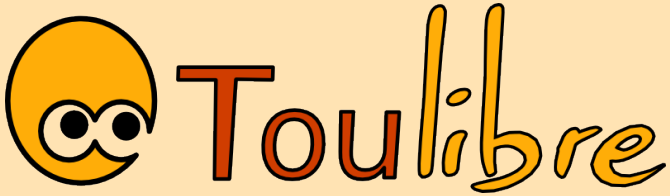
Gestion de version

centralisée et décentralisée

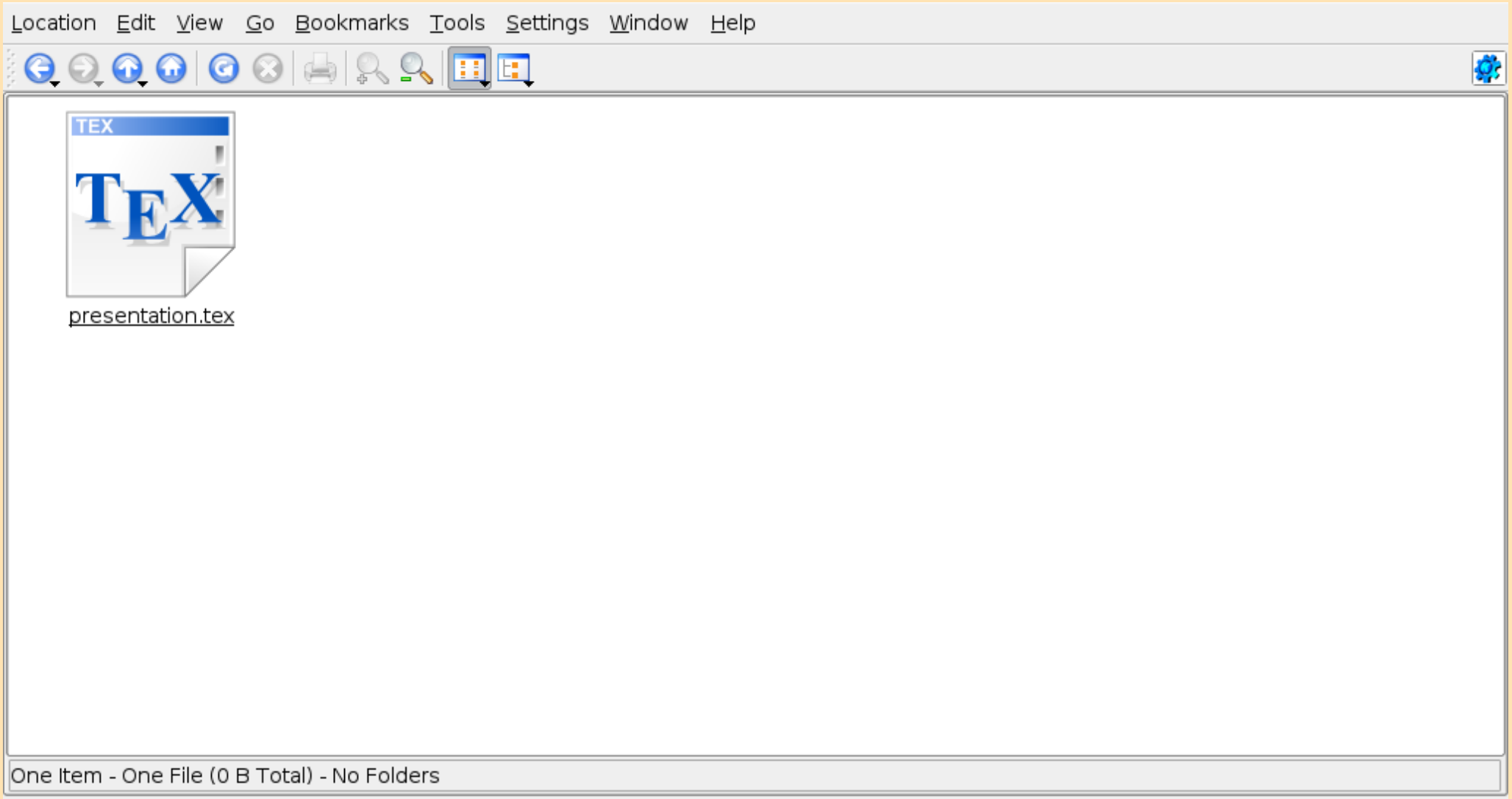
Principes et application avec Subversion et Arch



