

## Práctica 2. Implementación de un driver básico para la transmisión de datos a través de la UART.

### 1. Objetivo

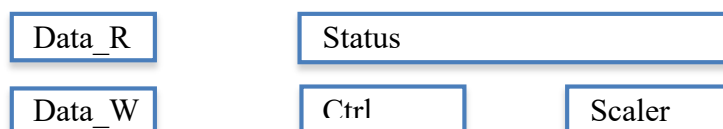
El objetivo de la siguiente práctica es el de implementar el software de control (*driver*) de un dispositivo de entrada/salida. El dispositivo elegido forma parte del conjunto de interfaces estándar disponibles en la configuración del simulador TSIM2 del procesador LEON2 que se está utilizando en este laboratorio. Concretamente se trata de la **UART-A** (Universal Asynchronous Receiver-Transmitter-A), que permite la **comunicación serie asíncrona** con otros dispositivos, tanto en modo entrada, como salida.

La implementación de un *driver* para un dispositivo de estas características requiere conocer el comportamiento de su **controlador** con un cierto nivel de detalle, ya que cada uno de los registros del controlador cuenta con una función específica, que debe ser analizada. Por otra parte, un mismo dispositivo puede ser configurado para trabajar de manera distinta y eso determinará el enfoque de diseño del *driver*. Algunas de las preguntas básicas que condicionan este diseño son:

- ¿debe accederse al dispositivo en **modo bloque** o en **modo carácter**?
- ¿debe el *driver* poder utilizarse por más de un hilo/proceso?
- ¿Las operaciones sobre el dispositivo implican una **espera activa** o se emplean buffers controlados por interrupciones para evitar ese bloqueo?

En esta práctica se va a comenzar con la implementación de un sencillo *driver* en modo carácter para la UART-A del LEON3. El *driver* cuenta con una función que permite simplemente transmitir un carácter a través de este dispositivo con una **espera activa**, de manera que el hilo/proceso permanecerá en la función hasta que el carácter se haya podido insertar en la cola de transmisión. La práctica además plantea un ejercicio relacionado con la implementación de rutinas auxiliares empleadas en funciones de formateo y redirección de la información hacia un dispositivo de salida (tipo *printf*).

El controlador de la UART-A está formado internamente por 5 registros. Dos registros de datos, **Data\_R** y **Data\_W**, que permiten, respectivamente, leer el dato recibido, y escribir el dato que se desea transmitir. Un registro denominado **Status** que proporciona información sobre el estado actual del dispositivo. El registro **Ctrl**, que permite controlar la UART y, finalmente, el registro **Scaler**, que permite configurar la velocidad a la que trabajará la UART. La siguiente figura muestra un esquema de los registros internos del controlador.

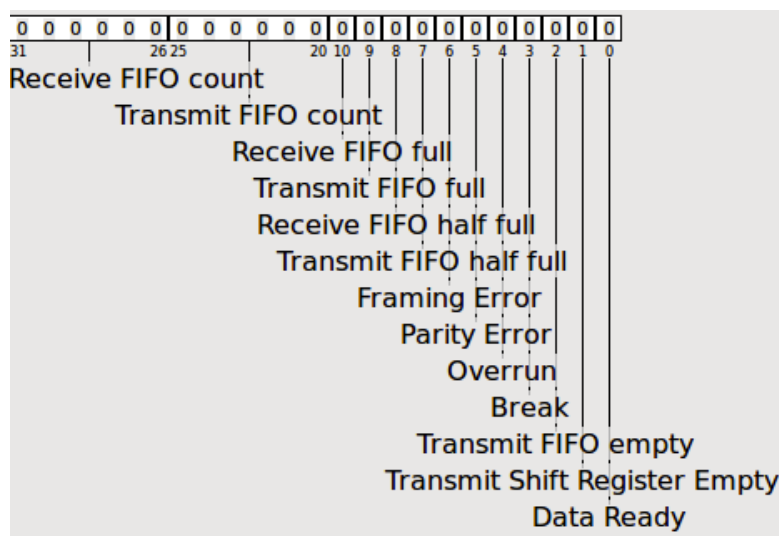


La configuración de interfaces que presenta el simulador TSIM2 ubica los registros de datos, control y estado del controlador de este dispositivo en el rango de direcciones de memoria 0x80000100-0x800010C.

Los registros **Data\_R** y **Data\_W** son accesibles a través de la misma dirección 0x80000100. El controlador internamente selecciona **Data\_R** en las operaciones de lectura y **Data\_W** en las operaciones de escritura. El que dos registros compartan la misma dirección no es extraño, ya que los controladores se diseñan intentando reducir el número de direcciones utilizadas. Sin embargo, tiene una implicación para la depuración, **ya que no será posible inspeccionar el valor que se escribe en Data\_W.**

El registro **Status** está localizado en la dirección 0x80000104, mientras que **Ctrl** y **Scaler** se ubican respectivamente, en 0x80000108 y 0x8000010C.

El control de la UART que se va a realizar en esta práctica requiere conocer primeramente el comportamiento del bit **Transmit FIFO Full (TFF)** del registro **Status** (ver figura siguiente). Este bit nos indica si la cola del transmisor está llena (valor de **TFF** = 1). El algoritmo a emplear para transmitir va consistir simplemente en comprobar el valor del bit **TFF** antes de escribir en el registro **Data\_W**. En caso de que valga 1 (cola llena), se esperará en un bucle durante un cierto tiempo a que el bit cambie de valor. En caso de que después de ese tiempo **TFF** permanezca a 1, se considerará que hay un error y que no es posible transmitir el dato. Además, para poder asegurarnos que todos los datos que hay en la FIFO se han transmitido, se va a monitorizar el bit **Transmit FIFO Empty (TFE)** que toma valor 1 cuando la FIFO de transmisión está vacía.



Descripción del registro Status de la UART-A

## 2. Creación de proyecto para LEON3

Crear un nuevo proyecto (utilizad la opción *Empty Project*) denominado **prac2** cuyo ejecutable sea para la plataforma Sparc Bare C. En ese proyecto crear dos subdirectorios **include** y **src**. En el directorio **include** crear el archivo **leon3\_types.h** con la siguiente redefinición de los tipos básicos para la arquitectura del procesador LEON3.

```

#ifndef LEON3_TYPES_H
#define LEON3_TYPES_H

typedef unsigned char      byte_t;
typedef unsigned short int word16_t;
typedef unsigned int       word32_t;
typedef unsigned long int  word64_t;

typedef signed char        int8_t;
typedef signed short int   int16_t;
typedef signed int          int32_t;
typedef signed long int    int64_t;

typedef unsigned char      uint8_t;
typedef unsigned short int uint16_t;
typedef unsigned int        uint32_t;
typedef unsigned long int  uint64_t;

#endif

