Version Control with Subversion

For Subversion 1.5

(Compiled from Revision4246)

Ben Collins-Sussman
Brian W. Fitzpatrick
C. Michael Pilato

Version Control with Subversion: For Subversion 1.5: (Compiled from Revision4246)

par Ben Collins-Sussman, Brian W. Fitzpatrick, et C. Michael Pilato

Date de publication (TBA)

Table des matières

Liste des illustrations

Liste des tableaux

Liste des exemples

Avant-propos

Karl Fogel
Chicago, le 14 mars 2004.

Une mauvaise FAQ est composée non pas des questions que posent les utilisateurs, mais de celles que l'auteur de la FAQ *voudrait* qu'on lui pose. Peut-être avez-vous rencontré ce type de FAQ :

> Q : Comment peut-on utiliser Glorbosoft XYZ pour maximiser la productivité de nos équipes ?
>
> R : Beaucoup de nos clients veulent savoir comment maximiser la productivité avec notre nouvelle suite bureautique brevetée. La réponse est simple : cliquez sur le menu `Fichier`, puis trouvez `Améliorer la productivité` plus bas, ensuite…

Le problème avec de telles FAQ, c'est qu'elles ne sont pas du tout, au sens propre, des FAQ. Personne n'a appelé le support technique et demandé « Comment pouvons-nous améliorer la productivité ? » Au lieu de ça, les gens posent des questions très précises, telles que « Comment pouvons-nous configurer le système de calendrier pour envoyer les rappels deux jours en avance au lieu de 24 heures ? » etc. Hélas, il est tellement plus facile d'imaginer des questions que de trouver celles qui sont vraiment fréquemment posées. Rédiger une vraie FAQ requiert un effort continu et une bonne organisation : tout au long de la vie du logiciel, les questions posées ainsi que leurs réponses doivent être suivies de près, puis rassemblées et organisées de façon claire et cohérente dans un tout qui doit refléter l'expérience des utilisateurs. Cela nécessite d'être patient et observateur, tel un naturaliste. Ici, pas de grandes théories ni de discours visionnaires, ce qu'il faut avant tout, c'est ouvrir les yeux et prendre des notes.

Ce que j'aime à propos de ce livre, c'est qu'il a été créé en suivant ce procédé, ce qui se ressent à chacune de ses pages. C'est le résultat direct de la rencontre des auteurs et des utilisateurs. Tout a commencé lorsque Ben Collins-Sussman remarqua que les gens posaient constamment les mêmes questions de base sur la liste de diffusion de Subversion : Quelles sont les procédures pour travailler avec Subversion ? Est-ce que les branches et les étiquettes fonctionnent comme dans les autres systèmes de gestion de versions ? Comment est-ce que je peux trouver qui a fait telle ou telle modification ?

Frustré de voir revenir les mêmes questions jour après jour, Ben travailla d'arrache-pied pendant un mois durant l'été 2002 pour écrire *The Subversion Handbook*, un manuel de soixante pages couvrant toutes les bases de Subversion. Le manuel ainsi écrit n'avait pas la prétention d'être complet, mais il fut distribué avec Subversion pour aider les utilisateurs à faire leurs premiers pas dans l'apprentissage de Subversion. Quand O'Reilly and Associates décidèrent de publier un livre complet sur Subversion, la voie la plus facile était la plus évidente : simplement améliorer *The Subversion Handbook*.

Une opportunité inhabituelle se présenta donc aux trois co-auteurs de ce nouveau livre. Officiellement, leur tâche était d'écrire un livre « académique », en partant d'une table des matières et d'une première ébauche. Mais ils avaient aussi accès à un flux constant, une quantité incontrôlable en fait, de réactions en provenance des utilisateurs. Subversion était déjà entre les mains de quelques milliers d'utilisateurs précoces, et ces derniers envoyaient des tonnes de commentaires, pas seulement sur Subversion, mais aussi sur sa documentation d'alors.

Pendant que Ben, Mike et Brian écrivaient ce livre, ils surveillèrent sans relâche la liste de diffusion et les salons de discussion de Subversion, notant consciencieusement les problèmes que rencontraient les utilisateurs dans la réalité. Assurer le suivi de ces retours d'expériences faisait de toutes façons partie intégrante de leur travail à CollabNet, et cela leur donna un énorme avantage quand ils commencèrent à rédiger la documentation de Subversion. Le livre qu'ils ont écrit repose sur un socle d'expérience pratique, pas sur une liste abstraite de bonnes intentions ; il possède à la fois les qualités du mode d'emploi et de la FAQ. Cette dualité ne saute pas immédiatement aux yeux. Lu dans l'ordre, de la première à la dernière page, ce livre décrit de manière simple un logiciel. Il y a la vue d'ensemble, l'incontournable visite guidée, le chapitre sur la configuration et l'administration, quelques sujets avancés, et bien évidemment une liste complète des commandes ainsi qu'un guide de débogage. Mais c'est quand on revient chercher dans ce livre une réponse à un problème spécifique qu'on réalise son authenticité, faite de détails révélateurs ne pouvant provenir que de cas concrets et inattendus, d'exemples tirés de situations réelles, et par-dessus tout de l'attention portée aux besoins et aux remarques des utilisateurs.

Bien sûr, personne ne peut affirmer que ce livre répondra à toutes vos questions sur Subversion. De temps en temps, la précision avec laquelle il anticipe vos questions vous semblera presque télépathique ; mais d'autres fois, vous tomberez sur une lacune dans le savoir de la communauté, et vous rentrerez bredouille. Quand cela arrive, le mieux que vous puissiez faire est d'envoyer un courrier électronique à `<users@subversion.tigris.org>` (en anglais si possible) en y décrivant votre problème. Les auteurs sont toujours là, à l'affût, et il ne s'agit pas seulement des trois personnes citées sur la couverture du livre, mais aussi de beaucoup d'autres contributeurs ayant apporté corrections et améliorations. Pour la communauté, résoudre

votre problème est une composante agréable d'un projet bien plus vaste, celui de peaufiner petit à petit ce livre, et finalement Subversion lui-même, pour encore mieux coller à l'utilisation que les gens en ont. Les auteurs sont très enthousiastes à l'idée de communiquer avec vous, pas seulement parce qu'ils peuvent vous aider, mais aussi parce que vous pouvez les aider. Avec Subversion, comme avec tous les projets de logiciels libres en activité, *vous n'êtes pas seul*.

Ce livre est votre premier compagnon.

Préface

> « Il est important de ne pas laisser la perfection devenir l'ennemi du bien, même lorsque vous pouvez être d'accord sur ce qu'est la perfection. Encore plus lorsque vous ne le pouvez pas. Aussi déplaisant qu'il soit d'être piégé par les erreurs du passé, vous ne pouvez pas faire de progrès en ayant peur de votre propre ombre pendant la conception. »
>
> —Greg Hudson, développeur de Subversion

Dans le monde du logiciel libre, le logiciel « Concurrent Versions System » (CVS) fut l'outil de choix pour la gestion des versions pendant de nombreuses années. Et à juste titre. CVS était lui-même open source et son mode de fonctionnement non-restrictif couplé à son support des opérations réseau permettait à des dizaines de programmeurs dispersés aux quatre coins du monde de partager leur travail. Cela collait très bien à la nature collaborative de l'open source. CVS et son modèle de développement semi-chaotique sont depuis devenus des pierres angulaires de la culture du logiciel libre.

Mais CVS n'était pas parfait et simplement corriger ses défauts promettait d'être un énorme effort. C'est ici que Subversion entre en jeu. Les créateurs de Subversion l'ont créé pour être un successeur de CVS et l'ont fait de façon à gagner le cœur des utilisateurs de CVS de deux façons : en concevant une interface similaire à CVS et en tentant d'éviter la plupart des défauts majeurs de CVS. Bien que le résultat ne soit pas forcément la prochaine évolution majeure dans les systèmes de gestion de versions, Subversion *est* très puissant, parfaitement utilisable et très flexible. Et la plupart des nouveaux projets en logiciel libre choisissent Subversion au lieu de CVS.

Ce livre est écrit pour documenter les versions 1.5 du système de gestion de versions Subversion. Nous avons tenté au maximum d'être complet dans cet ouvrage. Cependant, Subversion a une communauté prospère et dynamique ; il y a donc déjà un certain nombre de fonctionnalités et d'améliorations prévues pour des versions futures de Subversion qui peuvent modifier quelques commandes ou rendre caduques certaines notes spécifiques de ce livre.

Public visé

Ce livre est écrit pour les personnes désirant utiliser Subversion pour gérer leurs données. Subversion fonctionne sur un grand nombre de systèmes d'exploitation et son interface première est en ligne de commande. Ce programme (**svn**) et certains programmes auxiliaires sont le sujet de ce livre.

Par souci de cohérence, les exemples du livre supposent que le lecteur utilise un système de type Unix et qu'il est relativement à l'aise avec ce système ainsi qu'avec les interfaces en ligne de commande. Cela dit, le programme **svn** fonctionne également sur les systèmes qui ne sont pas basés sur Unix, tel que Microsoft Windows. Avec quelques petites exceptions, telles que l'utilisation d'anti-slashs (`\`) au lieu de slashs (`/`) dans les chemins, les entrées et sorties de ce programme sous Windows sont identiques à leurs équivalents Unix.

La plupart des lecteurs sont probablement des programmeurs ou des administrateurs systèmes qui ont besoin de suivre les changements faits à du code source. C'est l'utilisation la plus courante de Subversion et c'est ce que l'on supposera tout au long des exemples du livre. Cependant, Subversion peut être utilisé pour gérer les changements pour toutes sortes de données : images, musique, bases de données, documentation, etc… Pour Subversion, toutes les données sont justes des données.

Bien que ce livre soit écrit en supposant que le lecteur n'a jamais utilisé un système de gestion de versions, nous avons aussi essayé de rendre facile le passage de CVS (et autres systèmes) à Subversion. De temps en temps, quelques encadrés mentionnent d'autres systèmes de gestion de versions et une annexe particulière résume la plupart des différences entre CVS et Subversion.

Notez également que les exemples de code source présentés au cours du livre ne sont que des exemples. Même s'ils compileront avec les commandes de compilation adéquates, ils n'ont pour but que d'illustrer une situation particulière et ne sont pas nécessairement de bons exemples de style ou techniques de programmation.

Comment lire ce livre

Les manuels techniques doivent toujours faire face au dilemme suivant : choisir une approche d'apprentissage *descendante* ou *ascendante* pour le lecteur ? Un adepte de l'approche descendante préférera lire ou survoler la documentation, pour obtenir une vision globale du fonctionnement du système ; à partir de ce moment seulement, il commence à utiliser le logiciel. Un adepte de l'approche ascendante est plus un autodidacte, il se jette directement dans le logiciel et en comprend au fur et à mesure les

fonctionnalités, se référant au manuel en tant que de besoin. La plupart des livres sont écrits pour un certain type de lecteur et celui-ci est indubitablement orienté pour les adeptes de l'approche descendante (d'ailleurs, si vous lisez ce chapitre, c'est que vous êtes probablement dans cette catégorie !). Mais si vous êtes autodidacte, ne fuyez pas. Bien que le livre puisse être vu comme un large survol des fonctionnalités de Subversion, le contenu de chaque paragraphe est gorgé d'exemples et d'exercices pratiques. Pour les plus impatients, rendez-vous directement à l'Annexe A, *Guide de démarrage rapide avec Subversion*.

Quelle que soit votre manière d'apprendre, ce livre se veut utile pour des gens ayant des parcours et des compétences très variés, depuis le novice en gestion de versions jusqu'à l'administrateur système expérimenté. En fonction de votre expérience, certains chapitres vous sembleront plus ou moins importants. Nous proposons ci-dessous un « parcours » adapté à différents types de lecteurs :

Administrateur système expérimenté
: Nous supposons dans ce cas que vous avez déjà utilisé un système de gestion de versions et que vous voulez monter un serveur Subversion le plus rapidement possible. Le Chapitre 5, *Administration d'un dépôt* et le Chapitre 6, *Configuration du serveur* expliquent comment créer votre premier dépôt et le mettre à disposition sur le réseau. Ceci fait, le Chapitre 2, *Utilisation de base* et l'Annexe B, *Guide Subversion à l'usage des utilisateurs de CVS* sont le plus court chemin pour apprendre à utiliser le client Subversion.

Novice
: Votre administrateur vient probablement de mettre en place Subversion et vous devez apprendre à utiliser le client. Si vous n'avez jamais utilisé de système de gestion de versions, alors le Chapitre 1, *Notions fondamentales* est une introduction indispensable aux concepts de la gestion de versions. Le Chapitre 2, *Utilisation de base* est un tour du propriétaire du client Subversion.

Utilisateur avancé
: Que vous soyez utilisateur ou administrateur, votre projet va finir par prendre de l'importance. Il vous faudra apprendre comment effectuer des opérations plus pointues avec Subversion, comme par exemple utiliser des branches ou effectuer des fusions (Chapitre 4, *Gestion des branches*), utiliser les propriétés des objets Subversion (Chapitre 3, *Sujets avancés*), configurer les options d'exécution (Chapitre 7, *Personnalisation de Subversion*) et d'autres choses encore. Ces chapitres ne sont pas indispensables au début, mais pensez bien à les lire une fois que vous vous sentirez à l'aise avec les bases.

Développeur
: Vous êtes certainement déjà habitué à Subversion et vous voulez à présent étendre ses fonctionnalités ou développer un nouveau logiciel utilisant ses nombreuses API. Le Chapitre 8, *Intégration de Subversion* est là pour vous.

Le livre se termine par le Chapitre 9, *Références complètes de Subversion*. C'est le guide de référence pour toutes les commandes de Subversion, les annexes couvrant certaines notions particulièrement utiles. Ce sont certainement les chapitres vers lesquels vous retournerez une fois la première lecture terminée.

Conventions utilisées dans ce livre

Les conventions typographiques suivantes sont utilisées dans ce livre :

`Largeur fixe`
: Utilisée pour la forme littérale des commandes, les résultats de commandes et les paramètres des commandes.

`Italique`
: Utilisée pour les noms de programmes et de sous-commandes d'outils Subversion, les noms de fichier et de répertoire et les termes nouveaux.

`Largeur fixe italique`
: Utilisée pour les éléments à remplacer dans le texte et le code.

Nous avons également placé des éléments d'information particulièrement utiles ou importants en évidence, de façon à ce qu'ils soient faciles à trouver (à des emplacements cohérents avec le contexte). Cherchez les icônes suivantes au fur et à mesure de la lecture :

Cette icône indique un point particulier digne d'intérêt.

Cette icône indique un conseil utile ou une « bonne pratique ».

Cette icône indique un avertissement. Accordez-lui une grande attention pour éviter tout problème.

Organisation de ce livre

Les chapitres qui suivent, ainsi que leur contenu, sont listés ci-dessous :

Chapitre 1, *Notions fondamentales*
: Explique les bases du contrôle de versions et des différents modèles de gestion de versions, ainsi que les dépôts Subversion, les copies de travail et les révisions.

Chapitre 2, *Utilisation de base*
: Une balade dans l'utilisation quotidienne de Subversion. Ce chapitre explique comment récupérer, modifier et propager des données à l'aide du client Subversion.

Chapitre 3, *Sujets avancés*
: Ce chapitre couvre des fonctionnalités plus complexes, que les utilisateurs réguliers seront amenés à manipuler, comme les métadonnées suivies en versions, le verrouillage de fichiers et les piquets de révisions.

Chapitre 4, *Gestion des branches*
: Ce chapitre traite des branches, des fusions et des étiquettes, y compris les bonnes pratiques pour la gestion et la fusion de branches, des cas d'école, comment revenir en arrière sur des modifications et comment passer facilement d'une branche à une autre.

Chapitre 5, *Administration d'un dépôt*
: Ce chapitre décrit les bases d'un dépôt Subversion, comment le créer, le configurer et en assurer la maintenance. Il présente également les outils disponibles pour toutes ces actions.

Chapitre 6, *Configuration du serveur*
: Ce chapitre explique comment configurer votre serveur Subversion et présente différentes manières d'accéder à votre dépôt : `HTTP`, le protocole `svn` et l'accès au disque en local. Il couvre aussi l'authentification, les autorisations et les accès anonymes.

Chapitre 7, *Personnalisation de Subversion*
: Ce chapitre explore les fichiers de configuration du client Subversion, décrit la prise en compte des contenus internationaux et montre comment utiliser des programmes externes conjointement avec Subversion.

Chapitre 8, *Intégration de Subversion*
: Ce chapitre décrit l'architecture interne de Subversion, le système de fichiers associé et les zones administratives des copies de travail, du point de vue du programmeur. Il montre comment utiliser les API publiques pour écrire un programme qui utilise Subversion et, surtout, comment contribuer au développement de Subversion.

Chapitre 9, *Références complètes de Subversion*
: Ce chapitre explique de manière très détaillée chacune des sous-commandes **svn**, **svnadmin** et **svnlook** avec tout un tas d'exemples pour contenter l'ensemble de la famille !

Annexe A, *Guide de démarrage rapide avec Subversion*
: Pour les impatients, l'installation de Subversion et son utilisation en moins de deux minutes chrono. Vous êtes prévenu.

Annexe B, *Guide Subversion à l'usage des utilisateurs de CVS*
: Cette annexe couvre les similitudes et les différences entre Subversion et CVS, avec des suggestions pour perdre les mauvaises habitudes que vous avez acquises durant des années d'utilisation de CVS. Cela comprend les descriptions des numéros de révision de Subversion, les répertoires suivis en versions, les opérations sans connexion réseau, la distinction

entre **status** et **update**, les branches, les étiquettes, les métadonnées, la résolution de conflits et l'authentification.

Annexe C, *WebDAV et la gestion de versions automatique*
: Cette annexe décrit en détail WebDAV et DeltaV ; elle explique comment configurer votre dépôt Subversion pour qu'il puisse être monté en lecture/écriture par des clients DAV.

Annexe D, *Copyright*
: Cette annexe contient une copie de la Licence Creative Commons dont ce livre fait l'objet.

Ce livre est libre

Ce livre est parti de quelques morceaux de documentation écrits par les développeurs du projet Subversion, qui furent alors fusionnés en un seul travail et réécrits. En tant que tel, il a toujours été sous licence libre (cf. l'Annexe D, *Copyright*). En fait, le livre a été écrit sous le regard du public, faisant au départ partie intégrante du projet Subversion. Cela veut dire deux choses :

- Vous trouverez toujours la version la plus récente de ce livre dans le propre dépôt Subversion du livre.
- Vous pouvez modifier ce livre et le redistribuer comme vous le voulez, il est sous licence libre. Votre seule obligation est de conserver correcte l'attribution du copyright aux auteurs d'origine. Bien sûr, nous préférerions que vous envoyiez vos commentaires et vos correctifs à la communauté des développeurs Subversion, plutôt que de distribuer votre version privée de ce livre.

Le portail internet de développement de ce livre, et de la plupart de ses traductions, est accessible à l'adresse : http://code.google.com/p/svnbook/ [1] . Vous y trouverez des liens sur les dernières parutions et les versions étiquetées du livre dans différents formats, ainsi que des instructions pour accéder au dépôt Subversion du livre (où se trouve son code source XML DocBook). Vos réactions sont les bienvenues et même encouragées. Prière de soumettre tous vos commentaires, réclamations et correctifs concernant les sources du livre à `<svnbook-dev@red-bean.com>`.

Remerciements

Ce livre n'aurait pas été possible (ni très utile) si Subversion n'existait pas. Pour cela, les auteurs tiennent à remercier Biran Behlendorf et CollabNet pour avoir vu l'intérêt et avoir osé investir dans un nouveau projet de logiciel libre aussi risqué et ambitieux ; Jim Blandy pour le nom et le design original de Subversion, on t'aime Jim ; et Karl Fogel pour être, surtout, un si bon ami et, ensuite, un grand leader pour la communauté. [2]

Merci à O'Reilly et à nos différents éditeurs : Chuck Toporek, Linda Mui, Tatiana Apandi, Mary Brady et Mary Treseler. Leur patience et leur soutien ont été extraordinaires.

Enfin, nous aimerions remercier un nombre incalculable de personnes qui ont contribué à ce livre par des relectures informelles, des suggestions et des corrections. Bien qu'il ne s'agisse sans doute pas d'une liste complète, ce livre aurait été incomplet et incorrect sans leur aide : Bhuvaneswaran A, David Alber, C. Scott Ananian, David Anderson, Ariel Arjona, Seth Arnold, Jani Averbach, Charles Bailey, Ryan Barrett, Francois Beausoleil, Brian R. Becker, Yves Bergeron, Karl Berry, Jennifer Bevan, Matt Blais, Jim Blandy, Phil Bordelon, Sietse Brouwer, Tom Brown, Zack Brown, Martin Buchholz, Paul Burba, Sean Callan-Hinsvark, Branko Cibej, Archie Cobbs, Jason Cohen, Ryan Cresawn, John R. Daily, Peter Davis, Olivier Davy, Robert P. J. Day, Mo DeJong, Brian Denny, Joe Drew, Markus Dreyer, Nick Duffek, Boris Dusek, Ben Elliston, Justin Erenkrantz, Jens M. Felderhoff, Kyle Ferrio, Shlomi Fish, Julian Foad, Chris Foote, Martin Furter, Vlad Georgescu, Peter Gervai, Dave Gilbert, Eric Gillespie, David Glasser, Marcel Gosselin, Lieven Govaerts, Steve Greenland, Matthew Gregan, Tom Gregory, Maverick Grey, Art Haas, Mark E. Hamilton, Eric Hanchrow, Liam Healy, Malte Helmert, Michael Henderson, Øyvind A. Holm, Greg Hudson, Alexis Huxley, Auke Jilderda, Toby Johnson, Jens B. Jorgensen, Tez Kamihira, David Kimdon, Mark Benedetto King, Robert Kleemann, Erik Kline, Josh Knowles, Andreas J. Koenig, Axel Kollmorgen, Nuutti Kotivuori, Kalin Kozhuharov, Matt Kraai, Regis Kuckaertz, Stefan Kueng, Steve Kunkee, Scott Lamb, Wesley J. Landaker, Benjamin Landsteiner, Vincent Lefevre, Morten Ludvigsen, Dennis Lundberg, Paul Lussier, Bruce A. Mah, Jonathon Mah, Karl Heinz Marbaise, Philip Martin, Feliciano Matias, Neil Mayhew, Patrick Mayweg, Gareth McCaughan, Craig McElroy, Simon McKenna, Christophe Meresse, Jonathan Metillon, Jean-Francois Michaud, Jon Middleton, Robert Moerland, Marcel

[1] NDT: ceci correspond à l'adresse à jour.
[2] Et puis merci, Karl, d'être trop occupé pour rédiger ce livre toi-même.

Molina Jr., Tim Moloney, Alexander Mueller, Tabish Mustufa, Christopher Ness, Roman Neuhauser, Mats Nilsson, Greg Noel, Joe Orton, Eric Paire, Dimitri Papadopoulos-Orfanos, Jerry Peek, Chris Pepper, Amy Lyn Pilato, Kevin Pilch-Bisson, Hans Polak, Dmitriy Popkov, Michael Price, Mark Proctor, Steffen Prohaska, Daniel Rall, Srinivasa Ramanujan, Jack Repenning, Tobias Ringstrom, Jason Robbins, Garrett Rooney, Joel Rosdahl, Christian Sauer, Ryan Schmidt, Jochem Schulenklopper, Jens Seidel, Daniel Shahaf, Larry Shatzer, Danil Shopyrin, Erik Sjoelund, Joey Smith, W. Snyder, Stefan Sperling, Robert Spier, M. S. Sriram, Russell Steicke, David Steinbrunner, Sander Striker, David Summers, Johan Sundstroem, Ed Swierk, John Szakmeister, Arfrever Frehtes Taifersar Arahesis, Robert Tasarz, Michael W. Thelen, Mason Thomas, Erik van der Kolk, Joshua Varner, Eric Wadsworth, Chris Wagner, Colin Watson, Alex Waugh, Chad Whitacre, Andy Whitcroft, Josef Wolf, Luke Worth, Hyrum Wright, Blair Zajac, Florian Zumbiehl, et la communauté Subversion toute entière.

De Ben Collins-Sussman

Merci à ma femme Frances, qui, pendant de longs mois, a entendu :« Mais chérie, je n'ai pas fini de travailler sur le livre », au lieu de l'habituel « Mais chérie, je n'ai pas fini d'envoyer des mails ». Je ne sais pas d'où elle tire toute cette patience ! Elle est mon contrepoids parfait.

Merci à toute ma famille et à tous mes amis pour leurs encouragements sincères, malgré leur absence totale d'intérêt pour le sujet (vous savez, ceux qui disent, « Oh oh, tu écris un livre ? » et qui, lorsque vous leur dites qu'il s'agit d'un livre d'informatique, ne sont plus du tout intéressés).

Merci à tous mes amis proches, qui ont fait de moi un homme très riche. Ne me regardez pas comme ça, vous savez qui vous êtes.

Merci à mes parents pour le formatage bas niveau qui était parfait et pour avoir été d'incroyables modèles. Merci à mon fils pour avoir eu l'opportunité de lui transmettre cela.

De Brian W. Fitzpatrick

Un immense merci à ma femme Marie pour avoir été incroyablement compréhensive, pour m'avoir soutenu et, le plus important, pour avoir été aussi patiente. Merci à toi Éric, mon frère, qui fut le premier à me faire découvrir la programmation UNIX il y a bien longtemps. Merci à ma maman et à ma grand-mère pour tout leur soutien, sans parler de ce Noël où je suis rentré à la maison et ai immédiatement plongé la tête dans mon portable pour travailler sur le livre.

Pour Mike et Ben : cela a été un plaisir de travailler avec vous sur ce livre. Bon sang, c'est un plaisir de travailler avec vous au boulot !

A toute la communauté Subversion et à l'Apache Software Foundation, merci de me considérer comme l'un des vôtres. Il n'y a pas un jour où je n'apprends pas au moins une chose grâce à l'un d'entre vous.

Enfin, merci à mon grand-père qui me disait toujours que « Liberté égale Responsabilité. » Je suis tellement d'accord avec lui.

De C. Michael Pilato

Merci en particulier à Amy, ma meilleure amie et ma femme depuis neuf incroyables années, pour son amour et son soutien patient, pour avoir supporté les nuits tardives et pour avoir gracieusement supporté le processus de gestion de versions que je lui ai fait endurer. Ne t'en fais pas chérie, tu vas devenir un gourou de TortoiseSVN en un rien de temps !

Gavin, tu es maintenant capable de lire par toi-même la moitié des mots de ce livre ; malheureusement, c'est l'autre moitié qui contient les concepts clé. Et désolé Aidan, je n'ai pas réussi à inclure les personnages de Disney/Pixar dans ce texte. Mais Papa vous aime tous les deux et est impatient de vous apprendre à programmer.

Maman et Papa, merci pour votre soutien constant et votre enthousiasme. Belle-Maman et Beau-Papa, merci pour la même chose, *plus* votre fabuleuse fille.

Chapeau bas à Shep Kendall, à travers qui le monde des ordinateurs s'est ouvert à moi pour la première fois ; Ben Collins-Sussman, mon guide dans le monde du logiciel libre ; Karl Fogel, tu *es* mon `.emacs` ; Greg Stein, pour avoir laissé transpirer sa connaissance pratique de la programmation ; Brian FitzPatrick, pour avoir partagé avec moi cette expérience d'écriture. Aux nombreuses personnes de qui je récupère constamment de nouvelles connaissances, continuez à les partager !

Enfin, à Celle, l'unique, qui incarne parfaitement l'excellence créative, merci.

Qu'est-ce que Subversion ?

Subversion est un logiciel libre de gestion de versions. Cela veut dire que Subversion gère les fichiers et les répertoires, ainsi que les changements que vous y apportez au fil du temps. Cela vous permet de revenir à d'anciennes versions de vos données ou d'examiner la façon dont vos données ont évolué. De ce point de vue, beaucoup de gens se représentent un système de gestion de versions comme une sorte de « machine à remonter le temps ».

Subversion peut fonctionner en réseau, ce qui lui permet d'être utilisé par des personnes travaillant sur des ordinateurs différents. D'une certaine manière, la possibilité offerte à plusieurs personnes de modifier et de gérer le même ensemble de données depuis différents sites favorise la collaboration. Les choses progressent plus vite quand on évite d'avoir un canal unique à travers lequel toutes les modifications doivent passer. Et comme les modifications sont suivies en versions, pas d'inquiétude à avoir, l'absence d'un tel canal n'a pas pour contrepartie une perte de qualité : si des changements inadéquats sont appliqués aux données, il suffit de les annuler.

Certains systèmes de gestion de versions sont aussi des systèmes de gestion de configuration logicielle (GCL). Ces systèmes sont spécialement conçus pour gérer des arborescences de code source et possèdent de nombreuses fonctionnalités propres au développement logiciel, comme la reconnaissance des langages de programmation ou des outils de construction/compilation de logiciel. Subversion, cependant, ne fait pas partie de cette catégorie. C'est un système généraliste qui peut être utilisé pour gérer *n'importe quel ensemble* de fichiers. Pour vous ce sera peut-être du code source ; pour d'autres ça ira de de la liste de courses jusqu'aux vidéos des vacances et bien au-delà.

Subversion est-il l'outil approprié ?

Si, en tant qu'utilisateur ou administrateur système, vous réfléchissez à la mise en place de Subversion, la première question à vous poser est :« Est-ce que c'est l'outil adéquat pour ce que je veux faire ? » Subversion est un marteau fantastique, mais il faut faire attention à ne pas assimiler tout problème à un clou.

Si vous avez besoin d'archiver de vieilles versions de vos fichiers et dossiers, éventuellement de les ressusciter, ou d'examiner les journaux détaillant leurs évolutions, Subversion est l'outil idéal pour vous. Si vous avez besoin de travailler sur des documents en collaboration avec d'autres personnes (habituellement via un réseau) et de conserver la trace de qui a apporté quelles modifications, Subversion fera également l'affaire. C'est pourquoi Subversion est souvent utilisé dans des environnements de développement logiciel ; travailler au sein d'une équipe de développement est par nature une activité sociale et Subversion rend très facile la collaboration entre programmeurs. Bien sûr il existe aussi un coût lié à l'utilisation de Subversion en termes d'administration système. Vous devrez gérer un dépôt de données qui stockera les informations ainsi que tout leur historique et vous assurer qu'il est bien sauvegardé. En travaillant au jour le jour avec les données, vous ne pourrez pas copier, déplacer, renommer ou supprimer des fichiers de la façon dont vous le faisiez auparavant. À la place, vous devrez accomplir tout ceci via Subversion.

En supposant que cette quantité de travail supplémentaire ne vous pose pas de problème, vous devriez quand même vérifier que vous n'allez pas utiliser Subversion pour résoudre un problème que d'autres outils pourraient résoudre de manière bien plus efficace. Par exemple, parce que Subversion fournit une copie des données à tous les utilisateurs concernés, une erreur courante est de le traiter comme un système de distribution générique. Les gens utilisent parfois Subversion pour partager d'immenses collections de photos, de musique numérique ou de packs logiciels. Le problème est que ce type de donnée ne change en général jamais. La collection grandit au fil du temps, mais les fichiers individuels à l'intérieur de la collection ne changent pas. Dans ce cas, utiliser Subversion est « disproportionné ». [3] Il existe des outils plus simples, capables de copier des données efficacement *sans* s'embarrasser de toute la gestion du suivi des modifications, tels que **rsync** ou **unison**.

L'histoire de Subversion

Au début des années 2000, CollabNet, Inc.(http://www.collab.net) commença à rechercher des développeurs pour écrire un remplaçant à CVS. CollabNet fournit une suite logicielle collaborative appelée « CollabNet Enterprise Edition (CEE) » dont l'un des composants est la gestion de versions. Même si CEE utilisait CVS comme système de gestion de versions initial, les limitations de celui-ci étaient évidentes depuis le début, et CollabNet savait qu'il lui faudrait au final trouver quelque chose de mieux. Malheureusement, CVS était devenu le standard de fait dans le monde du logiciel libre, essentiellement parce qu'*il n'y avait rien de mieux*, en tout cas sous licence libre. Donc CollabNet décida d'écrire un nouveau système de gestion de versions ex nihilo, en conservant les idées de base de CVS, mais sans ses bogues ni ses limitations fonctionnelles.

En février 2000, CollabNet contacta Karl Fogel, l'auteur de *Open Source Development with CVS* (Coriolis, 1999), et lui demanda s'il aimerait travailler sur ce nouveau projet. Il se trouve qu'au même moment Karl ébauchait la conception d'un

[3] Ou comme le dit un de mes amis, c'est « tuer les mouches à coup de canon ».

nouveau système de gestion de versions avec son ami Jim Blandy. En 1995, ils avaient créé ensemble Cyclic Software, une société fournissant des contrats de support pour CVS, et bien qu'ils aient plus tard revendu la société, ils utilisaient toujours CVS quotidiennement dans leur travail. Leurs frustrations à propos de CVS avaient conduit Jim à élaborer mentalement de meilleures façons de gérer les données suivies en versions. Il avait déjà non seulement trouvé le nom de « Subversion », mais aussi les principes de base du stockage de données de Subversion. Quand CollabNet les appela, Karl accepta immédiatement de travailler sur le projet et Jim obtint de son employeur, Red Hat Software, qu'il le délègue au projet pour une durée indéterminée. CollabNet embaucha Karl et Ben Collins-Sussman, et le travail de conception détaillée commença en mai. Grâce à des coups de pouce efficaces de Brian Behlendorf et Jason Robbins de CollabNet, et de Greg Stein (qui travaillait alors en tant que développeur indépendant, et participait aux spécifications du projet WebDAV/DeltaV), Subversion attira rapidement une communauté de développeurs actifs. Il s'avéra que beaucoup d'entre eux avaient eu les mêmes expériences frustrantes avec CVS, et ils saisirent l'opportunité de pouvoir enfin y faire quelque chose.

L'équipe d'origine se mit d'accord sur quelques objectifs simples. Ils ne voulaient pas inventer de nouvelles méthodes de gestion de versions, ils voulaient juste corriger ce qui n'allait pas dans CVS. Ils décidèrent que Subversion reprendrait les fonctionnalités de CVS et préserverait son modèle de développement, mais ne reproduirait pas ses faiblesses les plus évidentes. Malgré le fait que Subversion devait pouvoir avoir ses propres spécificités, il devait être suffisamment semblable à CVS pour que n'importe lequel de ses utilisateurs puisse facilement passer à Subversion.

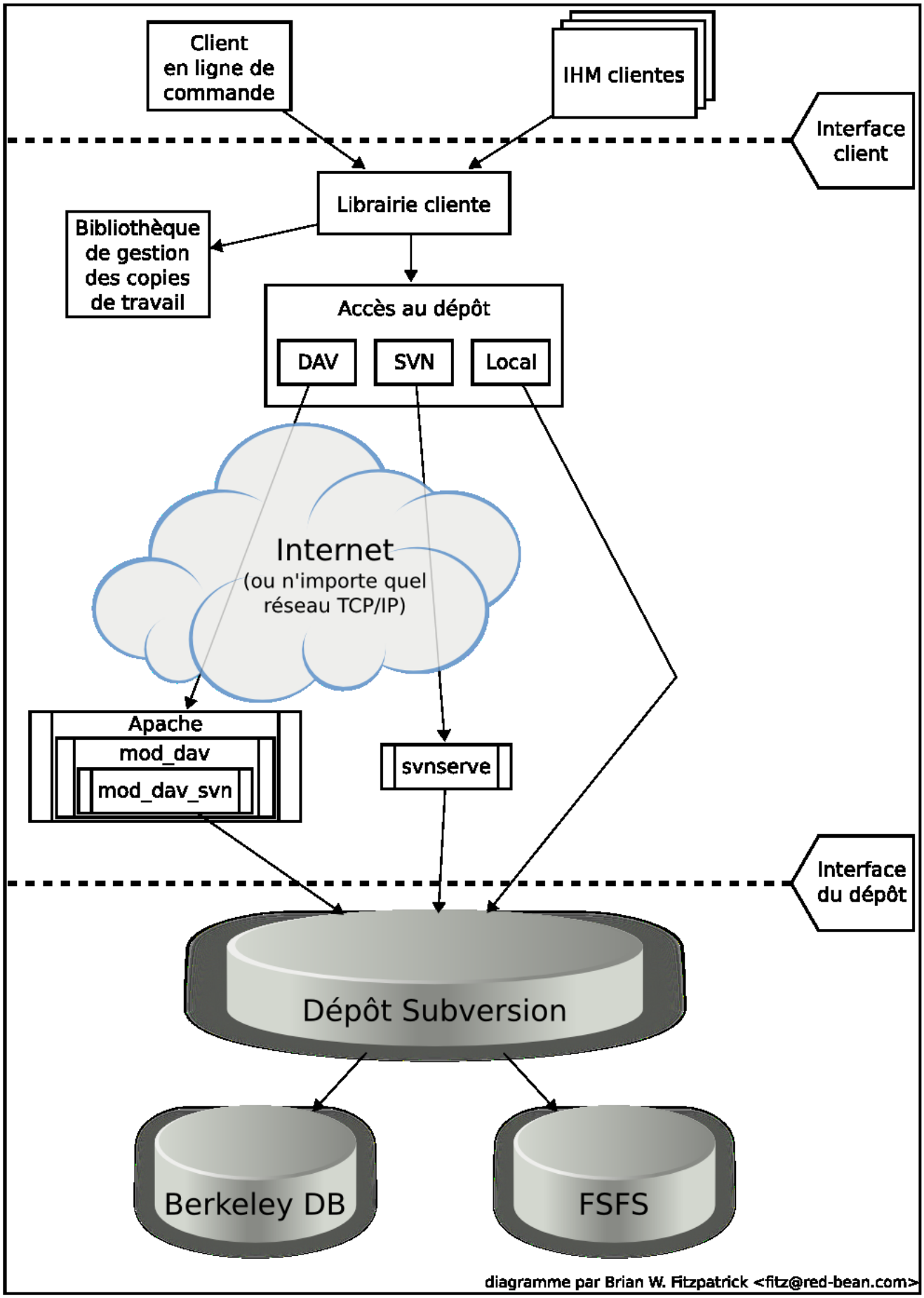
Le 31 août 2001, après 14 mois de codage, Subversion devint « auto-hébergeant ». Ce qui veut dire que les développeurs de Subversion cessèrent d'utiliser CVS pour gérer le propre code source de Subversion, et commencèrent à utiliser Subversion à la place.

Bien que CollabNet ait initié le projet, et qu'ils subventionnent encore une grosse partie du travail (en payant les salaires complets de quelques développeurs de Subversion), Subversion fonctionne comme la plupart des projets de logiciel libre, dirigé par un ensemble de règles vagues et transparentes qui encouragent la méritocratie. La licence CollabNet est parfaitement conforme aux principes du logiciel libre selon Debian (« Debian Free Software Guidelines » en anglais). Autrement dit, chacun est libre de télécharger, modifier et redistribuer Subversion comme il le veut ; aucune autorisation de CollabNet ou de quiconque n'est nécessaire.

L'architecture de Subversion

La Figure 1, « Architecture de Subversion » donne une vue d'ensemble du schéma de conception de Subversion.

Figure 1. Architecture de Subversion


Client
en ligne de
commande
IHM clientes
Interface
client
Librairie cliente
Bibliothèque
de gestion
des copies
de travail
Accès au dépôt
DAV
SVN
Local
Internet
(ou n'importe quel
réseau TCP/IP)
Apache
mod_dav
mod_dav_svn
svnserve
Interface
du dépôt
Dépôt Subversion
Berkeley DB
FSFS
diagramme par Brian W. Fitzpatrick <fitz@red-bean.com>

D'un côté, nous avons un dépôt Subversion qui contient toutes vos données suivies en versions. De l'autre côté, il y a votre programme client Subversion, qui gère des versions locales (appelées « copies de travail ») de certaines de ces données suivies en versions. Entre ces deux extrêmes, il y a des chemins variés utilisant différentes couches d'accès au dépôt. Certains de ces chemins passent par des réseaux informatiques et des serveurs réseau avant d'atteindre le dépôt. D'autres court-circuitent complètement le réseau et accèdent directement au dépôt.

Les composantes de Subversion

Une fois installé, Subversion est constitué de nombreux composants. Ce qui suit est un survol rapide de ce que vous obtenez. Ne vous inquiétez pas si certaines de ces brèves descriptions vous laissent dubitatif ; ce livre contient de *nombreuses* pages destinées à dissiper toute confusion.

svn
: Le programme client en ligne de commande.

svnversion
: Un programme permettant d'examiner l'état d'une copie de travail (en termes de révisions des éléments présents).

svnlook
: Un outil qui permet d'examiner directement un dépôt Subversion.

svnadmin
: Un outil destiné à la création, la modification ou la réparation d'un dépôt Subversion.

mod_dav_svn
: Un greffon pour le serveur HTTP Apache, utilisé pour rendre votre dépôt disponible à d'autres personnes à travers un réseau.

svnserve
: Un serveur autonome créé sur mesure pour Subversion, pouvant fonctionner comme un processus démon ou pouvant être invoqué par SSH ; une autre façon de rendre votre dépôt accessible à d'autres personnes à travers un réseau.

svndumpfilter
: Un programme qui permet de filtrer les flux d'exports de l'historique de vos dépôts.

svnsync
: Un programme capable de synchroniser de manière incrémentale un dépôt avec un autre dépôt à travers un réseau.

Ce qui a changé dans Subversion

La première édition de ce livre a été publiée en 2004, peu après que Subversion ait atteint la version 1.0. Durant les quatre années qui suivirent, cinq nouvelles versions majeures de Subversion sont sorties, corrigeant des bogues et ajoutant de nouvelles fonctionnalités majeures. Bien que nous ayons réussi à tenir à jour la version en ligne de ce livre jusqu'à aujourd'hui, nous sommes ravis que la seconde édition de O'Reilly traite jusqu'à la version 1.5, qui correspond à une étape très importante du projet. Voici un résumé rapide des changements majeurs qui ont eu lieu depuis Subversion 1.0. Cette liste n'est pas exhaustive ; pour tous les détails, merci de vous rendre sur le site web de Subversion à l'adresse http://subversion.tigris.org.

Subversion 1.1 (septembre 2004)
: En version 1.1 fut introduit FSFS qui permet de stocker le dépôt sous forme de fichiers textes. Bien que les bases Berkeley DB soient toujours très utilisées et supportées par la communauté, FSFS est devenu le choix par défaut pour la création de nouveaux dépôts, grâce à sa prise en main facile et à ses besoins minimes en termes de maintenance. Dans cette version ont également été ajoutées les possibilités de suivre en versions des liens symboliques et de prendre en compte automatiquement des URLs, ainsi qu'une interface utilisateur régionalisée.

Subversion 1.2 (mai 2005)
: La version 1.2 introduisit la possibilité de créer des verrous sur les fichiers côté serveur, sérialisant ainsi l'accès des propagations à certaines ressources. Bien que Subversion soit toujours fondamentalement un système de gestion de versions à accès simultanés, certains types de fichiers binaires (par exemple des images de synthèse) ne peuvent pas être fusionnés. Le mécanisme de verrouillage répond aux besoins de suivi en versions et de protection de ces données. Avec le

verrouillage est également apparue une implémentation complète de l'auto-versionnement WebDAV, permettant aux dépôts Subversion d'être accessibles sous la forme de dossiers partagés sur le réseau. Enfin, Subversion 1.2 commença à utiliser un nouvel algorithme plus rapide de différenciation de données binaires pour compresser et récupérer de vieilles versions de fichiers.

Subversion 1.3 (décembre 2005)
: Avec la version 1.3, le serveur svnserve sait contrôler les droits en fonction des chemins, ce qui correspondait à une fonctionnalité existant uniquement à cette époque dans le serveur Apache. Cependant, le serveur Apache bénéficia lui-même de nouvelles fonctionnalités de journalisation et les API de connexion entre Subversion et d'autres langages firent également de grands pas en avant.

Subversion 1.4 (septembre 2006)
: En version 1.4 fut introduit un tout nouvel outil, **svnsync**, permettant la réplication, dans une seule direction, d'un dépôt via le réseau. Des parties importantes des métadonnées des copies de travail changèrent de format afin de ne plus utiliser XML (avec pour conséquence des gains en rapidité côté client), tandis que le gestionnaire de base de données des dépôts Berkeley DB acquit la capacité de rétablir les bases automatiquement suite à un crash du serveur.

Subversion 1.5 (juin 2008)
: Sortir la version 1.5 prit beaucoup plus de temps que les autres versions, mais la fonctionnalité vedette était titanesque : le suivi semi-automatisé des branches et des fusions. Ce fut une véritable bénédiction pour les utilisateurs et propulsa Subversion bien au-delà des possibilités de CVS, le plaçant à la hauteur de ses concurrents commerciaux tels que Perforce et Clearcase. En version 1.5 tout un tas d'autres fonctionnalités axées sur l'utilisateur furent introduites, telles que la résolution interactive des conflits entre fichiers, les extractions partielles, la gestion des listes de modifications côté client, une nouvelle syntaxe très puissante pour les définitions externes et le support par le serveur **svnserve** de l'authentification par SASL.

Chapitre 1. Notions fondamentales

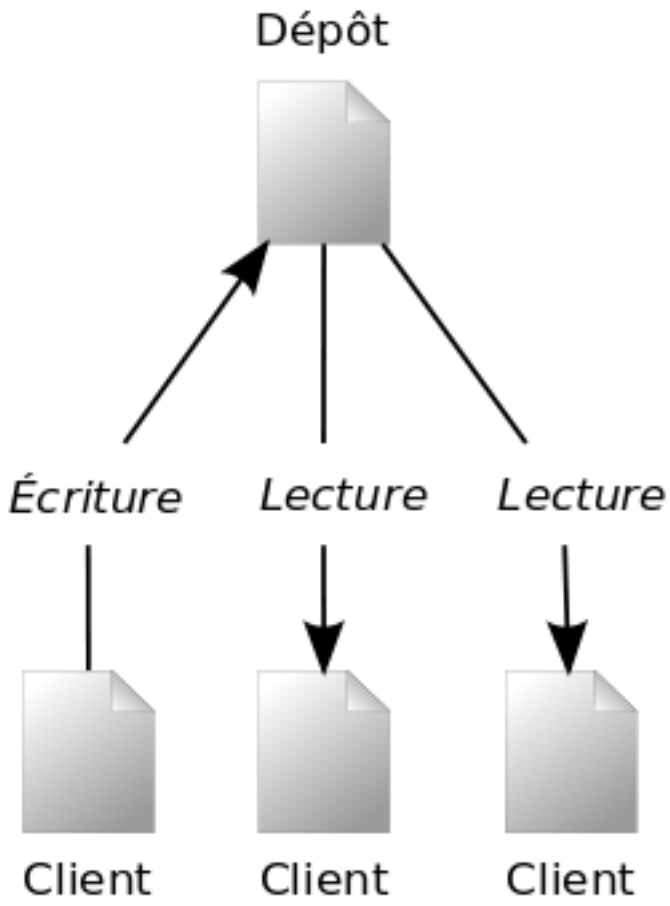
Ce chapitre est une introduction rapide à Subversion. Si vous ne connaissez rien à la gestion de versions, ce chapitre est à coup sûr pour vous. Nous allons commencer par une présentation des notions générales de la gestion de versions, puis étudier plus précisément les idées particulières qui se cachent derrière Subversion et enfin donner quelques exemples simples d'utilisation de Subversion.

Même si les exemples de ce chapitre mettent en scène des personnes partageant du code source, gardez à l'esprit que Subversion peut gérer n'importe quel type d'ensemble de fichiers, il n'est pas réservé aux programmeurs.

Le dépôt

Subversion est un système centralisé fait pour partager l'information. Le dépôt constitue le cœur de ce système, en tant que lieu de stockage central des données. Les informations y sont organisées sous la forme d'une *arborescence de fichiers*, c'est-à-dire une hiérarchie classique de fichiers et de répertoires. Un certain nombre de *clients* se connectent au dépôt, et parcourent ou modifient ces fichiers. En modifiant des données, un client rend ces informations disponibles à d'autres personnes ; en lisant des données, le client reçoit les informations des autres personnes. La Figure 1.1, « Un authentique système client/serveur » illustre cela.

Figure 1.1. Un authentique système client/serveur



Quel est l'intérêt ? Jusque là, cela ressemble à la définition d'un serveur de fichiers classique. En fait, le dépôt *est* bien une sorte de serveur de fichiers, mais d'un type particulier. Ce qui rend le dépôt Subversion spécial, c'est qu'*il se souvient de toutes les modifications* qui ont été apportées : chaque modification de chaque fichier, ainsi que les modifications de l'arborescence-même des répertoires, comme l'ajout, la suppression ou la réorganisation de fichiers et de répertoires.

Quand un client parcourt le dépôt, il consulte généralement la dernière version de l'arborescence du système de fichiers. Mais le client est également capable de visualiser des états *antérieurs* du système de fichiers. Par exemple, un client peut poser des questions concernant l'historique des données, comme « Que contenait ce répertoire mercredi dernier ? » ou « Quelle est la dernière personne qui a modifié ce fichier, et quels changements a-t-elle effectué ? ». C'est le genre de questions qui est au cœur de tout *logiciel de gestion de versions*, logiciel conçu pour conserver l'historique des modifications des données au cours du temps.

Modèles de gestion de versions

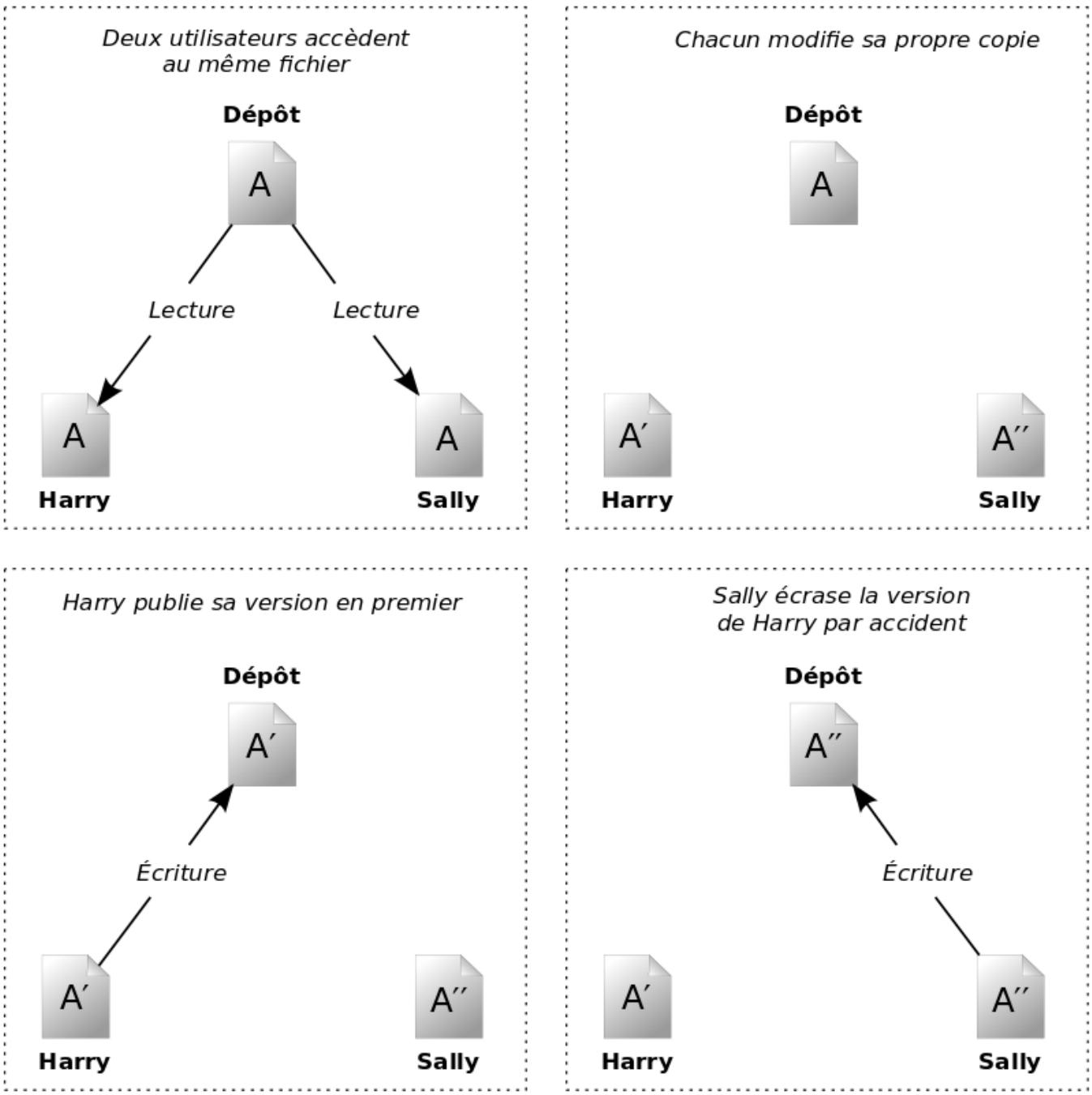
La mission essentielle d'un logiciel de gestion de versions est de permettre l'édition collaborative et le partage de données. Mais il existe différentes stratégies pour arriver à cette fin. Comprendre ces différentes stratégies est important pour plusieurs raisons. Tout d'abord, cela vous aidera à comparer et différencier les logiciels de gestion de versions existants, au cas où vous rencontriez d'autres logiciels similaires à Subversion. Ensuite, cela vous aidera également à utiliser plus efficacement Subversion, puisque Subversion lui-même autorise différentes façons de travailler.

Problématique du partage de fichiers

Tous les logiciels de gestion de versions doivent résoudre le même problème fondamental : comment le logiciel va-t-il permettre aux utilisateurs de partager l'information, tout en les empêchant de se marcher mutuellement sur les pieds par accident ? Il est vraiment trop facile pour les utilisateurs d'écraser malencontreusement les changements effectués par d'autres dans le dépôt.

Observons le scénario décrit à la Figure 1.2, « La situation à éviter ». Supposons que nous ayons deux collaborateurs, Harry et Sally. Ils décident tous les deux d'éditer au même moment le même fichier dans le dépôt. Si Harry sauvegarde ses modifications dans le dépôt en premier, il est possible que, quelques instants plus tard, Sally les écrase avec sa propre version du fichier. Bien que la version de Harry ne soit pas perdue pour toujours, car le système se souvient de tous les changements, *aucune* des modifications effectuées par Harry ne sera présente dans la nouvelle version du fichier de Sally, car elle n'aura jamais vu les changements réalisés par Harry. De fait, le travail de Harry est perdu ou, du moins, perdu dans la version finale du fichier, et ceci probablement par accident. Il s'agit précisément d'une situation que nous voulons à tout prix éviter !

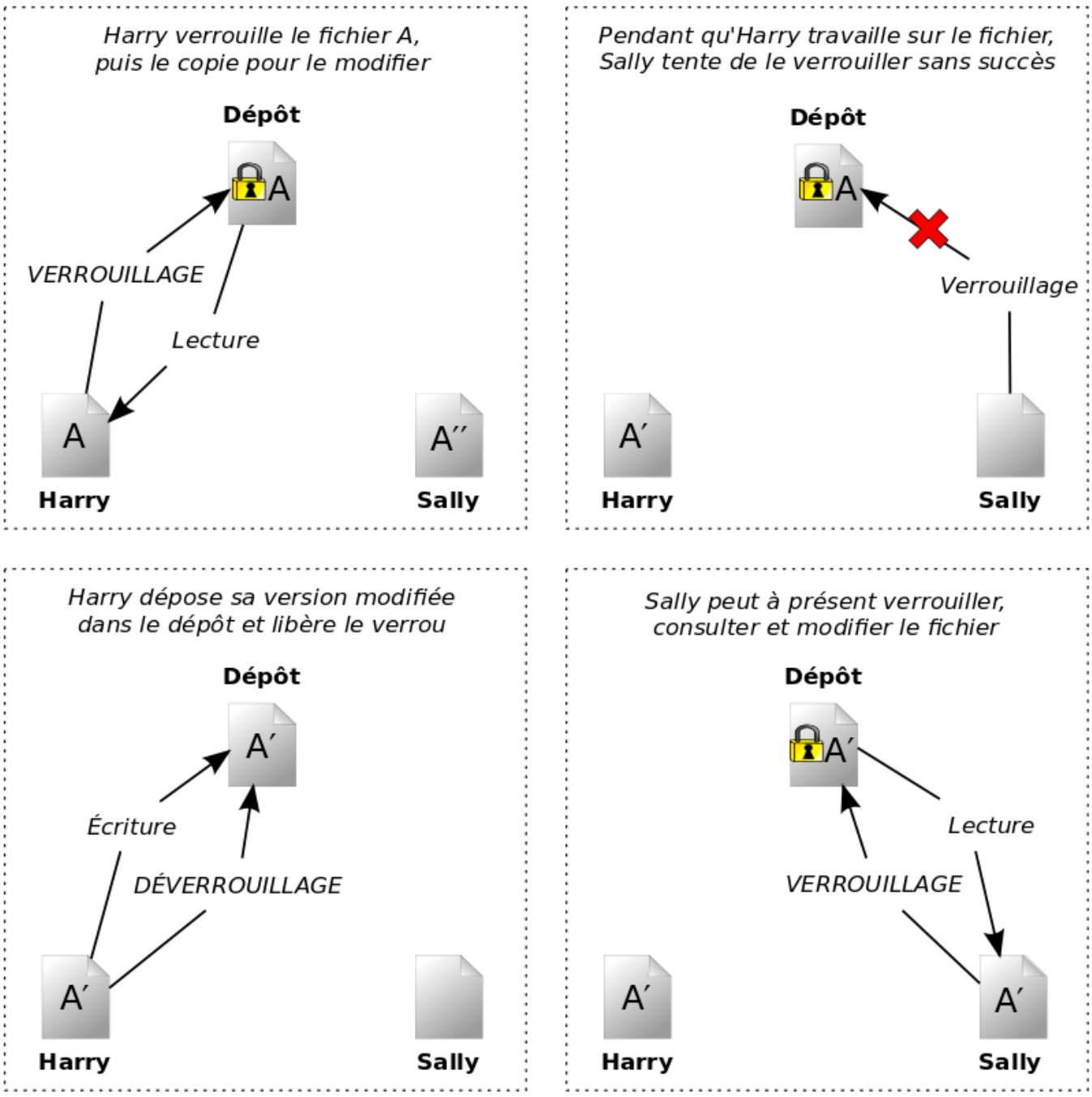
Figure 1.2. La situation à éviter



Modèle verrouiller-modifier-libérer

De nombreux logiciels de gestion de versions utilisent le modèle *verrouiller-modifier-libérer* pour résoudre le problème de plusieurs auteurs annihilant le travail des autres. Dans ce modèle, le dépôt ne permet qu'à une seule personne de modifier un fichier à un instant donné. Cette politique exclusive est gérée grâce à des verrous (« lock » en anglais). Harry doit « verrouiller » un fichier avant de commencer à le modifier. Si Harry a verrouillé un fichier, alors Sally ne pourra pas le verrouiller et ne pourra donc faire aucun changement dessus. Tout ce qu'elle pourra faire sera de lire le fichier et d'attendre que Harry ait fini ses changements puis libéré le verrou. Après que Harry ait libéré le fichier, Sally pourra à son tour le verrouiller et l'éditer. La Figure 1.3, « Modèle verrouiller-modifier-libérer » illustre cette solution très simple.

Figure 1.3. Modèle verrouiller-modifier-libérer



Le problème avec le modèle verrouiller-modifier-libérer est qu'il est relativement restrictif et devient souvent un barrage pour les utilisateurs :

- *Le verrouillage peut créer des problèmes d'administration.* Parfois, Harry va verrouiller un fichier et oublier qu'il l'a fait. Pendant ce temps, Sally, qui est encore en train d'attendre pour éditer le fichier, est bloquée. Puis Harry part en vacances. Sally doit alors aller trouver un administrateur pour libérer le verrou de Harry. La situation finit par générer beaucoup de délais inutiles et de temps perdu.

- *Le verrouillage peut créer une sérialisation inutile.* Que se passe-t-il lorsque Harry veut éditer le début d'un fichier texte et que Sally veut simplement éditer la fin de ce même fichier ? Ces changements ne se chevauchent pas du tout. Ils pourraient aisément éditer le fichier simultanément et il n'y aurait pas beaucoup de dégâts, en supposant que les changements soient correctement fusionnés. Dans cette situation, il n'est pas nécessaire de les forcer à éditer le fichier chacun à leur tour.

- *Le verrouillage peut créer un faux sentiment de sécurité.* Supposons que Harry verrouille et édite le fichier A, alors qu'au

même moment Sally verrouille et édite le fichier B. Que se passe-t-il si A et B dépendent l'un de l'autre et que les changements faits à chacun sont incompatibles d'un point de vue sémantique ? A et B ne fonctionnent soudainement plus ensemble. Le système de verrouillage a été incapable d'empêcher ce problème, bien qu'il ait d'une certaine manière instillé un faux sentiment de sécurité. Il est facile pour Harry et Sally d'imaginer qu'en verrouillant les fichiers, chacun commence une tâche isolée, sans danger et donc que ce n'est pas la peine de discuter à l'avance de leurs modifications incompatibles. Verrouiller devient souvent un substitut à une réelle communication.

Modèle copier-modifier-fusionner

Subversion, CVS et beaucoup d'autres logiciels de gestion de versions utilisent le modèle *copier-modifier-fusionner* comme alternative au verrouillage. Dans ce modèle, chaque utilisateur contacte le dépôt du projet via son client et crée une copie de travail personnelle, une sorte de version locale des fichiers et répertoires du dépôt. Les utilisateurs peuvent alors travailler, simultanément et indépendamment les uns des autres, et modifier leurs copies privées. Pour finir, les copies privées sont fusionnées au sein d'une nouvelle version finale. Le logiciel de gestion de versions fournit de l'aide afin de réaliser cette fusion, mais au final la responsabilité de s'assurer que tout se passe bien incombe à un être humain.

Voici un exemple. Supposons que Harry et Sally aient créé chacun des copies de travail du même projet, copiées à partir du dépôt. Ils travaillent simultanément et effectuent sur leur copie des modifications du même fichier A. Sally sauvegarde ses changements dans le dépôt en premier. Lorsque Harry essaie par la suite de sauvegarder ses modifications, le dépôt l'informe que son fichier A est *périmé*. En d'autres termes, le fichier A du dépôt a changé, d'une façon ou d'une autre, depuis la dernière fois qu'il l'avait copié. Harry demande donc à son client de *fusionner* tous les changements en provenance du dépôt dans sa copie de travail du fichier A. Il y a des chances que les modifications de Sally n'empiètent pas sur les siennes ; une fois qu'il a intégré les changements provenant des deux côtés, il sauvegarde sa copie de travail dans le dépôt. La Figure 1.4, « Modèle copier-modifier-fusionner » et la Figure 1.5, « Modèle copier-modifier-fusionner (suite) » illustrent ce processus.

