Capítulo 7
Sistema de control de versiones

1.- Introducción

Un **sistema de control de versiones** o SVC (System Version Control) es una herramienta que registra los cambios realizados sobre un archivo o conjunto de archivos de un proyecto a lo largo del tiempo, de modo que puedas recuperar la versión de esos archivos en un estado anterior en el tiempo.

El sistema de control de versiones es de gran utilidad para entornos de **desarrollo colaborativo** donde varios disenadores, maquetadores y/o programadores trabajan sobre un mismo proyecto. En todo momento tenemos los cambios que se realizan sobre cada uno de los ficheros por los diferentes programadores y podemos recuperar un fichero en un estado anterior.

En este capítulo nos vamos a centrar en la herramientas **Git y Subversión**, pero existen muchas otras como CVS, Mercurial, Bazaar, SourceSafe, etc.

2.- Definición y características de los SCV

Es un sistema que registra los cambios realizados sobre un archivo o conjunto de archivos a lo largo del tiempo, de modo que se pueda recuperar versiones específicas de los mismos en un determinado momento.

Un SVC posee tres capacidades importantes:

- **Reversibilidad:** retornar a un estado anterior del proyecto en caso de fallos.
- **Concurrencia:** Muchas personas modificando el mismo código o documento.
- **Anotación:** Adjuntar información relevante de los cambios realizados.

Un sistema de control de versiones debe proporcionar:

- Mecanismo de almacenamiento de los elementos que deba gestionar (ej. archivos de texto, imágenes, documentación...).
- Posibilidad de realizar cambios sobre los elementos almacenados (ej. modificaciones parciales, añadir, borrar, renombrar o mover elementos).
- Registro histórico de las acciones realizadas con cada elemento o conjunto de elementos (normalmente pudiendo volver o extraer un estado anterior del producto).

3.- Conceptos Básicos en los Sistemas de control de versiones

Repositorio: lugar en el que se almacenan los datos actualizados e históricos de cambios (sistema de archivos en un disco duro, un banco de datos, etc).

Revisión: Versión determinada de la información que se gestiona.

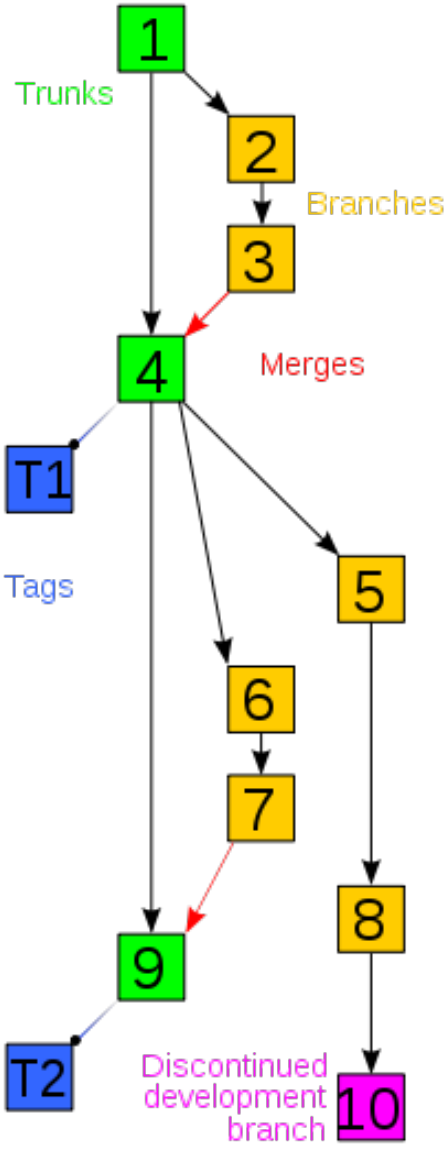
Tags: Permiten identificar de forma fácil revisiones importantes en el proyecto.

Módulo: Conjunto de directorios y/o archivos dentro del repositorio que pertenecen a un proyecto común.

Branch: Es una copia del proyecto aislada, de forma que los cambios realizados no afecten al resto del proyecto y vice versa, excepto cuando los cambios sean "unidos" de un lado al otro.

Baseline: Una revisión aprobada de un documento o fichero fuente, a partir del cual se pueden realizar cambios subsiguientes.

Checkout: crea una copia de trabajo local desde el repositorio.

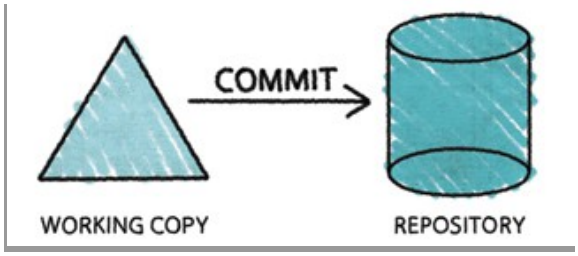


Merge: Une dos grupos de cambios en un archivo (o grupo de archivos), generando una revisión unificada.

