Version Control with Subversion and TortoiseSVN

Goal of this Lab

The goal of this lab is to become familiar with SVN (Subversion) on Windows with `TortoiseSVN`. The principle of this tool is to facilitate source code development (avoid executable or binary files!) in collaborative manner, in a synchronous or asynchronous way with a graphical interface. Other version control exist (Mercurial (http://mercurial.selenic.com/ – distributed repository); a comparison can be obtained there (http://better-scm.berlios.de/comparison/comparison.html).

TortoiseSVN installation (should already be done on your PC)

1. Download `TortoiseSVN` (http://tortoisesvn.net/downloads) and install it. Think to reboot your computer. Right click on a directory in a windows file administrator: new commands should appear (see opposite).
2. You can install other languages (http://tortoisesvn.net/translator_credits). Then, you should modify `TortoiseSVN` preferences.

Creating a repository

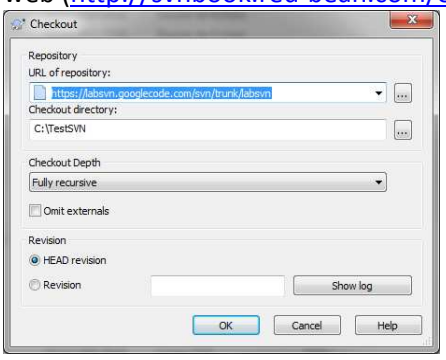
3. Create a directory on your computer/workspace named for instance `c:\repoSVN`. Right click on this directory and say to `TortoiseSVN` that this directory is a repository with dropdown menu of `TortoiseSVN > Create repository here…` (http://svnbook.red-bean.com/en/1.5/svn-book.html#svn.basic.repository). What do you see in this directory? Read the file named `README.txt`. This directory must be considered as a repository, by this way, your machine is considered as a server and should be reachable all the time on the net for people who want to work with this repository.
4. As your machine is not really a server and could be unreachable, you must use a server. I suggest you to work with GoogleCode (http://code.google.com/hosting/). Forget what you did in question 3 by deleting this directory and now, create your own repository on Googlecode. Other free forges are available as for instance Sourceforge (http://sourceforge.net/).
5. When you are the manager of a project, first thing to do is to *import* last version of your project in the repository. To do so, create a directory on your workspace (`c:\myRepo`). Copy some files in this directory (`c:\myRepo`). Then, use `TortoiseSVN` dropdown menu and choose `Import`. What happens? …on your workspace? …on your repository?

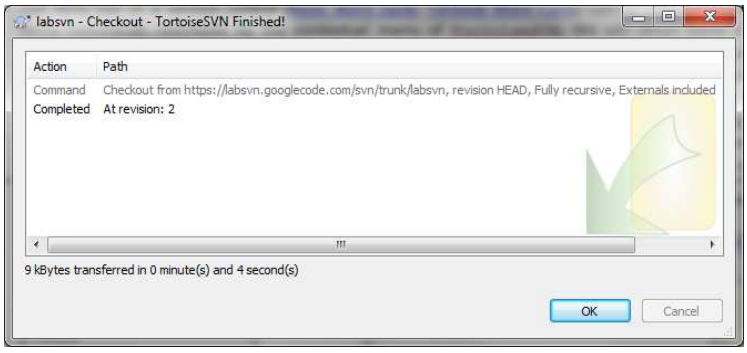
Using a repository

Now, consider that somebody has created a repository and you want to know how using it. First thing to do is to know the address of this repository. So until the end of this lab, you will use repository on Googlecode named LabSVN which address is http://code.google.com/p/labsvn/. To be associated to this project, that is to say to be

allowed to modify repository, you must give to your teacher your Google email to be added as a member of this project.