Figure 1.4. Modèle copier-modifier-fusionner

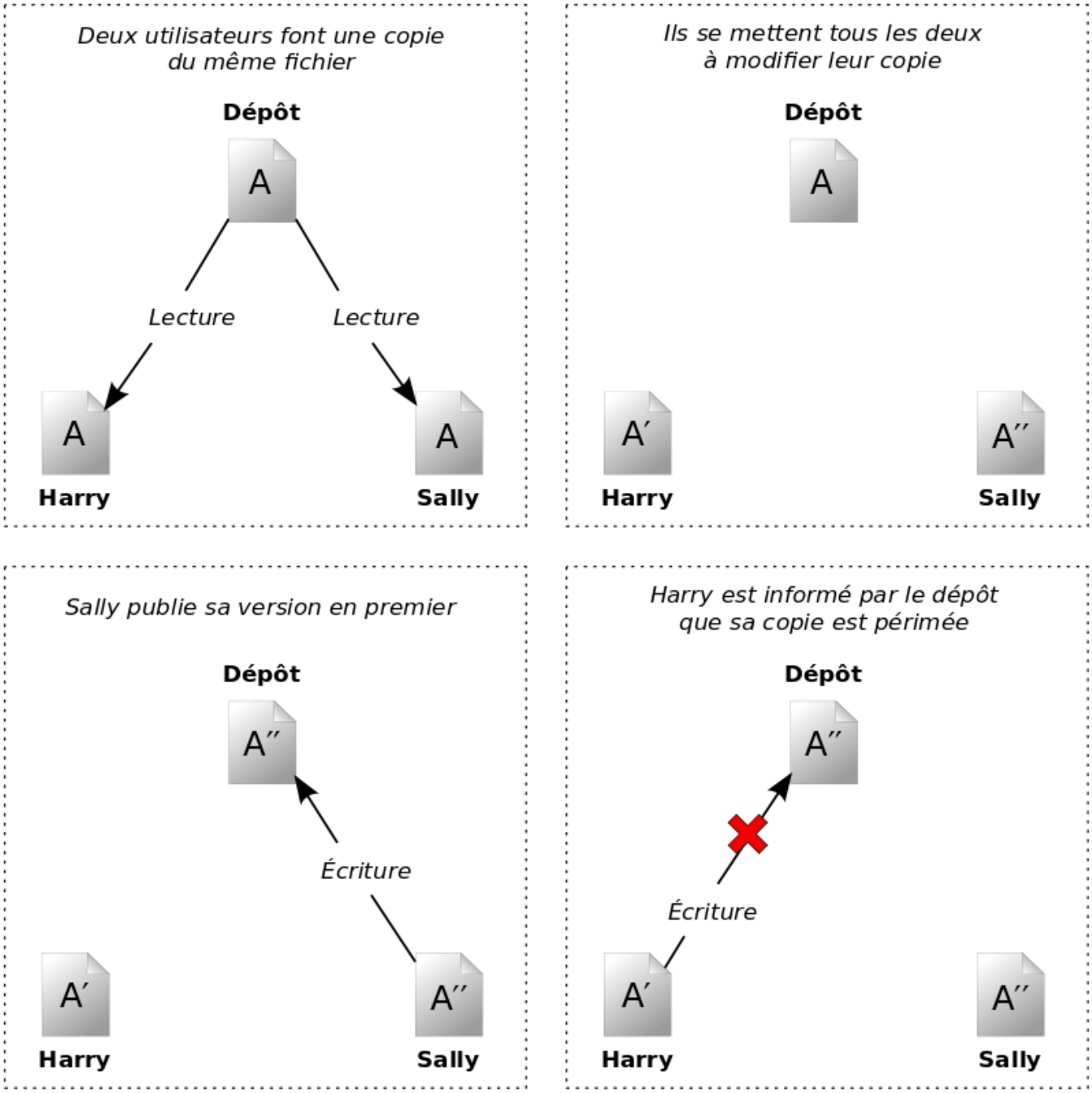
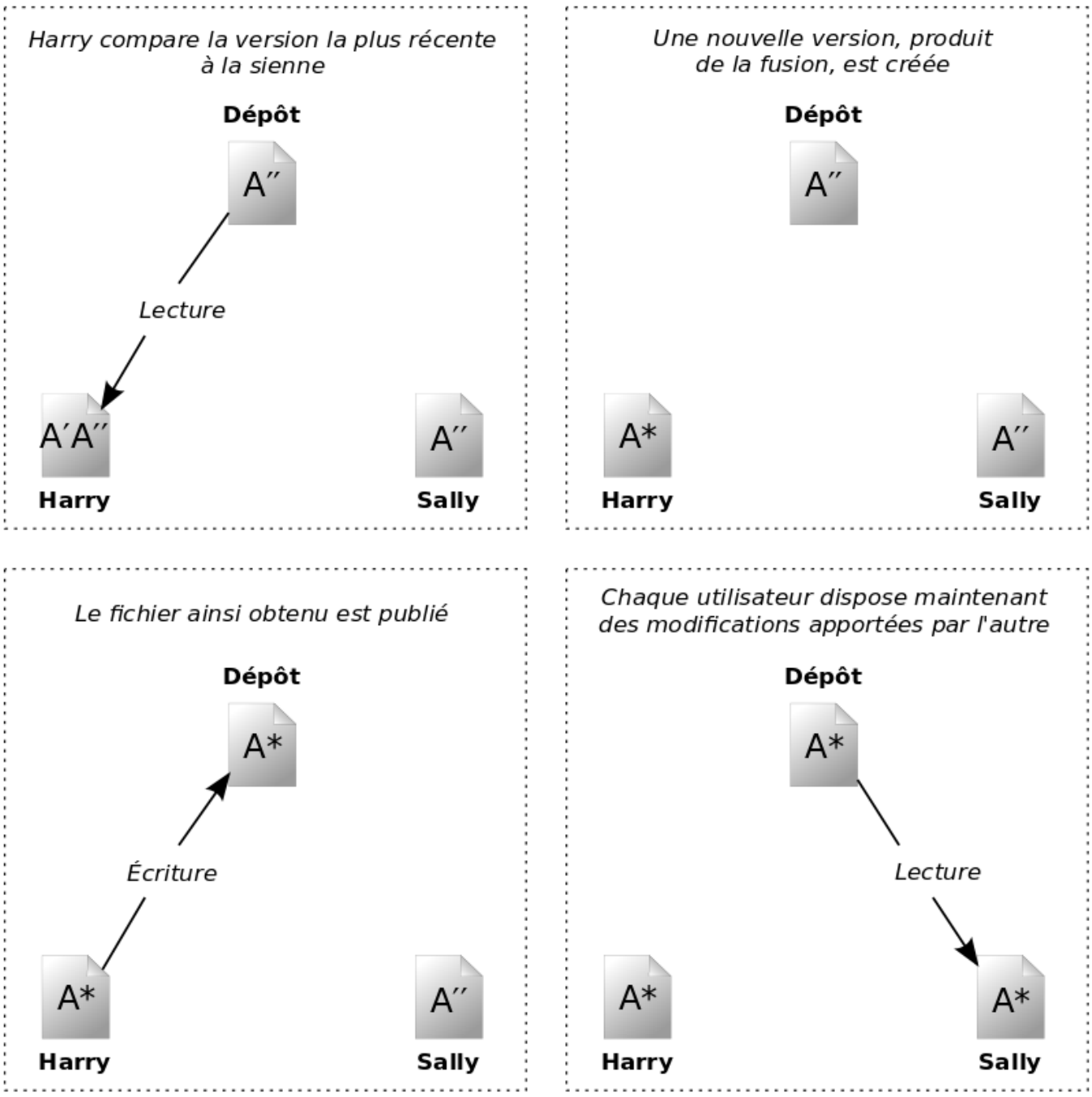


Figure 1.5. Modèle copier-modifier-fusionner (suite)



Mais que se passe-t-il quand les modifications de Sally empiètent sur celles de Harry ? Que fait-on dans ce cas-là ? Cette situation est appelée un *conflit* et ne constitue pas, en général, un gros problème. Lorsque Harry demande à son logiciel client de fusionner les changements les plus récents du dépôt dans sa copie de travail, sa copie du fichier est en quelque sorte marquée comme étant dans un état de conflit : il a la possibilité de voir les deux ensembles de changements entrant en conflit et de choisir manuellement entre les deux. Notez bien qu'un logiciel ne peut pas résoudre automatiquement les conflits ; seuls les humains sont capables de comprendre et de faire les choix intelligents nécessaires. Une fois que Harry a manuellement résolu les modifications se chevauchant, par exemple après une discussion avec Sally, il peut sauvegarder le fichier fusionné en toute sécurité dans le dépôt.

Le modèle copier-modifier-fusionner peut sembler un peu chaotique mais, en pratique, il fonctionne de façon très fluide. Les utilisateurs peuvent travailler en parallèle, sans jamais devoir s'attendre les uns les autres. Lorsqu'ils travaillent sur les mêmes fichiers, il s'avère que la plupart des changements réalisés en parallèle ne se chevauchent pas du tout ; les conflits sont rares. Et le temps nécessaire à la résolution des conflits est en général bien inférieur au temps gaspillé par un système de verrouillage.

Au final, tout revient à un facteur critique : la communication entre les utilisateurs. Lorsque les utilisateurs communiquent mal,

les conflits syntaxiques et sémantiques augmentent. Aucun système ne peut forcer les utilisateurs à communiquer parfaitement et aucun système ne peut détecter les conflits sémantiques. Il n'y a donc aucun intérêt à se laisser endormir par un faux sentiment de sécurité selon lequel un système de verrouillage permettrait d'éviter les conflits ; en pratique, le verrouillage semble limiter la productivité plus qu'aucun autre facteur.

Le verrouillage est parfois nécessaire

Même si le modèle verrouiller-modifier-libérer est en général considéré comme pénalisant pour la collaboration, il y a quand même des cas où le verrouillage est approprié.

Le modèle copier-modifier-fusionner est basé sur l'hypothèse que les fichiers sont contextuellement fusionnables, c'est-à-dire que la majorité des fichiers d'un dépôt sont des fichiers textes (comme le code source d'un programme). Mais pour les fichiers binaires, tels que des images ou du son, il est souvent impossible de fusionner les modifications en conflit. Dans ces cas-là, il est réellement nécessaire que les utilisateurs ne modifient le fichier qu'à tour de rôle. Sans accès sérialisé, quelqu'un finirait par perdre du temps sur des modifications qui seraient finalement perdues.

Bien que Subversion soit avant tout un système copier-modifier-fusionner, il reconnaît toutefois la nécessité du verrouillage pour certains fichiers et fournit donc un mécanisme pour cela. Cette fonctionnalité est traitée plus tard dans ce livre, dans la section intitulée « Verrouillage ».

Subversion en action

Il est temps de passer de l'abstrait au concret. Dans cette section, nous vous montrons des exemples réels d'utilisation de Subversion.

URL des dépôts Subversion

Tout au long de ce livre, Subversion utilise des URL pour identifier des fichiers et des répertoires suivis en version au sein de dépôts Subversion. Pour la plupart, ces URL utilisent la syntaxe standard, permettant de spécifier les noms des serveurs et les numéros de port à l'intérieur même de l'URL :

```
$ svn checkout http://svn.exemple.com:9834/depot
…
```

Mais il existe quelques nuances dans la gestion des URL par Subversion qui doivent être notées. Par exemple, les URL ayant pour méthode d'accès `file://` (utilisée pour les dépôts locaux) doivent posséder, en accord avec les conventions, soit un nom de serveur `localhost` , soit pas de nom de serveur du tout :

```
$ svn checkout file:///var/svn/depot
…
$ svn checkout file://localhost/var/svn/depot
…
```

D'autre part, les utilisateurs du procédé `file://` sur les plateformes Windows doivent se servir d'une syntaxe qui est un « standard » officieux pour accéder à leurs dépôts se trouvant sur la même machine mais sur un disque différent du disque de travail habituel du client. Les deux syntaxes de chemin d'URL suivantes fonctionnent, `X` étant le disque sur lequel le dépôt se trouve :

```
C:\> svn checkout file:///X:/var/svn/depot
…
C:\> svn checkout "file:///X|/var/svn/depot"
…
```

Dans la seconde syntaxe, vous devez entourer l'URL de guillemets pour éviter que la barre verticale ne soit interprétée comme un symbole de redirection (un « pipe »). De plus, remarquez qu'une URL utilise des barres obliques (`/`) alors que la forme

native (non-URL) d'un chemin sous Windows utilise des barres obliques inversées (\).

Les URL Subversion `file://` ne peuvent pas être utilisées dans un navigateur web classique de la même façon qu'une URL `file://` habituelle. Lorsque vous essayez de visualiser une URL `file://` dans un navigateur web classique, il lit et affiche le contenu du fichier situé à cet emplacement en interrogeant directement le système de fichiers. Cependant, les ressources de Subversion existent dans un système de fichier virtuel (cf. la section intitulée « Couche dépôt ») et votre navigateur ne comprend pas comment interagir avec ce système de fichiers.

Enfin, il faut noter que le client Subversion encode automatiquement les URL en cas de besoin, exactement comme le fait un navigateur web. Par exemple, si une URL contient un espace ou un caractère ASCII spécial, comme dans ce qui suit :

```
$ svn checkout "http://hote/chemin avec espace/projet/españa"
```

alors Subversion banalise les caractères spéciaux et se comporte comme si vous aviez tapé :

```
$ svn checkout http://hote/chemin%20avec%20espace/projet/espa%C3%B1a
```

Si l'URL contient des espaces, prenez bien soin de la placer entre guillemets, pour que votre shell traite le tout comme un unique argument du programme **svn**.

URL du dépôt

On peut accéder aux dépôts Subversion de nombreuses manières différentes, sur un disque local ou à travers différents protocoles réseau, en fonction de la façon dont votre administrateur a mis les choses en place pour vous. L'emplacement d'un dépôt, toutefois, est toujours une URL. Le Tableau 1.1, « URL d'accès au dépôt » décrit les différents procédés d'accès et les méthodes d'accès correspondantes.

Tableau 1.1. URL d'accès au dépôt

Procédé	Méthode d'accès
`file:///`	Accès direct au dépôt (sur un disque local).
`http://`	Accès via le protocole WebDAV à un serveur Apache configuré pour Subversion.
`https://`	Identique à `http://`, mais avec chiffrement SSL.
`svn://`	Accès via un protocole personnalisé à un serveur `svnserve`.
`svn+ssh://`	Identique à `svn://`, mais à travers un tunnel SSH.

Pour plus d'informations sur la façon dont Subversion analyse les URL, reportez-vous à la section intitulée « URL des dépôts Subversion ». Pour plus d'informations sur les différents types de serveurs réseau disponibles pour Subversion, reportez-vous au Chapitre 6, *Configuration du serveur*.

Copies de travail

Vous avez déjà découvert ce que sont les copies de travail ; nous allons maintenant vous expliquer comment le client Subversion les crée et les utilise.

Une copie de travail Subversion est une arborescence classique de répertoires de votre système local, contenant un ensemble de fichiers. Vous pouvez éditer ces fichiers comme vous le voulez et, s'il s'agit de code source, vous pouvez compiler votre programme à partir de ceux-ci de la façon habituelle. Votre copie de travail est votre espace de travail personnel privé : Subversion n'y incorporera jamais les changements d'autres personnes ni ne rendra jamais disponibles vos propres changements à d'autres personnes tant que vous ne lui demanderez pas explicitement de le faire. Vous pouvez même avoir

plusieurs copies de travail d'un même projet.

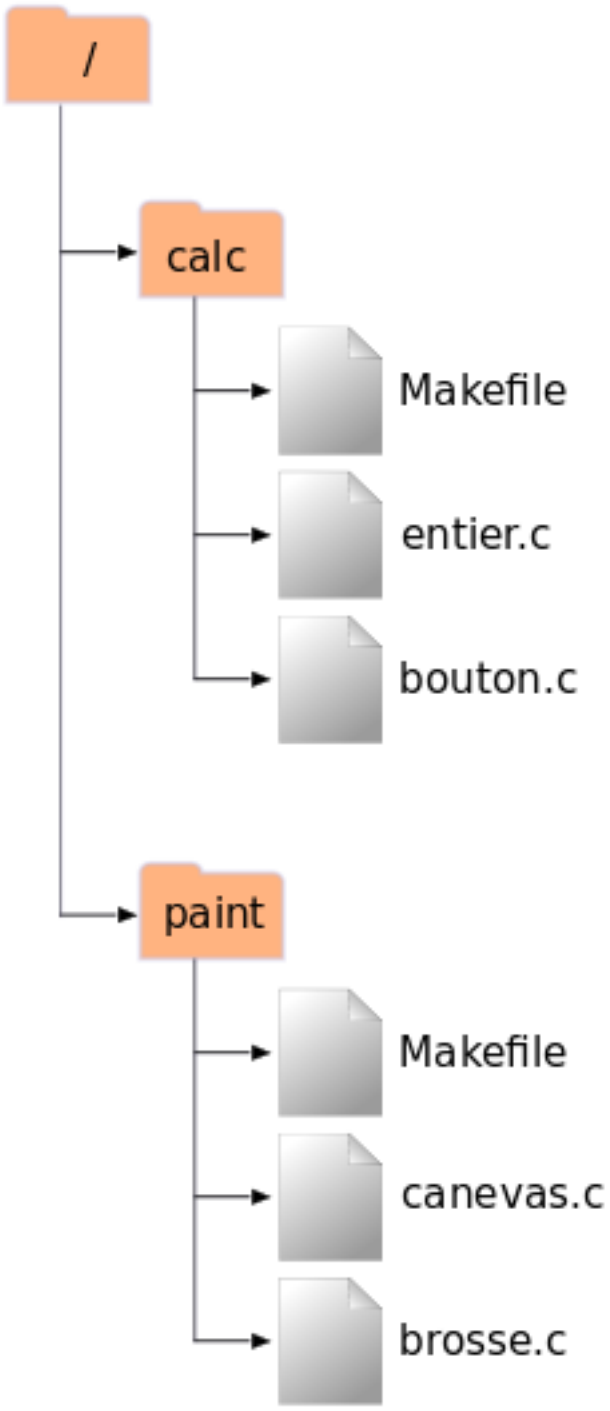
Après que vous ayez apporté quelques modifications aux fichiers de votre copie de travail et vérifié qu'elles fonctionnent correctement, Subversion vous fournit des commandes pour « publier » vos changements vers les autres personnes qui travaillent avec vous sur votre projet (en les transmettant au dépôt). Si d'autres personnes publient leurs propres modifications, Subversion vous fournit des commandes pour fusionner ces changements dans votre copie de travail (en les obtenant du dépôt).

Une copie de travail contient également quelques fichiers supplémentaires, créés et gérés par Subversion, pour l'aider à effectuer ces opérations. En particulier, chaque répertoire de votre copie de travail contient un sous-répertoire appelé `.svn`, aussi appelé *répertoire administratif* de votre copie de travail. Les fichiers de chacun de ces répertoires administratifs permettent à Subversion d'identifier quels fichiers contiennent des modifications non-publiées et quels fichiers sont périmés vis-à-vis du travail des autres personnes.

Un dépôt Subversion contient bien souvent les fichiers (ou code source) de plusieurs projets ; habituellement, chaque projet est un sous-répertoire de l'arborescence du système de fichiers du dépôt. Dans cette situation, la copie de travail d'un utilisateur correspond à une sous-arborescence particulière du dépôt.

Par exemple, supposons que votre dépôt contienne deux projets logiciels, `paint` et `calc`. Chaque projet réside dans son propre sous-répertoire racine, comme indiqué dans la Figure 1.6, « Système de fichiers du dépôt ».

Figure 1.6. Système de fichiers du dépôt



Pour obtenir une copie de travail, vous devez *extraire* une sous-arborescence du répertoire (le terme « extraire », « check out » en anglais, peut vous faire penser que cela a quelque chose à voir avec verrouiller ou réserver des ressources, mais ce n'est pas le cas ; cela crée simplement pour vous une copie privée du projet). Par exemple, si vous extrayez `/calc`, vous obtenez une copie de travail qui ressemble à ceci :

```
$ svn checkout http://svn.exemple.com/depot/calc
A    calc/Makefile
A    calc/entier.c
A    calc/bouton.c
```

```
Révision 56 extraite.

$ ls -A calc
Makefile  entier.c  bouton.c  .svn/
```

Les lettres `A` qui s'affichent dans la marge de gauche indiquent que Subversion est en train d'ajouter des éléments dans votre copie de travail. Vous avez désormais votre copie personnelle du répertoire `/calc` du dépôt, avec une entrée supplémentaire, `.svn`, qui contient des informations complémentaires nécessaires à Subversion, comme évoqué précédemment.

Supposons que vous fassiez des modifications à `bouton.c`. Comme le répertoire `.svn` se souvient de la date de modification et du contenu du fichier original, Subversion peut en déduire que vous avez modifié le fichier. Néanmoins, Subversion ne rend pas vos modifications publiques tant que vous ne lui dites pas de le faire. L'action de publication de vos modifications est plus communément appelée *propagation* (« commit » ou « check in » en anglais et, parfois, *archivage* ou *livraison* en français) des modifications au sein du dépôt.

Pour rendre publiques vos modifications, vous pouvez utiliser la commande Subversion **svn commit** :

```
$ svn commit bouton.c -m "Coquille corrigée dans bouton.c."
Ajout       bouton.c
Transmission des données .
Révision 57 propagée.
```

À présent, vos modifications de `bouton.c` ont été propagées au sein du dépôt, avec un commentaire décrivant ces changements (« vous avez corrigé une coquille »). Si un autre utilisateur extrait une copie de travail de `/calc/`, il va voir vos modifications dans la dernière version du fichier.

Supposons que vous ayez une collaboratrice, Sally, qui a extrait une copie de travail de `/calc` en même temps que vous. Lorsque vous propagez votre modification de `bouton.c`, la copie de travail de Sally reste inchangée ; Subversion ne modifie les copies de travail qu'à la demande des utilisateurs.

Pour mettre son projet à jour, Sally peut demander à Subversion de mettre à jour (« update » en anglais) sa copie de travail, en utilisant la commande **svn update**. Cela va intégrer vos modifications dans sa copie de travail, ainsi que celles qui ont été envoyées par d'autres personnes depuis qu'elle l'avait extraite.

```
$ pwd
/home/sally/calc

$ ls -A
Makefile bouton.c entier.c .svn/

$ svn update
U    bouton.c
Actualisé à la révision 57.
```

En sortie, la commande **svn update** indique que Subversion a mis à jour le contenu de `bouton.c`. Remarquez que Sally n'a pas eu besoin de spécifier quels fichiers devaient être mis à jour ; Subversion utilise les informations contenues dans le répertoire `.svn`, ainsi que d'autres informations en provenance du dépôt, pour décider quels fichiers doivent être mis à jour.

Révisions

Une opération **svn commit** publie les modifications d'un nombre quelconque de fichiers et de répertoires en une seule opération atomique. Dans votre copie de travail, vous pouvez modifier le contenu des fichiers : créer, supprimer, renommer et copier fichiers et répertoires ; puis propager un ensemble de modifications en une seule transaction atomique.

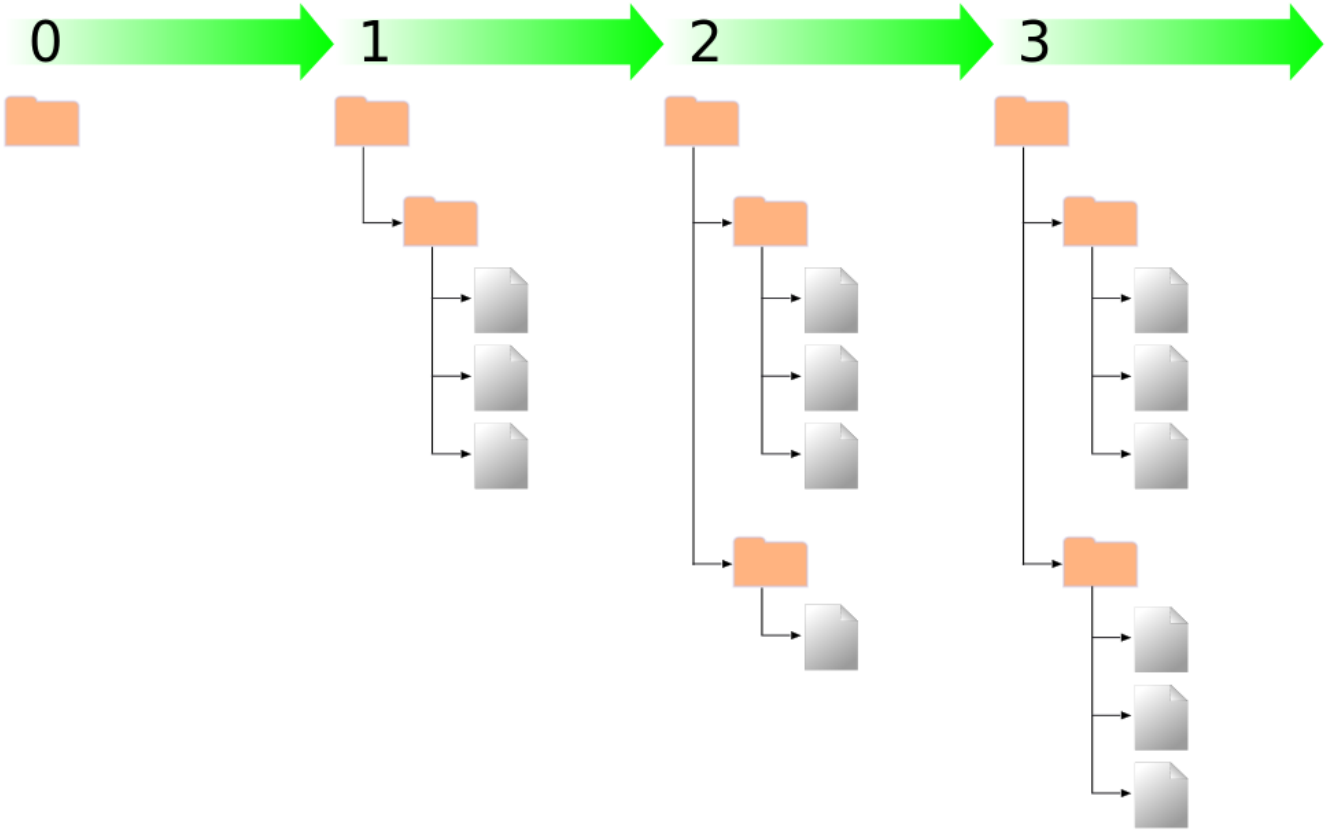
Par « transaction atomique », on entend simplement ceci : soit toutes les modifications sont propagées dans le dépôt, soit aucune ne l'est. Subversion tente de conserver cette atomicité aussi bien face à des « plantages » de programmes, de systèmes d'exploitation ou de réseau, que face aux actions des autres utilisateurs.

Chaque fois que le dépôt accepte une propagation, ceci crée un nouvel état de l'arborescence du système de fichiers, appelé *révision*. Un numéro unique est associé à chaque révision, correspondant au numéro de la révision précédente augmenté de 1.

La révision initiale d'un dépôt fraîchement créé porte le numéro 0 et ne consiste en rien d'autre qu'un répertoire racine vide.

La Figure 1.7, « Le dépôt » offre une vue intéressante du dépôt. Imaginez un tableau de numéros de révisions, commençant à 0 et s'étirant de la gauche vers la droite. Chaque numéro de révision correspond à une arborescence de système de fichiers située en-dessous de lui et chaque arborescence est une photo, un « instantané » (« snapshot » en anglais) du dépôt prise après une propagation.

Figure 1.7. Le dépôt



Numéros de révision globaux

Contrairement à la plupart des logiciels de gestion de versions, les numéros de révision de Subversion s'appliquent à *l'arborescence toute entière* et non à chaque fichier individuellement. À chaque numéro de révision correspond une arborescence toute entière, un état particulier du dépôt après une propagation. Une autre façon de voir cela est de considérer que la révision N représente l'état du système de fichiers du dépôt après la N-ième propagation. Quand des utilisateurs de Subversion parlent de la « révision 5 de `truc.c` », ils veulent en fait parler de « `truc.c` tel qu'il apparaît dans la révision 5 ». Remarquez bien qu'en règle générale, les révisions N et M d'un fichier ne sont *pas forcément* différentes ! De nombreux autres logiciels de gestion de versions gèrent les numéros de révision fichier par fichier ; ce concept peut donc sembler inhabituel à première vue (les anciens utilisateurs de CVS peuvent se référer à l'Annexe B, *Guide Subversion à l'usage des utilisateurs de CVS* pour plus de détails).

Il est important de noter que les copies de travail ne correspondent pas toujours à une unique révision du dépôt ; elles peuvent contenir des fichiers provenant de plusieurs révisions différentes. Par exemple, supposons que vous extrayiez une copie de travail d'un dépôt dont la révision la plus récente est la numéro 4 :

```
calc/Makefile:4
     integer.c:4
     button.c:4
```

À cet instant, le répertoire de travail correspond exactement à la révision 4 du dépôt. Néanmoins, supposons que vous modifiiez `bouton.c` et que vous propagiez cette modification. En supposant qu'aucune autre propagation n'a eu lieu, votre propagation crée la révision 5 du dépôt et votre copie de travail ressemble maintenant à ceci :

```
calc/Makefile:4
    entier.c:4
    bouton.c:5
```

Supposons maintenant qu'à ce moment précis, Sally propage une modification d'`entier.c`, créant la révision 6. Si vous utilisez **svn update** pour mettre à jour votre copie de travail, elle ressemble alors à ceci :

```
calc/Makefile:6
    entier.c:6
    bouton.c:6
```

Les modifications apportées par Sally à `entier.c` apparaissent dans votre copie de travail et vos modifications sont toujours présentes dans `bouton.c`. Dans cet exemple, le texte de `Makefile` est identique dans les révisions 4, 5 et 6 mais Subversion marque votre copie de travail de `Makefile` comme étant à la révision 6 pour indiquer qu'elle est à jour. Ainsi, quand vous effectuez une mise à jour au niveau de la racine de votre copie de travail, celle-ci correspond en général à une révision donnée du dépôt.

Les copies de travail suivent l'évolution du dépôt

Pour chaque fichier d'un répertoire de travail, Subversion enregistre deux informations essentielles dans la zone administrative `.svn/` :

- la révision sur laquelle votre fichier de travail est basé (qui est appelée la *révision de travail* du fichier) et
- la date et l'heure de la dernière mise à jour de la copie locale depuis le dépôt

À partir de ces informations, en dialoguant avec le dépôt, Subversion est capable de déterminer dans lequel des quatre états suivants se trouve un fichier de travail :

Inchangé et à jour
: Le fichier est inchangé dans le répertoire de travail et aucune modification de ce fichier n'a été propagée vers le dépôt depuis sa révision de travail. Un appel à **svn commit** sur le fichier ne fera rien, un appel à **svn update** sur le fichier ne fera rien non plus.

Modifié localement et à jour
: Le fichier a été modifié dans le répertoire de travail et aucune modification du fichier n'a été propagée dans le dépôt depuis la dernière mise à jour. Il existe des modifications locales qui n'ont pas été propagées vers le dépôt, donc un appel à **svn commit** sur le fichier permettra de publier vos modifications et un appel à **svn update** ne fera rien.

Inchangé et périmé
: Le fichier n'a pas été modifié dans le répertoire de travail mais a changé dans le dépôt. Le fichier devra être mis à jour à un moment ou à un autre, pour l'amener au niveau de la dernière révision publique. Un appel à **svn commit** sur le fichier ne fera rien et un appel à **svn update** incorporera les dernières modifications dans votre copie de travail.

Modifié localement et périmé
: Le fichier a été modifié à la fois dans le répertoire de travail et dans le dépôt. Un appel à **svn commit** sur le fichier va échouer, renvoyant comme erreur « Périmé » (« out-of-date » en anglais). Le fichier doit d'abord être mis à jour ; un appel à **svn update** va tenter de fusionner les modifications publiques avec les modifications locales. Si Subversion ne parvient pas à réaliser automatiquement cette fusion de manière crédible, il va laisser à l'utilisateur la tâche de résoudre le conflit.

Tout ceci peut sembler compliqué à gérer mais la commande **svn status** vous indique dans quel état se trouve n'importe quel élément de votre copie de travail. Pour plus d'informations sur cette commande, référez-vous à la section intitulée « Avoir une vue d'ensemble des changements effectués ».

Copies de travail mixtes, à révisions mélangées

Un principe général de Subversion est d'être aussi flexible que possible. Un type particulier de flexibilité est la capacité d'avoir une copie de travail contenant des fichiers et des répertoires avec un mélange de différents numéros de révision. Malheureusement, cette flexibilité a tendance à embrouiller un certain nombre de nouveaux utilisateurs. Si l'exemple précédent contenant des révisions mixtes vous laisse perplexe, voici une amorce d'explication à la fois sur les raisons pour lesquelles cette fonctionnalité existe et sur la façon de l'utiliser.

Mise à jour et propagation sont deux choses distinctes

Une des règles fondamentales de Subversion est que l'action de « pousser » ne déclenche pas une action de « tirer », ni l'inverse. Le simple fait que vous soyez prêt à soumettre vos nouvelles modifications au dépôt ne veut pas dire que vous êtes prêts à recevoir les modifications d'autres personnes. Et si vous avez de nouvelles modifications encore en cours, alors **svn update** fusionne élégamment les changements du dépôt avec les vôtres, plutôt que de vous forcer à les publier.

Le principal effet secondaire de cette règle est que la copie de travail a de la comptabilité supplémentaire à effectuer pour suivre les mélanges de révision et également être tolérante vis-à-vis de l'ensemble. Cela est rendu encore plus difficile par le fait que les répertoires eux-mêmes sont suivis en versions.

Par exemple, supposons que vous ayez une copie de travail qui soit intégralement à la révision 10. Vous éditez le fichier `truc.html` et réalisez ensuite un **svn commit** qui crée la révision 15 dans le dépôt. Après que la propagation ait réussi, nombreux sont ceux parmi les nouveaux utilisateurs qui s'attendraient à ce que toute la copie de travail soit à la révision 15, mais ce n'est pas le cas ! Un certain nombre de modifications ont pu avoir lieu dans le dépôt entre les révisions 10 et 15. Le client ne sait rien de ces changements qui ont été apportés au dépôt, puisque vous n'avez pas encore exécuté la commande **svn update** et la commande **svn commit** ne récupère pas les nouvelles modifications. D'un autre côté, si la commande **svn commit** téléchargeait automatiquement les modifications les plus récentes, alors il serait possible d'avoir toute la copie de travail à la révision 15 mais, dans ce cas, nous enfreindrions la règle fondamentale selon laquelle « pousser » et « tirer » doivent demeurer des actions distinctes. Ainsi, la seule chose que le client Subversion peut faire en toute sécurité est de marquer le fichier `truc.html`, et lui seulement, comme étant à la révision 15. Le reste de la copie de travail reste à la révision 10. Seule l'exécution de la commande **svn update** permet de récupérer les dernières modifications et de marquer la copie de travail comme étant à la révision 15.

Des révisions mélangées sont normales

Le fait est qu' *à chaque fois* que vous exécutez la commande **svn commit**, votre copie de travail se retrouve composée d'un mélange de révisions. Les éléments que vous venez juste de propager sont marqués comme ayant un numéro de révision plus élevé que tous les autres. Après plusieurs propagations (sans mise à jour entre-temps), votre copie de travail va contenir tout un mélange de révisions. Même si vous êtes la seule personne à utiliser le dépôt, vous constaterez quand même ce phénomène. Pour étudier votre propre mélange de révisions de travail, utilisez la commande **svn status** avec l'option `--verbose` (voir la section intitulée « Avoir une vue d'ensemble des changements effectués » pour plus d'informations).

Souvent, les nouveaux utilisateurs n'ont pas du tout conscience que leur copie de travail contient des révisions mélangées. Cela peut être déroutant car beaucoup de commandes client sont sensibles à la révision de travail de l'élément qu'elles examinent. Par exemple, la commande **svn log** est utilisée pour afficher l'historique des modifications d'un fichier ou d'un répertoire (cf. la section intitulée « Affichage de l'historique »). Lorsque l'utilisateur appelle cette commande sur un objet de la copie de travail, il s'attend à obtenir l'historique complet de celui-ci. Mais si la révision de travail de l'objet est assez ancienne (souvent parce que **svn update** n'a pas été lancé depuis un certain temps), alors c'est l'historique de l'*ancienne* version de l'objet qui est affiché.

Un mélange de révisions est utile

Si votre projet est suffisamment complexe, vous allez découvrir qu'il est parfois pratique d'effectuer un *retour en arrière* forcé (c'est-à-dire de faire une mise à jour vers une version plus ancienne que celle que vous avez déjà) sur certaines parties de votre copie de travail vers des révisions plus anciennes ; vous apprendrez comme le faire dans le Chapitre 2, *Utilisation de base*. Vous avez peut-être envie de tester une version précédente d'un sous-module contenu dans un sous-répertoire ou bien de comprendre comment un bogue est apparu pour la première fois dans un fichier donné. C'est le côté « machine à voyager dans le temps » d'un logiciel de gestion de versions, la fonctionnalité qui vous permet de déplacer n'importe quelle partie de votre

copie de travail en avant ou en arrière dans le temps.

Les mélanges de révisions ont des limites

Quelle que soit la façon dont vous utilisez les mélanges de révision dans votre copie de travail, il existe des limites à cette flexibilité.

Premièrement, vous ne pouvez pas propager la suppression d'un fichier ou d'un répertoire qui n'est pas complètement à jour. Si une version plus récente de l'élément existe dans le dépôt, votre tentative de suppression est rejetée, afin de vous empêcher de détruire accidentellement des modifications dont vous n'aviez pas encore connaissance.

Deuxièmement, vous ne pouvez propager la modification des métadonnées d'un répertoire que si celui-ci est complètement à jour. Vous apprendrez comment associer des « propriétés » à des éléments dans le Chapitre 3, *Sujets avancés*. La révision de travail d'un répertoire définit un ensemble précis d'entrées et de propriétés et propager la modification d'une propriété d'un répertoire périmé risquerait de détruire des propriétés dont vous n'aviez pas encore connaissance.

Résumé

Nous avons couvert un certain nombre de concepts fondamentaux de Subversion dans ce chapitre :

- Nous avons introduit les notions de dépôt central, de copie de travail du client et d'ensemble des révisions de l'arborescence du dépôt.
- Nous avons vu quelques exemples simples de la façon dont deux collaborateurs peuvent utiliser Subversion pour publier et recevoir des modifications en provenance l'un de l'autre, en utilisant le modèle « copier-modifier-fusionner ».
- Nous avons évoqué la façon dont Subversion suit et gère les informations dans une copie de travail.

À présent, vous avez probablement une bonne idée générale de la façon dont Subversion fonctionne. Armé de cette connaissance, vous devez désormais être prêt à passer au chapitre suivant qui traite en détail des commandes et des fonctionnalités de Subversion.

Chapitre 2. Utilisation de base

Nous allons maintenant voir plus en détail l'utilisation de Subversion. Quand vous aurez terminé ce chapitre, vous serez capable d'effectuer toutes les tâches nécessaires à une utilisation quotidienne de Subversion. Nous allons commencer par enregistrer nos fichiers dans Subversion, puis extraire notre code. Ensuite, nous expliquons comment modifier des fichiers et examiner ces changements. Nous voyons aussi comment faire pour intégrer les changements venant d'autres personnes dans notre copie de travail, les examiner et résoudre les conflits qui pourraient apparaître.

Notez que ce chapitre ne doit pas être vu comme une liste complète de toutes les commandes de Subversion, mais plutôt comme une introduction conviviale aux opérations Subversion les plus courantes que vous êtes susceptible de rencontrer. Ici, nous supposons que vous avez lu et compris le Chapitre 1, *Notions fondamentales* et que vous êtes familier avec le modèle général de Subversion. Pour une liste complète de toutes les commandes, reportez-vous au Chapitre 9, *Références complètes de Subversion*.

À l'aide !

Avant de poursuivre la lecture, voici la commande la plus importante de Subversion : **svn help**. Le client en ligne de commande de Subversion inclut sa propre documentation : à tout moment, un rapide **`svn help` *`sous-commande`*** vous renvoie la syntaxe, les options et le comportement de la sous-commande.

```
$ svn help import
import: Charge un fichier ou une arborescence non versionnée dans un dépôt.
usage : import [CHEMIN] URL

  Charge récursivement une copie de CHEMIN vers URL.
  Si CHEMIN est omis, '.' est utilisé.
  Les répertoires parents sont créés si nécessaire dans le dépôt.
  Si CHEMIN est un répertoire, son contenu est ajouté directement sous l'URL.
  Les objects non versionnables tels les périphériques ou les pipes sont
  ignorés si l'option '--force' est spécifiée.

Options valides:
  -q [--quiet]             : n'affiche rien, ou seulement des
                             informations résumées
  -N [--non-recursive]     : obsolète : essayer --depth=files ou
                             --depth=immediates
  --depth ARG              : limite l'opération à cette profondeur
                             (depth empty/files/immediates/infinity)
                             en argument
…
```

Options, sélecteurs, drapeaux… Doux Jésus !

Le client en ligne de commande de Subversion possède de nombreux modificateurs de commandes (que nous appelons options), mais il existe deux types d'options distincts : les options courtes sont constituées d'un unique tiret suivi d'une unique lettre, tandis que les options longues sont formées de deux tirets suivis d'un certain nombre de lettres (c'est-à-dire respectivement `-s` et `--ceci-est-une-option-longue`). Chaque option possède une option longue, mais seules certaines options ont aussi un format court (ce sont généralement des options qui sont utilisées très souvent). Par souci de clarté, nous utilisons *généralement* la forme longue dans les exemples de code, mais lorsque nous décrivons les options, s'il existe une forme courte, nous donnons à la fois la forme longue (pour plus de clarté) et la forme courte (car plus facile à retenir). Utilisez la forme qui vous convient le mieux, mais n'essayez pas d'utiliser les deux.

Enregistrement de données dans votre dépôt

Deux moyens sont à votre disposition pour enregistrer de nouveaux fichiers dans votre dépôt Subversion : **svn import** et **svn add**. Nous abordons ici la commande **svn import** et, plus loin dans le chapitre, la commande **svn add**, lorsque nous passerons en revue une journée typique avec Subversion.

svn import

La commande **svn import** est un moyen rapide de copier une arborescence non-suivie en versions dans le dépôt, créant des dossiers intermédiaires si nécessaire. **svn import** ne nécessite pas de copie de travail et vos fichiers sont immédiatement propagés dans le dépôt. Ce moyen est utilisé essentiellement quand vous avez une arborescence dont vous voulez suivre les changements dans votre dépôt Subversion. Par exemple :

```
$ svnadmin create /var/svn/nouveau-depot
$ svn import mon-arborescence file:///var/svn/nouveau-depot/un/projet \
             -m "Import initial"
Ajout         mon-arborescence/truc.c
Ajout         mon-arborescence/machin.c
Ajout         mon-arborescence/sous-repertoire
Ajout         mon-arborescence/sous-repertoire/bidule.h
```

L'exemple précédent copie le contenu du dossier `mon-arborescence` dans le dossier `un/projet` dans le dépôt :

```
$svn list file:///var/svn/nouveau-depot/un/projet
truc.c
machin.c
sous-repertoire/
```

Notez qu'après la fin de l'import, l'arborescence d'origine n'est *pas* transformée en copie de travail. Pour commencer à travailler, vous devez extraire grâce à **svn checkout** une copie de travail toute fraîche de l'arborescence.

Organisation conseillée de votre dépôt

Bien que Subversion vous permette d'organiser votre dépôt de la manière dont vous le voulez, nous vous recommandons de créer un dossier appelé `trunk` pour stocker la « ligne principale » du développement, un autre dossier `branches` qui contiendra les copies alternatives (ou branches) et un dossier `tags` pour les versions étiquetées. Par exemple :

```
$ svn list file:///var/svn/depot
/trunk
/branches
/tags
```

Vous en apprendrez plus sur les étiquettes et les branches dans le Chapitre 4, *Gestion des branches*. Pour plus de détails et pour voir comment gérer plusieurs projets, reportez-vous à la section intitulée « Agencement du dépôt », et à la section intitulée « Stratégies d'organisation d'un dépôt » pour en savoir plus sur les répertoires racines d'un projet.

Extraction initiale

En général, vous commencerez à utiliser un dépôt Subversion en faisant une *extraction* (« checkout » en anglais) de votre projet. Extraire un projet d'un dépôt crée sur votre ordinateur une « copie de travail » de ce projet. Cette copie contient la version `HEAD` (la dernière révision) du dépôt que vous indiquez dans la ligne de commande :

```
$ svn checkout http://svn.apache.org/repos/asf/subversion/
A    subversion/trunk
A    subversion/trunk/NOTICE
A    subversion/trunk/LICENSE
A    subversion/trunk/Makefile.in
A    subversion/trunk/build.conf
…
Révision 8810 extraite.
```

Qu'y a-t-il dans un *nom* ?

Subversion essaie au maximum de ne pas limiter le type de données que vous placez en suivi de versions. Le contenu des fichiers et les valeurs des propriétés sont stockés et transmis en tant que données binaires, la section intitulée « Type de contenu des fichiers » vous explique comment indiquer à Subversion que des opérations « textuelles » n'ont pas de sens pour un fichier particulier. Il existe toutefois quelques cas pour lesquels Subversion définit des limitations sur les informations qu'il stocke.

Subversion gère en interne certaines parties des données au format Unicode UTF-8, par exemple les noms de propriétés, les noms de chemins et les messages de trace. Cela ne veut toutefois pas dire que toutes vos interactions avec Subversion doivent faire intervenir l'UTF-8. En règle générale, les clients Subversion vont gérer de façon transparente les conversions entre l'UTF-8 et le système de codage utilisé par votre ordinateur, si cela a un sens de faire une telle conversion (ce qui est le cas pour les codages les plus courants utilisés de nos jours).

Dans les échanges WebDAV ainsi que dans des anciennes versions de certains fichiers de gestion interne de Subversion, les chemins sont utilisés en tant que valeurs d'attribut XML et les noms de propriétés en tant que noms de tags XML. Cela veut dire que les chemins ne peuvent contenir que des caractères XML (1.0) valides et que les noms de propriétés sont encore plus limités, ne pouvant contenir que des caractères ASCII. Subversion interdit également les caractères `TAB` (tabulation), `CR` et `LF` (caractères de fin de ligne) dans les noms de chemins pour empêcher les chemins d'être coupés en deux lors des comparaisons ou dans les sorties de commandes comme **svn log** ou **svn status**.

Bien que cela fasse beaucoup de choses à se rappeler, ces limitations sont rarement un problème en pratique. Tant que vos paramètres locaux sont compatibles avec UTF-8 et que vous n'utilisez pas de caractères de contrôle dans les chemins, vous ne devriez pas avoir de problème pour communiquer avec Subversion. Le client en ligne de commande apporte un peu d'aide supplémentaire : afin de créer des versions « valides » pour un usage interne, il banalise automatiquement, si nécessaire, les caractères illégaux contenus dans les chemins d'URL que vous tapez.

Alors que l'exemple précédent extraie le répertoire de base `trunk`, vous pouvez tout aussi facilement extraire un sous-répertoire situé à n'importe quelle profondeur dans le dépôt en spécifiant le sous-répertoire dans l'URL d'extraction :

```
$ svn checkout \
      http://svn.apache.org/repos/asf/subversion/trunk/subversion/tests/cmdline/
A    cmdline/cat_tests.py
A    cmdline/revert_tests.py
A    cmdline/entries_tests.py
A    cmdline/svneditor.bat
A    cmdline/import_tests.py
…
Révision 8810 extraite.
```

Comme Subversion utilise le modèle copier-modifier-fusionner à la place du modèle verrouiller-modifier-libérer (voir la section intitulée « Modèles de gestion de versions »), vous pouvez commencer immédiatement à modifier les fichiers et les répertoires de votre copie de travail. Votre copie de travail n'est qu'un ensemble de fichiers et de répertoires comme les autres dans votre système. Vous pouvez y éditer des fichiers, la modifier, la déplacer, vous pouvez même supprimer toute votre copie de travail et l'oublier définitivement.

Bien que votre copie de travail « n'est qu'un ensemble de fichiers et de répertoires comme les autres dans votre système », vous pouvez éditer vos fichiers comme vous le voulez, mais vous devez signaler à Subversion *toutes vos autres opérations*. Par exemple, si vous voulez copier ou déplacer un élément dans votre copie de travail, vous devez utiliser **svn copy** ou **svn move** à la place des commandes de copie ou de déplacement fournies par votre système d'exploitation. Nous aborderons plus en détail ces commandes plus loin dans ce chapitre.

À moins que vous ne soyez prêt à propager l'ajout d'un nouveau fichier ou d'un nouveau répertoire ou la modification d'un fichier ou répertoire existant, il n'est pas nécessaire d'informer davantage le serveur Subversion que vous avez fait quelque chose.

À propos du répertoire .svn ?

Chaque répertoire dans une copie de travail contient une zone administrative, un sous-répertoire nommé `.svn`. Habituellement, les commandes d'affichage du contenu des répertoires ne font pas apparaître ce sous-répertoire, mais il s'agit tout de même d'un répertoire important. Quoique vous fassiez, ne supprimez ni ne changez rien dans la zone administrative ! Subversion dépend d'elle pour gérer votre copie de travail.

Si vous supprimez accidentellement le sous-répertoire `.svn`, la façon la plus facile de régler le problème est de supprimer entièrement le répertoire qui le contenait (via une suppression classique du système, pas un appel à **svn delete**), puis de lancer **`svn update`** depuis un répertoire parent. Le client Subversion téléchargera alors le répertoire que vous avez supprimé, en incluant également une nouvelle zone `.svn`.

Certes, vous pouvez extraire une copie de travail avec l'URL du dépôt comme seul argument, mais vous pouvez également spécifier un répertoire après l'URL du dépôt. Cela place votre copie de travail dans le nouveau répertoire indiqué. Par exemple :

```
$  svn checkout http://svn.apache.org/repos/asf/subversion/trunk subv
A    subv/NOTICE
A    subv/LICENSE
A    subv/Makefile.in
A    subv/build.conf
…
Révision 8810 extraite.
```

Ceci place votre copie de travail dans un répertoire nommé `subv` au lieu d'un répertoire nommé `trunk` comme nous l'avions fait précédemment. Le répertoire `subv` est créé s'il n'existait pas auparavant.

Interdiction de la mise en cache du mot de passe

Lorsque vous réalisez une opération Subversion qui nécessite une authentification, Subversion met par défaut en cache sur le disque vos éléments d'authentification. Il fait cela par commodité, pour que vous ne soyez pas obligé de ré-entrer constamment votre mot de passe pour les opérations suivantes. Si la mise en cache des mots de passe Subversion vous dérange[1], vous pouvez désactiver la mise en cache de façon permanente ou au cas par cas.

Pour désactiver la mise en cache du mot de passe pour un appel donné d'une commande particulière, passez l'option `--no-auth-cache` sur la ligne de commande. Pour désactiver de façon permanente la mise en cache, vous pouvez ajouter la ligne `store-passwords = no` à votre fichier de configuration Subversion sur votre machine locale. Voir la section intitulée « Mise en cache des éléments d'authentification du client » pour plus de détails.

Authentification sous un autre nom

Puisque Subversion met en cache par défaut les éléments d'authentification (à la fois le nom d'utilisateur et le mot de passe), il se souvient commodément de qui vous étiez la dernière fois que vous avez modifié votre copie de travail. Mais parfois, cela n'aide pas, en particulier si vous travaillez sur une copie de travail partagée, comme par exemple un répertoire de configuration système, ou le répertoire racine des documents d'un serveur Web. Dans ce cas, il suffit de passer l'option `--username` à la ligne de commande et Subversion essaie alors de vous authentifier en tant que cet utilisateur, vous demandant votre mot de passe si nécessaire.

Cycle de travail de base

Subversion dispose de nombreuses fonctionnalités, d'options, d'avertissements et de garde-fous. Mais dans une utilisation au jour le jour, vous n'utilisez qu'un petit nombre d'entre eux. Dans cette section, nous passons en revue l'utilisation quotidienne de Subversion.

[1] Bien sûr, cela ne vous inquiète pas outre mesure : d'abord parce que vous savez qu'on ne peut jamais rien effacer *réellement* avec Subversion, et ensuite parce que votre mot de passe Subversion n'est pas le même que les trois millions d'autres mots de passe que vous avez déjà, pas vrai ? Pas vrai ?

Le cycle de travail typique ressemble à ceci :

1. Mettre à jour votre copie de travail.
   - **svn update**
2. Faire des changements.
   - **svn add**
   - **svn delete**
   - **svn copy**
   - **svn move**
3. Examiner les changements effectués.
   - **svn status**
   - **svn diff**
4. Éventuellement annuler des changements.
   - **svn revert**
5. Résoudre les conflits (fusionner les modifications).
   - **svn update**
   - **svn resolve**
6. Propager les changements.
   - **svn commit**

Mettre à jour votre copie de travail

Si vous travaillez en équipe sur un projet donné, vous voulez mettre à jour votre copie locale pour recevoir toutes les modifications qui ont pu être faites par les autres développeurs du projet depuis votre dernière mise à jour. Utilisez **svn update** pour synchroniser votre copie de travail avec la dernière version présente dans le dépôt :

```
$ svn update
U  truc.c
U  machin.c
Actualisé à la révision 2.
```

Dans cet exemple, il se trouve que quelqu'un a apporté des modifications à `truc.c` ainsi qu'à `machin.c` depuis votre dernière mise à jour et Subversion vient de répercuter ces modifications dans votre copie de travail.

Lorsque le serveur envoie des modifications vers votre copie de travail via **svn update**, un code, sous forme de lettre, est affiché à coté de chaque élément pour vous permettre de savoir quelles actions Subversion a effectuées pour mettre votre copie de travail à jour. Pour en savoir plus sur le sens de ces lettres, exécutez **`svn help update`**.

Apporter des modifications à votre copie de travail

Vous pouvez à présent vous mettre au travail et apporter des modifications à votre copie de travail. Il est plus commode, habituellement, de démarrer par une modification (ou un ensemble de modifications) précise, comme par exemple écrire une nouvelle fonctionnalité, corriger un bogue, etc. Les commandes Subversion dont vous vous servez pour cela sont **svn add**, **svn**

delete, **svn copy** et **svn mkdir**. Cependant, si vous ne faites qu'éditer des fichiers qui sont déjà dans Subversion, vous n'avez besoin d'aucune de ces commandes jusqu'à ce que vous propagiez vos modifications.

Vous pouvez apporter deux types de modifications à votre copie de travail : des *modifications de fichier* et des *modifications d'arborescence*. Vous n'avez pas besoin de signaler à Subversion que vous allez modifier un fichier ; faites vos changements en utilisant votre éditeur de texte, un logiciel de traitement de texte, un logiciel de graphisme, ou n'importe quel autre outil que vous utilisez d'habitude. Subversion détecte automatiquement quels fichiers ont été modifiés et, en plus, il traite les fichiers binaires tout aussi facilement et tout aussi efficacement que les fichiers textes. Pour les modifications d'arborescence, vous pouvez demander à Subversion de « marquer » les fichiers et répertoires afin de programmer leur suppression, ajout, copie ou renommage. Bien que ces modifications soient immédiatement répercutées sur votre copie de travail, aucun ajout et aucune suppression ne sera répercuté sur le dépôt avant que vous ne les propagiez.

Gérer des liens symboliques

Dans des environnements non-Windows, Subversion est capable de suivre en versions les fichiers de type *lien symbolique* (ou « symlink »). Un symlink est un fichier qui agit comme une sorte de référence transparente vers un autre objet du système de fichiers, permettant à des programmes de consulter et de modifier ces objets de façon indirecte en effectuant les opérations sur le symlink lui-même.

Quand un symlink est propagé vers un dépôt Subversion, Subversion enregistre que ce fichier était en fait un symlink et note l'objet vers lequel il « pointait ». Quand ce symlink est extrait vers une autre copie de travail sur un système non-Windows, Subversion reconstruit un véritable lien symbolique, au niveau du système de fichiers, à partir du symlink enregistré. Mais ceci ne limite en aucune façon l'usage des copies de travail sur des systèmes tels que Windows, qui ne possèdent pas d'implémentation des symlinks. Sur de tels systèmes, Subversion se contente de créer un fichier texte ordinaire, qui contient le chemin vers lequel le symlink pointait à l'origine. Bien que ce fichier ne puisse être utilisé comme un symlink sur un système Windows, il n'empêche pas les utilisateurs de Windows d'effectuer leurs autres opérations Subversion.

Voici un aperçu des cinq sous-commandes Subversion les plus utilisées pour faire des modifications sur l'arborescence :

`svn add truc`
: Marque le fichier, le répertoire ou le lien symbolique `truc` pour ajout. Lors de la prochaine propagation, `truc` devient un fils de son répertoire parent. Notez que si `truc` est un répertoire, tout ce qui se trouve à l'intérieur de `truc` est marqué pour ajout. Si vous ne désirez ajouter que `truc` lui-même, passez l'option `--depth empty`.

`svn delete truc`
: Marque le fichier, le répertoire ou le lien symbolique `truc` pour suppression. Si `truc` est un fichier ou un lien, il est immédiatement supprimé de votre copie de travail. Si `truc` est un répertoire, il n'est pas supprimé, mais Subversion le marque pour suppression. Quand vous propagez vos modifications, `truc` est complètement supprimé de votre copie de travail et du dépôt. [2]

`svn copy truc bidule`
: Crée un nouvel élément `bidule` par duplication de `truc` et marque automatiquement `bidule` pour ajout. Lorsque `bidule` est ajouté au dépôt, lors de la prochaine propagation, son historique est enregistré (comme ayant été créé à partir de `truc`). **svn copy** ne crée pas de répertoires intermédiaires, à moins que vous ne lui passiez l'option `--parents`.

`svn move truc bidule`
: Cette commande équivaut exactement à **`svn copy truc bidule; svn delete truc`**. C'est-à-dire que `bidule` est marqué pour ajout en tant que copie de `truc` et que `truc` est marqué pour suppression. **svn move** ne crée pas de répertoires intermédiaires, à moins que vous ne lui passiez l'option `--parents`.

`svn mkdir blort`
: Cette commande équivaut exactement à **`mkdir blort; svn add blort`**. C'est-à-dire qu'un nouveau répertoire nommé `blort` est créé et marqué pour ajout.

[2] Bien sûr, rien n'est jamais totalement supprimé du dépôt — seulement de la révision HEAD du dépôt. Vous pouvez récupérer tout ce que vous avez supprimé en extrayant (ou en mettant à jour votre copie de travail à) une révision précédente de celle dans laquelle vous avez fait la suppression. Lisez également la section intitulée « Résurrection des éléments effacés ».

Modifier le dépôt sans copie de travail

Dans *certains* cas, les modifications d'arborescence sont propagées immédiatement vers le dépôt. Cela n'arrive que quand une sous-commande opère directement sur une URL et non sur le chemin d'une copie de travail. En particulier, certains usages de **svn mkdir**, **svn copy**, **svn move** et **svn delete** peuvent fonctionner avec des URL (et n'oubliez pas que **svn import** agit toujours sur une URL).

Les opérations sur les URL se comportent de cette manière parce que les commandes qui opèrent sur une copie de travail peuvent utiliser la copie de travail comme une sorte de « zone de transit » pour y préparer vos modifications avant de les propager vers le dépôt. Les commandes qui opèrent sur des URL ne disposent pas d'un tel luxe, donc quand vous opérez directement sur une URL, toute action mentionnée ci-dessus est propagée immédiatement.

Examiner les changements apportés

Une fois vos modifications apportées, vous devez les intégrer au dépôt. Avant de le faire, il est souvent utile de jeter un coup d'œil sur ces modifications pour savoir exactement ce que vous avez changé. En examinant les modifications avant de les intégrer au dépôt, le commentaire associé à la propagation sera souvent plus pertinent. Éventuellement, vous verrez que vous avez modifié un fichier par inadvertance et cela vous donne une chance de revenir sur ces modifications avant de les propager au dépôt. En outre, c'est une bonne occasion de passer en revue et d'examiner les modifications avant de les publier. Vous pouvez obtenir une vue d'ensemble des modifications que vous avez faites en utilisant **svn status** et voir le détail de ces changements en utilisant **svn diff**.

Regardez ça : pas besoin de réseau !

Vous pouvez utiliser les commandes **svn status**, **svn diff**, et **svn revert** sans aucun accès réseau même si votre dépôt *est* distant. C'est ainsi plus facile de gérer vos changements lorsque vous êtes isolé, dans un avion, dans un train de banlieue ou même sur la plage[3].

En effet, Subversion garde en cache des copies privées des versions originales de chaque fichier suivi en version dans les zones administratives `.svn`. Cela lui permet d'afficher (et éventuellement d'annuler) les modifications faites localement à ces fichiers, *sans le moindre accès réseau*. Ce cache (appelé la « base texte ») permet également à Subversion, lors d'une propagation, de n'envoyer au dépôt, sous forme de *delta*, que les modifications (compressées) faites par l'utilisateur. Disposer d'un tel cache est un énorme avantage, même dans le cas d'une connexion Internet haut débit, car il est beaucoup plus rapide d'envoyer des différences sur un fichier plutôt que l'ensemble du fichier au serveur.

Subversion a été optimisé pour vous aider dans cette tâche et il est capable de faire beaucoup de choses sans communiquer avec le dépôt. En particulier, votre copie de travail conserve, dans le répertoire `.svn`, la « copie » de l'original de chaque fichier suivi en versions. C'est pour ça que Subversion peut vous indiquer rapidement en quoi vos fichiers ont changé, ou même vous permettre d'annuler vos changements sans contacter le dépôt.

Avoir une vue d'ensemble des changements effectués

Pour avoir une vue d'ensemble des changements que vous avez effectués, utilisez la commande **svn status**. C'est certainement la commande que vous utiliserez le plus.

Utilisateurs de CVS : attention à la commande update !

Vous avez certainement l'habitude d'utiliser **cvs update** pour visualiser les changements que vous avez effectués sur votre copie de travail. **svn status** vous donne toutes les informations utiles à ce propos, sans accès au dépôt et sans incorporer les changements effectués par d'autres utilisateurs.

Dans Subversion, **svn update** ne s'occupe que de la mise à jour : il met à jour votre copie de travail avec tous les changements propagés dans le dépôt depuis votre dernière mise à jour. Il faut vous débarrasser de l'habitude d'utiliser la commande **update** pour visualiser les modifications que vous avez effectuées sur votre copie de travail locale.

[3] Et que vous n'avez pas de carte Wifi. Vous ne croyiez tout de même pas nous avoir aussi facilement ?

Si vous lancez **svn status** sans argument à la racine de votre copie de travail, Subversion détecte toutes les modifications effectuées sur les fichiers et sur l'arborescence. Voici quelques exemples de codes que la commande **svn status** affiche (notez que le texte après # n'est pas affiché par **svn status**).

```
?       gribouillage.c          # le fichier n'est pas suivi en versions
A       bazar/pognon/machin.h   # le fichier sera Ajouté
C       bazar/pognon/tas.c      # le fichier entre en Conflit
                                  avec une mise à jour
D       bazar/poisson.c         # le fichier sera supprimé
                                  (Deletion en anglais)
M       truc.c                  # le contenu de truc.c a subi
                                  des Modifications
```

Dans ce format d'affichage, **svn status** affiche six colonnes de caractères, suivis par plusieurs espaces, suivis par un nom de fichier ou de répertoire. La première colonne indique le statut du fichier ou du répertoire et/ou son contenu. Les codes sont :

`A élément`
: Le fichier, répertoire ou lien symbolique `élément` est marqué pour ajout au dépôt.

`C élément`
: Le fichier `élément` est dans un état de conflit. C'est-à-dire que des modifications ont eu lieu dans le dépôt depuis votre dernière mise à jour et ces modifications interfèrent avec les modifications que vous avez effectuées sur votre copie de travail (et la mise à jour n'a pas résolu ce conflit). Vous devez résoudre ce conflit avant de propager vos changements vers le dépôt.

`D élément`
: Le fichier, répertoire ou lien symbolique `élément` est marqué pour suppression (« Deletion » en anglais).

`M élément`
: Le contenu du fichier `élément` a été modifié.