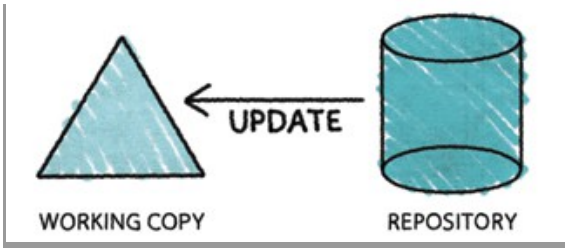
Conflicto: Sucede cuando dos o más personas intentan realizar diferentes cambios en la misma porción de código.

Commit: Consiste en realizar un cambio local en el proyecto y luego almacenar dicho cambio en el repositorio.

Change set: Conjunto de cambios realizados en un único commit.

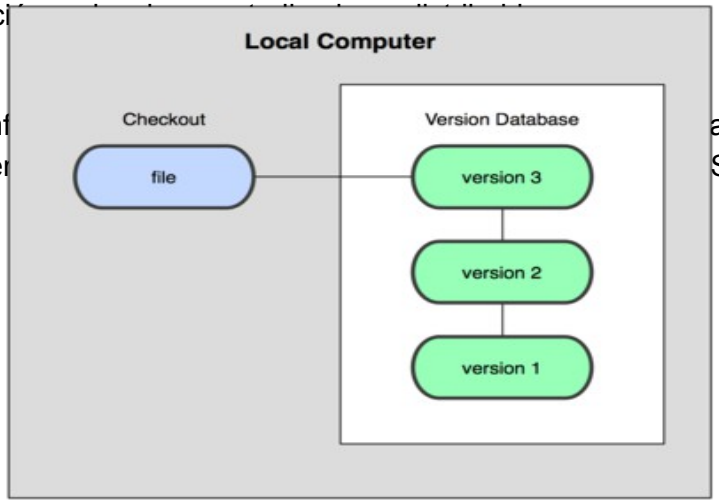


Update: Integra los cambios que han sido realizados en el repositorio en la copia de trabajo local.



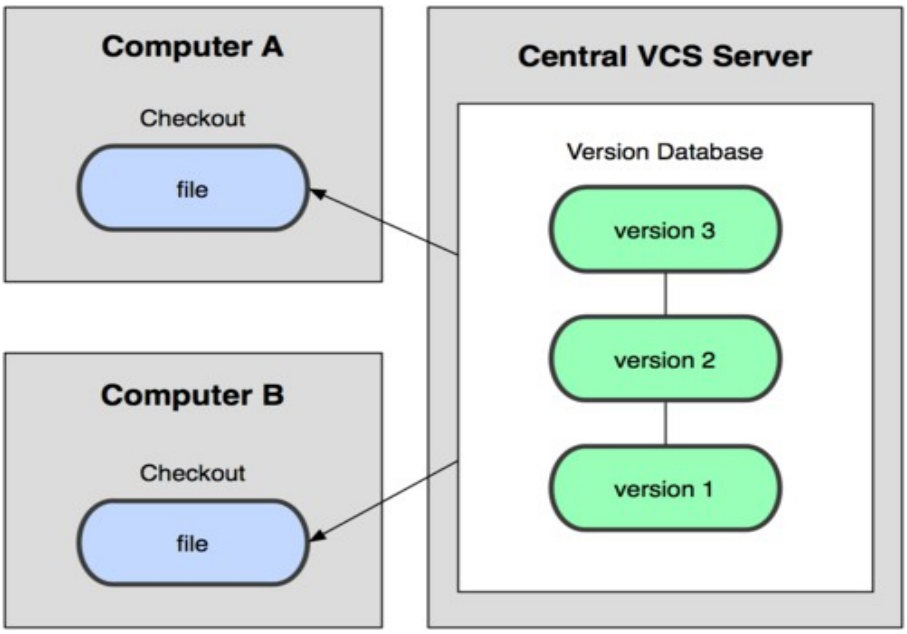
4.- Clasificación de los SCV

Podemos clasificar los sistemas de control de versiones según la arquitectura para almacenar la informaci