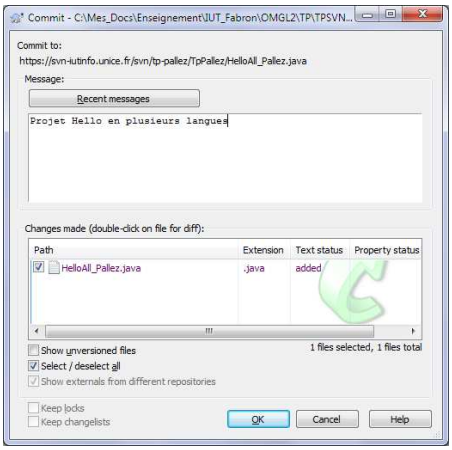
6. The daily use of a version control (Work Cycle: http://svnbook.red-bean.com/en/1.5/svn-book.html#svn.tour.cycle) can be summarized by only some commands reachable by the `TortoiseSVN` dropdown menu. During this entire lab, we will detail some of those commands. First, create a new directory called `Working Copy` or `WC` in Subversion's vocabulary on your personal workspace (for instance `c:\TestSVN`). On this directory, make a reference on an existing repository using `checkout` commands: To do this, use below information (on the left). To access to this repository, you must give your Google account (blabla@gmail.com) and the password given on your GoogleCode Settings page. If it is correctly working, you will obtain the window as shown below (right). Normally, the aspect of your working copy has changed like this: TestSVN. It indicates that you have on your working copy the last version of the project you had referenced. In fact, there are several ways to reference a project on the web (http://svnbook.red-bean.com/en/1.5/svn-book.html#svn.basic.in-action.wc.tbl-1).





7. The global project of this lab consists in saying "hello" in several languages. As you are the manager of this project, you firstly create the beginning of the project by writing java source code (see opposite) in `TestSVN` directory. Then, you had to add this code in the repository by using dropdown menu (`TortoiseSVN> Add...`). What happens? See if your file is *really* in the repository (use *GoogleCode>Source>Browse* to browse the code of the project).

```
public class HelloAll_YourName {
    public static void main(String[] args) {
    }
}
```



8. In order to really modify the repository, you add to "validate" all your work by "committing" it. To do so, right click on the file `HelloAll_YourName.java` and use dropdown menu `SVN Commit…` Before validating by clicking on `OK` button, you can write a message to sum up your work to other developers. When you click on `OK` button, a new window appears to explain you what happens (see opposite). How displays the file `HelloAll_YourName.java` in your directory? To be sure that everything is ok, look at the list of icons used by TortoiseSVN (http://tortoisesvn.net/docs/release/TortoiseSVN_en/tsvn-dug-wcstatus.html#tsvn-dug-wcstatus-1) and check that the displayed one on your file is the correct one.

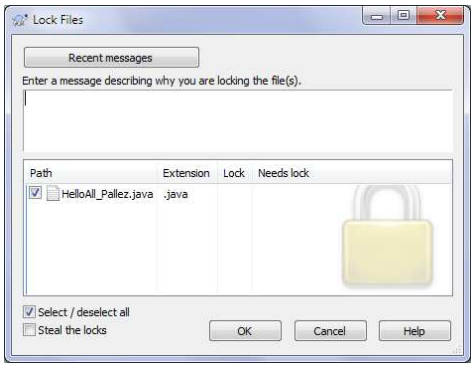
Work in a collaborative manner

9. From now, you must consider that you are NOT the project's manager but a single programmer who receive a single and precise task from his manager. Ask your teacher (which is the manager) to assign you a language. Your goal is to modify the file `HelloAll_Pallez.java` in order to print "hello" in the language the teacher assigned you. To do so, you have to respect following steps:
   a) By using `TortoiseSVN` dropdown menu, choose `SVN Update` to be sure that you are working on the last version of the

```
public class HelloAll_Pallez {
    public static void main(String[] args) {
        String langue = args[0];
        if (langue=="FR")
            System.out.println("Bonjour");
        else if (langue=="US")
            System.out.println("Hello");
        else if (langue=="DE")
            System.out.println("Hallo");
        else if (langue=="IT")
            System.out.println("Bonjourno");
        else if (langue=="ES")
            System.out.println("Hola");
        else System.out.println("#!^¨£$%ù");
    }
}
```

project. You can apply `update` command on several files or directly on a tree (that was not allowed in CVS – http://cvs.nongnu.org/).

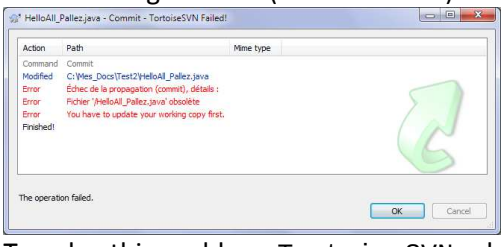
b) Now, as you local code is up to date, use `TortoiseSVN` dropdown menu `Get Lock…` to lock files on which you will work. You can also write a message to other developers to explain what you are doing (see opposite).
c) Make your updates. How displays the modified file?
d) Look at the repository to see whether modifications are reported?
e) Use `TortoiseSVN` dropdown menu SVN Commit… to validate your modifications and report modifications to the repository. Do you need to unlock the file?

