



# GESTIÓN DEL SOFTWARE, COLABORACIÓN Y EL CONTROL DE VERSIONES

© 2017, ACTIBYTI PROJECT SLU, Barcelona  
Autor: Ricardo Ahumada

# ÍNDICE DE CONTENIDOS

1. Gestión del software, colaboración y el control de versiones
2. Subversión
3. GIT

# 1

## GESTIÓN DEL SOFTWARE, COLABORACIÓN Y EL CONTROL DE VERSIONES

# Evolución del software

## ➤ Durante el desarrollo

- El desarrollo del software siempre es progresivo, incluso en el ciclo de vida en cascada
- El desarrollo evolutivo consiste, precisamente, en una evolución controlada (ciclo de vida espiral, prototipos evolutivos)



## ➤ Durante la explotación

- Durante la fase de mantenimiento se realizan modificaciones sucesivas del producto
- Además de ello, el desarrollo de software se realiza normalmente en equipo, por lo que es necesario integrar varios desarrollos en un solo producto



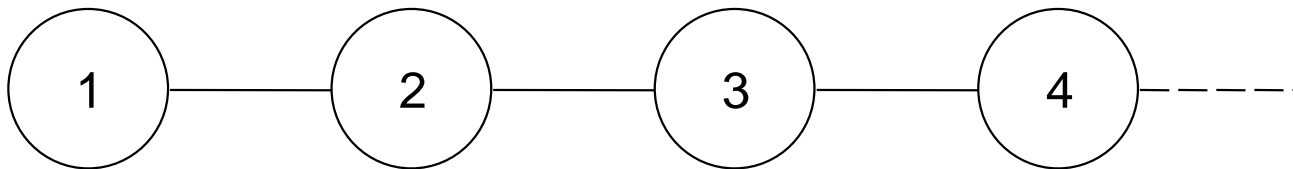
# Control de versiones

## ➤ **Motivo: evolución**

- El software cambia con el tiempo, por diversas razones
- Es necesario controlar esta evolución
- En algunas ocasiones, suele ser necesario recuperar versiones antiguas

## ➤ Concepto de **versión** (revisión)

- “**Versión**” es la Forma particular que adopta un objeto en un contexto dado
- Desde el punto de vista de evolución, es la forma particular de un objeto en un instante dado. Se suele denominar “**revisión**”



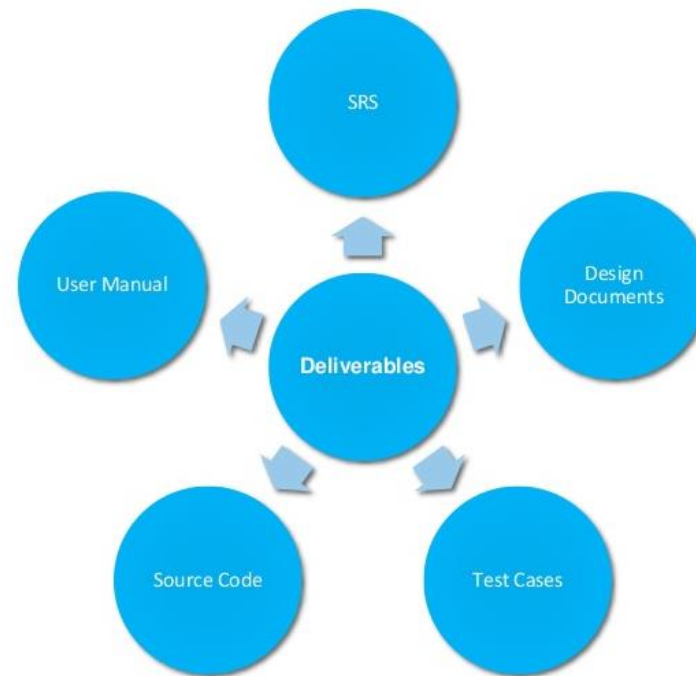
# Control de versiones

- El control de versiones es el arte del manejo de los cambios en la información”.
- Herramienta crítica para los programadores.
- Facilita volver a una versión anterior.
- Facilita el trabajo colaborativo.
- Se envían sólo las diferencias realizadas.
- El control de versiones no sólo es necesario para el software.