```

### 3. Implementación de funciones básicas para la transmisión a través de la UART-A del LEON3.

Crear en el directorio *include* el archivo *leon3\_uart.h* con el siguiente contenido:

```

#ifndef LEON3_UART_H_
#define LEON3_UART_H_

#include "leon3_types.h"

int8_t leon3_putchar(char c);
int8_t leon3_uart_tx_fifo_is_empty();

#endif /* LEON3_UART_H_ */

```

Crear en el directorio *src* el archivo *leon3\_uart.c* con el siguiente contenido en el que se declara la estructura *UART\_regs* que permite acceder a los registros de la UART de LEON3.

```

#include "leon3_uart.h"

//Estructura de datos que permite acceder a los registros de la
//UART de LEON3

struct UART_regs
{
    /** \brief UART Data Register */
    volatile uint32_t Data;      /* 0x80000100 */
    /** \brief UART Status Register */
    volatile uint32_t Status;    /* 0x80000104 */
    /** \brief UART Control Register */
    volatile uint32_t Ctrl;      /* 0x80000108 */
    /** \brief UART Scaler Register */
    volatile uint32_t Scaler;    /* 0x8000010C */
};

```

En la estructura se utiliza el atributo **volatile** que fuerza a que el compilador no optimice el código de ejecución condicional basado en el valor que puedan tomar esos campos. Esto permite que se puedan modificar vía hardware, y el software compruebe siempre el valor que tienen, sin considerar si acaba de asignarse a un valor constante. En los siguientes dos ejemplos se ve la diferencia de declarar una variable como **volatile** o no.

```
uint8_t aux;
uint8_t i;

aux=0;

if(aux>0){

    // El optimizador no incluye este código, ni el código de
    // comprobación de aux, puesto que acaba de asignarse a 0

}else{

}

}
```

```
volatile uint8_t aux;
uint8_t i;

aux=0;

if(aux>0){

    // El optimizador SI incluye este código, y el de la
    // comprobación de aux, puesto que aux es volatile

}else{

}

}
```

Completar en el archivo *leon3\_uart.c* las macros `LEON3_UART_TFF` y `LEON3_UART_TFE` que van a actuar, respectivamente, de máscaras del bit **Transmit FIFO Full** y **Transmit FIFO Empty** del registro de Status. Recordad que para que una caracter pueda insertarse en la cola de transmisión es necesario que el bit que se hace visible al aplicar la máscara `LEON3_UART_TFF` **NO** esté a 1,

```
(Completar MACROS ↓)

//! LEON3 UART A Transmit FIFO is FULL
#define LEON3_UART_TFF

//! LEON3 UART A Transmit FIFO is EMPTY
#define LEON3_UART_TFE
```

Añadir al archivo *leon3\_uart.c* la declaración del puntero `pLEON3_UART_REGS`. Completar el código para que este puntero apunte a la dirección `0x80000100`, que es la dirección donde se encuentra ubicados los registros de la UART-A

```

//COMPLETAR ↓
struct UART_regs * const pLEON3_UART_REGS=

```

Completar la macro `leon3_UART_TF_IS_FULL` y las funciones `leon3_putchar` y `leon3_uart_tx_fifo_is_empty` (ubicadas todas ellas en *leon3\_uart.c*) de acuerdo a las siguientes definiciones.

- `leon3_UART_TF_IS_FULL` es una *function-like macro* que debe devolver falso sólo si el bit TFF del registro de **Status** vale 0, lo que indica que la cola fifo de transmisión no está llena, y se puede insertar un carácter en la cola de transmisión.
- `leon3_putchar` es una función que, tras comprobar que la cola fifo de transmisión no está llena, escribe el carácter `c` pasado por parámetro en el registro **Data** la UART. En el caso de que después de un intervalo de tiempo (controlado por la variable `write_timeout`) la fifo siga llena, el carácter no se escribirá y la función retornará un valor distinto de 0.
- `leon3_uart_tx_fifo_is_empty` función debe devolver cierto si el bit TFE del registro de **Status** vale 1, lo que indica que la FIFO de transmisión está vacía.

```

//COMPLETAR MACRO ↓
#define leon3_UART_TF_IS_FULL() (    )

int8_t leon3_putchar(char c)
{
    uint32_t write_timeout=0;

    while(leon3_UART_TF_IS_FULL() && (write_timeout < 0xAAAAAA))
    {
        write_timeout++;
    } //Espera mientras la cola de transmisión esté llena

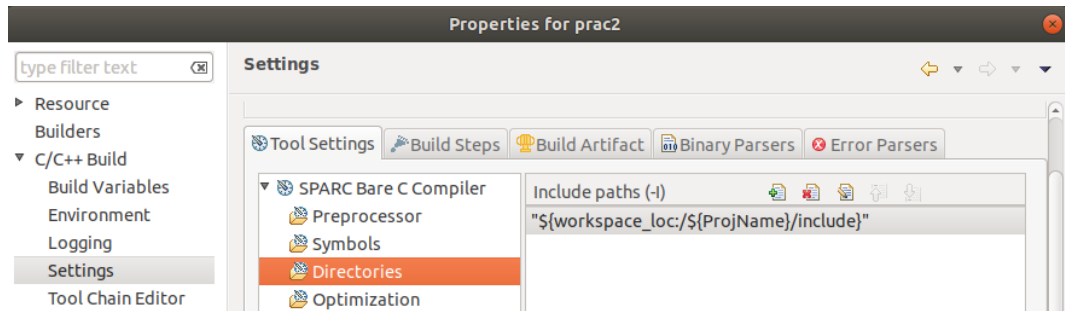
    if(write_timeout < 0xAAAAAA){
        //COMPLETAR. Escribir el carácter en el registro Data
    }
    return (write_timeout == 0xAAAAAA);
}

int8_t leon3_uart_tx_fifo_is_empty(){
    //COMPLETAR ↓
}

```

Antes de compilar, configurar el *path* de búsqueda de archivos de cabecera del proyecto para que busque en el directorio *include* del proyecto. Para ello seleccionar el proyecto **prac2**, y con el botón derecho abrir el panel de configuración *Properties*, y siguiendo la secuencia de menús *C/C++|Build-Settings|Sparc Bare C Compiler|Directories*, añadir en *Include paths* la ruta del directorio **include** del proyecto, utilizando una ruta relativa

al *workspace*, tal como aparece en la siguiente imagen. Usar una ruta relativa al *workspace* os evitará problemas al llevar vuestro proyecto a otros computadores.