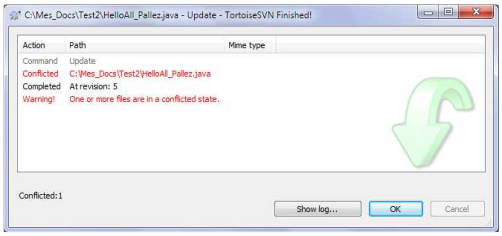


Conflicts management

10. Question 9 corresponds to an asynchronous work because you lock the access on a resource. On the same time, other developers cannot modify the same resource. However, it could be necessary in some situations that several developers modify simultaneously same resource but not at the same place in the resource: it corresponds to synchronous work and it is allowed by SVN.
    Ask your neighbor to work with you to test this most common situation on the file named `HelloAll_YourName.java` without using the `lock` option and follow those steps (look at remark below if you are alone):
    a) `Person1` and `person2` make an `update` ;
    b) `Person1` modify file named `HelloAll_YourName.java` ;
    c) `Person1` make a `commit` ;
    d) `Person2` modify same file named `HelloAll_YourName.java` ;
    e) `Person2` make a `commit` (do not touch anything… wait for question 11).

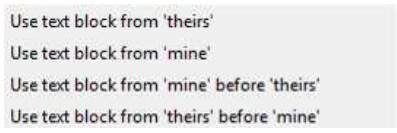
    Remark: Unfortunately, if you are alone, you can simulate the fact you are two different developers by creating TWO working copies on your PC and making a `checkout` on the same repository. Then, use one directory for `person1` and the other directory for `person2` in previous steps.
11. So, when `person2` made the `commit` action, `TortoiseSVN` indicates you that there are some errors by the following window (see left below):

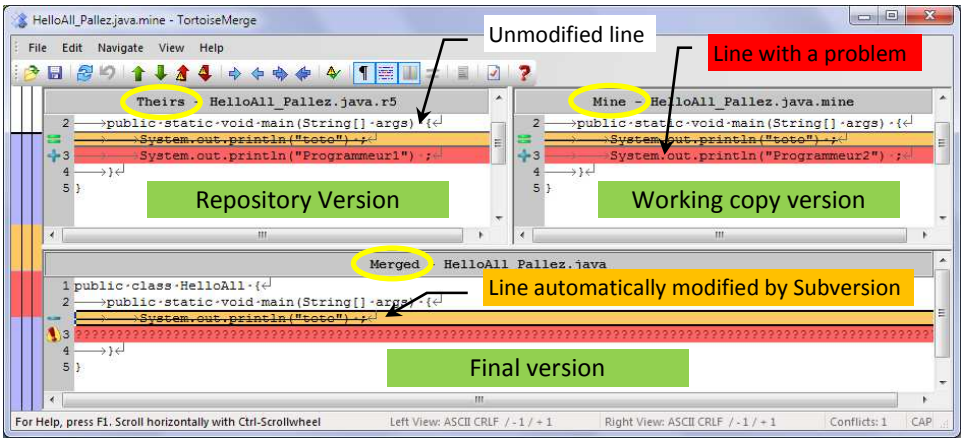




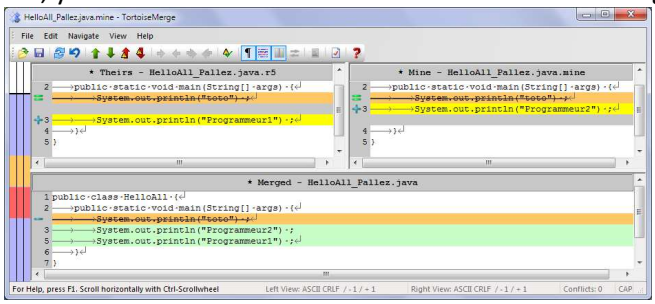
To solve this problem, `TortoiseSVN` ask you to make an `update` action on the conflicted file. **Note that the update action will not affect your last modifications!** Once the update action is made, it might happen that TortoiseSVN solves automatically conflicts or not (see right above). In order to test the case where `TortoiseSVN` is not able to solve automatically conflicts, follow steps of *question 10* to modify the same file at the same line with your neighbor.

