Introduction à subversion

SUBVERSION

http://subversion.tigris.org

La gestion de version avec subversion (SVN).

SUBTRAIN by POLARION

IMERIR

Qu'est-ce qu'un VCS* ?

**Version Control System = système de gestion de version*

- Un outil pour maintenir l'ensemble des versions d'un logiciel.
- Conserve les révisions successives du projet
  - possibilités de revenir en arrière, de voir les changements
- Facilite la collaboration entre les intervenants.
  - chacun travaille avec son environnement.
  - plusieurs personnes travaillent sur les mêmes fichiers simultanément.
- Chacun accède à une copie des fichiers, les originaux restent sur le référentiel.

IMERIR

Un VCS n'est pas..

- Un système de construction de version.
- Un système de déploiement d'applications.
- Garant de la qualité d'un projet.
- Une alternative à la communication entre les membres d'une équipe projet.

A quoi ça sert un VCS ?

- Avez-vous déjà connu ça ?

C'est laquelle la dernière version stable ?

Comment corriger le bug de l'ancienne version livrée au client il y a un mois ?

Utiliser un VCS pour quoi faire ?

- Le développement logiciel.
- L'administration système
- La gestion de site internet.
- La documentation.
- La gestion de tout projet utilisant des documents électronique comme support principal.

Bref historique

- Les « dinosaures »
- Accès local uniquement
  - SCCS 1980 (Source Code Control System)
  - RCS 1982 (Revision Control System)
  - CSSC 1998 (Compatibly Stupid Source Control) SCCS libre

Bref historique

- Centralisés avec un accès réseau:
  - CVS 1989 Concurrent Versions System
  - Perforce 1995
  - Subversion 2001
- Distribués:
  - GNU Arch 2001
  - Darcs 2003
  - Git 2005
  - Mercurial 2005

Les concurrents de subversion

- Perforce: http://perforce.com/
- Arch: http://gnuarch.org/
- sourcessafe: http://msdn.microsoft.com/vstudio/previous/ssafe/
- bitkeeper: http://www.bitkeeper.com/
- clearcase: http://clearcase.com

etc... voir sur wikipedia:

http://en.wikipedia.org/wiki/Comparison_of_revision_control_software

Subversion

Bref historique:

- **June 2000 -** Début du codage.
- **2002** – 1ère release
- **2004** – Feb. Release 1.0.0
- **2006** – Sept. Release 1.4.0
- **2007** – Juin Release 1.4.4
- à venir Release 1.5.0

Estimation du nombre de developpeurs: 1.4 million

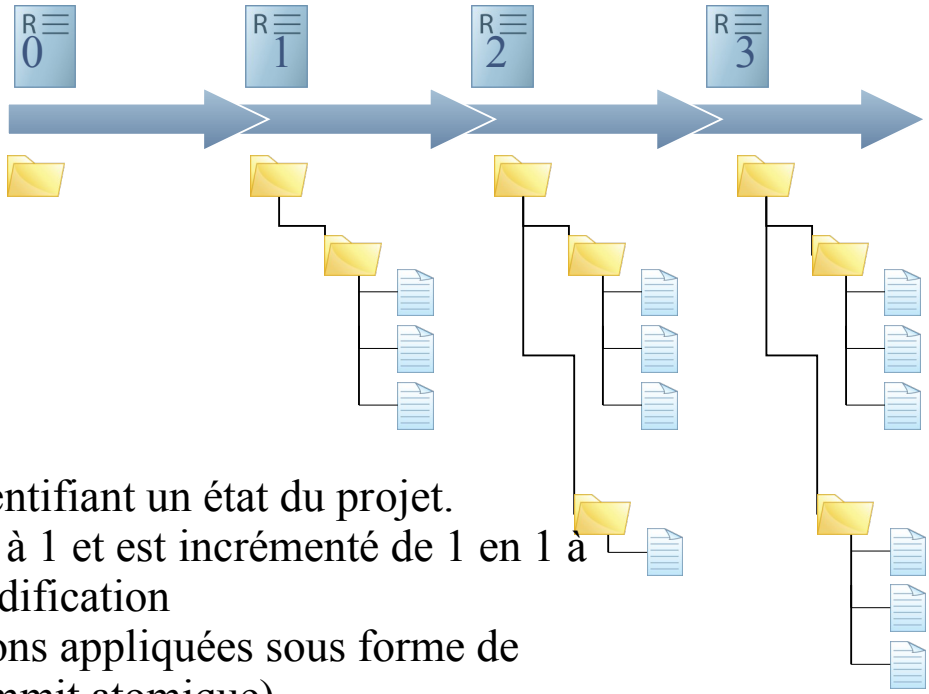
Où est utilisé subversion ?

- Quasiment partout !
- KDE, Gnome,
- Sourceforge,
- GoogleCode, etc...
- et à IMERIR ?

Quelques notions

- Le dépôt (repository):
  - un emplacement où sont gardées les informations concernant les projets.
    - l'historique des versions des fichiers stockés,
    - les journaux de chaque modification,
    - la date, l'auteur d'une modification etc...
- Révision:
  - un numéro identifiant un état du projet.
  - Il commence à 1 et est incrémenté de 1 en 1 à chaque modification (subversion)
  - modifications appliquées sous forme de patchs (commit atomique)

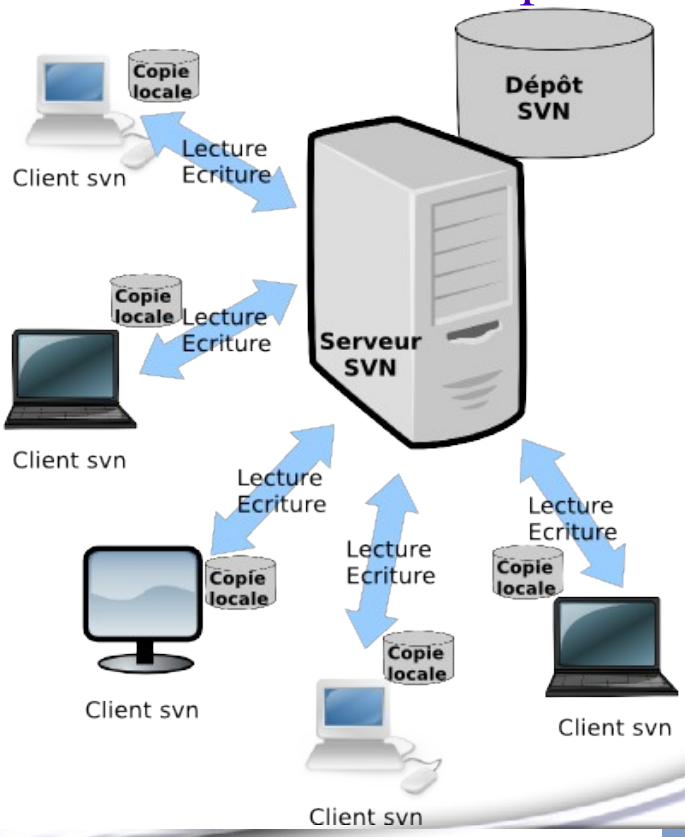
Numéros de revision:



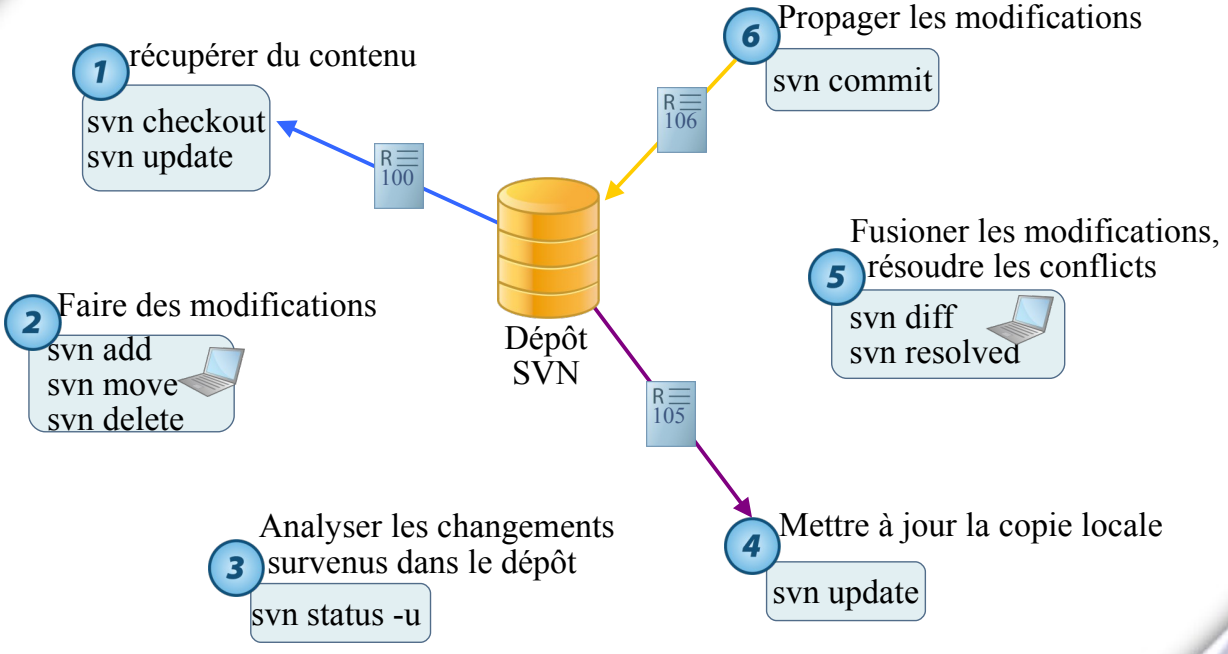
- numéro identifiant un état du projet.
- commence à 1 et est incrémenté de 1 en 1 à chaque modification
- modifications appliquées sous forme de patchs (commit atomique)

Architecture de subversion

- un outil client/serveur multi-plateforme



Le Cycle de travail



Fonctionnement de subversion

- Les bases:
  - Obtenir de l'aide.
  - commande « **`svn help`** »
    - Exemple:
    - `svn h elp h elp`
    - `help (?, h ): Dé crit l'usa ge de ce progr amme ou de ses sous- comma ndes.`
    - `usage : he lp [S OUS_C OMMAN DE... ]`

    `...`

Fonctionnement de subversion

- Les bases:
  - Importation (ajout) d'un projet dans le référentiel.
  - commande « **`svn import`** »
    - `svn i mport (CHE MIN) URL - m "co mment aire oblig atoir e"`
    - Charge récursivement une copie de CHEMIN vers URL.
    - Avec CHEMIN, le chemin vers le répertoire contenant le projet à importer
    - et URL, l'url du dépôt sur le serveur.
- Attention: il ne faut plus travailler dans le répertoire CHEMIN, mais obtenir une copie de travail du serveur !

Fonctionnement de subversion

- Les bases:
  - Organisation du dépôt
    - Le dépôt peut contenir plusieurs projets. Chaque projet peut contenir un ou plusieurs répertoires

```
|------ projet alpha
|       |------ module m1
|       |       \------ sous-module m1-a
|       \------ module m2
\----- projet beta
```

  - On peut ajouter, supprimer ou déplacer des répertoires, comme sur un système de fichiers distant, avec:
    - « mkdir url/répertoire »,
    - « delete (del, remove, rm) url/répertoire »
    - « move (mv, rename, ren) url/répertoire »

Fonctionnement de subversion

- Les bases:
  - Organisation standard du dépôt:
  - il est recommandé d'organiser le projet de la façon suivante:
  - un répertoire « trunk » (développement courant),
  - un répertoire « tags » (stockage des versions identifiées),
  - un répertoire « branches » (expérimentations et bugs)

```
|------ projet alpha
|       |------ trunk
|       |       |------ README
|       |       |------ source.c
|       |------ tags
|       |------ branches
...
```

Fonctionnement de subversion

- Les bases:
  - Récupération d'une version dans un répertoire de travail.
  - Avec la commande « **`svn checkout`** »:

```
svn checkout repositoryUrl (destination)
```

avec repositoryUrl le chemin vers le dépôt et destination le chemin où le projet doit être copié.

On travaille toujours sur une copie locale du projet, dans un répertoire appelé « bac à sable » (sandbox). On reporte ensuite les modifications sur le serveur.

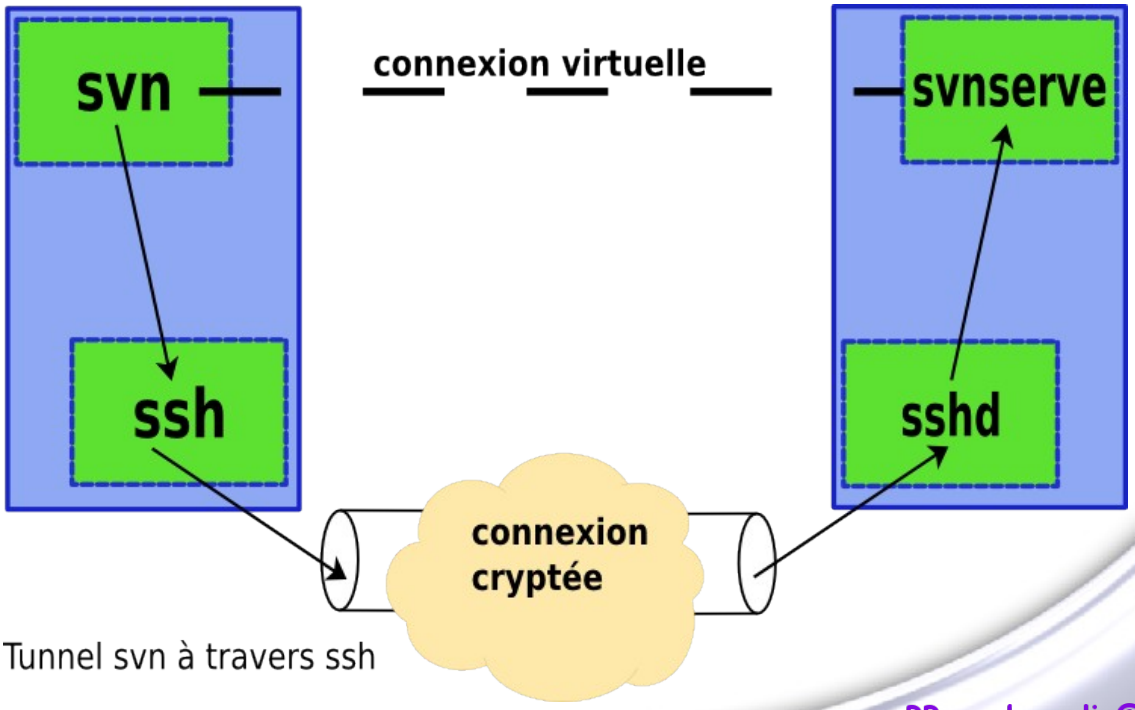
Fonctionnement de subversion

- Les bases:
  - Moyens d'accès au dépôt:

Protocoles	Méthode d'accès
`file:///`	accès direct au dépôt (au disque local).
`http://`	accès via le protocole WebDAV a un serveur Apache + Subversion.
`https://`	comme `http://`, mais avec cryptage SSL.
`svn://`	accès à travers le protocole svn à un serveur `svnserve`.
`svn+ssh://`	comme `svn://`, mais à travers un tunnel SSH.