Si vous spécifiez un chemin à **svn status**, vous obtenez uniquement les informations relatives à ce chemin :

```
$ svn status bazar/poisson.c
D      bazar/poisson.c
```

svn status possède aussi une option `--verbose` (`-v`) pour le rendre plus verbeux : il affiche alors le statut de *tous* les éléments de votre copie de travail, même ceux qui n'ont pas subi de modification :

```
$ svn status -v
M               44        23    sally     LISEZMOI
                44        30    sally     INSTALL
M               44        20    harry     truc.c
                44        18    ira       bazar
                44        35    harry     bazar/truite.c
D               44        19    ira       bazar/poisson.c
                44        21    sally     bazar/divers
A                0         ?     ?        bazar/divers/machin.h
                44        36    harry     bazar/divers/bidule.c
```

C'est la « version longue » de l'affichage de **svn status**. Les lettres de la première colonne ont la même signification que précédemment, mais la deuxième colonne indique le numéro de révision de travail de l'élément. Les troisième et quatrième colonnes indiquent le numéro de la révision dans laquelle a eu lieu le changement le plus récent et qui l'a effectué.

Aucune des commandes citées ci-dessus n'induit de connexion vers le dépôt : elles comparent les métadonnées du répertoire `.svn` avec la copie de travail. Enfin, il y a l'option `--show-updates` (`-u`) qui effectue une connexion au dépôt et ajoute les informations sur les éléments périmés :

```
$ svn status -u -v
M      *        44        23    sally     LISEZMOI
M               44        20    harry     truc.c
       *        44        35    harry     bazar/truite.c
D               44        19    ira       bazar/poisson.c
A                0         ?     ?        bazar/divers/machin.h
État par rapport à la révision   46
```

Notez les deux astérisques : si vous lanciez la commande **svn update**, vous recevriez les changements relatifs à `LISEZMOI` et `truite.c`. Cela vous procure des informations particulièrement intéressantes : vous devez faire une mise à jour et récupérer les changements effectués sur `LISEZMOI` avant de propager les vôtres, sinon le dépôt rejettera votre propagation en la considérant comme périmée (le sujet est approfondi plus tard).

svn status peut afficher beaucoup plus d'informations sur les fichiers et répertoires de votre copie de travail que ce que nous venons de voir ici. Pour obtenir une description exhaustive de **svn status** et de ses modes d'affichage, reportez-vous à svn status.

Voir en détail les modifications que vous avez effectuées

La commande **svn diff** offre une autre façon d'examiner vos changements. Vous pouvez retrouver *exactement* ce que vous avez modifié en lançant la commande **`svn diff`** sans argument : elle affiche les changements au format *diff unifié*.

```
$ svn diff
Index: truc.c
===================================================================
--- truc.c (révision 3)
+++ truc.c (copie de travail)
@@ -1,7 +1,12 @@
+#include <sys/types.h>
+#include <sys/stat.h>
+#include <unistd.h>
+
+#include <stdio.h>
 int main(void) {
-  printf("Soixante-quatre tranches de fromage...\n");
+  printf("Soixante-cinq tranches de fromage...\n");
 return 0;
 }
Index: LISEZMOI
===================================================================
--- LISEZMOI (révision 3)
+++ LISEZMOI (copie de travail)
@@ -193,3 +193,4 @@
+Pense-bête : passer au pressing.
Index: bazar/poisson.c
===================================================================
--- bazar/poisson.c (révision 1)
+++ bazar/poisson.c (copie de travail)
-Bienvenue dans le fichier 'poisson'.
-Plus d'informations seront disponibles prochainement.
Index: bazar/divers/machin.h
===================================================================
--- bazar/divers/machin.h (révision 8)
+++ bazar/divers/machin.h (copie de travail)
+Voici un nouveau fichier pour
+écrire sur les machins.
```

La commande **svn diff** produit ces lignes en comparant vos fichiers de travail aux copies « originales » en cache dans la zone `.svn`. Les fichiers marqués pour ajout sont affichés comme toute section de texte ajoutée, et les fichiers marqués pour suppression sont affichés comme toute section de texte supprimée.

L'affichage est conforme au format diff unifié. C'est-à-dire que les lignes supprimées commencent par le signe `-` et que les lignes ajoutées commencent par le signe `+`. **svn diff** affiche également le nom du fichier et la localisation dans le fichier, à

l'intention du programme **patch**. Vous pouvez ainsi créer des « correctifs » en redirigeant la sortie de **svn diff** vers un fichier :

```
$ svn diff > fichier-correctif
```

Vous pouvez, par exemple, envoyer par mail le fichier correctif à un autre développeur pour relecture ou test avant de le propager vers le dépôt.

Subversion utilise son propre moteur de calcul de différences, qui produit par défaut des résultats au format diff unifié. Si vous désirez obtenir les différences dans un autre format, spécifiez un programme de comparaison externe en utilisant l'option `--diff-cmd` et en fournissant les paramètres que vous voulez à l'aide de l'option `--extensions` (`-x`). Par exemple, pour obtenir les différences entre votre version locale du fichier et l'original de `truc.c` au format « contexte » et en ignorant la casse des caractères, vous pouvez lancer la commande **`svn diff --diff-cmd /usr/bin/diff --extensions '-i' truc.c`**.

Annuler des changements sur la copie de travail

Supposons qu'en examinant la sortie de **svn diff**, vous vous rendiez compte que tous les changements effectués sur un fichier donné sont erronés. Peut-être auriez-vous dû laisser le fichier tel quel, ou bien peut-être qu'il serait plus facile de reprendre les changements depuis le début.

C'est l'occasion idéale pour utiliser **svn revert** :

```
$ svn revert LISEZMOI
'LISEZMOI' réinitialisé
```

Subversion ramène le fichier dans son état d'avant les modifications en le remplaçant par la copie de l'original stockée dans la zone `.svn`. Mais notez aussi que **svn revert** peut annuler *n'importe quelle* opération. Par exemple, vous pouvez décider que, après tout, vous ne voulez pas ajouter tel fichier :

```
$ svn status truc
?      truc

$ svn add truc
A         truc

$ svn revert truc
'truc' réinitialisé

$ svn status truc
?      truc
```

`svn revert` *`element`* produit exactement le même effet qu'effacer *`element`* de votre copie de travail puis de lancer la commande **`svn update -r BASE` *`element`***. Toutefois, si vous voulez revenir à une version antérieure d'un fichier, **svn revert** a un comportement notablement différent : il n'a pas besoin de contacter le dépôt pour restaurer le fichier.

Ou que vous avez peut-être effacé un fichier par mégarde :

```
$ svn status LISEZMOI

$ svn delete LISEZMOI
D         LISEZMOI

$ svn revert LISEZMOI
'LISEZMOI' réinitialisé

$ svn status LISEZMOI
```

Résoudre les conflits (fusionner des modifications)

Nous avons déjà vu que **`svn status -u`** est capable de prévoir les conflits. Supposons que vous lanciez **`svn update`** et que le résultat suivant apparaisse :

```
$ svn update
U  INSTALL
G  LISEZMOI
Conflit découvert dans 'machin.c'.
Sélectionner : (p) report, (df) diff complet, (e) édite,
        (h) aide pour plus d'options :
```

Les codes `U` et `G` ne doivent pas vous inquiéter, les fichiers correspondants ayant absorbé sans problème les modifications venant du dépôt. Les fichiers notés `U` (pour « Updated ») ne contenaient aucun changement local mais ont été mis à jour à partir de changements présents dans le dépôt. Le `G` (pour « merGed ») signifie *fusionné*, ce qui veut dire que le fichier avait subi des changements localement et que les changements en provenance du dépôt ont pu être appliqués sans affecter les changements locaux.

Mais les deux lignes suivantes font partie d'une fonctionnalité (apparue dans Subversion 1.5) appelée *résolution interactive des conflits*. Cela signifie que les changements du dépôt interfèrent avec les vôtres et que vous avez la possibilité de résoudre ce conflit. Les options les plus utilisées sont affichées, mais vous pouvez voir toutes les options possibles en tapant **`h`** :

```
…
(p)  report        - marque ce conflit pour résolution ultérieure
(df) diff-complet  - montre toutes les différences du fichier fusionné
(e)  édite         - résout manuellement le conflit avec un éditeur
(r)  résolu        - utilise la version fusionnée
(mf) mien complet  - utilise ma version (ignore les autres éditions)
(tf) autre complet - prends la version du dépôt (perds mes éditions)
(l)  lance         - utilise un outil externe pour résoudre le conflit
(h)  aide          - affiche cette liste
```

Regardons brièvement ce que recèle chaque option avant de les détailler :

`(p) report`
: laisser le fichier en état de conflit, conflit que vous devrez résoudre après la fin de la mise à jour.

`(df) diff-complet`
: afficher les différences entre la révision de base et le fichier en conflit au format diff unifié.

`(e) édite`
: ouvrir le fichier en conflit avec votre éditeur de texte favori, qui est spécifié dans la variable d'environnement `EDITOR`.

`(r) résolu`
: après édition du fichier, indiquer à Subversion que vous avez résolu les conflits à l'intérieur du fichier et qu'il doit accepter son contenu actuel ; en bref, vous avez « résolu » le conflit.

`(mf) mien complet`
: ignorer les changements envoyés par le serveur et utiliser uniquement votre version locale pour le fichier concerné.

`(tf) autre complet`
: ignorer vos changements sur le fichier concerné et utiliser la version envoyée par le serveur.

`(l) lance`
: lancer un programme externe pour résoudre le conflit. Ceci nécessite un peu de préparation en amont.

`(h) aide`

afficher la liste de toutes les commandes que vous pouvez utiliser dans la résolution interactive des conflits.

Nous allons maintenant passer en revue chaque commande, en les classant par fonctionnalité.

Voir les lignes en conflit de façon interactive

Avant de décider comment résoudre un conflit de manière interactive, il est probable que vous vouliez examiner le détail des lignes en conflit. La commande « diff » (**d**) est faite pour ça :

```
…
Sélectionner : (p) report, (df) diff-complet, (e) édite,
        (h) aide pour plus d'options : d
--- .svn/text-base/sandwich.txt.svn-base      mar. 11 déc. 2007, 21:33:57
+++ .svn/tmp/tempfile.32.tmp     mar. 11 déc. 2007, 21:34:33
@@ -1 +1,5 @@
-Achète-moi un sandwich.
+<<<<<<< .mien
+Va chercher un hamburger.
+=======
+Apporte moi un taco !
+>>>>>>> .r32
…
```

La première ligne du diff correspond à ce que contenait la copie de travail dans l'ancienne version (la révision `BASE`), la ligne suivante correspond à vos modifications et la dernière ligne contient les modifications reçues du serveur (la révision `HEAD` la plupart du temps). Une fois en possession de ces informations, vous êtes prêts pour la suite.

Résoudre les conflits en mode interactif

Il y a quatre façons de résoudre un conflit avec l'interface interactive : deux d'entre elles vous permettent de fusionner et d'adapter les modifications de manière interactive, alors que les deux autres vous permettent simplement de choisir une version du fichier parmi celles proposées et de passer à la suite.

Si vous désirez choisir une combinaison de vos modifications locales, vous pouvez utiliser la commande « édite » (**e**) pour modifier manuellement le fichier avec des marqueurs indiquant les conflits dans un éditeur de texte (déterminé par la valeur de la variable d'environnement `EDITOR`). L'édition manuelle de ce fichier avec votre éditeur préféré peut sembler quelque peu « bas de gamme » (voir la section intitulée « Résoudre les conflits à la main » pour une description détaillée), c'est pourquoi certains préfèrent utiliser des outils graphiques plus évolués et spécialisés dans la fusion de documents.

Pour utiliser un outil de fusion, vous devez soit configurer la variable d'environnement `SVN_MERGE`, soit définir l'option `merge-tool-cmd` dans votre fichier de configuration Subversion (voir la section intitulée « Options de configuration » pour plus de détails). Subversion passera quatre arguments à l'outil de fusion : le fichier dans sa révision `BASE`, la version du fichier envoyée par le serveur lors de la mise à jour, une copie du fichier contenant vos propres modifications et une copie fusionnée du fichier (contenant des marqueurs de conflits). Si votre outil attend les arguments dans un ordre ou un format différents, vous devrez écrire un script de transformation que Subversion appellera. Après avoir édité le fichier, si vous êtes satisfait de vos changements, vous pouvez indiquer à Subversion que le fichier n'est plus en conflit en utilisant la commande « résolu » (**r**).

Si vous décidez qu'il n'y a pas lieu d'effectuer de fusion et si choisir l'une ou l'autre des versions proposées du fichier vous convient, vous pouvez soit opter pour vos changements (c'est-à-dire « mien ») en utilisant la commande « mien complet » (**mf**) ou opter pour la version des autres collaborateurs en utilisant la commande « autre complet » (**tf**).

Remettre à plus tard la résolution d'un conflit

Le titre peut laisser penser à un paragraphe sur l'amélioration des relations conjugales, mais il s'agit bien toujours de Subversion, voyez plutôt. Si, lorsque vous effectuez une mise à jour, Subversion soulève un conflit que vous n'êtes pas prêt à résoudre, vous pouvez, fichier par fichier, taper **p**, pour remettre à plus tard la résolution du conflit. Si, lors de votre mise à jour, vous ne voulez résoudre aucun conflit, vous pouvez passer l'option `--non-interactive` à **svn update** et les fichiers en conflit seront automatiquement marqués `C`.

Le `C` indique un conflit, c'est-à-dire que les changements sur le serveur interfèrent avec les vôtres et vous devez donc choisir manuellement entre les différentes modifications après la fin de la procédure de mise à jour. Quand vous repoussez à plus tard

la résolution d'un conflit, Subversion va accomplir trois actions qui vous aideront à repérer et à résoudre ce conflit :

- Subversion affiche un `C` pendant la mise à jour et enregistre que le fichier est dans un état de conflit.
- Si Subversion considère que le fichier peut être fusionné, il place dans le fichier des *marqueurs de conflit* (des chaînes de caractères spéciales qui dénotent les « contours » des conflits) pour mettre en exergue les zones de conflit (Subversion utilise la propriété `svn:mime-type` pour déterminer si un fichier peut subir une fusion contextuelle ligne par ligne ; voir la section intitulée « Type de contenu des fichiers » pour en apprendre davantage).
- Pour chaque fichier en conflit, Subversion place trois fichiers supplémentaires non-suivis en versions dans votre copie de travail :

  `nom_du_fichier.mine`
  C'est votre fichier tel qu'il était dans votre copie de travail avant la mise à jour, c'est-à-dire sans les marqueurs de conflits. Ce fichier ne comporte que vos derniers changements (si Subversion considère que le fichier ne peut pas être fusionné, le fichier `.mine` n'est pas créé, car il serait identique à la version de travail).

  `nom_du_fichier.r`*`ANCIENNE_REVISION`*
  C'est le fichier tel qu'il était à la révision `BASE`, avant la mise à jour de votre copie de travail. C'est donc le fichier que vous avez extrait avant de faire vos dernières modifications.

  `nom_du_fichier.r`*`NOUVELLE_REVISION`*
  C'est le fichier que vous venez de recevoir du serveur quand vous avez effectué la mise à jour. Ce fichier correspond à la révision `HEAD` du dépôt.

  Ici, *`ANCIENNE_REVISION`* est le numéro de révision du fichier dans votre répertoire `.svn` et *`NOUVELLE_REVISION`* est le numéro de révision de `HEAD` dans le dépôt.

Par exemple, Sally effectue un changement sur le fichier `sandwich.txt` mais elle ne propage pas immédiatement ses modifications. Pendant ce temps, Harry propage des changements sur ce même fichier. Sally met à jour sa copie de travail avant d'effectuer la propagation et un conflit apparaît, dont elle remet la résolution à plus tard :

```
$ svn update
Conflit découvert dans 'sandwich.txt'.
Sélectionner : (p) report, (df) diff-complet, (e) édite,
        (h) aide pour plus d'options : d
C  sandwich.txt
Actualisé à la révision 2.
$ ls -1
sandwich.txt
sandwich.txt.mine
sandwich.txt.r1
sandwich.txt.r2
```

À partir de là, Subversion n'autorisera *pas* Sally à propager le fichier `sandwich.txt` avant que les trois fichiers temporaires ne soient effacés :

```
$ svn commit -m "Quelques petits ajouts"
svn: Échec de la propagation (commit), détails :
svn: Arrêt de la propagation : '/home/sally/travail-svn/sandwich.txt'
                               demeure en conflit
```

Si vous avez remis à plus tard la résolution d'un conflit, vous devez le résoudre pour que Subversion vous autorise à propager vos changements. Vous pouvez le faire avec la commande **svn resolve** et l'option `--accept` suivie d'un argument.

Si vous choisissez la version du fichier que vous avez extraite avant de faire vos changements, utilisez l'argument *`base`*.

Si vous choisissez la version qui contient uniquement vos changements, utilisez l'argument *`mine-full`*.

Si vous choisissez la version la plus récente venant du serveur (et donc abandonnez tous vos changements), utilisez l'argument

theirs-full.

Cependant, si vous comptez effectuer un mélange de vos modifications et des modifications rapatriées du serveur, fusionnez le fichier en conflit « à la main » (examinez et éditez les marqueurs de conflit dans le fichier) puis utilisez l'argument *working*.

svn resolve supprime les trois fichiers temporaires et retient la version du fichier que vous avez spécifié avec l'option `--accept`. Subversion considère dès lors que le fichier n'est plus dans un état de conflit :

```
$ svn resolve --accept working sandwich.txt
Conflit sur 'sandwich.txt' résolu
```

Résoudre les conflits à la main

Résoudre les conflits à la main peut paraître quelque peu intimidant la première fois. Mais avec un peu de pratique, un enfant de cinq ans y arriverait.

Prenons un exemple. Par manque de communication entre Sally (votre collaboratrice) et vous-même, vous éditez en même temps le fichier `sandwich.txt`. Sally propage ses changements et, quand vous mettez à jour votre copie de travail, un conflit apparaît, que vous devez résoudre en éditant `sandwich.txt`. Jetons un œil à ce fichier :

```
$ cat sandwich.txt
Tranche de pain supérieure
Mayonnaise
Laitue
Tomate
Comté
<<<<<<< .mine
Saucisson
Mortadelle
Jambon
=======
Choucroute
Poulet rôti
>>>>>>> .r2
Moutarde
Tranche de pain inférieure
```

Les suites de caractères inférieur-à (<), égal(=) ou supérieur-à (>) sont des marqueurs de conflit, il ne font pas partie des données elles-mêmes. Vous devrez en général vous assurer qu'elles ont disparu du fichier avant de propager vos modifications. Le texte entre les deux premiers marqueurs est constitué des modifications que vous avez apportées dans la zone de conflit :

```
<<<<<<< .mine
Saucisson
Mortadelle
Jambon
=======
```

Le texte entre le deuxième et le troisième marqueur est celui du fichier propagé par Sally :

```
=======
Choucroute
Poulet rôti
>>>>>>> .r2
```

Normalement, vous n'allez pas juste supprimer les marqueurs et les changements effectués par Sally (elle sera affreusement déçue quand on lui apportera un sandwich différent de ce qu'elle a commandé). Vous décrochez donc le téléphone, ou vous traversez le bureau, pour expliquer à Sally qu'on ne met pas de choucroute dans un sandwich. [4] Après vous être mis d'accord

[4] Et si vous commandez ça, on vous chassera de la ville à coup de baguette rassie.

sur les changements à enregistrer, éditez votre fichier et enlevez les marqueurs de conflit.

```
Tranche de pain supérieure
Mayonnaise
Laitue
Tomate
Comté
Saucisson
Mortadelle
Jambon
Moutarde
Tranche de pain inférieure
```

Maintenant utilisez **svn resolve** et vous êtes paré pour propager vos changements :

```
$ svn resolve --accept working sandwich.txt
Conflit sur 'sandwich.txt' résolu
$ svn commit -m "Va pour mon sandwich et au diable celui de Sally !"
```

Notez que **svn resolve**, contrairement à la plupart des autres commandes que nous présentons dans ce chapitre, requiert que vous listiez explicitement les noms de tous les fichiers concernés. Dans tous les cas, soyez prudent et ne lancez **svn resolve** qu'une fois certain que vous avez résolu le conflit dans votre fichier (une fois les fichiers temporaires effacés, Subversion vous laisse propager le fichier même s'il contient toujours des marqueurs de conflit).

Si jamais vous êtes perdu lors de l'édition du fichier en conflit, vous pouvez toujours consulter les trois fichiers que Subversion a créé pour vous dans votre copie de travail, y compris le fichier tel qu'il était avant que vous ne lanciez la mise à jour. Vous pouvez même utiliser un outil externe interactif spécialisé dans les fusions pour examiner ces trois fichiers.

Abandonner vos modifications au profit de la révision la plus récente

Si vous faites face à un conflit et que vous décidez d'abandonner vos changements, vous pouvez lancer **`svn resolve --accept theirs-full`** ***`CHEMIN-DU-CONFLIT`***, et Subversion abandonne vos modifications et supprime les fichiers temporaires :

```
$ svn update
Conflit découvert dans 'machin.c'.
Sélectionner : (p) report, (df) diff complet, (e) édite,
        (h) aide pour plus d'options :
C    sandwich.txt
Actualisé à la révision 2.
$ ls sandwich.*
sandwich.txt  sandwich.txt.mine  sandwich.txt.r2  sandwich.txt.r1
$ svn resolve --accept theirs-full sandwich.txt
Conflit sur 'sandwich.txt' résolu
```

Revenir en arrière : utiliser svn revert

Si vous faites face à un conflit et qu'après examen de la situation, vous décidez d'abandonner vos changements et de repartir de zéro (peu importe en fait que ce soit après un conflit ou à n'importe quel autre moment), contentez-vous de revenir en arrière sur vos changements :

```
$ svn revert sandwich.txt
'sandwich.txt' réinitialisé
$ ls sandwich.*
sandwich.txt
```

Notez que quand vous revenez en arrière sur un fichier en conflit, vous n'avez pas besoin de lancer **svn resolve**.

Propager vos modifications

Enfin ! Vos modifications sont terminées, vous les avez fusionnées avec celles du serveur et vous êtes prêt à les propager vers le dépôt.

La commande **svn commit** envoie vos changements au dépôt. Quand vous propagez un changement, vous devez l'accompagner d'un *message de propagation* qui décrit ce changement. Votre message est associé à la nouvelle révision que vous créez. Si votre message est bref, vous pouvez le passer en ligne de commande en utilisant l'option `--message` (ou `-m`) :

```
$ svn commit -m "J'ai corrigé le nombre de tranches de fromage."
Envoi        sandwich.txt
Transmission des données .
Révision 3 propagée.
```

Cependant, si vous avez rédigé votre message au fur et à mesure, vous souhaitez sûrement indiquer à Subversion de récupérer le message dans un fichier en lui donnant le nom du fichier avec l'option `--file` (`-F`) :

```
$ svn commit -F message_de_propagation
Envoi        sandwich.txt
Transmission des données .
Révision 4 propagée.
```

Si vous ne spécifiez ni l'option `--message` ni l'option `--file`, Subversion lance automatiquement votre éditeur de texte favori (voir les détails de `editor-cmd` dans la section intitulée « Fichier `config` ») pour que vous rédigiez le message de propagation.

Si, au moment où vous rédigez votre message de propagation, vous décidez d'annuler la propagation, vous n'avez qu'à quitter l'éditeur de texte sans sauvegarder les changements. Si vous avez déjà sauvegardé le message, effacez le texte, sauvegardez à nouveau puis choisissez d'annuler :

```
$ svn commit
Attente de Emacs...Fait

Entrée du journal non modifié ou non précisé
a)nnule, c)ontinue, e)dite
a
$
```

Le dépôt ne sait pas si vos changements ont un sens ou pas (d'ailleurs, il s'en fiche) ; il vérifie seulement que personne n'a modifié, pendant que vous aviez le dos tourné, un des fichiers que vous-même avez modifié. Si *c'est le cas*, la propagation toute entière échoue, affichant un message vous informant qu'un ou plusieurs de vos fichiers ne sont plus à jour :

```
$ svn commit -m "Ajout d'une autre règle"
Envoi        regles.txt
svn: Echec de la propagation (commit), détails :
svn: Fichier '/regles.txt' obsolète
…
```

Notez que le phrasé exact de ce message d'erreur dépend du protocole réseau et du serveur que vous utilisez, mais l'idée reste la même.

Maintenant, vous devez lancer **`svn update`**, traiter les fusions ou conflits qui apparaissent et retenter une propagation.

Nous en avons terminé avec le cycle d'utilisation de base de Subversion. Subversion offre beaucoup d'autres fonctionnalités pour gérer votre dépôt et votre copie de travail, mais l'utilisation quotidienne de Subversion ne requiert pratiquement que les commandes que nous venons de voir dans ce chapitre. Intéressons-nous quand même à quelques commandes supplémentaires

utilisées relativement souvent.

Utilisation de l'historique

Votre dépôt Subversion est comme une machine à remonter le temps. Il garde une trace de tous les changements jamais propagés et permet de parcourir cet historique en examinant aussi bien les versions précédentes des fichiers et des répertoires que les métadonnées associées. D'une simple commande Subversion, vous pouvez extraire (ou restaurer) une copie de travail du dépôt tel qu'il était à n'importe quelle date ou numéro de révision passée. Cependant, vous voulez parfois juste *sonder* le passé sans y retourner.

Plusieurs commandes renvoient des informations sur l'historique des données présentes dans le dépôt :

svn log
: fournit beaucoup d'informations : les messages de propagation avec la date et l'auteur de la révision ainsi que les chemins qui ont été modifiés à chaque révision.

svn diff
: affiche les détails, ligne par ligne, d'un changement donné.

svn cat
: récupère le fichier tel qu'il existait à une révision donnée et l'affiche à l'écran.

svn list
: liste les fichiers contenus dans un répertoire à une révision donnée.

Affichage de l'historique

Pour connaître l'historique d'un fichier ou d'un répertoire, utilisez la commande **svn log**. Elle affiche la liste des gens qui ont modifié le fichier ou le répertoire en question, le numéro de chaque révision où il a changé, l'heure et la date de cette révision et, s'il y en avait un, le message associé à la propagation :

```
$ svn log
------------------------------------------------------------------------
r3 | sally | 2008-05-15 23:09:28 -0500 (jeu. 15 Mai 2008) | 1 ligne

Ajout des lignes include et correction du nombre de tranches de fromage.
------------------------------------------------------------------------
r2 | harry | 2008-05-14 18:43:15 -0500 (mer. 14 Mai 2008) | 3 lignes

Ajout des méthodes main().
------------------------------------------------------------------------
r1 | sally | 2008-05-10 19:50:31 -0500 (sam. 10 Mai 2008) | 1 ligne

Import initial
------------------------------------------------------------------------
```

Notez que, par défaut, l'historique est affiché en ordre chronologique inverse. Si vous voulez afficher un intervalle de révisions donné dans un ordre particulier ou juste une seule révision, ajoutez l'option `--revision (-r)` :

```
$ svn log -r 5:19    # affiche l'historique a partir de la révision 5
                     # jusqu'à la révision 19 dans l'ordre chronologique

$ svn log -r 19:5    # affiche l'historique à partir de la révision 5
                     # jusqu'à la révision 19 dans l'ordre
                     # chronologique inverse

$ svn log -r 8       # affiche l'historique de la révision 8
```

Vous pouvez aussi afficher l'historique d'un fichier ou d'un répertoire particulier. Par exemple :

```
$ svn log machin.c
…
$ svn log http://machin.com/svn/trunk/code/machin.c
…
```

Ceci n'affiche le contenu de l'historique *que* pour les révisions dans lesquelles le fichier de travail (ou l'URL) a changé.

Pourquoi svn log n'affiche-t-il pas ce que je viens de propager ?

Si vous effectuez une propagation puis tapez immédiatement **`svn log`** sans argument, vous remarquerez peut-être que votre propagation la plus récente est absente de l'historique obtenu. Ceci est dû à une combinaison de deux facteurs : la façon dont fonctionne **svn commit** et le fonctionnement par défaut de **svn log**. Tout d'abord, quand vous propagez des modifications vers le dépôt, Subversion ne récupère que la révision des fichiers (et répertoires) qu'il propage, donc le répertoire parent demeure généralement à l'ancienne révision (voir la section intitulée « Mise à jour et propagation sont deux choses distinctes » pour savoir pourquoi). La commande **svn log** ne récupère ensuite par défaut que l'historique du répertoire à la révision actuelle et n'affiche donc pas les modifications propagées dernièrement. La solution à ce problème consiste soit à mettre à jour votre copie de travail soit à fournir explicitement à **svn log** un numéro de révision grâce à l'option `--revision` (`-r`).

Si vous voulez obtenir plus d'informations sur un fichier ou un répertoire, svn log accepte également l'option `--verbose` (`-v`). Comme Subversion autorise les déplacements et les copies de répertoires et de fichiers, il est important de pouvoir tracer ces modifications de chemin dans le système de fichiers. Ainsi, en mode verbeux, **svn log** affiche la liste des déplacements au cours de la révision concernée :

```
$ svn log -r 8 -v
------------------------------------------------------------------------
r8 | sally | 2008-05-21 13:19:25 -0500 (mer. 21 Mai 2008) | 1 ligne
Chemins modifiés :
M /trunk/code/machin.c
M /trunk/code/bidule.h
A /trunk/code/doc/LISEZMOI

Machination du bidule.

------------------------------------------------------------------------
```

svn log accepte aussi l'option `--quiet` (`-q`), qui permet de ne pas afficher le contenu du message de propagation. En combinaison avec `--verbose`, **svn log** n'affiche que les noms des fichiers qui ont changé.

Pourquoi svn log me donne-t-il une réponse vide ?

Après un certain temps de pratique de Subversion, la plupart des utilisateurs sont confrontés à un affichage de ce genre :

```
$ svn log -r 2
------------------------------------------------------------------------
$
```

Au premier abord, cela ressemble à une erreur. Mais rappelez-vous que chaque révision concerne l'ensemble du dépôt et que **svn log** n'opère que sur une arborescence à l'intérieur du dépôt. Si vous ne passez pas d'argument pour le chemin, Subversion utilise le répertoire courant par défaut. En conséquence, si vous êtes dans un sous-répertoire de votre copie de travail et que vous demandez à voir l'historique d'une révision pour laquelle aucun changement n'a eu lieu sur lesdits fichiers et répertoires, Subversion affiche un historique vierge. Si vous voulez connaître tous les changements relatifs à cette révision, invoquez **svn log** avec l'URL du répertoire racine de votre dépôt, par exemple **`svn log -r 840076 http://svn.apache.org/repos/asf/subversion/`**.

Détail des modifications passées

Nous avons déjà vu la commande **svn diff**, qui affiche les différences entre fichiers au format diff unifié ; nous l'avons utilisée pour afficher les modifications locales effectuées sur notre copie de travail avant de les propager vers le dépôt.

En fait, il y a *trois* façons différentes d'utiliser **svn diff** :

- Examiner des modifications locales.
- Comparer votre copie de travail au dépôt.
- Comparer des révisions du dépôt.

Modifications locales

Comme nous l'avons vu précédemment, **`svn diff`**, s'il est invoqué sans option, compare les fichiers de votre copie de travail à leurs versions « originales » gardées en cache dans la zone `.svn` :

```
$ svn diff
Index: regles.txt
===================================================================
--- regles.txt (révision 3)
+++ regles.txt (copie de travail)
@@ -1,4 +1,5 @@
 Être attentif envers les autres
 Liberté = Responsabilité
 Tout dans la modération
-Mâcher la bouche ouverte
+Mâcher la bouche fermée
+Écouter quand les autres parlent
$
```

Comparaison d'une copie de travail au dépôt

Si un seul numéro de révision est fourni à l'option `--revision` (`-r`), votre copie de travail est comparée à la révision spécifiée du dépôt :

```
$ svn diff -r 3 regles.txt
Index: regles.txt
===================================================================
--- regles.txt (révision 3)
+++ regles.txt (copie de travail)
@@ -1,4 +1,5 @@
 Être attentif envers les autres
 Liberté = Responsabilité
 Tout dans la modération
-Mâcher la bouche ouverte
+Mâcher la bouche fermée
+Écouter quand les autres parlent
$
```

Comparaison de révisions du dépôt

Si deux numéros de révision sont fournis à l'option `--revision` (`-r`), séparés par le caractère deux-points (`:`), les deux révisions sont directement comparées :

```
$ svn diff -r 2:3 regles.txt
Index: regles.txt
===================================================================
```

```
--- regles.txt (révision 2)
+++ regles.txt (révision 3)
@@ -1,4 +1,4 @@
 Être attentif envers les autres
-Liberté = Glace Au Chocolat
+Liberté = Responsabilité
 Tout dans la modération
 Mâcher la bouche ouverte
$
```

Une autre façon de comparer une révision à la précédente, plus conviviale, est d'utiliser l'option `--change` (`-c`) :

```
$ svn diff -c 3 regles.txt
Index: regles.txt
===================================================================
--- regles.txt (révision 2)
+++ regles.txt (révision 3)
@@ -1,4 +1,4 @@
 Être attentif envers les autres
-Liberté = Glace Au Chocolat
+Liberté = Responsabilité
 Tout dans la modération
 Mâcher la bouche ouverte
$
```

Enfin, vous pouvez comparer des révisions du dépôt même si vous n'avez pas de copie de travail en local sur votre ordinateur, simplement en incluant l'URL appropriée sur la ligne de commande :

```
$ svn diff -c 5 http://svn.exemple.com/depot/exemple/trunk/texte/regles.txt
…
$
```

Navigation dans le dépôt

Grâce aux commandes **svn cat** et **svn list**, vous pouvez afficher des révisions variées des fichiers et répertoires sans changer la révision de votre copie de travail. En fait, vous n'avez même pas besoin d'avoir une copie de travail pour les utiliser.

svn cat

Si vous voulez examiner une version antérieure d'un fichier et pas nécessairement les différences entre deux fichiers, vous pouvez utiliser **svn cat** :

```
$ svn cat -r 2 regles.txt
Être attentif envers les autres
Liberté = Glace Au Chocolat
Tout dans la modération
Mâcher la bouche ouverte
$
```

Vous pouvez également rediriger la sortie de **svn cat** directement dans un fichier :

```
$ svn cat -r 2 regles.txt > regles.txt.v2
$
```

svn list

La commande **svn list** liste les fichiers présents dans le dépôt sans pour autant les télécharger :

```
$ svn list http://svn.apache.org/repos/asf/subversion/
README
branches/
developer-resources/
mk.xiv/
site/
svn-logos/
tags/
trunk/
```

Si vous désirez une liste plus détaillée, passez l'option `--verbose` (`-v`) et vous obtenez alors quelque chose comme ceci :

```
$ svn list -v http://svn.apache.org/repos/asf/subversion/
 904709 hwright               30 janv., 03:06 ./
 880872 cmpilato         5362 16 nov., 18:49 README
 904644 danielsh              29 janv., 23:05 branches/
 861356 lgo                   27 août 2006 developer-resources/
 868798 brane                 02 janv. 2008 mk.xiv/
 904663 cmpilato              30 janv., 00:19 site/
 863801 anonymou              02 sept. 2002 svn-logos/
 901971 cmpilato              22 janv., 04:39 tags/
 904709 hwright               30 janv., 03:06 trunk/
```

Les colonnes vous indiquent la révision à laquelle le fichier ou le répertoire a été modifié pour la dernière fois, qui est l'auteur de ce changement, la taille du fichier si c'en est un, la date de dernière modification et le nom de l'élément.

La commande **svn list** sans argument prend pour cible l'*URL du dépôt* correspondant au répertoire local en cours, pas le répertoire en cours de la copie de travail. Après tout, si vous voulez voir le contenu de votre répertoire local, vous pouvez utiliser **ls**, tout simplement (ou l'équivalent sur votre système non-Unix).

Anciennes versions d'un dépôt

En plus de toutes les commandes citées précédemment, vous pouvez utiliser **svn update** et **svn checkout** avec l'option `--revision` pour ramener une copie de travail complète « dans le passé » : [5]

```
$ svn checkout -r 1729 # extrait une nouvelle copie de travail
                       # à la révision r1729
…
$ svn update -r 1729 # met à jour une copie de travail existante
                     # à la révision r1729
…
```

Beaucoup de nouveaux utilisateurs de Subversion essaient d'utiliser **svn update** comme dans l'exemple précédent pour annuler des changements propagés, mais ça ne marche pas, puisque vous ne pouvez pas propager des changements obtenus en ramenant à une vieille version une copie de travail, si les fichiers modifiés ont subi des modifications depuis. Voir la section intitulée « Résurrection des éléments effacés » pour une description de la manière d' « annuler » une propagation.

Enfin, si vous êtes en train de réaliser une version officielle et que vous voulez extraire vos fichiers de Subversion sans avoir ces satanés répertoires `.svn`, vous pouvez utiliser **svn export** pour créer une copie locale de tout ou partie de votre dépôt sans les répertoires `.svn`. De même que pour **svn update** et **svn checkout**, vous pouvez passer l'option `--revision` à **svn export** :

[5] Vous voyez, on vous avait bien dit que Subversion était une machine à remonter le temps.

```
$ svn export http://svn.exemple.com/svn/depot1 # Exporte la dernière révision
…
$ svn export http://svn.exemple.com/svn/depot1 -r 1729
# Exporte la révision r1729
…
```

Parfois, il suffit de faire le ménage

Maintenant que nous avons traité les tâches quotidiennes pour lesquelles vous utiliserez Subversion, nous allons passer en revue quelques tâches administratives liées à votre copie de travail.

Se débarrasser d'une copie de travail

Subversion ne conserve sur le serveur aucune trace de l'état ni de l'existence des copies de travail, il n'y a donc aucun impact côté serveur si des copies de travail traînent un peu partout. De la même façon, pas besoin de prévenir le serveur quand vous effacez une copie de travail.

Si vous envisagez de réutiliser une copie de travail, ça ne pose aucun problème de la laisser sur le disque jusqu'à ce que vous soyez prêts à l'utiliser à nouveau et, le moment venu, il suffit de lancer **svn update** pour la mettre à jour et ainsi la rendre utilisable.

Cependant, si vous êtes certain de ne plus utiliser une copie de travail, vous pouvez la supprimer entièrement, mais vous seriez bien inspirés d'y jeter un œil au cas où des fichiers non-suivis en versions s'y trouveraient encore. Pour trouver ces fichiers, lancez **`svn status`** et examinez tous les fichiers marqués d'un `?` pour vous assurer qu'ils ne sont d'aucune importance. Une fois cet examen terminé, vous pouvez supprimer votre copie de travail en toute sécurité.

Reprendre après une interruption

Quand Subversion modifie votre copie de travail (ou toute information dans `.svn`), il essaie de le faire de la manière la plus sûre possible. Avant de modifier votre copie de travail, Subversion inscrit ses intentions dans un fichier de traces. Ensuite, il exécute les commandes du fichier de traces pour appliquer les modifications demandées, en plaçant un verrou sur la partie concernée de la copie de travail pendant cette opération (pour empêcher d'autres clients Subversion d'accéder à cette copie de travail au beau milieu des changements). Pour finir, Subversion supprime le fichier de traces. D'un point de vue architectural, c'est le même fonctionnement qu'un système de fichiers journalisé. Si une opération Subversion est interrompue (c'est-à-dire le processus est tué ou la machine plante), le fichier de traces reste sur le disque. En exécutant de nouveau le fichier de traces, Subversion peut terminer l'opération en cours et votre copie de travail retrouve un état cohérent.

C'est exactement ce que fait la commande **svn cleanup** : elle trouve et exécute les fichiers de traces restant dans votre copie de travail, en enlevant les verrous au passage. Si un beau jour Subversion vous indique qu'une partie de votre copie de travail est verrouillée (« locked » en anglais), c'est la commande qu'il faut lancer. Par ailleurs, **svn status** affiche un `L` devant les éléments verrouillés :

```
$ svn status
  L    un-repertoire
M      un-repertoire/machin.c

$ svn cleanup
$ svn status
M      un-repertoire/machin.c
```

Ne confondez pas ces verrous agissant sur la copie de travail avec les verrous ordinaires que les utilisateurs de Subversion créent quand ils utilisent le modèle de gestion de versions parallèles verrouiller-modifier-libérer ; voir l'encadré Les trois types de « verrous » pour des éclaircissements.

Résumé

Nous en avons maintenant terminé avec la plupart des commandes du client Subversion. Les exceptions notables concernent les branches et la fusion (voir le Chapitre 4, *Gestion des branches*) ainsi que les propriétés (voir le Chapitre 9, *Références*

complètes de Subversion). Cependant, prenez le temps de parcourir le Chapitre 9, *Références complètes de Subversion* pour vous faire une idée de toutes les commandes de Subversion et de la manière dont vous pouvez les utiliser pour rendre votre travail plus convivial.

Chapitre 3. Sujets avancés

Si vous lisez ce livre chapitre par chapitre, du début à la fin, vous avez acquis maintenant suffisamment de connaissance du fonctionnement de Subversion pour effectuer les opérations les plus courantes de gestion de versions. Vous savez comment extraire une copie de travail du dépôt Subversion. Vous n'avez aucune difficulté à propager vos modifications et à recevoir des mises à jour en utilisant les commandes **svn commit** et **svn update**. Vous avez probablement acquis le réflexe, presque inconscient, de lancer la commande **svn status**. Bref, vous êtes apte à utiliser Subversion dans un environnement normal pour tout type de projet.

Mais les fonctionnalités de Subversion ne s'arrêtent pas aux « opérations courantes de gestion de versions ». Il possède d'autres atouts, en plus de permettre le partage de fichiers et de répertoires depuis un dépôt central.

Ce chapitre dévoile certaines fonctionnalités de Subversion qui, bien qu'importantes, ne sont pas d'une utilisation quotidienne pour un utilisateur normal. Nous supposons que vous êtes familier avec les possibilités de base de gestion de versions sur les fichiers et répertoires. Sinon, reportez-vous au Chapitre 1, *Notions fondamentales* et au Chapitre 2, *Utilisation de base*. Une fois que vous maîtriserez ces bases et que vous aurez assimilé ce chapitre, vous serez un super-utilisateur de Subversion !

Identifiants de révision

Comme vous avez pu le constater dans la section intitulée « Révisions », les numéros de révision dans Subversion sont d'une grande simplicité, formant une suite d'entiers incrémentés au fur et à mesure des changements propagés dans le dépôt. Néanmoins, il ne faudra pas longtemps avant que vous ne puissiez plus vous rappeler exactement quel changement correspond à quelle révision. Heureusement, le fonctionnement normal de Subversion ne requiert pas souvent que vous fournissiez explicitement un numéro de révision pour une opération. Pour les opérations qui nécessitent *vraiment* un numéro de révision, vous pouvez fournir un numéro de révision que vous avez vu soit dans un mail de propagation, soit dans la sortie d'une autre opération Subversion, soit dans un autre contexte où ce numéro possédait une signification particulière.

Occasionnellement, vous avez besoin d'identifier un moment précis pour lequel vous n'avez pas de numéro de révision en tête ou sous la main. C'est pourquoi, en plus des numéros de révision, **svn** accepte également en entrée d'autres formats d'appellations pour les révisions : les *mots-clés de révision* et les dates de révision.

Les différentes formes d'appellations pour les révisions peuvent être mélangées et comparées pour définir des intervalles de révisions. Par exemple, vous pouvez spécifier `-r` *`REV1`*`:`*`REV2`* où *`REV1`* est un mot-clé de révision et *`REV2`* est un numéro de révision, ou bien où *`REV1`* est une date et *`REV2`* est un numéro de révision. Comme chaque appellation de révision est évaluée indépendamment, vous pouvez placer n'importe quel type d'appellation de chaque côté du symbole deux-points.

Mots-clés de révision

Le client Subversion accepte une grande variété de mots-clés de révision. En tant qu'argument de l'option `--revision` (`-r`) ces mots-clés peuvent être utilisés en lieu et place des numéros et sont remplacés par les numéros correspondants par Subversion :

`HEAD`
: La dernière (c'est-à-dire la plus récente) révision présente dans le dépôt.

`BASE`
: Le numéro de révision d'un élément de la copie de travail. Si l'élément a été modifié localement, la « version BASE » fait référence à l'élément tel qu'il était sans ces modifications locales.

`COMMITTED`
: La révision la plus récente avant (ou égale à) `BASE`, dans laquelle un élément a changé.

`PREV`
: La révision *précédant* immédiatement la dernière révision dans laquelle un élément a changé. Techniquement, cela revient à `COMMITTED`#1.

Comme vous pouvez le deviner d'après leur description, les mots-clés de révision `PREV`, `BASE` et `COMMITTED` ne sont utilisés que pour faire référence à un chemin dans la copie de travail ; ils ne s'appliquent pas à des URL du dépôt. En revanche, `HEAD` peut être utilisé avec les deux types de chemin (local ou URL du dépôt).

Vous trouvez ci-dessous des exemples de l'utilisation de ces mots-clés :

```
$ svn diff -r PREV:COMMITTED machin.c
# affiche le dernier changement propagé concernant machin.c

$ svn log -r HEAD
# affiche le message associé à la dernière propagation dans le dépôt.

$ svn diff -r HEAD
# compare votre copie de travail (avec tous ses changements locaux)
# à la dernière version de l'arborescence correspondante du dépôt.

$ svn diff -r BASE:HEAD machin.c
# compare la version non modifiée localement de machin.c avec la dernière
# version de machin.c dans le dépôt.

$ svn log -r BASE:HEAD
# affiche, pour le répertoire suivi en versions courant, les messages
# de propagation depuis la dernière mise à jour (svn update).

$ svn update -r PREV machin.c
# revient une version en arrière pour le fichier machin.c. Ceci diminue
# de un la révision de la version de travail du fichier machin.c.

$ svn diff -r BASE:14 machin.c
# compare la version non modifiée localement de machin.c avec
# la version de ce fichier à la révision 14.
```

Dates de révision

Les numéros de révision n'ont aucune signification en dehors du système de gestion de versions. Cependant, parfois, vous avez besoin d'associer une date réelle à un moment précis de l'historique des versions. À cette fin, l'option `--revision` (`-r`) accepte comme argument une date placée entre accolades (`{` et `}`). Subversion accepte les dates et les heures aux formats définis dans le standard ISO-8601 ainsi que quelques autres formats. Voici quelques exemples (n'oubliez pas de mettre les dates qui contiennent des espaces entre guillemets) :

```
$ svn checkout -r {2006-02-17}
$ svn checkout -r {15:30}
$ svn checkout -r {15:30:00.200000}
$ svn checkout -r {"2006-02-17 15:30"}
$ svn checkout -r {"2006-02-17 15:30 +0230"}
$ svn checkout -r {2006-02-17T15:30}
$ svn checkout -r {2006-02-17T15:30Z}
$ svn checkout -r {2006-02-17T15:30-04:00}
$ svn checkout -r {20060217T1530}
$ svn checkout -r {20060217T1530Z}
$ svn checkout -r {20060217T1530-0500}
…
```

Quand vous spécifiez une date, Subversion convertit cette date vers le numéro de révision le plus récent du dépôt à la date spécifiée. Puis, il continue son travail avec ce numéro de révision :

```
$ svn log -r {2006-11-28}
------------------------------------------------------------------------
r12 | ira | 2006-11-27 12:31:51 -0600 (lun. 27 nov. 2006) | 6 lignes
…
```

Subversion retarde-t-il d'une journée ?

Si vous spécifiez une date de révision sans préciser l'heure (par exemple `2006-11-27`), vous pourriez penser que Subversion vous donne la dernière révision qui a eu lieu le 27 novembre. En fait, vous aurez une révision datant du 26, voire même avant. Souvenez-vous que Subversion renvoie *la révision la plus récente du dépôt* à la date spécifiée. Si vous spécifiez une date sans préciser l'heure, comme `2006-11-27`, Subversion utilise alors 00h00 comme heure et la recherche de la plus récente révision ne renvoit donc pas de résultat correspondant au 27 novembre.

Si vous voulez inclure le 27 dans votre recherche, vous pouvez soit spécifier une heure (`{"2006-11-27 23:59"}`), soit simplement spécifier le jour suivant (`{2006-11-28}`).

Vous pouvez également utiliser des intervalles de dates. Subversion trouve alors les révisions incluses entre ces deux dates :

```
$ svn log -r {2006-11-20}:{2006-11-29}
…
```

Puisque l'horodatage d'une révision est stocké comme une propriété modifiable et non suivie en versions de la révision (reportez-vous à la section intitulée « Propriétés »), les horodatages peuvent être changés et ne pas refléter la chronologie réelle. Ils peuvent même être tous supprimés. Or la capacité de Subversion à convertir correctement les dates en numéros de révision dépend des horodatages de révisions et de leur ordonnancement correct dans le temps : à une révision antérieure correspond un horodatage antérieur. Si cet ordonnancement n'est pas maintenu, il y a de grandes chances que l'utilisation des dates pour spécifier des intervalles de révisions dans votre dépôt ne fournisse pas les résultats attendus.

Propriétés

Nous avons vu en détail comment Subversion stocke et récupère les différentes versions des fichiers et répertoires dans le dépôt. Des chapitres entiers ont décrit cette fonctionnalité fondamentale de l'outil. Et si la gestion de versions se limitait à ça, Subversion couvrirait déjà complètement les besoins attendus.

Mais ce n'est pas tout.

En plus de gérer les versions de vos répertoires et de vos fichiers, Subversion fournit une interface pour ajouter, modifier et supprimer des méta-données suivies en versions pour chacun de vos répertoires et de vos fichiers. On appelle ces méta-données des *propriétés*. Elles peuvent être pensées comme des tableaux à deux colonnes, qui associent des noms de propriétés à des valeurs arbitraires, pour chaque élément de votre copie de travail. En termes simples, vous pouvez assigner n'importe quel nom et n'importe quelle valeur à vos propriétés, à la seule condition que le nom soit un texte lisible par un humain. Et l'atout principal de ces propriétés réside dans le fait qu'elles sont également suivies en versions, tout comme le contenu textuel de vos fichiers. Vous pouvez modifier, propager et revenir en arrière sur les propriétés aussi facilement que sur le contenu des fichiers. L'envoi et la réception des changements concernant les propriétés intervient lors de vos propagations et mises à jour : vous n'avez pas à changer vos habitudes pour les utiliser.

Subversion a réservé pour son propre usage les propriétés dont le nom commence par `svn:`. Bien qu'il n'y en ait seulement que quelques unes d'utilisées actuellement, vous ne devez pas créer vos propres propriétés avec un nom commençant par ce préfixe. Sinon, vous courez le risque qu'une future version de Subversion définisse une propriété ayant le même nom mais un usage tout autre.

Les propriétés sont aussi présentes ailleurs dans Subversion. De la même manière que pour les fichiers et répertoires, chaque révision en tant que telle peut avoir des propriétés arbitraires associées. Les mêmes contraintes s'appliquent : nom lisible par un humain et valeur arbitraire, éventuellement binaire. La différence principale est que les propriétés des révisions ne sont pas suivies en versions. Autrement dit, si vous changez la valeur ou si vous supprimez une propriété d'une révision, il n'y a pas moyen, en utilisant Subversion, de revenir à la valeur précédente.

Subversion ne fournit pas de recommandation précise quant à l'utilisation des propriétés. Il demande seulement de ne pas utiliser de nom de propriété qui commence par le préfixe `svn:`. C'est l'espace de noms qu'il garde pour son propre usage. Et

Subversion utilise bien lui-même les propriétés, suivies en versions ou pas. Certaines propriétés suivies en versions ont une signification particulière ou des effets particuliers quand elles font référence à un fichier ou à un répertoire, ou stockent des informations relatives à la révision à laquelle elles font référence. Certaines propriétés de révision sont automatiquement rattachées à une révision par la procédure de propagation et stockent des informations relatives à cette révision. La plupart de ces propriétés sont mentionnées ailleurs dans ce chapitre ou dans d'autres chapitres comme faisant partie de sujets plus généraux. Pour une liste exhaustive des propriétés pré-définies de Subversion, référez-vous à la section intitulée « Propriétés dans Subversion ».

Dans cette section, nous examinons l'utilité des propriétés, à la fois pour l'utilisateur et pour Subversion lui-même. Vous apprendrez les sous-commandes **svn** relatives aux propriétés et comment la modification des propriétés change votre manière habituelle d'utiliser Subversion.

Utilisation des propriétés

À l'instar de Subversion, qui utilise les propriétés pour stocker des méta-données sur les fichiers, les répertoires et les révisions qu'il gère, vous pouvez faire une utilisation similaire des propriétés. Vous pouvez trouver utile d'avoir un endroit, près de vos données suivies en versions, pour stocker des méta-données relatives à vos données.

Imaginons que vous vouliez créer un site Web qui héberge beaucoup de photos et qui les affiche avec une légende et une date. D'accord, mais votre collection de photos change constamment, donc vous voulez automatiser le plus possible la gestion du site. Ces photos peuvent être relativement volumineuses et vous voulez pouvoir fournir des miniatures à vos visiteurs, comme c'est généralement le cas sur ce genre de site.

Certes, vous pouvez le faire en utilisant des fichiers traditionnels. C'est-à-dire que vous avez votre `image123.jpg` et une `image123-thumbnail.jpg` côte à côte dans un répertoire. Ou, si vous voulez garder les mêmes noms de fichier, vous placez vos miniatures dans un répertoire différent, comme `thumbnails/image123.jpg`. Vous pouvez également stocker vos légendes et dates de la même façon, séparées encore une fois du fichier image original. Mais le problème est que votre collection de fichiers s'agrandit de plusieurs fichiers à chaque nouvelle photo ajoutée au site.

Maintenant, considérons le même site Web conçu en utilisant les propriétés des fichiers fournies par Subversion. Imaginez un simple fichier image, `image123.jpg`, et un ensemble de propriétés relatives à ce fichier nommées `légende`, `date` et même `miniature`. À présent, le répertoire de votre copie de travail se gère beaucoup plus facilement ; en fait, vu du navigateur, il semble ne contenir que des images. Mais vos scripts d'automatisation vont plus loin : ils savent qu'ils peuvent utiliser les commandes **svn** (ou mieux, ils peuvent utiliser les connecteurs spécifiques au langage utilisé, voir la section intitulée « Utiliser les API ») pour extraire les informations dont votre site a besoin sans avoir à lire un fichier d'index ou à jouer avec des chemins de fichiers.

Bien que Subversion n'impose que peu de restrictions sur les noms et les valeurs des propriétés, il n'a pas été conçu pour gérer de façon optimale des valeurs de propriétés de grande taille ou un grand nombre de propriétés sur un fichier ou un répertoire donné. Subversion garde souvent en mémoire en même temps tous les noms et valeurs de propriétés associés à un élément, ce qui peut engendrer des problèmes de performance lors de l'utilisation de très gros ensembles de propriétés.

On utilise également fréquemment des propriétés de révisions personnalisées. Une utilisation classique est d'avoir une propriété qui contient un identifiant en provenance d'un autre outil de gestion et de l'associer à une révision. Par exemple, l'outil de gestion est utilisé pour suivre les bogues et la révision corrige le bogue associé à l'identifiant. Il s'agit parfois aussi d'utiliser des noms plus conviviaux pour les révisions : il peut être difficile de se remémorer que la révision 1935 correspond à une révision qui a subi la totalité des tests, alors qu'une propriété `resultat-des-tests` avec la valeur `tout ok` est autrement plus utile.

Retrouver ses petits (ou savoir *ne pas utiliser* les propriétés)

Bien que très utiles, les propriétés Subversion, ou plus exactement les interfaces disponibles pour y accéder, ont une lacune majeure : alors qu'il est très simple de *définir* une propriété personnalisée, la *retrouver* plus tard est une toute autre affaire.

Trouver une propriété de révision personnalisée implique généralement d'effectuer un parcours linéaire de toutes les révisions du dépôt, en demandant à chacune : « Avez-vous la propriété que je cherche ? ». Trouver une propriété personnalisée suivie en versions est également difficile et implique souvent un appel récursif à **svn propget** sur toute une copie de travail. Dans votre situation, c'est peut-être moins pire que le parcours linéaire de toutes les révisions. Mais cela

laisse certainement beaucoup à désirer en termes de performance et de probabilité de réussite, surtout si, pour votre recherche, il faut une copie de travail de la racine de votre dépôt.

C'est pourquoi vous pouvez choisir, en particulier pour ce qui concerne les propriétés de révisions, de simplement ajouter les méta-données au message de propagation. Par exemple, utilisez une politique de formatage (idéalement appliquée automatiquement par un script) conçue pour être rapidement analysée à partir de la sortie de **svn log**. Ainsi, il est assez fréquent de voir dans Subversion des messages de propagation qui ressemblent à :

```
Problème(s): IZ2376, IZ1919
Corrigé par:  sally

Corrige un méchant plantage dans la fonction machin bidule
…
```

Mais hélas, cela ne résout pas tout. Subversion ne fournit pas encore de mécanisme pour gérer des modèles de messages associés aux propagations, ce qui aiderait pourtant beaucoup les utilisateurs à respecter le format des méta-données qu'ils placent dans les messages de révision.

Manipulation des propriétés

La commande **svn** offre différentes possibilités pour ajouter ou modifier des propriétés sur les fichiers et les répertoires. Pour les propriétés avec des valeurs courtes, lisibles par un humain, la solution la plus simple est sûrement de spécifier le nom de la propriété et sa valeur en ligne de commande avec la sous-commande **svn propset** :

```
$ svn propset copyright '(c) 2006 Red-Bean Software' calc/bouton.c
Propriété 'copyright' définie sur 'calc/bouton.c'
$
```

Mais nous avons vanté la souplesse de Subversion pour spécifier les valeurs des propriétés. Ainsi, si vous envisagez d'avoir des valeurs de plusieurs lignes de texte, ou même une valeur binaire, la passer en ligne de commande ne vous convient pas. La sous-commande **svn propset** accepte donc l'option `--file` (`-F`) pour spécifier le nom d'un fichier qui contient la nouvelle valeur de la propriété.

```
$ svn propset licence -F /chemin/vers/LICENCE calc/bouton.c
Propriété 'licence' définie sur 'calc/bouton.c'
$
```

Il y a quelques restrictions sur les noms de propriétés. Un nom de propriété doit commencer par une lettre, le caractère deux points (`:`), ou le caractère souligné (_) ; ensuite, vous pouvez utiliser des chiffres, des tirets (`-`) et des points (`.`) [1].

En plus de la commande **propset**, **svn** dispose de la commande **propedit**. Cette commande utilise l'éditeur de texte pré-configuré (reportez-vous à la section intitulée « Fichier `config` ») pour ajouter ou modifier des propriétés. Quand vous exécutez la commande, **svn** lance votre éditeur de texte avec un fichier temporaire qui contient la valeur actuelle de la propriété (ou un contenu vierge si vous ajoutez une nouvelle propriété). Vous pouvez alors modifier la valeur dans l'éditeur de texte pour y placer votre nouvelle valeur, sauvegarder le fichier temporaire et quitter l'éditeur. Si Subversion détecte que la valeur a effectivement changé, il la prend en compte. Si vous quittez l'éditeur sans faire de changement, la propriété n'est pas modifiée :

```
$ svn propedit copyright calc/bouton.c  ### sortez de l'éditeur sans faire de
modification
Pas de modification de la propriété 'copyright' sur 'calc/bouton.c'
$
```

Vous pouvez noter que, à l'instar des autres commandes **svn**, celles relatives aux propriétés fonctionnent aussi sur des chemins multiples. Vous pouvez ainsi modifier les propriétés d'un ensemble de fichiers en une seule commande. Par exemple, nous

[1] Pour ceux qui connaissent le XML, c'est à peu près le sous-ensemble ASCII pour la syntaxe du champ "Name" en XML.

aurions pu taper :

```
$ svn propset copyright '(c) 2006 Red-Bean Software' calc/*
Propriété 'copyright' définie sur 'calc/Makefile'
Propriété 'copyright' définie sur 'calc/bouton.c'
Propriété 'copyright' définie sur 'calc/entier.c'
…
$
```

Toutes ces manipulations de propriétés ne seraient pas vraiment utiles si vous ne pouviez pas récupérer facilement la valeur d'une propriété. Subversion propose donc deux sous-commandes pour afficher les noms et les valeurs des propriétés associées aux fichiers et répertoires. La commande **svn proplist** fournit la liste des noms de propriétés qui existent dans un chemin. Une fois que vous connaissez les noms des propriétés d'un élément, vous pouvez obtenir les valeurs correspondantes avec la commande **svn propget**. Cette commande affiche sur la sortie standard la valeur de la propriété dont le nom et le chemin (ou l'ensemble des chemins) ont été passés en paramètres.

```
$ svn proplist calc/bouton.c
Propriétés sur 'calc/bouton.c':
  copyright
  licence
$ svn propget copyright calc/bouton.c
(c) 2006 Red-Bean Software
$
```

Il y a même une variante de la commande **proplist** qui liste à la fois le nom et la valeur de toutes les propriétés. Ajoutez simplement l'option `--verbose` (`-v`) à la commande :

```
$ svn proplist -v calc/bouton.c
Propriétés sur 'calc/bouton.c':
  copyright : (c) 2006 Red-Bean Software
  license : ================================================================
Copyright (c) 2006 Red-Bean Software.  All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions, and the recipe for Fitz's famous
red-beans-and-rice.
…
```

La dernière sous-commande relative aux propriétés est **propdel**. Puisque Subversion vous autorise à stocker des propriétés avec une valeur vide, vous ne pouvez pas supprimer une propriété en utilisant **svn propedit** ou **svn propset**. Par exemple, la commande suivante *ne fournit pas* le résultat escompté :

```
$ svn propset licence calc/bouton.c
Propriété 'licence' définie sur 'calc/bouton.c'
$ svn proplist -v calc/bouton.c
Propriétés sur 'calc/bouton.c':
  copyright : (c) 2006 Red-Bean Software
  licence :
$
```

Vous devez utiliser la sous-commande **propdel** pour supprimer complètement une propriété. La syntaxe est similaire aux autres commandes sur les propriétés :

```
$ svn propdel license calc/bouton.c
Propriété 'licence' supprimée de 'calc/bouton.c'.
```

```
$ svn proplist -v calc/bouton.c
Propriétés sur 'calc/bouton.c':
  copyright : (c) 2006 Red-Bean Software
$
```

Vous souvenez-vous des propriétés de révision non suivies en versions ? Vous pouvez les modifier elles-aussi en utilisant les mêmes sous-commandes **svn** que nous venons de décrire. Il suffit juste d'ajouter l'option `--revprop` à la ligne de commande et de spécifier la révision à laquelle s'applique la modification. Puisque les numéros de révisions s'appliquent à l'ensemble de l'arborescence, vous n'avez pas besoin d'indiquer un chemin pour ces commandes, du moment que vous êtes dans une copie de travail du dépôt contenant la révision dont vous voulez modifier la propriété. Autrement, vous pouvez simplement fournir n'importe quelle URL du dépôt en question (y compris l'URL racine). Par exemple, imaginons que vous vouliez remplacer le message associé à la propagation d'une révision précédente [2]. Si le répertoire actuel fait partie de votre copie de travail du dépôt, vous pouvez simplement lancer la commande **svn propset** sans spécifier de chemin :

```
$ svn propset svn:log '* bouton.c: Corrige un avertissement du compilateur.' -r11
--revprop
Nouvelle valeur définie pour la propriété 'svn:log' à la révision du dépôt '11'
$
```

Et même si vous n'avez pas extrait de copie de travail du dépôt, vous pouvez toujours modifier la propriété en indiquant l'URL racine du dépôt :

```
$ svn propset svn:log '* bouton.c: Corrige un avertissement du compilateur.' -r11
--revprop \
               http://svn.exemple.com/depot/projet
Nouvelle valeur définie pour la propriété 'svn:log' à la révision du dépôt '11'
$
```

Notez que le droit de modifier cette propriété non suivie en versions doit être explicitement ajouté par l'administrateur du dépôt (voir la section intitulée « Correction des messages de propagation »). En effet, la propriété n'étant pas suivie en versions, vous risquez une perte d'informations si vous la modifiez à tort et à travers. L'administrateur du dépôt peut mettre en place des protections contre ce type d'incident et, par défaut, la modification de propriétés non suivies en versions est désactivée.