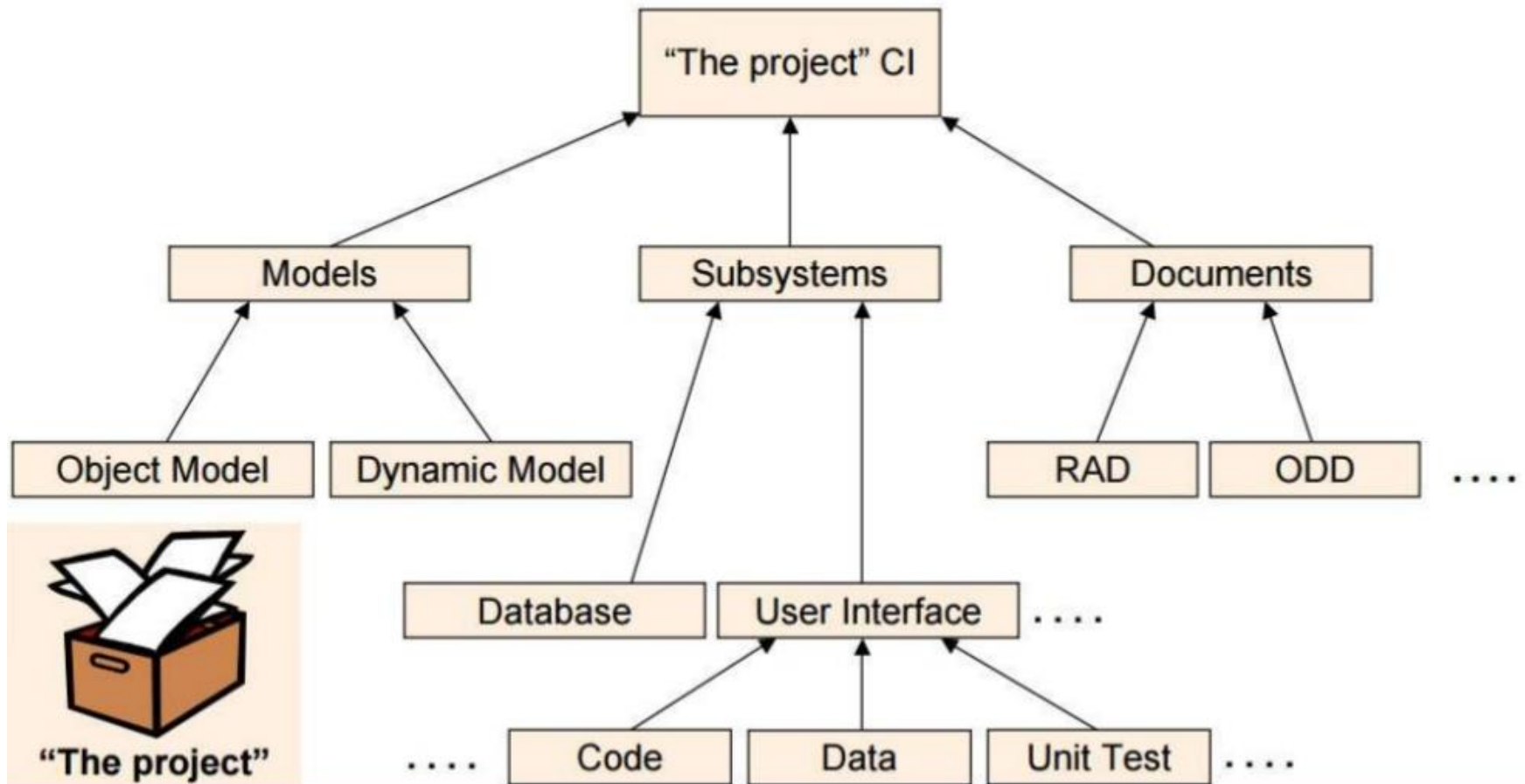
# Gestión de configuración

- Un sistema software comprende distintos componentes (items), que evolucionan individualmente
- Hay que garantizar la **consistencia del conjunto del sistema**
- Una “**configuración**” es una combinación de versiones particulares de los componentes que forman un sistema consistente
- Desde el punto de vista de evolución, es el conjunto de las versiones de los objetos componentes en un instante dado

- **Configuration Item (CI)** refers to the fundamental structural unit of a **SCM**
- Deliverables of Large Software Development Effort



# Ejemplo de ítems sujetos a gestión de configuración





# Control de cambios

## ➤ Línea base

- Llamaremos “línea base” a una configuración operativa del sistema software
- La evolución del sistema puede verse como evolución de la línea base