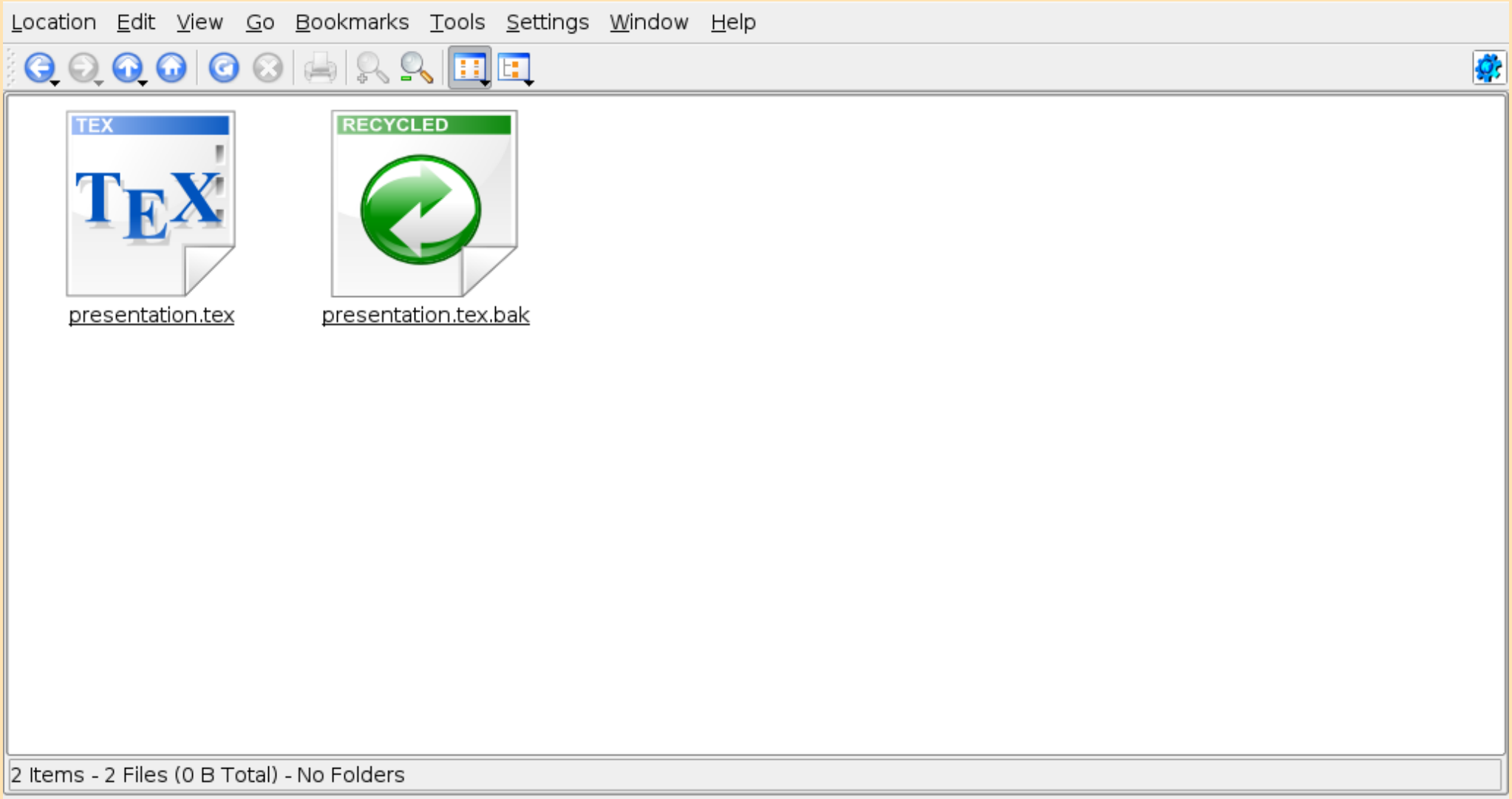
Thomas Petazzoni
Ludovic Courtès

Mercredi 10 Janvier