Dans la mesure du possible, il est recommandé d'utiliser **svn propedit** au lieu de **svn propset**. Bien que le résultat soit identique, la première permet de visualiser la valeur actuelle de la propriété que l'on veut modifier, ce qui aide à vérifier que l'on fait bien ce que l'on pense faire. C'est particulièrement vrai dans le cas des propriétés non suivies en versions. Il est aussi beaucoup plus facile de modifier un texte de plusieurs lignes dans un éditeur de texte qu'en ligne de commande.

Les propriétés et le cycle de travail Subversion

Maintenant que vous êtes familier avec toutes les sous-commandes **svn** relatives aux propriétés, voyons comment la modification des propriétés change le cycle habituel d'utilisation de Subversion. Comme mentionné précédemment, les propriétés des fichiers et répertoires sont suivies en versions, à l'instar du contenu des fichiers. En conséquence, Subversion offre les mêmes possibilités pour fusionner (proprement ou quand apparaissent des conflits) vos modifications avec celles des autres collaborateurs.

De même que pour le contenu des fichiers, les modifications de propriétés sont locales. Elles ne deviennent permanentes que quand vous les propagez dans le dépôt via **svn commit**. Vos modifications sur les propriétés peuvent aussi être annulées facilement : la commande **svn revert** restaure vos fichiers et répertoires dans leur état d'avant les modifications, y compris pour les propriétés. Vous pouvez également obtenir des informations intéressantes sur l'état des propriétés de vos fichiers et répertoires en utilisant les commandes **svn status** et **svn diff**.

```
$ svn status calc/bouton.c
```

[2] Corriger les fautes d'orthographe, les erreurs de grammaire et les informations simplement erronées au sein des messages de propagation est peut-être l'usage le plus courant de l'option `--revprop`.

```
 M     calc/bouton.c
$ svn diff calc/bouton.c
Modification de propriétés sur calc/bouton.c
___________________________________________________________________
Ajouté: copyright
   + (c) 2006 Red-Bean Software

$
```

Remarquez que la sous-commande **status** place le `M` dans la deuxième colonne plutôt que dans la première. C'est parce que nous avons modifié les propriétés de `calc/bouton.c`, mais pas son contenu. Si nous avions changé les deux, nous aurions vu le `M` dans la première colonne également (reportez-vous à la section intitulée « Avoir une vue d'ensemble des changements effectués »).

Conflits sur les propriétés

De la même manière que pour le contenu des fichiers, les modifications locales effectuées sur les propriétés peuvent entrer en conflit avec les changements effectués par d'autres collaborateurs. Si vous faites une mise à jour de votre copie de travail et que vous recevez un changement incompatible avec vos propres modifications d'une propriété d'un objet suivi en versions, Subversion vous indique que l'objet est dans un état de conflit.

```
% svn update calc
M  calc/Makefile.in
 C calc/bouton.c
À la révision 143.
$
```

Subversion crée également, dans le même répertoire que l'objet en conflit, un fichier avec l'extension `.prej` qui contient les détails du conflit. Vous devez examiner le contenu de ce fichier pour décider comment résoudre le conflit. Tant que le conflit n'est pas résolu, la sortie de **svn status** affiche un `C` dans la deuxième colonne pour cet objet et vos tentatives de propagation échouent.

```
$ svn status calc
 C     calc/bouton.c
?      calc/bouton.c.prej
$ cat calc/bouton.c.prej
prop 'nombre_lignes': user set to '1256', but update set to '1301'.
$
```

Pour résoudre les conflits sur les propriétés, assurez-vous simplement que les propriétés en question contiennent bien les valeurs qu'elle doivent contenir, puis utilisez la commande **svn resolved** pour indiquer à Subversion que vous avez résolu le problème manuellement.

Vous avez peut-être remarqué que Subversion affiche les différences au niveau des propriétés d'une manière non standard. Certes, vous pouvez toujours rediriger la sortie de **svn diff** pour créer un fichier correctif utilisable : le programme patch ignore ce qui concerne les propriétés (comme il ignore tout ce qu'il ne comprend pas). Malheureusement, cela signifie aussi que pour appliquer intégralement un correctif généré par **svn diff**, les modifications concernant les propriétés doivent être faites à la main.

Configuration automatique des propriétés

Les propriétés constituent une fonctionnalité très puissante de Subversion et sont un élément central de nombreuses fonctionnalités de Subversion présentées ailleurs dans ce chapitre ainsi que dans les autres chapitres : comparaisons et fusions textuelles, substitution de mots-clés, transformation des retours à la ligne, etc. Mais pour profiter pleinement des propriétés, il faut les placer sur les répertoires et fichiers adéquats. Malheureusement, cette étape peut passer à la trappe dans le train-train quotidien, d'autant plus qu'oublier de configurer une propriété n'engendre généralement pas une erreur qui saute aux yeux (du moins comparativement à oublier d'ajouter un fichier dans la gestion de versions). Pour vous aider à placer vos propriétés au

bon endroit, Subversion propose deux fonctionnalités simples mais néanmoins utiles.

Au moment d'introduire un fichier en suivi de versions à l'aide de la commande **svn add** ou **svn import**, Subversion essaie de vous aider en configurant automatiquement certaines propriétés communes des fichiers. D'abord, sur les systèmes d'exploitation dont le système de fichiers utilise un bit « exécutable », Subversion ajoute automatiquement la propriété `svn:executable` aux nouveaux fichiers, ajoutés ou importés, qui ont ce bit activé (voir la section intitulée « Fichiers exécutables ou non » pour plus de détails sur cette propriété).

Ensuite, Subversion essaie de déterminer le type MIME du fichier. Si vous avez configuré le paramètre `mime-types-files`, Subversion essaie de trouver un type MIME correspondant à l'extension du nom de fichier. Si un tel type MIME existe, il définit automatiquement la propriété `svn:mime-type` avec la valeur du type trouvé. S'il ne trouve pas de type MIME correspondant ou s'il n'existe pas de fichier définissant les correspondances, Subversion applique une heuristique très basique pour déterminer si le contenu du fichier est lisible par un humain. Si le résultat est négatif, Subversion ajoute automatiquement la propriété `svn:mime-type` à ce fichier avec la valeur `application/octet-stream` (type MIME générique indiquant « une suite d'octets »). Bien sûr, si Subversion se trompe ou si vous voulez indiquer un type plus précis (par exemple `image/png` ou `application/x-shockwave-flash`), vous pouvez toujours supprimer ou modifier cette propriété (pour d'avantage d'informations sur la gestion des types MIME par Subversion, reportez vous à la section intitulée « Type de contenu des fichiers » plus loin dans ce chapitre).

Subversion fournit également, via sa zone de configuration (voir la section intitulée « Zone de configuration des exécutables »), une fonction de renseignement automatique des propriétés, plus flexible, qui vous permet de créer des associations entre, d'une part, des motifs de noms de fichiers et, d'autre part, des noms de propriétés / valeurs de propriétés. Là encore, ces associations modifient le comportement des commandes **add** et **import**, pouvant non seulement passer outre la décision prise par défaut d'attribution d'une propriété de type MIME mais pouvant aussi définir d'autres propriétés, qu'elles soient utilisées par Subversion ou personnalisées. Par exemple, vous pouvez créer une association qui, à chaque ajout d'un fichier JPEG (c'est-à-dire dont le nom est du type `*.jpg`), fixe la propriété `svn:mime-type` de ce fichier à la valeur `image/jpeg`. Ou alors affecte à tout fichier de type `*.cpp` la propriété `svn:eol-style` avec la valeur `native` et la propriété `svn:keywords` avec la valeur `Id`. L'affectation automatique de propriétés est sûrement l'outil de manipulation des propriétés le plus commode de tout Subversion. Reportez-vous à la section intitulée « Fichier `config` » pour plus d'informations sur la configuration de cette fonction.

Portabilité des fichiers

Heureusement pour les utilisateurs de Subversion qui travaillent sur différents ordinateurs et systèmes d'exploitation, le comportement du programme en ligne de commande est pratiquement identique sur tous les systèmes. Si vous vous débrouillez avec **svn** sur un système, vous devriez vous en sortir sur n'importe quel système.

Cependant, ce n'est pas toujours le cas pour d'autres types de logiciels ou pour les fichiers que vous gérez dans Subversion. Par exemple, sur un système Windows, la définition d'un « fichier texte » est similaire à la définition de Linux, mais avec une différence notable pour ce qui concerne les retours à la ligne. Il y a aussi d'autres différences. Les plateformes Unix (et Subversion) supportent la notion de lien symbolique ; Windows non. Les plateformes Unix utilisent les permissions du fichier pour déterminer si un fichier est exécutable ; Windows utilise l'extension du fichier.

Subversion n'a pas la possibilité d'unifier toutes ces définitions et ces implémentations. Tout ce qu'il peut faire, c'est aider au maximum l'utilisateur qui travaille sur plusieurs systèmes et plusieurs ordinateurs. Cette section décrit comment Subversion s'y prend.

Type de contenu des fichiers

Subversion fait partie des nombreuses applications qui reconnaissent et utilisent les types MIME (Multipurpose Internet Mail Extensions). Ainsi, la valeur de la propriété `svn:mime-type` permet, en plus de stocker le type de contenu d'un fichier, de changer le comportement de Subversion lui-même.

Identification des types de fichiers

Beaucoup de programmes sur les systèmes d'exploitation modernes font des suppositions sur le type et le format du contenu d'un fichier à partir de son nom, notamment son extension. Par exemple, les fichiers qui se terminent par `.txt` sont généralement considérés comme lisibles par un être humain, aptes à être compris pratiquement tels quels, sans nécessiter un processus de décodage compliqué. Les fichiers dont le nom se termine par `.png`, en revanche, sont considérés comme des fichiers du type "Portable Network Graphics", illisibles pour un être humain, et utilisables

uniquement au travers d'un logiciel capable de comprendre le format PNG et de l'afficher en tant qu'image matricielle.

Malheureusement, certaines de ces extensions ont changé de sens au fil du temps. Au début des ordinateurs personnels, un fichier `LISEZMOI.DOC` aurait certainement été un simple fichier texte, comme aujourd'hui les fichiers `.txt`. Mais, rendu au milieu des années 1990, vous pouviez parier que ce fichier ne serait plus un simple fichier texte, mais un document "Microsoft Word", format propriétaire et illisible pour un être humain. Ce changement n'a pas eu lieu du jour au lendemain et il y a eu une période de confusion pour les utilisateurs qui, lorsqu'ils tombaient sur un fichier `.doc`, ne savaient pas trop de quel type était ce fichier [3].

L'essor des réseaux informatiques n'a fait qu'ajouter à la confusion sur la relation entre le nom d'un fichier et son contenu. Avec l'information circulant à travers les réseaux, souvent générée dynamiquement par des programmes sur les serveurs, il n'y avait plus de fichier en tant que tel et donc plus de nom de fichier. Les serveurs Web, par exemple, avaient besoin d'un autre moyen pour indiquer au navigateur quel type de contenu il télécharge afin qu'il puisse appliquer un traitement cohérent à cette information : soit afficher les données à l'aide d'un programme qui sait traiter ce type de contenu, soit demander à l'utilisateur où stocker les données téléchargées.

Finalement, un standard est apparu pour, entre autres, décrire le contenu d'un flux de données. En 1996 était publiée la RFC2045, la première des cinq RFC à décrire le format MIME. Elle décrit le concept de types de média et de sous-types et recommande une syntaxe pour représenter ces types. Aujourd'hui, les types de média MIME (ou simplement types MIME) sont utilisés de manière pratiquement universelle par les clients de messagerie, les serveurs Web et autres logiciels, pour déterminer de manière sûre le type de contenu d'un fichier.

Par exemple, un avantage fourni par cette reconnaissance de type par Subversion est la possibilité de fusion contextuelle, ligne par ligne, des changements reçus lors d'une mise à jour. En revanche, pour les fichiers contenant autre chose que du texte, il n'y a souvent pas de concept de « ligne ». En conséquence, pour les fichiers suivis en versions dont la propriété `svn:mime-type` contient une valeur de type MIME non textuel (généralement un intitulé qui ne commence pas par `text/`, bien qu'il y ait des exceptions), Subversion ne tente pas de fusion contextuelle pendant la mise à jour. À la place, chaque fois que vous avez modifié localement un fichier binaire qui a été mis à jour sur le dépôt, Subversion ne touche pas à votre fichier mais crée deux nouveaux fichiers. Un fichier avec l'extension `.oldrev` qui contient la version du fichier à la révision BASE. Un autre fichier avec l'extension `.newrev` qui contient la version à jour du fichier. Ce comportement est dicté par la volonté d'éviter que l'utilisateur ne tente d'effectuer une fusion qui échouerait parce que les fichiers ne peuvent tout simplement pas être fusionnés.

La propriété `svn:mime-type`, si elle n'est pas correctement définie à une valeur qui indique un contenu non textuel, peut causer des comportements inattendus. Par exemple, comme la « fin de ligne » n'a pas de sens dans un fichier binaire, Subversion vous empêche de définir la propriété `svn:eol-style` sur ces fichiers. Cela saute aux yeux quand vous travaillez sur un seul fichier et que **svn propset** génère une erreur. C'est beaucoup moins évident si vous effectuez une opération récursive, où Subversion omet silencieusement les fichiers qu'il considère inappropriés pour une propriété donnée.

Depuis Subversion 1.5, les utilisateurs peuvent configurer un nouveau paramètre `mime-types-file` dans la zone de configuration qui identifie un fichier de correspondance des types MIME. Subversion consulte ce fichier pour déterminer le type MIME de tout nouveau fichier ajouté ou importé.

Par ailleurs, si la propriété `svn:mime-type` est définie, alors le greffon Apache pour Subversion utilise cette valeur pour renseigner le champ `Content-type:` de l'en-tête HTTP en réponse à une requête GET. Cela fournit au navigateur Web une indication très importante pour pouvoir afficher correctement le fichier, quand vous l'utilisez pour parcourir le contenu du dépôt Subversion.

Fichiers exécutables ou non

Sur beaucoup de systèmes d'exploitation, la capacité de rendre un fichier exécutable dépend d'un bit dit « exécutable ». Habituellement, ce bit est désactivé par défaut et doit être explicitement activé par l'utilisateur pour chaque fichier concerné. Ce serait une perte de temps énorme d'avoir à se rappeler exactement quel fichier, parmi ceux que l'on vient d'extraire du dépôt, doit avoir le bit exécutable positionné et ensuite de devoir le faire manuellement. C'est pourquoi Subversion fournit la propriété `svn:executable` pour spécifier que le bit exécutable doit être activé pour le fichier concerné. Subversion s'occupe lui-même de cette tâche quand il rapatrie de tels fichiers dans la copie de travail locale.

[3] Ca vous semble dur ? Et bien, à la même période, WordPerfect utilisait aussi `.DOC` comme extension préférée de son format de fichier propriétaire !

Cette propriété n'a aucun effet sur les systèmes de fichiers qui ne possèdent pas le concept du bit exécutable, tels que FAT32 et NTFS [4]. Par ailleurs, bien qu'elle n'ait pas de valeurs définies, Subversion lui attribue la valeur `*` lorsqu'il active cette propriété. Enfin, cette propriété n'est valide que sur des fichiers, pas sur des répertoires.

Caractères de fin de ligne

Pour tout fichier suivi en versions, Subversion considère que le contenu est lisible par un humain sauf si la propriété `svn:mime-type` indique le contraire. En règle générale, Subversion utilise cette information pour déterminer s'il est possible d'effectuer une comparaison contextuelle pour ce fichier. Sinon, pour Subversion, les octets sont des octets.

Cela veut dire que par défaut, Subversion ne s'intéresse pas au type de caractère utilisé pour marquer les *fins de lignes* (« EOL » en anglais, pour « End Of Line »). Malheureusement, des conventions différentes sont utilisées suivant les systèmes d'exploitation pour indiquer une fin de ligne de texte dans un fichier. Par exemple, les logiciels sous Windows utilisent généralement une paire de caractères de contrôle ASCII : un retour chariot (`CR`, « carriage return ») suivi par un saut de ligne (`LF`, « line feed »). Les logiciels Unix, cependant, utilisent uniquement le caractère `LF` pour indiquer les fins de lignes.

Tous les programmes ne savent pas gérer les fichiers utilisant un marqueur de fin de ligne « exogène » au système d'exploitation sur lequel ils tournent. Ainsi, il n'est pas rare de voir les programmes Unix traiter le marqueur `CR` des fichiers Windows comme un caractère normal (en affichant à l'écran un `^M`) et les programmes Windows combiner en une seule ligne immense un fichier Unix parce qu'ils n'y ont pas trouvé la combinaison retour chariot-passage à la ligne (`CR-LF`).

Cette incapacité de traiter correctement les marqueurs de fin de ligne d'autres plates-formes peut être assez frustrante pour ceux qui partagent des fichiers entre différents systèmes d'exploitation. Prenons l'exemple d'un fichier de code source qui est édité par des développeurs à la fois sous Windows et sous Unix. Si tous les développeurs utilisent des outils qui se plient à la convention utilisée par le fichier, pas de problème.

Mais, en pratique, de nombreux outils largement utilisés soit ne parviennent pas à lire correctement un fichier utilisant une convention différente pour les fins de ligne, soit ils convertissent les fins de lignes au format local lors de la sauvegarde. Dans le premier cas, le développeur doit utiliser des outils externes (tels que **dos2unix** et son compagnon **unix2dos**) pour préparer le fichier avant l'édition. Dans le deuxième cas, pas besoin de préparation. Mais dans les deux cas, le fichier résultant diffère de l'original littéralement pour toutes les lignes ! Avant de propager ses changements, l'utilisateur a deux choix. Soit il utilise un utilitaire de conversion pour revenir à la même convention qu'avant l'édition. Soit il propage le fichier avec la nouvelle convention de fin de ligne.

Au final, les deux hypothèses conduisent à une perte de temps et des modifications inutiles sur les fichiers propagés. La perte de temps est déjà pénible. Mais si en plus la propagation change chaque ligne du fichier, trouver quelle ligne a effectivement changé devient non trivial. À quel endroit ce bogue a-t-il réellement été corrigé ? Dans quelle ligne y avait-il cette erreur de syntaxe ?

La solution à ce problème est la propriété `svn:eol-style` (eol pour "End Of Line"). Quand cette propriété possède une valeur valide, Subversion l'utilise pour déterminer quel traitement il doit appliquer pour que le fichier ne change pas de convention à chaque propagation provenant d'un système d'exploitation différent. Les valeurs valides sont :

`native`
: Ceci force le fichier à adopter la convention utilisée par le système d'exploitation sur lequel s'exécute Subversion. En d'autres termes, si un utilisateur d'une machine Windows récupère une copie de travail d'un fichier dont la propriété `svn:eol-style` vaut `native`, ce fichier contiendra le marqueur `CRLF` pour indiquer les fins de ligne. Un utilisateur Unix qui récupère une copie de travail qui contient le même fichier verra simplement `LF` pour indiquer les fins de ligne sur sa copie.

  Notez que Subversion stocke en fait le fichier dans le dépôt en utilisant le marqueur standard `LF` indépendamment du système d'exploitation. Cela reste toutefois tout à fait transparent pour l'utilisateur.

`CRLF`
: Le fichier contiendra le marqueur `CRLF` pour indiquer les fins de ligne, quel que soit le système d'exploitation.

`LF`
: Le fichier contiendra le marqueur `LF` pour indiquer les fins de ligne, quel que soit le système d'exploitation.

`CR`

[4] Les systèmes de fichiers Windows utilisent les extensions des fichiers (telles que `.EXE`, `.BAT` et `.COM`) pour indiquer que les fichiers sont exécutables.

Le fichier contiendra le marqueur `CR` pour indiquer les fins de ligne, quel que soit le système d'exploitation. Ce marqueur de fin de ligne n'est pas très courant. Il était utilisé sur les vieux Macintosh (machines sur lesquelles Subversion ne tourne même pas).

Occultation des éléments non suivis en versions

Dans n'importe quelle copie de travail, il y a de grandes chances que les fichiers et répertoires suivis en versions côtoient d'autres fichiers et répertoires non suivis en versions ou qui n'ont pas lieu de l'être. Les éditeurs de texte remplissent les répertoires avec des fichiers de sauvegarde. Les compilateurs créent des fichiers intermédiaires (ou même des fichiers finaux) que vous ne voudrez pas suivre en versions. Et les utilisateurs eux-mêmes déposent des fichiers et des répertoires où bon leur semble, souvent dans des copies de travail locales.

Il est ridicule de penser que les copies de travail Subversion échappent à ce type de méli-mélo. En fait, Subversion prend en compte (c'est une *fonctionnalité*) dès le début que les copies de travail sont des répertoires comme les autres, comme ceux qui ne sont pas suivis en versions. Mais ces fichiers et répertoires qui-n-ont-pas-vocation-à-être-suivis-en-versions peuvent perturber les utilisateurs de Subversion. Par exemple, comme les commandes **svn add** et **svn import** sont récursives par défaut et ne savent pas quels fichiers de l'arborescence vous voulez suivre ou non en versions, il est relativement facile d'ajouter au suivi de versions des éléments que vous ne vouliez pas suivre. Et comme la commande **svn status** liste, par défaut, tous les éléments intéressants de la copie de travail, y compris les fichiers et répertoires non suivis en versions, son affichage devient rapidement confus avec de tels imbroglios.

C'est pourquoi Subversion fournit deux méthodes pour lui indiquer quels fichiers vous souhaitez ignorer. La première implique l'utilisation de la zone de configuration (voir la section intitulée « Zone de configuration des exécutables ») et, par conséquent, s'applique à toutes les opérations Subversion qui utilisent cette zone de configuration, généralement toutes celles de l'ordinateur ou d'un utilisateur particulier de l'ordinateur. L'autre méthode utilise les propriétés Subversion des répertoires et est plus liée à l'arborescence suivie en versions elle-même. Par conséquent, elle affecte tous ceux qui possèdent une copie de travail de cette arborescence. Les deux mécanismes utilisent des *motifs de noms de fichiers* (des chaines de caractères simples ou des jokers pour trouver des correspondances avec les noms de fichiers).

La zone de configuration de Subversion propose une option, `global-ignores`, dont la valeur est un ensemble de motifs de noms de fichiers (appelés aussi globs) séparés par des espaces. Le client Subversion compare ces motifs aux noms des fichiers que l'on tente d'ajouter au suivi de versions, ainsi qu'aux noms des fichiers non suivis en versions détectés par la commande **svn status**. Si un nom de fichier correspond au motif, Subversion ignore totalement ce fichier. C'est particulièrement utile pour les fichiers que vous ne voulez jamais suivre en versions, comme les fichiers de sauvegarde créés par les éditeurs de texte (par exemple, les fichiers `*~` et `.*~` créés par Emacs).

Les motifs de noms de fichiers dans Subversion

Les motifs de noms de fichiers (également appelés *globs* ou motifs de filtrages du shell) sont des chaînes de caractères destinées à être comparées à des noms de fichiers, en général dans le but de sélectionner rapidement un sous-ensemble de fichiers similaires au sein d'un ensemble plus large, sans avoir à nommer explicitement chaque fichier. Les motifs contiennent deux types de caractères : les caractères standard, qui sont comparés explicitement aux noms de fichiers, et les caractères spéciaux (aussi nommés quantificateurs), qui sont interprétés différemment.

Il y a différents types de syntaxe de motifs, mais Subversion utilise celle qui est la plus répandue sur les systèmes Unix, implémentée dans la fonction `fnmatch`. Elle reconnaît les caractères spéciaux suivants, décrits ici à titre d'information :

`?`
Correspond à n'importe quel caractère unique.

`*`
Correspond à n'importe quelle chaîne de caractères, y compris la chaîne vide.

`[`
Marque le début de la définition d'une classe de caractères, se terminant par `]`, utilisée pour décrire un sous-ensemble de caractères.

Vous pouvez observer ce même filtrage de motifs à l'invite de commandes d'un shell Unix. Voici des exemples de motifs utilisés pour différentes choses :

```
$ ls   ### les sources du livre
appa-quickstart.xml             ch06-server-configuration.xml
appb-svn-for-cvs-users.xml      ch07-customizing-svn.xml
appc-webdav.xml                 ch08-embedding-svn.xml
book.xml                        ch09-reference.xml
ch00-preface.xml                ch10-world-peace-thru-svn.xml
ch01-fundamental-concepts.xml   copyright.xml
ch02-basic-usage.xml            foreword.xml
ch03-advanced-topics.xml        images/
ch04-branching-and-merging.xml  index.xml
ch05-repository-admin.xml       styles.css
$ ls ch*   ###  les chapitres du livre
ch00-preface.xml                ch06-server-configuration.xml
ch01-fundamental-concepts.xml   ch07-customizing-svn.xml
ch02-basic-usage.xml            ch08-embedding-svn.xml
ch03-advanced-topics.xml        ch09-reference.xml
ch04-branching-and-merging.xml  ch10-world-peace-thru-svn.xml
ch05-repository-admin.xml
$ ls ch?0-*   ### les chapitres du livre dont le numéro se termine par 0
ch00-preface.xml  ch10-world-peace-thru-svn.xml
$ ls ch0[3578]-*   ### les chapitres du livre dont Mike est responsable
ch03-advanced-topics.xml   ch07-customizing-svn.xml
ch05-repository-admin.xml  ch08-embedding-svn.xml
$
```

Le filtrage par motif de fichiers est un peu plus complexe que ce que nous avons décrit ici, mais ce niveau d'utilisation semble suffire à la majorité des utilisateurs de Subversion.

Pour un répertoire suivi en versions, la propriété `svn:ignore` est supposée contenir une liste de motifs de noms de fichiers (un motif par ligne) que Subversion utilise pour déterminer quels objets ignorer dans le répertoire concerné. Ces motifs ne remplacent pas les motifs inscrits dans la directive `global-ignores` de la zone de configuration, mais s'ajoutent à cette liste. Veuillez également noter que, contrairement à la directive `global-ignores`, les motifs de la propriété `svn:ignore` s'appliquent uniquement au répertoire pour lequel la propriété est définie et pas à ses sous-répertoires. La propriété `svn:ignore` est utile pour indiquer à Subversion d'ignorer les fichiers susceptibles d'être présents dans la copie de travail de ce répertoire chez chaque utilisateur comme par exemple les fichiers produits par les compilateurs ou, pour citer un exemple plus approprié à ce livre, les fichiers HTML, PDF ou PostScript générés par la conversion des fichiers sources DocBook XML vers un format de fichier plus lisible.

Le support des motifs de fichiers à ignorer dans Subversion s'applique uniquement à la procédure d'ajout de fichiers et répertoires non suivis en versions vers la gestion de versions. Une fois que l'objet est suivi en versions par Subversion, les mécanismes permettant d'ignorer certains fichiers selon des motifs prédéfinis ne s'appliquent plus. Autrement dit, ne pensez pas que Subversion ne propagera pas les changements que vous avez faits à un fichier suivi en versions simplement parce que son nom correspond à un motif à ignorer : Subversion prend *toujours* en compte l'ensemble des objets qu'il gère.

Motifs de fichier à ignorer pour les utilisateurs de CVS

La syntaxe et le fonctionnement de la propriété `svn:ignore` de Subversion sont très similaires à ceux du fichier `.cvsignore` de CVS. Si vous migrez une copie de travail CVS vers Subversion, vous pouvez migrer directement les motifs à ignorer en utilisant le fichier `.cvsignore` comme entrée à la commande **svn propset** :

```
$ svn propset svn:ignore -F .cvsignore .
Propriété 'svn:ignore' définie sur '.'
$
```

Il y a quand même quelques différences entre CVS et Subversion concernant les motifs à ignorer. Les deux systèmes n'utilisent pas les motifs au même moment et il y a quelques légères divergences sur ce sur quoi ils s'appliquent.

D'ailleurs, Subversion ne reconnaît pas le motif `!` pour revenir à une situation où aucun motif n'est ignoré.

La liste globale des motifs à ignorer est une affaire de goût, devant plus s'intégrer à la collection d'outils d'un utilisateur particulier que répondre aux besoins d'une copie de travail particulière. C'est pourquoi le reste de cette section s'attache à décrire l'utilisation de la propriété `svn:ignore`.

Prenons par exemple le résultat suivant de la commande **svn status** :

```
$ svn status calc
 M      calc/bouton.c
?       calc/calculatrice
?       calc/donnees.c
?       calc/debug_log
?       calc/debug_log.1
?       calc/debug_log.2.gz
?       calc/debug_log.3.gz
```

Dans cet exemple, des modifications ont été faites sur les propriétés de `bouton.c` et il y a aussi des fichiers non suivis en versions : le programme `calculatrice` (résultat de votre dernière compilation du code source), un fichier source `donnees.c` et un ensemble de fichiers de traces pour le débogage. Vous êtes conscient du fait que compiler votre code engendre à chaque fois la création du programme `calculatrice` [5]. Vous savez également que vous avez toujours des fichiers de traces qui traînent. On peut faire ce constat pour toutes les copies de travail locales de ce projet, pas seulement la vôtre. Et vous savez que cela ne vous intéresse pas et que cela n'intéresse très probablement aucun autre développeur, de voir ces fichiers apparaître à chaque commande **svn status**. Vous allez donc utiliser **`svn propedit svn:ignore calc`** pour ajouter des motifs à ignorer pour le répertoire `calc`. Par exemple, vous pouvez ajouter ce qui suit comme nouvelle valeur de la propriété `svn:ignore` :

```
calculatrice
debug_log*
```

Après avoir ajouté cette propriété, vous avez une propriété modifiée localement dans votre répertoire `calc`. Mais notez les autres différences sur le résultat de la commande **svn status** :

```
$ svn status
 M      calc
 M      calc/bouton.c
?       calc/donnees.c
```

Maintenant, tout le superflu a disparu ! Bien sûr, votre programme compilé et les fichiers de trace sont toujours présents dans votre copie locale. Subversion ne vous présente pas ces fichiers présents mais non suivis en versions, c'est tout. Et maintenant que ces parasites sont supprimés de l'affichage, il ne vous reste plus que les éléments intéressants, tels que le fichier source `donnees.c` que vous avez probablement oublié d'ajouter au suivi de versions.

Bien évidemment, ce compte-rendu plus succinct de l'état de votre copie de travail locale n'est pas le seul possible. Si vous voulez voir les fichiers ignorés dans le compte-rendu, vous pouvez ajouter l'option `--no-ignore` à la commande Subversion :

```
$ svn status --no-ignore
 M      calc
 M      calc/bouton.c
I       calc/calculatrice
?       calc/donnees.c
I       calc/debug_log
I       calc/debug_log.1
I       calc/debug_log.2.gz
```

[5] N'est-ce pas précisément la finalité d'un système de compilation ?

```
I      calc/debug_log.3.gz
```

Comme mentionné auparavant, la liste des motifs de fichiers à ignorer est aussi utilisée par **svn add** et **svn import**. Ces deux opérations demandent à Subversion de gérer un ensemble de fichiers et de répertoires. Plutôt que de forcer l'utilisateur à choisir dans l'arborescence quels fichiers il souhaite suivre en versions, Subversion utilise les motifs de fichiers à ignorer, à la fois la liste globale et ceux définis par répertoire, pour déterminer quels fichiers suivre (ou ne pas suivre) en versions dans sa procédure récursive d'ajout ou d'import. Là encore, vous pouvez utiliser l'option `--no-ignore` pour indiquer à Subversion d'ignorer ces listes et de d'agir effectivement sur tous les fichiers et répertoires présents.

Même si `svn:ignore` est défini, vous risquez de rencontrer des problèmes si vous utilisez des caractères spéciaux du shell dans une commande. Les caractères spéciaux sont remplacés par une liste explicite de cibles avant que Subversion n'agisse sur eux et donc lancer **svn *SOUS-COMMANDE* *** revient à lancer **svn *SOUS-COMMANDE* fichier1 fichier2 fichier3 …**. Dans le cas de la commande **svn add**, ceci a un effet similaire à l'option `--no-ignore`. Par conséquent, au lieu d'utiliser un caractère spécial, utilisez plutôt **svn add --force .** pour marquer d'un seul coup les éléments non suivis en versions pour ajout. La cible explicite permet de s'assurer que le répertoire en cours ne sera pas négligé car déjà suivi en versions et l'option `--force` force Subversion à parcourir ce répertoire, ajoutant les fichiers non suivis en versions, tout en respectant la propriété `svn:ignore` et la variable `global-ignores` de la zone de configuration. Pensez à rajouter l'option `--depth files` à la commande **svn add** si vous ne voulez pas que la recherche de fichiers à ajouter au suivi de versions ne parcoure le répertoire de façon récursive.

Substitution de mots-clés

Subversion a la capacité de substituer des *mots-clés* dans les fichiers suivis en versions par des informations dynamiques et utiles. Les mots-clés fournissent généralement des indications sur les dernières modifications faites au fichier. Comme ces informations changent à chaque fois que le fichier change et, plus spécifiquement, juste *après* que le fichier change, c'est compliqué pour tout processus, excepté pour le système de gestion de versions, de garder les données à jour. Sans outil automatique, adieu la pertinence de ces informations !

Par exemple, prenons un document dont vous voulez afficher la date de dernière modification. Vous pouvez charger chaque contributeur du document de renseigner le champ correspondant juste avant de propager ses changements. Mais un jour ou l'autre, quelqu'un oubliera de le faire. Demandez plutôt à Subversion de substituer le mot-clé `LastChangedDate`. Vous contrôlez où est inséré le mot-clé dans votre document en plaçant un signet à l'endroit voulu dans le fichier. Ce signet est juste une chaîne de caractères formatée comme ceci : `$`*`NomDuMotCle`*`$`.

Tous les mots-clés sont sensibles à la casse des caractères quand ils apparaissent en tant que signets : vous devez placer les majuscules aux bons endroits pour que le mot-clé soit effectivement remplacé. Vous devez aussi considérer que la valeur de la propriété `svn:keywords` est sensible à la casse (« case-sensitive » en anglais) : certains mots-clés seront reconnus indépendamment de la casse, mais ce comportement est obsolète.

Subversion définit la liste des mots-clés disponibles pour les substitutions. Cette liste contient les cinq mots-clés suivants (certains d'entre eux ont des alias que vous pouvez aussi utiliser) :

`Date`
: Ce mot-clé indique la date du dernier changement connu dans le dépôt. Il est de la forme `$Date: 2006-07-22 21:42:37 -0700 (sam. 22 juil. 2006) $`. Il peut également être spécifié en tant que `LastChangedDate`. Contrairement au mot-clé `Id`, qui utilise l'heure UTC, le mot-clé `Date` affiche la date et l'heure locales.

`Revision`
: Ce mot-clé indique la dernière révision connue pour laquelle le fichier a changé dans le dépôt. Il fournit une réponse du type `$Revision: 144 $`. Il peut aussi être spécifié en tant que `LastChangedRevision` ou `Rev`.

`Author`
: Ce mot-clé indique le dernier utilisateur qui a changé le fichier dans le dépôt et retourne une valeur du type `$Author: harry $`. Il peut aussi être spécifié en tant que `LastChangedBy`.

`HeadURL`
: Ce mot-clé décrit l'URL complète de la dernière version du fichier dans le dépôt et ressemble à `$HeadURL:`

`http://svn.apache.org/repos/asf/subversion/trunk/README $`. Il peut être abrégé en `URL`.

`Id`
: Ce mot-clé est une combinaison abrégée des autres mots-clés. Sa substitution donne quelque chose comme `$Id: calc.c 148 2006-07-28 21:30:43Z sally $`, que l'on interprète comme suit : « Le fichier `calc.c` a été modifié en dernier par l'utilisateur `sally` lors de la révision 148 le 28 juillet 2006 au soir. » La date et l'heure affichées sont en heure UTC, contrairement au mot-clé `Date` qui utilise l'heure locale.

Une bonne partie des définitions qui précèdent utilisent la locution « dernière … connue » ou quelque chose d'équivalent. Rappelez-vous que la substitution des mots-clés est une opération effectuée côté client et que votre client ne connaît pas les changements qui ont eu lieu dans le dépôt depuis votre dernière mise à jour. Si vous ne mettez jamais à jour de votre copie de travail locale, vos mots-clés restent figés à la même valeur même si des changements ont lieu régulièrement dans le dépôt.

Ajouter des signets dans votre fichier ne fait rien de spécial. Subversion n'essaie jamais d'effectuer la substitution dans votre fichier tant que vous ne le lui demandez pas explicitement. Après tout, vous écrivez peut-être un document [6] sur la manière d'utiliser les mots-clés et vous ne voulez pas que Subversion substitue à vos beaux exemples de signets non substitués leur valeur réelle !

Pour indiquer à Subversion de substituer ou pas les mots-clés d'un fichier particulier, nous utilisons une fois de plus les commandes sur les propriétés. La propriété `svn:keywords`, quand elle est activée pour un fichier, contrôle quels mots-clés doivent être substitués dans ce fichier. Elle doit contenir une liste de mots-clés ou d'alias cités précédemment, séparés par des espaces.

Par exemple, pour un fichier nommé `meteo.txt` qui ressemble à ceci :

```
Voici les dernières prévisions de nos spécialistes :
$LastChangedDate$
$Rev$
Les cumulus sont de plus en plus nombreux au fur et à mesure que l'été approche.
```

Sans la propriété `svn:keywords` activée sur ce fichier, Subversion ne fait rien de spécial. À présent, si nous activons les substitutions pour le mot-clé `LastChangedDate` :

```
$ svn propset svn:keywords "Date Author" meteo.txt
property 'svn:keywords' set on 'meteo.txt'
$
```

Vous venez d'effectuer une modification locale des propriétés du fichier `meteo.txt`. Vous ne verrez aucun changement dans le contenu du fichier (à moins d'avoir fait des modifications avant d'activer la propriété). Notez que le fichier contenait un signet pour le mot-clé `Rev` et que nous n'avons pas inclus ce mot-clé dans la valeur de la propriété. Subversion ignore simplement les requêtes de substitutions de mots-clés qui ne sont pas présents dans le fichier et ne substitue pas de mot-clé qui ne soit pas présent dans la valeur de la propriété `svn:keywords`.

Immédiatement après avoir propagé ces modifications de propriété, Subversion met à jour votre copie de travail avec le nouveau texte substitué. Au lieu de voir votre signet `$LastChangedDate$`, vous voyez le résultat de la substitution. Ce résultat contient aussi le nom du mot-clé et est toujours entouré par des caractères dollar (`$`). Comme prévu, le mot-clé `Rev` n'a pas été substitué parce que nous n'avons pas demandé qu'il le soit.

Notez également que la substitution s'est bien passée alors que nous avons indiqué `Date Author` comme valeur de propriété `svn:keywords` et que le signet utilisait l'alias `$LastChangedDate$` :

```
Voici les dernières prévisions de nos spécialistes :
$LastChangedDate: 2006-07-22 21:42:37 -0700 (sam. 22 juil. 2006) $
$Rev$
Les cumulus sont de plus en plus nombreux au fur et à mesure que l'été approche.
```

Si quelqu'un d'autre propage une modification de `meteo.txt`, votre copie de ce fichier continue à afficher la même valeur

[6] … ou même peut-être un paragraphe de ce livre …

substituée de mot-clé, jusqu'à ce que vous mettiez à jour votre copie de travail. Aux mots-clés de `meteo.txt` seront alors à nouveau substituées les informations qui se rapportent à la plus récente propagation du fichier.

Où est passé $GlobalRev$?

Les nouveaux utilisateurs sont parfois surpris par le fonctionnement du mot-clé `$Rev$`. Puisque le dépôt a un numéro de révision à la fois unique, global et croissant, beaucoup de gens pensent que c'est par ce numéro qu'est remplacé le mot-clé `$Rev$`. Mais `$Rev$` indique la dernière révision dans laquelle le fichier a *changé*, pas la révision de la dernière mise à jour. Le malentendu est ainsi dissipé, mais peut-être pas la frustration de ne pas avoir automatiquement le numéro global de la dernière révision dans vos fichiers, n'est-ce pas ?

Pour l'obtenir, vous avez besoin d'un outil externe. Subversion est livré avec un outil appelé **svnversion**, qui a été conçu spécifiquement pour cela. svnversion parcourt votre copie de travail et affiche toutes les révisions qu'il trouve. Vous pouvez utiliser ce programme, avec d'autres outils de traitement, pour insérer l'information de révision désirée dans vos fichiers. Pour davantage d'information sur **svnversion**, voir la section intitulée « svnversion ».

Subversion 1.2 introduisit une nouvelle variante pour la syntaxe des mots-clés. Cette syntaxe offre des fonctionnalités supplémentaires et utiles, bien que parfois atypiques. Vous pouvez désormais demander à Subversion de maintenir une longueur constante (en nombre d'octets consommés) pour les mots-clés substitués, longueur que vous définissez en utilisant la séquence double deux-points (`::`) après le nom du mot-clé, suivie du nombre de caractères espace ( ) voulus. Quand Subversion doit effectuer la substitution du mot-clé par le mot-clé et sa valeur, il ne remplace que ces espaces, laissant la taille du champ inchangée. Si la valeur est plus courte que la largeur du champ, il reste des espaces pour combler la fin ; si la valeur est trop longue, elle est tronquée avec le caractère dièse (#) juste avant le caractère dollar ($) final.

Par exemple, pour un document dans lequel vous avez une section avec les mots-clés Subversion dans un tableau, l'utilisation de la syntaxe originale de substitution de Subversion donne quelque chose comme :

```
$Rev$:     Numéro de révision de la dernière propagation
$Author$:  Auteur de la dernière propagation
$Date$:    Date de la dernière propagation
```

C'est joli et bien aligné au début. Mais quand vous propagez ce fichier (avec la substitution des mots-clés activée, bien évidemment), vous obtenez :

```
$Rev: 12 $:     Numéro de révision de la dernière propagation
$Author: harry $:  Auteur de la dernière propagation
$Date: 2006-03-15 02:33:03 -0500 (mer. 15 mars 2006) $:    Date de la dernière
propagation
```

Le résultat n'est pas très heureux. Vous êtes alors tenté de modifier le fichier après la substitution pour que le contenu soit mieux aligné. Mais cette modification ne serait valable que tant que les valeurs des mots-clés gardent la même taille. Si le numéro de dernière révision change de valeur (par exemple de 99 à 100), ou si une autre personne avec un nom d'utilisateur plus long modifie le fichier, tout le travail d'alignement est à refaire. Cependant, si vous utilisez la version 1.2 (ou plus) de Subversion, vous pouvez utiliser la nouvelle syntaxe et définir des largeurs de champs adéquates et constantes. Votre fichier ressemble alors à ceci :

```
$Rev::                $:  Numéro de révision de la dernière propagation
$Author::             $:  Auteur de la dernière propagation
$Date::               $:  Date de la dernière propagation
```

Propagez ce fichier. Cette fois, Subversion prend en compte la syntaxe d'un champ de mot-clé à largeur fixe et maintient la largeur de ce champ comme indiqué entre le double deux-points et le signe dollar final. Après substitution, la largeur des champs n'a pas changé : les valeurs courtes comme `Rev` et `Author` sont comblées avec des espaces et le champ `Date`, trop long, est tronqué par un caractère dièse :

```
$Rev:: 13             $:  Numéro de révision de la dernière propagation
$Author:: harry       $:  Auteur de la dernière propagation
```

```
$Date:: 2006-03-15 0#$:  Date de la dernière propagation
```

L'utilisation des mots-clés à longueur fixe est particulièrement efficace lors de substitutions dans des fichiers aux formats complexes, qui utilisent eux-mêmes des champs de données de longueur fixe ou qui stockent les données dans des champs dont la taille est particulièrement difficile à changer en dehors de l'application native elle-même (les documents Microsoft Office en sont un bon exemple).

Soyez conscient que, comme la taille d'un mot-clé est mesurée en octets, les valeurs utilisant des données codées sur plusieurs octets peuvent être corrompues. Par exemple, un nom d'utilisateur qui contient des caractères au format UTF-8 codés sur plusieurs octets risque d'être tronqué en plein milieu d'un de ces caractères multi-octets. Cette troncature est valide au niveau du traitement des octets mais résulte en une chaîne UTF-8 incorrecte en raison du caractère final tronqué. Il est ainsi possible que certaines applications, au moment de charger le fichier, remarquent que le texte UTF-8 est invalide, considèrent tout le fichier comme corrompu et refusent de travailler dessus. En conséquence, lorsque vous utilisez les mots-clés à longueur fixe, veillez à choisir une taille adaptée à des valeurs pouvant contenir des caractères éventuellement codés sur plusieurs octets.

Répertoires clairsemés

Par défaut, la plupart des opérations Subversion sur des répertoires agissent de manière récursive. Par exemple, **svn checkout** crée une copie de travail avec tous les fichiers et répertoires de la zone spécifiée du dépôt, en descendant récursivement dans l'arborescence du dépôt pour en copier la structure complète sur votre disque local. Subversion 1.5 introduit une nouvelle fonctionnalité appelée *répertoires clairsemés* (ou *extractions superficielles*) qui permet d'obtenir facilement une copie de travail (ou une simple portion d'une copie de travail) moins profonde que via la récursion complète, avec la possibilité de n'extraire que plus tard les répertoires et les fichiers ignorés auparavant.

Par exemple, imaginons un dépôt dont l'arborescence des fichiers et répertoires est constituée des noms des membres d'une famille et de leurs animaux de compagnie (c'est assurément un exemple bizarre, mais soit). Un **svn checkout** standard nous donne une copie de travail de l'ensemble de l'arborescence :

```
$ svn checkout file:///var/svn/depot maman
A    maman/fils
A    maman/fils/petit-fils
A    maman/fille
A    maman/fille/petite-fille1
A    maman/fille/petite-fille1/lapinou1.txt
A    maman/fille/petite-fille1/lapinou2.txt
A    maman/fille/petite-fille2
A    maman/fille/poissonou.txt
A    maman/minou1.txt
A    maman/toutou1.txt
Révision 1 extraite.
$
```

Maintenant, extrayons la même arborescence, mais cette fois en demandant à Subversion de nous donner uniquement le répertoire racine sans les enfants :

```
$ svn checkout file:///var/svn/depot maman-vide --depth empty
Révision 1 extraite.
$
```

Remarquez que nous avons ajouté l'option `--depth` à la commande **svn checkout** originale. Cette option existe pour de nombreuses sous-commandes Subversion et est similaire aux options `--non-recursive` (`-N`) et `--recursive` (`-R`). En fait, elle combine, améliore, remplace et, à terme, rend obsolète ces deux options plus anciennes. Déjà, elle permet à l'utilisateur de spécifier le niveau de récursion de façon plus précise, en ajoutant des niveaux auparavant non supportés (ou supportés de manière peu satisfaisante). Voici les valeurs de niveau de récursion que vous pouvez ajouter à vos requêtes Subversion :

`--depth empty`
Inclut uniquement la cible immédiate de l'opération, sans aucun fichier ou répertoire fils.

`--depth files`
Inclut la cible immédiate de l'opération et tous les fichiers fils immédiats.

`--depth immediates`
Inclut la cible immédiate de l'opération et tous ses sous-répertoires et fils immédiats. Les répertoires fils seront eux-mêmes vides.

`--depth infinity`
Inclut la cible immédiate, les fichiers et répertoires fils, les fils des fils et ainsi de suite de façon à réaliser une récursion complète.

Bien sûr, la simple combinaison de deux options existantes en une seule ne constitue pas une nouvelle fonctionnalité méritant une section complète de ce livre. Heureusement, il y a plus à en dire. Ce concept de profondeur ne s'applique pas uniquement aux opérations réalisées avec le client Subversion mais il s'étend aussi à la description de la copie de travail elle-même, en tant que *niveau associé* de manière permanente par la copie de travail à chaque élément. La force de ce concept est cette permanence. La copie de travail se rappelle le niveau de récursion que vous avez choisi pour chaque élément qui la compose, jusqu'à ce que vous en changiez. Par défaut, les commandes Subversion agissent sur les éléments présents dans la copie de travail, indépendamment de leur niveau de récursion propre.

Vous pouvez vérifier le niveau de récursion d'une copie de travail en utilisant la commande **svn info**. Si le niveau de récursion n'est pas infini, **svn info** affiche une ligne indiquant le niveau de récursion :

```
$ svn info maman-immediats | grep '^Profondeur :'
Profondeur : immédiates
$
```

Ces premiers exemples comportaient des extractions avec un niveau infini de récursion (la valeur par défaut de **svn checkout**) ou avec un niveau nul. Voyons maintenant des exemples avec d'autres valeurs de niveau de récursion :

```
$ svn checkout file:///var/svn/depot maman-fichiers --depth files
A    maman-fichiers/minou1.txt
A    maman-fichiers/toutou1.txt
Révision 1 extraite.
$ svn checkout file:///var/svn/depot maman-immediats --depth immediates
A    maman-immediats/fils
A    maman-immediats/fille
A    maman-immediats/minou1.txt
A    maman-immediats/toutou1.txt
Révision 1 extraite.
$
```

Comme indiqué, chacun de ces deux niveaux se situe quelque part entre la cible toute simple et la récursion complète.

Nous avons utilisé la commande **svn checkout** pour nos exemples, mais l'option `--depth` est également accessible depuis beaucoup d'autres commandes Subversion. Pour ces autres commandes, spécifier un niveau de récursion est une manière de limiter le rayon d'action d'une opération à un niveau, à l'instar des vieilles options `--non-recursive` (`-N`) et `--recursive` (`-R`). Cela veut dire que lorsque vous travaillez sur une copie de travail d'un certain niveau et que vous faites une opération sur un niveau plus faible, l'opération est limitée à ce niveau faible. En fait, on peut généraliser ce raisonnement : pour une copie de travail d'un niveau de récursion arbitraire (éventuellement hétérogène) et pour une commande Subversion comportant un niveau de récursion, la commande conserve le niveau de récursion associé aux éléments de la copie de travail tout en limitant le rayon d'action de l'opération au niveau demandé (ou celui par défaut).

En plus de l'option `--depth`, les sous-commandes **svn update** et **svn switch** acceptent une deuxième option relative au niveau de récursion : `--set-depth`. C'est cette option qui vous permet de changer le niveau de récursion associé à un

élément d'une copie de travail. Regardez ce qui se passe après avoir extrait notre niveau zéro puis graduellement augmenté le niveau de récursion en utilisant la commande **svn update --set-depth *NOUVELLE-PROFONDEUR CIBLE***:

```
$ svn update --set-depth files maman-vide
A    maman-vide/minou1.txt
A    maman-vide/toutou1.txt
Actualisé à la révision 1.
$ svn update --set-depth immediates maman-vide
A    maman-vide/fils
A    maman-vide/fille
Actualisé à la révision 1.
$ svn update --set-depth infinity maman-vide
A    maman-vide/fils/petit-fils
A    maman-vide/fille/petite-fille1
A    maman-vide/fille/petite-fille1/lapinou1.txt
A    maman-vide/fille/petite-fille1/lapinou2.txt
A    maman-vide/fille/petite-fille2
A    maman-vide/fille/poissonou1.txt
Actualisé à la révision 1.
$
```

Au fur et à mesure que nous avons augmenté le niveau de récursion, le dépôt a complété progressivement notre arborescence.

Dans notre exemple, nous n'avons agi que sur la racine de notre copie de travail, en changeant la valeur du niveau de récursion associé. Mais nous pouvons aussi changer de façon indépendante le niveau de récursion associé à chaque sous-répertoire de la copie de travail. Une utilisation minutieuse de cette option nous permet de récupérer uniquement certaines portions de la copie de travail, en laissant de côté toutes les autres portions (d'où le nom « clairsemé » de la fonctionnalité). Voici un exemple montrant comment construire une portion d'une branche de notre arbre généalogique, activer la récursion totale sur une autre branche et élaguer le reste (qui ne sera donc pas sur notre disque dur).

```
$ rm -rf maman-vide
$ svn checkout file:///var/svn/depot maman-vide --depth empty
Révision 1 extraite.
$ svn update --set-depth empty maman-vide/fils
A    maman-vide/fils
Actualisé à la révision 1.
$ svn update --set-depth empty maman-vide/fille
A    maman-vide/fille
Actualisé à la révision 1.
$ svn update --set-depth infinity maman-vide/fille/petite-fille1
A    maman-vide/fille/petite-fille1
A    maman-vide/fille/petite-fille1/lapinou1.txt
A    maman-vide/fille/petite-fille1/lapinou2.txt
Actualisé à la révision 1.
$
```

Heureusement, même avec différents niveaux de récursion définis au sein d'une même copie de travail, les actions sur la copie de travail ne s'en trouvent pas plus compliquées. Vous pouvez toujours effectuer des modifications et les propager, revenir en arrière ou afficher les modifications locales de votre copie de travail sans spécifier d'option particulière (y compris `--depth` et `--set-depth`) aux dites commandes. Même **svn update** fonctionne normalement quand on ne lui fournit pas de niveau de récursion spécifique : elle met à jour les cibles de la copie de travail qui sont présentes en tenant compte des niveaux de récursion qui leur sont associés.

Vous devez vous demander : « Bien. Mais quand aurais-je besoin d'utiliser ça ? » Un cas classique est lié à une architecture du dépôt particulière : lorsque de nombreux projets et modules logiciels liés cohabitent au même niveau dans un dépôt (`trunk/projet1`, `trunk/projet2`, `trunk/projet3`, etc.). Dans de tels scénarios, il est probable que seuls quelques projets vous intéressent personnellement, sans doute pas plus d'un projet principal et de quelques autres modules dont il dépend. Vous pouvez extraire une copie de travail pour chacune de ces arborescences, mais ces copies de travail sont séparées et, par conséquent, il peut être fastidieux d'effectuer des opérations sur plusieurs ou sur l'ensemble des copies de travail en même temps. L'autre solution est d'utiliser la fonctionnalité de répertoires clairsemés, en construisant une seule copie de travail qui ne contient que les modules qui vous intéressent. Vous partez d'une extraction du répertoire parent commun aux différents projets avec un niveau zéro de récursion (empty-depth), puis vous mettez à jour avec un niveau infini de récursion les éléments que vous voulez récupérer, comme nous l'avons fait dans l'exemple précédent. Voyez ça comme un système d'inclusion optionnelle

des éléments qui peuplent la copie de travail.

L'implémentation des extractions superficielles de Subversion 1.5 est relativement bonne mais il y a deux choses intéressantes qu'elle ne permet pas de faire. Premièrement, vous ne pouvez pas diminuer le niveau de récursion d'un élément de la copie de travail. Si vous lancez **`svn update --set-depth empty`** sur une copie de travail de niveau de récursion infini, vous n'aurez pas l'effet attendu, qui serait de supprimer tout sauf le répertoire racine — cela renverra simplement une erreur. Deuxièmement, il n'y a pas de valeur de récursion pour indiquer que vous voulez exclure explicitement un élément. Vous devez effectuer une exclusion implicite de l'élément en incluant tous les éléments sauf celui-là.

Verrouillage

Le modèle copier-modifier-fusionner de gestion de versions de Subversion repose sur ses algorithmes de fusion, notamment sur la manière dont ils gèrent les conflits quand de multiples collaborateurs modifient le même fichier simultanément. Subversion lui-même ne propose qu'un seul algorithme de ce type, un algorithme qui détecte les modifications par trois méthodes et qui est suffisamment intelligent pour gérer les données à la ligne près. Subversion vous permet également d'utiliser en plus des outils externes lors du processus de fusion (comme indiqué dans la section intitulée « Programme externe de comparaison de trois fichiers »), parfois encore meilleurs que ceux inclus dans Subversion, proposant par exemple une granularité plus fine allant jusqu'au mot, voire au caractère, au lieu de s'arrêter à la ligne. Mais, en règle générale, ces algorithmes ne fonctionnent que sur des fichiers texte. Le paysage est beaucoup plus sombre lorsque l'on recherche des outils de fusion pour des formats de fichiers non-texte. Et quand vous ne trouvez pas d'outil capable de fusionner de tels fichiers, les limites du modèle copier-modifier-fusionner se font vite sentir.

Prenons un exemple de la vie réelle où ce type de problème apparaît. Harry et Sally sont deux graphistes travaillant sur le même projet (du marketing pour le patron d'un garage). Au cœur d'une affiche de ce projet se trouve l'image d'une voiture dont la carrosserie a besoin d'être réparée, stockée dans un fichier image au format PNG. L'agencement de l'affiche est pratiquement terminé, et Harry et Sally sont contents de la photo qu'ils ont choisie pour leur voiture endommagée : une Ford Mustang bleue de 1967, avec un gnon sur l'aile avant gauche.

C'est alors, comme c'est souvent le cas dans le domaine du graphisme, que des contraintes extérieures imposent de changer la couleur de la voiture. Sally met donc à jour sa copie de travail à la révision `HEAD`, lance son outil d'édition de photos et commence à modifier la photo de manière à obtenir une voiture rouge cerise. Pendant ce temps, Harry, particulièrement inspiré ce jour-là, décide que l'image serait plus percutante si la voiture était davantage endommagée. Lui aussi met à jour sa copie de travail à la révision `HEAD`, puis dessine des fissures sur le pare-brise. Il termine son travail avant que Sally ne termine le sien, admire son chef-d'œuvre et propage les changements. Peu après, Sally en termine avec la nouvelle couleur de la voiture et essaie de propager ses modifications. Mais, comme prévu, Subversion ne parvient pas à valider la propagation et informe Sally que sa version de l'image n'est pas à jour.

Voilà où résident les difficultés : si Harry et Sally avaient effectué leurs changements sur un fichier texte, Sally aurait simplement mis à jour sa copie de travail, recevant au passage les modifications de Harry. Dans le pire des cas, ils auraient modifié la même portion du fichier et Sally aurait eu à résoudre les conflits manuellement. Mais, ici, nous avons affaire à des images binaires, pas des fichiers texte. Et s'il est relativement facile de décrire ce que devrait être l'image finale, il y a très peu de chances qu'un logiciel soit suffisamment intelligent pour détecter les parties communes de l'image sur laquelle les artistes ont travaillé, les changements effectués par Harry et les changements effectués par Sally et pour en tirer une image d'une Mustang rouge avec un pare-brise fissuré !

Clairement, les choses se seraient mieux passées si Harry et Sally avaient sérialisé leurs modifications : par exemple, si Harry avait attendu et dessiné ses fissures sur la voiture nouvellement rouge de Sally, ou si Sally avait changé la couleur d'une voiture avec un pare-brise déjà fissuré. Comme indiqué dans la section intitulée « Modèle copier-modifier-fusionner », la plupart de ces problèmes disparaissent complètement quand une communication parfaite existe entre Harry et Sally [7]. Mais comme un système de gestion de versions est en fait un mode de communication, il s'ensuit que si ce type de logiciel facilite la sérialisation de tâches d'édition non parallélisables, c'est plutôt une bonne chose. C'est ici que l'implémentation du concept verrouiller-modifier-libérer dans Subversion prend tout son sens. Il est temps de parler de la fonctionnalité de *verrouillage* de Subversion, qui est similaire aux mécanismes permettant de « réserver pour modifications » des fichiers dans d'autres systèmes de gestion de versions.

En fin de compte, la fonctionnalité de verrouillage existe afin de minimiser les pertes de temps et les efforts. En autorisant un utilisateur à s'arroger logiciellement le droit exclusif de modifier un fichier dans le dépôt, cet utilisateur peut être suffisamment confiant dans le fait que son travail ne sera pas vain — la propagation de ses changements réussira. Aussi, en signifiant aux autres utilisateurs qu'une sérialisation a lieu pour un objet suivi en versions, ces utilisateurs peuvent raisonnablement s'attendre à ce que cet objet soit modifié par quelqu'un d'autre. Eux aussi peuvent alors éviter de perdre leur temps et leur énergie sur des modifications qui ne pourront pas être fusionnées en raison d'un problème de mise à jour du fichier correspondant.

[7] À ce propos, un peu de communication n'aurait pas non plus été un mauvais remède pour leurs homonymes hollywoodiens.

La fonctionnalité de verrouillage de Subversion comporte en fait plusieurs facettes, qui permettent entre autres de verrouiller un fichier suivi en versions [8] (demander le droit exclusif de modification sur le fichier), de le déverrouiller (abandonner le droit exclusif de modification), de voir la liste des fichiers qui sont verrouillés et par qui, d'annoter des fichiers pour lesquels le verrouillage est fortement recommandé avant édition, etc. Dans cette section, nous abordons toutes les facettes de cette fonctionnalité de verrouillage.

Les trois types de « verrous »

Dans cette section, comme pratiquement partout dans ce livre, les mots « verrou » et « verrouillage » décrivent un mécanisme d'exclusion mutuelle entre utilisateurs pour éviter des propagations incompatibles. Malheureusement, il existe deux autres sortes de « verrous » auxquels Subversion et donc ce livre sont confrontés.

Les premiers sont des *verrous des copies de travail*, utilisés en interne par Subversion pour éviter des collisions entre de multiples instances du client Subversion travaillant sur la même copie de travail. Ce type de verrou est repérable au `L` situé dans la troisième colonne de la sortie de **svn status** et peut être supprimé par la commande **svn cleanup**, comme indiqué à la section intitulée « Parfois, il suffit de faire le ménage ».

Ensuite, il y a les *verrous des bases de données*, utilisés en interne par les bases de données Berkeley DB pour éviter les collisions entre de multiples programmes accédant à la base de données. C'est le type de verrou qui, s'il est encore présent à notre insu après une erreur, peut provoquer un « plantage » du dépôt, comme indiqué dans la section intitulée « Rétablissement de bases de données Berkeley DB ».

En général, vous pouvez faire abstraction de ces autres types de verrous, du moins tant que tout va bien. Si les choses se gâtent, vous aurez peut-être à vous y intéresser. Dans ce livre, le terme « verrou » désigne la fonctionnalité Subversion, sauf si le contexte indique le contraire ou si c'est mentionné explicitement.

Création d'un verrou

Dans un dépôt Subversion, un *verrou* est une méta-donnée qui alloue à un utilisateur un accès exclusif en écriture sur un fichier. Cet utilisateur est appelé *détenteur du verrou*. Chaque verrou possède également un identifiant unique, en général une longue chaîne de caractères, appelé *jeton de verrouillage*. Le dépôt gère les verrous en assurant in fine leur création, leur application et leur suppression. Si une propagation tente de modifier ou effacer un fichier verrouillé (ou effacer un répertoire parent dudit fichier), le dépôt demande deux informations : que le client effectuant la propagation s'authentifie en tant que détenteur du verrou et que le jeton de verrouillage soit fourni lors de la procédure de propagation afin de montrer que le client sait bien quel verrou il utilise.

Pour illustrer la création d'un verrou, reprenons notre exemple de graphistes travaillant sur les même fichiers image binaires. Harry a décidé de changer cette image JPEG. Pour interdire aux autres collaborateurs d'effectuer des changements sur le fichier pendant qu'il le modifie (et pour les avertir qu'il va modifier ce fichier), il verrouille le fichier dans le dépôt en utilisant la commande **svn lock** :

```
$ svn lock banane.jpg -m "Édition du fichier pour la livraison de demain."
'banane.jpg' verrouillé par l'utilisateur 'harry'.
$
```

Il y a plusieurs points intéressants dans l'exemple ci-dessus : d'abord, notez que Harry utilise l'option `--message` (`-m`) de **svn lock**. Comme **svn commit**, la commande **svn lock** accepte des commentaires (soit via `--message` (`-m`), soit via `--file` (`-F`)) pour indiquer la raison du verrouillage du fichier. En revanche, contrairement à **svn commit**, **svn lock** n'exige pas automatiquement un message en lançant votre éditeur de texte préféré. Les commentaires de verrouillage sont optionnels, mais néanmoins recommandés pour faciliter la communication entre collaborateurs.

Ensuite, la tentative de verrouillage a réussi. Cela signifie que le fichier n'était pas préalablement verrouillé et que Harry disposait de la dernière version du fichier. Si la copie de travail de Harry avait été obsolète, le dépôt aurait refusé la demande, forçant Harry à effectuer une mise à jour (**svn update**) et à relancer ensuite la commande de verrouillage. La commande de verrouillage aurait également échoué si le fichier avait déjà été verrouillé par quelqu'un d'autre.