Fonctionnement de subversion

- Les bases:
  - Accès au dépôt par svn+ssh:



Tunnel svn à travers ssh

Fonctionnement de subversion

- Les bases:
  - Ajouter un fichier ou un répertoire avec la commande svn add:
    - **`svn add CHEMIN`**

      avec CHEMIN, le chemin vers le ou les éléments (fichiers ou répertoires) à ajouter.
  - Les éléments sont marqués ajoutés, il faut ensuite propager la modification, pour qu'elle soit effective.
- Les fichiers n'appartenant pas au projet ne sont pas transférés sur le serveur.
- Ne pas ajouter un fichier que l'on peut générer à partir d'un autre fichier déjà dans le dépôt.

Fonctionnement de subversion

- Les bases:
  - Validation d'une modification:

```
svn commit (CHEMIN) -m "description"
```

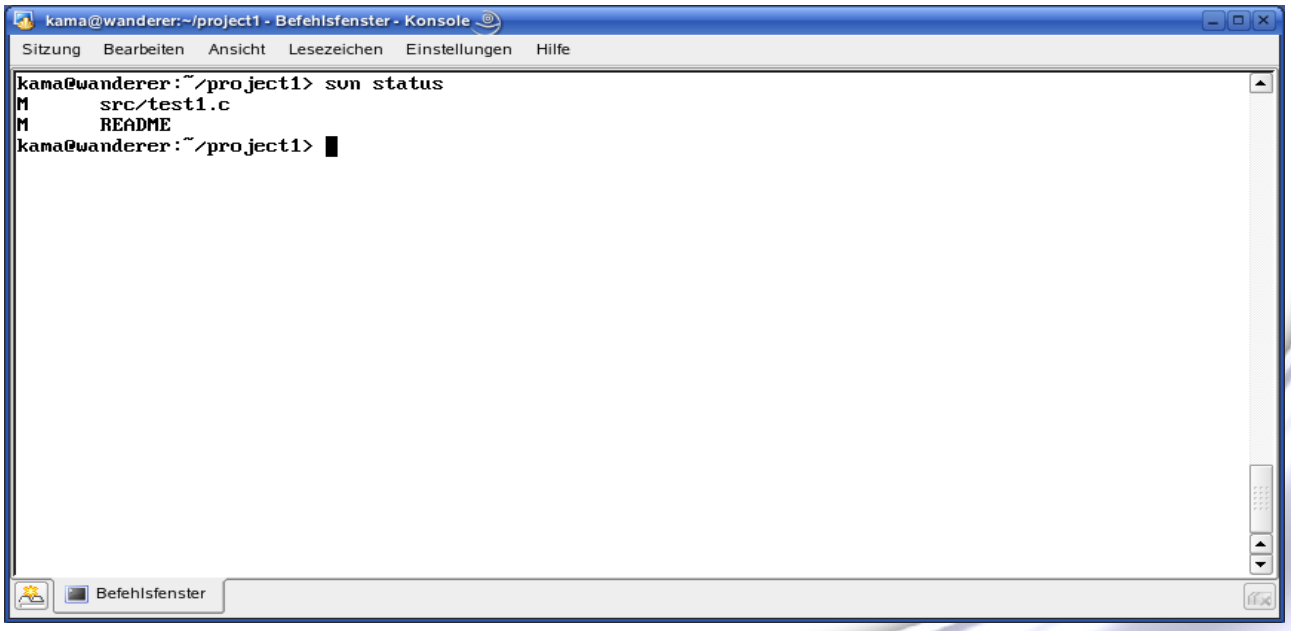
CHEMIN, le chemin vers le ou le(s) élément(s) que l'on souhaite publier sur le repository. La commande est récursive par défaut.

Si CHEMIN n'est pas indiqué, le « commit » s'applique au répertoire courant ".". L'ensemble des fichiers connus modifiés en local seront mis à jour.

- Remarques:
  - Si pas de description, l'éditeur par défaut est lancé...
- Si le dépôt a été modifié sur le serveur: le « commit » échoue. Il faut mettre à jour la copie de travail.

Fonctionnement de subversion

- Les bases:
  - connaître le statut des fichiers avec la commande « **`svn status`** »:



Fonctionnement de subversion

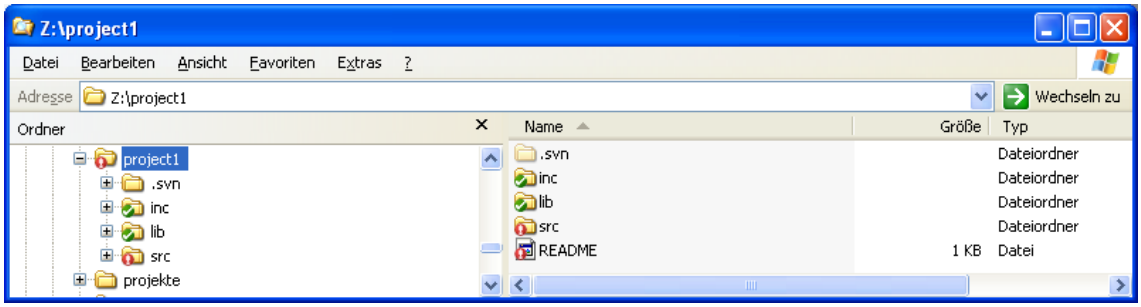
- Les bases:
  - Analyse du résultat d'un « status »:

La première colonne:

- **A**: Ajout dans le dépôt programmé.
- **C**: L'objet est en Conflit.
- **D**: Suppresion du dépôt programmée.
- **M:** L'objet est modifié.
- **R**: Remplacement programmé.
- **X**: Le répertoire n'est pas versionné, mais est lié avec une définition externe.
- **?**: Element en dehors du système de gestion de version.
- **!**: Element manquant ou incomplet géré par subversion (supprimé en dehors de svn).
  => **svn update** récupère l'élément depuis le dépôt, ou
  => **svn revert file** restore l'élément.
- ~: L'objet existe normalement dans le dépôt, mais n'a pas la même nature..
- **I**: L'élément est ignoré.

Fonctionnement de subversion

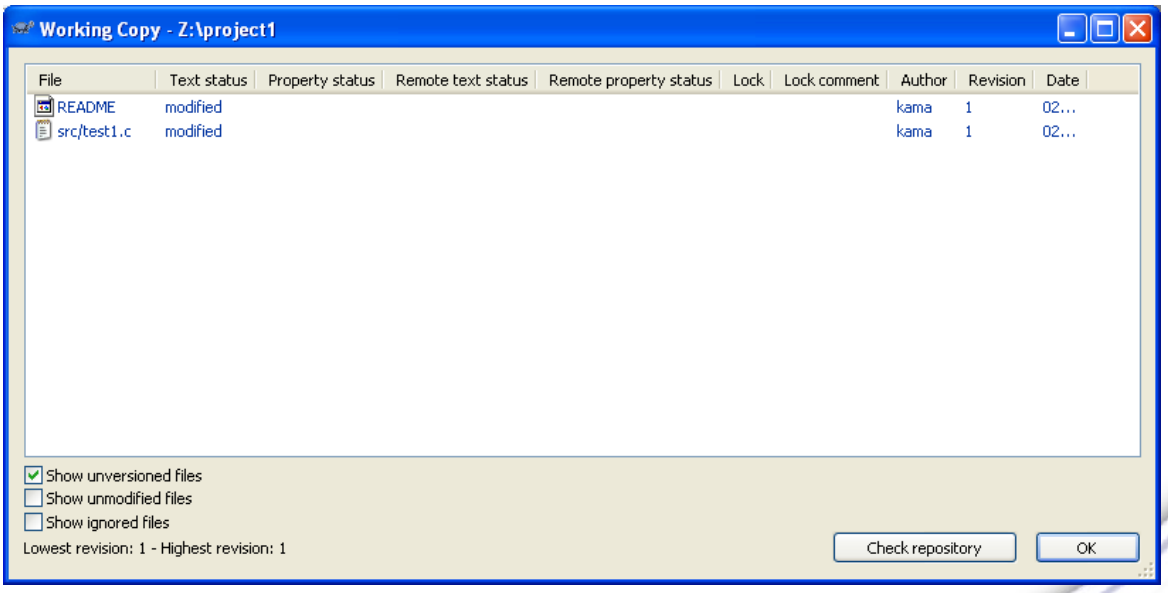
- Statut avec avec TortoiseSVN



Status normal	Locked
Added file/directory	Modified
deleted file/directory	Read only

Fonctionnement de subversion

- TortoiseSVN -> Check For Modifications



Fonctionnement de subversion

- Les bases:
  - Synchronisation entre le repository et un répertoire de travail avec la commande:

```
svn update (CHEMIN)
```

    - CHEMIN, le chemin vers la copie locale à mettre à jour. Si pas de chemin, la commande s'applique au répertoire courant « . ».
  - Pourquoi ?
    - Après avoir obtenu du dépôt une version du projet, les autres développeurs continuent à travailler et peuvent valider des modifications sur le référentiel...
  - Remarque:
    - La commande **`svn status`** permet de savoir si la version locale est à jour.

Fonctionnement de subversion

- Copie de travail actualisée par rapport à la révision HEAD (défaut), mais possibilité de préciser une révision avec « -r ARG »:

```
svn update -r PREV foo.c
svn update -r {"2006-02-17
 15:30"}
```

  - Si possible, les modifications qui ont été faites sur le serveur sont reportées automatiquement dans les fichiers de la copie locale,

    sinon => **conflit**.

- Remarque:
  - Plus on est à jour, moins on risque de générer des conflits. => Il faut faire des « update » souvent.

Fonctionnement de subversion

- Les bases:
  - Résolution des conflits:
  - Dans 90% un « update » suffit...

```
svn commit
Sending        foo.c
svn: Commit failed (details follow):
svn: Your file or directory 'foo.c' is
 probably out-of-date [...]
svn update
G   foo.c
Updated to revision 12.
```

  - Svn fusionne « auto-magiquement » les modifications.
  - Le conflit est résolu, il reste à (re)faire le commit.

Fonctionnement de subversion

- Les bases:
  - Résolution des conflits:
  - Si la même ligne est modifiée, pas de fusion auto, le conflit est signalé par la lettre « C »:

```
svn update
C   foo.c
Updated to revision 13.
```

  - Des nouveaux fichiers apparaissent dans la sandbox:

```
foo.c.mine ( version modifiée en local )
foo.c.r12 (version de base, commune au 2)
foo.c.r13 (version modifiée du dépôt)
foo.c ( spéciale résolution avec les différences entre les versions).
```


Fonctionnement de subversion

- Les bases:
  - Résolution des conflits:
  - subversion indique le conflit dans « foo.c »:

```
<<<<<<< .mine
bool isReady () { if (_bReady) return true
; else return false; }
=======
bool isReady () { return _bReady ; }
>>>>>>> .r13
```

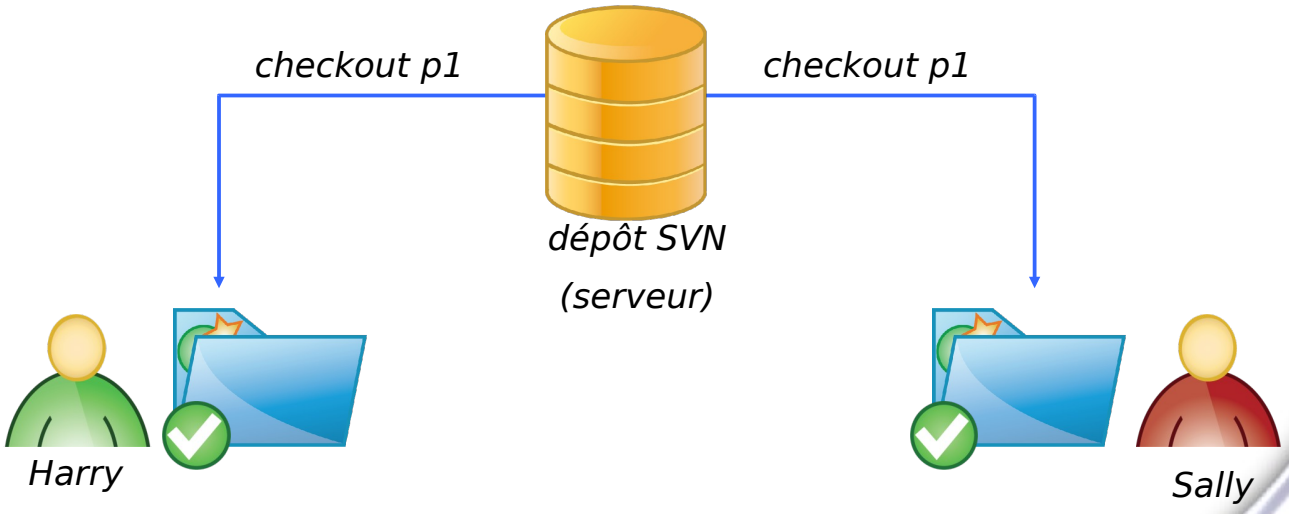
  - Choisir une version et signaler que le conflit est résolu:

```
svn resolved foo.c
Resolved conflicted state of 'foo.c'
```

  - il faut propager la nouvelle version du fichier avec un « commit ».