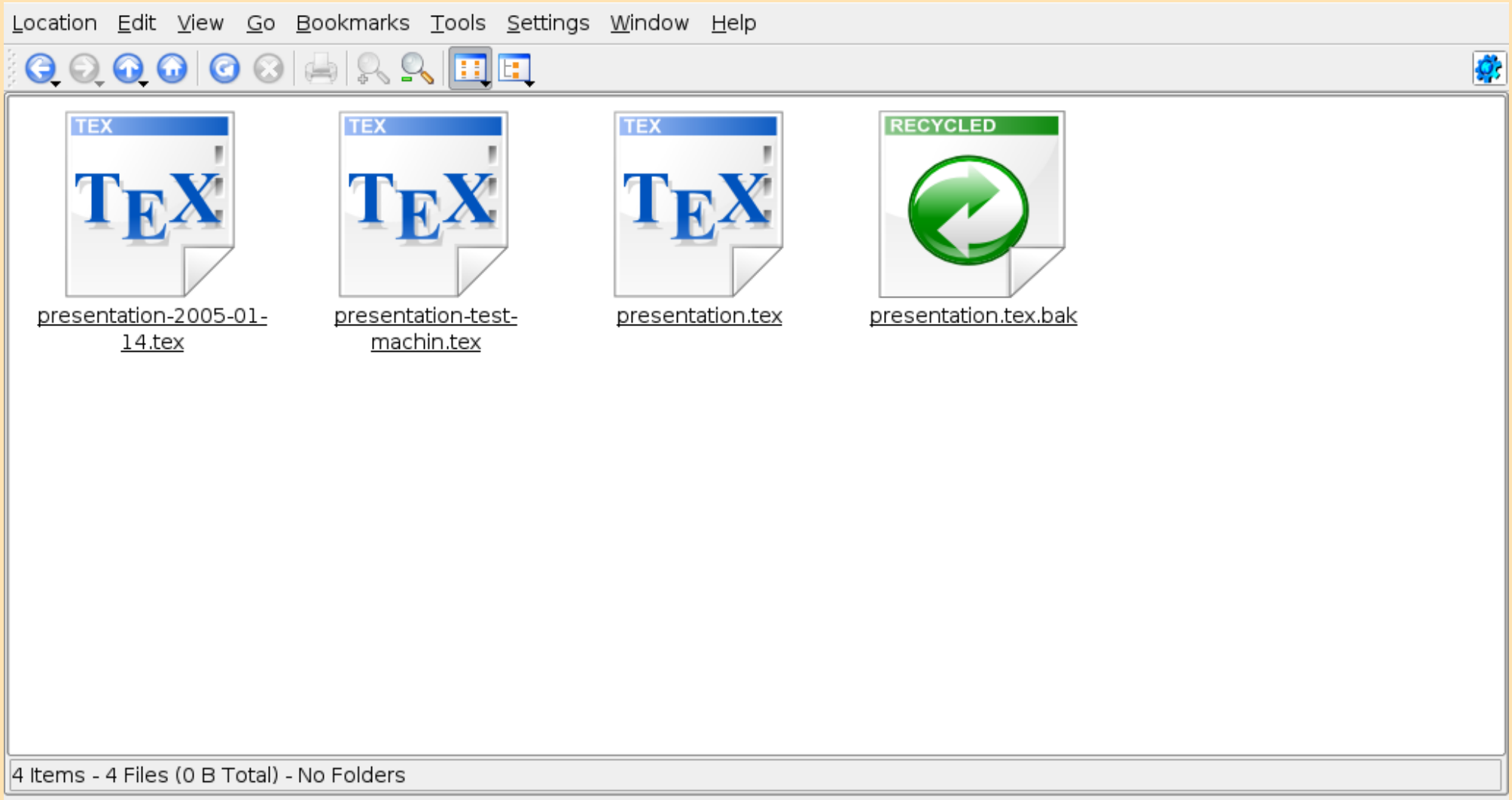
Je commence un document



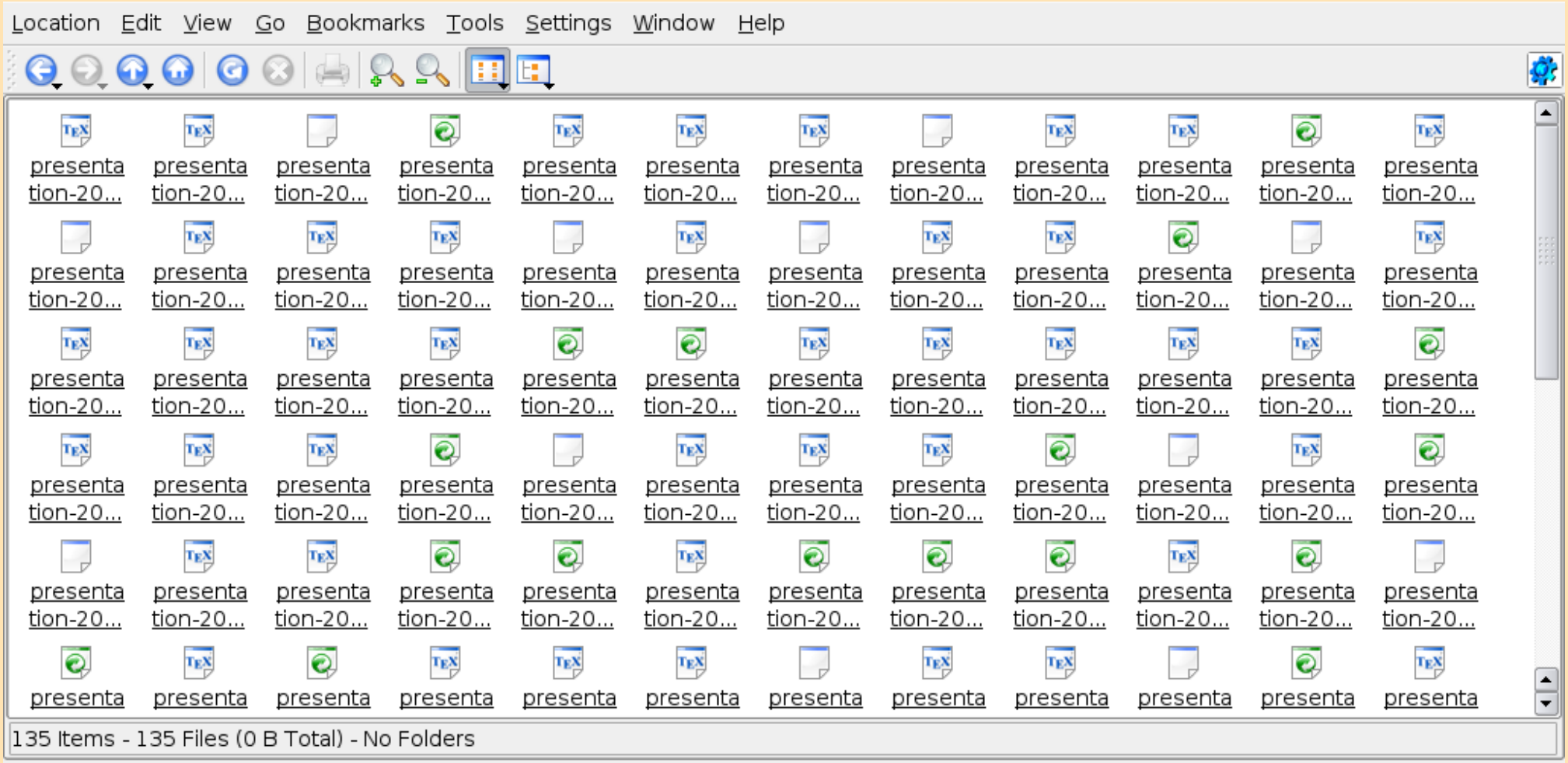
Je fais une sauvegarde



On copie pour tester un truc



Euh, j'éditais quel fichier déjà ?