#### 4. Primera versión del programa principal.

Crear el archivo `main.c` con el siguiente código de prueba del driver básico y comprobar que la salida es

**p2**

```
#include "leon3_uart.h"

int main()
{
    leon3_putchar('p');
    leon3_putchar('2');
    leon3_putchar('\n');
    while(!leon3_uart_tx_fifo_is_empty())
        ;

    return 0;
}
```

**Cuestión:** ¿Qué crees que saldría por pantalla si utilizas el siguiente programa principal?  
¿Y si la asignación inicial fuera `char aux=0;` ?

```
int main(){
    int i=0;
    char aux='0';
    for (i=0; i<10; i++){
        leon3_putchar(aux); leon3_putchar('\n');
        aux++;
    }
    while(!leon3_uart_tx_fifo_is_empty())
        ;

    return 0;
}
```

## 5. Rutinas de formateo y envío a través de la UART.

Haced una copia del proyecto `prac2` que se denomine `prac2b`. Podéis hacerlo seleccionando el proyecto, y con el botón derecho eligiendo la opción **Copy**, y **después haciendo Paste sobre panel de proyectos**. Añadir al nuevo proyecto los archivos `leon3_bprint.h` (en el directorio *include*) y `leon3_bprint.c` (en el directorio *src*). Implementar en estos archivos y **SIN USAR `sprintf` (ni ninguna otra función de librería)** las siguientes rutinas auxiliares de formateo de información y redirección hacia la UART. Se da un ejemplo de implementación de la función `leon3_print_uint8`

```
//transmite a través del puerto serie la cadena de caracteres
//pasada por parámetro.

int8_t leon3_print_string(char* str); //ver **

//transmite a través del puerto serie el código ASCII de cada
//uno los dígitos del entero de 8 bits que se pasa como
//parámetro. No transmite los ceros a la izquierda

int8_t leon3_print_uint8(uint8_t i); // ver ***

//transmite a través del puerto serie el código ASCII de cada
//uno los dígitos del entero de 16 bits que se pasa como
//parámetro. No transmite los ceros a la izquierda

int8_t leon3_print_uint16(uint16_t i); // ver ****
```

**\*\*** (Tened en cuenta para la implementación de `leon3_print_string` que el último caracter de una cadena es el `'\0'`)

**\*\*\*** (Tened en cuenta para la implementación de `leon3_print_uint8`, que un entero de 8 bits tiene un valor máximo de 255, por lo que para su transformación bastará con calcular 3 caracteres: el de las centenas, el de las decenas y el de las unidades)

**\*\*\*\*** (Tened en cuenta para la implementación de `leon3_print_uint16`, que un entero de 16 bits tiene un valor máximo de 65535, por lo que para su transformación bastará con calcular los 5 caracteres correspondientes a los 5 dígitos)

Utilizad internamente en la implementación de ambas funciones la función `leon3_uart_tx_fifo_is_empty()` para asegurar que todos los caracteres (de la cadena o el entero) se han transmitido antes de retornar de la función.

Ejemplo de implementación de `leon3_print_uint8`

```
int8_t leon3_print_uint8(uint8_t i){

    int8_t error=0;
    int8_t first_digit=1; //Evita ceros a la izquierda
    uint8_t aux=100; //Número auxiliar para extraer dígitos
```

```

        //Si el número es 0 transmitir un único dígito
        if(i==0){
            error=leon3_putchar('0');
        }else{
            //Si no es 0 transmitir dígito a dígito
            while(aux&&(!error)){
                uint8_t digit;

                digit=i/aux;//extrae el dígito más significativo

                i-=aux*digit; //número con el resto de dígitos

                aux=aux/10; //centenas->decenas->unidades

                //Si ya se ha transmitido el primer dígito
                //o el dígito extraído es !=0,
                //transmitir el caracter del dígito
                if((0==first_digit)||digit){
                    error=leon3_putchar('0'+digit);
                    first_digit=0; //ya se impri
                }
            }
        }

        //espera a que la fifo de transmisión se vacíe
        while(!leon3_uart_tx_fifo_is_empty())
            ;
        return error;
    }
}

```

Comprobad que su implementación es correcta mediante el siguiente programa principal que debe dar como salida (ten en cuenta que no deben aparecer ceros a la izquierda en los números):

```

cadena
3
143
1080

```

```

#include "leon3_uart.h"
#include "leon3_bprint.h"

int main()
{
    char * pchar="cadena\n";
    leon3_print_string(pchar);
    leon3_print_uint8(3); leon3_putchar('\n');
    leon3_print_uint8(143); leon3_putchar('\n');
    leon3_print_uint16(1080); leon3_putchar('\n');
    return 0;
}

```

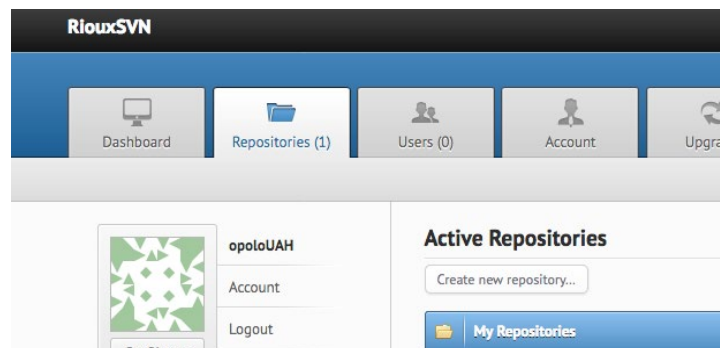


## 6. Control de configuración de las Prácticas

El control de configuración de proyectos permite a los ingenieros tener en todo momento la trazabilidad de los cambios que se han producido en el código fuente de sus proyectos, así como asociar etiquetas a la situación del proyecto en fases concretas del desarrollo. Para nuestra asignatura vamos a emplear como herramienta de control de configuración un cliente de Subversion (SVN) integrado como plugin en el entorno Eclipse (También podéis usar git si lo deseáis, sólo tenéis que compartir el proyecto con los profesores). Además, siguiendo el guion de la práctica aprenderéis a crear un repositorio SVN que compartiréis con vuestros profesores.

### Crear un repositorio SVN en riouxsvn.com.

Entrar en la web <https://riouxsvn.com> y crear un usuario para acceder al “hosting” de SVN libre. Ir a la pestaña *Repositories* tal como aparece en la siguiente figura, y en la sección *Active Repositories* crear un nuevo repositorio utilizando el botón *Create new repository*.



Definid el **Repository Title** con vuestro nombre y apellidos, más el prefijo IC-SE, mientras que al **Repository Name** le asignáis el nombre `ic_se_DNI`, siendo DNI el número de vuestro documento nacional de identidad, tal como aparece en la siguiente figura:

Activad en la lista **Creation Options**, la opción que crea los directorios trunk, branches and tags, tal como aparece en la siguiente figura.

**Create a New Repository (STEP 2 of 3)**

**Data Import**

Import Options

☐ Create the repository from scratch

Creation Options ([More info](#))

☒ Create trunk, branches and tags directories

[Cancel](#) [Next step](#)

En el paso tres, activar la opción que permite recibir notificaciones cuando haya un cambio en el repositorio.

**Create a New Repository (STEP 3 of 3)**

**Confirmation**

Please review the information about your repository.