Conflits

- Deux développeurs “checkout” le même projet depuis le dépôt.



Conflits

- Harry et Sally récupèrent le même projet:

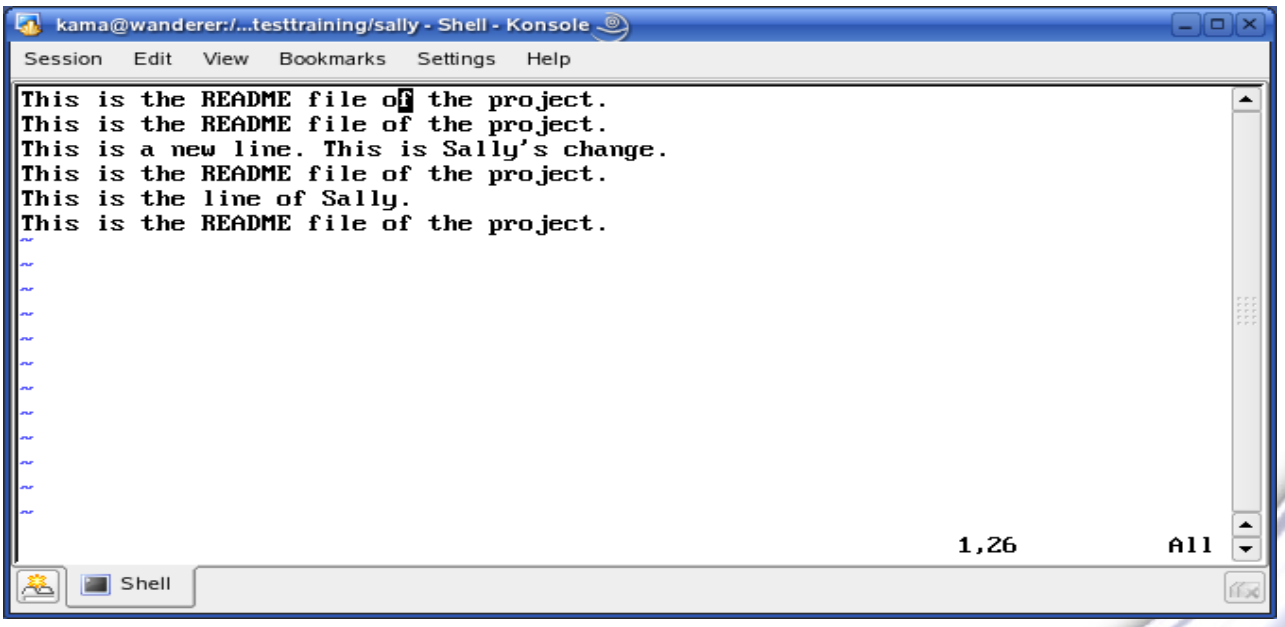
```
kama@wanderer:~/testtraining - Shell - Konsole
Session  Edit  View  Bookmarks  Settings  Help
kama@wanderer:~/testtraining> svn checkout file:///home/kama/repos/project1/trunk harry
A    harry/x.txt
A    harry/lib
A    harry/lib/x.lib
A    harry/lib/t.jar
A    harry/src
A    harry/src/test2.asm
A    harry/src/test1.c
A    harry/newdir
A    harry/newdir/a.doc
A    harry/newdir/x.txt
A    harry/inc
A    harry/inc/test.h
A    harry/LIESMICH
A    harry/README
Checked out revision 4.
kama@wanderer:~/testtraining> svn checkout file:///home/kama/repos/project1/trunk sally
A    sally/x.txt
A    sally/lib
A    sally/lib/x.lib
A    sally/lib/t.jar
A    sally/src
A    sally/src/test2.asm
A    sally/src/test1.c
A    sally/newdir
A    sally/newdir/a.doc
A    sally/newdir/x.txt
A    sally/inc
A    sally/inc/test.h
A    sally/LIESMICH
A    sally/README
Checked out revision 4.
kama@wanderer:~/testtraining>
Shell
```

Conflits

- Ils éditent le **même fichier** dans leur copie locale et leurs modifications se **recoupent**.

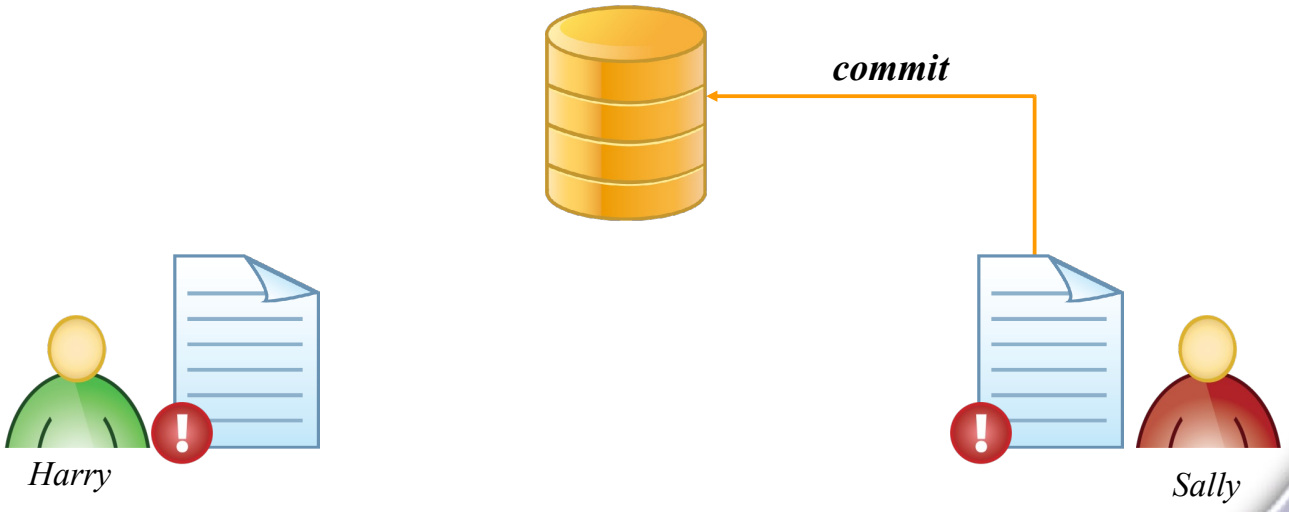
Conflits

- Sally modifie le README:



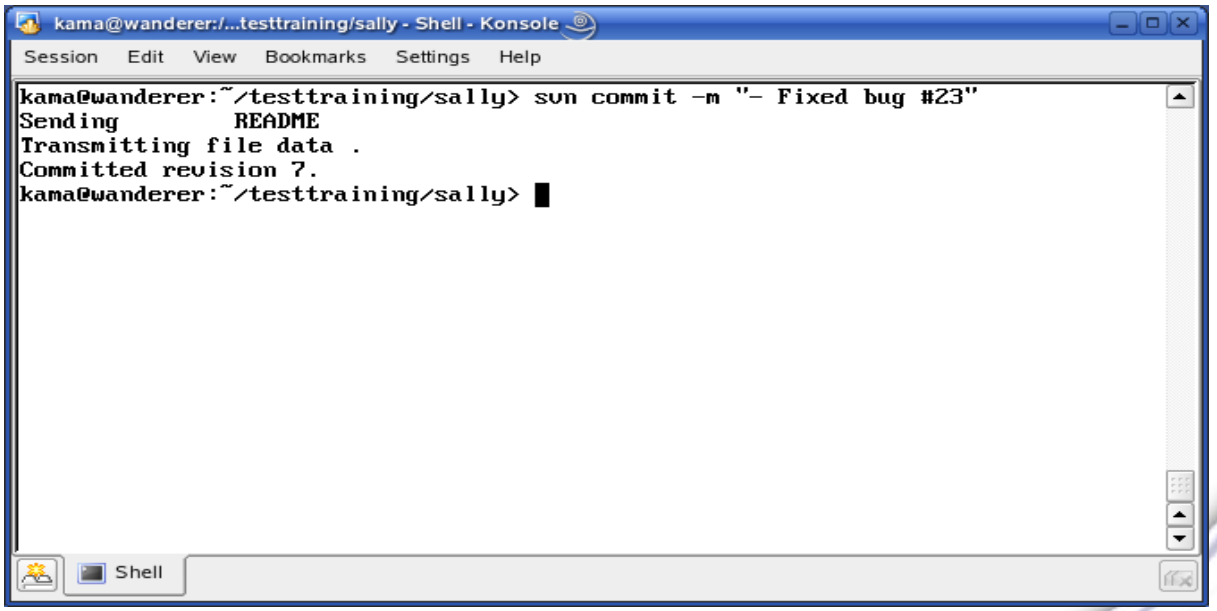
Conflits

- Sally “commit” son travail en premier:



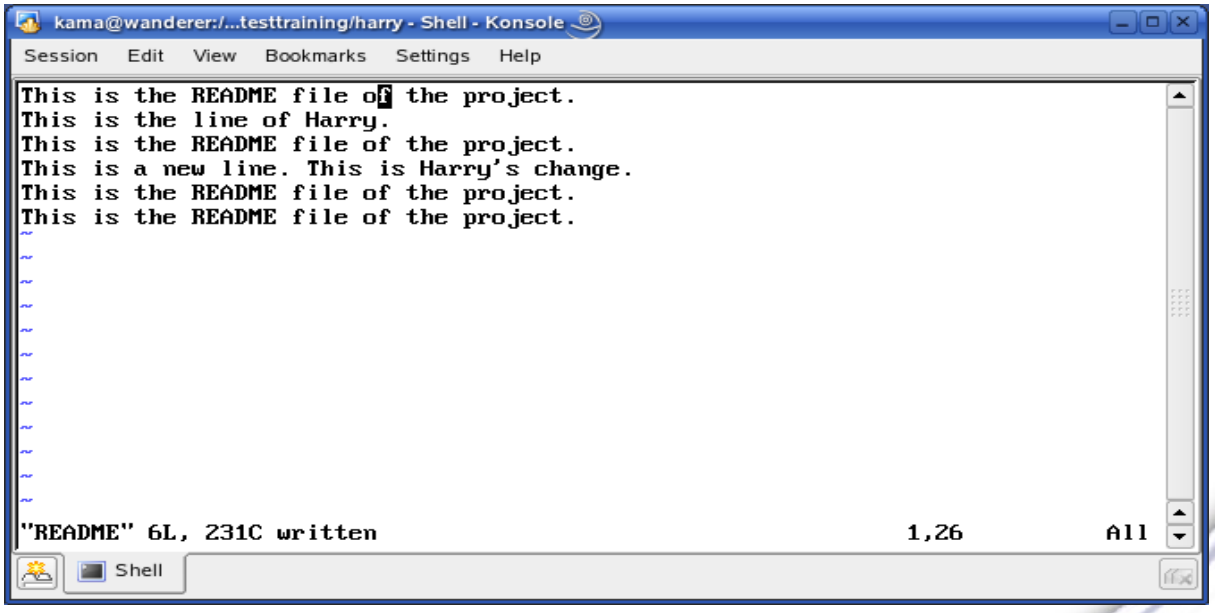
Conflits

- Ce qui donne:



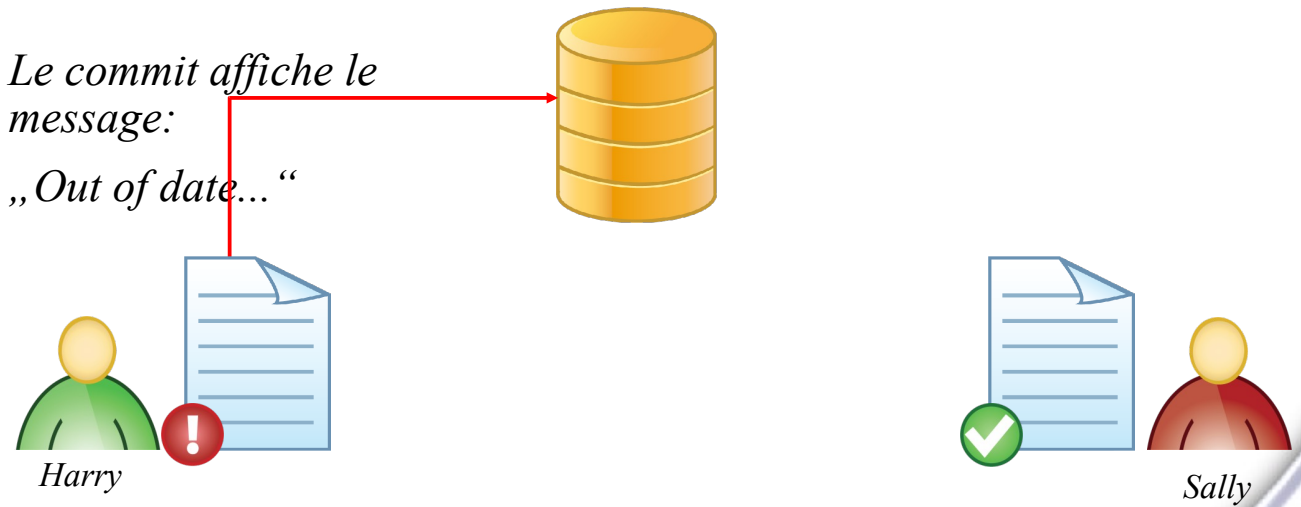
Conflits

- Harry a lui aussi modifié le README:



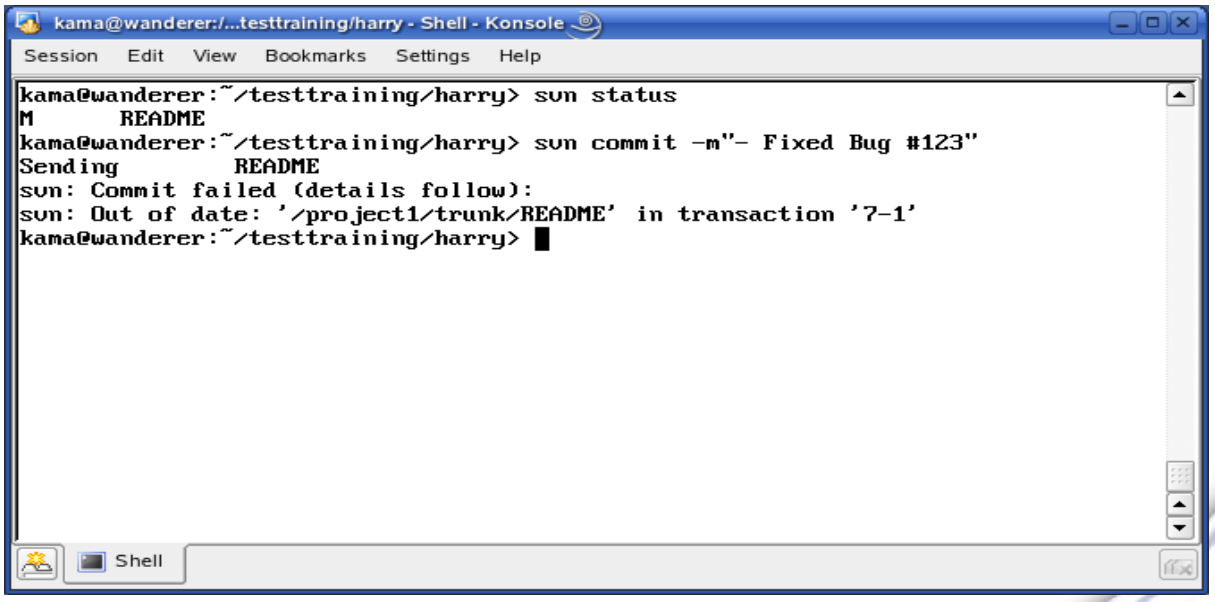
Conflits

- Harry essaie de “commiter” son travail après Sally.



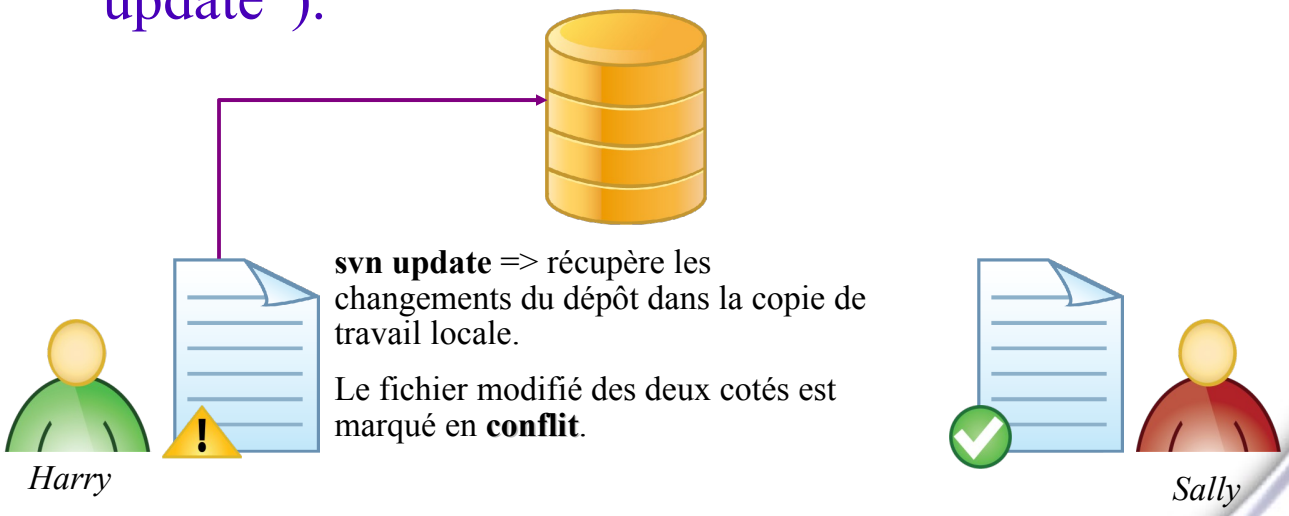
Conflits

- Ce qui donne le résultat suivant:



Conflits

- Harry doit mettre à jour sa copie locale pour avoir les modifiactions faîtes par Sally (faire un “update”).

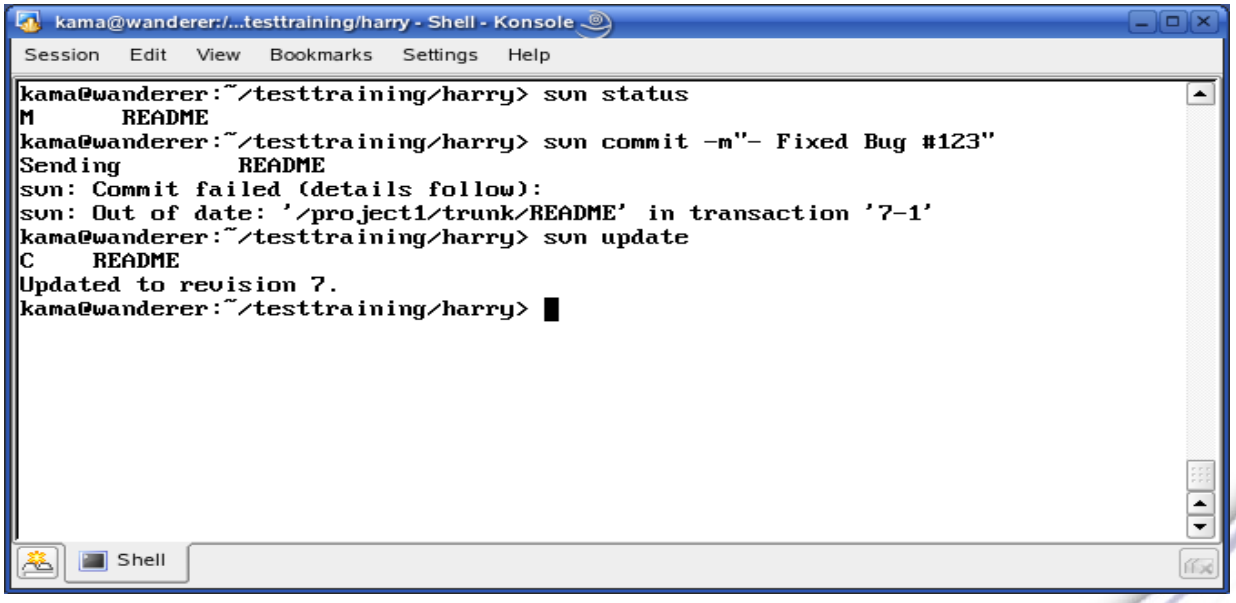


svn update => récupère les changements du dépôt dans la copie de travail locale.

Le fichier modifié des deux cotés est marqué en **conflit**.

Conflits

- Harry “update”sa copie de travail:



Conflits

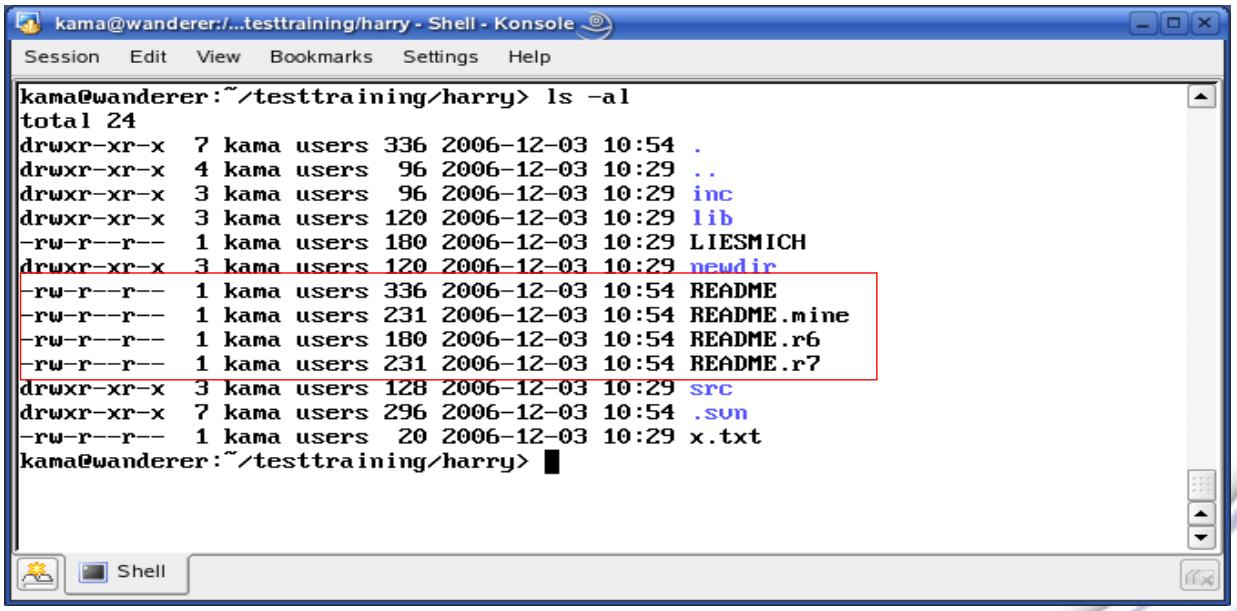
- ***Status des fichiers après un “update”:***
  - ***U:*** *Updated* (received changes from the server).
  - ***A:*** *Added* to working copy.
  - ***D:*** *Deleted* from working copy.
  - ***R:*** *Replaced* in working copy.
  - ***G:*** *merGed* from the repository into working copy.
  - *C: Conflicting* changes from the server.

Conflits

- Quelle aide apporte subversion ?
  - Pour chaque fichier en conflit, svn ajoute 3 nouveaux fichiers:
    1. filename.mine
       - C'est le fichier de la copie locale (avec les derniers changements que l'on a fait).
    2. filename.rOLDREV
       - C'est le fichier qui était la révision de BASE avant l'update de la copie de travail. C'est-à-dire le fichier tel qu'il était avant qu'on le modifie.
    3. filename.rNEWREV
       - C'est le fichier que l'on vient de recevoir du serveur quand on a fait l'update. C'est-à-dire le fichier tel qu'il est à la révision HEAD du dépôt.
  - Le fichier filename contient des marqueurs des différences pour guider le développeur.

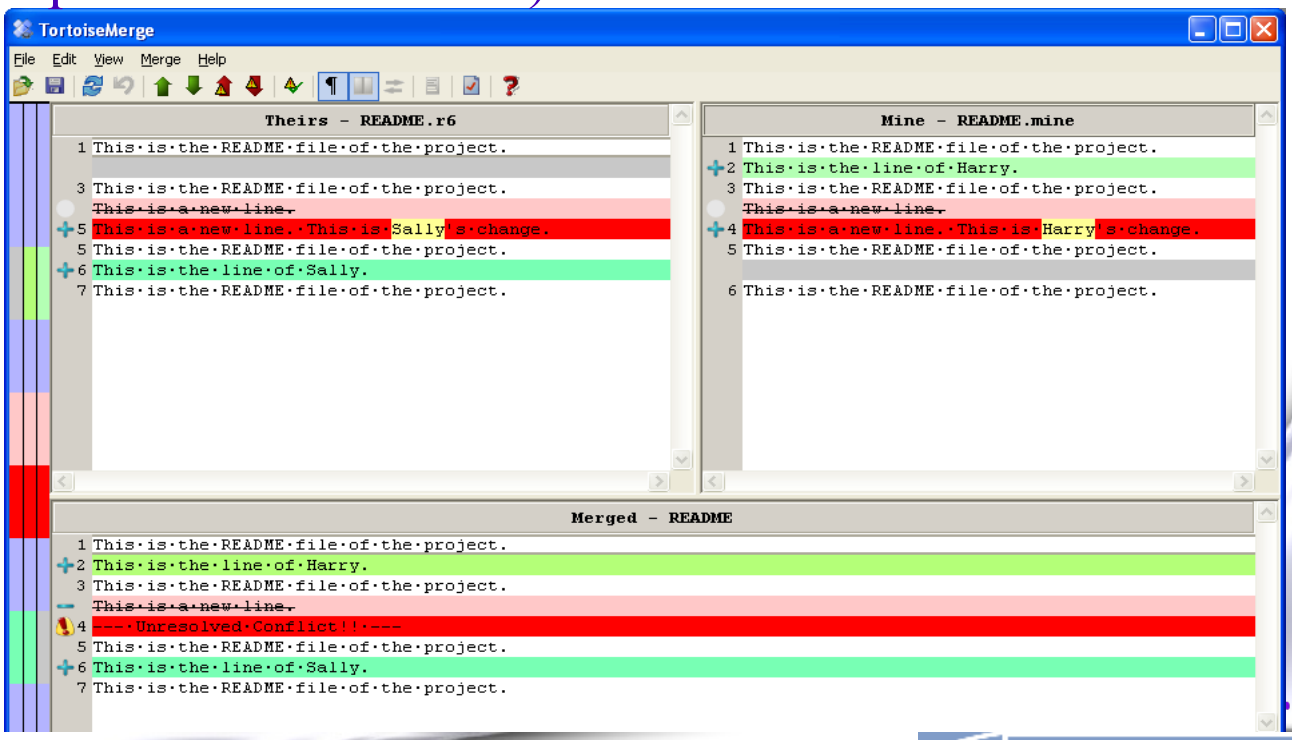
Conflits

- Les fichiers dans la copie de travail:



Conflits

- Edition du conflit avec TortoiseSVN (accessible par le context menu):



Conflits

- Comment résoudre le conflit ?
  - Aucun outil ne peut résoudre ce conflit.
  - Seuls les développeurs peuvent comprendre la situation et faire un choix.
  - Dans l'exemple, les deux membres de l'équipe doivent parler du conflit et se mettre d'accord sur une solution...