Comme vous pouvez le constater, la commande **svn lock** affiche la confirmation que le verrouillage a réussi. Dès lors, le

[8] Pour l'instant, Subversion ne permet pas de poser de verrou sur un répertoire.

verrouillage du fichier apparaît dans le résultat des commandes **svn status** et **svn info** :

```
$ svn status
     K banane.jpg

$ svn info banane.jpg
Chemin : banane.jpg
Nom : banane.jpg
URL : http://svn.exemple.com/depot/projet/banane.jpg
Racine du dépôt : http://svn.exemple.com/depot
UUID du dépôt : edb2f264-5ef2-0310-a47a-87b0ce17a8ec
Revision: 2198
Type de nœud : file
Tâche programmée : normale
Auteur de la dernière modification : frank
Révision de la dernière modification : 1950
Date de la dernière modification : 2006-03-15 12:43:04 -0600 (mer. 15 mars 2006)
Texte mis à jour : 2006-06-08 19:23:07 -0500 (jeu. 08 juin 2006)
Propriétés mis à jour : 2006-06-08 19:23:07 -0500 (jeu. 08 juin 2006)
Somme de contrôle: 3b110d3b10638f5d1f4fe0f436a5a2a5
Nom de verrou : opaquelocktoken:0c0f600b-88f9-0310-9e48-355b44d4a58e
Propriétaire du verrou : harry
Verrou créé : 2006-06-14 17:20:31 -0500 (mer. 14 juin 2006)
Commentaire du verrou (1 ligne):
Édition du fichier pour la livraison de demain.

$
```

Le fait que la commande **svn info**, qui ne contacte pas le dépôt quand elle porte sur un chemin d'une copie de travail, affiche bien le jeton de verrouillage, révèle une caractéristique importante des jetons de verrouillage : ils sont intégrés dans la copie de travail. La présence du jeton de verrouillage est primordiale. Il authorise la copie de travail à utiliser le verrou ultérieurement. Par ailleurs, la commande **svn status** affiche un `K` (raccourci pour « locKed » — « verrouillé » en anglais) avant le nom du fichier, indiquant que le jeton de verrouillage est présent.

À propos des jetons de verrouillage

Un jeton de verrouillage n'est pas un jeton d'authentification mais plutôt un jeton d'*autorisation*. Le jeton n'est pas protégé comme un secret. En fait, le jeton de verrouillage peut être vu par n'importe quel utilisateur qui lance la commande **`svn info` *`URL`***. Un jeton de verrouillage n'a de propriété spéciale que quand il est placé dans une copie de travail. C'est la preuve que le verrou a été créé dans cette copie de travail et non ailleurs par quelqu'un d'autre. Seulement s'authentifier en tant que propriétaire du verrou n'est pas suffisant pour éviter les accidents.

Par exemple, supposons que vous verrouilliez un fichier avec votre ordinateur au bureau, puis que vous quittiez le travail avant d'avoir fini vos changements sur ce fichier. Il ne doit pas être possible de propager accidentellement des modifications de ce même fichier depuis votre ordinateur à la maison plus tard dans la soirée simplement parce que vous vous êtes authentifié en tant que détenteur du verrou. En d'autres termes, le verrou interdit à une instance d'un quelconque client Subversion de saboter le travail d'une autre instance (dans notre exemple, si vous avez réellement besoin de modifier le fichier depuis votre copie de travail à la maison, vous devrez *casser* le verrou puis re-verrouiller le fichier).

Maintenant que Harry a verrouillé `banane.jpg`, Sally ne peut ni modifier ni effacer ce fichier :

```
$ svn delete banane.jpg
D         banane.jpg
$ svn commit -m "Suppression des fichiers inutiles."
Suppression    banane.jpg
svn: Échec de la propagation (commit), (détails):
svn: Suppression de
'/depot/projet/!svn/wrk/64bad3a9-96f9-0310-818a-df4224ddc35d/banane.jpg':
423 Verrouillé (http://svn.exemple.com)
$
```

Mais Harry, après avoir fait ses retouches sur sa belle banane jaune, peut propager ses changements sur le fichier. C'est parce qu'il s'authentifie en tant que détenteur du verrou et aussi parce que sa copie de travail possède le bon jeton de verrouillage :

```
$ svn status
M    K banane.jpg
$ svn commit -m "Rendu la banane plus jaune."
Envoi       banane.jpg
Transmission des données .
Révision 2201 propagée.
$ svn status
$
```

Notez qu'après que la propagation est terminée, **svn status** permet de voir que le jeton de verrouillage n'est plus présent dans la copie de travail. C'est le comportement normal de **svn commit** : elle recherche dans la copie de travail (ou dans une liste de cibles, si vous fournissez une telle liste) les modifications effectuées localement et elle envoie les jetons de verrouillage qu'elle trouve durant sa recherche au serveur, en tant que partie intégrante du processus de propagation. Après que la propagation a réussi, tous les verrous du dépôt qui ont été mentionnés sont libérés, *même ceux pour lesquels les fichiers n'ont pas été propagés*. Ce comportement a pour but de dissuader les utilisateurs d'être négligents avec leurs verrous ou de garder des verrous trop longtemps. Si Harry verrouille au hasard trente fichiers dans un répertoire nommé `Images` parce qu'il n'est pas sûr de savoir quels fichiers il doit modifier et qu'il ne modifie finalement que quatre fichiers, alors quand il lance la commande **`svn commit Images`**, la procédure libère les trente verrous.

Ce mode de fonctionnement (libérer automatiquement les verrous) peut être modifié avec l'option `--no-unlock` de **svn commit**. C'est utile quand vous voulez propager des changements mais que vous prévoyez d'effectuer des changements supplémentaires et que donc vous avez toujours besoin des verrous. Vous pouvez également en faire le fonctionnement par défaut en réglant l'option `no-unlock` dans la zone de configuration (voir la section intitulée « Zone de configuration des exécutables »).

Bien sûr, verrouiller un fichier n'oblige pas l'utilisateur à le modifier. Le verrou peut être libéré n'importe quand avec la commande **svn unlock** :

```
$ svn unlock banane.c
'banane.c' déverrouillé.
```

Identification d'un verrou

Quand une propagation échoue parce que quelqu'un d'autre a posé un verrou, il est facile de savoir pourquoi. La commande la plus simple est **`svn status --show-updates`**:

```
$ svn status -u
M              23   truc.c
M    O         32   raisin.jpg
      *        72   machin.h
État par rapport à la révision      105
$
```

Dans cet exemple, Sally peut voir que non seulement sa copie de travail de `machin.h` n'est plus à jour, mais aussi qu'un des deux fichiers qu'elle prévoie de propager est verrouillé dans le dépôt. La lettre `O` (« Others » — « autres » en anglais) indique qu'un verrou existe sur ce fichier et qu'il a été créé par quelqu'un d'autre. Si elle essayait de lancer **svn commit**, le verrou sur `raisin.jpg` l'en empêcherait. Sally est laissée dans l'expectative de savoir qui a posé le verrou, quand et pourquoi. Là encore, **svn info** trouve la réponse :

```
$ svn info http://svn.exemple.com/depot/projet/raisin.jpg
Chemin : raisin.jpg
Nom : raisin.jpg
URL: http://svn.exemple.com/depot/projet/raisin.jpg
```

```
UUID du dépôt : edb2f264-5ef2-0310-a47a-87b0ce17a8ec
Révision: 105
Type de nœud : file
Auteur de la dernière modification : sally
Révision de la dernière modification : 32
Texte mis à jour : 2006-01-25 12:43:04 -0600 (Sun, 25 Jan 2006)
Nom de verrou : opaquelocktoken:fc2b4dee-98f9-0310-abf3-653ff3226e6b
Propriétaire du verrou : harry
Verrou créé : 2006-02-16 13:29:18 -0500 (jeu. 16 févr. 2006)
Commentaire du verrou (1 ligne):
Besoin de faire une retouche rapide sur cette image.
$
```

De la même manière que **svn info** peut être utilisée pour examiner les objets de la copie de travail, elle peut être utilisée pour examiner les objets du dépôt. Si l'argument principal de **svn info** est un chemin de la copie de travail, alors toutes les informations stockées localement sont affichées ; toute mention d'un verrou signifie que la copie de travail détient un jeton de verrouillage (si le fichier est verrouillé par un autre utilisateur ou depuis une autre copie de travail, alors lancer **svn info** sur la copie de travail ne renvoit aucune information relative au verrou). Si l'argument principal de **svn info** est une URL, alors les informations affichées se rapportent à la dernière version de l'objet dans le dépôt et toute mention d'un verrou concerne le verrou en cours sur l'objet.

Ainsi, dans notre exemple, Sally peut voir que Harry a verrouillé le fichier le 16 février pour effectuer une « retouche rapide ». Comme nous sommes en juin, elle suspecte qu'il a probablement oublié le verrou. Elle pourrait téléphoner à Harry pour le lui signaler et lui demander de libérer le verrou. S'il n'est pas joignable, elle peut toujours essayer de forcer le verrou elle-même, ou demander à un administrateur de le faire.

Cassage et vol d'un verrou

Un verrou n'est pas quelque chose de sacré : dans la configuration par défaut de Subversion, les verrous peuvent être libérés non seulement par leur détenteur, mais aussi par n'importe qui d'autre. Quand quelqu'un d'autre que le détenteur d'un verrou le libère, nous appelons ça *casser le verrou*.

Avec un statut d'administrateur, il est facile de casser un verrou. Les programmes **svnlook** et **svnadmin** peuvent afficher et casser les verrous directement dans le dépôt (pour plus d'informations sur ces outils, reportez-vous à la section intitulée « Boîte à outils de l'administrateur »).

```
$ svnadmin lslocks /usr/local/svn/depot
Chemin : /projet2/images/banane.jpg
Chaîne UUID : opaquelocktoken:c32b4d88-e8fb-2310-abb3-153ff1236923
Propriétaire : frank
Créé : 2006-06-15 13:29:18 -0500 (jeu. 15 juin 2006)
Expire :
Commentaire (1 ligne):
J'améliore encore la couleur jaune.

Chemin : /projet/raisin.jpg
Chaîne UUID : opaquelocktoken:fc2b4dee-98f9-0310-abf3-653ff3226e6b
Propriétaire : harry
Créé : 2006-02-16 13:29:18 -0500 (jeu. 16 fév. 2006)
Expire :
Commentaire (1 ligne):
Besoin de faire une retouche rapide sur cette image.

$ svnadmin rmlocks /usr/local/svn/depot/projet/raisin.jpg
'/projet/raisin.jpg' déverrouillé
$
```

L'option la plus intéressante est celle qui permet aux utilisateurs de casser les verrous détenus par d'autres personnes à travers le réseau. Pour ce faire, Sally doit simplement ajouter l'option `--force` à la commande **svn unlock** :

```
$ svn status -u
M             23   truc.c
```

```
M    O         32   raisin.jpg
      *        72   machin.h
État par rapport à la révision     105
$ svn unlock raisin.jpg
svn: 'raisin.jpg' n'est pas verrouillé dans cette copie de travail
$ svn info raisin.jpg | grep URL
URL: http://svn.exemple.com/depot/projet/raisin.jpg
$ svn unlock http://svn.exemple.com/depot/projet/raisin.jpg
svn: Unlock request failed: 403 Forbidden (http://svn.exemple.com)
$ svn unlock --force http://svn.exemple.com/depot/projet/raisin.jpg
'raisin.jpg' déverrouillé.
$
```

Ainsi, la tentative initiale de Sally pour libérer le verrou a échoué parce qu'elle a lancé **svn unlock** directement sur le fichier de sa copie de travail, où aucun jeton de verrouillage n'était présent. Pour casser le verrou directement dans le dépôt, elle doit passer une URL à **svn unlock**. Son premier essai pour casser le verrou avec l'URL échoue car elle ne peut pas s'authentifier comme détentrice du verrou (et elle n'a pas non plus le jeton de verrouillage). Mais quand elle passe l'option `--force`, les pré-requis d'authentification et d'autorisation sont ignorés et le verrou est cassé.

Casser le verrou peut ne pas être suffisant. Dans l'exemple, Sally ne veut pas seulement casser le verrou oublié par Harry, mais également re-verrouiller le fichier pour son propre usage. Elle peut le faire en lançant **svn unlock** avec l'option `--force` puis **svn lock** à la suite, mais il existe une petite chance que quelqu'un d'autre verrouille le fichier entre les deux commandes. La meilleure solution est donc de *voler le verrou*, ce qui implique de casser et re-verrouiller le fichier en une seule opération atomique. Pour ce faire, Sally passe l'option `--force` à la commande **svn lock** :

```
$ svn lock raisin.jpg
svn: avertissement : Échec de la demande de verrou : 423 verrouillé
(http://svn.exemple.com)
$ svn lock --force raisin.jpg
'raisin.jpg' verrouillé par l'utilisateur 'sally'.
$
```

Dans tous les cas, que le verrou soit cassé ou volé, Harry est bon pour une bonne surprise. La copie de travail de Harry contient toujours le jeton de verrouillage original, mais le verrou n'existe plus. Le jeton de verrouillage est dit *défunt*. Le verrou associé au jeton de verrouillage a été soit cassé (il n'existe plus dans le dépôt) soit volé (remplacé par un autre verrou). Quoi qu'il en soit, Harry peut voir ce qu'il en est en demandant à **svn status** de contacter le dépôt :

```
$ svn status
     K raisin.jpg
$ svn status -u
     B         32   raisin.jpg
$ svn update
  B  raisin.jpg
$ svn status
$
```

Si le verrou dans le dépôt a été cassé, alors **`svn status --show-updates`** affiche un `B` (pour « Broken » — « cassé » en anglais) à côté du fichier. Si un nouveau verrou existe en lieu et place de l'ancien, alors un `T` (pour « sTolen » — « volé » en anglais) est affiché. Finalement, **svn update** détecte les jetons de verrouillage défunts et les supprime de la copie de travail.

Politiques de verrouillage

Il existe différentes visions de la résistance que doit avoir un verrou. Certains considèrent que les verrous doivent être respectés à tout prix et donc libérables uniquement par leur détenteur ou par un administrateur. Ils affirment que si n'importe qui peut casser un verrou c'est la pagaille et tout le concept de verrouillage est mis par terre. D'autres pensent que les verrous sont d'abord et avant tout un outil de communication. Si les utilisateurs cassent les verrous des autres en permanence, c'est un problème culturel de l'équipe qui ne peut pas être résolu par un outil logiciel.

Subversion souscrit à la version « douce » mais autorise cependant les administrateurs à mettre en place une politique plus stricte via l'utilisation de procédures automatiques. En particulier, les procédures automatiques de pré-verrouillage

(fichier `pre-lock`) et de pré-déverrouillage (fichier `pre-unlock`) permettent aux administrateurs de décider dans quelles situations la création ou la libération d'un verrou est autorisée. En fonction de l'existence préalable ou non d'un verrou, ces deux procédures automatiques décident s'il convient ou non d'autoriser tel utilisateur à casser ou voler tel verrou. Des procédures automatiques de post-verrouillage (fichier `post-lock`) et de post-déverrouillage (fichier `post-unlock`) sont également disponibles et peuvent être utilisées pour envoyer des emails suite aux actions de verrouillage. Pour en savoir plus sur les procédures automatiques, voir la section intitulée « Mise en place des procédures automatiques ».

Communication par l'intermédiaire des verrous

Nous avons vu comment **svn lock** et **svn unlock** peuvent être utilisés pour poser, libérer, casser ou voler des verrous. Cela résout le problème de la sérialisation des accès à un fichier. Mais qu'en est-il du problème plus vaste d'éviter les pertes de temps ?

Par exemple, supposons que Harry verrouille un fichier image et commence à l'éditer. Pendant ce temps, loin de là, Sally veut faire la même chose. Elle ne pense pas à faire un **`svn status --show-updates`** et n'a donc pas la moindre idée que Harry a déjà verrouillé le fichier. Elle passe des heures à modifier le fichier et quand elle tente de propager ses changements, elle découvre soit que le fichier est verrouillé, soit que son propre fichier n'était pas à jour. Quoi qu'il en soit, ses modifications ne peuvent pas être fusionnées avec celles de Harry. L'un des deux doit passer ses modifications par pertes et profits, un temps conséquent a été gaspillé.

La solution proposée par Subversion à ce problème est de fournir un mécanisme pour rappeler aux utilisateurs qu'un fichier devrait être verrouillé *avant* de faire des modifications. Ce mécanisme est mis en œuvre par une propriété spéciale : `svn:needs-lock`. Si cette propriété est associée à un fichier (quelle que soit sa valeur, qui n'est pas prise en compte), alors Subversion essaie d'utiliser les permissions du système de fichiers pour le placer en lecture seule — à moins, bien sûr, que l'utilisateur ait explicitement verrouillé le fichier. Quand un jeton de verrouillage est présent (indiquant que **svn lock** a été lancée), le fichier est placé en lecture-écriture. Quand le verrou est libéré, le fichier passe de nouveau en lecture seule.

La théorie est donc que si le fichier image a cette propriété définie, alors Sally remarquera tout de suite quelque chose d'étrange à l'ouverture du fichier : beaucoup d'applications avertissent l'utilisateur immédiatement quand un fichier en lecture seule est ouvert pour édition et pratiquement toutes l'empêcheront de sauvegarder ses modifications dans le fichier. Cela lui rappellera de verrouiller le fichier avant de l'éditer, découvrant ainsi le verrou pré-existant :

```
$ /usr/local/bin/gimp raisin.jpg
gimp: erreur: le fichier est en lecture seule !
$ ls -l raisin.jpg
-r--r--r--   1 sally   sally   215589 juin  8 19:23 raisin.jpg
$ svn lock raisin.jpg
svn: avertissement : Échec de la demande de verrou : 423 verrouillé
(http://svn.exemple.com)
$ svn info http://svn.exemple.com/depot/projet/raisin.jpg | grep errou
Nom de verrou : opaquelocktoken:fc2b4dee-98f9-0310-abf3-653ff3226e6b
Propriétaire du verrou : harry
Verrou créé : 2006-06-08 07:29:18 -0500 (jeu. 08 juin 2006)
Commentaire de verrouillage (1 ligne):
J'effectue quelques retouches.  Je le verrouille pour deux heures.
$
```

Les utilisateurs et les administrateurs sont tous encouragés à positionner la propriété `svn:needs-lock` sur les fichiers qui ne peuvent pas être contextuellement fusionnés. C'est la technique de base pour favoriser les bonnes habitudes de verrouillage et éviter les pertes de temps.

Notez que cette propriété est un outil de communication qui fonctionne indépendamment de la politique de verrouillage. Autrement dit, n'importe quel fichier peut être verrouillé, que cette propriété existe ou pas. Et réciproquement, l'existence de cette propriété ne rend pas obligatoire le verrouillage pour pouvoir propager des modifications.

Malheureusement, le système n'est pas parfait. Il est possible que, même si le fichier possède la propriété, l'avertissement de lecture seule ne marche pas. Quelquefois, les applications ne suivent pas les normes et « piratent » le fichier en lecture seule, autorisant sans rien dire l'utilisateur à modifier et sauvegarder le fichier. Subversion ne peut pas faire grand chose dans ce

genre de cas : au final, rien ne remplace une bonne communication entre les membres d'une équipe [9].

Définition de références externes

Parfois il peut être utile de construire une copie de travail issue de différentes extractions. Par exemple, vous pouvez avoir envie d'avoir différents sous-répertoires provenant de différents endroits du dépôt ou même carrément de différents dépôts. Vous pouvez arriver à un tel enchevêtrement manuellement, en utilisant **svn checkout**, pour créer le genre de structure voulu pour votre copie de travail. Mais si cette configuration est importante pour tous les utilisateurs de votre dépôt, chacun doit effectuer les mêmes opérations d'extraction que vous.

Heureusement, Subversion supporte les *définitions de références externes*. Une définition de référence externe est une association entre un répertoire local et une URL (et idéalement un numéro de révision particulier) pour un répertoire suivi en versions. Dans Subversion, vous déclarez les définitions de références externes dans des groupes en utilisant la propriété `svn:externals`. Vous pouvez créer et modifier cette propriété en utilisant **svn propset** ou **svn propedit** (voir la section intitulée « Manipulation des propriétés »). Elle peut être définie sur tous les répertoires suivis en versions et sa valeur décrit à la fois l'URL du dépôt externe et le répertoire côté client dans lequel est extrait cette URL.

L'un des attraits de la propriété `svn:externals` est qu'une fois qu'elle est définie pour un répertoire suivi en versions, chaque utilisateur qui extrait une copie de travail de ce répertoire bénéficie des définitions de références externes. En d'autres termes, une fois qu'un utilisateur a fait l'effort de définir la structure de la copie de travail imbriquée, tout le monde en bénéficie automatiquement : Subversion, lors de l'extraction de la copie de travail originale, extraie également les copies de travail externes.

Les sous-répertoires cibles des définitions de références externes *ne doivent pas* déjà exister sur votre système ou sur le systèmes des autres utilisateurs : Subversion les crée lors de l'extraction des copies de travail externes.

Vous bénéficiez avec les définitions de références externes de tous les avantages liés aux propriétés Subversion. Les définitions sont suivies en versions. Si vous avez besoin de changer une définition de référence externe, vous pouvez le faire à l'aide des sous-commandes classiques sur les propriétés. Quand vous propagez des modifications relatives à la propriété `svn:externals`, Subversion synchronise les éléments extraits par rapport à la définition de références externes modifiée dès que vous lancez **`svn update`**. Tous ceux qui mettent à jour leur copie de travail reçoivent vos modifications concernant les définitions de références externes.

Comme la valeur de la propriété `svn:externals` est constituée de plusieurs lignes, nous vous recommandons fortement d'utiliser **svn propedit** plutôt que **svn propset**.

Les versions de Subversion antérieure à 1.5 utilisent un format de définitions externes qui est un tableau sur plusieurs lignes composées de sous-répertoires (relativement au répertoire suivi en versions sur lequel est définie la propriété), d'indicateurs de révision optionnels et l'URL, absolue et complètement qualifiée, du dépôt Subversion. Par exemple :

```
$ svn propget svn:externals calc
tierce-partie/sons                 http://svn.exemple.com/depot/sons
tierce-partie/themes -r148         http://svn.exemple.com/projet-themes
tierce-partie/themes/outils -r21   http://svn.exemple.com/outils-themes
```

Quand quelqu'un extrait une copie de travail du répertoire `calc` décrit dans l'exemple ci-dessus, Subversion extrait également les éléments trouvés dans les définitions de références externes.

```
$ svn checkout http://svn.exemple.com/depot/calc
A  calc
A  calc/Makefile
A  calc/entier.c
A  calc/bouton.c
Révision 148 extraite.
```

[9] À part, peut-être, la fusion mentale vulcaine.

```
Récupération de la référence externe dans 'calc/tierce-partie/sons'
A  calc/tierce-partie/sons/ding.ogg
A  calc/tierce-partie/sons/dong.ogg
A  calc/tierce-partie/sons/clang.ogg
…
A  calc/tierce-partie/sons/bang.ogg
A  calc/tierce-partie/sons/twang.ogg
Révision 14 extraite.

Récupération de la référence externe dans 'calc/tierce-partie/themes'
…
```

À partir de la version 1.5 de Subversion, un nouveau format de la propriété `svn:externals` est supporté. Les références externes sont toujours multi-lignes mais l'ordre et le format des différentes informations ont changé. La nouvelle syntaxe ressemble plus à l'ordre des arguments que vous passez à la commande **svn checkout** : l'indicateur optionnel de révision est placé en premier, puis l'URL du dépôt Subversion externe et, enfin, le sous-répertoire local relatif. Notez cependant que cette fois-ci nous n'avons pas indiqué « URL absolue et complètement qualifiée » pour le dépôt externe. En effet, le nouveau format accepte les URL relatives et les URL avec des piquets de révision. L'exemple précédent sur les références externes donne avec Subversion 1.5 :

```
$ svn propget svn:externals calc
      http://svn.exemple.com/depot/sons     tierce-partie/sons
-r148 http://svn.exemple.com/projet-themes  tierce-partie/themes
-r21  http://svn.exemple.com/outils-themes  tierce-partie/themes/outils
```

En utilisant la syntaxe avec les piquets de révision (décrite en détail dans la section intitulée « Piquets de révisions et révisions opérationnelles »), il peut aussi être écrit comme ceci :

```
$ svn propget svn:externals calc
http://svn.exemple.com/depot/sons         tierce-partie/sons
http://svn.exemple.com/projet-themes@148  tierce-partie/themes
http://svn.exemple.com/outils-themes@21   tierce-partie/themes/outils
```

Il est particulièrement conseillé d'utiliser des numéros de révision explicites dans toutes vos références externes. Ainsi, vous conservez la possibilité de décider quand rapatrier une nouvelle version de vos informations externes et quelle version exacte rapatrier. En plus de vous éviter la surprise de recevoir des changements effectués sur des dépôts tiers dont vous n'avez pas la maîtrise, l'utilisation de numéros de révisions explicites signifie aussi que, si vous revenez à une version de travail antérieure, vos références externes reviendront elles aussi dans l'état où elles étaient au moment de cette version antérieure. Cela signifie aussi que les copies de travail externes sont actualisées pour refléter *leur* état au moment de la révision antérieure. Pour des projets logiciels, cela peut faire la différence entre la réussite et l'échec de la compilation d'une version antérieure d'un code source complexe.

Pour la plupart des dépôts, les trois formats de références externes ont le même effet au final. Ils apportent tous les mêmes avantages. Malheureusement, ils possèdent aussi les mêmes inconvénients. Puisque les références indiquées utilisent des URL absolues, déplacer ou copier un répertoire auquel elles sont rattachées n'affecte pas ce qui est extrait en externe (alors qu'une référence relative est, bien évidemment, déplacée avec le répertoire). Cela peut vous induire en erreur, voire être assez frustrant, dans certaines situations. Par exemple, imaginons un répertoire racine appelé `mon-projet` pour lequel nous avons défini des références externes dans un sous-répertoire (`mon-projet/un-rep`) vers la dernière révision d'un autre sous-répertoire (`mon-projet/rep-externe`).

```
$ svn checkout http://svn.exemple.com/projets .
A    mon-projet
A    mon-projet/un-rep
A    mon-projet/rep-externe
…
Récupération de la référence externe dans 'mon-projet/un-rep/sous-rep'
Référence externe actualisée à la révision 11.
```

```
Actualisé à la révision 11.
$ svn propget svn:externals mon-projet/un-rep
sous-rep http://svn.exemple.com/projets/mon-projet/rep-externe

$
```

Maintenant utilisez la commande **svn move** pour renommer le répertoire `mon-projet`. À ce moment là, vos définitions de références externes pointent toujours vers un chemin sous le répertoire `mon-projet`, même si ce répertoire n'existe plus.

```
$ svn move -q mon-projet nouveau-projet
$ svn commit -m "Renommé mon-projet en nouveau-projet."
Suppression    mon-projet
Ajout          nouveau-projet

Révision 12 propagée.
$ svn update

Récupération de la référence externe dans 'nouveau-projet/un-rep/sous-rep'
svn: Le dépôt http://svn.exemple.com/projets/mon-projet/rep-externe n'existe pas

$
```

De plus, les URL absolues utilisées par les références externes peuvent causer des problèmes pour les dépôts accessibles via plusieurs types d'URL. Par exemple, si votre serveur Subversion est configuré pour autoriser tout le monde à consulter le dépôt via `http://` ou `https://`, mais que les opérations de propagation doivent être effectuées uniquement via `https://`, vous vous retrouvez bien embêté. Si vos références externes pointent vers une URL de type `http://`, vous ne pouvez pas effectuer de propagation depuis les copies de travail créées via ces références externes. D'un autre côté, si vous utilisez la forme `https://` pour les URL, ceux qui voudront effectuer des consultations via `http://`, parce que leur client ne sait pas traiter le `https://`, seront incapables de récupérer les éléments externes. Soyez conscient également que si vous avez besoin de déplacer toute votre copie de travail (avec **svn switch** et l'option `--relocate`), les références externes ne seront pas mises à jour en conséquence.

Subversion 1.5 franchit un grand pas dans la résolution de ces soucis. Comme indiqué précédemment, les URL utilisées dans le nouveau format des définitions des références externes peuvent être relatives. Par ailleurs, Subversion autorise une syntaxe magique pour spécifier plusieurs types d'URL relatives.

`../`
: Relative à l'URL du répertoire sur lequel la propriété `svn:externals` est définie.

`^/`
: Relative à la racine du dépôt pour lequel la propriété `svn:externals` est suivie en versions.

`//`
: Relative au type d'URL du répertoire sur lequel la propriété `svn:externals` est définie.

`/`
: Relative à l'URL du serveur sur lequel la propriété `svn:externals` est suivie en versions.

Donc, considérons pour la quatrième fois la définition de nos références externes de l'exemple précédent et utilisons la nouvelle syntaxe de différentes manière. Nous obtenons :

```
$ svn propget svn:externals calc
^/sons                            tierce-partie/sons
/themes@148                       tierce-partie/themes
//svn.exemple.com/outils-themes@21 tierce-partie/themes/outils
```

Le support des références externes dans Subversion demeure imparfait. Une définition de référence externe ne peut pointer que vers des répertoires et non vers des fichiers. Par ailleurs, le sous-répertoire local dans la définition ne peut pas contenir

d'indicateur de répertoire père (`..`) ; ainsi, `../../themes/mon-theme` est interdit. Et peut-être encore plus dommage, les copies de travail créées par la définition de références externes sont toujours déconnectées de la copie de travail primaire (celle dans laquelle la propriété `svn:externals` est définie sur le répertoire suivi en versions). Et Subversion ne peut fonctionner pleinement que sur des copies de travail d'un seul tenant. Ainsi, si vous voulez propager des changements que vous avez effectués sur une ou plusieurs de ces copies de travail externes, vous devez lancez **svn commit** explicitement sur ces copies de travail — effectuer une propagation sur la copie de travail primaire ne traite pas récursivement les copies de travail externes.

Nous avons déjà mentionné certains défauts de l'ancien format de `svn:externals` et comment la version 1.5 de Subversion les corrige. Mais faites attention, en utilisant les nouveaux formats, à ne pas pénaliser accidentellement des utilisateurs qui accèdent à votre dépôt avec de vieux clients Subversion. Alors que les clients Subversion 1.5 supportent toujours le format original des définitions de références externes, les vieux clients *ne sont pas capables* d'analyser le nouveau format.

En plus des commandes **svn checkout**, **svn update**, **svn switch** et **svn export** qui gèrent effectivement les sous-répertoires *disjoints* (ou déconnectés) où sont extraites les références externes, la commande **svn status** reconnaît également les définitions de références externes. Elle affiche un code de statut `X` pour les sous-répertoires externes disjoints et parcourt ces sous-répertoires pour afficher le statut des éléments externes eux-mêmes. Vous pouvez passer l'option `--ignore-externals` à n'importe laquelle de ces commandes pour désactiver le traitement des définitions de références externes.

Piquets de révisions et révisions opérationnelles

Continuellement, nous copions, déplaçons, renommons et remplaçons des fichiers et des répertoires sur nos ordinateurs. Et votre système de gestion de versions ne doit pas être un obstacle à ces opérations sur les fichiers et répertoires suivis en versions. La gestion des fichiers par Subversion se fait pratiquement oublier, étant presque aussi flexible pour les fichiers suivis en versions que pour les autres. Mais cette flexibilité signifie qu'au cours de la vie de votre dépôt un objet suivi en versions a un certain nombre de chemins et qu'un chemin donné peut représenter plusieurs objets suivis en versions tout à fait différents. Cela ajoute un niveau de complexité supplémentaire dans les actions sur les chemins et les objets.

Subversion est plutôt adroit pour détecter les « changements d'adresses » dans l'historique du suivi de versions d'un objet. Par exemple, si vous demandez l'historique d'un fichier qui a été renommé la semaine dernière, Subversion fournit ce journal (la révision dans laquelle s'est produit le changement de nom) et les journaux pertinents avant et après ce renommage. Ainsi, la plupart du temps, vous n'avez pas à vous préoccuper de ces opérations. Mais il arrive que Subversion ait besoin de votre aide pour lever des ambiguïtés.

L'exemple correspondant le plus simple est quand un fichier ou un répertoire est supprimé du suivi de versions, puis qu'un nouveau répertoire ou fichier est créé avec le même nom et ajouté au suivi de versions. L'objet qui a été effacé et celui qui a été ajouté plus tard ne sont pas les mêmes. Ils se trouve qu'ils ont juste le même chemin (`/trunk/objet` par exemple). Que signifie alors de demander à Subversion l'historique de `/trunk/objet` ? La question concerne-t-elle l'objet actuellement à cet emplacement ou l'objet précédent qui a été supprimé ? Ou encore les opérations sur *tous* les objets qui ont résidé à cet emplacement ? Subversion a besoin de savoir ce que vous demandez réellement.

Et, en raison des déplacements, l'historique des objets suivis en versions peut être beaucoup plus tordu que cela. Par exemple, vous pouvez avoir un répertoire appelé `concept`, contenant une ébauche de projet logiciel sur lequel vous vous êtes essayé. Il se peut que ce projet mûrisse et que l'idée soit pertinente au point que, chose inimaginable, vous décidiez de donner un nom au projet [10]. Imaginons que vous nommiez ce logiciel `frabnaggilywort`. Il semble alors logique de renommer le répertoire `concept` en `frabnaggilywort` pour refléter le nom du projet. L'eau coule sous les ponts et Frabnaggilywort sort en version 1.0, est téléchargé et utilisé quotidiennement par des tonnes de gens qui veulent se faciliter la vie.

Quelle belle histoire ! Mais elle ne s'arrête pas là. Comme vous avez une âme d'entrepreneur, vous avez déjà une autre idée derrière la tête : vous créez donc un nouveau répertoire `concept` et la boucle est bouclée. En fait, ce cycle recommence plusieurs fois au fil du temps, à chaque fois à partir de ce vieux répertoire `concept` qui, quelquefois, est renommé, quand l'idée plaît et, d'autres fois, est effacé quand l'idée ne convient pas. En plus, pour être réellement tordu, vous donnez parfois à `concept` un autre nom temporaire, puis renommez ce même répertoire `concept` pour une raison quelconque.

Avec de tels scénarios, demander à Subversion d'apprendre à travailler avec ces renommages multiples est un peu comme dire à un automobiliste de la banlieue de prendre la direction de Paris et de prendre à gauche sur « la rue du Château » : il croisera la rue du Château à Asnières, La Garenne-Colombes, Nanterre, Neuilly, Rueil-Malmaison, … et, non, ce n'est pas la même rue à chaque fois. De la même manière, Subversion a besoin d'un peu plus de précisions pour travailler correctement.

[10] « Vous n'êtes pas supposés lui donner un nom. Une fois que vous l'avez nommé, vous allez forcément vous y attacher », Bob Razowski (le cyclope de Monstres et Cie).

Dans sa version 1.1, Subversion a introduit une façon de spécifier de quelle rue du Château on parle. Cela s'appelle le *piquet de révision* et c'est uniquement destiné à identifier de manière unique une branche de l'historique. Comme il y a au plus un objet suivi en versions à un endroit et à un moment donnés (ou plus précisément à une révision donnée), la combinaison d'un chemin et d'un piquet de révision est tout ce dont vous avez besoin pour désigner une branche spécifique de l'historique. Les piquets de révision sont indiqués au client Subversion en utilisant la notation *at* (on l'appelle ainsi parce que la syntaxe de la commande utilise le « signe arobase » `@`) suivi du piquet de révision demandé, en fin de chemin.

Mais alors qu'en est-il de l'option `--revision` (`-r`) dont nous avons tant parlé dans ce livre ? Cette révision (ou ensemble de révisions) est appelée la *révision opérationnelle* (ou *ensemble de révisions opérationnelles*). Une fois qu'une branche particulière de l'historique a été identifiée en utilisant un chemin et un piquet de révision, Subversion effectue la requête en utilisant la révision opérationnelle (ou l'ensemble de révisions opérationnelles). Pour reprendre notre analogie avec les rues françaises, si on vous dit d'aller au 15 de la rue du Château à Rueil-Malmaison [11], vous pouvez penser que « la rue du Château » est le chemin dans le système de fichiers et « Rueil-Malmaison » le piquet de révision. Ces deux informations identifient de manière unique une route donnée et vous évitent de parcourir une autre rue du Château à la recherche de votre destination finale. Maintenant, vous pouvez rechercher le « 15 » comme numéro de révision opérationnelle puisque nous savons exactement où aller.

Algorithme des piquets de révision

Subversion utilise l'algorithme des piquets de révision chaque fois qu'il doit résoudre une ambiguïté dans les chemins et numéros de versions fournis en ligne de commande. Voici un exemple d'une telle ligne de commande :

```
$ svn commande -r REVISION-OPERATIONNELLE element@PIQUET-DE-REVISION
```

Si *`REVISION-OPERATIONNELLE`* est plus vieille que *`PIQUET-DE-REVISION`*, alors l'algorithme est le suivant :

1. Trouver *`element`* dans la révision identifiée par *`PIQUET-DE-REVISION`*. Il ne peut y avoir qu'un seul objet.
2. Parcourir l'historique de l'objet à l'envers (y compris en tenant compte d'éventuels renommages) jusqu'à son ancêtre dans la révision *`REVISION-OPERATIONNELLE`*.
3. Effectuer la requête sur cet ancêtre, où qu'il soit et quel que soit son nom (actuel et à ce moment là).

Mais que se passe-t-il si *`REVISION-OPERATIONNELLE`* est *plus récente* que *`PIQUET-DE-REVISION`* ? Et bien, cela ajoute un peu de complexité à la recherche du chemin dans *`REVISION-OPERATIONNELLE`*, parce que l'historique du chemin peut avoir bifurqué à plusieurs reprises (en raison d'opérations de copie) entre *`PIQUET-DE-REVISION`* et *`REVISION-OPERATIONNELLE`*. Et ce n'est pas tout car, de toute façon, Subversion ne stocke pas suffisamment d'informations pour retracer de façon performante l'historique d'un élément dans le sens chronologique. Donc, dans ce cas, l'algorithme est un peu différent :

1. Trouver *`element`* dans la révision identifiée par *`REVISION-OPERATIONNELLE`*. Il ne peut y avoir qu'un seul objet.
2. Parcourir l'historique de l'objet à l'envers (y compris en tenant compte d'éventuels renommages) jusqu'à son ancêtre dans la révision *`PIQUET-DE-REVISION`*.
3. Vérifier que la position de l'objet (son chemin) dans *`PIQUET-DE-REVISION`* est la même que dans *`REVISION-OPERATIONNELLE`*. Si c'est le cas, puisque les deux positions sont directement liées, effectuer la requête sur la position dans *`REVISION-OPERATIONNELLE`*. Sinon, la relation entre les deux n'étant pas établie, renvoyer une erreur expliquant qu'aucune position viable n'a été trouvée. On peut espérer qu'un jour Subversion sera plus flexible et saura mieux gérer ce type de cas.

Notez que même quand vous ne spécifiez pas explicitement de piquet de révision ni de numéro de révision opérationnelle, ils sont néanmoins présents. Par défaut, la valeur du piquet de révision est `BASE` pour les éléments de la copie de travail et `HEAD` pour les URL du dépôt. Et quand aucune révision opérationnelle n'est fournie, la valeur par

[11] Au 15 de la rue du Château à Rueil-Malmaison se trouve un musée d'*histoire* (consacré à Joséphine, épouse de Napoléon). Cela nous a semblé approprié…

défaut est celle du piquet de révision.

Supposons que nous ayons créé notre dépôt il y a longtemps et que dans la révision 1 nous ayons ajouté notre premier répertoire `concept` ainsi qu'un fichier `IDEE`, situé dans ce répertoire, contenant la description du concept. Nous avons ensuite ajouté et modifié de véritables lignes de code. À la révision 20, nous avons renommé ce répertoire en `frabnaggilywort`. Lors de la révision 27, nous développons un nouveau concept et un nouveau répertoire `concept` est créé pour l'héberger, avec un nouveau fichier `IDEE` pour le décrire. Cinq ans et vingt mille révisions passent, comme dans tout bon roman d'amour.

A présent, plusieurs années plus tard, nous nous demandons à quoi ressemblait le fichier `IDEE` en révision 1. Mais Subversion a besoin de savoir si nous demandons à quoi ressemble le fichier *actuel* tel qu'il était lors de la révision 1 ou si nous demandons le contenu du fichier `concept/IDEE` (quel qu'il soit) de la révision 1. Ces questions ont certainement des réponses différentes et grâce aux piquets de révisions, nous pouvons poser ces deux questions. Pour obtenir le contenu du fichier `IDEE` actuel tel qu'il était dans l'ancienne révision, tapez :

```
$ svn cat -r 1 concept/IDEE
svn: Impossible de trouver la localisation dans le dépôt de 'concept/IDEE' pour la
révision 1
```

Bien sûr, dans cet exemple, le fichier `IDEE` actuel n'existait pas lors de la révision 1, c'est pourquoi Subversion renvoie une erreur. La commande ci-dessus est un raccourci pour la notation plus longue qui explicite le piquet de révision. La notation complète est donc :

```
$ svn cat -r 1 concept/IDEE@BASE
svn: Impossible de trouver la localisation dans le dépôt de 'concept/IDEE' pour la
révision 1
```

On obtient le résultat attendu.

Le lecteur perspicace est certainement en train de se demander si la syntaxe des piquets de révision ne pose pas de problèmes pour les chemins ou les URL qui comportent déjà le signe arobase. Après tout, comment **svn** peut-il savoir si `nouveau@11` est le nom d'un répertoire dans mon arborescence ou juste la syntaxe pour « révision 11 de `nouveau` » ? Dieu merci, alors que **svn** opte par défaut pour cette dernière hypothèse, il existe une solution de contournement triviale : il suffit juste d'ajouter un signe arobase à la fin du chemin, comme ceci : `nouveau@11@`. **svn** ne s'intéresse qu'au dernier arobase de l'argument et il n'est pas considéré comme illégal d'omettre le spécificateur de piquet de révision après cet arobase. Cette solution de contournement s'applique même aux chemins qui se terminent par arobase (utilisez `nom-du-fichier@@` pour désigner le fichier `nom-du-fichier@`.

Posons maintenant l'autre question : dans la révision 1, quel était le contenu du fichier qui occupait l'adresse `concept/IDEE` à ce moment là ? Nous allons utiliser explicitement un piquet de révision pour nous aider :

```
$ svn cat concept/IDEE@1
L'idée de ce projet est de fournir un logiciel qui peut frabber un
naggily wort. Frabber les naggilys worts est particulièrement difficile
et ne pas le faire correctement aurait des conséquences inimaginables.
Nous devons donc utiliser des mécanismes de vérification des
entrées et des données du dernier cri.
```

Remarquez que cette fois nous n'avons pas fourni de révision opérationnelle. C'est parce que, quand aucune révision opérationnelle n'est spécifiée, Subversion considère que le numéro de révision opérationnelle est égal au piquet de révision.

Comme vous pouvez le constater, la résultat de la commande semble être correct. Le texte parle même de "frabber les naggilys worts", ce qui laisse supposer que c'est certainement le fichier décrivant le logiciel maintenant connu sous le nom de Frabnaggilywort. En fait, on peut le vérifier en combinant un piquet de révision explicite et un numéro de révision opérationnelle explicite. Nous savons que dans `HEAD`, le projet Frabnaggilywort se situe dans le répertoire `frabnaggilywort`. Nous spécifions donc que nous voulons voir à quoi ressemblait le fichier `frabnaggilywort/IDEE` identifié dans `HEAD` au moment de la révision 1.

```
$ svn cat -r 1 frabnaggilywort/IDEE@HEAD
L'idée de ce projet est de fournir un logiciel qui peut frabber un
naggily wort. Frabber les naggilys worts est particulièrement difficile
et ne pas le faire correctement aurait des conséquences inimaginables.
Nous devons donc utiliser des mécanismes de vérification des
entrées et des données du dernier cri.
```

Vous pouvez aussi spécifier des piquets de révision et des révisions opérationnelles moins triviales. Par exemple, disons que `frabnaggilywort` a été effacé de `HEAD`, mais nous savons qu'il existait en révision 20 et nous voulons voir les différences entre la révision 4 et la révision 10 pour son fichier `IDEE`. Nous pouvons utiliser le piquet de révision 20 en conjonction avec l'URL qu'avait le fichier `frabnaggilywort/IDEE` dans la révision 20 et utiliser 4 et 10 pour spécifier l'intervalle de révisions opérationnelles.

```
$ svn diff -r 4:10 http://svn.red-bean.com/projets/frabnaggilywort/IDEE@20
Index: frabnaggilywort/IDEE
===================================================================
--- frabnaggilywort/IDEE (révision 4)
+++ frabnaggilywort/IDEE (révision 10)
@@ -1,5 +1,5 @@
-L'idée de ce projet est de fournir un logiciel qui peut frabber un
-naggily wort. Frabber les naggilys worts est particulièrement difficile
-et ne pas le faire correctement aurait des conséquences inimaginables.
-Nous devons donc utiliser des mécanismes de vérification des
-entrées et des données du dernier cri.
+L'idée de ce projet est de fournir un logiciel client-serveur qui peut
+frabber un naggily wort de manière distante. Frabber les naggilys worts
+est particulièrement difficile et ne pas le faire correctement aurait
+des conséquences inimaginables. Nous devons donc utiliser des mécanismes
+de vérification des entrées et des données du dernier cri.
```

Heureusement, la plupart d'entre vous n'auront pas à faire face à des situations aussi complexes. Mais si jamais c'est le cas, rappelez-vous que les piquets de révisions sont les informations complémentaires dont a besoin Subversion pour lever toute ambiguïté.

Listes de modifications

Il est très courant pour un développeur d'avoir à effectuer, en même temps, des modifications multiples n'ayant rien à voir entre elles sur une portion de code donnée. Ce n'est pas nécessairement la preuve d'une mauvaise gestion de son temps, ni d'une forme de masochisme numérique. Un ingénieur de développement repère souvent des bogues annexes lorsqu'il travaille sur un morceau de code particulier. Ou alors c'est une fois rendu à mi-chemin d'un changement important qu'il prend conscience que la solution qu'il a choisie serait mieux propagée sous la forme de plusieurs unités logiques plus petites, ces unités pouvant se recouper, affecter des fichiers différents d'un même module ou même toucher à des lignes différentes d'un même fichier.

Plusieurs méthodes sont à disposition des développeurs pour gérer ces ensembles de modifications. Certains développeurs utilisent des copies de travail séparées, du même dépôt, pour y conserver les progrès faits pour chaque changement. D'autres développeurs choisissent de créer au sein du dépôt des branches fonctionnelles à durée de vie très courte et d'utiliser une unique copie de travail qu'ils feront pointer selon les besoins du moment vers une branche de ce type ou vers une autre. Enfin, d'autres développeurs utilisent les outils **diff** et **patch** pour sauvegarder et restaurer des changements non propagés sur les fichiers touchés par les modifications. Chacune de ces méthodes a des avantages et des inconvénients et, en général, le détail des changements à effectuer influence fortement le choix de la méthode utilisée pour les distinguer.

Subversion 1.5 a introduit une nouvelle fonctionnalité, les *listes de modifications*, qui vient s'ajouter aux méthodes mentionnées ci-dessus. Les listes de modifications sont en gros des étiquettes arbitraires (une au plus par fichier pour l'instant) attachées à des fichiers de la copie de travail dans le seul but de regrouper plusieurs fichiers ensemble. Les utilisateurs de bon nombre de logiciels fournis par Google sont déjà habitués à ce concept. Dans Gmail [http://mail.google.com/], vous associez des étiquettes arbitraires à des e-mails et plusieurs e-mails peuvent être considérés comme faisant partie du même groupe s'ils partagent une étiquette donnée. Dès lors, ne voir qu'un groupe d'e-mails portant la même étiquette n'est plus qu'un simple jeu d'affichage. De nombreux autres sites web 2.0 possèdent des mécanismes similaires. Prenez par exemple les « tags » (« étiquette » en anglais) utilisés sur des sites comme YouTube [http://www.youtube.com/] ou Flickr [http://www.flickr.com/], les « catégories » utilisées pour regrouper les articles de blogs, etc. Aujourd'hui les gens ont compris que l'organisation des

données est essentielle et la façon dont ces données sont organisées doit être flexible. Le vieux paradigme des répertoires et des fichiers est bien trop rigide pour certaines applications.

Cette fonctionnalité de Subversion vous permet de créer des listes de modifications en étiquetant les fichiers que vous voulez inclure, de supprimer ces étiquettes et de limiter le rayon d'action des sous-commandes aux seuls fichiers qui portent l'étiquette donnée. Dans ce paragraphe, nous allons voir en détails comment accomplir ces actions.

Création et modification d'une liste de modifications

Vous pouvez créer, modifier et supprimer des listes de modifications en utilisant la commande **svn changelist**. Plus précisément, vous pouvez utiliser cette commande pour activer ou désactiver l'association d'une liste de modifications avec un fichier donné de la copie de travail. La création d'une liste de modifications a lieu la première fois que vous étiquetez un fichier avec ce nom de liste ; elle n'est supprimée que quand vous effacez l'étiquette du dernier fichier qui la portait. Examinons un cas concret pour illustrer ces notions.

Harry est en train de corriger des bogues dans le module de logique mathématique de l'application « calculatrice ». Son travail l'amène à modifier deux fichiers :

```
$ svn status
M      entier.c
M      ops-math.c
$
```

En testant son correctif, Harry s'aperçoit que ses modifications lui indiquent qu'un bogue collatéral existe au sein de la logique de l'interface utilisateur, située dans le fichier `bouton.c`. Harry décide alors qu'il va aussi corriger ce bogue, dans une propagation séparée de ses propres correctifs mathématiques. Dans une copie de travail de petite taille, ne contenant qu'une poignée de fichiers, et pour juste quelques modifications logiques, Harry pourrait probablement gérer mentalement ces deux ensembles logiques de modifications sans le moindre problème. Mais aujourd'hui il a décidé qu'il allait utiliser les listes de modifications de Subversion, pour faire une faveur aux auteurs de ce livre.

Harry commence par créer une première liste de modifications et y associe les deux premiers fichiers qu'il a déjà modifié. Pour ce faire, il utilise la commande **svn changelist** afin d'associer le même nom arbitraire de liste de modifications aux deux fichiers :

```
$ svn changelist correctifs-maths entier.c ops-math.c
Le chemin 'entier.c' fait maintenant partie de la liste de changement
'correctifs-maths'.
Le chemin 'ops-math.c' fait maintenant partie de la liste de changement
'correctifs-maths'.
$ svn status

--- Liste de changements 'correctifs-maths' :
M      entier.c
M      ops-math.c
$
```

Comme vous pouvez le constater, le résultat de **svn status** reflète bien ce nouvel ensemble.

Harry se lance alors dans la correction du problème d'interface graphique collatéral. Puisqu'il sait quel fichier il va modifier, il associe également ce chemin à une liste de modifications. Mais malencontreusement Harry associe ce troisième fichier à la même liste de modifications que les deux premiers :

```
$ svn changelist correctifs-maths bouton.c
Le chemin 'bouton.c' fait maintenant partie de la liste de changement
'correctifs-maths'.
$ svn status

--- Liste de changements 'correctifs-maths' :
       bouton.c
M      entier.c
M      ops-math.c
```

```
$
```

Par chance, Harry prend conscience de son erreur. Deux options se présentent alors à lui. Il peut supprimer l'association de `bouton.c` avec la liste de modifications, puis lui associer un nouveau nom de liste de modifications :

```
$ svn changelist --remove bouton.c
Le chemin 'bouton.c' n'est plus associé à une liste de changements.
$ svn changelist correctifs-graphiques bouton.c
Le chemin 'bouton.c' fait maintenant partie de la liste de changement
'correctifs-graphiques'.
$
```

Ou alors il peut sauter l'étape de suppression et juste associer un nouveau nom de liste de modifications à `bouton.c`. Dans ce cas, Subversion signale à Harry que `bouton.c` va être supprimé de la première liste de modifications :

```
$ svn changelist correctifs-graphiques bouton.c
svn:  avertissement : Retrait de 'bouton.c' de la liste de changements (changelist)
'correctifs-maths'.
Le chemin 'bouton.c' fait maintenant partie de la liste de changement
'correctifs-graphiques'.
$ svn status

--- Liste de changements 'correctifs-graphiques' :
       bouton.c

--- Liste de changements 'correctifs-maths' :
M      entier.c
M      ops-maths.c
$
```

Harry dispose donc à présent de deux listes de modifications distinctes dans sa copie de travail et **svn status** présente ses résultats en les regroupant par liste de modifications. Notez que bien qu'Harry n'ait pas encore modifié `bouton.c`, celui-ci est quand même mentionné par **svn status** car une liste de modifications lui est associée. Les listes de modifications peuvent être associées, ou enlevées, aux fichiers à tout moment, indépendamment du fait que ces fichiers contiennent des modifications locales ou pas.

Harry règle maintenant le problème de l'interface graphique dans `bouton.c`.

```
$ svn status

--- Liste de changements 'correctifs-graphiques':
M      bouton.c

--- Liste de changements 'correctifs-maths':
M      entier.c
M      ops-math.c
$
```

Listes de modifications : des filtres pour vos opérations

Le regroupement visuel qu'Harry constate en sortie de **svn status**, comme indiqué précédemment, est intéressant d'un point de vue esthétique, mais pas vraiment utile. La commande **status** n'est qu'une des commandes qu'il est susceptible de lancer sur sa copie de travail. Heureusement, bon nombre des autres opérations de Subversion sont capables d'agir sur les listes de modifications grâce à l'option `--changelist`.

Quand l'option `--changelist` est présente, les commandes Subversion limitent leur champ d'action aux fichiers auxquels est associé le nom de liste de modifications donné. Si Harry veut voir quels changements il a effectué sur les fichiers de sa liste `correctifs-maths`, il *peut* lister explicitement les fichiers faisant partie de cette liste de modifications avec la commande **svn diff**.

```
$ svn diff entier.c ops-math.c
Index: entier.c
===================================================================
--- entier.c (révision 1157)
+++ entier.c (copie de travail)
…
Index: ops-math.c
===================================================================
--- ops-math.c (révision 1157)
+++ ops-math.c (copie de travail)
…
$
```

Cette méthode fonctionne correctement pour un petit nombre de fichiers, mais qu'en est-il si Harry a modifié une vingtaine ou une trentaine de fichiers ? Fournir la liste de tous ces fichiers serait assez pénible. Mais puisqu'il utilise les listes de modifications, Harry peut désormais éviter de lister explicitement tous les fichiers et ne donner à la place que le nom de la liste de modifications :

```
$ svn diff --changelist correctifs-maths

Index: entier.c
===================================================================
--- entier.c (révision 1157)
+++ entier.c (copie de travail)
…
Index: ops-math.c
===================================================================
--- ops-math.c (révision 1157)
+++ ops-math.c (copie de travail)
…
$
```

Et au moment de lancer la propagation, Harry peut à nouveau se servir de l'option `--changelist` pour limiter le rayon d'action de la propagation aux fichiers de sa liste de modifications. Par exemple, il peut propager ses changements concernant l'interface graphique en lançant :

```
$ svn ci -m "Corrigé un bug de l'interface graphique découvert en travaillant sur la
logique mathématique." \
      --changelist correctifs-graphiques
Envoi        bouton.c
Transmission des données .
Révision 1158 propagée.
$
```

En fait, la commande **svn commit** accepte une deuxième option liée aux listes de modifications : `--keep-changelists`. Normalement, l'association des listes de modifications avec les fichiers est supprimée dès que ceux-ci ont été propagés. Mais si l'option `--keep-changelists` est ajoutée sur la ligne de commande, les fichiers propagés (qui ne sont donc plus dans l'état modifié) restent associés aux listes de modifications en question. Dans tous les cas, propager des fichiers faisant partie d'une liste de modification laisse les autres listes de modifications intactes.

```
$ svn status

--- Liste de changements 'correctifs-maths':
M      entier.c
M      ops-math.c
$
```

L'option `--changelist` agit comme un filtre sur les cibles des commandes Subversion et n'ajoute jamais de cible à une opération. Par exemple, lors d'une opération de propagation lancée via **`svn commit /chemin/vers/rep`**, la cible est le répertoire `/chemin/vers/rep` et ses fils (avec une profondeur infinie). Si ensuite vous ajoutez une option spécifiant une liste de modifications à cette commande, seuls les fichiers se trouvant sous le chemin `/chemin/vers/rep` et associés à cette liste de modifications sont pris en compte en tant que cibles de la propagation ; ne sont pas inclus les fichiers situés ailleurs (tels ceux sous `/chemin/vers/autre-rep`), quelle que soit la liste de modifications à laquelle ils appartiennent, même s'il font partie de la même copie de travail que la ou les cibles de l'opération.

Même la commande **svn changelist** accepte l'option `--changelist`. Ceci vous permet de renommer ou supprimer facilement une liste de modifications :

```
$ svn changelist bogues-maths --changelist correctifs-maths --depth infinity .
svn: avertissement : Retrait de 'entier.c' de la liste de changements (changelist)
'correctifs-maths'.
Le chemin 'entier.c' fait maintenant partie de la liste de changement 'bogues-maths'.
svn: avertissement : Retrait de 'ops-math.c' de la liste de changements (changelist)
'correctifs-maths'.
Le chemin 'ops-math.c' fait maintenant partie de la liste de changement
'bogues-maths'.
$ svn changelist --remove --changelist bogues-maths --depth infinity .
Le chemin 'entier.c' n'est plus associé à une liste de changements.
Le chemin 'ops-math.c' n'est plus associé à une liste de changements.
$
```

Enfin, vous pouvez spécifier plusieurs instances de l'option `--changelist` dans une même ligne de commande. Ceci limite le champ d'action de votre opération aux fichiers faisant partie de toutes les listes de modifications spécifiées.

Limitations des listes de modifications

La fonctionnalité de listes de modifications de Subversion est un outil très pratique pour créer des groupes de fichiers au sein de la copie de travail, mais elle a cependant quelques limitations. Les listes de modifications sont des objets contenus à l'intérieur d'une copie de travail, ce qui signifie que les associations entre fichiers et listes de modifications ne peuvent pas être propagées vers le dépôt, ni partagées avec d'autres utilisateurs. Les listes de modifications ne peuvent être associées qu'à des fichiers, Subversion n'offre pas cette possibilité pour les répertoires. Enfin, vous pouvez avoir au plus un nom de liste de modifications associé à un fichier donné. C'est ici que l'analogie avec les articles de blogs et les services d'étiquetage de photos en ligne part en fumée. S'il vous faut associer un fichier à plusieurs listes de modifications, vous êtes coincé.

Modèle de communication réseau

À un moment ou à un autre, vous aurez besoin de comprendre comment le client Subversion communique avec le serveur. La couche réseau de Subversion est abstraite, c'est-à-dire que les clients Subversion ont le même comportement quel que soit le type de serveur auquel ils ont affaire. Qu'ils communiquent via le protocole HTTP (`http://`) avec un serveur HTTP Apache ou via le protocole Subversion (`svn://`) avec **svnserve**, le modèle de communication réseau est le même. Dans cette section, nous expliquons les fondamentaux de ce modèle de communication réseau, y compris la façon dont Subversion gère les authentifications et les autorisations.

Requêtes et réponses

Le client Subversion passe la plupart de son temps à gérer des copies de travail. Cependant, quand il a besoin d'informations disponibles dans un dépôt distant, il envoie une requête sur le réseau et le serveur lui répond. Les détails du protocole réseau sont cachés à l'utilisateur : le client essaie d'accéder à une URL et, suivant le format de cette URL, utilise un protocole particulier pour contacter le serveur (voir URL du dépôt).

Tapez **`svn --version`** pour voir quels types d'URL et de protocoles sont utilisables par votre client.

Quand le serveur reçoit une requête d'un client, il demande souvent au client de s'identifier. Il envoie un défi d'authentification vers le client et le client répond en fournissant les *éléments d'authentification* au serveur. Une fois cette authentification terminée, le serveur répond à la requête originale du client. Remarquez que ce fonctionnement est différent de celui de CVS où le client envoie systématiquement au préalable ses identifiants de connexion (procédure de « log in »), avant même de formuler la moindre requête. Dans Subversion, le serveur requiert explicitement l'authentification du client (par un défi d'authentification) au moment approprié, au lieu que ce soit le client qui s'authentifie a priori. Certaines opérations sont donc effectuées plus élégamment. Par exemple, si un serveur est configuré pour laisser tout le monde accéder au dépôt en lecture, alors le serveur n'envoie jamais de défi d'authentification quand un client tente un **svn checkout**.

Si une requête d'un client conduit à la création d'une nouvelle révision du dépôt (par exemple un **svn commit**), alors Subversion utilise le nom d'utilisateur fourni lors de la phase d'authentification comme auteur de la révision. C'est-à-dire que le nom d'utilisateur est stocké dans la propriété `svn:author` de la nouvelle révision (voir la section intitulée « Propriétés dans Subversion »). Si le client n'a pas été authentifié (en d'autres termes, si le serveur n'a jamais envoyé de défi d'authentification), alors la propriété `svn:author` de la révision est vide.

Mise en cache des éléments d'authentification du client

Beaucoup de serveurs sont configurés pour demander une authentification à chaque requête. Ce serait particulièrement pénible pour les utilisateurs s'ils devaient taper leur mot de passe à chaque fois. Heureusement, le client Subversion a une solution : la mise en cache sur le disque des éléments d'authentification. Par défaut, si le client en ligne de commande s'authentifie avec succès auprès d'un serveur, le client sauvegarde les éléments d'authentification dans une zone privée propre à l'utilisateur (`~/.subversion/auth/` sur les systèmes de type Unix ou `%APPDATA%/Subversion/auth/` sur Windows) ; voir la section intitulée « Zone de configuration des exécutables » pour plus de détails). Les éléments d'authentification sont mis en cache sur le disque, classés suivant une combinaison du nom, du port et du domaine d'authentification du serveur.

Quand le client reçoit un défi d'authentification, il regarde d'abord s'il dispose des éléments appropriés dans le cache disque de l'utilisateur. Si ce n'est pas le cas, ou si les éléments conduisent à un échec lors de la tentative d'authentification, le client demande alors (c'est son comportement par défaut) à l'utilisateur les informations nécessaires.

Ici, le lecteur soucieux de sécurité va immédiatement commencer à s'inquiéter : « Mettre en cache des mots de passe sur le disque ? Quelle horreur ! Jamais ! »

Les développeurs de Subversion reconnaissent le bien fondé du problème et c'est pourquoi Subversion est capable de fonctionner avec les différents mécanismes offerts par le système d'exploitation et l'environnement de travail pour minimiser les risques de fuites d'information. Voici ce que cela veut dire sur les plates-formes les plus courantes :

- Sur Windows 2000 et ses successeurs, le client Subversion utilise les services cryptographiques standards de Windows pour chiffrer le mot de passe sur le disque. Comme la clé de chiffrement est gérée par Windows et qu'elle est associée à l'identifiant de connexion de l'utilisateur, lui seul peut déchiffrer le mot de passe en cache. Notez que si le mot de passe de l'utilisateur est réinitialisé par un administrateur, tous les mots de passe en cache deviennent indéchiffrables. Le client Subversion agit comme s'ils n'existaient pas, en redemandant le mot de passe quand c'est nécessaire.

- De manière similaire, sur Mac OS X, le client Subversion stocke tous les mots de passe dans le jeton de connexion (géré par le service « Keychain »), qui est protégé par le mot de passe du compte utilisateur. La configuration des « préférences utilisateur » peut imposer une politique plus stricte, comme par exemple demander le mot de passe du compte utilisateur à chaque fois qu'un mot de passe Subversion est utilisé.

