# Applications orientées données (NSY135)

## 15 – Gestion de version

Auteurs: Raphaël Fournier-S'niehotta et Philippe Rigaux
(philippe.rigaux@cnam.fr,fournier@cnam.fr)

Département d'informatique
Conservatoire National des Arts & Métiers, Paris, France

# Plan du cours

# Subversion ?

Subversion est un logiciel de contrôle de versions

- il enregistre tous les états d'une arborescence au fil du temps
- arborescence = structure des dossiers + contenus des fichiers
- il gère donc les différentes versions d'un répertoire
- Subversion a un prédécesseur, CVS, que vous rencontrerez peut-être parfois
- on abrège souvent **S**ub**V**ersio**N** en SVN

Pourquoi Subversion

- dans un projet de développement informatique, il y a de nombreux fichiers, dont le contenu évolue au cours du temps
- SVN permet d'enregistrer tous ces changements pour, par exemple, avoir une trace explicite et exhaustive de tous les changements faits au code source
- ou pouvoir revenir au code tel qu'il était il y a 5 mois.

# Fonctionnement élémentaire

- Sans rentrer dans les détails, SVN réalise des instantanés de l'arborescence à chaque changement
- Imaginez que vous faites un dessin sur une feuille de papier, et qu'à chaque fois que vous ajoutez un trait, vous photocopiez votre feuille et la rangez dans une archive.
- Ensuite, imaginez qu'à la place de photocopier votre dessin, vous le scannez pour le mettre à disposition d'autres dessinateurs dans le monde. Chacun peut travailler simultanément sur la même version.
- Pour éviter que cette archive ne grossisse trop, SVN ne garde pas une copie complète de chaque fichier, mais il est capable de réaliser et d'utiliser des instantanés incrémentaux (on ne garde que ce qui permet de passer d'une version à une autre)

# Révisions

- L'organisation de SVN repose sur la notion de **révisions**
- il s'agit de l'état de l'arborescence à un moment donné dans le temps
- exemple : révision 201 le 9 juin 2015, révision 202 le 16 juin 2015
- la première révision est la révision 0, ou **r0**
- ensuite on incrémente de 1 à chaque fois

# Dépot et copie locale

Un système utilisant Subversion est à diviser en deux parties distinctes :

- le dépôt (**repository** en anglais)
  - côté “serveur”, c’est l’archive dans laquelle sont envoyées les modifications
  - peut se trouver sur un serveur distant, sur un serveur situé dans le réseau local ou même sur votre propre machine
  - Un utilisateur simple n’a pas besoin de savoir ce qui se passe dans le dépôt, ce sont les mécanismes internes du processus.
- la copie locale (**lcoal copy**)
  - Une copie locale est une copie de l’état du dépôt à un moment précis, située sur l’ordinateur d’un des utilisateurs
  - chacun des développeurs a une copie locale et travaille dessus

# Actions de base

- Créer un dépot
- Checkout - créer copie locale
- Update
- Commit
- Supprimer, copier, déplacer

# Créer le dépôt

```
$ mkdir svnrepos
$ cd svnrepos
$ svnadmin create repo1

$ ls repo1
  conf  dav  db  format  hooks  locks  README.txt
```

- On obtient un dépôt vide
- On va maintenant le remplir

# Checkout

- le **checkout** permet de créer une copie locale
- Habituellement, le checkout est à faire une seule fois (par développeur)
- Un dépôt est appelé par son adresse sous forme d'URL
  - `file:///chemin/vers/le/dépot`
  - `http://server.web.com/repository`
  - `svn://server.svn.com/repository`

```
$ mkdir svnlocal; cd svnlocal
$ svn checkout file:///home/<USER>/svnrepos/repo1 repo1local
  Révision 0 extraite.
$ ls repo1local/ -a
  .  ..  .svn
```