La communication dans la gestoin d'un projet est très importante !

Conflits

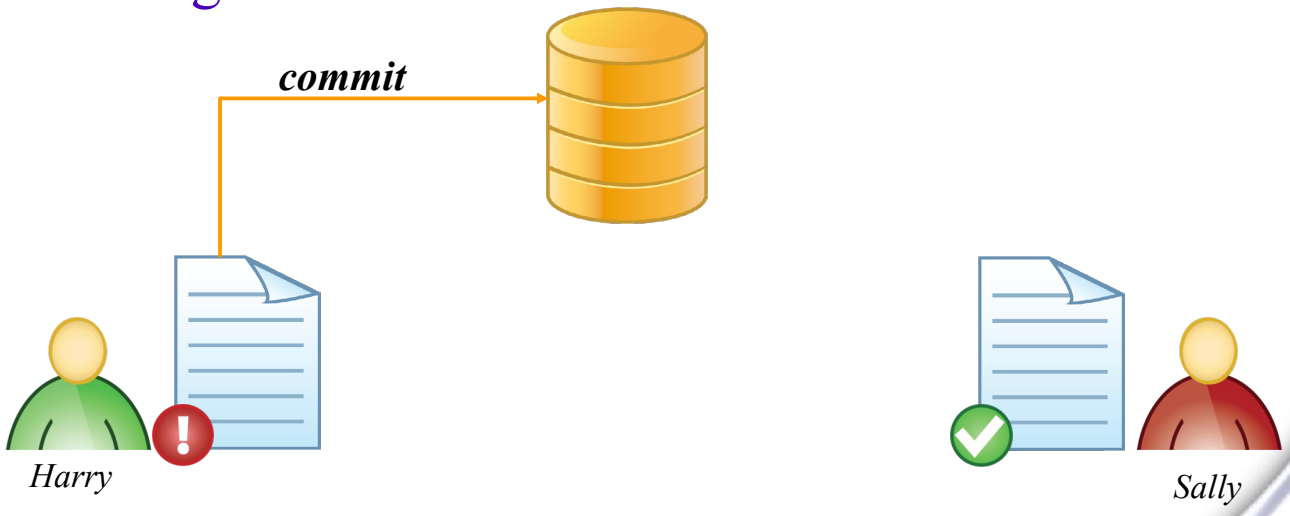
- Le conflit a été tranché. Le code est corrigé.
- Avant de pouvoir publier la modification, il faut indiquer à subversion que le conflit est résolu:

```
svn resolved destination
```

Avec destination le fichier ou le répertoire concerné.

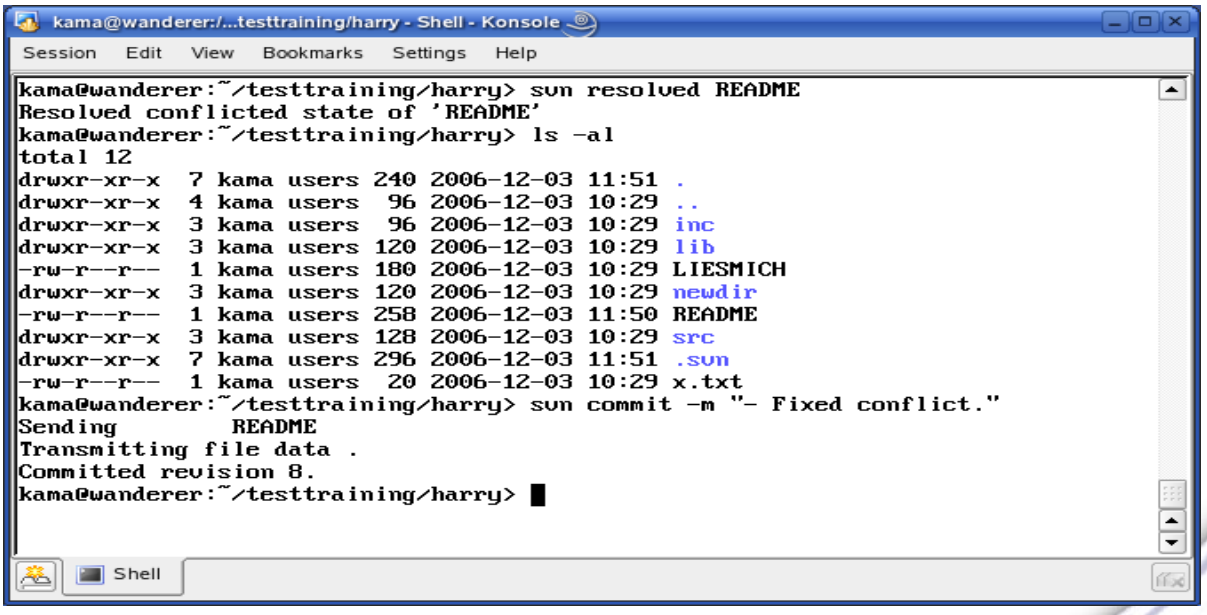
Conflits

- Harry peut maintenant “commiter” ses changements.



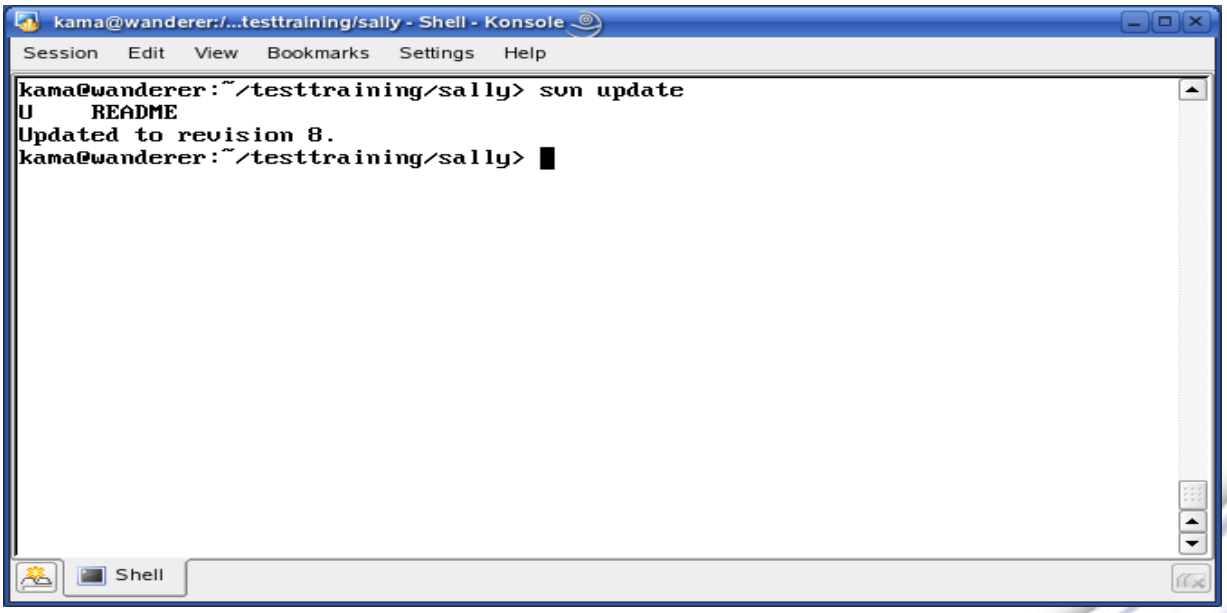
Conflits

- On “commit” la résolution:



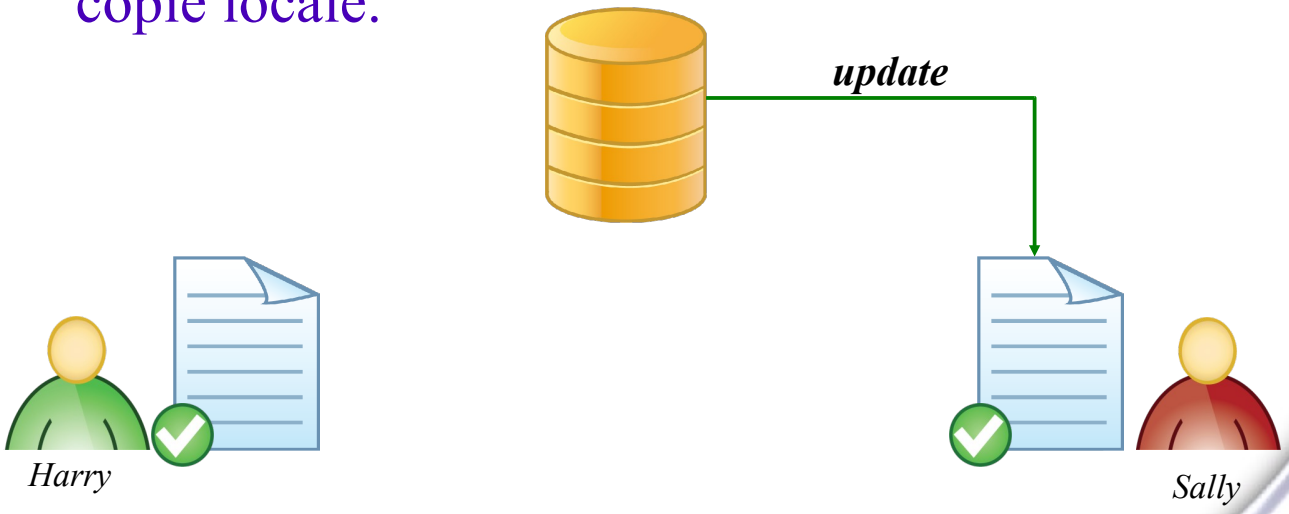
Conflits

- Au final Sally update sa copie locale:



Conflits

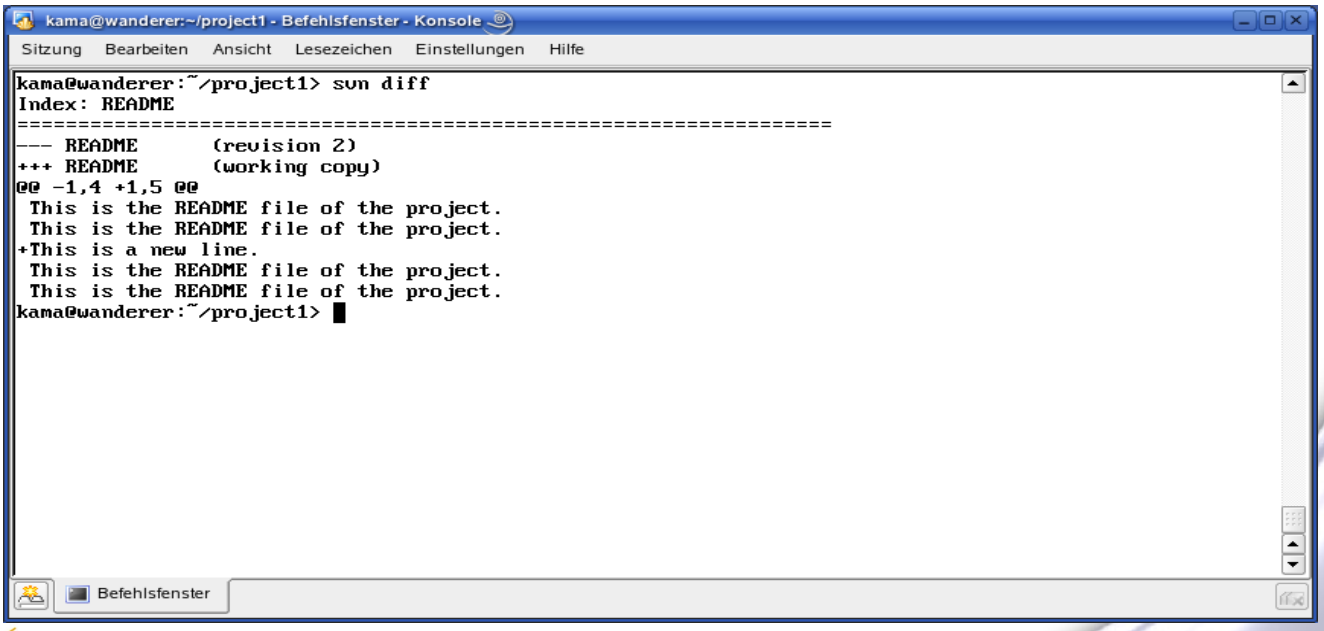
- A la dernière étape Sally doit mettre à jour sa copie locale.



- Denouveau les deux copies sont identiques.