- Pour les systèmes de type Unix, il n'existe pas de standard de service de type « Keychain ». Cependant, la zone de cache `auth/` est toujours protégée par les droits système et seul l'utilisateur (le propriétaire) des données peut les lire. Les droits sur les fichiers fournis par le système d'exploitation protègent les mots de passe.

Bien sûr, si vous êtes complètement paranoïaque, aucun de ces mécanismes n'est parfait. Ainsi, pour ceux qui sont prêts à sacrifier le confort d'utilisation au profit de la sécurité absolue, Subversion fournit de nombreuses façons de désactiver le système de cache d'authentification.

Pour désactiver le système de cache pour une seule commande, utilisez l'option `--no-auth-cache` :

```
$ svn commit -F log_msg.txt --no-auth-cache
Domaine d'authentification : <svn://hote.exemple.com:3690> exemple de domaine
Nom d'utilisateur :  paul
```

```
Mot de passe pour 'paul' :

Ajout         nouveau_fichier
Transmission des données .
Révision 2324 propagée.

# le mot de passe n'a pas été mis dans le cache, donc la deuxième propagation
# nous redemandera le mot de passe

$ svn delete nouveau_fichier
$ svn commit -F nouveau_msg.txt
Domaine d'authentification : <svn://hote.exemple.com:3690> exemple de domaine
Nom d'utilisateur :  paul
…
```

Autrement, si vous voulez désactiver la mise en cache des éléments d'authentification de manière permanente, vous pouvez éditer le fichier `config` de votre zone de configuration et mettre l'option `store-auth-creds` à `no`. La mise en cache des éléments d'authentification est alors désactivée pour toutes les commandes Subversion que vous effectuez sur cet ordinateur. Ceci peut être étendu à l'ensemble des utilisateurs de l'ordinateur en modifiant la configuration globale de Subversion (voir la section intitulée « Agencement de la zone de configuration »).

```
[auth]
store-auth-creds = no
```

Il arrive que les utilisateurs veuillent effacer certains mots de passe du cache disque. Pour ce faire, vous devez vous rendre dans la zone `auth/` et effacer manuellement le fichier de cache approprié. Les éléments d'authentification sont mis en cache dans des fichiers individuels ; si vous affichez chaque fichier, vous voyez des clés et des valeurs. La clé `svn:realmstring` décrit le domaine du serveur auquel est associé le fichier :

```
$ ls ~/.subversion/auth/svn.simple/
5671adf2865e267db74f09ba6f872c28
3893ed123b39500bca8a0b382839198e
5c3c22968347b390f349ff340196ed39

$ cat ~/.subversion/auth/svn.simple/5671adf2865e267db74f09ba6f872c28

K 8
username
V 3
paul
K 8
password
V 4
blah
K 15
svn:realmstring
V 45
<https://svn.domaine.com:443> Dépôt de Paul
END
```

Une fois le bon fichier trouvé, effacez-le.

Un dernier mot sur la façon dont **svn** gère l'authentification, avec un zoom sur les options `--username` et `--password`. Beaucoup de sous-commandes du client acceptent ces options, mais il est important de comprendre que l'utilisation de ces options *n'envoie pas* automatiquement les éléments d'authentification au serveur. Comme vu précédemment, le serveur *demande* explicitement l'authentification au client quand il estime que c'est nécessaire ; le client ne les envoie pas à sa convenance. Même si un nom d'utilisateur et/ou un mot de passe sont passés en option, ils ne sont envoyés au serveur que si celui-ci les demande. Ces options sont couramment utilisées pour s'authentifier sous un nom d'utilisateur différent de celui que Subversion aurait choisi par défaut (comme votre nom de compte système), ou quand on ne veut pas de commande interactive (par exemple, utilisation de la commande **svn** dans un script).

Une erreur classique consiste à mal configurer un serveur de telle sorte qu'il n'envoie jamais de défi d'authentification. Quand les utilisateurs passent les options `--username` et `--password`, ils sont surpris de voir qu'elles ne sont jamais utilisées, c'est-à-dire que les nouvelles révisions semblent toujours avoir été propagées de façon anonyme !

En résumé, voici comment un client Subversion se comporte quand il reçoit un défi d'authentification :

1. D'abord, le client vérifie si l'utilisateur a spécifié explicitement des éléments d'authentification dans la ligne de commande (options `--username` et/ou `--password`). Si c'est le cas, le client essaie de s'authentifier auprès du serveur avec ces éléménts.

2. Si les éléments d'authentification ne sont pas passés en ligne de commande, ou si ceux qui ont été fournis ne sont pas valides, le client regarde dans la zone `auth/` s'il trouve le nom, le port et le domaine du serveur pour voir si l'utilisateur a déjà les éléments d'authentification en cache. Si c'est le cas, il essaie d'utiliser ces éléments pour s'authentifier.

3. Finalement, si les mécanismes précédents ont abouti à des échecs d'authentification sur le serveur, le client se résout à demander les éléments à l'utilisateur (à moins qu'il ne lui ait été indiqué de ne pas le faire via l'option `--non-interactive` ou son équivalent spécifique au client).

Si le client réussit à s'authentifier par l'une ou l'autre de ces méthodes, il essaie de mettre en cache les éléments d'authentification sur le disque (à moins que cette fonctionnalité ne soit désactivée, comme indiqué auparavant).

Résumé

Après avoir lu ce chapitre, vous devez désormais avoir une bonne compréhension de certaines fonctionnalités de Subversion qui, bien qu'elles ne servent pas systématiquement à chaque utilisation du système de gestion de versions, peuvent rendre de grands services. Ne vous arrêtez pas là ! Lisez le chapitre suivant, où vous découvrirez les branches, les étiquettes et les fusions. Vous aurez alors la maîtrise quasi-complète du client Subversion. Bien que nos avocats ne nous autorisent pas à vous promettre quoi que ce soit, ces connaissances supplémentaires feront déjà de vous quelqu'un de bien plus branché. [12]

[12] Aucun achat nécessaire. Certaines conditions s'appliquent. Aucune garantie, ni explicite ni implicite, n'est fournie. Le kilométrage peut varier.

Chapitre 4. Gestion des branches

> « #### (C'est sur le Tronc qu'un gentleman travaille.) »
>
> —Confucius

La création, l'étiquetage et la fusion de branches sont des concepts communs à tous les systèmes de gestion de versions. Si vous n'êtes pas familier avec elles, nous fournissons dans ce chapitre une bonne introduction à ces idées. Si vous êtes familier avec elles, vous devriez, avec un peu de chance, être intéressé par la façon dont Subversion les met en pratique.

La gestion des branches est un élément fondamental de la gestion de versions. Si vous comptez utiliser Subversion pour gérer vos données, c'est une fonctionnalité dont vous ne pourrez plus vous passer. Ce chapitre suppose que vous êtes déjà familier avec les notions de bases de Subversion (Chapitre 1, *Notions fondamentales*).

Qu'est-ce qu'une branche ?

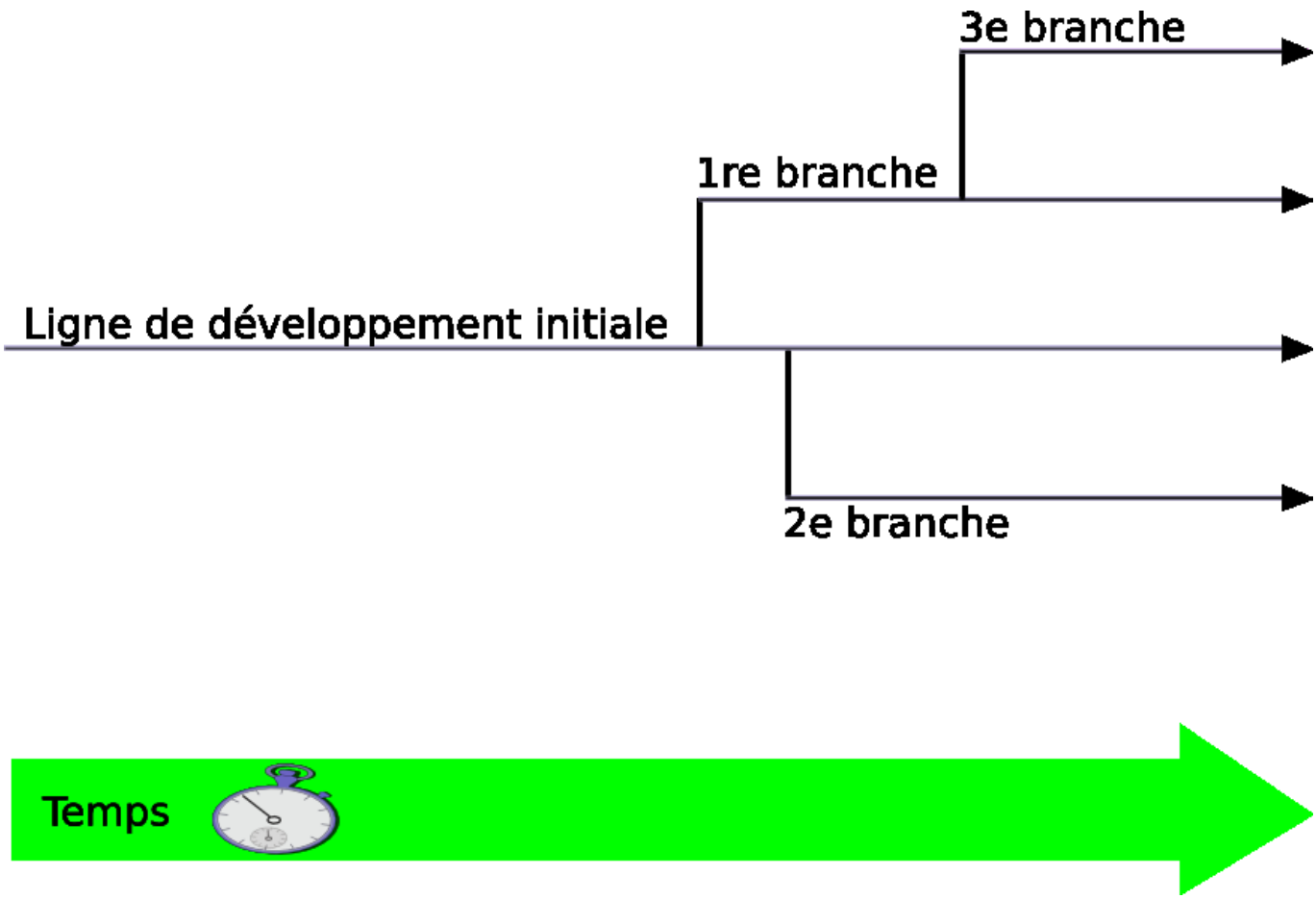
Supposons que votre travail soit de maintenir un document pour une division de votre entreprise, un manuel par exemple. Un beau jour, une autre division vous demande le même manuel, mais avec quelques parties « modifiées » spécialement pour elle, puisqu'elle fait les choses légèrement différemment.

Que faites-vous dans cette situation ? Tout naturellement, vous créez une seconde copie du document et commencez à maintenir les deux copies séparément. Puis, quand chaque division vous demande de faire des petites modifications, vous les incorporez dans une copie ou dans l'autre.

Vous voulez souvent faire la même modification dans les deux copies. Par exemple, si vous découvrez une coquille dans la première copie, il est très probable que la même coquille existe dans la deuxième copie. Les deux documents sont presque identiques, après tout ; ils ne diffèrent qu'en quelques points mineurs et spécifiques.

Voilà le concept de *branche*, c'est-à-dire 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. Une branche commence toujours sa vie en tant que copie de quelque chose, puis diffère à partir de là, selon une histoire qui lui est propre (voir la Figure 4.1, « Branches de développement »).

Figure 4.1. Branches de développement



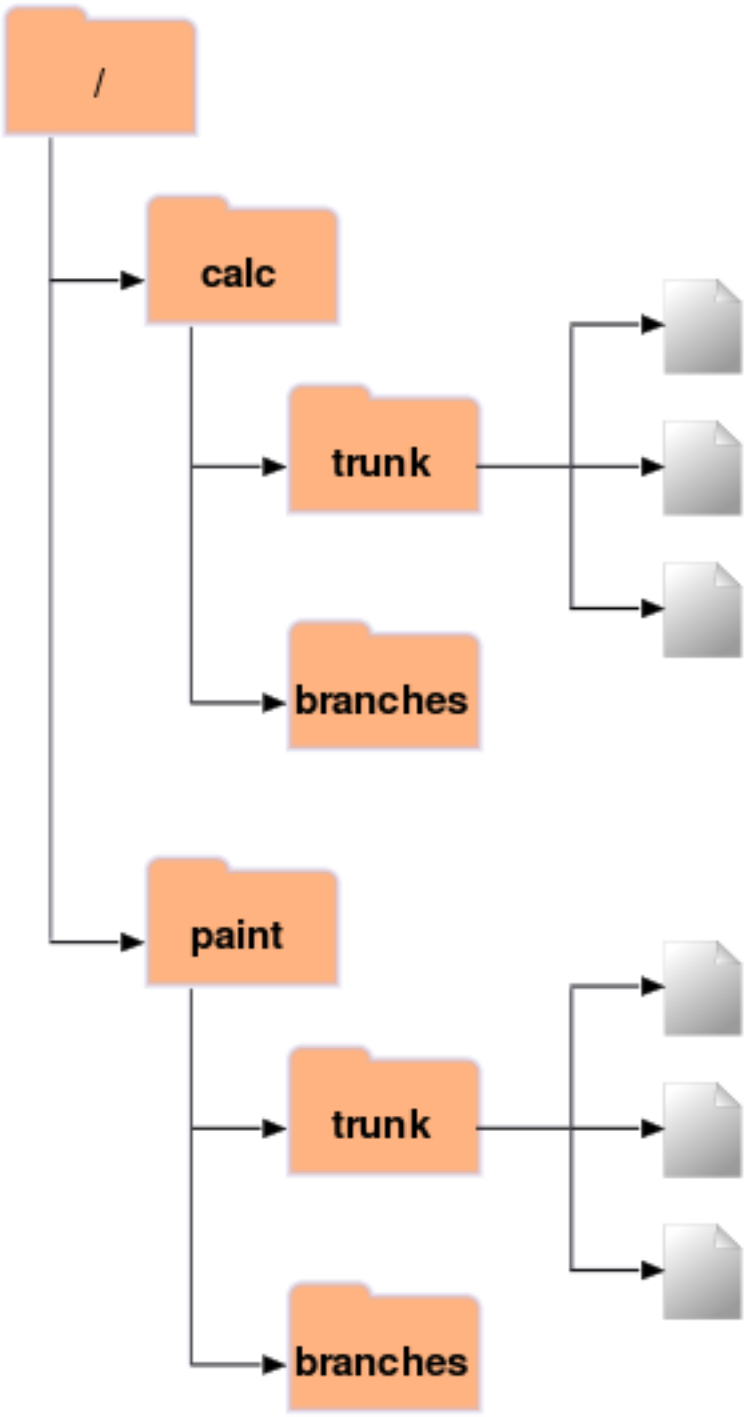
Subversion possède des commandes pour vous aider à maintenir des branches parallèles de vos fichiers et répertoires. Il vous permet de créer des branches en faisant des copies de vos données et se souvient que les copies sont liées les unes aux autres. Il vous aide aussi à dupliquer les modifications d'une branche vers une autre. Enfin, il permet que des portions de votre copie de travail correspondent à différentes branches, afin que vous puissiez « mélanger » différentes lignes de développement dans votre travail quotidien.

Utilisation des branches

Rendu à ce chapitre, vous devriez avoir compris que chaque propagation crée une arborescence de fichiers entièrement nouvelle (appelée « révision ») dans le dépôt. Si ce n'est pas le cas, retournez vous informer sur les révisions dans la section intitulée « Révisions ».

Pour ce chapitre, nous reprendrons le même exemple qu'au Chapitre 1, *Notions fondamentales*. Souvenez-vous que votre collaboratrice Sally et vous partagez un dépôt qui contient deux projets, `paint` et `calc`. Notez cependant que dans la Figure 4.2, « Structure initiale du dépôt », le dossier de chaque projet contient désormais des sous-dossiers nommés `trunk` et `branches`. Les raisons de cette arborescence apparaîtront bientôt clairement.

Figure 4.2. Structure initiale du dépôt



Comme avant, supposons que Sally et vous avez tous deux une copie de travail du projet « calc ». Plus spécifiquement, vous avez chacun une copie de travail de `/calc/trunk`. Tous les fichiers du projet sont dans ce sous-dossier plutôt que dans `/calc` lui-même, parce que votre équipe a décidé que la « ligne principale » de développement du projet allait se situer dans `/calc/trunk`.

Disons que l'on vous a attribué la tâche d'implémenter une fonctionnalité du logiciel qui prendra longtemps à écrire et touchera à tous les fichiers du projet. Le problème immédiat est que vous ne voulez pas déranger Sally, qui est en train de corriger des bogues mineurs ici et là. Elle a besoin que la dernière version du projet (dans `/calc/trunk`) demeure en permanence utilisable. Si vous commencez à propager des changements petit à petit, vous allez sûrement rendre les choses difficiles pour

Sally (ainsi que pour d'autres membres de l'équipe).

Une stratégie possible est de vous isoler : vous pouvez arrêter de partager des informations avec Sally pendant une semaine ou deux. C'est-à-dire commencer à modifier et à réorganiser les fichiers dans votre copie de travail, mais sans effectuer de propagation ni de mise à jour avant que vous n'ayez complètement terminé la tâche. Cette stratégie comporte certains risques. Premièrement, ce n'est pas sans danger. La plupart des gens aiment propager leurs modifications fréquemment, au cas où leur copie de travail aurait un accident. Deuxièmement, ce n'est pas très flexible. Si vous travaillez sur différents ordinateurs (vous avez peut-être une copie de travail de `/calc/trunk` sur deux machines différentes), vous aurez besoin de transférer manuellement vos changements entre les deux, ou bien de travailler sur une seule machine. De la même façon, il est difficile de partager vos changements en cours avec quelqu'un d'autre. Une des « bonnes pratiques » du monde du développement logiciel est de permettre à vos pairs de passer votre travail en revue au fur et à mesure. Si personne n'a accès à vos propagations intermédiaires, vous vous coupez d'éventuelles critiques et risquez de partir dans une mauvaise direction pendant des semaines avant que quelqu'un ne s'en aperçoive. Enfin, quand vous en aurez fini avec tous vos changements, vous pourriez avoir du mal à fusionner votre travail avec le code du reste de l'équipe. Sally (et les autres) peuvent avoir apporté de nombreux autres changements au dépôt, changements qui seront difficiles à incorporer dans votre copie de travail, notamment si vous lancez **svn update** après des semaines d'isolation.

Une solution bien meilleure est de créer votre propre branche, ou ligne de développement, dans le dépôt. Ceci vous permettra de sauvegarder fréquemment votre travail un peu boiteux sans interférer avec vos collaborateurs ; vous pourrez toutefois partager une sélection d'informations avec eux. Vous découvrirez comment tout cela fonctionne exactement au fur et à mesure de ce chapitre.

Création d'une branche

Créer une branche est très simple : il s'agit juste de faire une copie du projet dans le dépôt avec la commande **svn copy**. Subversion est capable de copier non seulement de simples fichiers, mais aussi des dossiers entiers. Dans le cas présent, vous voulez faire une copie du dossier `/calc/trunk`. Où doit résider la nouvelle copie ? Là où vous le désirez, cette décision faisant partie de la gestion du projet. Supposons que votre équipe ait pour convention de créer les branches dans la zone `/calc/branches` du dépôt et que vous vouliez nommer votre branche `ma-branche-calc`. Vous créez alors un nouveau dossier, `/calc/branches/ma-branche-calc`, qui commence ainsi sa vie en tant que copie de `/calc/trunk`.

Vous avez peut-être déjà utilisé **svn copy** pour copier un fichier vers un autre à l'intérieur d'une copie de travail. Mais il peut aussi être utilisé pour effectuer une copie « distante » entièrement à l'intérieur du dépôt. Il suffit de copier une URL vers une autre :

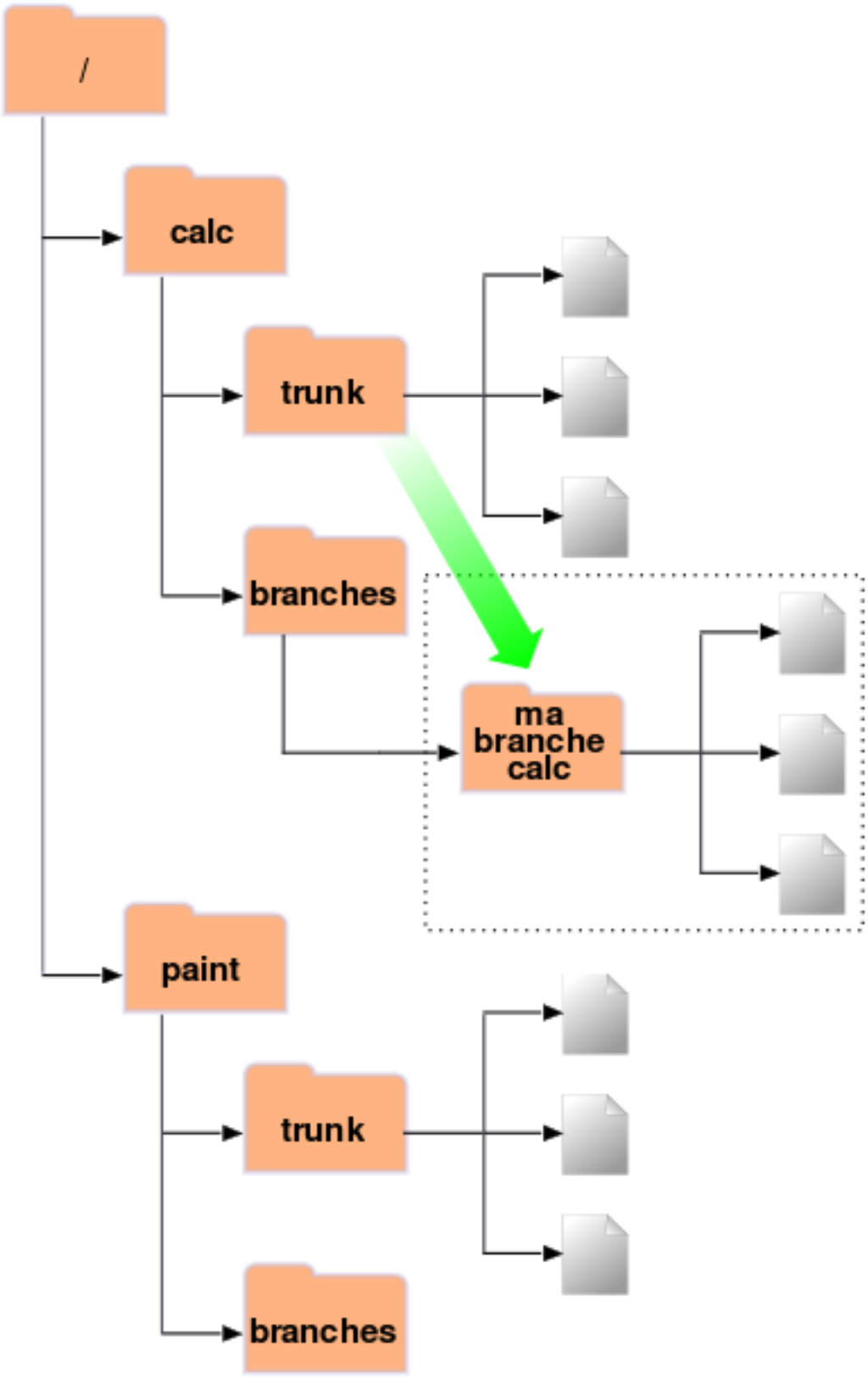
```
$ svn copy http://svn.exemple.com/depot/calc/trunk \
           http://svn.exemple.com/depot/calc/branches/ma-branche-calc \
      -m "Création d'une branche privée à partir de /calc/trunk."

Révision 341 propagée.
```

Cette commande entraîne une opération quasi-instantanée dans le dépôt, créant un nouveau dossier à la révision 341. Ce nouveau dossier est une copie de `/calc/trunk`, comme l'illustre la Figure 4.3, « Dépôt avec nouvelle copie » [1]. Bien qu'il soit aussi possible de créer une branche en utilisant **svn copy** pour dupliquer un dossier à l'intérieur de la copie de travail, cette technique n'est pas recommandée. Elle peut s'avérer assez lente, en fait ! Copier un dossier côté client est une opération linéaire en terme de durée, puisque chaque fichier et chaque dossier doit être dupliqué sur le disque local. Copier un dossier sur le serveur, par contre, est une opération dont la durée est constante et c'est ainsi que la plupart des gens créent des branches.

Figure 4.3. Dépôt avec nouvelle copie

[1] Subversion n'accepte pas les copies entre des dépôts distincts. Quand vous utilisez des URLs avec **svn copy** et **svn move**, vous ne pouvez copier que des éléments faisant partie du même dépôt.



Des copies peu coûteuses

Le dépôt Subversion a un design particulier. Quand vous copiez un dossier, il n'y a pas à s'en faire pour la taille du dépôt : en fait Subversion ne duplique aucune donnée. Au lieu de ça, il crée une nouvelle entrée de dossier qui pointe vers une arborescence *existante*. Si vous êtes un utilisateur expérimenté d'Unix, vous reconnaîtrez là le concept de lien matériel (« hard link »). Au fur et à mesure des modifications faites aux fichiers et dossiers sous le dossier copié, Subversion continue à employer ce concept de lien matériel quand il le peut. Il duplique les données seulement s'il est nécessaire de lever l'ambiguïté entre différentes versions d'objets.

C'est pourquoi vous entendrez souvent les utilisateurs de Subversion parler de « copies peu coûteuses » (*cheap copies* en anglais). Peu importe la taille du dossier, la durée de la copie est constante et très faible, tout comme l'espace disque nécessaire. En fait, cette fonctionnalité est à la base du fonctionnement des propagations dans Subversion : chaque révision est une « copie peu coûteuse » de la révision précédente, avec juste quelques éléments modifiés à l'intérieur (pour en savoir plus à ce sujet, visitez le site web de Subversion et lisez les paragraphes concernant la méthode « bubble up » dans les documents de conception de Subversion).

Bien sûr, cette mécanique interne de copie et de partage des données est transparente pour l'utilisateur, qui n'y voit que de simples copies d'arborescences. Le point essentiel ici est que les copies sont peu coûteuses, aussi bien en temps qu'en espace disque. Si vous créez une branche entièrement à l'intérieur du dépôt (en lançant **`svn copy URL1 URL2`**), c'est une opération rapide, à durée constante. Créez des branches aussi souvent que vous le souhaitez.

Travail sur votre branche

Maintenant que vous avez créé votre branche du projet, vous pouvez extraire une nouvelle copie de travail et commencer à l'utiliser :

```
$ svn checkout http://svn.exemple.com/depot/calc/branches/ma-branche-calc
A  ma-branche-calc/Makefile
A  ma-branche-calc/entier.c
A  ma-branche-calc/bouton.c
Révision 341 extraite.
```

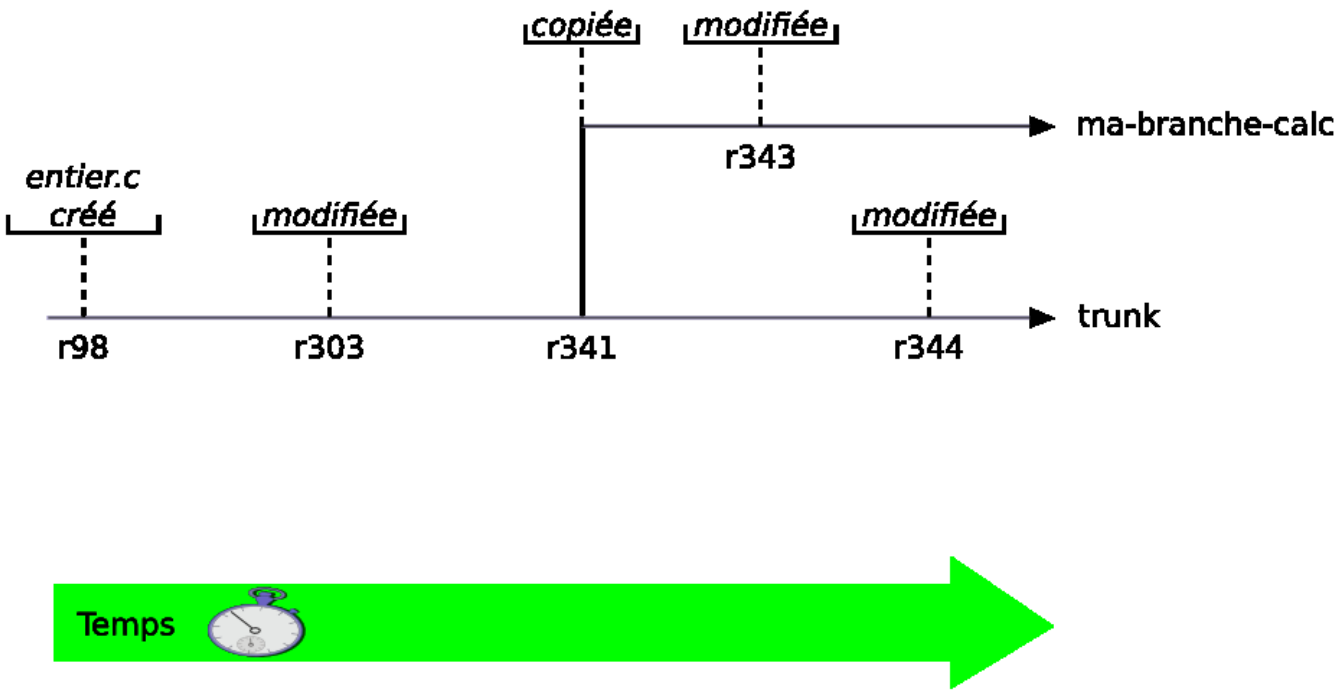
Cette copie de travail n'a rien de spéciale ; elle correspond juste à un dossier différent du dépôt. Cependant, quand vous propagerez vos modifications, Sally ne les verra pas quand elle effectuera une mise à jour, car sa copie de travail correspond à `calc/trunk` (pensez bien à lire la section intitulée « Parcours des branches » plus loin dans ce chapitre : la commande **svn switch** est une méthode alternative pour créer une copie de travail d'une branche).

Imaginons qu'une semaine passe et que les propagations suivantes ont lieu :

- Vous modifiez `/calc/branches/ma-branche-calc/bouton.c`, ce qui crée la révision 342.
- Vous modifiez `/calc/branches/ma-branche-calc/entier.c`, ce qui crée la révision 343.
- Sally modifie `/calc/trunk/entier.c`, ce qui crée la révision 344.

À présent, deux lignes de développement indépendantes (voir la Figure 4.4, « Historique des branches d'un fichier ») existent pour `entier.c`.

Figure 4.4. Historique des branches d'un fichier



Les choses deviennent intéressantes quand on regarde l'historique des modifications apportées à votre copie de `entier.c` :

```
$ pwd
/home/utilisateur/ma-branche-calc

$ svn log -v entier.c
------------------------------------------------------------------------
r343 | utilisateur | 2002-11-07 15:27:56 -0600 (jeu. 07 nov. 2002) | 2 lignes
Chemins modifiés :
   M /calc/branches/ma-branche-calc/entier.c

* entier.c:  machiné le bidule.

------------------------------------------------------------------------
r341 | utilisateur | 2002-11-03 15:27:56 -0600 (jeu. 07 nov. 2002) | 2 lignes
Chemins modifiés :
   A /calc/branches/ma-branche-calc (from /calc/trunk:340)

Création d'une branche privée à partir de /calc/trunk.

------------------------------------------------------------------------
r303 | sally | 2002-10-29 21:14:35 -0600 (mar. 29 oct. 2002) | 2 lignes
Chemins modifiés :
   M /calc/trunk/entier.c

* entier.c:  modifié une docstring.

------------------------------------------------------------------------
r98 | sally | 2002-02-22 15:35:29 -0600 (ven. 22 fev. 2002) | 2 lignes
Chemins modifiés :
  A /calc/trunk/entier.c
* entier.c:  ajout du fichier dans ce projet.

------------------------------------------------------------------------
```

Notez bien que Subversion reprend tout l'historique du `entier.c` de votre branche à travers le temps, remontant même au delà du point où il a été copié. Il liste la création d'une branche en tant qu'élément de l'historique, parce qu'`entier.c` a été copié implicitement lorsque `calc/trunk` tout entier a été copié. Maintenant regardez ce qui se passe quand Sally lance la

même commande sur sa copie du fichier :

```
$ pwd
/home/sally/calc

$ svn log -v entier.c
------------------------------------------------------------------------
r344 | sally | 2002-11-07 15:27:56 -0600 (jeu. 07 nov. 2002) | 2 lignes
 Chemins modifiés :
    M /calc/trunk/entier.c

 * entier.c:  corrigé un ensemble de coquilles.

 ------------------------------------------------------------------------
 r303 | sally | 2002-10-29 21:14:35 -0600 (mar. 29 oct. 2002) | 2 lignes
 Chemins modifiés :
    M /calc/trunk/entier.c

 * entier.c:  modifié une docstring.

 ------------------------------------------------------------------------
 r98 | sally | 2002-02-22 15:35:29 -0600 (ven. 22 fev. 2002) | 2 lignes
 Chemins modifiés :
   A /calc/trunk/entier.c

* entier.c:  ajout du fichier dans ce projet.

------------------------------------------------------------------------
```

Sally voit la modification due à sa propre révision 344, mais pas le changement que vous avez effectué dans la révision 343. Pour Subversion, ces deux propagations ont touché des fichiers différents dans des dossiers distincts. Néanmoins, Subversion indique bien que les deux fichiers partagent une histoire commune. Avant que la copie de branche n'ait été faite en révision 341, les fichiers ne faisaient qu'un. C'est pourquoi Sally et vous voyez tous les deux les modifications apportées en révisions 303 et 98.

Gestion des branches par Subversion : notions clé

Il y a deux leçons importantes à retenir de ce paragraphe. Premièrement, Subversion n'a pas de notion interne de branche — il sait seulement faire des copies. Quand vous copiez un dossier, le dossier qui en résulte n'est une « branche » que parce que *vous* le considérez comme tel. Vous aurez beau envisager ce dossier différemment ou le traiter différemment, pour Subversion c'est juste un dossier ordinaire auquel sont associées des informations extra-historiques.

Deuxièmement, à cause de ce mécanisme de copie, les branches de Subversion existent en tant que *dossiers classiques du système de fichiers* du dépôt. En cela, Subversion diffère des autres systèmes de gestion de versions, où les branches sont définies par l'ajout d'« étiquettes » extra-dimensionnelles à des groupes de fichiers. L'emplacement du dossier de votre branche importe peu à Subversion. La plupart des équipes ont pour convention de placer toutes les branches dans un dossier `/branches`, mais vous êtes libre d'inventer la convention qui vous plaît.

Fusions : pratiques de base

Désormais, Sally et vous travaillez sur des branches parallèles du projet : vous travaillez sur une branche privée et Sally travaille sur le *tronc*, la branche de développement principale.

Pour les projets qui ont un grand nombre de contributeurs, il est d'usage que la plupart des gens aient des copies de travail du tronc. Dès que quelqu'un doit faire des modifications de longue haleine, susceptibles de perturber le tronc, une procédure standard est qu'il crée une branche privée et qu'il y propage les modifications jusqu'à ce que tout le travail soit terminé.

Bref, la bonne nouvelle est que Sally et vous n'empiétez pas l'un sur l'autre. La mauvaise nouvelle est qu'il est très facile de *dériver* chacun de son côté. Rappelez-vous qu'un des problèmes lié à la stratégie d'« isolement » est que lorsque vous en aurez fini avec votre branche, il risque d'être quasi impossible de refusionner vos modifications dans le tronc sans avoir à faire face à un grand nombre de conflits.

À la place, Sally et vous pourriez continuer de partager vos changements au fur et à mesure de votre travail. C'est à vous de

décider quelles modifications valent la peine d'être partagées ; Subversion vous offre la possibilité de « copier » sélectivement des modifications entre les branches. Et quand vous aurez tout fini dans votre branche, l'ensemble de vos modifications pourra être recopié en entier vers le tronc. Dans la terminologie Subversion, l'action générale de réplication des modifications d'une branche vers une autre s'appelle la *fusion* et elle s'effectue à l'aide de plusieurs exécutions de la commande **svn merge**.

Dans les exemples qui suivent, nous supposerons que le client et le serveur Subversion sont tous deux en version 1.5 (ou plus récente). Si l'un ou l'autre sont en version plus ancienne, les choses sont plus compliquées : le système ne gére pas les changements de façon automatique et vous devrez utiliser des méthodes manuelles pénibles pour obtenir des résultats similaires. Vous devrez en effet toujours utiliser la syntaxe détaillée de la fusion spécifiant l'éventail des révisions à répliquer (voir la section intitulée « Syntaxe de la fusion : pour tout vous dire » plus loin dans ce chapitre) et penser à garder trace de ce qui a déjà été fusionné et de ce qui ne l'a pas encore été. Pour cette raison, nous recommandons *fortement* de vous assurer que client et serveur sont au moins en version 1.5.

Ensembles de modifications

Avant que nous n'allions plus loin, nous devons vous avertir que les pages suivantes contiennent de nombreuses discussions portant sur les « modifications ». Beaucoup de gens ayant de l'expérience dans les systèmes de gestion de versions utilisent le terme « modifications » et le terme « ensemble de modifications » de façon interchangeable et nous allons donc clarifier ce que Subversion entend par *ensemble de modifications*.

Chacun semble avoir sa propre définition, variant légèrement, d'un ensemble de modifications, ou tout du moins a une attente différente quant à leur traitement par le système de gestion de versions. En ce qui nous concerne, disons qu'un ensemble de modifications n'est qu'un simple regroupement de modifications identifié par un nom unique. Les modifications peuvent inclure des changements textuels du contenu des fichiers, des modifications de l'arborescence ou des ajustements portant sur les méta-données. En langage plus courant, un ensemble de modifications n'est qu'un correctif avec un nom auquel vous pouvez vous référer.

Dans Subversion, un numéro de révision globale N désigne une arborescence dans le dépôt : c'est ce à quoi le dépôt ressemblait après la N-ième propagation. C'est aussi le nom d'un ensemble de modifications implicite : si vous comparez l'arborescence N avec l'arborescence N#1, vous pouvez en déduire exactement le correctif qui a été propagé. Pour cette raison, il est facile de se représenter une révision N non seulement comme une arborescence, mais aussi comme un ensemble de modifications. Si vous utilisez un système de gestion des incidents pour gérer vos bogues, vous pouvez utiliser les numéros de révision pour vous référer à des correctifs particuliers permettant de résoudre des bogues — par exemple, « cet incident a été corrigé par r9238 ». Quelqu'un peut alors lancer **`svn log -r 9238`** pour obtenir le détail des modifications qui ont corrigé le bogue et lancer **`svn diff -c 9238`** pour voir le correctif lui-même. De plus (comme nous le verrons bientôt), la commande **svn merge** de Subversion est capable d'utiliser les numéros de révision. Vous pouvez fusionner des listes de modifications spécifiques d'une branche à une autre en les nommant dans les paramètres de la fusion : donner comme argument **`-c 9238`** à **svn merge** fusionne la liste de modifications r9238 avec votre copie de travail.

Comment garder une branche synchronisée

Continuons avec notre exemple précédent et imaginons qu'une semaine a passé depuis que vous avez commencé à travailler sur votre branche privée. Votre nouvelle fonctionnalité n'est pas encore terminée, mais en même temps vous savez que d'autres personnes de votre équipe ont continué à faire des modifications importantes sur le `/trunk` du projet. Vous avez intérêt à recopier ces modifications dans votre propre branche, juste pour vous assurer qu'elles se combinent bien avec vos propres modifications. En fait, c'est là une bonne pratique : synchroniser fréquemment votre branche avec la ligne de développement principale permet d'éviter les conflits « surprise » le jour où vous reversez vos modifications dans le tronc.

Subversion connaît l'historique de votre branche et sait à quel moment elle s'est séparée du tronc. Afin de récupérer les modifications du tronc les plus récentes et les plus importantes, assurez-vous en premier lieu que votre copie de travail est « propre », c'est-à-dire que **svn status** ne liste aucune modification locale. Puis lancez juste :

```
$ pwd
/home/user/ma-branche-calc

$ svn merge http://svn.exemple.com/depot/calc/trunk
--- Fusion de r345 à r356 dans '.':
U    bouton.c
U    entier.c
```

La syntaxe de base, **`svn merge` *`URL`***, indique à Subversion qu'il doit fusionner toutes les modifications récentes depuis l'URL vers le répertoire de travail actuel (qui est bien souvent la racine de votre copie de travail). Après l'exécution de la commande de l'exemple précédent, la copie de travail de votre branche contient de nouvelles modifications locales, ces changements étant des duplications de toutes les modifications qui ont eu lieu sur le tronc depuis que vous avez créé votre branche :

```
$ svn status
 M     .
M      bouton.c
M      entier.c
```

Une fois rendu là, le plus sage est d'examiner attentivement les modifications avec **svn diff** et ensuite de compiler et de tester votre branche. Notez que le répertoire de travail actuel (« `.` ») a aussi été modifié ; **svn diff** indique que sa propriété `svn:mergeinfo` a été créée ou modifiée. Ceci est une méta-information importante liée à la fusion, à laquelle vous ne devriez *pas* toucher, puisqu'elle sera nécessaire aux futures commandes **svn merge** (nous en apprendrons plus sur cette métadonnée plus loin dans ce chapitre).

Après avoir effectué la fusion, vous aurez peut-être aussi besoin de résoudre des conflits (comme vous le faites pour **svn update**) ou éventuellement d'effectuer de petites modifications afin que les choses fonctionnent correctement (souvenez-vous, ce n'est pas parce qu'il n'y a pas de conflits *syntaxiques* qu'il n'y a pas de conflits *sémantiques* !). Si vous rencontrez de graves difficultés, vous pouvez toujours annuler les modifications locales en lançant **`svn revert . -R`** (qui annule toutes les modifications locales) et entamer avec vos collègues une longue discussion sur le thème « Qu'est-ce qui se passe ? ». Par contre, si les choses se passent bien, vous pouvez propager ces modifications dans le dépôt :

```
$ svn commit -m "Fusionné les dernières modifications de trunk avec ma-branche-calc"
Envoi          .
Envoi          bouton.c
Envoi          entier.c
Transmission des données ..
Révision 357 propagée.
```

À présent, votre branche privée est désormais « synchro » avec le tronc, vous pouvez donc vous détendre, sachant qu'en continuant votre travail en isolation, vous ne dériverez pas trop loin de ce que les autres font.

Pourquoi ne pas utiliser des correctifs de type patch à la place ?

Une question vous trotte peut-être dans la tête, surtout si vous êtes un utilisateur d'Unix : pourquoi s'embêter à utiliser **svn merge** ? Pourquoi ne pas tout simplement utiliser la commande **patch** du système d'exploitation pour accomplir la même tâche ? Par exemple :

```
$ cd ma-branche-calc
$ svn diff -r 341:HEAD http://svn.exemple.com/depot/calc/tronc > fichierpatch
$ patch -p0 < fichierpatch
Patching file entier.c using Plan A...
Hunk #1 succeeded at 147.
Hunk #2 succeeded at 164.
Hunk #3 succeeded at 241.
Hunk #4 succeeded at 249.
done
```

Dans cet exemple, il n'y a pas vraiment de grande différence. Mais **svn merge** possède des fonctionnalités spécifiques qui surpassent le programme **patch**. Le format de fichier utilisé par **patch** est assez limité ; il ne sait manipuler que les contenus de fichier. Il n'y a pas moyen de représenter des changements dans *l'arborescence*, tels que l'ajout, la suppression ou le renommage de fichiers ou de dossiers. Le programme **patch** n'est pas non plus capable de prendre en compte des modifications de propriétés. Si Sally avait, par exemple, ajouté un nouveau dossier, **svn diff** ne l'aurait pas mentionné du tout en sortie. Le résultat de **svn diff** n'est qu'au format « patch », il y a donc des concepts qu'il ne peut tout simplement pas exprimer.

La commande **svn merge**, par contre, peut gérer des modifications dans l'arborescence et dans les propriétés en les appliquant directement à votre copie de travail. Et, ce qui est encore plus important, cette commande enregistre les modifications qui ont été dupliquées vers votre branche de telle sorte que Subversion sait exactement quelles modifications existent dans chaque endroit (voir la section intitulée « Mergeinfo et aperçus »). C'est une fonctionnalité cruciale qui rend la gestion des branches utilisable ; sans elle, les utilisateurs seraient forcés de conserver des notes manuelles relatant quelles listes de modifications ont été fusionnées (et lesquelles ne l'ont pas été).

Supposons qu'une autre semaine s'est écoulée. Vous avez propagé des modifications supplémentaires dans votre branche et vos camarades ont également continué à améliorer le tronc. Une fois encore, vous aimeriez répercuter les dernières modifications du tronc vers votre branche et ainsi être synchro. Lancez juste la même commande **svn merge** à nouveau !

```
$ svn merge http://svn.exemple.com/depot/calc/trunk
--- Fusion de r357 à r380 dans '.':
U    bouton.c
U    Makefile
A    LISEZMOI
```

Subversion sait quelles sont les modifications du tronc que vous avez déjà répercutées vers votre branche, il ne répercute donc que les modifications que vous n'avez pas encore. Une fois de plus, vous devrez compiler, tester et propager les modifications locales à votre branche.

Cependant, que se passe-t-il quand vous finissez enfin votre travail ? Votre nouvelle fonctionnalité est terminée et vous êtes prêt à fusionner les changements de votre branche avec le tronc (pour que votre équipe puisse bénéficier du fruit de votre travail). La procédure est simple. Premièrement, synchronisez à nouveau votre branche avec le tronc, comme vous le faites depuis le début :

```
$ svn merge http://svn.exemple.com/repos/calc/tronc
--- Fusion de r381 à r385 dans '.':
U    bouton.c
U    LISEZMOI

$ # compiler, tester, ...

$ svn commit -m "Fusion finale des modifications du tronc dans ma-branche-calc."
Envoi          .
Envoi          bouton.c
Envoi          LISEZMOI
Transmission des données ..
Révision 390 propagée.
```

À présent, utilisez **svn merge** pour répercuter les modifications de votre branche sur le tronc. Vous avez alors besoin d'une copie de travail de `/trunk` qui soit à jour. Vous pouvez vous la procurer soit en effectuant un **svn checkout**, soit en reprenant une vieille copie de travail du tronc, soit en utilisant **svn switch** (voir la section intitulée « Parcours des branches »). Quelle que soit la manière dont vous obtenez une copie de travail, souvenez-vous qu'une bonne méthode est d'effectuer la fusion dans une copie de travail qui n'a *pas* été modifiée localement et qui a été mise à jour récemment (en d'autres termes, qui n'est pas un mélange de révisions locales). Si votre copie de travail n'est pas « propre » comme expliqué à l'instant, vous risquez de rencontrer des problèmes facilement évitables liés à des conflits et **svn merge** renverra probablement une erreur en retour.

Une fois que vous avez une copie de travail propre du tronc, vous êtes prêt pour y fusionner votre branche :

```
$ pwd
/home/user/calc-tronc

$ svn update  # (s'assurer que la copie de travail est à jour)
À la révision 390.

$ svn merge --reintegrate http://svn.exemple.com/depot/calc/branches/ma-branche-calc
--- Fusionne toutes les modifications non fusionnées des URLs sources dans '.':
U    bouton.c
```

```
U    entier.c
U    Makefile
U   .

$ # compiler, tester, vérifier, ...

$ svn commit -m "Fusionner ma-branche-calc dans le tronc !"
Envoi          .
Envoi          bouton.c
Envoi          entier.c
Envoi          Makefile
Transmission des données ..
Révision 391 propagée.
```

Félicitations, votre branche a maintenant réintégré la ligne de développement principale. Notez bien l'utilisation de l'option `--reintegrate` à cette occasion. L'option est essentielle pour répercuter les modifications d'une branche sur sa ligne de développement d'origine, ne l'oubliez pas ! Elle est nécessaire car ce type de « réintégration » est un type de tâche différent de ce que vous avez fait jusqu'à présent. Précédemment, nous demandions à **svn merge** de faire la liste des modifications ayant eu lieu dans une ligne de développement (le tronc) et de les dupliquer vers une autre (votre branche). C'est assez simple à réaliser et à chaque fois Subversion sait reprendre là où il s'était arrêté. Dans nos exemples précédents, vous pouvez voir qu'il fusionne en premier les modifications 345:356 du tronc vers la branche ; ensuite il continue en fusionnant le groupe contigu immédiatement suivant, 356:380. Quand il effectue la synchronisation finale, il fusionne le groupe 380:385.

Cependant, lors de la réintégration d'une branche dans le tronc, la logique sous-jacente est assez différente. Votre branche dédiée est à présent un amoncellement de modifications provenant à la fois du tronc et de votre branche privée et il n'y a donc pas de groupe de révisions contigu à recopier. En spécifiant l'option `--reintegrate`, vous demandez explicitement à Subversion de ne recopier *que* les modifications spécifiques à votre branche (et en fait il le fait en comparant l'arborescence la plus récente du tronc avec l'arborescence la plus récente de la branche : la différence qui en résulte constitue exactement les modifications de votre branche !).

Votre branche privée ayant réintégré le tronc, vous voudrez peut-être la supprimer du dépôt :

```
$ svn delete http://svn.exemple.com/depot/calc/branches/ma-branche-calc \
      -m "Supprime ma-branche-calc."
Révision 392 propagée.
```

Mais attendez ! L'historique de votre branche ne possède-t-il pas une certaine valeur ? Et si un beau jour quelqu'un voulait auditer l'évolution de votre fonctionnalité et examiner toutes les modifications de votre branche ? Pas la peine de s'inquiéter. Souvenez-vous que même si votre branche n'est plus visible dans le dossier `/branches`, son existence demeure une partie immuable de l'historique du dépôt. Une simple commande **svn log** appliquée à l'URL `/branches` vous renverra l'historique complet de votre branche. Votre branche pourrait même ressusciter un jour ou l'autre, si vous le désirez (voir la section intitulée « Résurrection des éléments effacés »).

Dans Subversion 1.5, une fois que la fusion d'une branche vers le tronc a été faite avec l'option `--reintegrate`, la branche n'est plus utilisable. Elle ne peut absorber correctement de nouvelles modifications du tronc, ni être réintégrée à nouveau proprement dans le tronc. Pour cette raison, si vous voulez continuer à travailler sur la branche de votre fonctionnalité, nous vous recommandons de la détruire et de la recréer depuis le tronc :

```
$ svn delete http://svn.exemple.com/depot/calc/branches/ma-branche-calc \
      -m "Supprime ma-branche-calc."
Révision 392 propagée.

$ svn copy http://svn.exemple.com/depot/calc/trunk \
           http://svn.exemple.com/depot/calc/branches/nouvelle-branche
      -m "Crée une nouvelle branche a partir du tronc."
Révision 393 propagée.

$ cd ma-branche-calc

$ svn switch http://svn.exemple.com/depot/calc/branches/nouvelle-branche
À la révision 393.
```

La dernière commande de l'exemple précédent, **svn switch**, est une façon de mettre à jour une copie de travail existante afin qu'elle pointe vers un autre dossier du dépôt. Nous en parlerons plus en détail dans la section intitulée « Parcours des branches ».

Mergeinfo et aperçus

Le mécanisme de base que Subversion utilise pour gérer les ensembles de modifications, c'est-à-dire quelles modifications ont été fusionnées dans quelles branches, est l'enregistrement de données dans les propriétés. Plus précisément, les informations de fusion sont conservées dans la propriété `svn:mergeinfo` qui est associée aux fichiers et aux dossiers (si les propriétés de Subversion ne vous sont pas familières, c'est le moment de lire la section intitulée « Propriétés »).

Vous pouvez examiner cette propriété comme n'importe quelle autre :

```
$ cd ma-branche-calc
$ svn propget svn:mergeinfo .
/trunk:341-390
```

Il est *déconseillé* de changer soi-même la valeur de cette propriété, à moins de savoir vraiment ce que vous faites. Cette propriété est manipulée automatiquement par Subversion à chaque fois que vous lancez **svn merge**. Sa valeur indique quelles modifications (pour un chemin donné) ont été recopiées dans le dossier en question. Dans le cas présent, le chemin est `/trunk` et le dossier qui a reçu les modifications spécifiées est `/branches/ma-branche-calc`.

Il existe également une sous-commande, **svn mergeinfo**, qui peut être utile pour voir non seulement quels ensembles de modifications un dossier a absorbés, mais aussi quels ensembles de modifications il est encore susceptible de recevoir. Ceci donne une sorte d'aperçu du prochain ensemble de modifications que **svn merge** recopiera vers votre branche.

```
$ cd ma-branche-calc

# Quelles modifications ont déjà été fusionnées du tronc vers la branche ?
$ svn mergeinfo http://svn.exemple.com/depot/calc/trunk
r341
r342
r343
…
r388
r389
r390

# Quelles modifications sont encore susceptibles d'être fusionnées du tronc vers la
branche ?
$ svn mergeinfo http://svn.exemple.com/depot/calc/trunk --show-revs eligible
r391
r392
r393
r394
r395
```

La commande **svn mergeinfo** prend en paramètres une URL « source » (d'où les modifications viennent) et une URL « cible » optionnelle (vers laquelle les modifications sont fusionnées). Si aucune URL cible n'est fournie, elle suppose que le dossier actuel est la cible. Dans l'exemple précédent, puisque nous interrogeons la copie de travail de notre branche, la commande suppose que ce qui nous intéresse est de recevoir les modifications de l'URL spécifiée du tronc vers `/branches/ma-branche-calc`.

Une autre manière d'obtenir un aperçu plus précis d'une opération de fusion est d'utiliser l'option `--dry-run` :

```
$ svn merge http://svn.exemple.com/depot/calc/trunk --dry-run
U    entier.c

$ svn status
#  rien ne s'affiche, la copie de travail n'a pas changé.
```

L'option `--dry-run` n'effectue en fait pas de modification locale sur la copie de travail. Elle ne fait qu'indiquer les codes d'état qui *seraient* affichés par une vraie fusion. Ceci permet d'obtenir un « aperçu général » d'une fusion potentielle, pour les fois où **svn diff** renvoie trop de détails.

Après avoir effectué une opération de fusion, mais avant d'en avoir propagé les résultats, vous pouvez utiliser **`svn diff --depth=empty`** ***`/chemin/vers/la/cible/de/la/fusion`*** pour visualiser uniquement les modifications apportées à la cible immédiate de votre fusion. Si la cible de la fusion est un dossier, seules les différences de propriétés sont alors affichées. C'est un moyen très pratique pour voir les modifications de la propriété `svn:mergeinfo` enregistrées par l'opération de fusion, qui vous rappellera ce que vous venez juste de fusionner.

Bien sûr, la meilleure façon d'avoir un aperçu d'une opération de fusion est tout simplement de la réaliser ! Souvenez-vous que lancer **svn merge** n'est pas une opération risquée en soi (à moins que vous ayez effectué des modifications locales dans votre copie de travail, mais nous avons déjà souligné que vous ne devriez pas faire de fusion dans un tel environnement). Si les résultats de la fusion ne vous plaisent pas, lancez juste **`svn revert . -R`** pour ôter les modifications de votre copie de travail et réessayez la commande avec des options différentes. La fusion n'est définitive qu'une fois que vous en avez propagé les résultats.

Bien qu'il soit parfaitement légitime de lancer **svn merge** et **svn revert** à plusieurs reprises pour préparer la fusion, vous risquez de rencontrer quelques obstacles (facilement surmontables). Par exemple, si l'opération de fusion ajoute un nouveau fichier (ou plus exactement programme son ajout), **svn revert** ne supprime pas le fichier ; il va juste déprogrammer l'ajout. Vous vous retrouvez avec un fichier non suivi en versions. Si ensuite vous tentez à nouveau de lancer la fusion, il risque d'y avoir des conflits dus au fichier non suivi en versions resté « en travers du chemin ». La Solution ? Après avoir effectué un retour en arrière à l'aide de **svn revert**, pensez à nettoyer la copie de travail et supprimer les fichiers et dossiers non suivis en versions. Il faut que le résultat de **svn status** soit le plus propre possible, et dans l'idéal vide.

Retour en arrière sur des modifications

Un usage très répandu de **svn merge** est le retour en arrière sur une modification qui a déjà été propagée. Supposons que vous travaillez tranquillement sur une copie de travail de `/calc/trunk` et que vous découvrez tout à coup que la modification faite il y a longtemps lors de la révision 303, qui affectait `entier.c`, est complètement incorrecte. Elle n'aurait jamais du être propagée. Vous pouvez utiliser **svn merge** pour revenir en arrière sur cette modification dans votre copie de travail, puis propager la modification locale au dépôt. Il suffit juste de spécifier une différence *inversée* (en indiquant soit `--revision 303:302`, soit `--change -303`, les deux se valent).

```
$ svn merge -c -303 http://svn.exemple.com/depot/calc/trunk
--- Fusion inverse de r303 dans 'entier.c':
U    entier.c

$ svn status
 M     .
M      entier.c

$ svn diff
…
# vérifier que la modification est supprimée
…

$ svn commit -m "Retour en arrière sur la modification propagée en r303."
Envoi          entier.c
Transmission des données .
Révision 350 propagée.
```

Comme nous l'avons signalé précédemment, une façon de se représenter une révision du dépôt est de la considérer comme un ensemble de modifications spécifique. En utilisant l'option `-r`, vous pouvez demander à **svn merge** d'appliquer un ensemble

de modifications, ou tout un groupe d'ensembles de modifications, à votre copie de travail. Dans le cas présent, pour revenir en arrière, nous demandons à **svn merge** d'appliquer *dans le sens inverse* l'ensemble de modifications n°303 à notre copie de travail.

Gardez à l'esprit que revenir en arrière sur une modification de cette façon est similaire à toute autre opération **svn merge**, vous devez donc ensuite utiliser **svn status** et **svn diff** pour vous assurer que votre travail est dans l'état que vous voulez, puis utiliser **svn commit** pour propager la version finale au dépôt. Après la propagation, cet ensemble de modifications particulier n'est plus présent dans la révision HEAD.

À nouveau vous vous dites : bon, ceci n'a pas vraiment annulé la propagation, n'est-ce pas ? La modification existe toujours en révision 303. Si quelqu'un extrait une version du projet calc entre les révisions 303 et 349, il verra toujours la mauvaise modification, non ?

Oui, c'est vrai. Quand nous parlons de « supprimer » une modification, il s'agit de la supprimer de la révision HEAD. La modification originale existe toujours dans l'historique du dépôt. Dans la plupart des situations, c'est suffisant. La plupart des gens ne s'intéressent d'ailleurs qu'à la révision HEAD du projet. Il y a des cas particuliers, cependant, où l'on voudra vraiment détruire toute preuve de la propagation (quelqu'un a peut-être accidentellement propagé un document confidentiel). Cela ne s'avère pas si facile, parce que Subversion a été conçu délibérément pour ne jamais perdre d'information. Les révisions sont des arborescences immuables qui sont empilées les unes par dessus les autres. Supprimer une révision de l'historique créerait un effet domino, engendrant le chaos dans les révisions ultérieures et invalidant potentiellement toutes les copies de travail [2].

Résurrection des éléments effacés

Ce qu'il y a de formidable dans les systèmes de gestion de versions, c'est que les informations ne sont jamais perdues. Même si vous effacez un fichier ou un dossier, s'il disparaît bien de la révision HEAD, l'objet existe toujours dans les révisions précédentes. Une des questions les plus courantes que posent les nouveaux utilisateurs est : « Comment est-ce que je récupère mon ancien fichier ou dossier ? »

La première étape est de définir exactement *quel* élément vous essayez de ressusciter. Voici une métaphore utile : vous pouvez imaginer votre objet dans le dépôt comme existant dans une sorte de système à deux dimensions. La première coordonnée est une révision correspondant à une arborescence particulière ; la deuxième coordonnée est un chemin à l'intérieur de cette arborescence. Ainsi, toute version d'un fichier ou d'un dossier peut être définie par une paire de coordonnées qui lui est propre (souvenez-vous de la syntaxe des « piquets de révisions » : machin.c@224, mentionnée dans la section intitulée « Piquets de révisions et révisions opérationnelles »).

Tout d'abord, vous allez peut-être avoir besoin de **svn log** pour identifier précisément les coordonnées du fichier ou dossier que vous voulez ressusciter. À cette fin, une bonne stratégie est de lancer **`svn log --verbose`** dans un dossier qui contenait votre élément effacé. L'option `--verbose` (`-v`) renvoie la liste de tous les éléments modifiés par chaque révision ; il vous suffit alors de trouver la révision dans laquelle vous avez effacé le fichier ou le dossier en question. Vous pouvez accomplir cette recherche soit visuellement soit en utilisant un autre outil pour examiner le résultat de la commande **svn log** (via **grep** ou peut-être via une recherche incrémentale dans un éditeur).

```
$ cd dossier-parent
$ svn log -v
…
------------------------------------------------------------------------
r808 | paul | 2003-12-26 14:29:40 -0600 (ven. 26 déc 2003) | 3 lignes
Chemins modifiés :
   D /calc/trunk/reel.c
   M /calc/trunk/entier.c

Ajouté les fonctions des transformées de fourier dans entier.c.
Supprimé reel.c car code désormais dans double.c.
…
```

Dans l'exemple ci-dessus, nous supposons que vous recherchez un fichier effacé nommé `reel.c`. En examinant le journal du dossier parent, vous avez découvert que ce fichier a été effacé en révision 808. La dernière version du fichier à avoir existé était donc dans la révision précédant celle-ci. Conclusion : vous voulez ressusciter le chemin `/calc/trunk/reel.c` tel qu'il était en révision 807.

[2] Le projet Subversion prévoit néanmoins d'implémenter, un jour, une commande qui accomplirait la tâche de supprimer des informations de façon permanente. En attendant, en guise de palliatif, voir la section intitulée « svndumpfilter ».

Voilà, c'était la partie difficile : la recherche. Maintenant que vous savez ce que vous voulez récupérer, deux options s'offrent à vous.

Une possibilité serait d'utiliser **svn merge** pour appliquer la révision 808 « à l'envers » (nous avons déjà parlé de comment revenir sur des modifications dans la section intitulée « Retour en arrière sur des modifications »). Ceci aurait pour effet de ré-ajouter `reel.c` en tant que modification locale. Le fichier serait alors programmé pour être ajouté et après la propagation le fichier existerait à nouveau dans `HEAD`.

Cependant, dans cet exemple particulier, ce n'est probablement pas la meilleure stratégie. Appliquer la révision 808 à l'envers programmerait non seulement l'ajout de `reel.c`, mais le message de propagation indique qu'il reviendrait aussi sur certaines modifications de `entier.c`, ce que vous ne voulez pas. Vous pourriez certainement fusionner à l'envers la révision 808 et ensuite revenir sur les modifications locales faites dans `entier.c`, mais cette technique fonctionne mal à plus grande échelle. Que dire, si 90 fichiers avaient été modifiés en révision 808 ?