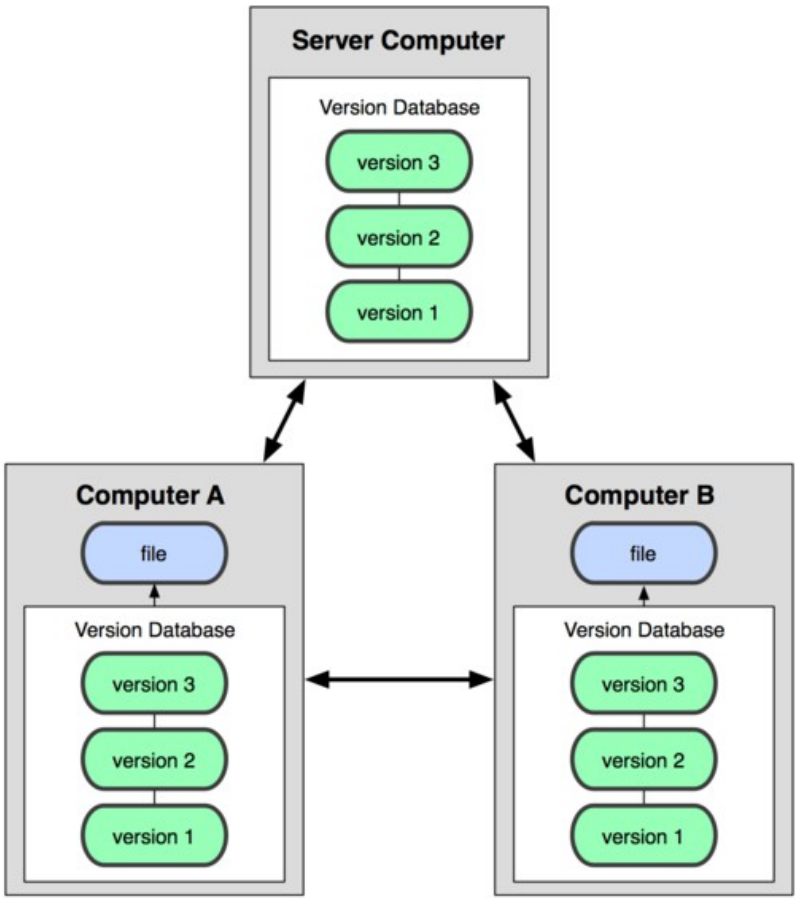


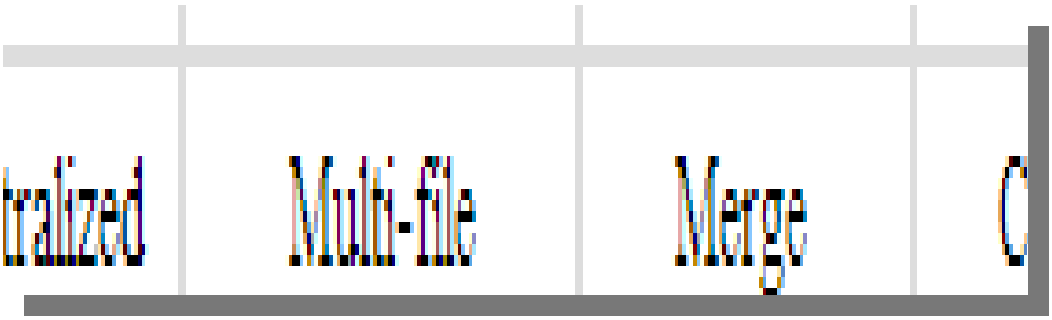
- **Locales:** La inf al con lo que no sirve para trabajar e System).

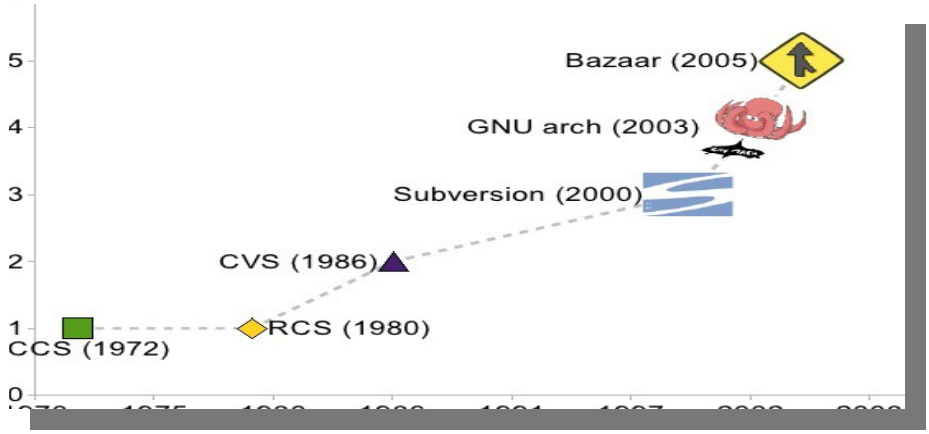
- **Centralizados o CVCS (Centralized Version Control System):** La información se guarda en un servidor dentro de un repositorio centralizado. Existe un usuario o usuarios responsables con capacidad de realizar tareas administrativas a cambio de reducir flexibilidad, necesitan la aprobación del responsable para realizar acciones, como crear una rama nueva. Ejemplos: Subversión y CVS.