Problèmes

- Explosion du nombre de fichiers à gérer
- Gestion manuelle du graphe de versions/modifications
- Encore plus complexe quand le travail se fait à plusieurs

La gestion de version se propose de résoudre ces problèmes

Gestion de version

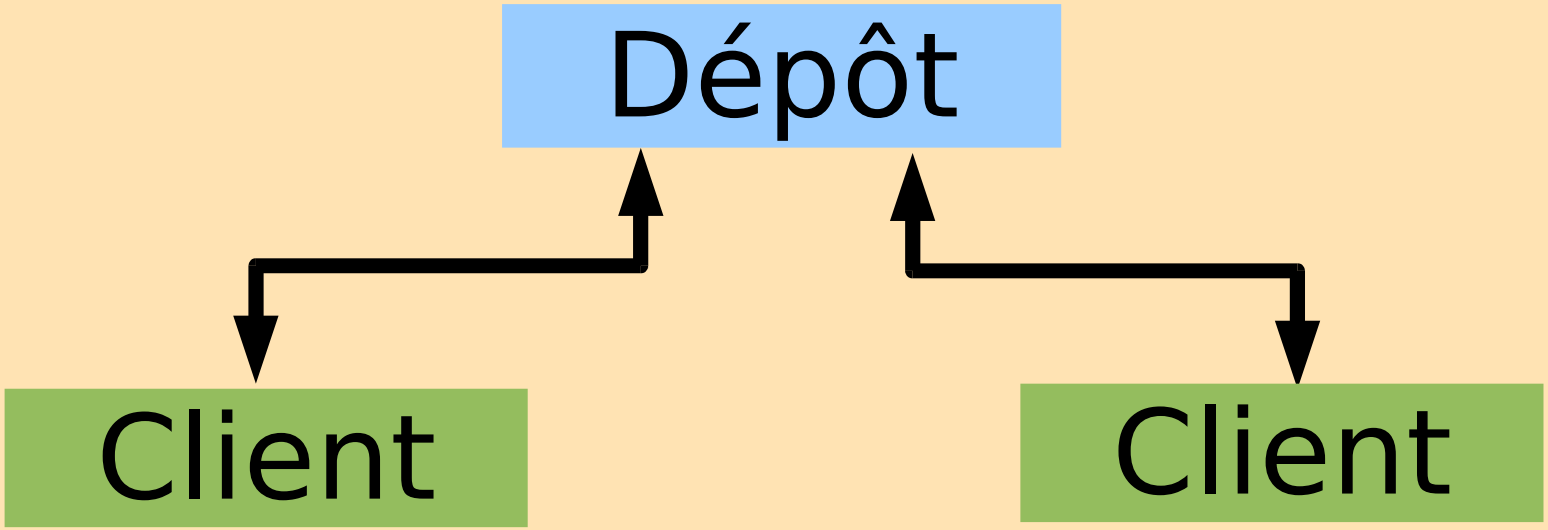
- Activité consistant à maintenir l'intégralité de l'historique des fichiers d'un projet
- Toutes les versions de chaque fichier sont enregistrées
- Permet de
  - revenir en arrière en cas d'erreur
  - relire les modifications introduites entre deux versions différentes
  - travailler à plusieurs sur un même projet
  - gérer des branches de développement parallèles
  - estampiller des versions bien identifiées du projet

Gestion de version

- Utile pour:
  - code source
  - documents
    - HTML, LaTeX, texte brut
  - données binaires, avec des fonctionnalités réduites
- Indispensable dès que l'on travaille à plusieurs
  - utilisé massivement pour le développement des Logiciels Libres
- Très pratique même lorsque l'on travaille seul (historique, etc.)

Gestion centralisée

- En gestion de version dite *centralisée:* un serveur central appelé **dépôt** ou **repository**



- **CVS:** l'outil historique, encore très utilisé
- **Subversion:** la version moderne

Opérations de base

- Créer une copie de travail du dépôt
  - svn checkout
- Éditer les fichiers
  - emacs, vi
- Voir les différences
  - svn diff
- Propager ses modifications vers le dépôt
  - svn commit
- Récupérer les autres modifications en provenance du dépôt
  - svn update

Démonstration !