Repository Title: IC-SE Nombre Apellidos

Repository Name: ic\_se\_dni

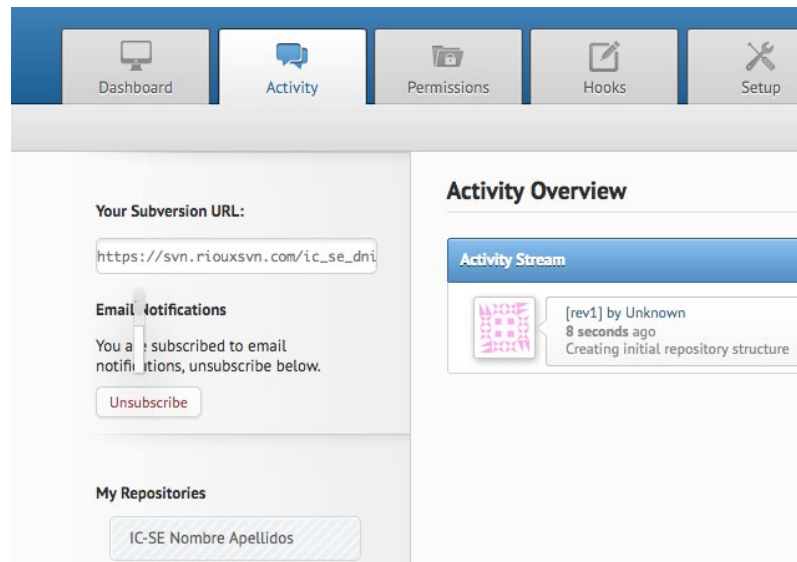
**Email notifications**

You can be notified by email each time a change is committed on that repository; the email will contain the details of the commit. This is useful if several users are working on the repository and you want to track the progress.

☒ Yes, subscribe to email notifications for this repository

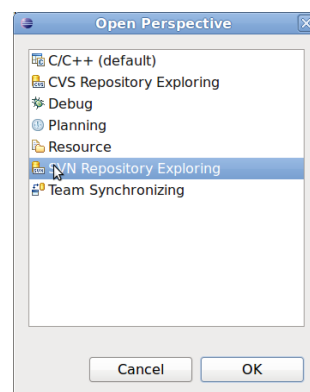
[Cancel](#) [Confirm creation](#)

Por último, en la sección *Checkout your repository* tomar nota de la URL de vuestro repositorio que aparece en *Your Subversion URL*.

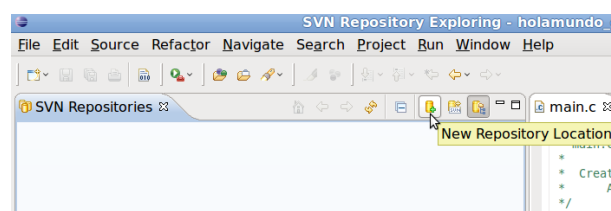


## Cómo subir un proyecto al repositorio creado.

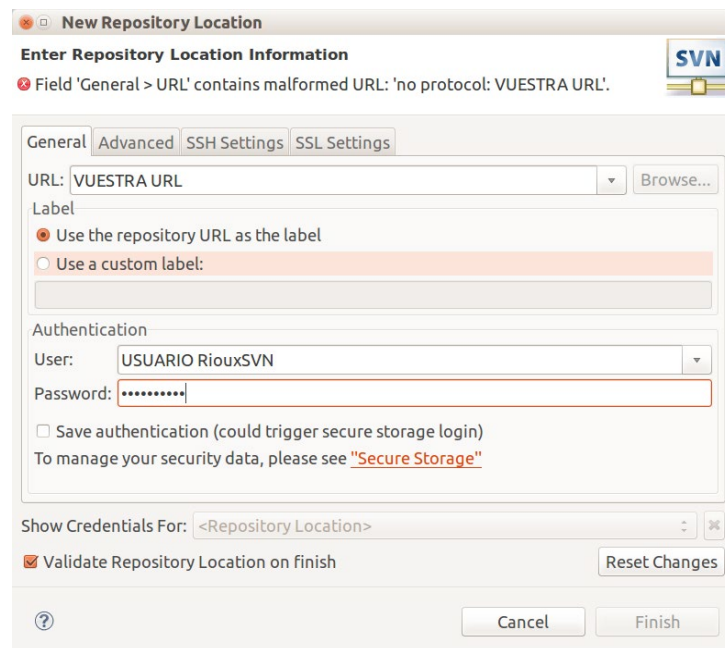
Abrir el entorno Eclipse, y dentro de él utilizar el menú *Window/Open Perspective*. Seleccionar en él la perspectiva *SVN Repository Exploring* como en la siguiente figura. Esta perspectiva os permitirá acceder al cliente SVN.



Añadir una nueva localización de repositorio (*New Repository Location*) tal como aparece en la siguiente figura.



Completar la información sobre la localización de repositorio, empleando como URL la de vuestro proyecto (obtenida en la página de Rioux en el menú *Checkout your repository*), como *User* vuestro usuario de Rioux y como *Password* la clave de vuestro usuario.

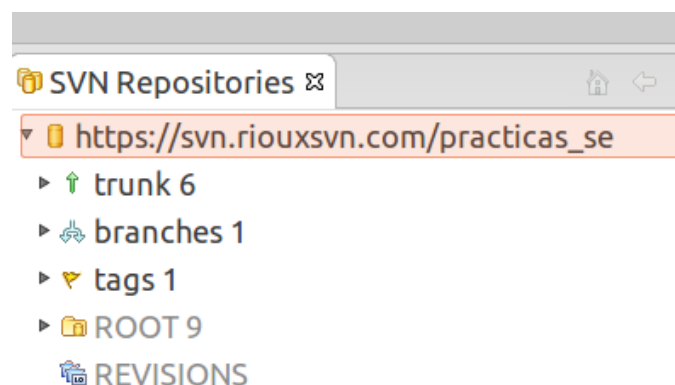


### ***Organización del repositorio SVN.***

Un repositorio suele estar organizado utilizando tres carpetas:

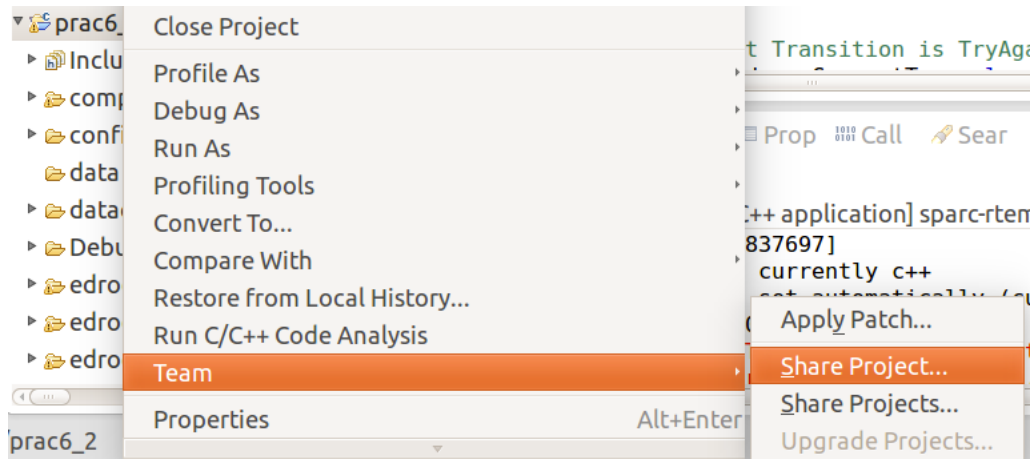
- trunk: donde se ubica la versión actual del proyecto en la que se está trabajando
- branches: donde distintos programadores pueden crear sus propias alternativas de implementación.
- tags: donde almacenar las distintas versiones de un proyecto. Una versión permite “congelar” el estado de todos los archivos fuente en un momento concreto. Cada versión de ellas puede recuperarse de forma independiente, se pueden comparar con otras, se puede restaurar como versión trunk, etc.