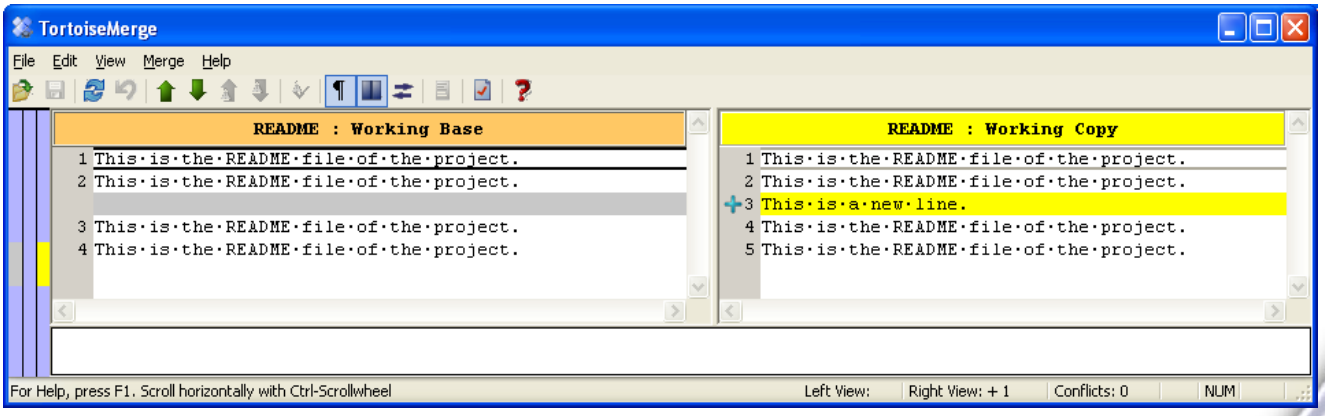
Fonctionnement de subversion

- Analyser les modifications avec « **`svn diff`** »:



Fonctionnement de subversion

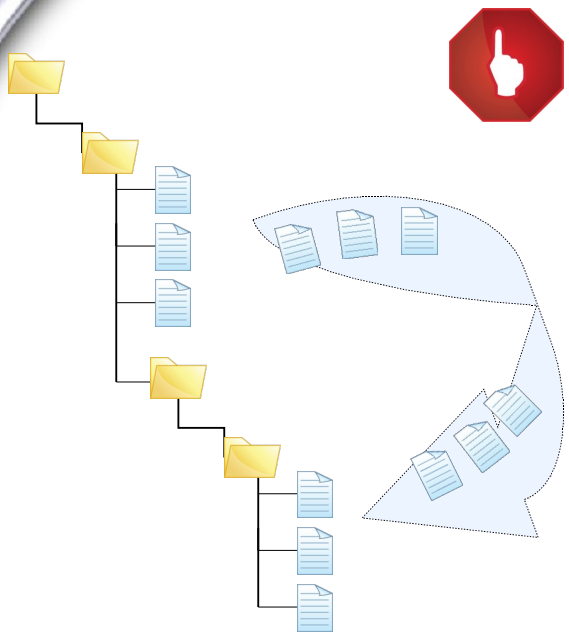
- Analyser les modifications avec TortoiseSVN:
  - Context Menu -> Diff



Fonctionnement de subversion

- Les bases:
  - Suppression, ajout, déplacement d'un fichier ou d'un répertoire:
  - Avec les commandes svn:
    - `svn add test.txt`
    - `svn mkdir tests`
    - `svn mv test.txt tests/test.txt`
    - `svn rm tests/test.txt`
    - `svn commit -m "test"`
  - Il faut faire un « commit » pour valider les modification locales, mais si on utilise une URL, les modifications sont immédiates.

Erreurs à éviter



ATTENTION

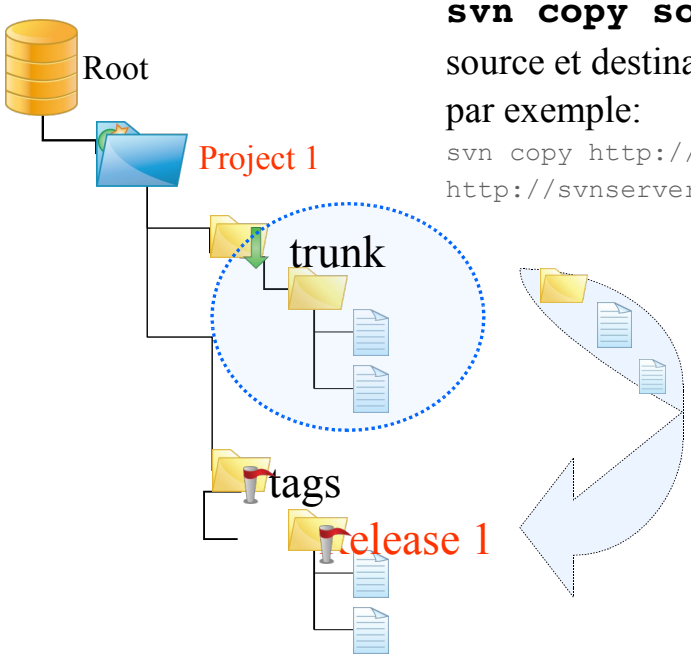
- Il ne faut jamais suppimer, déplacer, copier, ou ajouter des répertoires sans utilser les commandes svn:
  - svn move
  - svn delete
  - svn copy
  - svn mkdir
- Sinon la copie locale sera incohérente et on rsique de perdre des données.

Pourquoi utiliser des tags ?

- Pourquoi a-t-on besoin de tags* ?
  - Pour marquer l'état du projet à la sortie d'un produit (release).
  - Pour garder une image du projet a un instant donné du développement.
- Exemple de noms de release:
  - Release 1.0.0, Release 2.3.1, PRODUCT 1.0.0RC1 etc.
- Le nom d'un tag doit être unique et appliqué à tous les éléments composant un produit. Il sera utilisé pour obtenir le projet dans le même état dans le futur.

* tag = label = étiquette

Création de tag



`svn copy source destination -m ".."`

source et destination peuvent être des URLs.

par exemple:

```
svn copy http://svnserver/calc/trunk
http://svnserver/calc/tags/RELEASE-1.0.0
```

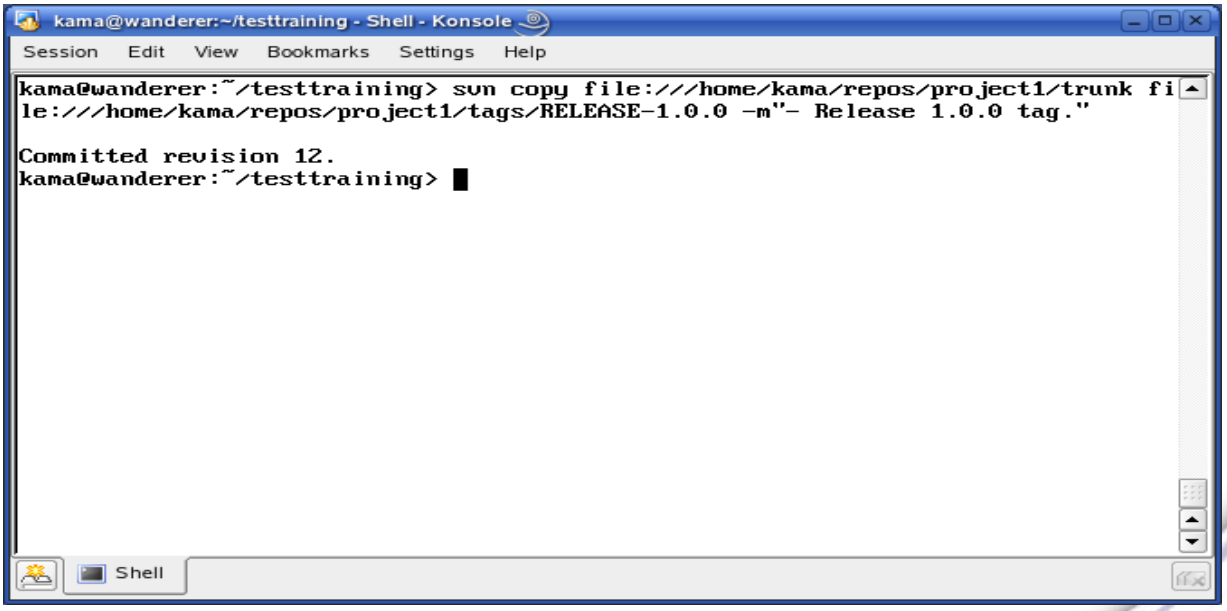
Pour créer un tag il suffit d'utiliser copy...

...de toute façon il ya de numéro de révision ...

Si le nom comporte des espaces, il faut utiliser des doubles quotes.

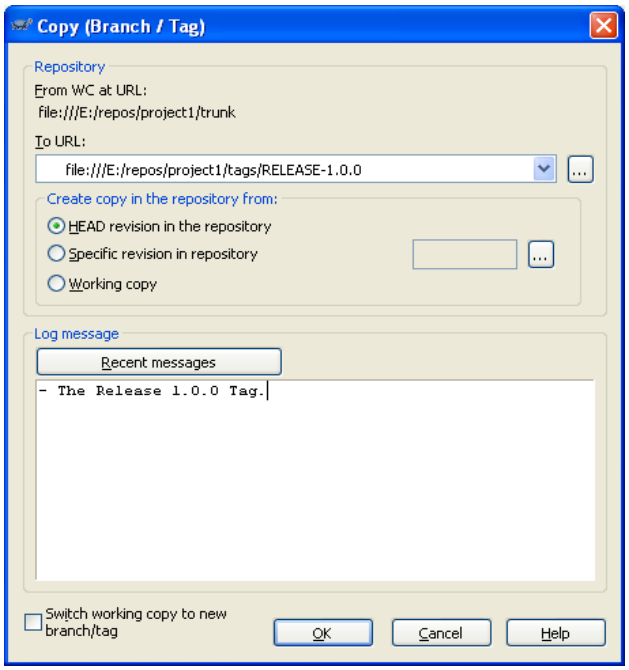
Création de tag

- Creation d'un tag en console:



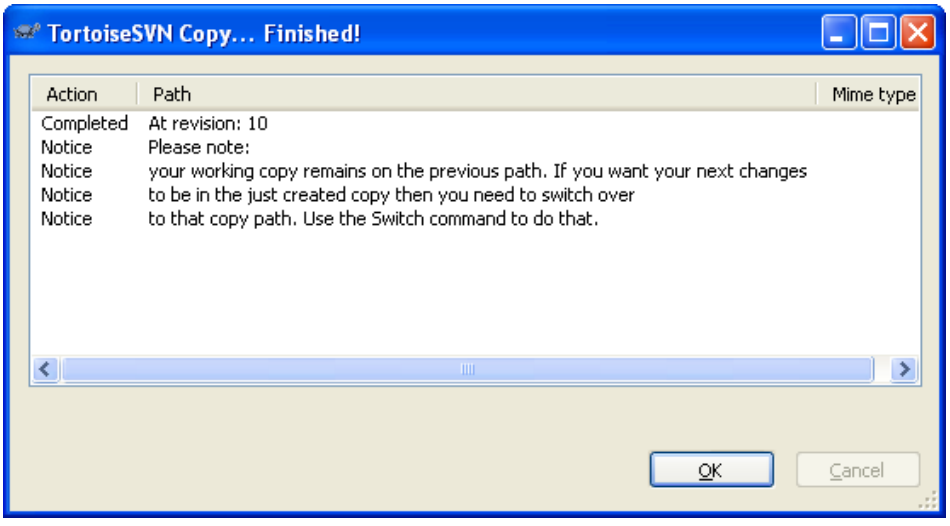
Création de tag

- Creation d'un tag avec tortoiseSVN:
  - Context Menu -> Branch/Tags (Il faut être dans sa copie de travail):



Création de tag

- Fin de la création du tag avec tortoiseSVN:



On peut aussi utiliser le Repository-Browser et le context menu -> Copy to

Gestion des branches

Un exemple pour comprendre l'intérêt des branches

- Une équipe développe le logiciel « Lambda » (projet ambitieux en plusieurs étapes).
- Démarrage dans le répertoire « trunk ».
  - Arrivé à l'étape N°1 ==> Première version publique
  - Création du tag « release-0.1 »
- Travail continue dans le « trunk » vers l'étape N°2
  - Utilisateurs de la release-0.1 trouve un bogue !
  - la version 0.2 n'est pas prête...
  - la version 0.1 est dépassée.
- Comment corriger le bogue sans perturber le développement vers la sortie de la version N°2 ?