- **Distribuidos o DVCS (Distributed Version Control System)**: Cada usuario tiene su propio repositorio. Los distintos repositorios pueden intercambiar y mezclar revisiones entre ellos. Es frecuente el uso de un repositorio, que está normalmente disponible, que sirve de punto de sincronización de los distintos repositorios locales. Ejemplos: Git y Mercurial, Bazaar y Darcs.


Server Computer
Version Database
version 3
version 2
version 1
Computer A
file
Version Database
version 3
version 2
version 1
Computer B
file
Version Database
version 3
version 2
version 1


tralized
Multi-file
Merge

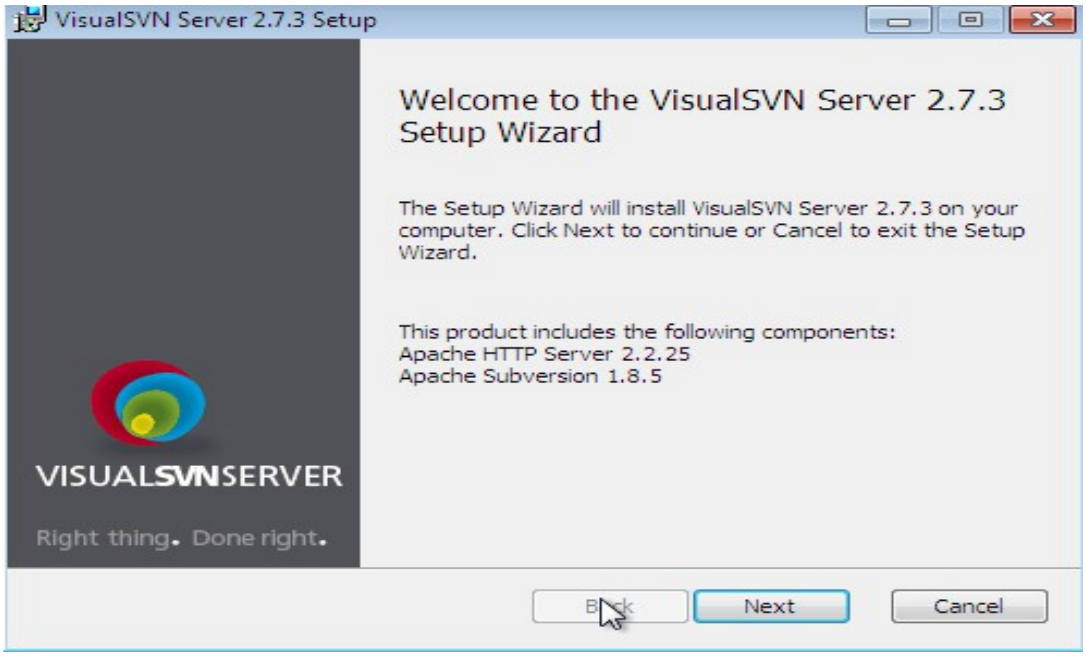

5
4
3
2
1
0
Bazaar (2005)
GNU arch (2003)
Subversion (2000)
CVS (1986)
RCS (1980)
CCS (1972)

5.- Subversión

6.- Instalación y configuración de subversión en modo gráfico Windows

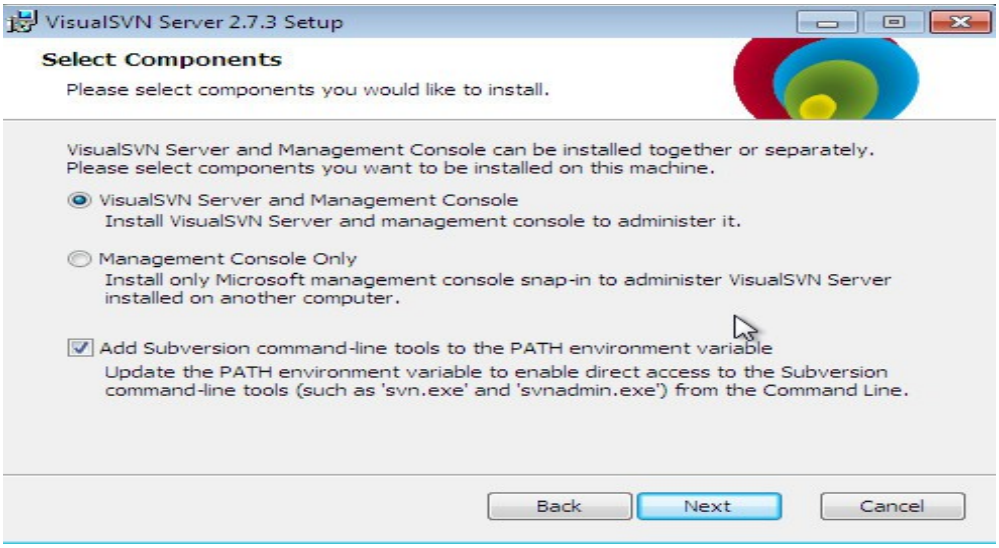
1.- Descargamos el servidor y lo instalamos

http://www.visualsvn.com/server/download/



Aceptamos la licencia:

Instalamos el servidor y la consola:



Instalamos la edición estandar:

Seleccionamos el directorio donde lo instalamos y el directorio del los repositorios:

Comenzamos la instalación:

Permitimos se ejecute el programa de instalación:

Finalizamos el asistente y arrancamos Visual SVN:

Permitimos la ejecución de VisualSVN:

Creamos un nuevo repositorio:

Le damos nombre:

Creamos un repositorio vacío:

Damos permiso de lectura y escritura a todos los usuarios:

Finalizamos la creación:

Creamos un nuevo usuario:

Modificamos los permisos de lectura y escritura del usuario:

Instalamos cliente TortoiseSVN

TortoiseSVN es un cliente subversion con licencia GNU GPL. Se integra al explorador de Windows y puede incorporar los comandos de subversión. Lo descargamos e instalamos.

SVN Checkout: Primero nos conectamos al servidor.

https://ip/svn/repositorio

Posteriormente:

SVN Update: Descargar repositorio

SVN Commit: Actualizar repositorio

Instalamos Subversión en ubuntu

http://proyectosbeta.net/2013/07/instalar-un-servidor-svn-en-ubuntu-13-04/

Herramienta gráfica cliente SVN para ubuntu

sudo apt-get install rapidsvn

Conceptos Subversión

Repositorio (repository)

Es el almacén donde se guardan todos los ficheros y sus versiones. Todas las operaciones registran un punto de referencia en el repositorio: por ejemplo, hacer un check-out (extraer un fichero del repositorio) o un commit (incorporar un fichero al repositorio).

Copia de trabajo (working copy)

Inicialmente es el directorio que se obtiene al hacer el check-out de un proyecto almacenado en un repositorio. La copia de trabajo contiene información para que Subversion pueda saber qué ficheros están sincronizados y cuáles no. Es el directorio donde después de hacer check-out editaremos nuestros ficheros y los iremos incorporando en el repositorio. La copia de trabajo también contiene ficheros extras que ayudan a mantener la sincronía con el repositorio y facilitan la implementación de las órdenes. Estos ficheros de administración se encuentran en la carpeta .svn, y no se deben borrar.

Directorio para inicializar el repositorio

Es el directorio que contiene los ficheros de la práctica o proyecto que queremos poner bajo control de revisiones. Si ejecutamos la orden svn import, Subversion incorporará todos los ficheros al repositorio. IMPORTANTE: Una vez finalizada la importación de ficheros, este directorio no se convierte en una copia de trabajo. Por lo tanto, para seguir trabajando con los ficheros que contenía este directorio, necesitaremos ejecutar svn check-out para obtener una copia de trabajo limpia y ubicada, para mayor seguridad, en un nuevo directorio. Entonces podremos empezar a modificar el código de la práctica que hay en el directorio donde se ha hecho el check-out. El directorio que hemos usado para inicializar el repositorio se puede borrar.

Sincronización

Es la clave para realizar el control de versiones. Debe ser nuestro objetivo en todo momento: tener un repositorio y una copia de trabajo en sincronía. Si partimos de una copia de trabajo sincronizada, nos pueden pasar dos cosas:

1. La copia de trabajo se ha modificado y no está sincronizada con el repositorio.

2. El repositorio ha sido actualizado por otro miembro del grupo de trabajo, en el caso de que haya un grupo de programadores, y el repositorio no está sincronizado con la copia de trabajo.