12. To solve situation previously presented, you must manually manage conflicts by choosing `Edit Conflicts…` of `TortoiseSVN` dropdown menu on the conflicted file. A window will appear (see next page). In fact, you launched the tool named `TortoiseMerge` which allows comparing file presently stored in the repository (see top left) named *Theirs* with the file presently store on your working copy (top right) named *Mine* and with the file you will send to the repository with solved conflicts (bottom) named *Merged*. To really solve conflicts, right-click on a conflicted line in the merged version and choose which is the best version (see opposite).

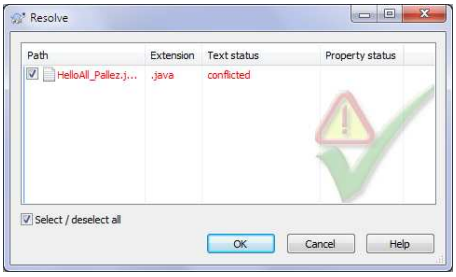


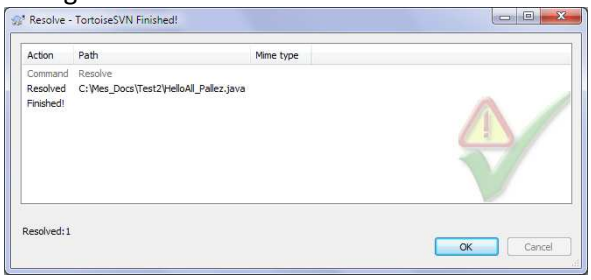


Once conflicts are solved, you should obtaina window similar to the following:



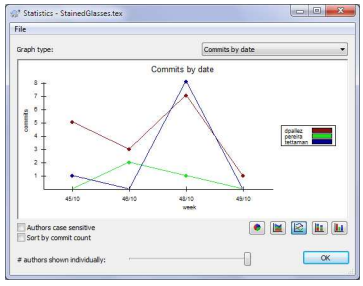
Save results and close the window. You should obtain following screens:





Look at the status of the file you have just modified? What should you do? Do it!

13. The final code of this project (5 different languages) can be found in *question 9*. But, rather that each person or binomial writes the complete source code of this project, we will share the global task into 5 tasks, as many as there is different languages. Ask your teacher to divide your group into 5 subgroups so that each subgroup implements one language. By this way, you will work on a file named `HelloAll.java` in a collaborative and synchronous manner.



14. Everybody who is working on a project can visualize who was really working on the project and how many times he had committed and when? To see that, use `TortoiseSVN` dropdown menu Show `log`… You can also see comments associated to each revision. You can have a graphic visualization by clicking on the `Statistics` button (see opposite) where you can change the information and style of graphic to show.

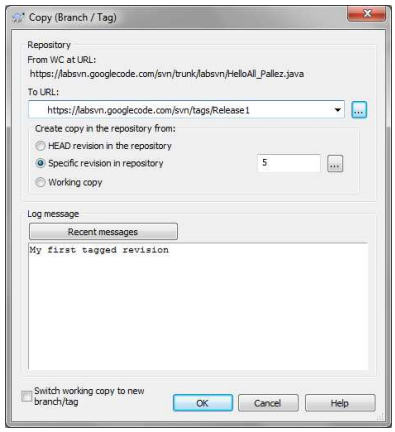
Revisions management

15. As you should already know, a good software is a tool that should evolve in the time in order to always satisfy user's needs. Software engineering suggests developing projects by steps. One step, what we can call *revision* corresponds to a source code allowing to execute the project in stable state. One principle of software engineering consists in "tagging" and saving each revision that corresponds to a state of the project during his history. Consequently, one repository is often structured by the following by analogy to a natural tree:

a) A directory corresponding to the last version currently developed of the project called the "trunk";
b) A directory storing all stable revisions that were saved during all the projects called "tags";
c) A directory saving all exploratory revisions called "branches". This part is developed in *question 19*.