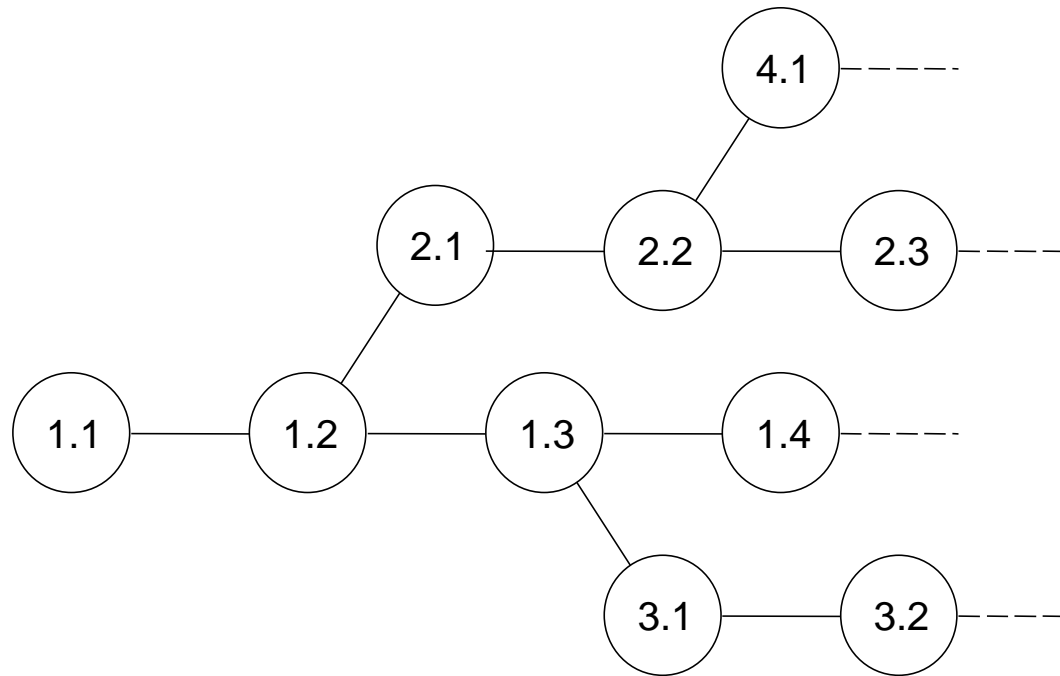
## ➤ Concepto de cambio

- Un “**cambio**” es el paso de una versión de la línea base a la siguiente
- Puede incluir modificaciones del contenido de algún componente, y/o modificaciones de la estructura del sistema, añadiendo o eliminando componentes

# Variantes

- Un sistema software puede adoptar distintas formas (configuraciones) dependiendo del lugar donde se instale.
  - Por ejemplo, dependiendo de la plataforma que la soporta (máquina + S.O.), o de las funciones opcionales que haya de realizar o no
- Una “**variante**” es una versión de un componente (o de la configuración global) que evoluciona por separado
- Las variantes representan una **variación espacial**, mientras que las revisiones representan una variación temporal

# Variantes



- **TRONCO:** Variante principal, p.ej. 1.1-1.2...
- **RAMAS:** Variantes secundarias, p.ej: 2.1..., 3.1...
- **DELTA:** Cambios de una revisión respecto a la anterior
  - Delta 3.2 =  $(3.1 \textcircled{R} 3.2)$

# Repositorio

- Es habitual centralizar el almacenamiento de los componentes de un mismo sistema, incluyendo las distintas versiones de cada componente. Este **almacén común** se denomina “**repositorio**”
- El repositorio permite ahorrar espacio de almacenamiento, evitando guardar por duplicado elementos comunes a varias versiones o configuraciones
- El repositorio facilita el almacenar información de la evolución del sistema (historia), y no sólo de los componentes en sí
- A veces se confunde el término 'repositorio' con el de 'línea base', pero no son lo mismo.
- Existen dos grandes tipos de sistemas de repositorios:
  - Sistema centralizados
  - Sistema distribuidos

# Sistema centralizados

- El repositorio reside en un punto único central
- Un poco más sencillos de utilizar que los distribuidos.
- Se tiene un control total sobre las versiones.
- Limitaciones en el acceso.
- Puede haber menos conflictos a resolver.

## Ejemplos:

- CVS y Subversion.