Gestion des branches

Un exemple pour comprendre l'intérêt des branches

- Pour corriger le bogue sans perturber le développement vers la sortie de la version N°2:
  - Création d'une branche « bugfix-0.1 »
- Correction du bogue dans la branche « bugfix-0.1 ».
  - Publication de la version 0.1 corrigée
  - Création du tag « release-0.1.1 »
- Travail continue dans le « trunk » vers l'étape N°2
  - Il faudrait aussi corriger le bogue dans cette version !
  - Fusion de la branche « bugfix-0.1 » dans le « trunk ».
- Travail continue dans le « trunk » jusqu'à l'étape N°2
  - Création du tag « release-0.2 »

Gestion des branches

- Aperçu du projet avec KDESVN:

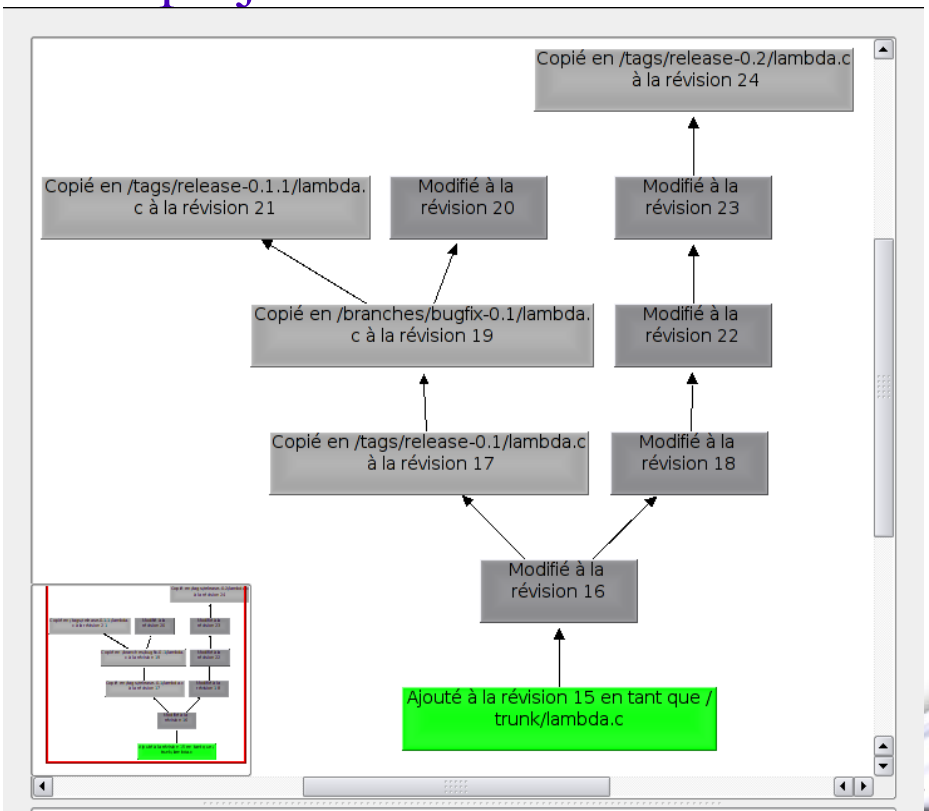
Fichier Signets Subversion Configuration Aide

Nom	État	Révision du dernier changement	Dernier auteur	Date du dernier changemer
sandbox		23	pbo	2007-10-07 21:13
branches		20	pbo	2007-10-07 20:49
bugfix-0.1		20	pbo	2007-10-07 20:49
lambda.c		20	pbo	2007-10-07 20:49
tags		21	pbo	2007-10-07 20:51
release-0.1		17	pbo	2007-10-07 20:44
lambda.c		17	pbo	2007-10-07 20:44
release-0.1.1		21	pbo	2007-10-07 20:51
lambda.c		21	pbo	2007-10-07 20:51
release-0.2		24	pbo	2007-10-07 21:15
lambda.c		24	pbo	2007-10-07 21:15
trunk		23	pbo	2007-10-07 21:13
lambda.c		23	pbo	2007-10-07 21:13
README		11	pbo	2007-08-31 23:54

Vérification de mises à jour démarré en tache de fond
Vérification de mises à jour terminé
Terminé

Gestion des branches

- Arbre du projet dans KDESVN:



Gestion des branches

ATTENTION:

- D'un point de vue technique les branches et les tags sont identiques. Elles sont des copies d'une branche du dépôt.
  MAIS:
  - (L'intention d'un tag) Un tag est fait pour être utiliser en lecture seule alors qu'une branche est faîte pour continuer le développement (code temporaire, bug-fixing, release candidate etc.).
  - Techniquement il est possible de continuer le développement dans un tag... Mais on ne devrait pas le faire.
  - La différence entre un tagg et une branche est une convention dans subversion.
  - Il n'y a pas de moyen de vérifier qu'une branche a déjà été fusionée dans une autre (merge tracking)... Donc il faut commenter les fusions de branches.

Fonctionnement de subversion

- Utilisation avancée:
  - Substitution de mot-clés.
  - C'est une fonction très pratique qui permet de mettre à jour automatiquement certaines variables dans les fichiers sources. Par exemple pour activer la substitution de tous les mots clés, on exécute:

```
svn propset svn:keywords "Rev URL Date Author
  Id" README

Propriété 'svn:keywords' définie sur 'README'

svn ci -m "Mise en place de la substitution
  des mots clés"
```

Fonctionnement de subversion

- Substitution de mot-clés:

```
 svn cat -r12 README
          Fichier explicatif
================================================
INFORMATIONS:
$Rev$
$URL$
$Date$
$Author$
$Id$
================================================
Pour compiler le projet
Taper dans le répertoire des sources du projet:
    ./configure
    make
    make install
```

Fonctionnement de subversion

- Substitution de mot-clés:

```
 svn cat README
          Fichier explicatif
================================================
INFORMATIONS:
$Rev: 13 $
$URL: file:///tmp/depot-svn/trunk/README $
$Date: 2007-10-06 11:06:18 +0200 (sam, 06 oct 2007) $
$Author: pbo $
$Id: README 13 2007-10-06 09:06:18Z pbo $
================================================
Pour compiler le projet
Taper dans le répertoire des sources du projet:
    ./configure
    make
    make install
```

Quelques conseils...

- svn laisse une grande liberté au développeur, si on respecte certaines règles:
  - Quand « mettre à jour » (commit) ?
    - D'autant plus fréquemment qu'un projet comporte peu de modules mais beaucoup de participants...
  - Comment se préserver des collisions ?
    - Vérifier qu'on dispose bien d'une révision « à jour » avant de se lancer dans une intervention.
    - User et abuser de la commande status: si c'est Up to date, alors tout va bien. Si svn indique Needs Checkout... inutile d'hésiter !
  - Ne pas intervenir sur la forme des sources, respecter plutôt une norme commune...

Trac un complément pour svn

- Trac: wiki et gestionnaire de tickets pour le développement logiciel fournissant une interface pour Subversion avec:
  - une interface WEB simple + documentation Wiki
  - un navigateur de sources basé sur subversion
  - des tickets d'intervention avec notifications mail, RSS, rapports personalisés
  - une feuille de route (roadmap), des objectifs (milestones)
  - un logiciel libre (license BSD)

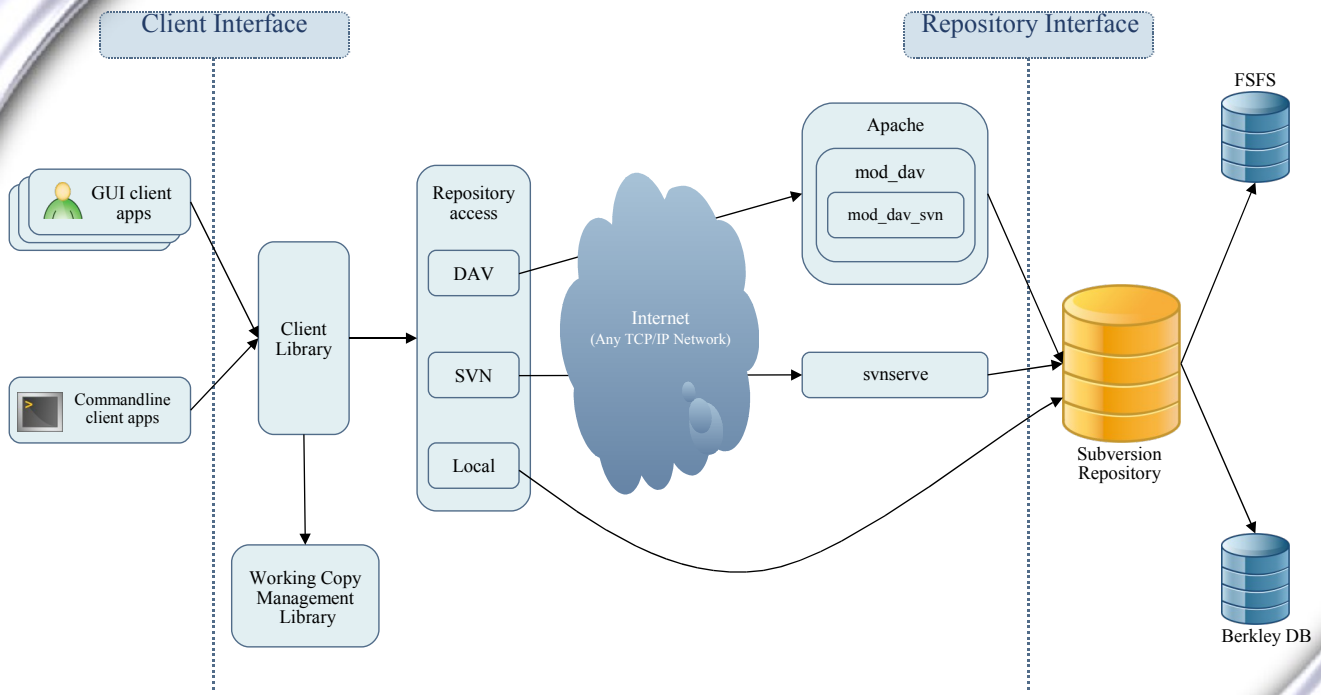
http://trac.edgewall.org/

Introduction à subversion

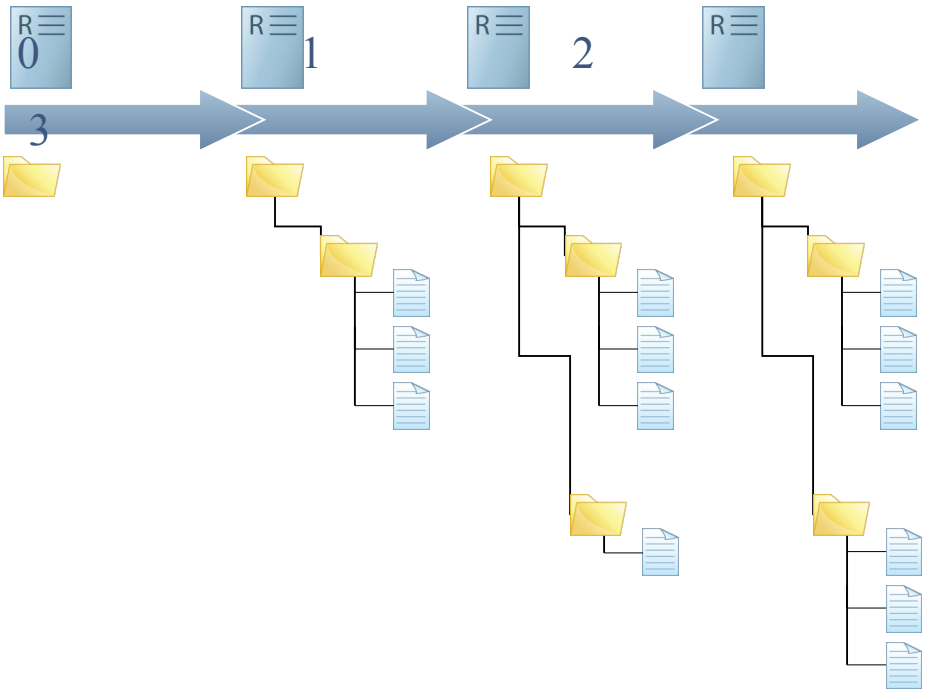
? Questions ?

http://subversion.tigris.org

Subversion – Architecture



Subversion – Architecture
The revision numbers



Properties

List and Set Properties
Special SVN Properties

Properties
List Properties

- You can get a list of properties using the following command:

(1) (2) (3)

```
svn proplist --verbose destination
```

(5) The Subversion “proplist”-command.
(6) Print out much information (verbose).
(7) The file/directory.

Properties
Set Properties

- You can set a property using the following command:

(1) (2) (3) (4)

```
svn propset propertyname value destination
```

(5) The Subversion “propset”-command.
(6) The name of the property e.g. svn:ignore.
(7) The value for the property.
(8) The file/directory for which the property should be set.

Properties
Special SVN Properties

- svn:ignore

  - Tell Subversion which files and subdirectories to ignore
  - Equivalent to CVS's .cvsignore file
  - You can define a special file/directory or patterns defining the files/directories which will be ignored.
  - You can suppress the exclusion if you use the --no-ignore flag for the svn status command.

Properties
Special SVN Properties

- svn:keywords

  - Keyword substitution in files.
    The only available keyword substitutions are:
    - LastChangedDate
    - LastChangedRevision
    - LastChangedBy
    - HeadURL
    - Id
  - Only keywords listed in the svn:keywords property value are replaced in the file

```
$LastChangedDate: 2002-07-22 21:42:37 -0700 (Mon, 22 Jul 2002) $
$LastChangedRevision: 144 $
$LastChangedBy: joe $
$HeadURL: http://svn.collab.net/repos/trunk/README $
$Id: calc.c 148 2002-07-28 21:30:43Z sally $
```

  - Special fixed width keyword substitutions
    - Use $Author:: $ instead of $Author$

Properties
Special SVN Properties

- Example for setting properties on CLI:

```
svn propset   svn:keywords      "Id"        file1.txt
```
(1) (2) (3) (4)

(3) The Subversion “propset”-command.
(4) The name of the property.
(5) The value for the property.
(6) The file/directory for which the property should be set.

Properties
Special SVN Properties

WARNING svn:keywords:

- The width of the field is measured in bytes, a potential for corruption of multi-byte values exists.
- This can happen if you have a name consisting of UTF-8 characters which are expanded during the check out but truncated based on the fixed width of the field.

Properties
Special SVN Properties

- svn:executable
  - For files only, sets the executable bit
  - Useful for scripts and programs
  - The executable bit is set when the property is applied to the file, not when the commit is performed
  - The executable bit is removed when this property is removed from the file.
  - The property value does not matter, any value will do.

Properties
Special SVN Properties

- svn:mime-type
  - SVN assumes a text file if svn:mime-type is not set or is set and matches “text/*”, otherwise it is a binary file.
  - If a file is a text file SVN can use diff and patch to update changes from the repository.
  - Useful for setting the MIME type of a file for web browsing.

Properties
Special SVN Properties

- svn:externals
  - If you use a library (source form) as part of your development which is used in more than one project, you can make a kind of a link to the repository for that library. This avoids redundant copys of the library's source code.

- Advantages:
  - Just a single point of development for 3rd party library or libraries at all.

- Disadvantages:
  - No automatic commits etc. in directories which are created by svn:externals.