Une seconde stratégie plus ciblée est de ne pas utiliser **svn merge** du tout, mais plutôt d'utiliser la commande **svn copy**. Copiez juste la révision et le chemin exacts (vos deux « coordonnées ») du dépôt vers votre copie de travail :

```
$ svn copy http://svn.exemple.com/depot/calc/trunk/reel.c@807 ./reel.c

$ svn status
A  +   reel.c

$ svn commit -m "Ressuscité reel.c à partir de la révision 807, /calc/trunk/reel.c."
Ajout         reel.c
Transmission des données .
Révision 1390 propagée.
```

Le symbole plus dans le résultat de la commande **svn status** indique que l'élément n'est pas simplement programmé pour ajout, mais programmé pour ajout « avec son historique ». Subversion se souviendra d'où il a été copié. Dans le futur, lancer **svn log** sur ce fichier parcourra tout son historique en passant par la résurrection du fichier ainsi que tout ce qui précédait la révision 807. En d'autres termes, ce nouveau `reel.c` n'est pas vraiment nouveau ; c'est un descendant direct du fichier original qui avait été effacé. En général c'est une bonne chose, dont l'utilité est avérée. Si cependant vous vouliez récupérer le fichier *sans* conserver de lien historique avec l'ancien fichier, la technique suivante fonctionnerait tout aussi bien :

```
$ svn cat http://svn.exemple.com/depot/calc/trunk/reel.c@807 > ./reel.c

$ svn add reel.c
A         reel.c

$ svn commit -m "Recréé reel.c à partir de la révision 807."
Ajout         reel.c
Transmission des données .
Révision 1390 propagée.
```

Bien que notre exemple ne porte que sur la résurrection d'un fichier, remarquez que ces mêmes techniques fonctionnent tout aussi bien pour ressusciter des dossiers effacés. Remarquez aussi que cette résurrection ne doit pas forcément avoir lieu dans votre copie de travail ; elle peut avoir lieu entièrement dans le dépôt :

```
$ svn copy http://svn.exemple.com/depot/calc/trunk/reel.c@807 \
           http://svn.exemple.com/depot/calc/trunk/ \
      -m "Ressuscite reel.c dans la révision 807."
Révision 1390 propagée.

$ svn update
A    reel.c
À la révision 1390.
```

Fusions : pratiques avancées

Ici finit la magie automatisée. Tôt ou tard, une fois que vous maîtrisez bien la gestion des branches et les fusions, vous allez vous retrouver à demander à Subversion de fusionner des modifications spécifiques d'un endroit à un autre. Pour faire cela, vous allez devoir commencer à passer des paramètres plus compliqués à **svn merge**. Le paragraphe suivant décrit la syntaxe complète de la commande et aborde un certain nombre de scénarios d'utilisation courants.

Sélection à la main

De la même façon que le terme « ensemble de modifications » est utilisé couramment dans les systèmes de gestion de versions, le terme « sélectionner à la main » pourrait l'être aussi. Il désigne l'action de choisir *une* liste de modifications particulière au sein d'une branche et de la recopier dans une autre. Sélectionner à la main peut aussi faire référence à l'action de dupliquer un ensemble de modifications (pas nécessairement contiguës !) d'une branche vers une autre. Ceci est en opposition avec des scénarios de fusion plus courants, où l'ensemble de révisions contiguës « suivant » est dupliqué automatiquement.

Pourquoi voudrait-on ne recopier qu'une modification unique ? Cela arrive plus souvent qu'on ne croit. Par exemple, remontons le temps et imaginons que vous n'avez pas encore réintégré votre branche de développement privée dans le tronc. À la machine à café, vous apprenez par hasard que Sally a apporté une modification intéressante à `entier.c` dans le tronc. Vous reportant à l'historique des propagations du tronc, vous vous apercevez qu'elle a corrigé un bogue crucial en révision 355, qui impacte directement la fonctionnalité sur laquelle vous êtes en train de travailler. Vous n'êtes peut-être pas encore prêt à fusionner toutes les modifications du tronc dans votre branche, mais vous avez certainement besoin de ce correctif pour continuer votre travail.

```
$ svn diff -c 355 http://svn.exemple.com/depot/calc/trunk

Index: entier.c
===================================================================
--- entier.c (revision 354)
+++ entier.c (revision 355)
@@ -147,7 +147,7 @@
     case 6:  sprintf(info->operating_system, "HPFS (OS/2 ou NT)"); break;
     case 7:  sprintf(info->operating_system, "Macintosh"); break;
     case 8:  sprintf(info->operating_system, "Z-System"); break;
-    case 9:  sprintf(info->operating_system, "CP/MM");
+    case 9:  sprintf(info->operating_system, "CP/M"); break;
     case 10:  sprintf(info->operating_system, "TOPS-20"); break;
     case 11:  sprintf(info->operating_system, "NTFS (Windows NT)"); break;
     case 12:  sprintf(info->operating_system, "QDOS"); break;
```

De la même façon que vous avez utilisé **svn diff** dans l'exemple précédent pour examiner la révision 355, vous pouvez passer le même paramètre à **svn merge** :

```
$ svn merge -c 355 http://svn.exemple.com/depot/calc/trunk
U    entier.c

$ svn status
M      entier.c
```

Vous pouvez à présent lancer les procédures habituelles de tests, avant de propager cette modification à votre branche. Après la propagation, Subversion marque r355 comme ayant été fusionnée dans la branche, afin qu'une future fusion « magique » synchronisant votre branche avec le tronc sache qu'elle doit sauter r355 (fusionner une même modification dans une même branche aboutit presque toujours à un conflit !).

```
$ cd ma-branche-calc

$ svn propget svn:mergeinfo .
/trunk:341-349,355

# Remarquez que r355 n'est pas listée comme "éligible" à la fusion,
# parce qu'elle a déjà été fusionnée.
$ svn mergeinfo http://svn.exemple.com/depot/calc/trunk --show-revs eligible
r350
```

```
r351
r352
r353
r354
r356
r357
r358
r359
r360

$ svn merge http://svn.example.com/depot/calc/trunk
--- Fusion de r350 à r354 dans '.':
 U   .
U    entier.c
U    Makefile
--- Fusion de r356 à r360 dans '.':
 U   .
U    entier.c
U    bouton.c
```

Ce type d'utilisation de la copie (ou rétroportage) de correctifs d'une branche à une autre est peut-être la raison la plus répandue pour sélectionner à la main des modifications ; le cas se présente très souvent, par exemple lorsqu'une équipe gère une « branche de production » du logiciel (ce thème est développé dans la section intitulée « Branches de publication »).

Avez-vous remarqué la façon dont, dans le dernier exemple, le lancement de la fusion a eu pour effet l'application de deux ensembles distincts de fusions ? La commande **svn merge** a appliqué deux correctifs indépendants à votre copie de travail, afin de sauter l'ensemble de modifications 355, que votre branche contenait déjà. Il n'y a rien de mal en soi là-dedans, sauf que ça risque de rendre plus délicate la résolution des conflits. Si le premier groupe de modifications engendre des conflits, vous *devrez* les résoudre de façon interactive pour que la procédure de fusion puisse continuer et appliquer le deuxième groupe de modifications. Si vous remettez à plus tard un conflit lié à la première vague de modifications, la commande de fusion renverra au final un message d'erreur [3].

Avertissement : bien que **svn diff** et **svn merge** soient conceptuellement très similaires, leur syntaxe est différente dans de nombreux cas. Pour plus de détails, reportez-vous au Chapitre 9, *Références complètes de Subversion* ou consultez **svn help**. Par exemple, **svn merge** demande en entrée, en tant que cible, le chemin d'une copie de travail, c'est-à-dire un emplacement où il va appliquer le correctif généré. Si la cible n'est pas spécifiée, il suppose que vous essayez d'exécuter l'une des opérations suivantes :

- Vous voulez fusionner les modifications du dossier dans votre dossier de travail en cours.
- Vous voulez fusionner les modifications d'un fichier donné dans un fichier du même nom existant dans votre dossier de travail en cours.

Si vous fusionnez un dossier et que vous n'avez pas encore spécifié de cible, **svn merge** suppose qu'il est dans la première situation et essaie d'appliquer les modifications dans votre dossier en cours. Si vous fusionnez un fichier et que ce fichier (ou un fichier du même nom) existe dans votre dossier de travail en cours, **svn merge** suppose qu'il est dans la seconde situation et essaie d'appliquer les modifications au fichier local du même nom.

Syntaxe de la fusion : pour tout vous dire

Nous venons de voir des exemples d'utilisation de la commande **svn merge** et nous allons bientôt en voir plusieurs autres. Si vous n'avez pas bien assimilé le le fonctionnement des fusions, rassurez-vous, vous n'êtes pas un cas isolé. De nombreux utilisateurs (en particulier ceux qui découvrent la gestion de versions) commencent par une phase de perplexité au sujet de la syntaxe de la commande, ainsi que quand et comment utiliser cette fonctionnalité. Mais, en fait, cette commande est bien plus simple que vous ne le pensez ! Il y a une technique très simple pour comprendre comment **svn merge** agit.

La raison principale de la confusion est le nom de la commande. Le terme « merge » indique en quelque sorte que les branches vont être combinées, ou qu'un mystérieux mélange des données va avoir lieu. Ce n'est pas le cas. Un nom plus approprié pour

[3] Ceci est au vrai pour Subversion 1.5 au moment où ces lignes sont écrites. Ce fonctionnement sera sans doute amélioré dans les versions futures de Subversion.

cette commande aurait pu être « comparer-et-appliquer », car c'est là tout ce qui se passe : deux arborescences sont comparées et les différences sont appliquées à une copie de travail.

Si vous utilisez svn merge pour effectuer de simples copies de modifications entre branches, elle fait généralement ce qu'il faut automatiquement. Par exemple, une commande telle que :

```
$ svn merge http://svn.exemple.com/depot/calc/une-branche
```

tente de dupliquer toutes les modifications faites dans `une-branche` vers votre répertoire de travail actuel, qui est sans doute une copie de travail partageant des liens historiques avec la branche. La commande est suffisamment intelligente pour ne copier que les modifications que votre copie de travail ne possède pas encore. Si vous répétez cette commande une fois par semaine, elle ne copie que les modifications « les plus récentes » qui ont eu lieu depuis la dernière fusion.

Si vous choisissez d'utiliser la commande **svn merge** dans sa version intégrale en lui fournissant les groupes de révisions spécifiques à copier, la commande prend trois paramètres :

1. une arborescence initiale (souvent appelée *côté gauche* de la comparaison) ;
2. une arborescence finale (souvent appelée *côté droit* de la comparaison) ;
3. une copie de travail qui reçoit les différences en tant que modifications locales (souvent appelée *cible* de la fusion).

Une fois ces trois paramètres fournis, les deux arborescences sont comparées et les différences sont appliquées à la copie de travail cible en tant que modifications locales. Une fois que la commande s'est terminée, le résultat est le même que si vous aviez édité les fichiers à la main ou lancé diverses commandes **svn add** ou **svn delete** vous-même. Si le résultat vous plaît, vous pouvez le propager. S'il ne vous plaît pas, vous pouvez toujours lancer **svn revert** pour revenir en arrière sur toutes les modifications.

La syntaxe de **svn merge** est assez flexible quant à la façon de spécifier les trois paramètres. Voici quelques exemples :

```
$ svn merge http://svn.exemple.com/depot/branche1@150 \
                         http://svn.exemple.com/depot/branche2@212 \
                         ma-copie-de-travail

$ svn merge -r 100:200 http://svn.exemple.com/depot/trunk ma-copie-de-travail

$ svn merge -r 100:200 http://svn.exemple.com/depot/trunk
```

La première syntaxe liste les trois arguments de façon explicite, spécifiant chaque arborescence sous la forme *URL@REV* et incluant la copie de travail cible. La deuxième syntaxe peut être utilisée comme raccourci pour les cas où vous comparez des révisions différentes de la même URL. La dernière syntaxe indique que le paramètre copie de travail est optionnel ; s'il est omis, elle utilise par défaut le répertoire en cours.

Si le premier exemple donne la syntaxe « complète » de **svn merge**, celle-ci doit être utilisée avec grande prudence ; elle peut en effet aboutir à des fusions qui n'enregistrent pas la moindre méta-donnée `svn:mergeinfo`. Le paragraphe suivant évoque ceci plus en détail.

Fusions sans mergeinfo

Subversion essaye de générer des métadonnées de fusion dès qu'il le peut, afin de rendre plus intelligentes les invocations suivantes de **svn merge**. Néanmoins, il reste des situations où les données `svn:mergeinfo` ne sont ni créées ni modifiées. Pensez à être prudent avec les scénarios suivants :

Fusionner des sources sans lien de parenté
: Si vous demandez à **svn merge** de comparer deux URLs qui n'ont pas de lien entre elles, un correctif est quand même généré et appliqué à votre copie de travail, mais aucune métadonnée de fusion n'est créée. Il n'y a pas d'historique commun aux deux sources et les futures fusions « intelligentes » dépendent de cet historique commun.

Fusionner avec des dépôts extérieurs
: Bien qu'il soit possible de lancer une commande telle que **`svn merge -r 100:200 http://svn.projetexterieur.com/depot/trunk`**, le correctif résultant ne comporte aucune métadonnée historique de fusion. À la date d'aujourd'hui, Subversion n'est pas capable de représenter des URL de dépôts différents au sein de la propriété `svn:mergeinfo`.

Utiliser `--ignore-ancestry`
: Si ce paramètre est passé à **svn merge**, il force la logique de fusion à générer les différences sans réfléchir, de la même façon que **svn diff** les génère, en ignorant toute considération historique. Nous traitons ce point plus loin dans ce chapitre dans la section intitulée « Prise en compte ou non de l'ascendance ».

Appliquer des fusions inversées à l'historique naturel de la cible
: Précédemment dans ce chapitre (dans la section intitulée « Retour en arrière sur des modifications »), nous avons vu comment utiliser **svn merge** pour appliquer un « correctif inversé », comme moyen de revenir en arrière sur des modifications. Si cette technique est utilisée pour revenir sur une modification faite à l'historique propre d'un objet (par exemple, propager r5 au tronc, puis revenir immédiatement en arrière sur r5 en utilisant **`svn merge . -c -5`**), ce type de fusion ne touche pas aux informations de fusion (`mergeinfo`) enregistrées [4].

Plus de détails sur les conflits liés aux fusions

Tout comme la commande **svn update**, **svn merge** applique les modifications à votre copie de travail. Elle est donc aussi susceptible de créer des conflits. Cependant, les conflits engendrés par **svn merge** sont parfois différents et ce paragraphe va expliquer ces différences.

Pour commencer, supposons que votre copie de travail n'a pas de modification locale en cours. Quand vous lancez **svn update** pour la mettre à jour à une révision particulière, les modifications envoyées par le serveur s'appliquent toujours « proprement » à votre copie de travail. Le serveur génère le delta en comparant deux arborescences : d'une part un instantané virtuel de votre copie de travail, d'autre part l'arborescence de la révision qui vous intéresse. Parce que la partie gauche de la comparaison est parfaitement égale à ce que vous avez déjà, il est garanti que le delta va convertir correctement votre copie de travail en l'arborescence de droite.

Mais **svn merge** ne dispose pas de telles garanties et peut être bien plus chaotique : l'utilisateur avancé peut demander au serveur de comparer *n'importe quelle* paire d'arborescences, même des arborescences n'ayant aucun rapport avec la copie de travail ! Cela laisse potentiellement beaucoup de place à l'erreur humaine. Les utilisateurs vont parfois comparer deux arborescences qui ne sont pas les bonnes, créant ainsi un delta qui ne s'appliquera pas proprement. **svn merge** fera de son mieux pour appliquer la plus grande partie possible du delta, mais ça risque d'être impossible pour certains morceaux. De la même façon que la commande Unix **patch** se plaint parfois de « morceaux ratés » (*failed hunks*), **svn merge** se plaint de « cibles manquantes omises » (*skipped targets*) :

```
$ svn merge -r 1288:1351 http://svn.exemple.com/depot/branche
U    truc.c
U    bidule.c
Cible manquante omise : 'baz.c'
U    blob.c
U    machin.h
Conflit découvert dans 'glorb.h'.
Sélectionner : (p) report, (df) diff complet, (e) édite,
        (h) aide pour plus d'options :
```

Dans l'exemple précédent, il est possible que `baz.c` existe dans les deux instantanés de la branche en question et que le delta résultant tente de modifier le contenu du fichier, mais que le fichier n'existe pas dans la copie de travail. Quoi qu'il en soit, le message « Cible manquante omise » signifie que l'utilisateur compare probablement les mauvaises arborescences ; c'est le signe classique d'une erreur de l'utilisateur. Quand ça arrive, il est facile de revenir en arrière de manière récursive sur toutes les modifications créées par la fusion (**`svn revert . --recursive`**), d'effacer tout fichier ou dossier non suivi en versions restant après le retour en arrière et de relancer **svn merge** avec des paramètres différents.

Remarquez également que l'exemple précédent indique un conflit sur `glorb.h`. Nous avons déjà mentionné que la copie de

[4] À noter qu'après être revenu en arrière sur une révision de cette manière, nous ne serions plus capables de ré-appliquer cette révision avec **`svn merge . -c 5`**, puisque les mergeinfo incluraient déjà r5 comme ayant été appliquée. Nous serions alors obligés d'utiliser l'option `--ignore-ancestry` pour forcer la commande de fusion à ignorer le contenu de mergeinfo !

travail n'a pas de modifications locales en cours : comment peut-il donc y avoir conflit ? Encore une fois, parce que l'utilisateur peut utiliser **svn merge** pour définir et appliquer n'importe quel vieux delta à la copie de travail, ce delta risque de contenir des modifications textuelles qui ne s'appliquent pas proprement à un fichier de la copie de travail, même si ce fichier n'a pas de modifications locales en cours.

Une autre petite différence entre **svn update** et **svn merge** concerne les noms des fichiers textes créés quand un conflit a lieu. Dans la section intitulée « Résoudre les conflits (fusionner des modifications) », nous avons vu qu'une mise à jour génère des fichiers appelés `nomdufichier.mine`, `nomdufichier.rOLDREV` et `nomdufichier.rNEWREV`. Cependant, quand **svn merge** cause un conflit, il crée trois fichiers appelés `nomdufichier.working`, `nomdufichier.left` et `nomdufichier.right`. Dans ce cas, les termes « left » et « right » décrivent de quel côté de la double comparaison d'arborescences le fichier venait. Dans tous les cas, ces noms de fichiers différents vous aideront à distinguer les conflits résultant d'une mise à jour de ceux résultant d'une fusion.

Blocage de modifications

Il peut parfois y avoir un ensemble de modifications particulier dont vous ne voulez pas qu'il soit fusionné automatiquement. Par exemple, peut–être que l'habitude dans votre équipe est d'effectuer tout nouveau travail de développement dans `/trunk`, mais d'être plus conservateur en ce qui concerne le rétroportage des modifications vers une branche stable que vous utilisez pour la publication. À l'extrême, vous pouvez sélectionner à la main des ensembles de modifications individuels du tronc à porter vers la branche : juste les changements qui sont suffisamment stables pour être acceptables. Peut-être que les choses ne sont pas aussi strictes après tout ; peut-être que la plupart du temps vous aimeriez juste laisser **svn merge** fusionner automatiquement la plupart des modifications du tronc vers la branche. Dans ce cas, il vous faudrait une façon de masquer quelques modifications particulières, c'est-à-dire d'empêcher qu'elles ne soient fusionnées automatiquement.

Dans Subversion 1.5, la seule manière de bloquer une liste de modifications est de faire croire au système que cette modification a déjà été fusionnée. Pour cela, il est possible de lancer la commande de fusion avec l'option `--record-only` :

```
$ cd ma-branche-calc

$ svn propget svn:mergeinfo .
/trunk:1680-3305

# Marquons l'ensemble de modifications r3328 comme déjà fusionné.
$ svn merge -c 3328 --record-only http://svn.exemple.com/depot/calc/tronc

$ svn status
M     .

$ svn propget svn:mergeinfo .
/trunk:1680-3305,3328

$ svn commit -m "Empêche r3328 d'être fusionnée vers la branche."
…
```

Cette technique fonctionne, mais elle est un petit peu dangereuse. Le problème principal est que nous ne faisons pas clairement la différence entre « J'ai déjà cette modification » et « Je n'ai pas cette modification ». En fait, nous mentons au système, en lui faisant croire que la modification a déjà été fusionnée. Ce qui transfère vers vous, l'utilisateur, la responsabilité de vous rappeler que la modification n'a en fait pas été fusionnée, qu'elle n'était tout simplement pas voulue. Il n'y a pas moyen de demander à Subversion la liste des « listes de modifications bloquées ». Si vous voulez en conserver la trace (afin de pouvoir les débloquer un jour), vous devrez les consigner dans un fichier texte quelque part, ou peut-être dans une propriété inventée de toutes pièces pour l'occasion. Dans Subversion 1.5, malheureusement, c'est la seule façon de gérer les révisions bloquées ; dans les versions futures il est prévu d'en améliorer l'interface.

Historiques et annotations tenant compte des fusions passées

Une des fonctionnalités principales de tout système de gestion de versions est de conserver la trace de qui a modifié quoi et quand ils l'ont fait. Les commandes **svn log** et **svn blame** sont les outils adaptés pour cela : quand on les applique à des fichiers individuels, ils renvoient non seulement l'historique des ensembles de modifications qui ont touché le fichier, mais aussi exactement quel utilisateur a écrit quelle ligne de code et quand il l'a fait.

Cependant, quand des modifications commencent à être copiées entre des branches, les choses commencent à se compliquer. Par exemple, si vous interrogiez **svn log** sur l'historique de votre branche fonctionnelle, il renverrait exactement toutes les révisions qui ont touché cette branche :

```
$ cd ma-branche-calc
$ svn log -q
------------------------------------------------------------------------
r390 | utilisateur | 2002-11-22 11:01:57 -0600 (ven. 22 nov. 2002) | 1 ligne
------------------------------------------------------------------------
r388 | utilisateur | 2002-11-21 05:20:00 -0600 (jeu. 21 nov. 2002) | 2 lignes
------------------------------------------------------------------------
r381 | utilisateur | 2002-11-20 15:07:06 -0600 (mer. 20 nov. 2002) | 2 lignes
------------------------------------------------------------------------
r359 | utilisateur | 2002-11-19 19:19:20 -0600 (mar. 19 nov. 2002) | 2 lignes
------------------------------------------------------------------------
r357 | utilisateur | 2002-11-15 14:29:52 -0600 (ven. 15 nov. 2002) | 2 lignes
------------------------------------------------------------------------
r343 | utilisateur | 2002-11-07 13:50:10 -0600 (jeu. 07 nov. 2002) | 2 lignes
------------------------------------------------------------------------
r341 | utilisateur | 2002-11-03 07:17:16 -0600 (dim. 03 nov. 2002) | 2 lignes
------------------------------------------------------------------------
r303 | sally | 2002-10-29 21:14:35 -0600 (mar. 29 oct. 2002) | 2 lignes
------------------------------------------------------------------------
r98 | sally | 2002-02-22 15:35:29 -0600 (ven. 22 fev. 2002) | 2 lignes
------------------------------------------------------------------------
```

Mais est-ce bien là une description adéquate de tous les changements qui ont eu lieu sur cette branche ? Ce qui manque ici, c'est le fait que les révisions 390, 381 et 357 résultaient en fait de fusions en provenance du tronc. Si vous regardez plus en détail l'historique d'une de ces révisions, vous ne verrez nulle part les multiples ensembles de modifications du tronc qui ont été reportés sur la branche :

```
$ svn log -v -r 390
------------------------------------------------------------------------
r390 | utilisateur | 2002-11-22 11:01:57 -0600 (ven. 22 nov. 2002) | 1 ligne
Chemins modifiés :
   M /branches/ma-branche-calc/bouton.c
   M /branches/ma-branche-calc/LISEZMOI

Fusion finale des modifications du tronc dans ma-branche-calc.
```

Il se trouve que nous savons que cette fusion vers la branche n'était qu'une fusion de modifications du tronc. Comment pouvons-nous également voir ces modifications du tronc ? La réponse est d'utiliser l'option `--use-merge-history` (`-g`). Cette option donne le détail des modifications « filles » qui faisaient partie de la fusion.

```
$ svn log -v -r 390 -g
------------------------------------------------------------------------
r390 | utilisateur | 2002-11-22 11:01:57 -0600 (ven. 22 nov. 2002) | 1 ligne
Chemins modifiés :
   M /branches/ma-branche-calc/bouton.c
   M /branches/ma-branche-calc/LISEZMOI

Fusion finale des modifications du tronc dans ma-branche-calc.
------------------------------------------------------------------------
r383 | sally | 2002-11-21 03:19:00 -0600 (jeu. 21 nov. 2002) | 2 lignes
Chemins modifiés :
   M /branches/ma-branche-calc/bouton.c
Fusion via: r390

Corrige l'erreur d'inversion graphique sur le bouton.
------------------------------------------------------------------------
r382 | sally | 2002-11-20 16:57:06 -0600 (mer. 20 nov. 2002) | 2 lignes
Chemins modifiés :
   M /branches/ma-branche-calc/LISEZMOI
```

```
Fusion via: r390

Documente mon dernier correctif dans LISEZMOI.
```

En forçant l'opération **svn log** à utiliser l'historique des fusions, nous obtenons non seulement la révision que nous avions demandé (r390), mais aussi les deux révisions qui l'accompagnaient — deux modifications du tronc faites par Sally. C'est une image bien plus complète de l'historique !

La commande **svn blame** accepte également l'option `--use-merge-history` (`-g`). Si cette option est omise, quelqu'un qui regarderait un relevé annoté ligne par ligne pour `bouton.c` risquerait d'avoir l'impression erronée que vous êtes responsable des lignes qui ont corrigé une certaine erreur :

```
$ svn blame bouton.c
…
   390    utilisateur  retval = inverse_func(button, path);
   390    utilisateur  return retval;
   390    utilisateur  }
…
```

Et bien qu'il soit vrai que vous avez propagé ces trois lignes lors de la révision 390, deux d'entre elles ont en fait été écrites par Sally auparavant, en révision 383 :

```
$ svn blame bouton.c -g
…
G    383    sally        retval = inverse_func(button, path);
G    383    sally        return retval;
     390    utilisateur  }
…
```

À présent, nous savons qui doit *réellement* être tenu responsable pour ces deux lignes de code !

Prise en compte ou non de l'ascendance

Quand vous discutez avec un développeur Subversion, il est très possible qu'il fasse référence au terme d'*ascendance*. Ce mot est utilisé pour décrire la relation entre deux objets dans un dépôt : s'ils sont liés l'un à l'autre, un des objets est alors qualifié d'ancêtre de l'autre.

Par exemple, supposons que vous propagiez la révision 100 qui contient une modification d'un fichier `truc.c`. Dès lors, `truc.c@99` est un « ancêtre » de `truc.c@100`. En revanche, supposons que vous propagiez la suppression de `truc.c` en révision 101 et ensuite l'ajout d'un nouveau fichier du même nom en révision 102. Dans ce cas, `truc.c@99` et `truc.c@102` pourraient sembler apparentés (ils ont le même chemin), mais en fait ce sont des objets complètement différents au sein du dépôt. Ils ne partagent aucun historique ou « ascendance ».

Nous abordons ce point pour mettre en évidence une différence importante entre **svn diff** et **svn merge**. La première commande ignore toute ascendance, tandis que la seconde y est particulièrement sensible. Par exemple, si vous demandez à **svn diff** de comparer les révisions 99 et 102 de `truc.c`, vous obtenez des différences basées sur les lignes ; la commande **svn diff** compare deux chemins à l'aveugle. Mais si vous demandez à **svn merge** de comparer les deux mêmes objets, il remarque qu'ils ne sont pas liés et essaie d'abord de supprimer l'ancien fichier, puis d'ajouter le nouveau fichier ; le résultat indique une suppression puis un ajout :

```
D    truc.c
A    truc.c
```

La plupart des fusions impliquent de comparer des arborescences qui ont une relation d'ascendance de l'une à l'autre ; c'est pourquoi **svn merge** a ce comportement par défaut. Cependant, à l'occasion, vous pourriez vouloir que la commande **svn merge** compare deux arborescences sans relation d'ascendance entre elles. Par exemple, vous avez peut-être importé deux arborescences de code source représentant des publications différentes de deux fournisseurs d'un projet logiciel (voir la section intitulée « Branches fournisseur »). Si vous demandez à **svn merge** de comparer les deux arborescences, vous verrez la

première arborescence complètement supprimée, puis l'ajout de la seconde arborescence toute entière ! Dans ce genre de situations, vous attendez de **svn merge** qu'il effectue une comparaison basée sur les chemins uniquement, en ignorant toute relation entre les fichiers et les dossiers. Ajoutez l'option `--ignore-ancestry` à votre commande **svn merge** et elle se comportera comme **svn diff** (et inversement, l'option `--notice-ancestry` fera se comporter **svn diff** comme la commande **svn merge**).

Fusions, copies et renommages

Il arrive souvent qu'on veuille réorganiser le code source, en particulier dans les projets logiciels en Java. Les fichiers et les répertoires sont déplacés et renommés, causant souvent de grandes perturbations pour tous ceux qui travaillent sur le projet. Ceci semble être le cas idéal où utiliser une branche, n'est-ce pas ? Créer une branche, réorganiser les choses, et ensuite fusionner la branche vers le tronc, non ?

Hélas, ce scénario ne fonctionne pas si bien pour le moment et est même considéré comme l'une des faiblesses de Subversion. Le problème est que la commande **svn update** de Subversion n'est pas aussi robuste qu'elle le devrait, en particulier en ce qui concerne les opérations de copies et de déplacements.

Quand vous utilisez **svn copy** pour dupliquer un fichier, le dépôt se souvient d'où venait le nouveau fichier, mais il ne transmet pas cette information au client qui lance **svn update** ou **svn merge**. Au lieu de dire au client « Copie ce fichier que tu as déjà vers ce nouvel emplacement », il envoie un fichier entièrement nouveau. Ceci peut engendrer des problèmes, en particulier parce que le fonctionnement est le même pour les fichiers renommés. Un fait peu connu à propos de Subversion est qu'il lui manque de « vrais renommages » - la commande **svn move** n'est rien de plus que l'agrégation de **svn copy** et **svn delete**.

Par exemple, supposons que pendant que vous travaillez sur votre branche privée, vous renommez `entier.c` en `tout.c`. Ce faisant, vous avez en fait créé un nouveau fichier dans votre branche, qui est une copie du fichier original, et vous avez supprimé le fichier original. Pendant ce temps, dans le tronc, Sally a propagé des améliorations d'`entier.c`. C'est alors que vous décidez de fusionner votre branche vers le tronc :

```
$ cd calc/trunk

$ svn merge --reintegrate http://svn.exemple.com/depot/calc/branches/ma-branche-calc
--- Fusion des différences des URLs du dépôt vers '.' :
D   entier.c
A   tout.c
U   .
```

Ceci n'a pas l'air si mal à première vue, mais ce n'est probablement pas ce à quoi Sally ou vous vous attendiez. L'opération de fusion a supprimé la version la plus récente du fichier `entier.c` (celle qui contenait les dernières modifications de Sally) et a ajouté machinalement votre nouveau fichier `tout.c`, qui est une copie de l'ancien fichier `entier.c`. Le résultat final est que la fusion de votre « renommage » a supprimé de la dernière révision les modifications récentes de Sally !

Il ne s'agit pas d'une vraie perte de données. Les modifications de Sally font toujours partie de l'historique du dépôt, mais le déroulement exact des événements n'est peut-être pas immédiatement évident. La morale de l'histoire est que jusqu'à ce que Subversion ne s'améliore, vous devrez faire très attention aux fusions de copies et de renommages d'une branche à une autre.

Blocage des clients qui ne prennent pas en compte les fusions

Si vous venez de mettre à niveau votre serveur Subversion à la version 1.5 ou plus, il existe un risque significatif que les clients Subversion pré-1.5 sèment la pagaille dans votre suivi automatique des fusions. Pourquoi ? Quand un client Subversion pré-1.5 exécute **svn merge**, il ne modifie pas du tout la valeur de la propriété `svn:mergeinfo`. La propagation qui s'ensuit, bien qu'elle soit le résultat d'une fusion, n'envoie donc aucune indication au dépôt au sujet des modifications dupliquées — ces informations sont perdues. Par la suite, lorsque des clients « qui prennent en compte les fusions » tentent d'effectuer une fusion automatique, ils rencontreront probablement toutes sortes de conflits résultant des fusions répétées.

Si votre équipe et vous dépendez des fonctionnalités de suivi des fusions de Subversion, vous voudrez peut-être configurer votre dépôt pour qu'il empêche les anciens clients de propager des modifications. La méthode la plus simple est d'examiner le paramètre « capabilities » dans la procédure automatique de début de propagation. Si le client indique être capable de gérer les `mergeinfo`, la procédure automatique peut l'autoriser à commencer la propagation. Si le client n'indique pas en être capable, la procédure automatique doit lui refuser la propagation. Nous en apprendrons plus sur les procédures automatiques dans le

chapitre suivant ; voir la section intitulée « Mise en place des procédures automatiques » et start-commit pour les détails.

Recommandations finales sur le suivi des fusions

En fin de compte, la fonctionnalité de suivi des fusions de Subversion possède une mécanique interne extrêmement complexe et la propriété `svn:mergeinfo` est la seule lorgnette dont l'utilisateur dispose pour observer cette mécanique. Parce que cette fonctionnalité est relativement nouvelle, un certain nombre de cas litigieux et de comportements potentiellement inattendus risquent d'être rencontrés.

Par exemple, `mergeinfo` sera parfois générée lors d'une simple commande **svn copy** ou **svn move**. Parfois `mergeinfo` apparaîtra sur des fichiers dont vous n'auriez pas imaginé qu'ils soient touchés par une opération. Parfois `mergeinfo` ne sera pas du tout générée, contrairement à ce que vous attendiez. De plus, la gestion de la métadonnée `mergeinfo` a tout un ensemble de taxonomies et de comportements qui lui sont associés, tels que des `mergeinfo` « explicites » par opposition à « implicites », des révisions « opérationnelles » par opposition à « non-opérationnelles », des mécanismes spécifiques d'« élision » de `mergeinfo` et même d'« héritage » de répertoires parents à sous-répertoires.

Nous avons choisi de ne pas couvrir en détail ces sujets dans ce livre pour plusieurs raisons. Premièrement, l'utilisateur moyen serait totalement submergé par le niveau de détail disponible. Deuxièmement, au fur et à mesure que Subversion s'améliore, nous estimons que l'utilisateur moyen *ne devrait pas* avoir à comprendre ces concepts ; en tant que détails d'implémentation agaçants, ils finiront par disparaître à l'arrière-plan. Malgré tout ce qui vient d'être dit, si vous appréciez ce genre de choses, vous en trouverez une formidable vue d'ensemble dans un article posté sur le site internet de Collabnet : http://www.collab.net/community/subversion/articles/merge-info.html.

Pour le moment, si vous voulez rester à l'écart des bogues et autres comportements inattendus des fusions automatiques, l'article de Collabnet recommande que vous vous en teniez simplement aux bonnes pratiques suivantes :

- Pour les branches fonctionnelles à courte durée de vie, suivez la procédure simple décrite dans la section intitulée « Fusions : pratiques de base ».
- Pour les branches de publication à longue durée de vie (comme décrites dans la section intitulée « Modèles courants de gestion des branches »), ne pratiquez de fusions que sur la racine de la branche, pas sur des sous-répertoires.
- Ne pratiquez jamais de fusion vers des copies de travail contenant un mélange de numéros de révisions de travail, ou ayant des sous-répertoires « déportés » (comme décrit par la suite dans la section intitulée « Parcours des branches »). La cible d'une fusion doit être une copie de travail qui représente un emplacement *unique* à un moment *unique* dans le dépôt.
- Ne modifiez jamais la propriété `svn:mergeinfo` directement ; utilisez **svn merge** avec l'option `--record-only` pour appliquer une modification désirée à cette métadonnée (comme expliqué dans la section intitulée « Blocage de modifications »).
- Assurez-vous de toujours avoir l'accès complet en lecture à toutes vos sources de fusion, et vérifiez que votre copie de travail cible n'a pas de dossiers clairsemés.

Parcours des branches

La commande **svn switch** transforme une copie de travail existante de telle sorte qu'elle pointe vers une branche différente. Bien que la connaissance de cette commande ne soit pas absolument nécessaire pour travailler avec des branches, elle fournit un raccourci utile. Dans notre exemple précédent, après avoir créé votre branche privée, vous avez extrait une toute nouvelle copie de travail du nouveau répertoire du dépôt. À la place, vous pouvez simplement demander à Subversion de modifier votre copie de travail de `/calc/trunk` pour qu'elle pointe vers l'emplacement de la nouvelle branche :

```
$ cd calc

$ svn info | grep URL
URL: http://svn.exemple.com/depot/calc/trunk

$ svn switch http://svn.exemple.com/depot/calc/branches/ma-branche-calc
U   entier.c
U   bouton.c
U   Makefile
```

```
Actualisé à la révision 341.

$ svn info | grep URL
URL: http://svn.exemple.com/depot/calc/branches/ma-branche-calc
```

« Faire pointer » (ou « déporter ») une copie de travail qui n'a pas de modifications locales vers une branche différente a pour résultat que la copie de travail a exactement le même aspect que si vous aviez effectué une extraction brute du répertoire. C'est en général plus efficace d'utiliser cette commande, car les différences entre les branches sont souvent minimes. Le serveur n'envoie que le minimum de modifications nécessaire pour faire pointer votre copie de travail vers le répertoire de la branche.

La commande **svn switch** accepte également l'option `--revision` (`-r`), pour que que vous ne soyez pas obligé de faire pointer votre copie de travail vers la révision `HEAD` de la branche.

Bien sûr, la plupart des projets sont plus compliqués que notre exemple `calc` et contiennent de multiples sous-dossiers. Les utilisateurs de Subversion suivent souvent un algorithme précis quand ils utilisent des branches :

1. Copier le « tronc » entier du projet vers une nouvelle branche
2. Ne déporter qu'*une partie* de la copie de travail du tronc pour qu'elle pointe sur la branche.

En d'autres termes, si un utilisateur sait que le travail sur la branche ne doit avoir lieu que sur un sous-dossier donné, il utilise **svn switch** pour ne faire pointer que ce sous-dossier vers la branche (ou parfois des utilisateurs ne vont faire pointer qu'un unique fichier de travail vers la branche !). De cette façon, l'utilisateur peut continuer à recevoir les mises à jour normales du « tronc » vers la plus grande partie de sa copie de travail, mais les portions déportées ne seront pas touchées (à moins que quelqu'un ne propage une modification à sa branche). Cette fonctionnalité ajoute une dimension complètement nouvelle au concept de « copie de travail mixte » : les copies de travail peuvent non seulement contenir un mélange de révisions de travail, mais elles peuvent également contenir un mélange d'emplacements du dépôt.

Si votre copie de travail contient un certain nombre de sous-arborescences pointant vers des emplacements variés du dépôt, elle continue à fonctionner normalement. Quand vous la mettez à jour, vous recevez comme il se doit les correctifs pour chaque sous-arborescence. Quand vous effectuez une propagation, vos modifications locales s'appliquent toujours au dépôt en tant qu'une unique modification atomique.

Remarquez que bien qu'il soit possible pour votre copie de travail de pointer vers une variété d'emplacements du dépôt, ces emplacements doivent tous faire partie du *même* dépôt. Les dépôts Subversion ne sont pas encore capables de communiquer entre eux ; cette fonctionnalité est prévue à l'avenir [5].

Déports et mises à jour

Avez-vous remarqué que les sorties des commandes **svn switch** et **svn update** se ressemblent ? La commande **svn switch** est en fait une généralisation de la commande **svn update**.

Quand vous lancez **svn update**, vous demandez au dépôt de comparer deux arborescences. C'est ce qu'il fait, puis il renvoie au client le détail des différences entre les deux. La seule différence entre **svn switch** et **svn update** est que cette dernière commande compare toujours deux chemins identiques du dépôt.

C'est-à-dire que si votre copie de travail pointe vers `/calc/trunk`, **svn update** compare automatiquement votre copie de travail de `/calc/trunk` au `/calc/trunk` de la révision `HEAD`. Si vous faites pointer votre copie de travail vers une branche, **svn switch** comparera votre copie de travail de `/calc/trunk` au répertoire d'une *autre* branche de la révision `HEAD`.

En d'autres termes, **svn update** déplace votre copie de travail à travers le temps. **svn switch** déplace votre copie de travail à travers le temps *et* l'espace.

Parce que **svn switch** est essentiellement une variante de **svn update**, elle se comporte de la même manière ; toute modification locale présente dans votre copie de travail est préservée lorsque de nouvelles données arrivent en provenance du

[5] Vous *pouvez* cependant utiliser **svn switch** avec l'option `--relocate` si l'URL de votre serveur change et si vous ne voulez pas abandonner votre copie de travail existante. Reportez-vous à svn switch pour des détails et des exemples.

dépôt.

Vous-êtes vous déjà trouvés dans une situation où vous effectuez des modifications complexes (dans votre copie de travail de `/trunk`) et réalisez soudainement : « Mais, ces modifications ne devraient-elles pas être dans leur propre branche ? » Une excellente technique pour accomplir ceci peut être résumée en deux étapes :

```
$ svn copy http://svn.exemple.com/depot/calc/trunk \
           http://svn.exemple.com/depot/calc/branches/nouvelle-branche \
      -m "Création de la branche 'nouvelle-branche'."
Révision 353 propagée.
$ svn switch http://svn.exemple.com/depot/calc/branches/nouvelle-branche
À la révision 353.
```

La commande **svn switch**, à l'instar de **svn update**, préserve vos modifications locales. Désormais, votre copie de travail pointe vers la branche nouvellement créée et la prochaine fois que vous lancerez **svn commit** vos modifications y seront envoyées.

Étiquettes

Un autre concept courant en gestion de versions est l'*étiquette* (parfois appelée *tag*). Une étiquette n'est qu'un « instantané » d'un projet à un moment donné. Dans Subversion, cette idée semble être présente partout. Chaque révision du dépôt est exactement cela : un instantané du système de fichiers pris après chaque propagation.

Cependant les gens veulent souvent donner des noms plus conviviaux aux étiquettes, tel que `version-1.0`. Et ils veulent prendre des instantanés de sous-sections plus restreintes du système de fichiers. Après tout, ce n'est pas si facile de se rappeler que la version 1.0 d'un logiciel donné correspond à un sous-dossier particulier de la révision 4822.

Création d'une étiquette simple

Une fois encore, **svn copy** vient à la rescousse. Si vous voulez créer un instantané de `/calc/trunk` identique à ce qu'il est dans la révision `HEAD`, faites-en une copie :

```
$ svn copy http://svn.exemple.com/depot/calc/trunk \
           http://svn.exemple.com/depot/calc/tags/version-1.0 \
      -m "Étiquetage de la version 1.0 du projet 'calc'."

Révision 902 propagée.
```

Cet exemple présuppose qu'un répertoire `/calc/tags` existe déjà (s'il n'existe pas, vous pouvez le créer en utilisant **svn mkdir**). Une fois la copie terminée, le nouveau dossier `version-1.0` sera pour toujours un instantané du dossier `/trunk` tel qu'il était en révision `HEAD` au moment où vous avez effectué la copie. Bien sûr, vous voudriez peut-être être plus précis quant à quelle révision vous copiez, au cas où quelqu'un d'autre aurait propagé des modifications au projet pendant que vous regardiez ailleurs. Donc si vous savez que la révision 901 de `/calc/trunk` est exactement l'instantané que vous voulez, vous pouvez le spécifier en passant `-r 901` à la commande **svn copy**.

Mais attendez un moment : cette procédure de création d'étiquette, n'est-ce pas la même procédure que nous avons utilisé pour créer une branche ? En fait, oui. Dans Subversion, il n'y pas de différence entre une étiquette et une branche. Toutes deux ne sont que des répertoires ordinaires qui sont créés par copie. Comme pour les branches, la seule raison qui fasse qu'un répertoire copié soit une « étiquette » est que les *humains* ont décidé de le traiter de cette façon : aussi longtemps que personne ne propage de modification à ce répertoire, il reste un instantané. Si les gens commencent à y propager des choses, il devient une branche.

Si vous administrez un dépôt, il y a deux approches possibles pour gérer les étiquettes. La première approche est une politique de « non-intervention » : en tant que convention définie pour le projet, vous décidez où vos étiquettes sont placées et vous vous assurez que tous les utilisateurs savent comment traiter les répertoires qu'ils copient (c'est-à-dire que vous vous assurez qu'ils savent qu'ils ne doivent rien y propager). La seconde approche est plus paranoïaque : vous pouvez utiliser un des contrôles d'accès fournis avec Subversion pour empêcher que quiconque ne puisse faire autre chose dans la zone des étiquettes que d'y créer de nouvelles copies (voir le Chapitre 6, *Configuration du serveur*). L'approche paranoïaque n'est cependant pas

nécessaire, en général. Si un utilisateur propage accidentellement une modification à un répertoire d'étiquettes, vous pouvez simplement revenir en arrière sur cette modification comme expliqué dans le paragraphe précédent. C'est ça la gestion de versions, après tout !

Création d'une étiquette complexe

Parfois vous voudrez peut-être que votre « instantané » soit plus compliqué qu'un simple répertoire à une unique révision donnée.

Par exemple, imaginons que votre projet est bien plus vaste que notre exemple `calc` : supposons qu'il contient un bon nombre de sous-répertoires et bien plus de fichiers encore. Au cours de votre travail, vous pouvez très bien décider que vous avez besoin de créer une copie de travail destinée à des fonctionnalités particulières et à des corrections de bogues. Pour cela vous pouvez antidater de manière sélective des fichiers ou dossiers à des révisions données (en utilisant généreusement **svn update** avec l'option `-r`), déporter des fichiers et des dossiers vers des branches particulières (au moyen de **svn switch**) ou même effectuer manuellement un tas de modifications locales. Quand vous en avez terminé, votre copie de travail est un vrai bazar, fait d'emplacements du dépôt à des révisions différentes. Mais après l'avoir testée, vous êtes alors certain que c'est l'exacte combinaison de données que vous vouliez étiqueter.

C'est alors le moment de prendre un cliché. Copier une URL vers une autre ne fonctionnera pas cette fois. Dans le cas présent, vous voulez prendre un cliché de l'arrangement exact de votre copie de travail et le placer dans le dépôt. Par chance, **svn copy** possède en fait quatre usages différents (au sujet desquels vous pouvez obtenir des informations au Chapitre 9, *Références complètes de Subversion*), dont la possibilité de copier une arborescence de travail vers le dépôt :

```
$ ls
ma-copie-de-travail/

$ svn copy ma-copie-de-travail \
           http://svn.exemple.com/depot/calc/tags/mon-etiquette \
           -m "Étiquette l'état de ma copie de travail existante."

Révision 940 propagée.
```

Désormais il y a un nouveau répertoire dans le dépôt, `/calc/tags/mon-etiquette`, qui est un instantané exact de votre copie de travail : révisions mixtes, URL, modifications locales et tout et tout…

D'autres utilisateurs ont trouvé des usages intéressants pour cette fonctionnalité. Il y a parfois des situations où votre copie de travail contient un paquet de modifications locales que vous aimeriez montrer à un collaborateur. Au lieu de lancer **svn diff** et d'envoyer un fichier patch (qui ne listera pas les modifications de répertoires, de liens symboliques ou de propriétés), vous pouvez utiliser **svn copy** pour « envoyer » votre copie de travail vers une zone privée du dépôt. Votre collaborateur peut ensuite soit extraire une copie carbone de votre copie de travail, soit utiliser **svn merge** pour recevoir exactement vos modifications.

Bien que ce soit une jolie méthode pour mettre à disposition un instantané rapide de votre copie de travail, remarquez que *ce n'est pas* une bonne manière de créer une branche initialement. La création de branche devrait être un évènement en soi, tandis que cette méthode combine la création d'une branche avec des modifications supplémentaires apportées aux fichiers, le tout au sein d'une seule révision. Ceci rend très difficile (à terme) d'identifier un unique numéro de révision en tant que point de création de la branche.

Maintenance des branches

À ce stade, vous vous êtes certainement rendu compte que Subversion est extrêmement flexible. Parce qu'il implémente les branches et les étiquettes avec le même mécanisme sous-jacent (des copies de répertoires) et parce que les branches et les étiquettes apparaissent au sein de l'espace standard du système de fichiers, beaucoup de gens trouvent Subversion intimidant. Il est presque *trop* flexible. Dans ce paragraphe, nous proposons des suggestions pour organiser et gérer vos données au fil du temps.

Agencement du dépôt

Il existe des méthodes standard recommandées pour structurer un dépôt. La plupart des gens créent un répertoire `trunk` pour la « ligne de développement principale » (le tronc), un répertoire `branches` qui contiendra les copies de branches et un

répertoire `tags` qui contiendra les copies étiquetées. Si un dépôt ne comprend qu'un seul projet, les gens créent souvent les dossiers suivants à la racine :

```
/trunk
/branches
/tags
```

Si un dépôt contient plusieurs projets, les administrateurs indexent généralement la structure du dépôt par projet (voir la section intitulée « Stratégies d'organisation d'un dépôt » pour en savoir plus sur les « dossiers racine d'un projet ») :

```
/paint/trunk
/paint/branches
/paint/tags
/calc/trunk
/calc/branches
/calc/tags
```

Bien sûr, vous restez libre d'ignorer ces agencements courants. Vous pouvez créer toutes sortes de variantes, selon ce qui fonctionne le mieux pour vous ou pour votre équipe. Souvenez-vous que quel que soit votre choix, ce n'est pas un engagement définitif. Vous pouvez réorganiser votre dépôt à tout moment. Parce que les branches et les étiquettes sont des répertoires ordinaires, la commande **svn move** peut les déplacer ou les renommer selon vos désirs. Passer d'un agencement à un autre consiste juste à lancer une série d'opérations de déplacement côté serveur ; si vous n'aimez pas la façon dont les choses sont organisées dans le dépôt, modifiez juste leur agencement.

Souvenez-vous néanmoins que bien qu'il soit facile de déplacer des dossiers, vous devez aussi rester attentif à vos utilisateurs. Vos modifications sont susceptibles de déboussoler ceux qui ont des copies de travail existantes. Si un utilisateur a une copie de travail d'un répertoire donné du dépôt, votre opération **svn move** risque de supprimer ce chemin de la révision la plus récente. Lorsque par la suite l'utilisateur lancera **svn update**, il se verra annoncer que sa copie de travail représente un chemin qui n'existe plus et sera contraint d'effectuer un **svn switch** vers le nouvel emplacement.

Durée de vie des données

Une autre fonctionnalité intéressante liée aux principes de fonctionnement de Subversion est que les branches et les étiquettes peuvent avoir des durées de vie limitées, tout comme n'importe quel autre élément suivi en versions. Par exemple, supposons que vous avez enfin terminé votre travail sur votre branche personnelle du projet `calc`. Après avoir fusionné toutes vos modifications vers `/calc/trunk`, le répertoire contenant votre branche privée n'a plus de raison d'exister :

```
$ svn delete http://svn.exemple.com/depot/calc/branches/ma-branche-calc \
             -m "Suppression d'une branche obsolète du projet calc."

Révision 375 propagée.
```

Et maintenant votre branche a disparu. Bien sûr, elle n'a pas vraiment disparu : le répertoire est juste absent de la révision `HEAD`, ne gênant plus personne. Si vous utilisez **svn checkout**, **svn switch** ou **svn list** pour examiner une révision plus ancienne, vous pourrez toujours voir votre vieille branche.

Si la navigation dans votre répertoire supprimé ne vous suffit pas, vous pouvez toujours le récupérer. Ressusciter des données est très facile dans Subversion. S'il y a un dossier (ou un fichier) supprimé que vous aimeriez faire réapparaître dans `HEAD`, utilisez simplement **svn copy** pour le copier depuis l'ancienne révision.

```
$ svn copy http://svn.exemple.com/depot/calc/branches/ma-branche-calc@374 \
           http://svn.exemple.com/depot/calc/branches/ma-branche-calc \
           -m "Restaure ma-branche-calc."

Révision 376 propagée.
```

Dans notre exemple, votre branche personnelle a eu une durée de vie relativement limitée : vous l'aviez peut-être créée pour corriger un bogue ou implémenter une nouvelle fonctionnalité. Quand votre tâche est finie, il en va de même pour la branche.

Cependant, en développement logiciel, il est aussi courant d'avoir deux branches « principales » côte à côte pour de très longues périodes. Par exemple, supposons que le moment est venu de publier une version stable du projet `calc` pour le public. Vous savez qu'il faudra quelques mois pour éliminer les bogues du logiciel. Vous ne voulez pas que les gens ajoutent de nouvelles fonctionnalités au projet, mais vous ne voulez pas non plus dire à tous les développeurs d'arrêter de programmer. Donc à la place, vous créez une branche « stable » du logiciel qui ne changera pas beaucoup :

```
$ svn copy http://svn.exemple.com/depot/calc/trunk \
           http://svn.exemple.com/depot/calc/branches/stable-1.0 \
           -m "Création de la branche stable du projet calc."

Révision 377 propagée.
```

Dès lors les développeurs sont libres de continuer à ajouter des fonctionnalités de pointe (ou expérimentales) à `/calc/trunk` et vous pouvez poser comme convention pour le projet que seules les corrections de bogues seront propagées dans `/calc/branches/stable-1.0`. C'est-à-dire qu'au fur et à mesure que les gens continueront de travailler sur le tronc, quelqu'un reportera de façon sélective les corrections de bogues vers la branche stable. Même après que la branche stable aura été publiée, vous continuerez probablement à maintenir la branche pendant longtemps, c'est-à-dire pour aussi longtemps que vous continuerez à fournir aux clients un support sur cette version. Nous évoquons ceci plus en détails dans le prochain paragraphe.

Modèles courants de gestion des branches

Il existe de nombreux usages pour la création et la fusion des branches ; ce paragraphe décrit les plus courants.

Le plus souvent, la gestion de versions est utilisée pour le développement de logiciels, voici donc un coup d'œil rapide à deux des modèles les plus courants de création et de fusion de branches utilisés par les équipes de programmeurs. Si vous ne vous servez pas de Subversion pour développer des logiciels, n'hésitez pas à sauter ce paragraphe. Si vous êtes un développeur de logiciels qui utilise la gestion de versions pour la première fois, soyez très attentifs, car ces modèles sont souvent considérés comme des bonnes pratiques par les développeurs plus expérimentés. Ces procédures ne sont pas spécifiques à Subversion ; elles sont applicables à tout système de gestion de versions. Néanmoins, les voir explicitées en termes Subversion peut aider.

Branches de publication

En général un logiciel suit un cycle de vie classique, répétant les trois étapes suivantes en boucle : code, test, publication. Il y a deux problèmes avec ce processus. Premièrement, les développeurs doivent continuer à écrire de nouvelles fonctionnalités pendant que les équipes d'assurance qualité prennent le temps de tester des versions supposées stables du logiciel. Les nouveaux développements ne peuvent pas s'arrêter pendant que le logiciel est en cours de test. Deuxièmement, l'équipe doit presque toujours effectuer le support des versions anciennes et publiées du logiciel ; si un bogue est découvert dans le code le plus récent, il existe probablement aussi dans les versions qui ont été publiées et les clients voudront obtenir le correctif pour ce bogue sans avoir à attendre la publication d'une nouvelle version majeure.

C'est là où la gestion de versions peut s'avérer utile. La procédure standard ressemble à ceci :

1. *Les développeurs propagent tout nouveau travail vers le tronc.* Les modifications quotidiennes sont propagées vers `/trunk` : nouvelles fonctionnalités, corrections de bogues, etc.

2. *Le tronc est copié vers une branche « de publication ».* Lorsque l'équipe estime que le logiciel est prêt à être publié (disons en version 1.0), `/trunk` peut être copié vers `/branches/1.0`.

3. *Les équipes continuent à travailler en parallèle.* Une équipe commence à tester rigoureusement la branche de publication, pendant qu'une autre équipe continue avec les nouvelles tâches (disons pour la version 2.0) sur `/trunk`. Si des bogues sont découverts dans l'un ou l'autre des emplacements, les correctifs sont reportés de l'un à l'autre selon les besoins. Il arrive cependant un moment où même ce processus s'arrête. La branche est « gelée » pour les tous derniers tests juste avant publication.

4. *La branche est étiquetée et publiée.* Quand les tests sont terminés, `/branches/1.0` est copiée vers `/tags/1.0.0` en tant que cliché de référence. L'étiquette est exportée et livrée aux clients.

5. *La branche est gérée au fil du temps.* Pendant que le travail continue sur `/trunk` en vue de la version 2.0, les correctifs de

bogues continuent à être reportés de `/trunk` à `/branches/1.0`. Lorsque suffisamment de correctifs se sont accumulés, les responsables peuvent décider de publier une version 1.0.1 : `/branches/1.0` est copiée vers `/tags/1.0.1` et cette étiquette est exportée et publiée.

Ce processus entier se répète au fur et à mesure que le logiciel gagne en maturité : quand le travail pour la version 2.0 est terminé, une nouvelle branche de publication 2.0 est créée, testée, étiquetée et finalement publiée. Au bout de quelques années, le dépôt finit par avoir un certain nombre de branches de publication en mode « maintenance » et un certain nombre d'étiquettes représentant les versions finales publiées.

Branches fonctionnelles

Une *branche fonctionnelle* est la sorte de branche qui est l'exemple dominant dans ce chapitre (celle sur laquelle vous travailliez pendant que Sally continuait à travailler sur `/trunk`). C'est une branche temporaire créée pour travailler sur un changement complexe sans interférer avec la stabilité de `/trunk`. À la différence des branches de publication (dont le support doit parfois être prolongé très longtemps), les branches fonctionnelles naissent, sont utilisées pendant un temps, sont fusionnées vers le tronc et sont finalement supprimées. Elles ont une utilité limitée dans le temps.

Encore une fois, les stratégies varient énormément au sujet du moment approprié pour créer une branche fonctionnelle. Certains projets n'utilisent jamais de branche fonctionnelle : n'importe qui peut propager des modifications à `/trunk`. L'avantage de ce système est qu'il est simple : personne n'a besoin d'être formé aux branches ou aux fusions. L'inconvénient est que le code du tronc est souvent instable ou inutilisable. D'autres projets utilisent les branches à l'extrême : une modification n'est *jamais* propagée directement dans le tronc. Même les modifications les plus triviales sont faites au sein d'une branche à courte durée de vie, vérifiées attentivement, puis fusionnées vers le tronc. La branche est ensuite supprimée. Ce système garantit que le tronc restera exceptionnellement stable et utilisable à tout moment, mais aux dépens des coûts de gestion liés à cette procédure très lourde.

La plupart des projets choisissent une approche à mi-chemin entre les deux. Ils insistent généralement pour qu'à tout moment `/trunk` puisse être compilé et passe avec succès les tests de régression. Une branche fonctionnelle n'est nécessaire que quand une modification nécessite un grand nombre de propagations susceptibles de déstabiliser le tronc. Une bonne méthode empirique est de se poser la question suivante : si le développeur travaillait pendant plusieurs jours en isolation et ensuite propageait cette grosse modification en une seule fois (afin que `/trunk` ne soit jamais déstabilisé), est-ce que ce serait une modification trop grosse à vérifier ? Si la réponse à cette question est « oui », alors la modification devrait être développée sur une branche fonctionnelle. Au fur et à mesure que le développeur propage ses modifications incrémentales dans la branche, elles peuvent facilement être vérifiées par ses pairs.

Finalement, il reste la question de savoir quelle est la meilleure méthode pour garder une branche synchronisée avec le tronc au fur et à mesure que le travail avance. Comme nous l'avons mentionné précédemment, il est très risqué de travailler sur une branche pendant des semaines ou des mois ; le tronc continuera peut-être à recevoir des modifications, au point que les deux lignes de développement risquent de s'éloigner tellement l'une de l'autre qu'essayer de fusionner la branche vers le tronc devienne un cauchemar.

Le mieux pour éviter une telle situation est de fusionner régulièrement les modifications du tronc vers la branche. Faites-en une habitude : une fois par semaine, fusionnez les modifications du tronc de la semaine précédente vers la branche.

Le moment arrivera où vous serez prêt à fusionner la branche fonctionnelle « synchronisée » vers le tronc. Commencez donc par effectuer une dernière fusion des modifications les plus récentes du tronc vers la branche. Une fois que c'est fait, les dernières versions de la branche et du tronc sont absolument identiques, mises à part vos propres modifications sur la branche. Vous êtes alors en mesure de fusionner la branche vers le tronc avec l'option `--reintegrate` :

```
$ cd copie-de-travail-du-tronc

$ svn update
À la révision 1910.

$ svn merge --reintegrate http://svn.exemple.com/depot/calc/branches/ma-branche
--- Fusion des différences des URLs du dépôt vers '.':
U    reel.c
U    entier.c
A    nouveau-dossier
A    nouveau-dossier/nouveau-fichier
 U   .
```

...

Une autre façon de concevoir ce modèle est d'imaginer que votre synchronisation hebdomadaire du tronc vers la branche est analogue au lancement de **svn update** dans une copie de travail, tandis que l'étape finale de fusion est analogue au lancement de **svn commit** depuis une copie de travail. Après tout, une copie de travail n'est rien d'autre qu'une branche privée très superficielle : c'est une branche qui n'est capable de ne contenir qu'une modification à la fois.

Branches fournisseur

Comme c'est particulièrement le cas en développement logiciel, les données que vous gérez dans votre système de gestion de versions ont souvent un lien étroit avec les données de quelqu'un d'autre, ou en sont peut-être dépendantes. Généralement, les besoins de votre projet vous obligeront à rester aussi à jour que possible avec les données fournies par cette entité externe, sans sacrifier la stabilité de votre propre projet. Ce scénario arrive très souvent, partout où les informations générées par un groupe de personnes ont un effet direct sur celles qui sont générées par un autre groupe de personnes.

Par exemple, il arrive que des développeurs de logiciel travaillent sur une application qui utilise une bibliothèque tierce. Subversion a justement une relation de ce type avec la bibliothèque Apache Portable Runtime (APR) (voir la section intitulée « APR, la bibliothèque Apache de portabilité des exécutables »). Le code source de Subversion dépend de la bibliothèque APR pour tous ses besoins de portabilité. Durant les étapes initiales de développement de Subversion, le projet suivait les changements de l'interface de programmation d'APR de près, restant toujours « à la pointe » des évolutions du code de la bibliothèque. Maintenant que APR et Subversion ont tous deux gagné en maturité, Subversion n'essaie de se synchroniser avec l'interface de programmation de l'APR qu'à des étapes de publication stables et bien testées.

Donc, si votre projet dépend des informations de quelqu'un d'autre, vous pourriez tenter de synchroniser ces informations avec les vôtres de plusieurs manières. La plus pénible serait de donner des instructions orales ou écrites à tous les contributeurs de votre projet, leur demandant de s'assurer qu'ils disposent des bonnes versions de ces informations tierces dont votre projet a besoin. Si les informations tierces sont gérées dans un dépôt Subversion, vous pourriez aussi utiliser les définitions externes de Subversion pour en fait « agrafer » des versions spécifiques de ces informations à un endroit quelconque dans le dossier de votre copie de travail (voir la section intitulée « Définition de références externes »).

Mais parfois vous voudrez gérer des modifications personnalisées de ce code tierce à l'intérieur de votre propre système de gestion de versions. En reprenant l'exemple du développement logiciel, les programmeurs pourraient vouloir apporter des modifications à cette bibliothèque tierce pour leurs propres besoins. Ces modifications incluraient peut-être de nouvelles fonctionnalités ou des corrections de bogues, gérées en interne seulement jusqu'à ce qu'elles soient incluses dans une version officielle de la bibliothèque tierce. Ou alors ces changements ne seront peut-être jamais remontés vers ceux qui gèrent cette bibliothèque, existant seulement en tant qu'optimisations « maison » permettant de mieux adapter la bibliothèque aux besoin des développeurs du logiciel.

À présent vous êtes face à une situation intéressante. Votre projet pourrait héberger ses modifications maison des données tierces de manière désordonnée, comme par exemple en utilisant des fichiers de patch ou des versions alternatives complètes des fichiers et dossiers. Mais ces méthodes deviennent rapidement de vrais casse-tête à gérer, nécessitant des mécanismes pour reporter vos modifications maison au code tierce et nécessitant le report de ces modifications à chaque version successive du code tierce dont vous dépendez.