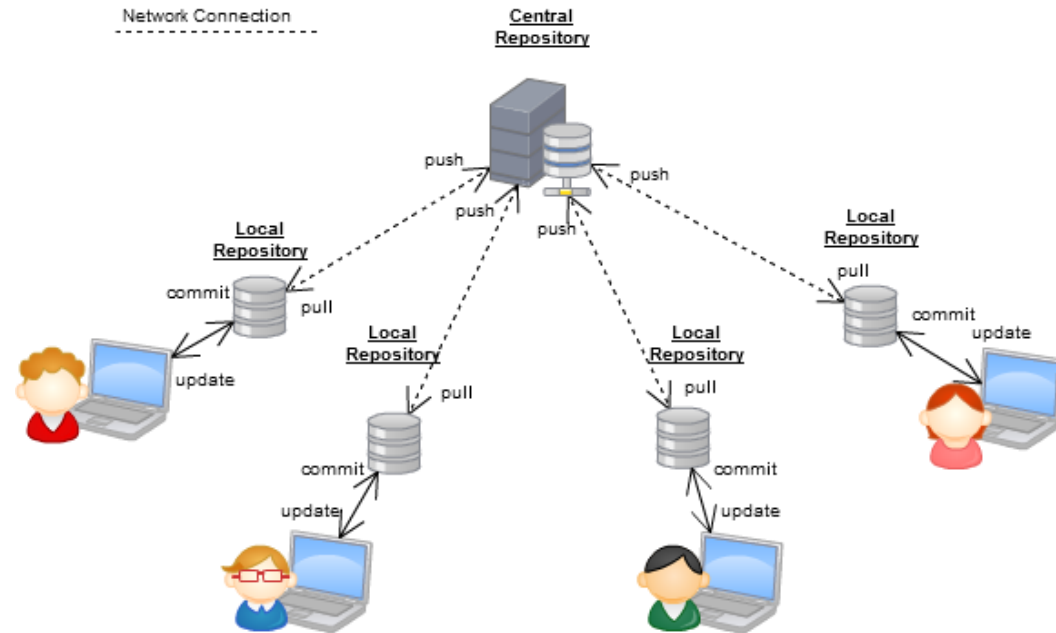


# Sistema distribuidos

- Existe un repositorio local y otro central
- Se puede trabajar de forma local. Permite operaciones más rápidas.
- No necesitas permiso para participar en los proyectos.
- No se depende de una sola máquina física.
- Puedes seguir teniendo un control centralizado del proyecto.

## Ejemplos:

- Git, Bazaar y Mercurial.



# 2

## SUBVERSIÓN

# Subversion

Es un sistema de gestión de versiones

- Ampliamente usado
- Tiene un esquema centralizado
- Multiplataforma
- Libre
- **Libro:** Version Control with Subversion
  - <http://svnbook.red-bean.com/en/1.7/index.html>





# Conceptos básicos

- Repositorio
- Copia local
- Revisión
- Etiqueta (tag)
- Actualización (update)
- Publicación (commit)
- Conflicto



## Instalando las herramientas



# Instala las siguientes herramientas

- Descargar SVN
  - <https://subversion.apache.org/packages.html>
  - Clientes:
    - SlikSVN (consola\*)
    - TortoiseSVN (gráfico)
  
- Crear una cuenta en Assembla o riouxSVN
  - [https://app.assembla.com/user/new\\_signup](https://app.assembla.com/user/new_signup)
  - <https://riouxsvn.com/>



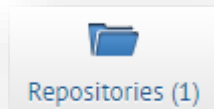
## Creando un repositorio SVN

# Creando un proyecto con SVN



## ➤ Crear un repositorio en riouxSVN

### ➤ Repositories



### ➤ Create Repository

Create new repository...

### ➤ Indica el título y nombre → Next

### ➤ Escoge la opción con trunk, branches y tags →

### ➤ Confirm

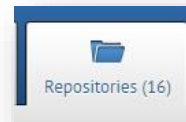
Creation Options (More info)



Create trunk, branches and tags directories

## ➤ Obtén la dirección del repositorio

### ➤ Accede a Repositories



### ➤ Accede a tu repositorio: <nombre de repositorio>

### ➤ Identifica la dirección

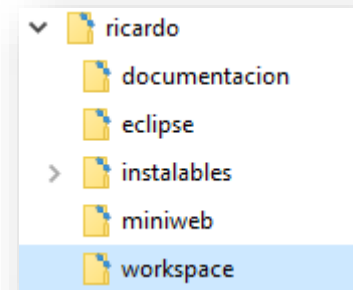
Your Subversion URL:

<https://svn.riouxsvn.com/mirepositori>

# Hacer checkout del repositorio

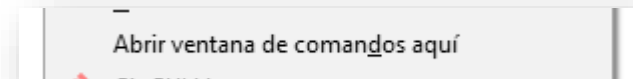


- En el explorador ve a un directorio donde quieras descargar el repositorio



- Abre una ventana de consola:

- Pulsa la tecla MAYUSCULAS y a la vez haz clic con el botón derecho en un espacio en blanco → Abrir ventana de comandos aquí