Properties
Special SVN Properties

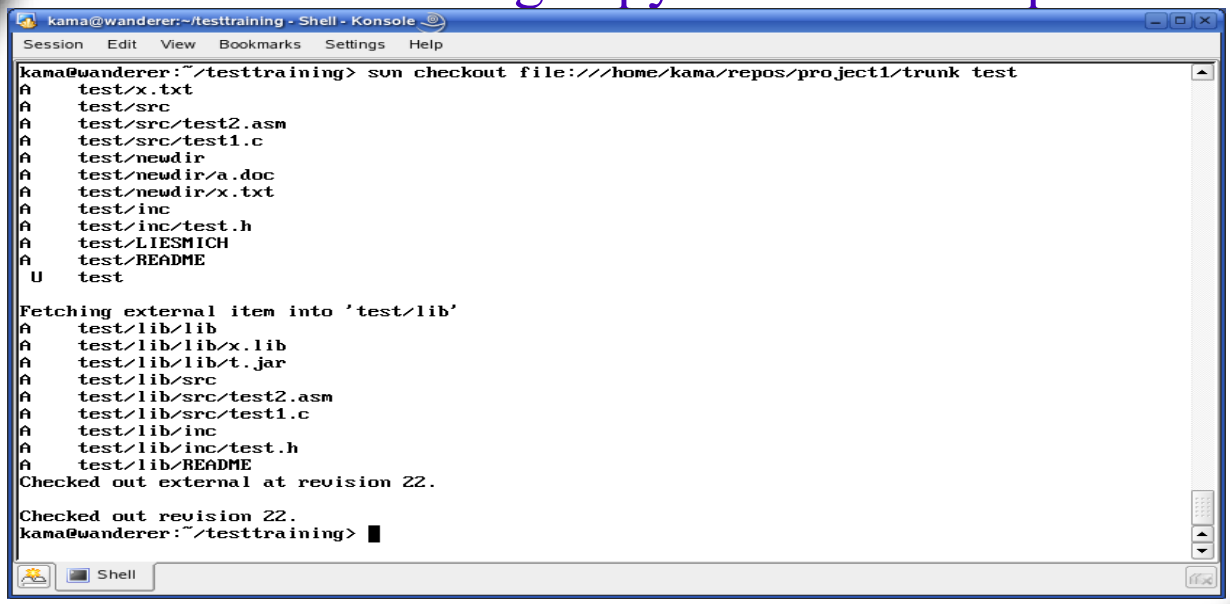
- Set svn:externals properties via CLI:

```
kama@wanderer:/...testtraining/harry - Shell - Konsole
Session  Edit  View  Bookmarks  Settings  Help
kama@wanderer:~/testtraining/harry> svn ps svn:externals "lib file:///home/kama/repos/lib/trunk" .
property 'svn:externals' set on '.'
kama@wanderer:~/testtraining/harry> svn pl -v .
Properties on '.':
  svn:externals : lib file:///home/kama/repos/lib/trunk

kama@wanderer:~/testtraining/harry>
Shell
```

Properties
Special SVN Properties

- Checkout a working copy with external parts on



Properties
Special SVN Properties

- svn:eol-style
  - Native
    - Convert all CR, CRLF and LF's to the system's native eol
  - CR
    - Convert all CRLF and LF's to CR's
  - CRLF
    - Convert all CR and LF's to CRLF's
  - LF
    - Convert all CR and CRLF's to LF's

- Property not set, this is the default when a file is added.
  Do not change end of lines upon check in or update.

Do's and Don't's

Do's and Dont's

- Do **not** commit just a single file if your change applies to more than one file.
- Commit the complete directory tree beginning at the root of your working copy.
- Make sure your change reflects a single purpose: fixing a specific bug, the addition of a new feature, or some specific task.

But why?

Do's and Dont's

- Log messages should describe the change from a technical point of view.
- Put in a reference to Ticket/Issue/Bug Id.
- Don't write log messages like:
  - Removed line 4
  - Inserted line 10 in file etc.
- The above is the job of Subversion not yours ;-)
- Often Log messages have to follow policies given by the company. Pay attention to yours.

Subversion Configuration

Auto Properties
Global Ignores

Subversion Configuration

- Under Linux you will find the configuration under:
  - /home/**Username**/.subversion/config
- Under Windows you will find the configuration under:
  - ? figure out
  - In the config file you can change the behaviour of parts of Subversion.

Subversion Configuration auto-properties

- Auto-properties can be useful if you like to set Word/Excel files to read-only.
  - You have to turn on enable-auto-props which means set it to „yes“.
  - After that you can set svn:needs-lock for particular file types.

Subversion Configuration auto-properties

```
[auto-props]
### The format of the entries is:
###   file-name-pattern = propname[=value][;propname[=value]...]
### The file-name-pattern can contain wildcards (such as '*' and
### '?').  All entries which match will be applied to the file.
### Note that auto-props functionality must be enabled, which
### is typically done by setting the 'enable-auto-props' option.
# *.c = svn:eol-style=native
# *.cpp = svn:eol-style=native
# *.h = svn:eol-style=native
# *.dsp = svn:eol-style=CRLF
# *.dsw = svn:eol-style=CRLF
# *.sh = svn:eol-style=native;svn:executable
# *.txt = svn:eol-style=native
# *.png = svn:mime-type=image/png
# *.jpg = svn:mime-type=image/jpeg
# Makefile = svn:eol-style=native
```

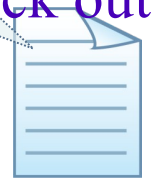
Subversion Configuration global-ignore

You can put whatever file extension into global-ignores to globally ignore files of a particular type.

```
[miscellany]
### Set global-ignores to a set of whitespace-delimited globs
### which Subversion will ignore in its 'status' output.
# global-ignores = *.o *.lo *.la #*# .*.rej *.rej .*~ *~ .#* .DS_Store
```

Subversion Configuration commit-times

- Sometimes you need to have the changed time to be set on the commit-time instead of the created local time. This is often used in relationship with Make-files.
- You can simply uncomment the line with “use-commit-times = yes“.
- After that every check out will set the time stamp to commit time and not to the check out time.



```
### Set use-commit-times to make check out/update/switch/revert
### put last-committed timestamps on every file touched.
# use-commit-times = yes
```


Appendix

Administration Permissions
Administration Hook Scripts
Backup
Cleaning

Appendix
Administration - Permission

- Configure permissions in Apache

```
<Location /repos>
        DAV svn
        SVNParentPath /usr/local/svn
        # our access control policy
        AuthzSVNAccessFile /path/to/access/file
        # only authenticated users may access the repository
        Require valid-user
        # how to authenticate a user
        AuthType Basic
        AuthName "Subversion repository"
        AuthUserFile /path/to/users/file
</Location>
```

Appendix
Administration - Permission

- Users file:

```
name1:password
name2.password
.....
```

- The password is created using the htpasswd command
  of the Apache distribution.

Appendix
Administration - Permission

- By default nobody has any access to the repository at all. That means that if you're starting with an empty file, you'll probably want to give at least read permission to all users at the root of the repository. You can do this by using the asterisk variable (*), which means “all users”:

```
[/]
* = r
```

IMERIR 2007
10
SUBVERSION

Appendix
Administration - Permission

- Access file:

(1) (2)

```
[calc:/branches/calc/bug-142]
harry = rw
sally = r
```

(6) The repository the permission belongs to, if you have used „SVNParentPath“. If don't use it just remove this and the colon as well.

(7) The path in the repository.

Appendix
Administration - Permission

- You should be aware that permissions are inherited from parent to child directories.

```
[calc:/branches/calc/bug-142]
harry = rw
sally = r
# give sally write access only to the 'testing' subdir
[calc:/branches/calc/bug-142/testing]
sally = rw
```

IMERIR 2007
10

Appendix
Administration - Permission

- E.g. you would like to prevent Harry from accessing everything in „/branches/calc/bug-142/secret“:

```
[calc:/branches/calc/bug-142]
harry = rw
sally = r
# give sally write access only to the 'testing' subdir
[calc:/branches/calc/bug-142/testing]
sally = rw
[calc:/branches/calc/bug-142/secret]
harry =
```

PBo - bourdin@imerir.com

A Appendix Administration - Permission

- You can organize users in groups which will simplify the life of the Administrator.

```
[groups]
calc-developers = harry, sally, joe
paint-developers = frank, sally, jane
everyone = @calc-developers, @paint-developers
[calc:/projects/calc]
@calc-developers = rw
[paint:/projects/paint]
@paint-developers = rw
jane = r
```

IMERIR 2007
10

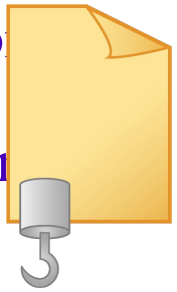
Appendix Administration - Permission

- The thing to remember is that the most specific path always matches first. The mod_authz_svn module tries to match the path itself, and then the parent of the path, then the parent of that and so on. The net effect is that mentioning a specific path in the access file will always override any permissions inherited from parent directories.

PBo - bourdin@imerir.com

Appendix
Administration – Hook Scripts

- Hook Scripts are called in Subversion fo different
  events. Basically there are three differe events:
  - Commit
  - Locking
  - Revision property changes



IMERIR 2007
10

Appendix
Administration – Hook Scripts

- The Commit event can be divided into the following three particular events:
  - start-commit
  - pre-commit
  - post-commit

PBo - bourdin@imerir.com

Appendix
Administration – Hook Scripts

- The ***start-commit*** Event:
  - This is run before the commit transaction is even created.
  - Usually used to decide if the user has commit privileges at all.
  - It has two arguments, the path to the repository and the user name.
  - Return non zero means to abort the operation.
  - stderr will be marshalled back to the client.

Appendix
Administration – Hook Scripts

- The ***pre-commit*** Event:
  - This is run when the transaction is complete but before it is committed.
  - Usually used to check the log message (templates; ticket id etc.)
  - It has two arguments, the path to the repository and the transaction name.
  - Return non zero means to abort the operation.
  - stderr will be marshalled back to the client.

Appendix
Administration – Hook Scripts

- The ***post-commit*** Event:
  - This is run after the transaction is committed and a new revision is created.
  - Usually used to send commit mails, make backups based on check ins, interaction with collaboration tools like Polarion, trac etc.
  - It has two arguments, the path to the repository and the new revision number.
  - The exit code will be ignored.

IMERIR 2007
11

Appendix
Administration – Hook Scripts

- The Lock event can be divided into the following three particular events as well:
  - pre-lock
  - post-lock
  - pre-unlock

PBo - bourdin@imerir.com

Appendix
Administration – Hook Scripts

- The ***pre-lock*** Event:
  - This hook runs whenever someone attempts to lock a file.
  - Usually used to create a complex lock/unlock/steal-lock policy.
  - It has three arguments, the path to the repository, the path being locked, and the user attempting to perform the lock
  - Return non zero means to abort the operation and stderr will be marshalled back to the client.

IMERIR 2007
11

Appendix
Administration – Hook Scripts

- The ***post-lock*** Event:
  - This hook runs after a path is locked.
  - Can be used to send email to users etc.
  - The locked path is passed to the hook's stdin, and the hook also receives two arguments: the path to the repository, and the user who performed the lock
  - The exit code will be ignored.

PBo - bourdin@imerir.com

Appendix
Administration – Hook Scripts

- The ***pre-unlock*** Event:
  - This hook runs whenever someone attempts to remove a lock on a file.
  - Used to control breaking locks.
  - Arguments: the path to the repository, the path being unlocked, and the user attempting to remove the lock.
  - Return non zero means to abort the operation and stderr will be marshalled back to the client.

IMERIR 2007
11
SUBVERSION

Appendix
Administration – Hook Scripts

- The Revision property change event can be divided into the following two particular events:
  - If you try to use one of them you have to supply the other one too for paranoia's sake.
    - pre-revprop-change
    - post-revprop-change

PBo - bourdin@imerir.com

Appendix
Administration – Hook Scripts

- The ***pre-revprop-change*** Event:
  - This hook runs just before such a modification is made to the repository.
  - Usually used to allow to change log messages.
  - Arguments: the path to the repository, the revision on which the to-be-modified property exists, the user name and the name of the property itself.
  - Return non zero means to abort the operation and stderr will be marshalled back to the client.

IMERIR 2007
11
SUBVERSION

Appendix
Administration – Hook Scripts

- The ***post-revprop-change*** Event:
  - This hook runs runs after a revision property has been changed.
  - Usually used to send an email with the new values.
  - Arguments: the path to the repository, the revision on which the property exists, the user name and the name of the property itself.
  - Return non zero means to abort the operation and stderr will be marshalled back to the client.

Appendix
Administration – Hook Scripts

- The ***svnlook*** command:
  - „Read-Only“, no changes made to the repository.
  - Usually used in relationship with hook-scripts to extract information about transaction etc.
  - Most of the time, parameters are passed to hook-scripts which can directly be used by **svnlook** (take a look into the hooks directory of your repository).

Appendix
Administration – Backup

- The ***svnadmin*** command is used for administration of the repository (create, backup, restore etc.)
  - You can dump a repository using ***svnadmin dump >file.dump***.
    - Produce a file which you can use to transfer from one machine to another (very large files!).
  - You can make a ***svnadmin hotcopy***
    - Make copy (like cp) to another path.

Appendix
Administration – Backup

- A repository dump can be loaded using the svnadmin load command.
- The dump file has a format which can be parsed in a simple way.
- You can use the --incremental option to reduce the dump file size e.g. for backup or transfer to other sites or maybe in relationship with hook scripts to make a backup every time a check in is performed.

IMERIR 2007
12
SUBVERSION

Appendix
Administration – Cleaning

- A repository might become very large in time.
- What if you really want to delete a revision or wrongly checked in information forever?
  - You can use a combination of svnadmin dump and the svndumpfilter command.
  - Dump the repository content and filter the things you don't like to have and create a new one in which you load the filtered parts into.
  - Be careful about revision numbers etc.

PBo - bourdin@imerir.com