# Update

- L'update sert à mettre à jour une copie locale, sans la recréer
- Quand on utilise cette commande, SVN compare la copie locale avec l'état du dépôt
- si celui-ci est à une révision supérieure que celle de la copie locale, il télécharge les différences entre les deux, pour synchroniser

```
    $ svn update repo1local
ou  $ cd repo1local; svn update
```

# Update

Exemple de cas

- deux développeurs sur un dépôt
- l'un part en vacances, laissant sa copie en r4, à jour avec le dépôt
- pendant qu'il n'est pas là, son collègue avance jusqu'à la r7
- à son retour, le premier lance un **svn update** pour revenir au niveau de son collègue et reprendre le travail

Il est fréquent que l'update ne serve à rien : la copie est à jour, il n'y a rien à télécharger. C'est une mesure de précaution pour éviter de travailler sur une version ancienne.

# Commit

- commit est l'action de valider des modifications et de les envoyer au dépôt
- il y a alors création d'une révision

```
[repo1local] $ nano file.txt
               <taper du texte>
               <ctrl+x>, <o>, <valider>

[repo1local] $ svn add file.txt
[repo1local] $ svn commit
               <taper le message de commit>

               Ajout          file.txt
               Transmission des données .
               Révision 1 propagée.
```

# Supprimer, copier, déplacer

- pour déplacer, supprimer ou copier des fichiers ou dossiers, il ne suffit plus de d'utiliser les fonctions fournies avec son système d'exploitation et de faire son commit,
- il faut utiliser les trois sous-programmes **move, delete et copy**.

```
$ svn move FICHIER DESTINATION
$ svn copy FICHIER DESTINATION
$ svn delete FICHIER
```

  - les fichiers sont dans une sorte de file d'attente où vont tous les fichiers "modifiés" en attendant le commit. Rappelez-vous qu'aucune modification n'est appliquée tant que le commit n'est pas lancé.

# Exercice

- Entraînez-vous à simuler le travail de deux développeurs, utilisant le même dépôt
- Ils effectuent des opérations de checkout, modification, commit, update
- Pour commencer, lancez un nouveau terminal (ce sera celui du deuxième développeur), créez un nouveau dossier et créez une autre copie locale.
- Celle-ci sera en r1 dès sa création.
- Sur la copie 1, modifiez file.txt, commitez, puis updatez la copie2.
- Avec un peu de chance, vous allez tomber sur un problème, que nous allons résoudre ensuite.

# Conflits

- Alice et Bob travaillent sur le même dépot
- Ils ont un checkout et sont tous les deux en r1
- ils commencent tous les deux à éditer le fichier
- Alice fait un commit avec ses modifications, le fichier est uploadé et la révision 2 est créée ;
- Bob, une fois ses propres modifications terminées, va vouloir en faire un commit
- sauf que là un message d'erreur lui dit qu'il ne peut pas le faire
- en effet, au moment de sa tentative de commit, sa copie locale n'est plus à jour (r1) avec le dépôt (r2)
- C'est une mesure de sécurité, pour ne pas que Bob détruise le travail d'Alice

# Fusion (Merge)

- Pour ne pas que le travail de Bob soit perdu, il va falloir résoudre le conflit
- pour cela, SVN va proposer une fusion (merge) des deux versions du fichier à problème
- soit par merge automatique (si possible), soit manuel

# Exemple

```
Bonjour !
```

```
# Je suis Alice et je commente ici

Bonjour !
```

```
Bonjour !

# Je suis Bob et je commente là
```

```
# Je suis Alice et je commente ici

Bonjour !

# Je suis Bob et je commente là
```

# Fusion manuelle

- Il arrive parfois (même plutôt souvent) que les modifications effectuées par deux utilisateurs se recoupent
- Subversion ne peut pas prendre le risque d'effectuer lui-même le merging
- C'est donc à l'utilisateur responsable du conflit de le corriger lui-même
- Exemple. Fichier sandwiches.txt en r4:

```
4 tranches de jambon
3 tomates
9 cornichons
```