- Escribe:

```
svn checkout https://svn.riouxsvn.com/<repositorio> --username <mi_usuario>
```

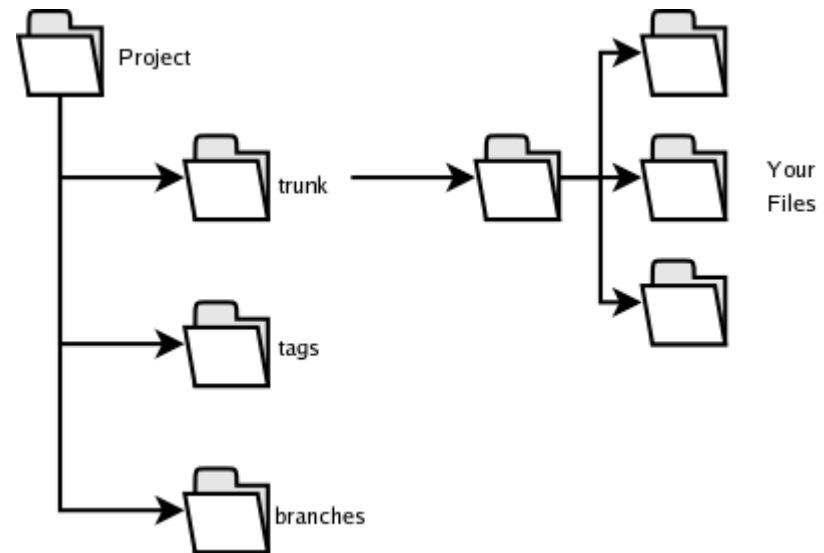
- **checkout** : Obtener una copia local de la ultima revisión
  - **--username** Solo en el checkout inicial
- Pulsa INTRO
- Aparecerá un directorio con el nombre de la aplicación que contiene los ficheros del repositorio
  - Cambia a al directorio trunk
 

```
cd <nombre_directorio>/trunk
```
  - Se lanzan las ordenes SVN sobre ese directorio

# Estructura común del repositorio



- **/trunk** Contiene la línea principal desarrollo
- **/branch** Contiene las ramas, versiones alternativas
- **/tags** Contiene los tags (etiquetas) aplicadas al código



- Si queremos hacer checkout de una línea concreta:

```
svn checkout https://svn.riouxsvn.com/myapprag/trunk --username mi_usuario
```

# Ciclo de Trabajo

- Primero siempre haremos checkout del proyecto

```
svn checkout https://svn.riouxsvn.com/<myapprag> --username <mi_usuario>
```

- A partir del primer checkout, debemos respetar el siguiente ciclo de trabajo cuando estamos trabajando con repositorios:
  1. **Actualizar** la copia de trabajo → svn update
  2. **Modificar** la copia de trabajo
  3. **Comprobar** los cambios realizados → svn update
  4. **Resolver** conflictos (si aparecen) → svn merge
  5. **Guardar** los cambios realizados → svn commit



# Update: actualizar la copia de Trabajo

- **svn update**
- Verifica los cambios que existen entre nuestra copia de trabajo y el repositorio, con respecto a la ultima revisión
- Si queremos una versión concreta: `svn update --revision 23`
- Los ficheros actualizados se mostrarán con distintos flags:

A Added  
D Deleted  
U Updated  
C Conflict  
G Merged  
E Existed

```
murdock@A-Team:~/svnos1$ svn update --revision 23
A    asistencia.txt
A    asignaturas
A    hola.cpp
A    leeme.txt
A    fichero_nano.txt
A    hola.h
A    img
Actualizado a la revisión 23.
murdock@A-Team:~/svnos1$ svn update
D    asistencia.txt
D    asignaturas
D    hola.cpp
D    leeme.txt
D    fichero_nano.txt
D    hola.h
D    img
Actualizado a la revisión 24.
```

# Añadir, borrar, copiar, mover ficheros y crear directorios

- svn **add** <fichero>: añade un fichero
- svn **delete** <fichero>: borra un fichero
- svn **copy** <fichero.old> <fichero.new>: copia un fichero
- svn **move** <fichero.old fichero.new>: mueve un fichero
- svn **mkdir** <directorio>: Crea el directorio y lo añade: mkdir + add

```
murdock@A-Team:~/svnosl$ nano
murdock@A-Team:~/svnosl$ svn add ejemplo.cpp
A      ejemplo.cpp
murdock@A-Team:~/svnosl$ nano
murdock@A-Team:~/svnosl$ svn add ejemplo.h
A      ejemplo.h
murdock@A-Team:~/svnosl$ █
```