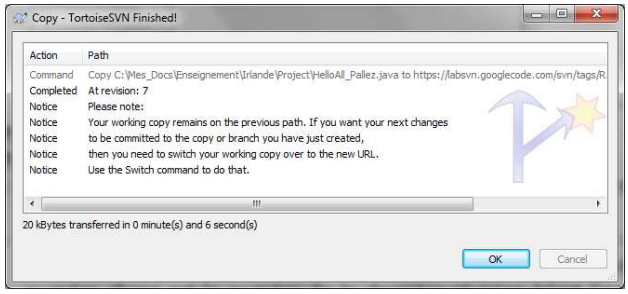
So, during the life cycle of the project, the trunk will always represent the last version under development of the project. Until trunk version could be considered as a stable version, it should be "tagged" and saved in tags directory. Trunk version can continue to evolve and so can once again be tagged as many times as you want. By this way, repository will contain a lot of revision of your project; that is why developers should not reference the entire repository on their working copy but only *one* revision *at a time* saved in this repository.

16. The interest of tagging revision in the repository is to browse through revisions and by this way for instance referencing one revision, executing it, testing it, and change the reference to another revision. For changing a reference to a revision, use `switch` command. Test this command by referencing a previous revision of your project. Cancel previous action by referencing last (or HEAD) version of your project.
17. Create a new directory called `Project` in your workspace that references `trunk` directory of LabSVN repository.
18. As in *question 13*, you have finished to develop a stable version of your project that consists in saying "hello" in several languages. Copy the file `HelloAll_YourName.java` in your working copy (which still references trunk of `Project`) and now tag your working copy by using `TortoiseSVN` dropdown menu `Branch/Tag`:



a) In the field `To URL`, you should give the directory where you want to save *tagged* revisions (`https://labsvn.googlecode.com/svn/tags/`) followed by a name of the tagged revision (choose your name for instance, see opposite).
b) Then, you should choose what you want to tag:
   i. The last version of the `trunk`, in this case, choose button `HEAD revision in the repository`;
   ii. A specific revision and in this case, choose `Specific revision in repository` and choose the number of this specific revision. This choice is equivalent of previous one if revision's number is equals to HEAD revision. This is the choice made by default by `TortoiseSVN`;
   iii. `Working copy` which is not obliged to be up to date on the repository.

Result of this operation is represented in opposite window.



Simultaneous revisions management of the same project (branches)

19. One of your colleagues gives you a new idea for dealing with locality (see opposite). As you are a very good manager, you don't want to lost what tour team has already done and which is still used by your customers. Moreover, actual version (`trunk`) is also used by another team of developers to solve some bugs and try to add others languages. However, as the idea is very intelligent, you want to test it without disturbing tour employees. To do so, you create a *branch* to your software development by using the same `TortoiseSVN` dropdown menu `Branch/tag`. A *branch* revision is physically equivalent to a *tag* revision (project's files are saved in a directory and named) but is conceptually different because it is used to add and test new functionalities to trunk version. In contrary, tag revisions should not evolve. The problem you should have in mind is that, once you create a branch, you will have two different softwares: trunk revision plus branches revisions because trunk could have evolved simultaneously. Usually, *one* branch is used for *one* new functionality of the project. Once this functionality is developed, tested and validated, `TortoiseSVN` allows you to *bring back* the functionality of the branch on the trunk:

HelloAll_Pallez.java

```
import java.util.Locale ;
import java.util.ResourceBundle ;
public class HelloAll {
    public static void main(String[] args) {
        Locale local_current = Locale.getDefault() ;
        ResourceBundle myRessources = ResourceBundle.getBundle("Resources", local_current);
        System.out.println (myRessources.getString("MsgHello")) ;
    }
}
```

Resources_fr_FR.properties

```
MsgHello = Bonjour
MsgBye = Au revoir
```

    a) Make an update of your working copy which should now reference trunk of LabSVN project. To know what references your working copy, look at the *subversion* properties of the directory and check it is the same URL as opposite.

    b) On this same directory (`Project`), create a new `branch` by choosing the `branches` directory of the repository followed by a directory named by your name (for instance on the opposite: …`/branches/Pallez`). Do not forget to tick the case `switch working copy`… in order to work from now on the revision corresponding to the branch you had just created rather than working on trunk revision. Result of this operation is shown below:

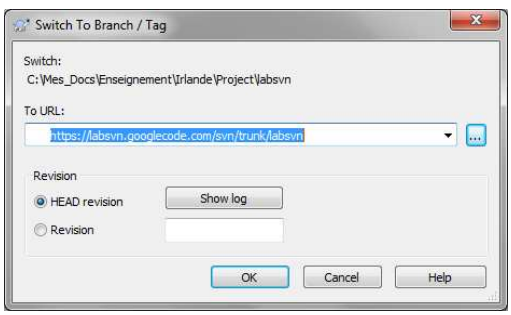
    Please, verify using your browser and URL googlecode/LabSVN that what you have done until correspond to what it was asked.

    c) As did previously, modify branch revision to take into account internationalization of the application using `Locale` and `ResourceBundle` java classes.