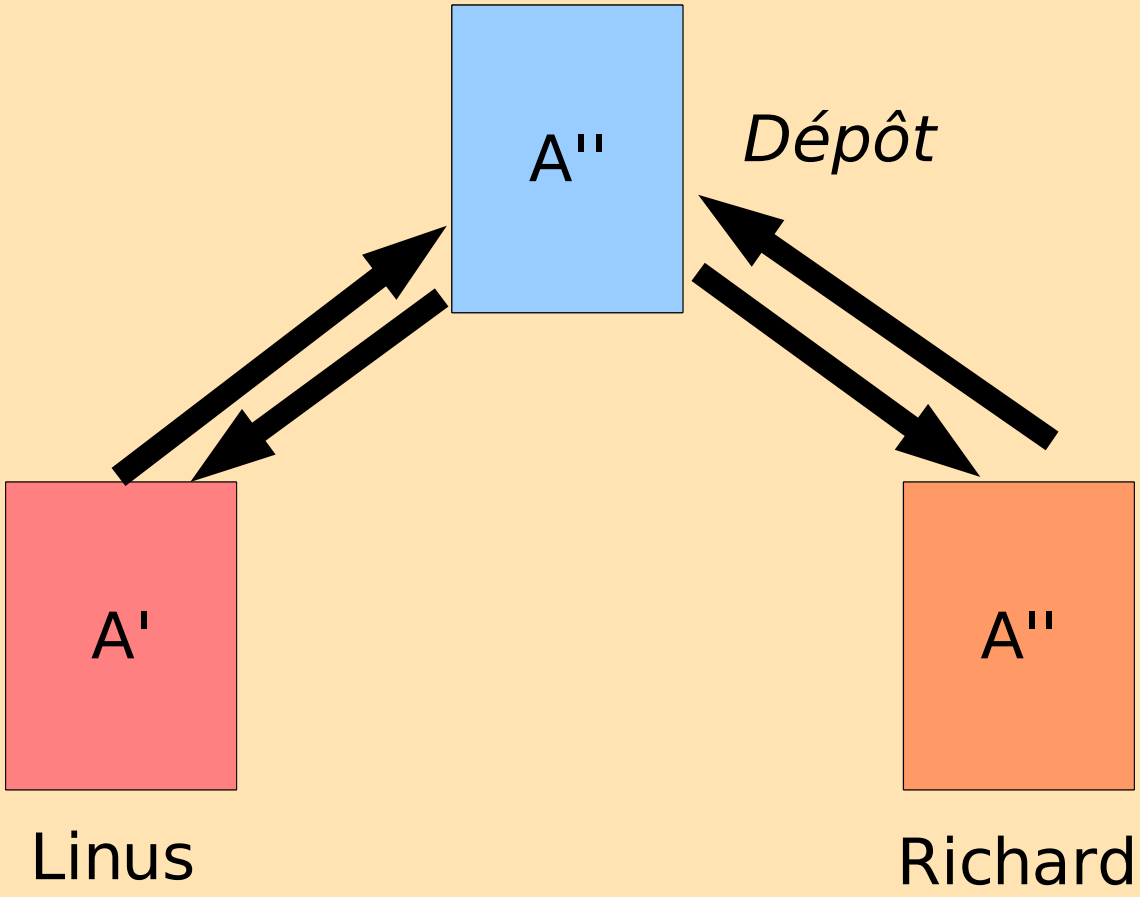
Changeset

- Après modification de plusieurs fichiers, on peut les *committer* ensemble
  - ces modifications forment un *changeset*
  - le numéro de version du dépôt s'incrémente, ce numéro identifie de manière unique le *changeset*
  - Subversion est différent de CVS sur ce point, pas de notion de changeset dans CVS, chaque fichier évolue séparément
- La bonne pratique est d'avoir un *changeset* pour chaque modification «logique»

Démonstration !