# Comprobar el Estado y los Cambios

- **svn status**
- Muestra el estado de los ficheros
- `svn status -q` → quiet o silencioso
- `svn status -u` → updates o mostrar actualizaciones
- `svn status -v` → verbose o con toda la información

```
?      scratch.c      # file is not under version control
A      stuff/loot/bluo.h  # file is scheduled for addition
C      stuff/loot/lump.c  # file has textual conflicts from an update
D      stuff/fish.c      # file is scheduled for deletion
M      bar.c            # the content in bar.c has local modifications
```

# Ejemplo

```

murdock@A-Team:~/svnosl$ svn status
?      leeme.txt
?      fichero.txt
A      asistentes
murdock@A-Team:~/svnosl$ svn status -q
A      asistentes
murdock@A-Team:~/svnosl$ svn status -u
A          0      asistentes
?          leeme.txt
?          fichero.txt
Estado respecto a la revisión:      26
murdock@A-Team:~/svnosl$ svn status -v
          26          26 Fran.Lucena      .
?          leeme.txt
?          fichero.txt
          26          26 Fran.Lucena      ejemplo.h
A          0          ?      ?      asistentes
          26          10 Fran.Lucena      plantilla.html
          26          26 Fran.Lucena      ejemplo.cpp
          26          25 Fran.Lucena      SCS_CAs.pem
murdock@A-Team:~/svnosl$

```

# Comprobar Cambios

- Para una determinada revisión
- `svn diff -r 26 ejemplo.h`

```
murdock@A-Team:~/svnos1$ svn diff -r26 ejemplo.h
Index: ejemplo.h
=====
--- ejemplo.h      (revisión: 26)
+++ ejemplo.h      (copia de trabajo)
@@ -1 +1,2 @@
-lo que sea
+
+He eliminado la linea anterior.
```

# Deshacer Cambios

- **svn revert**
- Revierte un cambio
- Mediante la orden diff se observa un error y se corrige el cambio mediante esta orden
- Por ejemplo, cuando ejecutamos svn delete por error

# Guardar Cambios

- `svn commit -m "<texto descriptivo>"`
- Permite **subir** los cambios al repositorio

```
murdock@A-Team:~/svnos1$ svn commit -m "Mas ficheros necesarios"
Añadiendo      fichero.txt
Añadiendo      leeme.txt
Transmitiendo contenido de archivos ..
Commit de la revisión 28.
```



## Pongámoslo en práctica

- Crea un fichero hola.txt y añádelo a tu repositorio
- Comprueba el estado del fichero y luego haz commit
- Comprueba en el repositorio que el archivo está subido
- Añade un directorio a tu repositorio: “subdirectorio”
- Crea otros ficheros dentro: como.txt y estas.txt
- Añádelos y haz commit
- Elimina estas.txt del repositorio



# Resolver Conflictos

- Aparecen cuando dos miembros del equipo moifican un mismo archivo (y svn no es capaz de mezclarlos automáticamente)

```
$ svn update
U  INSTALL
G  README
Conflict discovered in 'bar.c'.
Select: (p) postpone, (df) diff-full, (e) edit,
        (h) help for more options:
```

- Aparecen los siguientes ficheros:
  - **filename.mine**: Mi Archivo
  - **filename.rOLDREV**: Version original
  - **filename.rNEWREV**: Version repositorio

# Resolver Conflictos

- Primero observaremos las diferencias (**opción df**)

```
...
Select: (p) postpone, (df) diff-full, (e) edit,
        (h)elp for more options : df
--- .svn/text-base/sandwich.txt.svn-base      Tue Dec 11 21:33:57 2007
+++ .svn/tmp/tmpfile.32.tmp                  Tue Dec 11 21:34:33 2007
@@ -1,5 @@
-Just buy a sandwich.
+<<<<<<< .mine
+Go pick up a cheesesteak.
+=====
+Bring me a taco!
+>>>>>>> .r32
...
```

- Luego editaremos (e) para dejar la parte que nos interese
- Finalmente haremos **svn resolve**, para resolver

```
$ svn resolve --accept working sandwich.txt
Resolved conflicted state of 'sandwich.txt'
$ svn commit -m "Go ahead and use my sandwich, discarding Sally's edits."
```

# Configurar un editor personalizado

- Editar el fichero: %USERPROFILE%\Subversion\config
  - [helpers]
  - editor-cmd = c:/emacs-24.3/bin/runemacs.exe