# Fusion manuelle

- Alice ajoute une tranche de jambon (5) et commit
- Bob tente d'en ajouter deux de plus (il veut mettre 6 et non 4)
- Chez Bob, la copie locale sera :

```
Sending        sandwiches.txt
svn: Commit failed (details follow):
svn: Out of date: 'sandwiches.txt' in transaction '5-1'

$ ls
sandwiches.txt  sandwiches.txt.mine  sandwiches.txt.r4  sandwiches.txt.r5
```

- sandwiches.txt.mine, c'est celui qu'il a proposé, avec 6 tranches de jambon
- sandwiches.txt.r4 contient le fichier à la révision précédente, avant changement
- sandwiches.txt.r5 contient le fichier avec les modifications d'Alice

# Fusion manuelle

sandwiches.txt

```
<<<<<<< .mine
6 tranches de jambon
=======
5 tranches de jambon
>>>>>>> .r5
3 tomates
9 cornichons
```

- Tout ce qui est en dehors des signes est ce qui n'a pas été modifié
- Ici, il faut décider quel est le bon nombre à conserver (4,5,6 ou 7)
- On retire donc les marques en laissant la ligne, avec le nombre mis à jour.
- dans les projets de code, ça peut être bien plus compliqué. . .

# Fusion manuelle: résolution

sandwiches.txt

```
6 tranches de jambon
3 tomates
9 cornichons
```

```
svn resolved sandwiches.txt
svn commit -m"Conflit résolu ! On garde 6 parce que B s'est trompé"
```

# Lock

- Éviter les conflits, c'est donc important
- Pour cela, faire des mises à jour régulières, avec update
- Bien sûr, ça ne suffira pas toujours et il faut essayer d'empêcher qu'ils surviennent
- appeler les collègues ou envoyer un mail n'est pas une solution, on en revient à s'envoyer les différentes versions du code par mail
- On demande à SVN de verrouiller le fichier pour notre usage

```
$ svn lock sandwiches.txt
   'sandwiches.txt' verrouillé par l'utilisateur 'raph'.

 <travail>

$ svn unlock sandwiches.txt
```

# Contrôler ses commits : Status

- **status** permet de fouiller la copie locale pour donner des informations sur ses éléments constitutifs (fichiers et dossiers)

```
svn status
M       sandwiches.txt
```

- Le M signifie “modifié”, on sait donc qu’au prochain commit les modifications apportées au fichier sandwiches.txt seront prises en compte. La lettre changera en fonction du type d’actions réalisées sur le fichier ou dossier. Vous pouvez trouver plus d’informations à ce sujet grâce à la commande **svn help status**.

# Contrôle de commits : diff

- donne des informations sur les modifications effectuées sur la copie locale, mais cette fois-ci à une échelle plus petite
- pour chaque fichier modifié, un détail des ajouts, retraits et modifications apportés, à la ligne près.
- Si j'ajoute une ligne au fichier sandwiches.txt et que je lance svn diff sandwiches.txt, j'aurai cette sortie :

```
Index: sandwiches.txt
===================================================================
--- sandwiches.txt      (revision 6)
+++ sandwiches.txt      (working copy)
@@ -1,3 +1,4 @@
 7 tranches de jambon
 3 tomates
 9 cornichons
+1 pot de mayonnaise
```

- il y a eu une ligne ajoutée (le signe + en début de ligne) disant 1 pot de mayonnaise
- un signe - indiquerait un retrait de ligne
- puisque l'unité est la ligne, la correction d'un caractère provoquera la suppression de l'ancienne ligne et l'ajout de la ligne corrigée

# Historique

## Remonter dans le temps

```
svn update -r 3
U    foobar
Checked out revision 3.
```

- Les fichier marqués U sont les fichiers qui ont été modifiés par l'update. Ensuite, vous pouvez parcourir, utiliser et modifier les fichiers de votre dépôt comme d'habitude.
- À la fin de cette commande, vous pouvez très bien spécifier le nom d'un ou plusieurs fichiers, de sorte de ne mettre à jour que ceux-ci, et de laisser les autres à jour.
- Attention, les fichiers que vous modifierez ne seront pas à jour, vous ne pourrez donc pas les envoyer sur le serveur sans action de résolution de conflit

# Historique

- Il arrive souvent qu'on supprime un fichier du dépôt, puis qu'on veuille s'en resservir par la suite
- il faut identifier le moment de la suppression, pour cela on utilisera la commande **svn log -v**

```
------------------------------------------------------------------------
r3 | raph | 2015-06-09 17:06:02 +0200 (mar. 09 juin 2015) | 1 ligne
Chemins modifiésă:
   D /foobar.txt

retrait de foobar
------------------------------------------------------------------------
r2 | raph | 2015-06-09 17:05:34 +0200 (mar. 09 juin 2015) | 1 ligne
Chemins modifiésă:
   A /foobar.txt

ajout de foobar
------------------------------------------------------------------------
```

- le fichier a été supprimé à la révision 3, ce qui signifie qu'il existait à la révision 2
- on peut le récupérer

```
svn copy -r2 file:///home/raph/svntest/svnrepos/repo1/foobar.txt ./foobar-r2
A         foobar
```

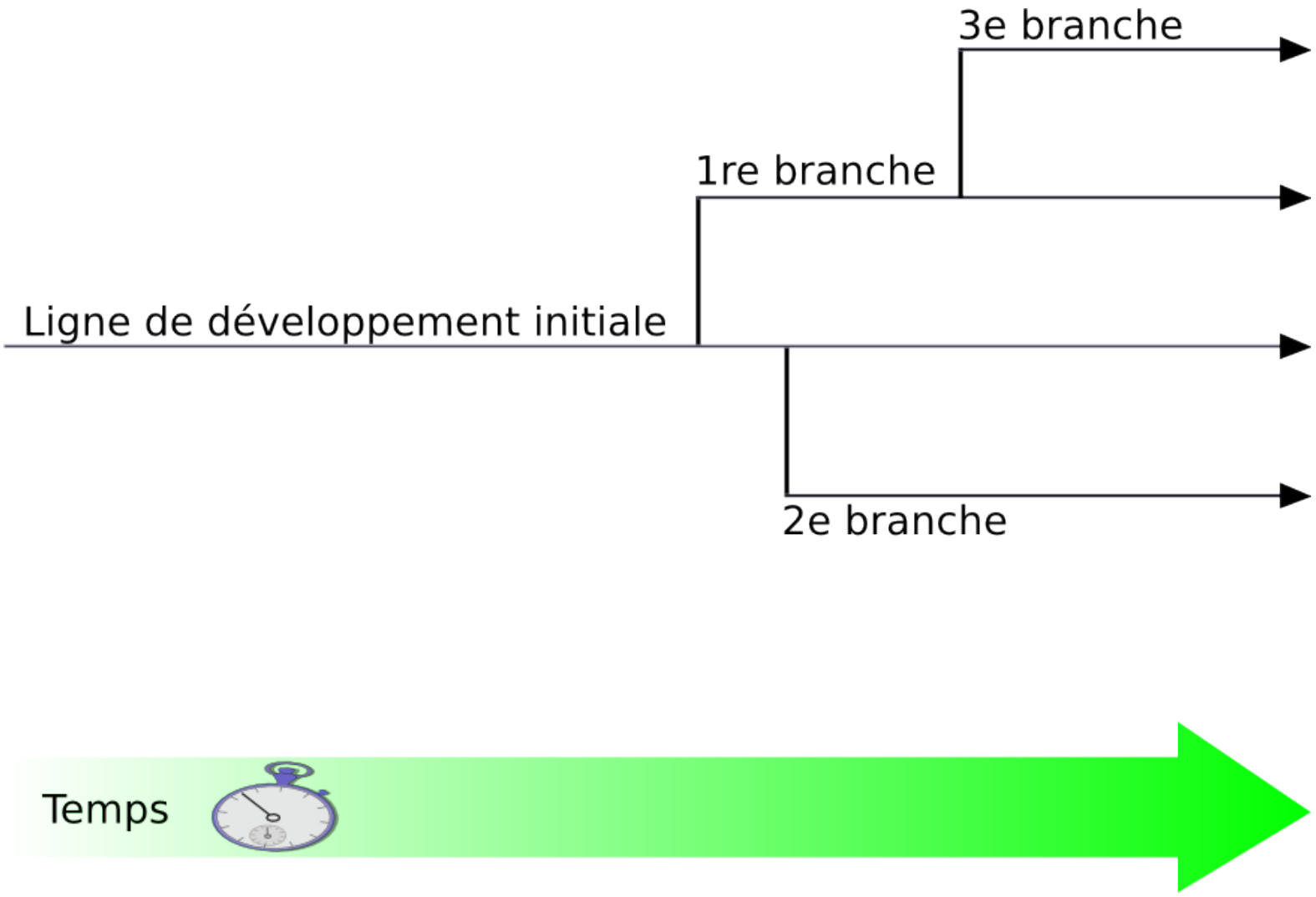
# Cycle de travail

- Mettre à jour votre copie de travail
- Apporter des modifications à votre copie de travail
- Examiner les changements apportés
  - Avoir une vue d'ensemble des changements effectués
  - Voir en détail les modifications que vous avez effectuées
- Annuler des changements sur la copie de travail
- Résoudre les conflits (fusionner des modifications)
  - Voir les lignes en conflit de façon interactive
  - Résoudre les conflits en mode interactif
  - Remettre à plus tard la résolution d'un conflit
  - Résoudre les conflits à la main
  - Abandonner vos modifications au profit de la révision la plus récente
  - Revenir en arrière : utiliser svn revert
- Propager vos modifications

# Les branches : scénario

- Vous travaillez sur un site Web. Vous souhaitez mettre en production votre travail, en en gardant une trace.
- Pendant que cette version est en ligne, vous souhaitez continuer à développer des fonctionnalités
- Vous pouvez aussi avoir à créer une version du site légèrement différente, pour un usage particulier
- Pour éviter le travail de dupliquer les mêmes modifications dans différentes copies (prod, développement), on a introduit le concept de branche.
- c'est une ligne de développement qui existe indépendamment d'une autre ligne, mais partage cependant une histoire commune avec elle, si vous remontez suffisamment loin en arrière dans le temps

# Ligne de vie des branches



# Création de branches

```
 $ svn copy file:///home/raph/svntest/svnrepos/repo1/trunk \
  file:///home/raph/svntest/svnrepos/repo1/branches/dev \
  -m "Création d'une branche privée."

  Puis on extrait cette branche, pour travailler dessus
$ svn checkout http://svn.exemple.com/depot/calc/branches/ma-branche-calc
```

- On ne va pas rentrer dans les détails, mais il est ensuite possible de fusionner des branches (pour réincorporer par exemple les modifications faites dans la partie développement dans la version de production)
- détruire des branches, etc.

# Encore plus loin

- Il est ensuite possible de créer des **étiquettes**, ou **tags**
- ce ne sont que des révisions particulières, auquel on a décidé de donner un nom, pour s'y retrouver plus facilement
- exemple : "version 1.0"
- Personnalisation : il y a de nombreuses possibilités de configuration, pour simplifier la vie/arranger svn comme on le souhaite (langue, alias pour raccourcir les commandes, etc.)
- De nombreux clients existent, notamment pour Eclipse, subclipse

# Références

- Gestion de version avec Subversion, version en ligne en français : `http://svnbook.red-bean.com/`