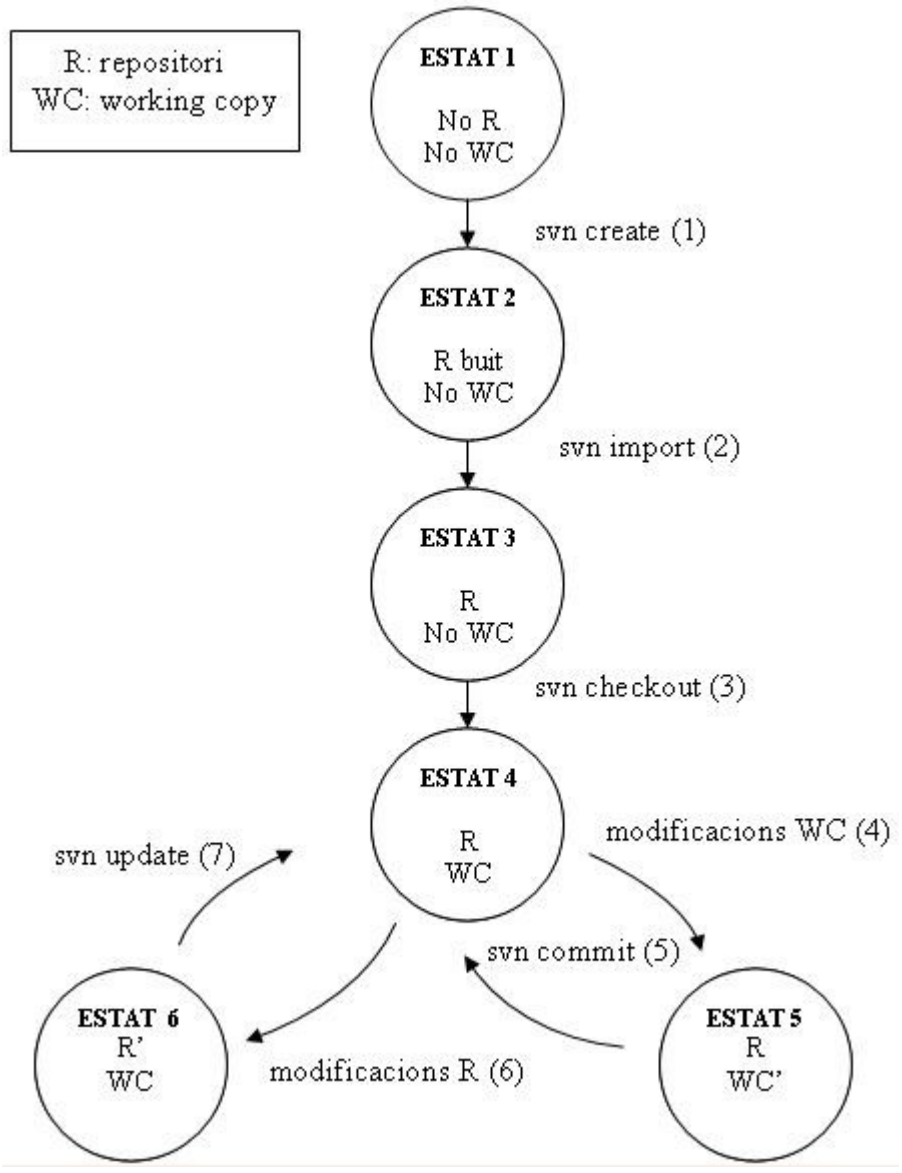
Para conseguir la sincronía en cada una de las dos situaciones anteriores, habrá que ejecutar:

1.svn commit, para actualizar el repositorio.

2.svn update, para descargarnos los cambios a nuestra copia de trabajo.

Ciclo habitual de trabajo

Podemos ver el ciclo habitual de trabajo en el siguiente gráfico que explicaremos a continuación:



- Estado 1

  No existe ni un repositorio ni una copia de trabajo. En este punto, podemos ejecutar la orden svn create para crear un repositorio.

  svn create

- Estado 2

  Disponemos de un repositorio, pero está vacío. Con la orden

svn import

cargamos el repositorio con los ficheros del directorio.

- Estado 3

  Disponemos de un repositorio con datos, pero todavía no podemos modificar sus ficheros. Hay que crear la copia de trabajo:

  svn checkout

- Estado 4

  El repositorio contiene datos y existe una copia de trabajo sincronizada. Ahora podemos modificar nuestra copia de trabajo:

  - Modificamos ficheros con un editor de texto, o
  - Añadimos, borramos, copiamos o renombramos ficheros:
    - **svn add**
    - **svn delete**
    - **svn copy**
    - **svn move**

- Estado 5

  Tenemos una copia de trabajo modificada con respecto al repositorio. Ahora necesitamos guardar los cambios en el repositorio:

  svn commit

  Con esta operación volvemos al estado 4.

- Estado 6 se puede llegar a este estado cuando se trabaja en grupo sobre un repositorio, y alguien lo modifica. En este caso, tendremos que incorporar estos cambios a nuestra copia de trabajo. Ejecutamos la orden:

  svn update

  Con esta operación volvemos al estado 4.

En cualquier momento del desarrollo podemos consultar las diferencias entre el repositorio y nuestra copia de trabajo, mediante las órdenes: **svn list, svn estatus, svn diff.**

Cuando se trabaja con repositorios, se suele utilizar una estructura formada por tres subdirectorios: trunk, branches y tags:

- Trunk:

  Es el directorio que contiene la última versión del proyecto que se está desarrollando, la versión común a todos los desarrolladores.