# Ver el log de operaciones

- **svn log**
- Muestra el historial de mensajes de un commit
- `svn log -r 20 -v`
- `svn log -r 10:20`
- `svn log -r 20:15`

```
murdock@A-Team:~/svnosl$ svn log -r 20 -v
-----
r20 | esloguin | 2009-12-17 21:13:56 +0100 (jue 17 de dic de 2009) | 3 lines
Rutas cambiadas:
  M /trunk/asistencia.txt

"Version antoñico"
-----
```

# Ver directorios

- **svn list -v** directorio (URL)
- Ficheros disponibles en el directorio

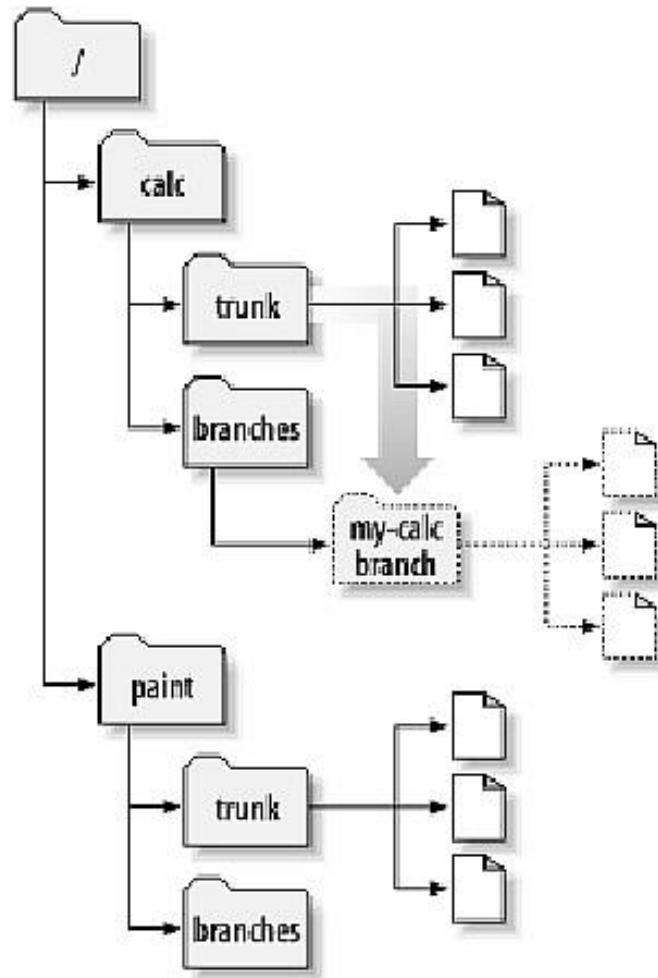
```
svn list -v http://svn.collab.net/repos/svn
20620  harry          1084 Jul 13  2006 README
23339  harry          Feb 04 01:40 branches/
21282  sally            Aug 27 09:41 developer-resources/
23198  harry            Jan 23 17:17 tags/
23351  sally            Feb 05 13:26 trunk/
```

# Trabajo con ramas

- **svn copy:** Genera una rama, copia de la raíz
- **svn merge:** Mezcla dos ramas
- **svn diff:** muestra la diferencia entre ramas
- **svn switch:** cambia a una rama

```
$ svn copy http://svn.example.com/repos/calc/trunk \  
           http://svn.example.com/repos/calc/branches/my-calc-branch \  
           -m "Creating a private branch of /calc/trunk."  
  
Committed revision 341.
```

# Operaciones con ramas



# Operaciones con tags

- Para crear tags se usa una **svn copy**

```
$ svn copy http://svn.example.com/repos/calc/trunk \
           http://svn.example.com/repos/calc/tags/release-1.0 \
           -m "Tagging the 1.0 release of the 'calc' project."

Committed revision 902.
```

```
A   svnosl/trunk
A   svnosl/trunk/plantilla.html
A   svnosl/branches
A   svnosl/branches/rama01
A   svnosl/branches/rama01/hola.cpp
A   svnosl/branches/rama01/leeme.txt
A   svnosl/branches/rama01/hola.h
A   svnosl/branches/rama01/img
A   svnosl/tags
A   svnosl/tags/miproyecto1.0
A   svnosl/tags/miproyecto1.0/hola.cpp
A   svnosl/tags/miproyecto1.0/leeme.txt
A   svnosl/tags/miproyecto1.0/hola.h
A   svnosl/tags/miproyecto1.0/img
Revisión obtenida: 24
```



# Mezcla de dos ramas: merge

- Para hacer un merge es necesario moverse al trunk y ejecutar

```
svn merge --reintegrate ../../branches/<nombre_rama>  
svn commit -m "<un mensaje>"
```

3

GIT

# GIT

- **Git** (pronunciado "guit" ) es un software de control de versiones diseñado por Linus Torvalds, pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente.
- No confundir con [GitHub](#), que es la plataforma de "hosting" de los proyectos.
- Una comunidad llena de personas que desarrollan y comparten, usando GIT.