Vuestros repositorios ya han sido creados por Rioux con esta organización. Podéis verlo en la siguiente pantalla

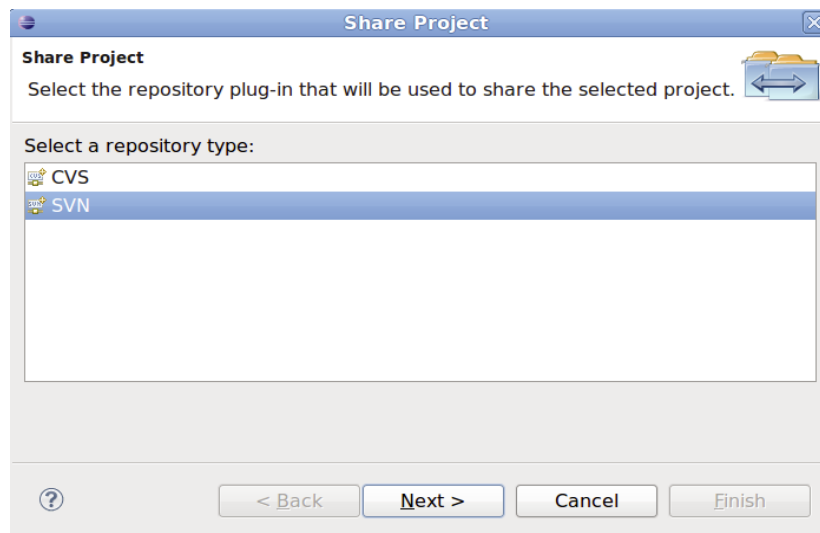


## ***Tarea a realizar. Configurar un proyecto en el repositorio SVN.***

Desde la vista *C++*, y una vez que un proyecto funcione correctamente (vamos a utilizar como ejemplo la práctica 2, que haréis a continuación), utilizar el menú contextual *Team->Share Project* sobre la carpeta del proyecto para poder configurarlo dentro del repositorio.



Seleccionar como *SVN* el tipo de repositorio a utilizar.



Seleccionar que se va a usar un repositorio ya existente.

**Share Project with SVN repository**

Select an existing repository location or create a new location.



This wizard will help you to share your files with the SVN repository for the first time. Your project will automatically be imported into the SVN repository.

☐ Create a new repository location

☒ Use existing repository location:

Label	URL
https://svn.riouxsvn.com/	https://svn.riouxsvn.com/practicas_se

Hacer **doble click** sobre la URL y elegir la URL de vuestro repositorio

En la siguiente ventana, seleccionar **Advanced Mode** con las opciones tal como aparecen a continuación, comprobando que la practica se añade en la URL `https://svn.riouxsvn.com/.../trunk/prac2` de forma que se ubique definitivamente el proyecto dentro del trunk del repositorio.

**Share Project Wizard**

**Specify the project(s) location**  
Specify the project(s) location in the SVN repository.

☐ Simple Mode:  
URL:

☒ **Advanced Mode:**

Name on Repository

☒ Use project name

☐ Use empty name

☐ Use specified name:

Project Repository Layout


☒ Use Repository Location layout

☐ Use single project layout

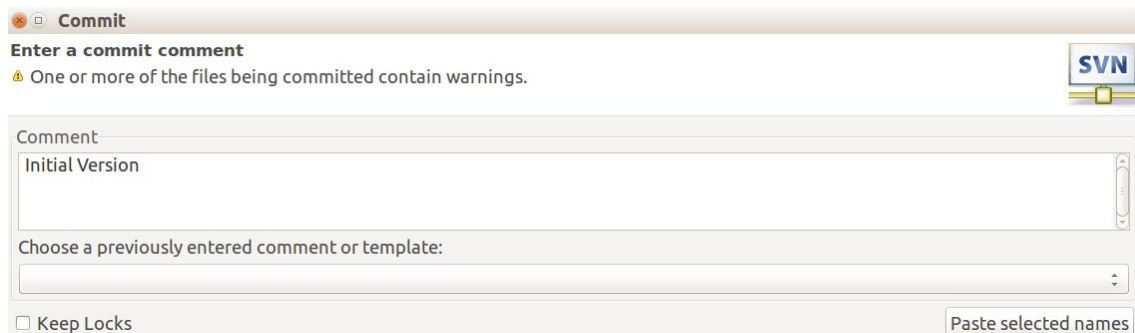
☐ Use multiple projects layout with the specified root name:

☒ Use Subversion recommended layout ('trunk', 'branches' and 'tags')

Project files location on the repository will be different depending on the selected layout type. You can see future files location below:



Para terminar, utilizar el botón *Finish* que permite subir el proyecto al repositorio. La ventana que surge permite poner un comentario a la versión. Poner como comentario “Initial Version” y pulsar OK.