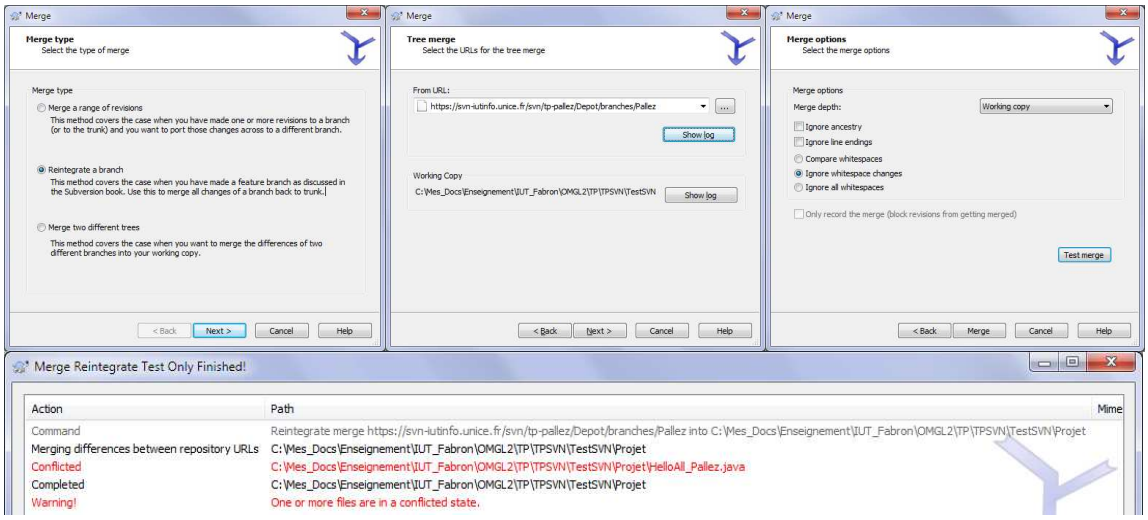
20. Now, as the branch is completely finished, you should modify trunk version so that it will take into account the functionality developed in the branch. However, you cannot simply "copy" the branch to the trunk because trunk revision might evolve during your development of the branch. You should better *merge* the two revisions in your `working copy` and then commit this merged revision to the trunk:

    a) Make a commit on your working copy to be sure you had saved last version of branch revision to repository;

    b) As you want to merge both revisions (trunk and branch), you should now reference trunk revision by using `Switch` command of `TortoiseSVN` dropdown menu (see opposite). By this way, your `working copy` will reference last version of the trunk;



    c) In a Windows file manager, select `Project` directory on your `working copy` and use `Merge` command of `TortoiseSVN` dropdown menu in order to merge revisions. At the third presented screen (see below), you should use button `Test Merge` before applying it:



d) As you merged two revisions that may have evolved independently and simultaneously, you might be able to manage conflicts (see *question 11*). You should also know that *merging* two revisions is rarely used because of those manually conflicts management, especially if trunk revision is consequent.

To go further

21. Test all other commands of `TortoiseSVN` dropdown menu that you didn't see during the lab. Especially,
    a) Try to find how to know whether files have been modified when you were not working without updating your working copy?
    b) Now, you know that some modifications have been done on the repository, you make an update, and now you want to know what are the modifications in the each file …
22. Test subversion on Unix like OS : you should use shell commands (`svn commit`, `svn update`, `svn add file1.java` …).

Resources used to write this lab (often in French)

http://kevin.fardel.perso.esil.univmed.fr/documentation/TutorielTortoiseSVN.pdf
http://tortoisesvn.net/docs/nightly/TortoiseSVN_fr/
http://svnbook.red-bean.com/nightly/fr/