# git



## Instalando las herramientas

# Instala las siguientes herramientas



- Descargar e instala Git
  - › <http://git-scm.com/download/win>
- Crear una cuenta en github
  - › <https://github.com/>

# Comandos Git

Comando	Descripción
git init	Initializes a new Git repository. Until you run this command inside a repository or directory, it's just a regular folder. Only after you input this does it accept further Git commands.
git config	Short for "configure," this is most useful when you're setting up Git for the first time.
git help	Forgot a command? Type this into the command line to bring up the 21 most common git commands. You can also be more specific and type "git help init" or another term to figure out how to use and configure a specific git command.
git status	Check the status of your repository. See which files are inside it, which changes still need to be committed, and which branch of the repository you're currently working on.
git add	This does not add new files to your repository. Instead, it brings new files to Git's attention. After you add files, they're included in Git's "snapshots" of the repository.

# Comandos Git (II)

Comando	Descripción
git commit	Git's most important command. After you make any sort of change, you input this in order to take a "snapshot" of the repository. Usually it goes git commit -m "Message here." The -m indicates that the following section of the command should be read as a message.
git branch	Working with multiple collaborators and want to make changes on your own? This command will let you build a new branch, or timeline of commits, of changes and file additions that are completely your own. Your title goes after the command. If you wanted a new branch called "cats," you'd type git branch cats.
git checkout	Literally allows you to "check out" a repository that you are not currently inside. This is a navigational command that lets you move to the repository you want to check. You can use this command as git checkout master to look at the master branch, or git checkout cats to look at another branch.

# Comandos Git

➤ A ejecutar en consola:

git merge	When you're done working on a branch, you can merge your changes back to the master branch, which is visible to all collaborators. git merge cats would take all the changes you made to the "cats" branch and add them to the master.
git push	If you're working on your local computer, and want your commits to be visible online on GitHub as well, you "push" the changes up to GitHub with this command.
git pull	If you're working on your local computer and want the most up-to-date version of your repository to work with, you "pull" the changes down from GitHub with this command.






## Pongámoslo en práctica - Creando un repositorio



# Creando los repositorios remoto y local

- Preséntate a Git
  - › git config –global user.name "Your Name Here"
  - › git config –global user.email [your\\_email@youremail.com](mailto:your_email@youremail.com)
- Crea un repositorio Online
  - › <https://github.com/> → 
- Abre una ventana de consola y muévete a la dirección donde quieres crear el repositorio
- Crea un repositorio Local

```
mkdir MyProject  
cd MyProject  
git init
```



# Trabajando con el repositorio

- Crea un archivo

```
copy nul Archivo.txt
```

- › Verifica que Git puede verlo:

```
git status
```

- Añádelo a Git

```
git add Archivo.txt
```

- Toma una foto del proyecto (sube el archivo al repositorio local)

```
git commit -m "Add Readme.txt"
```



# Conectando con github

- Conecta con el repositorio remoto

- Añádelo a Github

```
git remote add origin https://github.com/username/myproject.git
```

- Verifica la versión

```
git remote -v
```

- Envía los datos (haz push)

```
git push
```

- Para definir la rama principal: *git push origin master*



# Actualiza los datos

- Es necesario traer el estado actual primero
  - `git pull [origin master]`
- Modifica el archivo Archivo.txt
- Push
  - `git commit -m "Add Readme.txt"`
  - `git push`
- Verifica en Github



## Pongámoslo en práctica

- Añade mas archivos a tu repositorio y sincronízalo con tu repositorio remoto



## Creando nuestro repositorio BananaTube

- Decide en equipo qué sistema de gestión de versiones se usará
- Genera el repositorio y añade a los miembros del equipo



 **netmind**

**WeKnowIT**

## Barcelona

C. Almogàvers, 123  
08018 Barcelona  
Tel. 93 304.17.20  
Fax. 93 304.17.22

## Madrid

Plaza Carlos Trías Bertrán, 7  
28020 Madrid  
Tel. 91 442.77.03  
Fax. 91 442.77.07

[www.netmind.es](http://www.netmind.es)



MINISTERIO  
DE ENERGÍA, TURISMO  
Y AGENDA DIGITAL

**red.es**



**UNIÓN EUROPEA**

Fondo Social Europeo  
*"El FSE invierte en tu futuro"*