- Branches:

Este directorio contiene varios subdirectorios, uno para cada versión de "Trunk" que se quiera hacer. Es normal que cada desarrollador trabaje sobre su propio subdirectorio en "Branches", y que después ponga todo en común en el directorio "Trunk" con la orden **svn merge**. Cada subdirectorio de "Branches" normalmente se inicializará haciendo una copia de "Trunk" en el subdirectorio mediante **svn copy**.

- Tags:

En "Tags" se guardan copias de seguridad (snapshots) del proyecto, ejecutando la orden **svn copy**. La única diferencia entre los directorios "Tags" y "Branches" es que nunca se modifican ficheros pertenecientes al directorio "Tags".

Introducción Git

Práctica Control de versiones Git en local

En esta práctica vamos a realizar una instalación de versiones básica con Git. Vamos a crear un repositorio y vamos a añadir y modificar un fichero en el.

1.- Instalamos git

sudo apt-get install git git-core

2.- Creamos un usuario

git config --global user.name "Aitor LA"

aitor@dinux:~$ git config --global user.email "aitor@kaixo.com"

Nota: El usuario va entre comillas

3.- Creamos una carpetas

mkdir /home/aitor/git

4.- Nos posicionamos en la carpeta e iniciamos git

cd /home/aitor/git

git init

Initialized empty Git repository in /home/aitor/git/.git/

5.- Vemos que no tenemos nada cacheado

git status

6.- Creamos fichero.txt

vi fichero.txt en un principio vacío

7.- Vemos que el fichero se ha creado pero no se ha agregado nada

git status

En la rama master

#

Commit inicial

#

Archivos sin seguimiento:

(use «git add <archivo>...» para incluir lo que se ha de ejecutar)

#

fichero.txt

no se ha agregado nada al commit pero existen archivos sin seguimiento (use «git add» para darle seguimiento)

8.- Añadimos el fichero

git add fichero.txt

Nota: podemos añadir todos los archivos haciendo git add .

y vemos el estado:

git status

En la rama master

#

Commit inicial

#

Cambios para hacer commit:

(use «git rm --cached <archivo>...« para eliminar stage)

#

#	archivo nuevo:	fichero.txt

#

Nota: Para pasar de la zona de cache intermedia al repositorio git hay que hacer un commit.

9.- Hacemos un commit para hacer válidos los cambios

git commit -m “Primera prueba con git”

[master (root-commit) ec3b174] Primera prueba con git

1 file changed, 0 insertions(+), 0 deletions(-)

create mode 100644 fichero.txt

Nota: Si no le ponemos un comentario nos abre un editor vi para que isertemos el mensaje.

9.- Modificamos el fichero.txt y hacemos un commit

vi fichero.txt

git commit -a -m “Modificamos el fichero”

[master 8c0d578] Modificamos el fichero

1 file changed, 1 insertion(+)

Nota: Para no tener que hacer un add podemos pasar el parámetro -a

10.- Instalamos gitk que nos permite ver los cambios en modo gráfico

apt-get install gitk

y lo ejecutamos

gitk

Práctica Control de versiones Git en la nube

La idea es utilizar github.com para tener un repositorio en la nube.

11.- Nos registramos en hithub.com. Yo lo he hecho con el usuario aitorla.

Pinchamos dentro de nuestro usuario y dentro de la pestaña repositorios pinchamos en New:

Damos el nombre y la descripción a nuestro repositorio. Pinchamos en crear repositorio:

Vemos que nuestro repositorio se encuentra en la url:

https://github.com/aitorla/ejemplo.git

Creamos un fichero README.me y lo enviamos a nuestro repositorio:

touch README.md

```
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/aitorla/ejemplo.git
git push -u origin master
```

Username for 'https://github.com': aitorla

Password for 'https://aitorla@github.com':

Counting objects: 3, done.

Writing objects: 100% (3/3), 208 bytes, done.

Total 3 (delta 0), reused 0 (delta 0)

To https://github.com/aitorla/ejemplo.git

 * [new branch]   master -> master

Branch master set up to track remote branch master from origin.

Enviamos el repositorio

```
git remote add origin https://github.com/aitorla/ejemplo.git
git push -u origin master
```