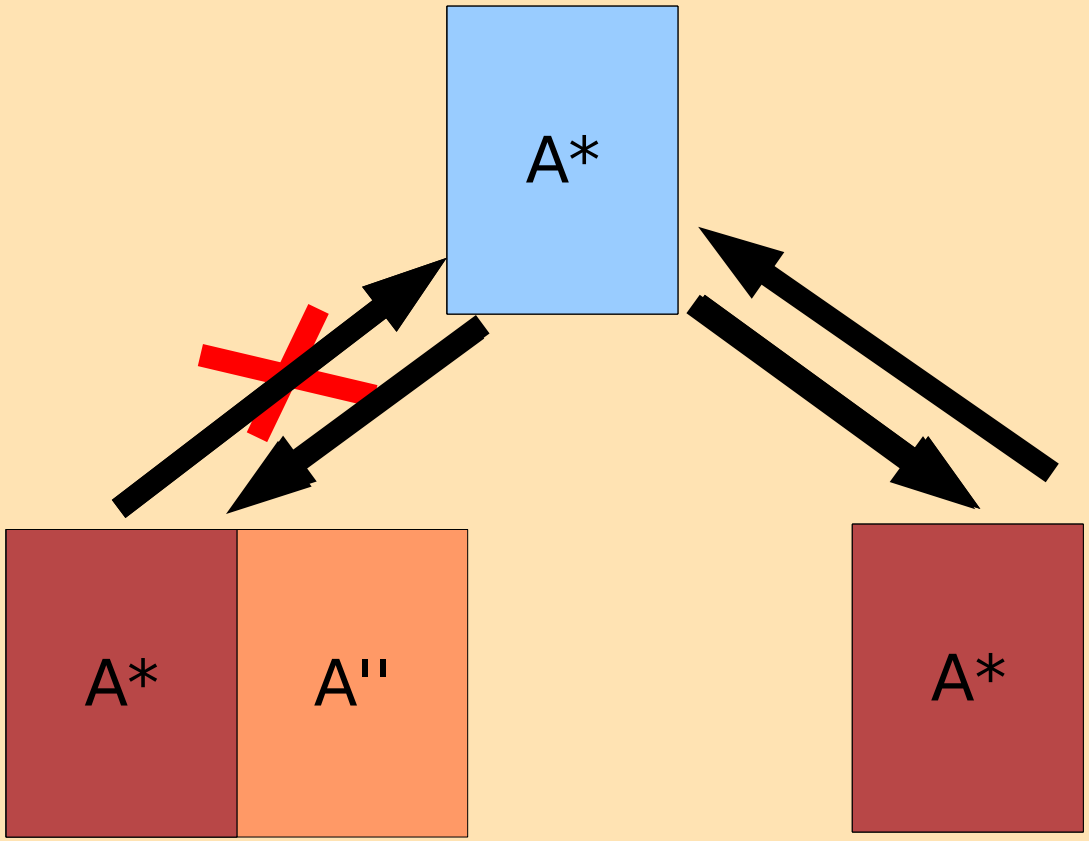
Gestion des conflits

- Si plusieurs personnes travaillent en même temps sur un fichier et le modifient, comment résoudre le conflit ?



Gestion des conflits

- Solution simpliste: verrouillage
- Solution efficace: **fusion**



Démonstration !

Opérations plus avancées

- Résoudre un conflit
  - éditer, puis svn resolved fichier
- Ajouter un fichier
  - svn add fichier
- Renommer/déplacer un fichier
  - svn mv fichier1 fichier2
- Supprimer un fichier
  - svn rm fichier1
- Copier un fichier
  - svn copy fichier1

Démonstration !

Étiquettes, branches

- On peut vouloir identifier par un nom symbolique une version donnée du projet
  - étiquette = tag
  - « poser un **tag** »
- On peut vouloir travailler sur une branche particulière d'un projet, pour
  - faire de la maintenance
  - développer une nouvelle fonctionnalité qui risque d'être assez instable au départ
  - « créer une **branche** »

Étiquettes, branches

- Contrairement à CVS, pas de gestion explicite des étiquettes et branches
  - tout se fait avec des copies
  - les copies sont paresseuses
  - permet de versionner les créations/destructions de tags/branches

Étiquettes, branches

- Organisation classique d'un projet
  - project/
    - trunk/ Branche principale
    - tags/ Répertoire où sont stockés les étiquettes
    - branches/ Répertoire où sont stockés les branches
- Créer une étiquette
  - svn copy ..../project/trunk ..../project/tags/v1.0
- Détruire une branche
  - svn rm ..../project/branches/new-feature
- Fusionner
  - svn merge .../project/trunk ../project/branches/new-feature

Démonstration !

Questions ?

Suite...

Gestion de version décentralisée, avec Arch