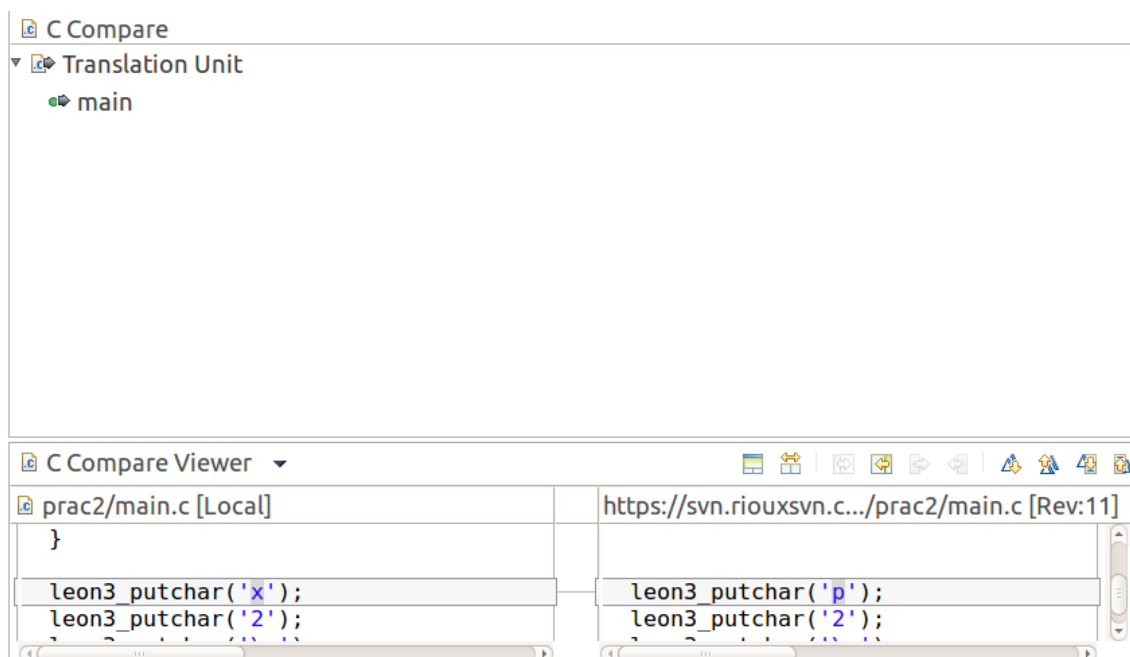
Una vez que se haya configurado el proyecto, cualquier cambio será detectado por el plugin SVN de eclipse.

En la siguiente ventana se ha cambiado la línea original del archivo main.c con la sentencia `leon3_putchar('p');` por otra `leon3_putchar('x');`

Comprobar como el símbolo > aparece en el archivo main.c para señalar las modificaciones.

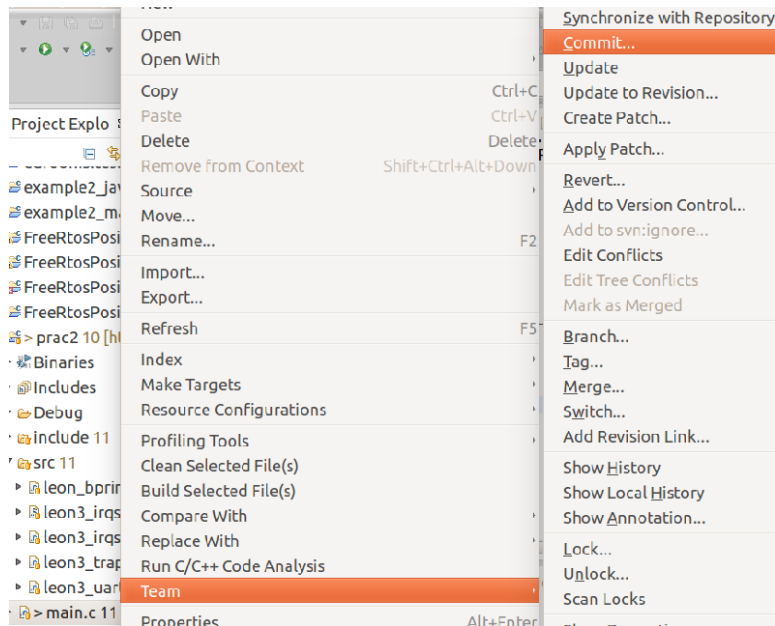
Seleccionando el archivo se puede utilizar el menú contextual (botón derecho) *Compare With->Latest from Repository* para obtener los cambios con la ultima versión configurada.

La siguiente ventana muestra los cambios.

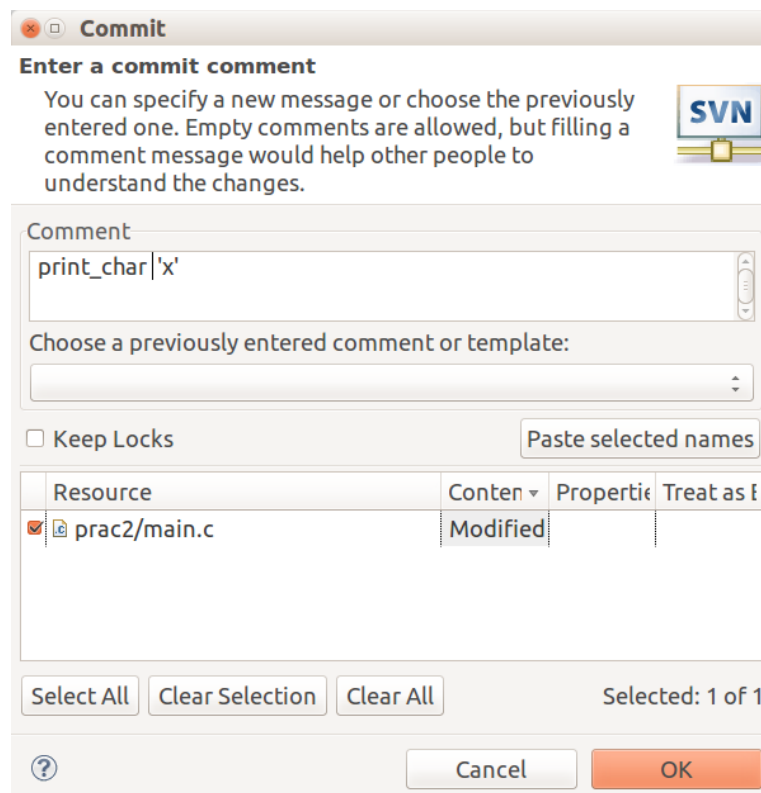


Con esta herramienta es posible comparar directorios o proyectos completos, de forma que los archivos que tengan diferencias aparecerán marcados y se podrá navegar por cada uno de ellos para comprobar las diferencias.

Finalmente, usando el menú *Team->Commit* se puede subir y configurar la nueva versión del archivo en el repositorio.



Añadiendo el comentario que describe la modificación.



Otros comandos SVN útiles del menú Team son:

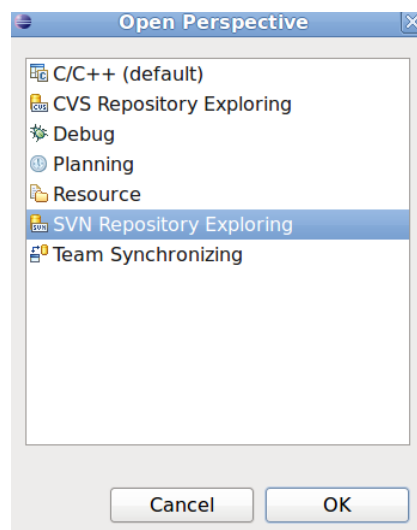
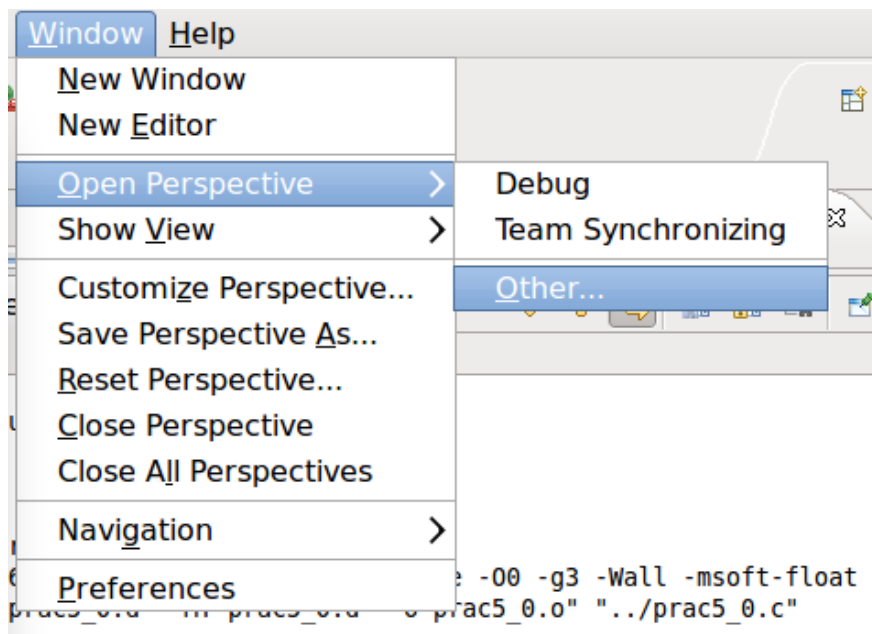
- Revert: Deshace los cambios hechos desde la última versión configurada)
- Update: Se actualiza el ítem local seleccionado (archivos, directorios o el proyecto) con la última versión del repositorio.



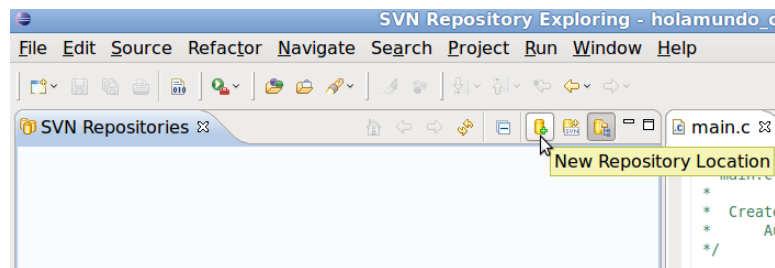
- Synchronize with Repository: Se muestra en una ventana los cambios en los items locales seleccionados con respecto a sus versiones configuradas.
- Branch: Crea un branch a partir de la versión en la que se está trabajando.
- Tag: Crea un tag a partir de la versión en la que se está trabajando.
- Switch: Ponerse a trabajar con otra versión de un Branch o un Tag.
- Disconnect: Dejar de estar vinculado al repositorio.
- Merge: Converger la versión de trabajo con la actual del repositorio.

### ***Check Out de un proyecto.***

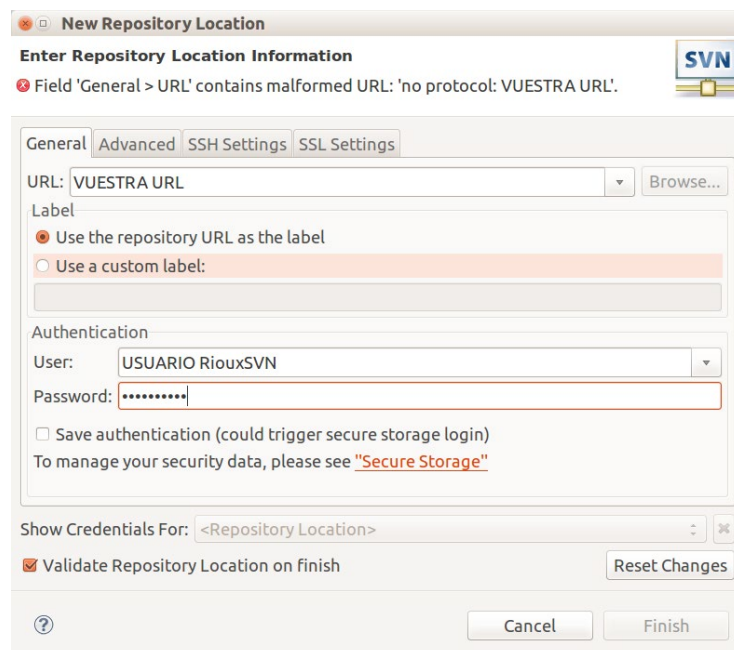
Hacer un check out de un proyecto permite bajarse a una máquina el proyecto completo configurado para poder trabajar con él en una máquina cualquiera. Esto deberá hacerse cuando cambies de máquina, y os permitirá trabajar en cualquier puesto del laboratorio con la última versión que hayáis subido al repositorio. Para hacer un *Check Out* es necesario ir al menú *Window->Open Perspective* que permite abrir la perspectiva *SVN Repository Exploring* que permite acceder a los repositorios SVN.



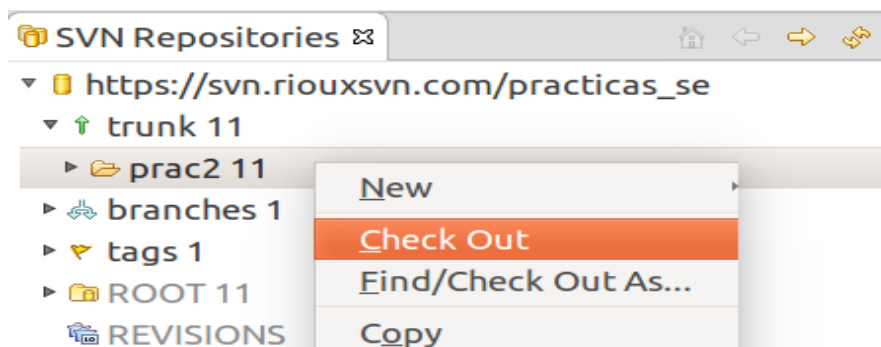
Una vez en esa perspectiva, y suponiendo que se está en una máquina distinta, se añade como nueva localización de repositorio (*New Repository Location*), utilizando el botón correspondiente tal como aparece en la siguiente figura



y se completar la información sobre la localización de repositorio, empleando de nuevo como URL la de vuestro proyecto, como *User* vuestro usuario de en *User* y como *Password* la clave obtenida anteriormente.



