enmascarado, mencey, Parallax, Artir.

Licencia:
