# Conduite et Gestion de Projet
# Systèmes de gestion de version

*Thierry Hamon*

Bureau H202 - Institut Galilée
Tél. : 33 1.48.38.35.53
Bureau 150 – LIM&BIO – EA 3969
Université Paris 13 - UFR Léonard de Vinci
74, rue Marcel Cachin, F-93017 Bobigny cedex
Tél. : 33 1.48.38.73.07, Fax. : 33 1.48.38.73.55
thierry.hamon@univ-paris13.fr
http://www-limbio.smbh.univ-paris13.fr/membres/hamon/CGP-20132014

# Introduction

Gestion de version :

- Maintenance d'un ensemble de fichiers et de leurs versions
- Utilisation importante dans le développement logiciel (gestion du code source) :
- Nombreuses fonctionnalités nécessaires au développement logiciel :
  - Sauvegarde régulière des modifications
  - Historique des modifications (date, nom du développeur, zone de code modifié)
  - Partage des fichiers entre plusieurs développeurs
  - Développement en parallèle sur le même fichier
  - Définition de distribution
  - Maintenance et correction (définition de *patch*)
- Exemples (les plus connus) : CVS (obsolète), **SVN**, GIT, Mercurial (Hg)

# Système optimiste

- Plusieurs utilisateurs peuvent travailler sur le même fichier en même temps
- le système fait l'hypothèse que les conflits (dus à des modifications) sont rares : les utilisateurs ne travaillent pas sur la même partie du fichier
- le système n'a qu'à fusionner les modifications (au niveau des lignes)
- les conflits sont réglés manuellement par les utilisateurs

Autre solution : réserver le fichier à modifier, de manière à travailler en exclusivité

Mode de fonctionnement optimiste : avantages importants

- un développeur voulant faire modifier un fichier n'est pas pas empêché par quelqu'un qui a réservé ce même fichier
- pas de cassure dans le rythme de développement

# Dépôt / Référentiel

Centralisation des fichiers et des informations associées :

- Stockage dans une base de données
- Accès local et distant partagé par tous les utilisateurs
- Sauvegarde des différentes versions des fichiers
- Sauvegarde des informations associées aux fichiers et au dépôt

Chaque utilisateur travaille sur une copie locale et envoie ses modifications sur le dépôt

# Principe de la gestion de version

- Chaque élément (fichier, répertoire, etc.)
  - est contrôlé par le système de gestion de version
  - se voit associer un numéro de version (révision)
- L'enregistrement d'une modification d'un élément de la copie locale provoque une incrémentation du numéro de révision
- Définition d'une arborescence des versions (organisée suivant un tronc et des branches)

# Principe de numérotation des versions

- format : `m.n.p`
  voir par exemple la numérotation des noyaux Linux
- `m` : numéro de révision majeure
  Incrémentation lors d'ajout de fonctionnalités incompatibles avec la version précédente
- `n` : numéro de révision mineure
  Incrémentation lors d'ajout de fonctionnalités compatibles avec la version courante (maintenance évolutive)
  La parité peut être utilisée pour indiquer la stabilité :
  - pair : version stable
  - impair : version de développement
- `p` : numéro de patch
  Incrémentation lors de la correction de bugs (maintenance corrective)

# Rôles des différentes arborescences

- Distribution :
  - Identifier des repères dans le développement (phase de développement, version, etc.)
  - Faciliter le support en figeant des versions
- Branches :
  - Correction de bugs
  - Tester de nouvelles fonctionnalités « risquées »

  sur des versions antérieures ou en parallèle à la version principale

# Opérations principales

- `import` : Initialisation/importation d'un projet dans un dépôt à partir d'une arborescence non gérées par un système de gestion de version
- `checkout` : récupération d'un projet dans un dépôt pour obtenir une copie locale
- `commit` : mise à jour du dépôt avec la copie locale
- `update` : mise à jour de la copie locale avec le dépôt Synchronisation avec des modifications des autres développeurs

# Terminologie

## Résumé

- Dépôt : ensemble des fichiers sous le système de gestion de version
- Copie locale : ensemble des fichiers sur la machine locale du développeur
- Révision : version d'un élément (fichier, répertoire, projet, etc.) du dépôt
- Tronc : arborescence principale d'un projet
- Distribution (`tags`) : arborescence figée d'un projet
- Branches : arborescences secondaires, zones de travail utilisées pour des développements ou des corrections en parallèle à l'arborescence principale
- Checkout : récupération des fichiers du dépôt dans leur version courante
- Commit : soumision des modifications et/ou des ajouts de la copie locale
- Update : Mise à jour de la copie locale par rapport aux fichiers sur le dépôt

# Subversion (SVN)

http://subversion.apache.org/

- Système de gestion de version permettant
  - le développement collaboratif
  - le partage de fichiers et de répertoire dans un dépôt
  - la numérotation des version des fichiers et répertoires
  - la récupération des anciennes versions des fichiers
  - la gestion des conflits sur un fichier
  - la diffusion des mises à jour du dépôt
- Centralisation du dépôt (modèle client-serveur)
- Disponible indépendamment de tout IDE
  mais accessible sous forme d'un module dans Eclipse
- Commande utilisateur (client) : `svn`
- Aide : `svn help` ou `svn help commande`
- Documentation : `http://svnbook.red-bean.com/`

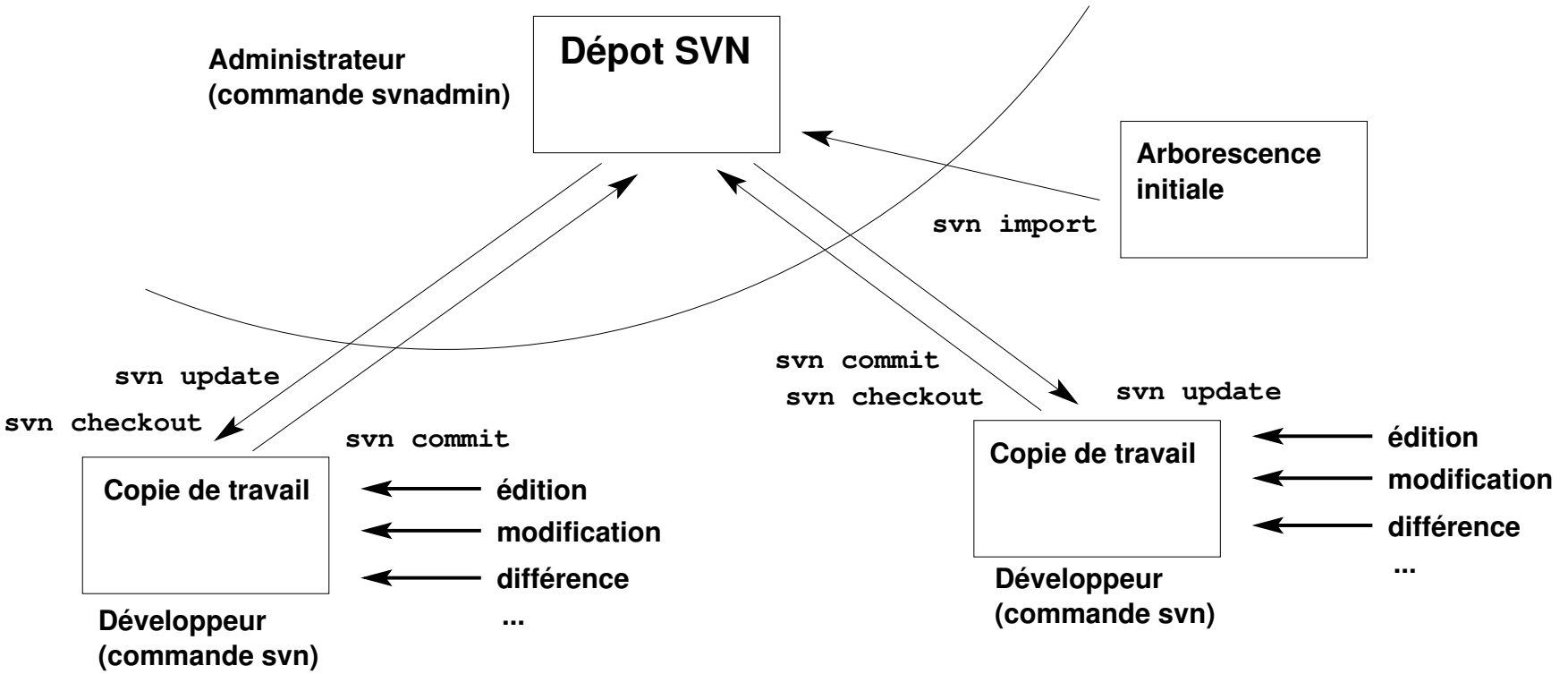
# Fonctionnalités de Subversion

- Système optimiste
  (mais possibilité de verrouillage si c'est vraiment nécessaire)
- Définition de méta-données (informations associées) sur les fichiers et les répertoires
- Révision des fichiers, répertoires et méta-données
- Manipulation de l'arborescence du dépôt (ajout, suppression et déplacement des fichiers et répertoires)
- Opérations de révision (`commit`/`update`/`import`)
  - regroupées sur plusieurs fichiers
  - atomiques (travail au niveau de la ligne et non du fichier)
- Gestion des branches et des distributions

# Caractéristiques techniques

- Stockage différentiel des fichiers
  Seules les modifications par rapport à la révision précédente sont enregistrées
- Gestion particulière des fichiers binaires
- Fonctionnement *offline* pour certaines opérations :
  - Etat des fichiers de la copie locale (`svn status`), différences entre deux révisions (`svn diff`), annulation des modifications locales (`svn revert`)
  - Utilisation des informations se trouvant dans les répertoires `.svn`
- Différents protocoles de communication (SGF, HTTP, SVN, SVN+SSH)

# Architecture SVN



# Création du dépôt

- En tant qu'administrateur de Subversion
- Utilisation de la commande `svnadmin`
- Création du dépôt : `svnadmin create <depot>`

- Aide pour `svadmin` : `svnadmin help`

# Sauvegarde/restauration du dépôt

`svnadmin dump <depot> > fichier`

- Sauvegarde l'ensemble du dépôt : fichiers, répertoires, méta-données, etc.
- Le fichier en sortie permet la restauration du dépôt

`svnadmin load <depot> < fichier`

- Restauration du dépôt à partir d'un fichier sauvegardé

# Création d'un projet

- Importation d'une arborescence initiale dans le dépôt
  Arborescence UNIX contenant et organisation les fichiers à ajouter au dépôt SVN
- Organisation sur le dépôt, dans le répertoire contenant le projet :
  - un répertoire `trunk` contenant la branche principale où sera importée l'arborescence principale
  - un répertoire `branches` où seront stockés plus tard les arborescences secondaires
  - un répertoire `tags` où seront stockées les distributions

Remarques :

- Organisation purement conventionnelle
  (rien d'obligatoire, mais quand même fortement conseiller)
- un dépôt peut héberger plusieurs projets

# Quels fichiers placer sur le dépôt ?

Tous les fichiers modifiés par les développeurs :

- Fichiers sources
  s'assurer qu'il est possible de reconstituer le produit à partir de ces fichiers
- Fichiers de documentation et de construction du logiciel (`makefile`, etc.)

Ne surtout pas mettre :

- les fichiers temporaires
- les fichiers compilés ou générés à partir des fichiers sources (fichiers objets, exécutables, librairies, archives)

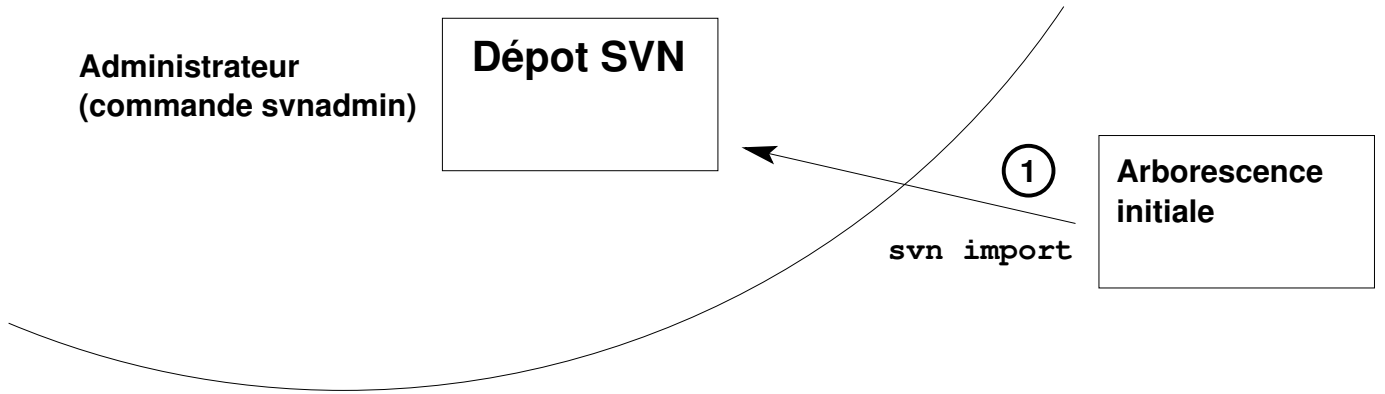
# Importation initiale

```
svn import -m "Message" <arborescence initiale>" <depot>
```

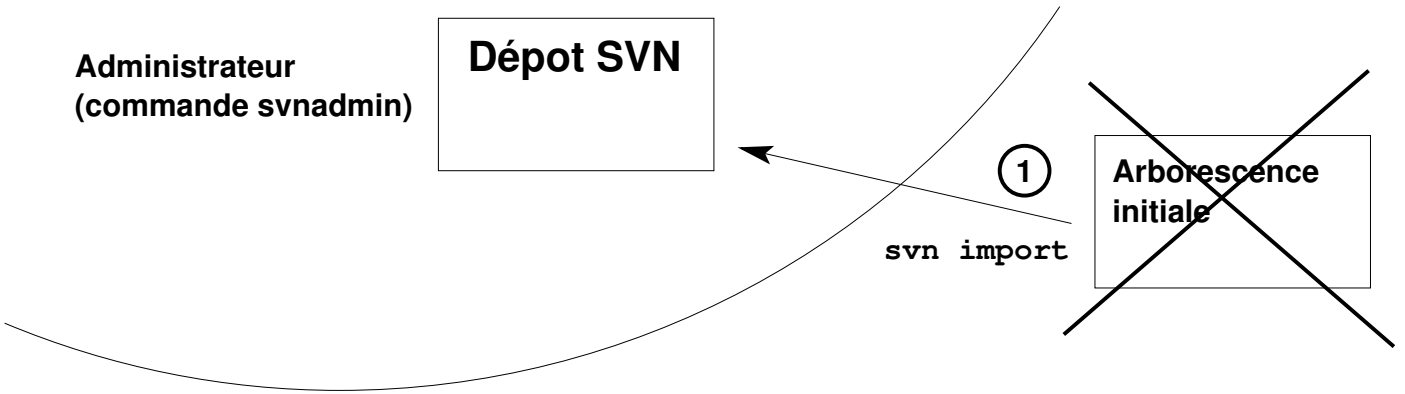
- Intégration de l'arborescence initiale (fichiers et répertoire dans le dépôt)
- Tous les fichiers et répertoires sont gérés sous SVN
- Création de la version 1 du dépôt
- Option `-m` : spécification d'un message associé à l'opération effectuée
  NB : sans l'option `-m`, un éditeur par défaut est exécuté pour rédiger un message (l'éditeur est défini dans la variable d'environnement `SVNEDITOR`.

Remarque importante : supprimer ou ne plus utiliser l'arborescence initiale par la suite (**elle n'est pas gérée par SVN**)

# Importation initiale (1)



# Importation initiale (2)



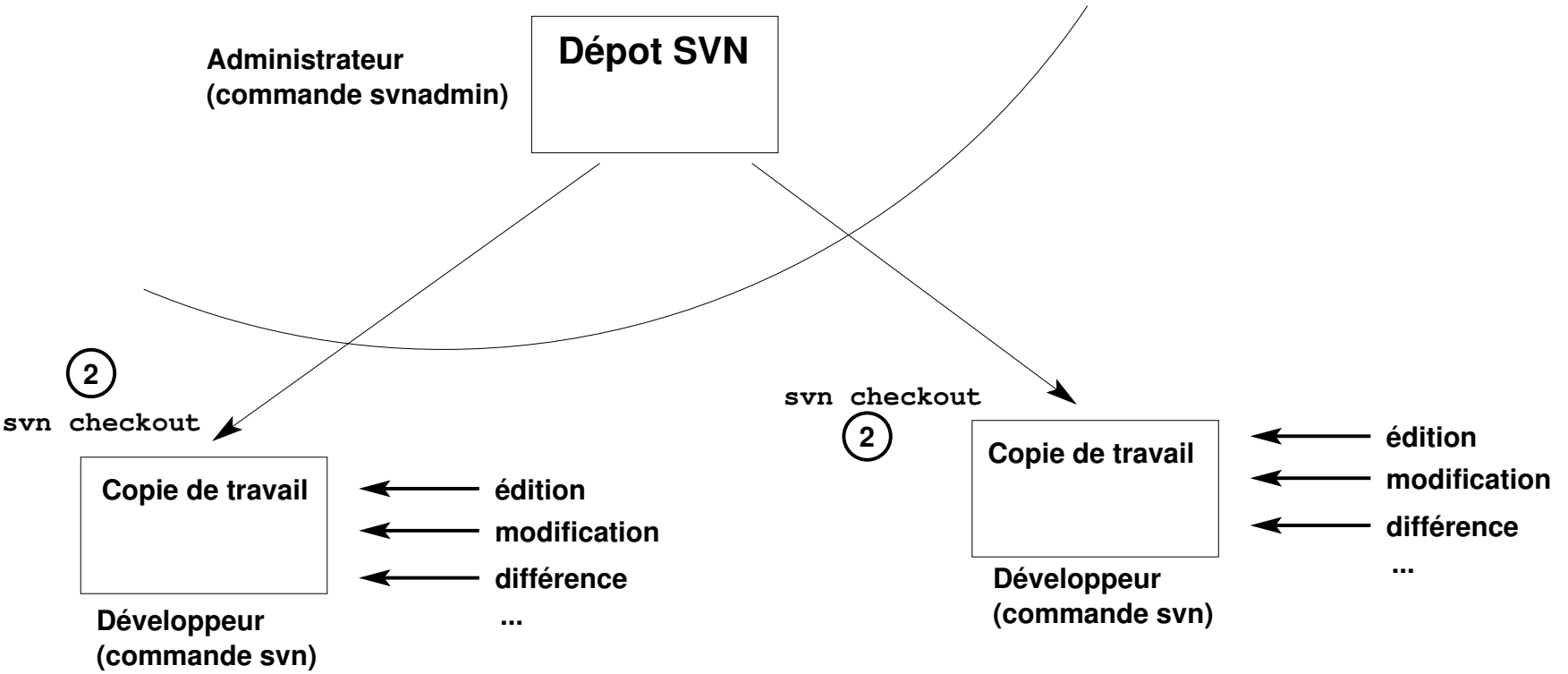
# Récupération d'une copie locale

`svn checkout <chemin du depot>`

- Création d'une copie locale du dépôt
- Chaque développeur doit effectuer l'opération avant de travailler sur les sources (édition, modification, ajout, suppression, etc.)
- Raccourci pour `checkout` : `co`

  `svn co <chemin du depot>`

# Récupération de la copie locale



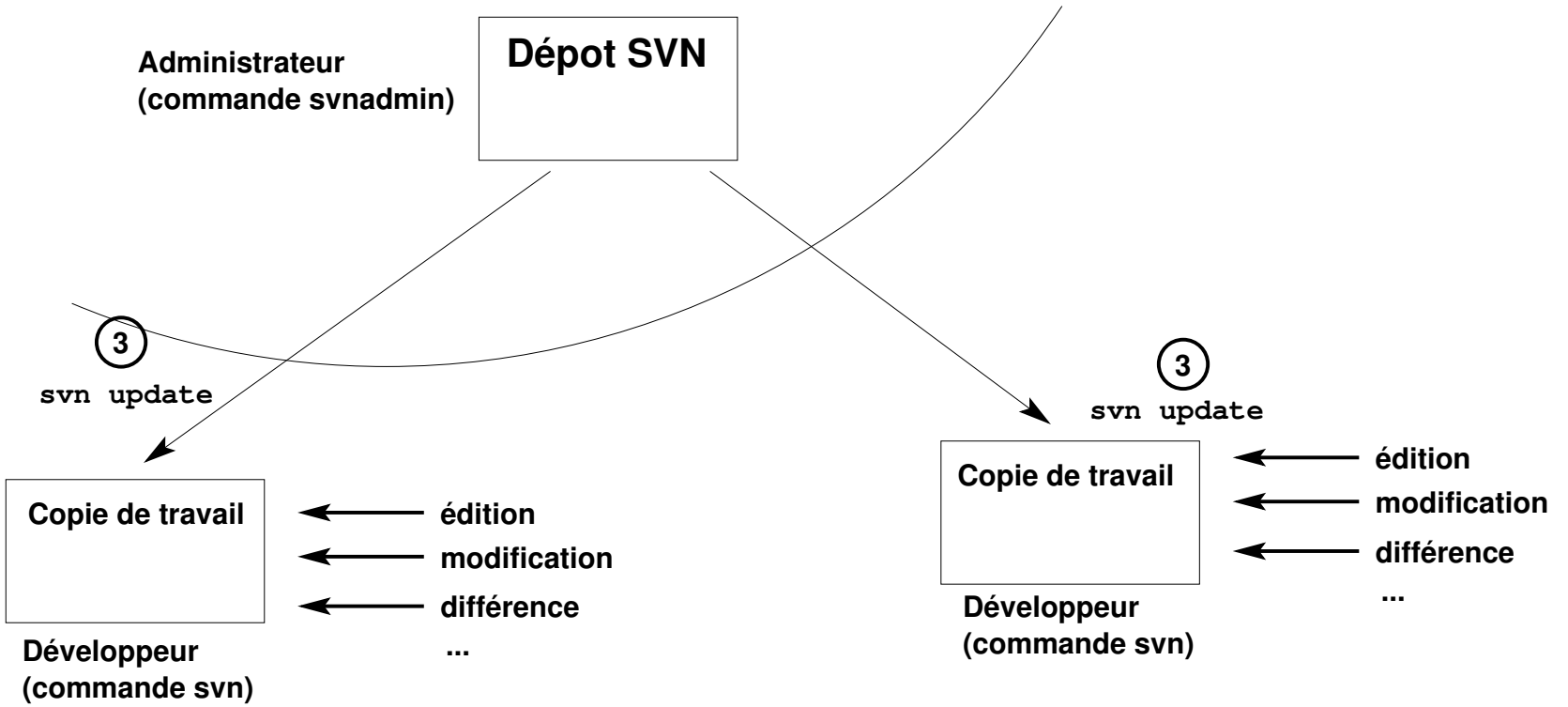
# Mise à jour de la copie locale

`svn update`
(en étant présent dans un répertoire sous SVN)

- Récupération/synchronisation de la copie locale avec les dernières modifications sur le dépôt
- Fusionne les données si nécessaire
- En cas d'échec de la fusion,
  - un conflit est identifié
  - l'utilisateur doit choisir entre les modifications sur le dépôt et ses propres modifications
- La mise à jour peut n'être demandée pour qu'une partie de l'arborescence
- Raccourci pour `update` : `up`

  `svn up`

# Mise à jour de la copie locale

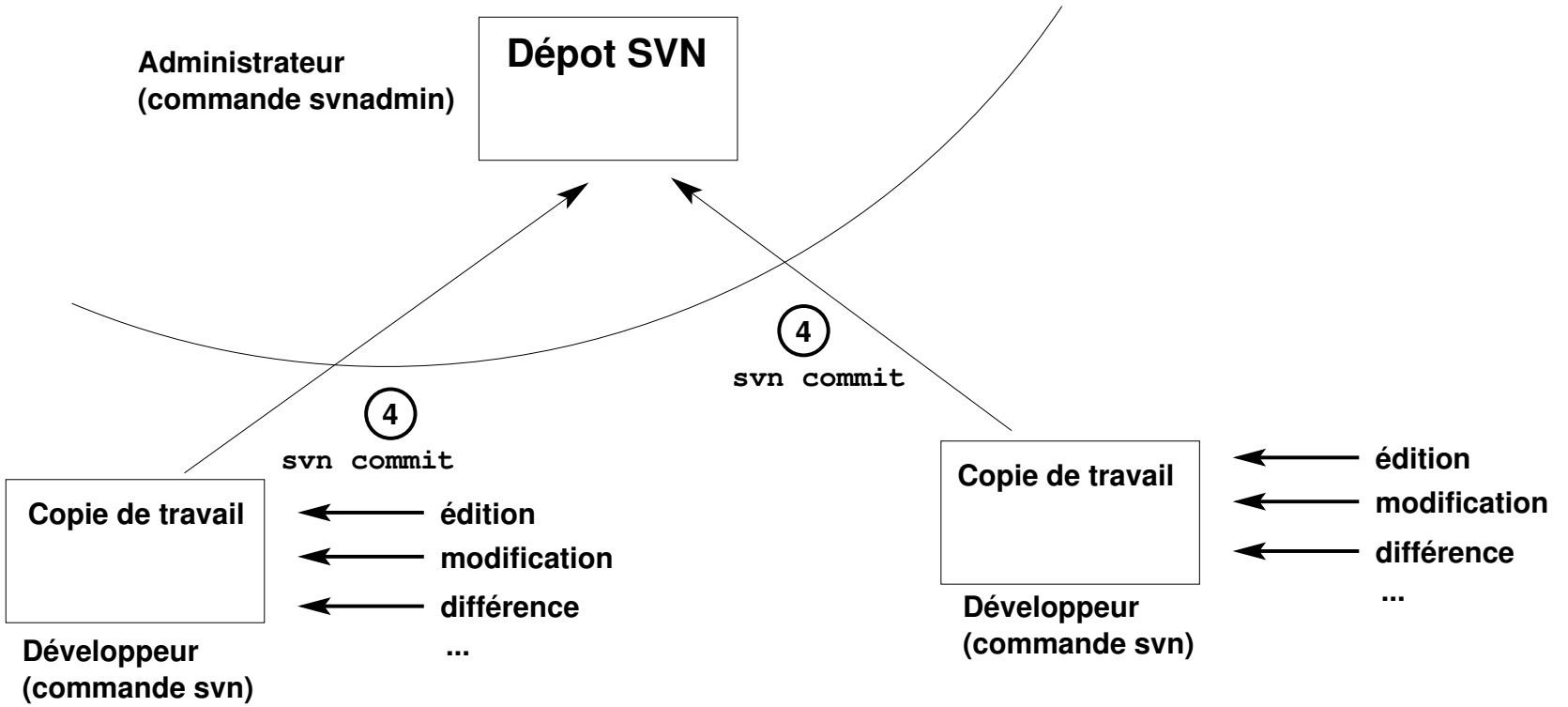


# Mise à jour du dépôt

```
svn commit -m "Message"
```

- Mise à jour du dépôt avec les modifications locales
- Possibilité d'échec si le dépôt a été mis à jour depuis le dernier `svn update`
  Il est nécessaire de faire une nouvelle mise à jour
- Option `-m` : spécification d'un message associé à l'opération effectuée (idem `svn import`)

# Mise à jour du dépôt



# Manipulation de l'arborescence

... sous SVN

(en étant présent dans un répertoire sous SVN)

- Ajout de fichier

  `svn add <fichier>`
- Ajout de répertoire

  `svn mkdir <fichier>`
- Déplacement/renommage de fichier

  `svn mv <fichier> <nouveau fichier`
- Suppression de fichier ou de répertoire

  `svn rm <fichier>`

  `svn delete <fichier>`

NB : Effectuer un `commit` après ces opérations pour qu'elles soient effectives sur le dépôt

**Attention** : ne pas utiliser de commandes UNIX pour réaliser ces opérations. SVN prend tout en charge (même sur la copie locale).

# Résolution d'un conflit

- Apparition possible d'un conflit lors d'un `commit` ou d'un `update`
- le statut est à `C` pour les fichiers où un conflit est apparu
- Résolution du conflit :
  - Editer du fichier local
    les zones en conflit sont marquées par

    `>>>>>>>>>>>>>>>`

    et

    `<<<<<<<<<<<<<<<`
  - Faire un choix dans les modifications à conserver et sauvegarder le fichier
  - Indiquer à SVN que le conflit est résolu :
    `svn resolved <fichier>`
  - Effectuer une mise à jour sur le dépôt (`commit`)

# Résolution d'un conflit

## Remarque

- SVN propose la résolution des conflits (l'édition des fichiers où se trouve un conflit) lors d'un `commit`
- les modifications locales peuvent être abandonnées :
  `svn revert <fichier>`
  (retour à la dernière mise à jour locale – `COMMITTED`)

# Examen du dépôt

- Informations sur le dépôt : `svn info`
- Liste des fichiers sur le dépôt : `svn list` (`svn ls`)
- Statut des fichiers : `svn status`
- Différences entre la copie locale et le dépôt : `svn diff`
  NB : Il est possible de mentionner des numéros de version :

  ```
  svn diff -r 11:20 fichier.txt
  ```

- Visualisation des logs : `svn log`

# Révisions

## numéros de version

- Réalisées par groupe d'actions (modification de fichiers, manipulation de l'arborescence, etc.)
- Numéro de version : entier croissant à chaque commit
- Un numéro de version par fichier
- Il existe des raccourcis :
  - `HEAD` : dernier numéro de version sur le dépôt
  - `BASE` : dernier numéro de version dans la copie locale (avant modification)
  - `COMMITTED` : numéro de version pour laquelle une modification a été placée sur le dépôt ($COMMITTED <= BASE$)
  - `PREV` : numéro de version juste avant `COMMITTED` ($PREV = COMMITTED - 1$)

# Statut des fichiers

`svn status`

- `A` : Il est prévu d'ajouter le fichier lors du prochain `commit`
- `D` : Il est prévu de supprimer le fichier lors du prochain `commit`
- `U` : Le fichier vient d'être mis à jour
- `M` : Le fichier de la copie locale a subi des modifications (par rapport au dernier `update`)
- `C` : Le fichier contient des conflits (apparus lors de la dernier mise à jour/fusion avec la version sur le dépôt)
- `G` : Le fichier n'est pas intégré au dépôt
- `?` : Le fichier n'est pas intégré au dépôt
- `!` : Le fichier est incomplète ou absent
- `X` : Le fichier n'est pas intégré dans la copie de travail

NB : Ces informations sont également disponibles lors de l'utilisation de svn update

# Substitution de mots-clés

- Insertion dans les fichiers d'informations relatives aux révisions (numéros de version, auteur, date, etc.)
- Substitution lors des `commit`
- Activation de la propriété de substitution des mots-clés :

```
svn propset svn:keywords "Id" fichier.txt
```

- Utilisation des mots-clés dans les fichiers (`$Id$`)
- Mots-clés disponibles :
  - `Date` (ou `LastChangedDate`) : date de la dernière révision sur le fichier
  - `Revision` :dernière révision sur le fichier
  - `Author` : Auteur de la dernière révision sur le fichier
  - `HeadURL` : URL du fichier sur le dépôt
  - `Id` : `File Revision Date Author`

# Création d'une distribution

- Copie d'une révision dans le répertoire `tags`
- `svn copy <depot>/trunk <depot>/tags/<release>`
  `release` est le nom de la distribution (par exemple `version-1.2`)
- ATTENTION : les modifications restent possibles sur la distribution (`à éviter absolument !`)
- Récupération de la distribution :
  `svn co <depot>/tags/<release>`
- Création d'une copie « non-versionnée » de la distribution :
  `svn export <depot>/tags/<release>`

# Création d'un patch

- Objectif : transmettre des modifications en dehors du dépôt SVN
  Modification de sources déjà publiées
- Création du patch : génération d'un fichier des différences entre deux révisions du dépôt (fichier ou arborescence)
  Exemple sur un fichier :

```
svn diff -r 12:13 fichier > fichier-r12-13.patch
```

- Application du patch sur une arborescence quelconque :
  (la où se trouve le fichier à `patcher`)

```
patch < fichier-r12-13.patch
```

# Création d'une branche

- Objectif : Réaliser des développements ou des corrections en parallèle au tronc (`trunk`)
- Branche : copie d'une révision de l'arborescence principale (`trunk`)
- `svn copy <depot>/trunk <depot>/branches/<feature>`
  `feature` est l'identifiant de la branche créée (par exemple, `ma-branche`)
- Utilisation :

  `svn co <depot>/branches/<feature>`

  Puis travail sur la branche de la même manière que le tronc
- Possibilité de fusion des modifications de la branche dans le tronc, et inversement

# Fusion des branches

- Rapatriement des modifications dans le tronc (ou dans un autre branche) et inversement
- `svn merge -rRi <depot>/branches/<feature>`
  Application de la révision Ri de la branche `feature` à la copie locale courante (tronc ou autre branche)
- NB : il est fortement conseiller de commencer par examiner les différences :

  `svn diff -rRi <depot>/branches/<feature>`

# Conseils et bonnes pratiques

- Effectuer des mises à jour (`commit`) régulières de la copie locale (limite les possibilités de conflits)
- Préférer les mises à jour (`update`) de l'ensemble de l'arborescence
- Toujours réaliser un `update` avant un `commit`
- Réaliser des `commit` correspondant à un ensemble cohérent de modifications
- Toujours compiler et tester les sources avant de réaliser un `commit`
- Vérifier que les fusions sont réalisées correctement
- Ne pas prendre à la légère les conflits
- Ne pas utiliser les commandes système pour les modifications d'arborescence (utiliser `svn add|mv|rm`)
- Documenter les commits avec des messages explicites (accessibles ensuite avec `svn log`)

  Les messages de commit peuvent être envoyés automatiquement à toutes l'équipe

Ne pas oublier : `svn help` ou `svn help commande`

# Sources des transparents

- Georges-Etienne Legendre, Félix-Antoine Bourbonnais
- Jean-Fran¸cois Mari, IUT Charlemagne, dpt. informatique
- Didier DONSEZ, Université Joseph Fourier - Grenoble 1, PolyTech'Grenoble LIG/ADELE
- Nicolas Hernandez, IUT de Nantes - Département Informatique