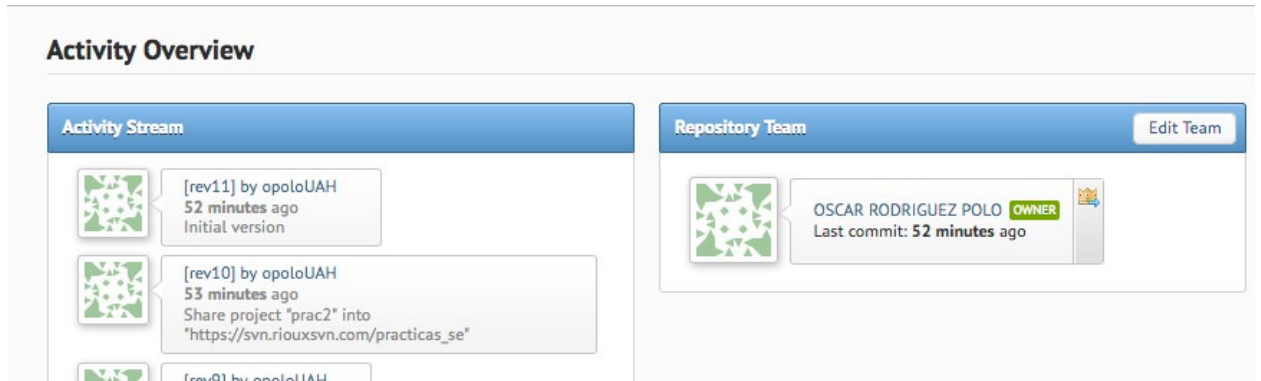
la exploración del directorio *trunk* del proyecto nos permite hacer un Check Out.



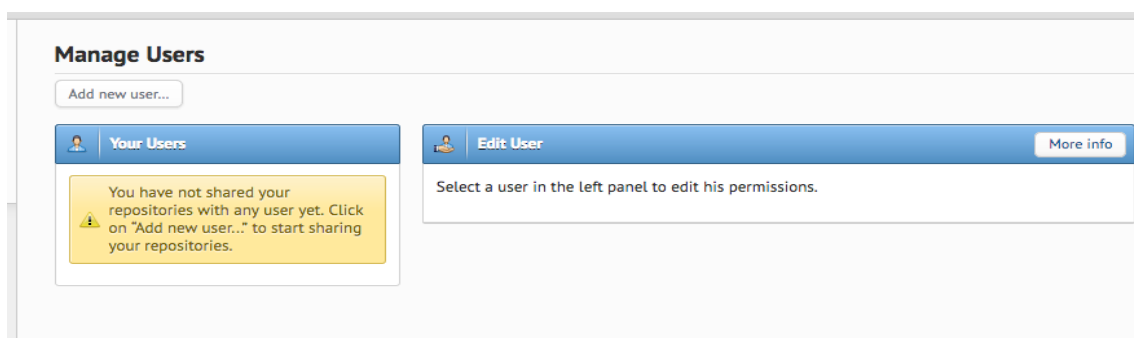
Una vez hecho el Check Out, en la perspectiva C++ se dispone del proyecto listo para trabajar con él.

## Tarea a realizar. Compartir el proyecto con el profesor.

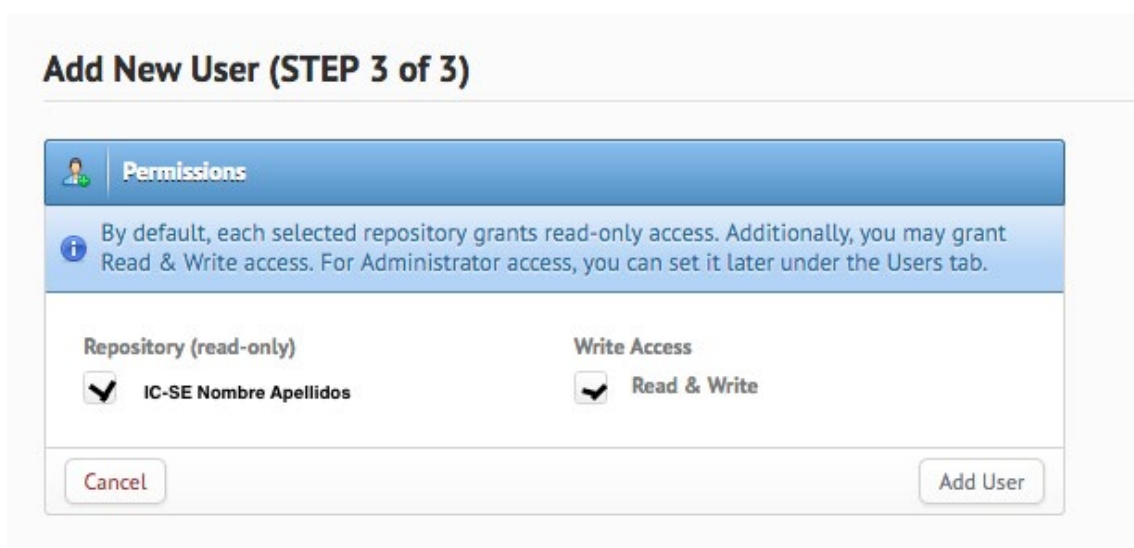
RiouxSVN permite compartir el proyecto. Haciendo *clic* sobre el proyecto se entra en **Activity**, donde se puede compartir el repositorio con otros usuarios mediante el botón **Edit Team**.



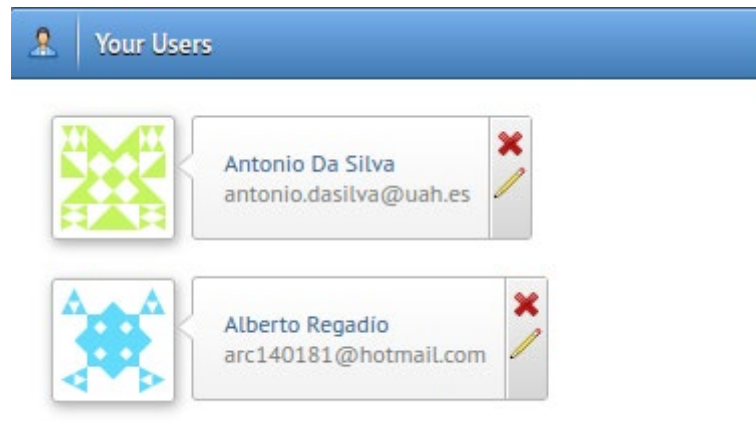
Utilizad el botón **Add New User** para compartir vuestro proyecto con el usuario **dasiUAH**. De esa forma será posible que el profesor de laboratorio compruebe que la práctica ha sido subida al repositorio.



En los menús de **Add New User** activar el Read & Write Access, como aparece en la figura siguiente, para que el profesor pueda hacer comentarios



Utilizad de nuevo el menú **Add New User** para añadir al otro profesor de la asignatura **arc140181** de forma que la configuración de usuarios con los que compartís vuestros proyectos. La configuración final de los usuarios con los que compartís los proyectos será la siguiente:



Una vez compartidos los proyectos prac2 y prac2b, cualquier comentario que el profesor haga en vuestros proyectos se os notificará vía correo electrónico, y podréis analizar, **desde eclipse**, el detalle de los comentarios del profesor con la opción *Compare With->Latest from Repository*, tal como se ha explicado anteriormente.