Borrar archivos

rm fichero1.txt

git status

git rm fichero1.txt

git commit

Mover archivos

mkdir ficherosviejos

mv fichero2.txt ficherosviejos/fichero2.txt

git status // vemos que hay una nueva carpeta ficheros viejos y se ha borrado ficheros2.txt

git rm fichero2.txt

git add ficherosviejos/fichero2.txt

Deshacer cambios

vi ficheros.txt

git add ficheros.txt (y nos damos cuenta que no queremos pasar de cache al repositorio)

git reset HEAD fichero3.txt

git checkout – - nombre archivo

Examinar el registros

git log // visualiza los diferentes commit que se han realizado. Abre un vi por lo tanto se cierra con :q

git log – - oneline // solo una linea

Git RAMAS

git branch rama1 // crea la rama 1

git branch // vemos las ramas que tenemos con * la rama seleccionada

get checkout rama1 // seleccionamos la rama1

git checkout -b rama2 // crea la rama y la selecciona

git brach -d rama2 // borramos una rama

Git Merges union de ramas de ramas

git log – - online – - decorate

git checkout master

git merge rama1

Resolucion de conflictos

Si hay cambios en la rama principal y paralela en la misma region de codigo

git checkout -b ramacorrecion

vi fichero1.txt

git add fichero1.txt

git commit

git checkout -b ramacorrecion2

vi fichero1.txt

git add fichero1.txt

git commit

git merge ramacorrecion

git merge ramacorrecion2

git status

vi fichero.txt // lo dejams como creemos que esta correcto

git add fichero.txt // automaticamente soluciona el conflicto

git status

Ir a un commit anterior

git log - - oneline

git checkout 12fss886 // identificador del commit

git checkout master // volvemos a la rama inicial

TAG: alias a commits concretos

Para en lugar del identificador del commit utilizar un nombre

git tag // ver lista de tags

git tag version2.0 // da el tag al master

git tag 12fss886 version10 // da el tag a

git checkout version10

Push y pull a repositorios remotos

git remote add origin /Dropbox/repositorio.git // o direccion github

git push origin master // donde master es la rama que querems enviarse

git pull origin master // descargamos los cambios que han hecho hay que hacerlo de vez en cuando

git pull - -rebase // para no vperder cambios iportantes en local.

Git: Branch y Merges

1.- Creamos una rama:

git checkout -b Rama1

Ahora hacemos cambios sobre la rama

git checkout Rama1

2.- Unimos las dos ramas:

git checkout master

git checkout Rama1

3.- Borramos la rama

git branch -d Rama1

Nota: para borrar una rama que no ha sido mezclada utilizaremos -D en lugar de d.

COMANDOS BÁSICOS GIT

Instalamos git

sudo apt-get install git git-core

Nota: Si queremos instalarlo en otro sistema que no sea linux lo podemos hacer desde: http://git-scm.com/

Creamos un usuario

git config --global user.name "Aitor LA"

git config --global user.email "aitor@kaixo.com"

Nota: El usuario va entre comillas. Nos permite saber quien ha realizado los cambios.

Iniciamos el repositorio

mkdir /home/aitor/git

cd /home/aitor/git

git init

Nota: Se crea un directorio oculto .git, que es donde se guardan los cambios

Comprobar el estado

git status

Nota: Nos permite saber si hay algún fichero cacheado, modificado,..

Añadir fichero a la caché intermedia o stage

Vamos a crear un fichero, vemos el estado que no indica que no está cacheado, lo añadimos a la caché. Esto hay que hacer antes de un commit.

vi fichero.txt

git status

git add fichero.txt

git status

Realzar un commit

git commit -m “Primera prueba con git”

[master (root-commit) ec3b174] Primera prueba con git

1 file changed, 0 insertions(+), 0 deletions(-)

create mode 100644 fichero.txt

Nota: Para pasar de la zona de cache intermedia al repositorio git hay que hacer un commit.

Nota: Si no le ponemos un comentario nos abre un editor vi para que insertemos el mensaje.

Añadimos un registro y realizamos un commit directamente

vi fichero.txt

git commit -a -m “Modificamos el fichero”

[master 8c0d578] Modificamos el fichero

1 file changed, 1 insertion(+)

Nota: Para no tener que hacer un add podemos pasar el parámetro -a

Borramos archivos

rm fichero1.txt

git status

git rm fichero1.txt

git commit

Movemos archivos

mkdir ficherosviejos

mv fichero2.txt ficherosviejos/fichero2.txt

git status // vemos que hay una nueva carpeta ficheros viejos y se ha borrado ficheros2.txt

git rm fichero2.txt

git add ficherosviejos/fichero2.txt

Deshacer cambios

vi ficheros.txt

git add ficheros.txt // y nos damos cuenta que no queremos pasar de cache al repositorio)

git reset HEAD fichero3.txt

git checkout -- nombre_archivo

Examinar el registros

git log // visualiza los diferentes commit que se han realizado. Abre un vi por lo tanto se cierra con :q

git log -- oneline // solo una linea

Trabajando con ramas (Branch)

git branch rama1 // crea la rama 1

git branch // vemos las ramas que tenemos con * la rama seleccionada

get checkout rama1 // seleccionamos la rama1

git checkout -b rama2 // crea la rama y la selecciona

git brach -d rama2 // borramos una rama

Union de Ramas (merge)

git log -- online -- decorate

git checkout master

git merge rama1