La solution de ce problème est d'utiliser des *branches fournisseur*. Une branche fournisseur est une arborescence au sein de votre propre système de gestion de versions qui contient des informations fournies par une entité tierce, ou fournisseur. Chaque version des données du fournisseur que vous décidez d'incorporer dans votre projet est appelée une *livraison fournisseur*.

Les branches fournisseur présentent deux avantages. Premièrement, en incluant la livraison fournisseur actuellement supportée dans votre propre système de gestion de versions, vous avez la garantie que les membres de votre projet n'auront jamais besoin de se demander s'ils ont la bonne version des données du fournisseur. Ils reçoivent simplement la bonne version pendant les mises à jour usuelles de leur copie de travail. Deuxièmement, parce que ces données font partie de votre propre dépôt Subversion, vous pouvez y conserver vos modifications maison : vous n'avez plus besoin d'une méthode automatisée (ou pire, manuelle) pour reporter vos propres changements.

Procédure générale de gestion des branches fournisseur

La gestion des branches fournisseur fonctionne généralement de la façon suivante : vous créez d'abord un répertoire à la racine (par exemple `/fournisseur`) qui contiendra les branches fournisseur. Ensuite vous importez le code tierce dans un sous-dossier de ce dossier racine. Vous copiez ensuite ce sous-répertoire vers l'emplacement approprié de votre branche de

développement principale (par exemple `/trunk`). Vous faites bien attention à toujours effectuer vos modifications locales dans la branche de développement principale. À chaque nouvelle version du code tierce, vous le déposez dans la branche fournisseur et en fusionnez les modifications vers `/trunk`, en résolvant les conflits qui apparaissent entre vos modifications locales et les modifications tierces.

Un exemple va rendre cet algorithme plus clair. Nous allons utiliser un scénario dans lequel votre équipe de développement crée un programme de calcul qui dépend d'une bibliothèque tierce d'arithmétique des nombres complexes, libcomplex. Nous commencerons par la création initiale de la branche fournisseur et l'import de la première livraison fournisseur. Nous appellerons notre dossier contenant la branche fournisseur `libcomplex` et nos livraisons fournisseur iront dans un sous-dossier de notre branche fournisseur appelé `actuel`. Et puisque **svn import** crée tous les dossiers parents intermédiaires dont il a besoin, nous pouvons en fait accomplir ces deux étapes en une seule commande :

```
$ svn import /chemin/vers/libcomplex-1.0 \
             http://svn.exemple.com/depot/fournisseur/libcomplex/actuel \
             -m 'import initial de la livraison fournisseur 1.0'
…
```

Nous avons désormais la version actuelle du code source de libcomplex dans `/fournisseur/libcomplex/actuel`. À présent, nous étiquetons cette version (voir la section intitulée « Étiquettes ») et ensuite nous la copions dans la branche de développement principale. Cette opération de copie crée un nouveau dossier appelé `libcomplex` au sein du répertoire de notre projet existant `calc`. C'est dans cette copie des données du fournisseur que nous ferons nos ajustements maison :

```
$ svn copy http://svn.exemple.com/depot/fournisseur/libcomplex/actuel  \
           http://svn.exemple.com/depot/fournisseur/libcomplex/1.0      \
           -m 'étiquetage de libcomplex-1.0'
…
$ svn copy http://svn.exemple.com/depot/fournisseur/libcomplex/1.0  \
           http://svn.exemple.com/depot/calc/libcomplex             \
           -m 'amène libcomplex-1.0 dans la branche principale'
…
```

Nous extrayons ensuite la branche principale de notre projet, qui inclut désormais une copie de la première livraison fournisseur, et nous nous mettons au travail pour personnaliser le code de libcomplex. Avant même d'en avoir pris conscience, notre version modifiée de libcomplex est complètement intégrée dans notre programme de calcul [6].

Quelques semaines plus tard, les développeurs de libcomplex publient une nouvelle version de leur bibliothèque, la version 1.1, qui contient des fonctionnalités dont nous avons besoin. Nous aimerions pouvoir utiliser cette nouvelle version, sans toutefois perdre les évolutions que nous avons apportées à la version existante. En gros, ce que nous voudrions faire c'est remplacer notre version actuelle de libcomplex, la 1.0, par une copie de libcomplex 1.1 et ensuite ré-appliquer les modifications que nous avions effectuées précédemment sur cette bibliothèque à la nouvelle version. Mais, en fait, nous allons aborder le problème sous un autre angle, en appliquant les changements apportés à libcomplex entre les versions 1.0 et 1.1 à notre copie modifiée de celle-ci.

Pour effectuer cette mise à niveau, nous allons extraire une copie de notre branche fournisseur et remplacer le code du répertoire `actuel` par le nouveau code source de libcomplex 1.1. Nous copions en fait littéralement de nouveaux fichiers par-dessus des fichiers existants, par exemple en extrayant le contenu de l'archive de libcomplex 1.1 à l'endroit où se trouvent nos fichiers et dossiers. L'objectif ici est d'aboutir à ce que notre répertoire `actuel` ne contienne que le code de libcomplex 1.1 et de s'assurer que la totalité de ce code est suivi en versions. Ah, et puis nous voulons faire ceci avec le moins possible de perturbations liées à l'historique de la gestion de versions.

Après avoir remplacé le code de la 1.0 par le code de la 1.1, **svn status** liste les fichiers ayant des modifications locales et peut-être aussi des fichiers non suivis en versions. Si nous avons fait ce que nous étions censés faire, les fichiers non suivis en versions sont uniquement de nouveaux fichiers introduits par la version 1.1 de libcomplex ; nous lançons donc **svn add** sur ces fichiers pour les inclure dans la gestion de versions. Si le code de la 1.1 ne contient plus certains fichiers qui étaient dans l'arborescence de la 1.0, il sera peut-être difficile de les repérer ; il vous faudrait comparer les deux arborescences avec un outil extérieur et ensuite faire un **svn delete** sur tous les fichiers présents en 1.0 mais pas en 1.1 (bien qu'il soit peut-être parfaitement acceptable de laisser traîner ces mêmes fichiers, inutilisés, dans l'ombre). Finalement, une fois que notre copie de travail de `actuel` ne contient plus que le code de libcomplex 1.1, nous pouvons propager les modifications que nous avons faites pour lui donner cet aspect.

[6] Et ne contient pas le moindre bogue, cela va de soi !

Notre branche `actuel` contient désormais la nouvelle livraison fournisseur. Nous étiquetons donc la nouvelle version en 1.1 (de la même façon que nous avions précédemment étiqueté la livraison fournisseur de la version 1.0) et ensuite nous fusionnons les différences entre les étiquettes de la version précédente et de la nouvelle version vers notre branche de développement principale :

```
$ cd copies-de-travail/calc
$ svn merge http://svn.exemple.com/depot/fournisseur/libcomplex/1.0      \
            http://svn.exemple.com/depot/fournisseur/libcomplex/actuel  \
            libcomplex
… # résoudre tous les conflits entre leurs modifications et nos modifications
$ svn commit -m 'fusion de libcomplex-1.1 vers la branche principale'
…
```

Dans le cas le plus trivial, la nouvelle version de notre outil tierce ressemblerait à la version précédente, du point de vue des fichiers et dossiers. Aucun des fichiers sources de libcomplex n'aurait été effacé, renommé, ou déplacé ; la nouvelle version ne contiendrait que des modifications textuelles par rapport à la précédente. Dans l'idéal, nos modifications s'appliqueraient proprement à la nouvelle version de la bibliothèque, sans la moindre complication ou conflit.

Mais les choses ne sont pas toujours aussi simples et, en fait, il arrive assez fréquemment que des fichiers sources changent d'emplacement d'une version à l'autre d'un logiciel. Ceci complique la tâche de s'assurer que nos modifications sont toujours valides pour la nouvelle version du code, et les choses peuvent rapidement dégénérer, jusqu'au point où nous pouvons être forcés de reporter manuellement nos évolutions maison dans la nouvelle version. Une fois que Subversion connaît l'historique d'un fichier source donné, incluant tous ses emplacements précédents, la procédure pour incorporer la nouvelle version de la bibliothèque est assez simple. Mais c'est à nous qu'incombe la responsabilité d'indiquer à Subversion de quelle manière l'agencement des fichiers sources a changé d'une livraison fournisseur à une autre.

svn_load_dirs.pl

Les livraisons fournisseur comportant plus de quelques suppressions, ajouts et déplacements compliquent le processus de mise à niveau à chaque version successive des données tierces. Subversion fournit donc le script **svn_load_dirs.pl** pour faciliter ce processus. Ce script automatise les étapes importantes que nous avons mentionnées dans la procédure générale de gestion des branches fournisseurs afin de minimiser les erreurs. Vous êtes toujours responsable de l'utilisation des commandes **svn merge** pour fusionner les nouvelles versions des données tierces vers votre branche de développement principale, mais **svn_load_dirs.pl** peut vous aider à parvenir à cette étape plus rapidement et plus facilement.

En bref, **svn_load_dirs.pl** est une version améliorée de **svn import** qui possède plusieurs caractéristiques importantes :

- On peut le lancer n'importe quand, dans le but d'amener un répertoire existant du dépôt à refléter exactement un répertoire extérieur, en effectuant toutes les opérations d'ajouts et de suppressions et, en option, de déplacements.
- Il prend soin de séries compliquées d'opérations entre lesquelles Subversion a besoin d'une propagation intermédiaire, telles qu'avant de renommer un fichier ou un dossier pour la deuxième fois.
- En option, il peut étiqueter les nouveaux dossiers importés.
- En option, il peut ajouter des propriétés arbitraires aux fichiers et dossiers qui correspondent à une expression régulière.

svn_load_dirs.pl prend trois paramètres obligatoires. Le premier paramètre est l'URL du répertoire de base de Subversion à modifier. Ce paramètre est suivi par l'URL, relative au premier paramètre, dans laquelle la livraison fournisseur sera importée. Enfin, le troisième paramètre est le dossier local à importer. En utilisant notre exemple précédent, une exécution type de **svn_load_dirs.pl** donne :

```
$ svn_load_dirs.pl http://svn.exemple.com/depot/fournisseur/libcomplex \
                   actuel                                              \
                   /chemin/vers/libcomplex-1.1
…
```

Vous pouvez indiquer que vous aimeriez que **svn_load_dirs.pl** étiquette la nouvelle livraison fournisseur en passant l'option `-`

`t` et en spécifiant un nom d'étiquette. Cette étiquette est aussi une URL relative au premier paramètre du programme.

```
$ svn_load_dirs.pl -t libcomplex-1.1                                  \
                   http://svn.exemple.com/depot/fournisseur/libcomplex \
                   actuel                                             \
                   /chemin/vers/libcomplex-1.1
…
```

Lorsque vous lancez **svn_load_dirs.pl**, il examine le contenu de votre livraison fournisseur existante, `actuel`, et le compare à la nouvelle livraison fournisseur. Dans le cas le plus trivial, aucun fichier n'est présent dans une version sans l'être dans l'autre et le script effectue le nouvel import sans incident. Cependant, s'il y a des divergences dans l'agencement des fichiers entre les versions, **svn_load_dirs.pl** vous demande comment résoudre ces différences. Par exemple, vous avez l'opportunité d'indiquer au script que vous savez que le fichier `math.c` de la version 1.0 de libcomplex a été renommé en `arithmetique.c` dans libcomplex 1.1. Toutes les divergences qui ne sont pas liées à des renommages sont traitées comme des ajouts et des suppressions classiques.

Le script peut également prendre en compte un fichier de configuration séparé, permettant de spécifier des propriétés sur des fichiers et dossiers, correspondants à une expression régulière, qui vont être *ajoutés* au dépôt. Ce fichier de configuration est indiqué à **svn_load_dirs.pl** en utilisant l'option `-p` en ligne de commande. Chaque ligne du fichier de configuration est un ensemble de deux ou quatre valeurs délimitées par des espaces : une expression régulière du style Perl à laquelle comparer le chemin ajouté, un mot clé de contrôle (soit `break` soit `cont`) et ensuite, en option, un nom de propriété et une valeur.

```
\.png$              break   svn:mime-type   image/png
\.jpe?g$            break   svn:mime-type   image/jpeg
\.m3u$              cont    svn:mime-type   audio/x-mpegurl
\.m3u$              break   svn:eol-style   LF
.*                  break   svn:eol-style   native
```

Pour chaque chemin ajouté, les modifications de propriétés configurées dont l'expression régulière correspond au chemin sont appliquées dans l'ordre, sauf si le terme de contrôle est `break` (ce qui signifie qu'aucune autre modification de propriété ne doit être appliquée à ce chemin). Si le terme de contrôle est `cont` (abréviation de continuer), la comparaison continue avec la ligne suivante du fichier de configuration.

Toute espace faisant partie de l'expression régulière, du nom de la propriété ou de la valeur de la propriété doit être entourée d'apostrophes ou de guillemets. Vous pouvez banaliser les guillemets et apostrophes qui ne sont pas utilisés pour entourer une espace en les faisant précéder d'une barre oblique inversée (ou « antislash » : \). L'antislash ne banalise que les guillemets et apostrophes pendant le traitement du fichier de configuration, ce n'est donc pas la peine de protéger d'autres caractères au-delà de ce qui est nécessaire pour l'expression régulière.

Résumé

Nous avons traité de nombreux sujets dans ce chapitre. Nous avons présenté les concepts d'étiquettes et de branches et montré comment Subversion implémente ces concepts en copiant des répertoires avec la commande **svn copy**. Nous avons expliqué comment utiliser **svn merge** pour copier des modifications d'une branche à l'autre ou pour revenir en arrière sur des modifications non-satisfaisantes. Nous avons étudié l'utilisation de **svn switch** pour créer des copies de travail mixtes, pointant vers des emplacements variés d'un dépôt. Et nous avons évoqué la façon dont on peut gérer l'organisation et le cycle de vie des branches dans un dépôt.

Tâchez de garder en mémoire la devise de Subversion : les branches et les étiquettes ne coûtent quasiment rien. Donc n'ayez pas peur de les utiliser quand vous en avez besoin !

En guise de rappel utile de toutes les opérations dont nous avons parlé, voici un tableau de référence très pratique, à consulter lorsque vous commencerez à utiliser des branches.

Tableau 4.1. Commandes de gestion des branches et des fusions

Action	Commande
Créer une branche ou une étiquette	**svn copy** ***URL1 URL2***

Action	Commande
Faire pointer une copie de travail vers une branche ou une étiquette	`svn switch URL`
Synchroniser une branche avec le tronc	`svn merge trunkURL; svn commit`
Voir l'historique des fusions ou les ensembles de modifications susceptibles d'être fusionnés	`svn mergeinfo target --from-source=URL`
Fusionner une branche vers le tronc	`svn merge --reintegrate branchURL; svn commit`
Fusionner une modification précise	`svn merge -c REV URL; svn commit`
Fusionner un ensemble de modifications	`svn merge -r REV1:REV2 URL; svn commit`
Empêcher qu'une modification ne soit fusionnée automatiquement	`svn merge -c REV --record-only URL; svn commit`
Prévisualiser une fusion	`svn merge URL --dry-run`
Abandonner une fusion	`svn revert -R .`
Ressusciter un élément de l'historique	`svn copy URL@REV localPATH`
Revenir en arrière sur une modification déjà propagée	`svn merge -c -REV URL; svn commit`
Examiner l'historique en tenant compte des informations de fusion	`svn log -g; svn blame -g`
Créer une étiquette à partir d'une copie de travail	`svn copy . tagURL`
Réorganiser une branche ou une étiquette	`svn mv URL1 URL2`
Supprimer une branche ou une étiquette	`svn rm URL`

Chapitre 5. Administration d'un dépôt

Le dépôt Subversion est le centre de stockage de toutes vos données suivies en versions. Ainsi, il est *de facto* l'objet de toute l'attention et de tous les soins de l'administrateur. Bien que ce soit un élément ne nécessitant pas énormément de maintenance, il est important de comprendre comment le configurer et le surveiller de manière à éviter d'éventuels problèmes et à résoudre proprement ceux qui se présentent.

Dans ce chapitre, nous expliquons comment créer et configurer un dépôt Subversion. Nous abordons également la maintenance du dépôt, en donnant des exemples d'utilisation des outils **svnlook** et **svnadmin** fournis avec Subversion. Nous étudions quelques questions et erreurs communes et nous donnons des conseils sur l'organisation des données dans le dépôt.

Si vous n'envisagez pas d'utiliser un dépôt Subversion autrement qu'en simple utilisateur des données (c'est-à-dire en utilisant un client Subversion), vous pouvez sauter ce chapitre. Cependant, si vous êtes (ou si vous êtes appelé à être) l'administrateur d'un dépôt[1], ce chapitre est fait pour vous.

Définition d'un dépôt Subversion

Avant d'aborder le vaste sujet de l'administration d'un dépôt, définissons plus précisément ce qu'est un dépôt. À quoi ressemble-t-il ? Que ressent-il ? Est-ce qu'il préfère son thé chaud ou glacé, sucré, avec une tranche de citron ? En tant qu'administrateur, vous vous devez de comprendre de quoi est composé un dépôt, à la fois au niveau du système d'exploitation (à quoi ressemble le dépôt et comment il réagit vis-à-vis des outils autres que Subversion) et au niveau logique de l'organisation des données (comment elles sont représentées *à l'intérieur* du dépôt).

Du point de vue d'un explorateur de fichiers classique (comme Windows Explorer) ou d'un outil de navigation du système de fichiers en ligne de commande, un dépôt Subversion n'est rien d'autre qu'un répertoire contenant plein de choses. Il y a des sous-répertoires avec des fichiers de configuration lisibles par un humain, des sous-répertoires avec des fichiers de données binaires déjà bien moins lisibles, etc. À l'instar d'autres parties de Subversion, la modularité est une préoccupation majeure et l'organisation hiérarchique prévaut sur le bazar. Un coup d'œil rapide dans un dépôt ordinaire est suffisant pour obtenir la liste des composants essentiels d'un dépôt :

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

Effectuons un survol rapide de ce que nous voyons dans ce répertoire (ne vous inquiétez pas si vous ne comprenez pas tous les termes employés, ils sont expliqués dans ce chapitre ou ailleurs dans ce livre) :

conf
: Un répertoire contenant des fichiers de configuration.

dav
: Un répertoire à disposition de `mod_dav_svn` pour y stocker ses informations privées.

db
: Le magasin de données pour toutes vos données suivies en versions.

format
: Un fichier contenant uniquement le numéro de version de l'organisation du dépôt.

hooks
: Un répertoire plein de modèles de procédures automatiques (et les procédures automatiques elles-mêmes, une fois installées).

locks

[1] Cela peut sembler prestigieux et noble, mais il s'agit juste en fait d'une personne intéressée par le monde mystérieux qui se cache derrière la copie de travail que chacun détient.

Un répertoire fait pour les fichiers de verrous du dépôt Subversion, utilisé pour gérer les accès au dépôt.

README.txt
Un fichier qui ne fait qu'informer son lecteur qu'il est tombé sur un dépôt Subversion.

Bien sûr, quand on y accède via les bibliothèques Subversion, cet ensemble de fichiers et de répertoires se transforme en un système de fichiers suivis en versions virtuel, complet et comportant une gestion des événements personnalisable. Ce système de fichiers possède ses propres notions de répertoires et de fichiers, très similaires aux notions des systèmes de fichiers réels (tels que NTFS, FAT32, ext3, etc.). Mais c'est un système de fichiers spécial : il base ces répertoires et ces fichiers sur les révisions, conservant l'historique de tous les changements effectués de manière sûre et pérenne. C'est là que la totalité de vos données suivies en versions réside.

Stratégies de déploiement d'un dépôt

En grande partie grâce à la conception épurée du dépôt Subversion et des technologies sous-jacentes, il est particulièrement aisé de créer et configurer un dépôt. Il y a quelques choix préliminaires à faire mais l'essentiel du travail de création et de configuration d'un dépôt Subversion est simple et convivial, facilement reproductible si vous êtes amené à effectuer des installations multiples.

Voici quelques questions à se poser avant toute chose :

- Quelles données vont être hébergées dans le dépôt (ou les dépôts) et quelle en sera l'organisation ?
- Où sera placé le dépôt et comment les utilisateurs y accéderont-ils ?
- De quels types de contrôle d'accès et de notifications d'événements avez-vous besoin ?
- Quel type de magasin de données désirez-vous utiliser ?

Dans cette section, nous essayons de vous aider à répondre à ces questions.

Stratégies d'organisation d'un dépôt

Bien que Subversion vous permette de déplacer des fichiers et des répertoires suivis en versions sans perte d'information et qu'il fournisse même des outils pour déplacer des ensembles complets de données suivies en version d'un dépôt à un autre, ces opérations peuvent perturber le travail des autres collaborateurs qui accèdent souvent au dépôt et qui s'attendent à trouver chaque chose à sa place. Ainsi, avant de créer un nouveau dépôt, essayez de vous projeter un peu dans le futur ; préparez à l'avance le passage de vos données sous gestion de versions. Cette réflexion sur la manière d'organiser vos données dans le dépôt vous évitera de futurs et nombreux maux de tête.

Supposons qu'en tant qu'administrateur d'un dépôt, vous êtes responsable de l'administration du système de gestion de versions pour plusieurs projets. La première décision à prendre est de choisir entre un seul dépôt pour tous les projets et un dépôt par projet, ou bien un compromis entre ces deux solutions.

Un seul dépôt pour tous les projets offre des avantages, ne serait-ce que pour la maintenance unifiée. Un seul dépôt signifie qu'il n'y a qu'un seul jeu de procédures automatiques, une seule sauvegarde à gérer, un seul jeu d'opérations de déchargement et de chargement à effectuer si la nouvelle version de Subversion est incompatible avec l'ancienne version, etc. Vous pouvez également déplacer facilement des données entre les projets, sans perdre l'historique de ces informations.

Les inconvénients à utiliser un seul dépôt sont que les différents projets auront certainement des besoins différents en termes de gestion des événements, comme la notification par e-mail des propagations à des listes d'adresses différentes ou des définitions différentes de ce qui constitue une propagation légitime. Bien sûr, ce ne sont pas des problèmes insurmontables — cela implique juste que vos procédures automatiques doivent tenir compte de l'organisation du dépôt dans lequel elles sont invoquées plutôt que de considérer que l'ensemble du dépôt est associé à un seul groupe d'utilisateurs. Rappelez-vous également que Subversion utilise des numéros de révisions globaux au dépôt. Bien que ces numéros ne possèdent pas de pouvoirs magiques particuliers, certaines personnes n'aiment pas voir le numéro de révision augmenter alors qu'elles n'ont pas touché à leur propre projet[2].

[2]Que ce soit par ignorance ou par la définition absurde de métriques de développement, les numéros globaux de révision sont craints alors qu'ils sont bien peu de chose et *certainement pas* à prendre en considération quand vous décidez de l'agencement de vos projets et de vos dépôts.

On peut aussi adopter une approche intermédiaire. Par exemple, les projets peuvent être regroupés par thème. Vous pouvez avoir quelques dépôts, avec une poignée de projets dans chaque dépôt. Ainsi, les projets susceptibles de partager des données le font aisément et les développeurs sont tenus au courant des avancées des projets en relation avec les leurs par le biais des nouvelles révisions du dépôt.

Une fois l'organisation des dépôts définie, il faut se préoccuper de la hiérarchie des répertoires à l'intérieur des dépôts eux-mêmes. Comme Subversion utilise de simples copies de répertoires pour créer les branches et les étiquettes (voir le Chapitre 4, *Gestion des branches*), la communauté Subversion recommande de choisir un endroit dans le dépôt pour la *racine* de chaque projet (le répertoire dont la sous-arborescence contient toutes les données relatives à un projet) et d'y placer trois sous-répertoires : `trunk` (« tronc » en français), le répertoire qui héberge les principaux développements du projet ; `branches`, le répertoire dans lequel seront créées les différentes variations de la ligne de développement principale ; et `tags` (« étiquettes » en français), qui contient un ensemble d'instantanés de l'arborescence (les instantanés sont créés, voire détruits, mais jamais modifiés)[3].

Par exemple, votre dépôt peut ressembler à ceci :

```
/
   calculatrice/
      trunk/
      tags/
      branches/
   calendrier/
      trunk/
      tags/
      branches/
   tableur/
      trunk/
      tags/
      branches/
 …
```

Veuillez noter que l'emplacement du projet dans le dépôt n'est pas important. Si vous n'avez qu'un seul projet par dépôt, il est logique de placer la racine du projet à la racine du dépôt correspondant. Si vous avez plusieurs projets, vous voulez peut-être les classer par groupes dans des sous-répertoires communs du dépôt, en fonction des objectifs ou du code à partager par exemple, ou tout simplement en les groupant par ordre alphabétique. Voici un exemple :

```
/
   utilitaires/
      calculatrice/
         trunk/
         tags/
         branches/
      calendrier/
         trunk/
         tags/
         branches/
    …
   bureautique/
      tableur/
         trunk/
         tags/
         branches/
    …
```

Organisez votre dépôt comme vous le sentez. Subversion n'a aucune exigence en la matière — pour lui, un répertoire est un répertoire. L'objectif est d'avoir une organisation qui réponde aux besoins des collaborateurs des différents projets.

Cependant, par souci de transparence, nous indiquons une autre organisation également très répandue. Dans cette organisation, les répertoires `trunk`, `tags` et `branches` sont situés à la racine du dépôt et les projets sont placés dans des sous-répertoires juste en dessous, comme ceci :

[3] Le trio `trunk`, `tags` et `branches` est quelquefois appelé « les répertoires TTB » (*the TTB directories* en anglais).

```
/
    trunk/
       calculatrice/
       calendrier/
       tableur/
     …
    tags/
       calculatrice/
       calendrier/
       tableur/
     …
    branches/
       calculatrice/
       calendrier/
       tableur/
     …
```

Il n'y a rien d'incorrect dans une telle organisation, mais elle peut ne pas être très intuitive pour vos utilisateurs. En particulier dans des situations complexes avec plusieurs projets et un grand nombre d'utilisateurs, dont la plupart ne connaissent qu'un ou deux projets du dépôt. Mais cette approche « plusieurs projets par branche » a tendance à favoriser l'ouverture de chaque projet sur les autres et pousse à envisager l'ensemble des projets comme une seule entité. Cela reste un problème social. Nous aimons l'organisation suggérée au début pour des raisons purement pratiques : il est plus facile de faire des requêtes (ou des modifications, des migrations) sur l'historique complet d'un projet quand une sous-arborescence du dépôt contient l'ensemble des données (passé, présent, étiquettes et branches) de ce projet et elles seules.

Stratégies d'hébergement d'un dépôt

Avant de créer votre dépôt Subversion, vous devez vous demander où il va résider. Cette question est fortement liée à une myriade d'autres questions telles que : qui sont les utilisateurs (sont-ils à l'intérieur de votre réseau interne, derrière le pare-feu de votre entreprise, ou bien s'agit-il de n'importe qui, n'importe où sur Internet ?), comment les utilisateurs accèdent au dépôt (via un serveur Subversion ou directement), quels autres services vous fournissez autour de Subversion (une interface pour navigateur Web, des notifications par email des propagations, etc.), quelle est votre politique de sauvegarde et ainsi de suite.

Le choix et la configuration du serveur sont abordés au Chapitre 6, *Configuration du serveur*, mais nous voulons signaler dès maintenant que certains choix pour l'une ou l'autre de ces questions ont des implications sur l'endroit où implémenter votre serveur. Par exemple, certains scénarios de déploiement nécessitent, pour plusieurs ordinateurs, l'accès au dépôt via un système de fichiers distant et, dans ce cas, le choix du type de magasin de données n'en est plus un, puisqu'un seul type de magasin de données convient dans ce scénario.

Décrire l'ensemble des possibilités de déploiement de Subversion est impossible et n'est pas l'objectif de ce livre. Nous vous encourageons simplement à évaluer vos choix avec ces quelques pages ainsi qu'avec d'autres ressources en guise de référence afin de planifier correctement les opérations.

Choix du magasin de données

Depuis la version 1.1, Subversion offre deux types de stockage interne pour le magasin de données[4] utilisé par le dépôt. Un des types de magasin de données conserve tout dans une base de données Berkeley DB (ou BDB) ; les dépôts qui utilisent ce type de magasin sont qualifiés de « dépôts gérés par BDB » ou « dépôts BDB ». L'autre type de magasin stocke les données dans des fichiers ordinaires, en utilisant un format particulier. Les développeurs de Subversion ont pris l'habitude d'appeler ce type de stockage *FSFS*[5] — une implémentation d'un système de fichiers suivis en versions qui utilise le système de fichiers natif du système d'exploitation directement plutôt que par l'intermédiaire d'une bibliothèque de gestionnaire de base de données ou toute autre couche d'abstraction.

Une comparaison des dépôts utilisant Berkeley DB et FSFS fait l'objet du Tableau 5.1, « Comparaison des magasins de données de dépôts ».

[4] NdT : souvent désigné par *backend* en anglais (sans équivalent en français) ou, ce qui peut être source de confusion, « le système de fichiers (suivi en versions) »
[5] Souvent prononcé « feuzz-feuzz » si Jack Repenning en parle (ce livre, cependant, suppose que le lecteur pense « eff-ess-eff-ess »).

Tableau 5.1. Comparaison des magasins de données de dépôts

Catégorie	Fonctionnalité	Berkeley DB	FSFS
Fiabilité	Intégrité des données	Très fiable quand déployé correctement ; Berkeley DB 4.4 apporte l'auto-restauration	Les vieilles versions avaient quelques bogues (rarement démontrés) qui détruisaient des données
	Sensibilité aux interruptions	Forte ; les « plantages » et les problèmes de droits peuvent laisser la base de données dans un état instable, nécessitant le recours aux procédures de restauration issues de la journalisation	Quasiment insensible
Accessibilité	Utilisable depuis un montage en lecture seule	Non	Oui
	Stockage indépendant de la plate-forme	Non	Oui
	Utilisable sur des systèmes de fichiers en réseau	Généralement non	Oui
	Gestion des droits pour les groupes	Sensible aux problèmes de umask de l'utilisateur ; c'est mieux si un seul utilisateur y accède	Contourne les problèmes de umask
Extensibilité	Utilisation des disques sur le dépôt	Plus grande (surtout si les fichiers de journalisation ne sont pas purgés)	Plus faible
	Nombre de révisions	Base de données, pas de problème	De vieux systèmes de fichiers fonctionnent moins bien lorsqu'il y a plusieurs milliers d'entrées dans un seul répertoire
	Répertoires avec beaucoup de fichiers	Plus lent	Plus rapide
Performances	Extraire la dernière révision	Pas de différence significative	Pas de différence significative
	Grosses propagations	Globalement plus lent, mais mais cette lenteur est répartie sur toute la durée de la propagation	Globalement plus rapide, mais le délai de finalisation peut amener le client à considérer que sa requête a expiré avant qu'il ne reçoive la réponse

Chaque type de magasin de données a ses avantages et ses inconvénients. Aucun n'est plus « officiel » que l'autre, même si le nouveau FSFS est le magasin par défaut depuis Subversion 1.2. Les deux sont suffisamment fiables pour y stocker vos données suivies en versions en toute confiance. Mais comme l'indique le Tableau 5.1, « Comparaison des magasins de données de dépôts », FSFS est un peu plus souple à déployer. Plus de souplesse signifie que vous devez y mettre un peu plus du vôtre pour faire des erreurs lors du déploiement. C'est pourquoi, en plus du fait que ne pas utiliser Berkeley DB permet de compter un composant de moins dans le système, aujourd'hui, presque tout le monde utilise FSFS lors de la création de nouveaux dépôts.

Heureusement, la plupart des programmes qui accèdent aux dépôts Subversion ignorent royalement quel type de magasin de données est utilisé. Et vous n'êtes même pas prisonnier de votre premier choix de magasin : si vous changez d'avis plus tard, Subversion offre différentes façons de migrer les données de votre dépôt dans un autre dépôt utilisant un magasin de données différent. Nous en reparlons plus loin dans ce chapitre.

Les paragraphes suivants abordent plus en détail les différents types de magasins de données disponibles.

Berkeley DB

Lors de la conception initiale de Subversion, les développeurs ont décidé d'utiliser le gestionnaire de bases de données Berkeley DB pour tout un tas de raisons, entre autres sa licence Open Source, son support des transactions, sa fiabilité, ses performances, la simplicité de son interface de programmation (API), le bon support des processus légers (threads), le support des curseurs, etc.

Le gestionnaire de bases de données Berkeley DB apporte un support réel des transactions (c'est peut-être sa fonctionnalité la plus puissante). Si de nombreux processus accèdent en même temps au dépôt, ils n'ont pas à se soucier d'éventuelles corruptions de données de la part des autres processus. L'isolement créé par le système de transaction est tel que, pour une opération donnée, Subversion voit une base de données statique — pas une base de données en perpétuel changement en raison des autres processus — et peut donc prendre des décisions à partir de cette perspective. Si la décision entraîne un conflit avec ce que fait un autre processus, l'opération complète est annulée, tout se passe comme si elle n'avait jamais eu lieu et Subversion essaie une nouvelle fois son opération sur la base de données mise à jour (qui apparaît toujours statique).

Une autre fonctionnalité phare du gestionnaire de bases de données Berkeley DB est la *sauvegarde à chaud* — la capacité de sauvegarder l'environnement de la base de données sans la couper du réseau. Nous voyons comment réaliser une sauvegarde de votre dépôt plus loin dans ce chapitre (dans la section intitulée « Sauvegarde d'un dépôt »), mais le bénéfice de pouvoir faire des copies opérationnelles de vos dépôts sans interruption de service doit vous sauter aux yeux.

Le gestionnaire de bases de données Berkeley DB est aussi très fiable quand il est utilisé correctement. Subversion utilise les fonctions de journalisation du gestionnaire de bases de données Berkeley DB, ce qui veut dire que la base de données consigne d'abord dans un fichier de journalisation situé sur disque chaque modification qu'elle s'apprête à effectuer, puis effectue la modification elle-même. Cela garantit que si quelque chose se passe mal, le gestionnaire de base de données peut revenir à un *point de contrôle* précédent — un point précis des fichiers de journalisation dont il sait qu'il n'est pas corrompu — et rejouer les transactions jusqu'à ce que les données soient dans un état opérationnel. Voir la section intitulée « Gestion de l'espace disque » plus loin dans ce chapitre pour plus de détails sur les fichiers de journalisation Berkeley DB.

Mais chaque médaille à son revers et nous devons vous avertir de quelques limitations du gestionnaire de bases de données Berkeley DB. Premièrement, les environnements du gestionnaire de bases de données Berkeley DB ne sont pas portables. Vous ne pouvez pas simplement copier un dépôt Subversion qui a été créé sur un système Unix vers un système Windows et espérer qu'il fonctionne. Bien que la majeure partie de la base de données Berkeley DB soit indépendante de l'architecture, d'autres aspects de l'environnement ne le sont pas. Deuxièmement, Subversion utilise le gestionnaire de bases de données Berkeley DB de telle façon que cela ne fonctionne pas sur un système Windows 95/98 : si vous avez besoin d'héberger un dépôt géré par BDB sur une machine Windows, adoptez Windows 2000 ou plus.

Alors que le gestionnaire de bases de données Berkeley DB prétend fonctionner correctement sur un système de fichiers en réseau, pour peu que celui-ci respecte des caractéristiques particulières[6], la plupart des systèmes de fichiers en réseau et des systèmes dédiés *n'atteignent pas* ces pré-requis. Et en aucun cas il ne vous est possible de partager ce dépôt sur un système de fichiers en réseau entre plusieurs clients (alors que c'est quand même l'intérêt principal d'un dépôt accessible sur un partage réseau).

Si vous tentez d'utiliser le gestionnaire de bases de données Berkeley DB sur un système de fichiers en réseau non compatible, les résultats sont imprévisibles : vous vous apercevrez peut-être immédiatement de mystérieuses erreurs, mais il se peut qu'il se passe des mois avant que vous ne découvriez que votre base de données de dépôt est corrompue. Songez sérieusement à utiliser un magasin FSFS pour les dépôts qui doivent être hébergés sur un partage réseau.

Finalement, parce que la bibliothèque du gestionnaire de bases de données Berkeley DB est directement incluse dans Subversion, elle est plus sensible aux interruptions qu'une base de données relationnelle classique. La plupart des systèmes SQL, par exemple, disposent d'un processus serveur dédié qui coordonne tous les accès aux tables. Si un programme qui accède aux tables plante pour une raison ou une autre, le processus serveur de la base de données s'en aperçoit et fait le ménage. Et comme le processus serveur est le seul processus accédant réellement aux tables, les applications n'ont pas à se soucier des conflits de droits. Cependant, ce n'est pas le cas avec le gestionnaire de bases de données Berkeley DB. Subversion (et les programmes utilisant les bibliothèques de Subversion) accèdent aux tables directement, ce qui veut dire que le plantage d'un programme peut laisser la base de données dans un état temporairement incohérent et inaccessible. Quand cela arrive, un administrateur doit demander au gestionnaire de bases de données Berkeley DB de revenir à un point de contrôle, ce qui est assez ennuyeux. D'autres incidents peuvent faire planter la base de données, comme des conflits entre programmes pour la

[6] Berkeley DB requiert que le système de fichiers sous-jacent implémente strictement la sémantique POSIX sur les verrous et, plus important encore, la possibilité de projeter les fichiers directement en mémoire vive.

possession ou les droits sur les fichiers de la base de données.

La version 4.4 du gestionnaire de bases de données Berkeley DB permet à Subversion (version 1.4 ou plus) de restaurer un environnement Berkeley DB automatiquement et de manière transparente en cas de besoin. Quand un processus Subversion se greffe sur l'environnement d'un dépôt Berkeley DB, il utilise un mécanisme d'enregistrement pour détecter d'éventuels problèmes de déconnexion antérieurs , effectue les restaurations nécessaires puis passe à la suite comme si de rien n'était. Cela n'élimine pas complètement les plantages du dépôt, mais les actions humaines nécessaires pour revenir à une situation normale sont considérablement réduites.

Ainsi, bien qu'un dépôt Berkeley DB soit rapide et capable de monter en puissance, il faut privilégier une utilisation par un seul processus serveur tournant avec une identité unique (comme les serveurs Apache **httpd** ou **svnserve** : voir le Chapitre 6, *Configuration du serveur*) à un accès par de nombreux utilisateurs via des URL `file://` ou `svn+ssh://`. Si de multiples utilisateurs doivent avoir accès à un dépôt Berkeley DB, lisez la section intitulée « Accès au dépôt par plusieurs méthodes » plus loin dans ce chapitre.

FSFS

Mi-2004, un deuxième type de stockage pour le dépôt, qui ne fait pas appel à une base de données, a fait son apparition. Un dépôt FSFS stocke les changements associés à une révision dans un fichier unique, ce qui fait que l'ensemble des révisions du dépôt se trouvent dans un seul sous-répertoire rempli de fichiers numérotés. Les transactions sont créées, en tant que fichiers individuels, dans des sous-répertoires séparés. Une fois la transaction terminée, le fichier de transaction est renommé et placé dans le répertoire des révisions, ce qui garantit l'atomicité des propagations. Et puisqu'un fichier de révision est permanent et non modifiable, le dépôt peut également être sauvegardé « à chaud » comme un dépôt BDB.

Les fichiers de révision FSFS décrivent, pour une révision donnée, la structure des répertoires, le contenu des fichiers et les deltas entre les fichiers et les autres arborescences de révisions. Contrairement à une base de données Berkeley DB, le format de stockage est portable, transférable entre différents systèmes d'exploitation et n'est pas sensible à l'architecture CPU. Comme il n'y a pas de journalisation ou de fichiers en mémoire partagée, le dépôt est accessible sans risque via un partage de fichiers sur le réseau ou depuis un environnement en lecture seule. L'absence des en-têtes liés à une base de données réduit aussi quelque peu la taille globale du dépôt.

FSF diffère également du point de vue des performances. Quand vous propagez un répertoire comptant un nombre de fichiers très élevé, FSFS est capable d'ajouter plus rapidement les éléments du répertoire. D'un autre côté, FSFS écrit la dernière version d'un fichier sous forme de delta par rapport à la version précédente, par conséquent une extraction de l'arborescence la plus récente est un peu plus lente que l'obtention des fichiers entiers stockés dans la révision HEAD d'une base de données Berkeley DB. FSFS est également plus long lors de la fin de la propagation, ce qui peut amener le client à considérer, dans des cas extrêmes, que sa requête a expiré avant qu'il ne reçoive la réponse.

La différence la plus importante reste quand même la résistance aux plantages lorsque quelque chose va mal. Si un processus qui utilise une base de données Berkeley DB rencontre un problème de droits ou plante soudainement, la base de données risque de se retrouver dans un état instable jusqu'à ce qu'un administrateur la restaure. Si la même chose arrive à un processus utilisant un dépôt FSFS, le dépôt n'est en rien affecté. Au pire, quelques données de transaction sont égarées.

Le seul véritable argument contre FSFS est sa relative immaturité face à Berkeley DB. Berkeley DB a une histoire de plusieurs années, avec une équipe de développement dédiée et, aujourd'hui, il est adossé à la toute-puissante marque Oracle[7]. FSFS est de conception plus récente. Avant la version 1.4 de Subversion apparaissaient de temps en temps quelques bogues assez sérieux concernant l'intégrité des données qui, bien que n'arrivant que dans de très rares cas, étaient bien réels. Ceci dit, FSFS est rapidement devenu le magasin de données de référence pour quelques-uns des plus vastes dépôts Subversion publics et privés et il promet de rendre globalement plus facile le passage à Subversion.

Création et configuration d'un dépôt

Dans ce chapitre (dans la section intitulée « Stratégies de déploiement d'un dépôt »), nous avons passé en revue quelques décisions importantes à prendre avant de créer et de configurer votre dépôt Subversion. Maintenant nous allons enfin mettre les mains dans le cambouis ! Dans cette section, nous voyons comment créer un dépôt Subversion et comment le configurer pour qu'il effectue des actions personnalisées lorsque certains événements ont lieu.

[7] Oracle a acheté Sleepycat et son logiciel vedette, Berkeley DB, le jour de la Saint Valentin 2006.

Création d'un dépôt

La création d'un dépôt Subversion est une tâche incroyablement simple. L'utilitaire **svnadmin**, fourni avec Subversion, dispose d'une sous-commande qui est justement destinée à cela (**svnadmin create**) :

```
$ # Créer un dépôt
$ svnadmin create /var/svn/depot
$
```

Cette commande crée un dépôt dans le répertoire `/var/svn/depot` avec le magasin de données par défaut. Avant la version 1.2 de Subversion, le choix par défaut était l'utilisation d'une base de données Berkeley DB ; maintenant c'est FSFS. Vous pouvez choisir explicitement le type de système de fichiers avec l'option `--fs-type` qui accepte comme argument soit `fsfs`, soit `bdb`.

```
$ # Créer un dépôt FSFS
$ svnadmin create --fs-type fsfs /var/svn/depot
$
```

```
$ # Créer un dépôt Berkeley DB
$ svnadmin create --fs-type bdb /var/svn/depot
$
```

Après l'exécution de cette simple commande, vous disposez d'un dépôt Subversion.

Le chemin en argument de **svnadmin** est juste un chemin classique du système de fichiers, pas une URL comme celles que le client **svn** utilise pour spécifier un dépôt. Les commandes **svnadmin** et **svnlook** sont toutes les deux considérées comme des utilitaires coté serveur : elles sont utilisées sur la machine qui héberge le dépôt pour examiner ou modifier certains aspects du dépôt et ne sont pas capables d'effectuer des actions via le réseau. Une erreur classique des nouveaux utilisateurs de Subversion est d'essayer de passer une URL (même « locale » comme `file://`) à ces deux programmes.

Dans le sous-répertoire `db/` de votre dépôt, vous trouvez l'implémentation du système de fichiers suivi en versions. Le nouveau système de fichiers suivi en versions de votre dépôt commence sa vie à la révision 0, qui est définie comme contenant le répertoire racine (`/`) et lui seul. Initialement, la révision 0 possède une seule propriété de révision, `svn:date`, dont la valeur est la date de création du dépôt.

Maintenant que vous disposez d'un dépôt, il est temps de le personnaliser.

Alors que certaines parties d'un dépôt Subversion sont conçues pour être examinées et modifiées « à la main » (comme les fichiers de configuration et les procédures automatiques), vous ne devez pas (et vous ne devriez pas avoir besoin de) modifier les autres parties « à la main ». L'outil **svnadmin** est censé être suffisant pour toutes les modifications à apporter à votre dépôt, mais vous pouvez également vous servir d'outils tiers (comme la suite d'outils Berkeley DB) pour configurer les parties adéquates du dépôt. Ne tentez *surtout pas* de manipuler manuellement l'historique du suivi de versions en touchant aux fichiers du magasin de données du dépôt !

Mise en place des procédures automatiques

Une *procédure automatique* (*hook* en anglais) est un programme activé par certains événements du dépôt, comme la création d'une nouvelle révision ou la modification d'une propriété non suivie en versions. Certaines procédures automatiques (appelées « pre hooks ») sont déclenchées avant l'opération sur le dépôt et permettent à la fois de rendre compte de ce qui va se passer et d'empêcher que cela se passe. D'autres procédures automatiques (appelées « post hooks ») sont déclenchées après la fin d'un événement et servent à effectuer des tâches de surveillance (mais pas de modification) du dépôt. Chaque procédure automatique reçoit suffisamment d'informations pour déterminer la nature de l'événement, les modifications proposées (ou effectuées) du dépôt et le nom d'utilisateur de la personne qui a déclenché l'événement.

Le sous-répertoire `hooks` contient, par défaut, des modèles pour diverses procédures automatiques :

```
$ ls depot/hooks/
post-commit.tmpl          post-unlock.tmpl  pre-revprop-change.tmpl
post-lock.tmpl            pre-commit.tmpl   pre-unlock.tmpl
post-revprop-change.tmpl  pre-lock.tmpl     start-commit.tmpl
$
```

Il y a un modèle pour chaque type de procédure automatique que le dépôt Subversion sait prendre en charge ; en examinant le contenu de ces modèles de scripts, vous pouvez voir ce qui déclenche le script et quelles données sont passées en paramètres. Vous trouvez également dans beaucoup de ces scripts des exemples d'utilisation permettant de réaliser des tâches récurrentes utiles, en conjonction avec d'autres programmes fournis avec Subversion. Concrètement, pour activer une procédure automatique, il suffit de placer dans le répertoire `depot/hooks` un programme ou un script exécutable, qui sera invoqué via le nom de la procédure automatique (comme **start-commit** pour le début d'une propagation ou **post-commit** pour la fin d'une propagation).

Sur les plates-formes Unix, cela veut dire fournir un programme ou un script (pouvant être un script shell, un programme Python, l'exécutable binaire d'un programme en C ou tout un tas d'autres choses) dont le nom est exactement le nom de la procédure automatique. Bien sûr, les modèles qui sont fournis ne le sont pas juste à titre d'information. Le moyen le plus facile pour mettre en place une procédure automatique sur les plates-formes Unix consiste tout simplement à copier le fichier du modèle adéquat vers un nouveau fichier qui n'aura pas l'extension `.tmpl`, d'adapter son contenu à votre environnement et de vous assurer qu'il est exécutable. Sous Windows, comme l'extension du fichier détermine s'il est exécutable ou non, vous devez fournir un programme dont la base du nom est le nom de la procédure automatique et dont l'extension est l'une de celles reconnue comme exécutable par Windows, comme `.exe` pour les programmes ou `.bat` pour les fichiers batch.

Pour des raisons de sécurité, le dépôt Subversion exécute les procédures automatiques avec un environnement vide — c'est-à-dire sans aucune variable d'environnement définie, même pas `$PATH` (ou `%PATH%` sous Windows). C'est ainsi que de nombreux administrateurs sont perplexes lorsque leurs programmes fonctionnent correctement à la main mais pas dans Subversion. Assurez-vous de définir explicitement toutes les variables d'environnement nécessaires dans votre procédure automatique et/ou d'utiliser des chemins absolus vers les programmes.

Les procédures automatiques de Subversion sont lancées par l'utilisateur propriétaire du processus ayant accès au dépôt Subversion. La plupart du temps, on accède au dépôt via un serveur Subversion, donc cet utilisateur est le même que celui qui fait tourner le processus serveur sur le système. Les procédures automatiques elles-mêmes doivent être configurées pour être exécutables, au niveau du système d'exploitation, par ledit utilisateur. Cela implique également que tout programme ou fichier (y compris le dépôt Subversion) utilisé directement ou indirectement par la procédure automatique l'est par ledit utilisateur. En d'autres termes, faites bien attention aux problèmes de droits d'exécution qui peuvent empêcher les scripts d'effectuer correctement les tâches pour lesquelles ils ont été conçus.

Il y a plusieurs procédures automatiques implémentées dans le dépôt Subversion et vous pouvez obtenir des détails sur chacune d'elles dans la section intitulée « Procédures automatiques du dépôt ». En tant qu'administrateur du dépôt, vous devez décider quelles procédures automatiques vous voulez mettre en œuvre (c'est-à-dire les nommer correctement et leur donner les droits appropriés) et de quelle manière. Lorsque vous prennez cette décision, gardez à l'esprit l'architecture de votre dépôt. Par exemple, si vous vous servez de la configuration du serveur pour déterminer les droits de propagation sur votre dépôt, vous n'avez pas besoin de mettre en place un contrôle d'accès de ce style via les procédures automatiques.

Les exemples de procédures automatiques librement accessibles sont légions, fournis par la communauté Subversion elle-même ou par d'autres. Ces scripts couvrent une large variété de besoins tels que le contrôle d'accès basique, le contrôle de cohérence, l'intégration avec les outils de suivis de bogues, les notifications de propagation par e-mail ou flux RSS, etc. Sinon, si vous voulez écrire votre propre programme, penchez-vous sur le Chapitre 8, *Intégration de Subversion*.

Bien que les procédures automatiques soient capables de faire tout et n'importe quoi, leurs auteurs devraient faire preuve de modération dans un domaine précis : *ne modifiez pas* une transaction de propagation en utilisant une procédure automatique. Bien que cela soit tentant de corriger automatiquement certaines erreurs, raccourcis ou violations de politique constatées dans les fichiers propagés, cela peut causer des problèmes. Subversion conserve en cache, côté client, certaines parties des données du dépôt et si vous modifiez une transaction de propagation de cette façon, ces caches seront périmés sans que cela ne puisse être détecté. De telles incohérences peuvent aboutir à des comportements surprenants et inattendus. Au lieu de modifier la transaction, contentez-vous de vérifier la

transaction dans la procédure automatique `pre-commit` et rejetez-la si elle ne remplit pas les conditions nécessaires. Entre autre avantages, vos utilisateurs prendront ainsi des habitudes de travail empreintes de respect des procédures et de qualité.

Configuration de la base de données Berkeley DB

Un environnement Berkeley DB peut encapsuler une ou plusieurs bases de données, fichiers de journalisation, de région et de configuration. L'environnement Berkeley DB a un ensemble propre de valeurs configurées par défaut comme par exemple le nombre de verrous autorisés à un instant donné, la taille maximum des fichiers de journalisation, etc. La logique du système de fichiers Subversion ajoute des valeurs par défaut pour différentes options de configuration du gestionnaire Berkeley DB. Cependant, il se peut que votre dépôt nécessite une configuration différente en raison de l'architecture de vos données et des méthodes d'accès.

Les concepteurs du gestionnaire de bases de données Berkeley DB comprennent que les besoins varient entre les différentes applications et environnements de bases de données, c'est pourquoi ils fournissent des mécanismes pour modifier, à l'exécution, une grande partie des valeurs des options de configuration. BDB vérifie la présence d'un fichier nommé `DB_CONFIG` dans le répertoire d'environnement (à savoir le sous-répertoire `db` du dépôt) et en extrait les valeurs des options. Subversion crée ce fichier lorsqu'il crée le reste du dépôt. Le fichier contient initialement des options par défaut ainsi que des pointeurs vers la documentation en ligne de Berkeley DB afin de vous renseigner sur l'utilisation de ces options. Bien sûr, vous êtes libre d'ajouter n'importe quelle option prise en compte par Berkeley DB dans votre fichier `DB_CONFIG`. Soyez juste attentif au fait que, bien que Subversion n'essaie jamais de lire ou interpréter le contenu de ce fichier et qu'il n'en utilise pas directement la configuration, les changements induits dans le comportement de Berkeley DB ne doivent pas aller à l'encontre du comportement attendu par Subversion. Par ailleurs, les changements effectués dans `DB_CONFIG` ne sont pris en considération qu'après avoir effectué une restauration de l'environnement de la base de données avec la commande **svnadmin recover**.

Maintenance d'un dépôt

Assurer la maintenance d'un dépôt Subversion peut être intimidant, certainement parce que les systèmes qui comprennent une base de données sont complexes. Il faut pour cela connaître les outils — ceux dont on dispose, quand les utiliser et comment. Cette section vous présente les outils fournis par Subversion pour assurer l'administration du dépôt et décrit leur maniement pour réaliser des opérations telles que migrations de données, mises à jour, sauvegardes et nettoyages.

Boîte à outils de l'administrateur

Subversion fournit une poignée d'utilitaires pour créer, inspecter, modifier et réparer votre dépôt. Étudions de plus près chacun de ces outils. Ensuite, nous abordons rapidement quelques utilitaires inclus dans le gestionnaire de bases de données Berkeley DB qui fournissent des fonctionnalités spécifiques au magasin de données de votre dépôt qui ne sont pas assurées par les propres outils de Subversion.

svnadmin

Le programme **svnadmin** est le meilleur ami de l'administrateur de dépôts. En plus de fournir la possibilité de créer des dépôts Subversion, ce programme vous permet d'effectuer de nombreuses opérations de maintenance sur ces dépôts. La syntaxe de **svnadmin** est similaire à celle des autres programmes en ligne de commande de Subversion :

```
$ svnadmin help
usage général : svnadmin SOUS_COMMANDE DÉPÔT [ARGS & OPTIONS ...]
Entrer 'svnadmin help <sous-commande>' pour une aide spécifique.
Entrer 'svnadmin --version' pour avoir la version et les modules de stockages.

Sous-commandes disponibles :
   crashtest
   create
   deltify
…
```

Au début de ce chapitre (dans la section intitulée « Création d'un dépôt »), nous vous avons présenté la sous-commande

svnadmin create. La plupart des autres sous-commandes **svnadmin** sont couvertes plus loin dans ce chapitre. Vous pouvez également consulter la section intitulée « svnadmin » pour une liste complète des sous-commandes et des fonctionnalités qu'elles apportent.

svnlook

svnlook est un outil de Subversion pour examiner les différentes révisions et *transactions* (qui sont des révisions en cours de création) dans un dépôt. Aucune modification n'est faite au dépôt par cet outil. **svnlook** est généralement utilisé par les procédures automatiques du dépôt pour signaler les changements qui vont être propagés (dans le cas de la procédure automatique **pre-commit**) ou qui viennent d'être propagés (dans le cas de la procédure automatique **post-commit**). Un administrateur peut être amené à utiliser cet outil à des fins de diagnostic.

La syntaxe de **svnlook** est particulièrement simple :

```
$svnlook help
usage général : svnlook SOUS_COMMANDE CHEMIN_DÉPÔT [ARGS & OPTIONS...]
Note : Quand --revision ou --transaction ne sont pas précisées, les sous-
       commandes qui en ont besoin utilisent la révision la plus récente.
Entrer 'svnlook help <sous-commande>' pour une aide spécifique.
Entrer 'svnlook --version' pour avoir la version et les modules de stockage.
…
```

La plupart des sous-commandes **svnlook** peuvent être appliquées soit à une révision soit à une arborescence de transaction, affichant les informations à propos de l'arborescence elle-même ou les différences par rapport à la révision précédente du dépôt. Pour spécifier quelle révision ou quelle transaction examiner, utilisez respectivement les options `--revision` (`-r`) et `--transaction` (`-t`). En l'absence des options `--revision` (`-r`) ou `--transaction` (`-t`), **svnlook** examine la révision la plus récente (la révision `HEAD`) du dépôt. Ainsi, les deux commandes suivantes font exactement la même chose si la révision la plus récente du dépôt situé à l'emplacement `/var/svn/depot` porte le numéro 19 :

```
$ svnlook info /var/svn/depot
$ svnlook info /var/svn/depot -r 19
```

Signalons une exception à ces règles concernant les sous-commandes : la sous-commande **svnlook youngest** ne prend aucune option et affiche simplement le numéro de la révision la plus récente du dépôt :

```
$ svnlook youngest /var/svn/depot
19
$
```

Gardez à l'esprit que les seules transactions que vous pouvez examiner sont celles qui n'ont pas été propagées. La plupart des dépôts ne comportent pas de transactions de ce type parce que les transactions sont habituellement soit propagées (auquel cas vous devriez y avoir accès sous la forme de révisions via l'option `--revision` (`-r`)), soit annulées et supprimées.

La sortie de **svnlook** est conçue pour être à la fois lisible par un humain et analysable par une machine. Prenons, par exemple, la sortie de la sous-commande **svnlook info** :

```
$ svnlook info /var/svn/depot
sally
2002-11-04 09:29:13 -0600 (lun. 04 nov. 2002)
27
J'ai ajouté le traditionnel
Arbre grec.
$
```

La sortie de **svnlook info** est constituée des éléments suivants, par ordre d'apparition :

1. L'auteur, suivi d'un passage à la ligne.

2. La date, suivie d'un passage à la ligne.

3. Le nombre de caractères du message de propagation, suivi d'un passage à la ligne.

4. Le message de propagation lui-même, suivi d'un passage à la ligne.

Cette sortie est lisible par un humain, ce qui veut dire que les éléments tels que la date sont représentés par du texte simple au lieu d'un obscur code (comme le nombre de nanosecondes depuis le passage aux nouveaux francs). Mais cette sortie est aussi analysable par une machine — parce que le message de propagation peut comporter plusieurs lignes et n'est pas limité en taille, **svnlook** affiche la longueur du message avant le message lui-même. Cela permet aux scripts et autres utilitaires faisant appel à cette commande de prendre des décisions opportunes à propos du message de propagation, comme savoir combien de mémoire allouer pour le message ou au moins savoir combien d'octets sauter dans le cas où les données affichées par **svnlook** ne sont pas les dernières données du flux.

svnlook peut répondre à un tas d'autres requêtes : afficher des sous-ensembles des informations précédemment citées, lister récursivement les arborescences suivies en versions des répertoires, lister les chemins modifiés lors de telle révision ou transaction, afficher les différences de contenu et de propriétés pour les fichiers et répertoires, etc. Reportez-vous à la section intitulée « svnlook » pour la liste complète des fonctionnalités offertes par **svnlook**.

svndumpfilter

Bien que ce ne soit pas l'outil le plus utilisé mis à disposition de l'administrateur, **svndumpfilter** fournit une fonctionnalité d'un genre très particulier qui est d'une grande utilité : la possibilité de modifier rapidement et facilement des flux de l'historique du dépôt Subversion en agissant en tant que filtre sur les chemins.

La syntaxe de **svndumpfilter** est la suivante :

```
$ svndumpfilter help
usage général : svndumpfilter SOUS_COMMANDE [ARGS & OPTIONS ...]
Entrer 'svndumpfilter help <sous-commande>' pour l'aide spécifique.
Entrer 'svndumpfilter --version' pour avoir le numéro de version du programme.

Sous-commandes disponibles :
   exclude
   include
   help (?, h)
```

Il n'y a que deux sous-commandes intéressantes : **svndumpfilter exclude** et **svndumpfilter include**. Elles vous permettent de choisir entre l'inclusion implicite et l'inclusion explicite des chemins dans le flux. Vous en saurez plus sur ces sous-commandes et sur l'utilité si particulière de **svndumpfilter** plus loin dans ce chapitre, dans la section intitulée « Filtrage de l'historique d'un dépôt ».

svnsync

Le programme **svnsync**, apparu dans la version 1.4 de Subversion, fournit toutes les fonctionnalités requises pour faire fonctionner un miroir en lecture seule d'un dépôt Subversion. Ce programme a une et une seule fonction : transférer l'historique d'un dépôt vers un autre dépôt. Et, bien qu'il y ait différentes manières de faire, sa force réside dans sa capacité de travailler à distance : les dépôts « source » et « destination » peuvent être sur deux ordinateurs différents et **svnsync** sur un troisième.

Comme vous vous en doutez, **svnsync** possède une syntaxe très proche des autres programmes déjà mentionnés dans ce chapitre :

```
$svnsync help
usage général : svnsync SOUS_COMMANDE DÉPÔT [ARGS & OPTIONS ...]
Entrer 'svnsync help <sous-commande>' pour une aide spécifique.
Entrer 'svnsync --version' pour la version et les modules d'accès (RA).

Sous-commandes disponibles :
```

```
   initialize (init)
   synchronize (sync)
   copy-revprops
   help (?, h)
$
```

Nous revenons en détail sur la réplication de dépôts avec **svnsync** plus loin dans ce chapitre (voir la section intitulée « Réplication d'un dépôt »).

fsfs-reshard.py

Bien qu'il ne fasse pas officiellement partie des outils Subversion, le script **fsfs-reshard.py** (situé dans le répertoire `tools/server-side` du code source de Subversion) est un outil particulièrement utile à l'administrateur pour optimiser les performances de dépôts Subversion utilisant un magasin de données FSFS. Les dépôts FSFS contiennent des fichiers qui décrivent les changements apportés dans une seule révision et des fichiers qui contiennent les propriétés de révision associées à une seule révision. Les dépôts créés avec Subversion avant la version 1.5 conservent ces fichiers dans deux répertoires : un pour chaque type de fichiers. Au fur et à mesure des révisions propagées dans le dépôt, Subversion dépose de plus en plus de fichiers dans ces deux répertoires — au bout d'un certain temps, le nombre de fichiers dans chaque répertoire peut devenir particulièrement élevé. On a pu constater dans cette situation des problèmes de performances sur certains systèmes de fichiers en réseau.

Subversion 1.5 crée les dépôts FSFS en utilisant un schéma légèrement différent pour lequel ces deux répertoires sont répartis (*sharded* en anglais, d'où le nom du script) dans plusieurs sous-répertoires. Cela peut réduire de façon drastique le temps de recherche d'un de ces fichiers et, en conséquence, améliorer la performance globale de Subversion pour la lecture du dépôt. Le nombre de sous-répertoires utilisés pour héberger ces fichiers est d'ailleurs configurable et c'est là qu'intervient le script **fsfs-reshard.py**. Le script remanie la structure du dépôt pour se conformer au nombre de sous-répertoires demandés. C'est particulièrement utile pour convertir un vieux dépôt Subversion vers le nouveau schéma réparti de Subversion 1.5 (ce que Subversion ne fait pas automatiquement pour vous) ou pour modifier cette valeur dans un dépôt déjà réparti.

Utilitaires Berkeley DB

Si vous utilisez un dépôt avec une base Berkeley DB, à la fois les données et la structure de votre système de fichiers suivis en version résident dans un ensemble de tables de la base de données qui sont situées dans le sous-répertoire `db/` de votre dépôt. Ce sous-répertoire est un répertoire d'environnement classique de base de données Berkeley DB et n'importe quel outil de base de données Berkeley, généralement fourni avec la distribution Berkeley, peut y être utilisé.

Pour un usage quotidien, ces outils ne sont pas nécessaires. La plupart des fonctionnalités dont les dépôts Subversion ont besoin ont été dupliquées dans l'outil **svnadmin**. Par exemple, **svnadmin list-unused-dblogs** et **svnadmin list-dblogs** fournissent un sous-ensemble des fonctionnalités offertes par l'utilitaire **db_archive** de Berkeley DB et **svnadmin recover** reproduit les utilisations courantes de l'utilitaire **db_recover**.

Cependant, il reste quelques utilitaires Berkeley DB que vous pourriez trouver utiles. Les programmes **db_dump** et **db_load** fonctionnent avec, pour la lecture et l'écriture respectivement, un format de fichier personnalisé qui décrit les clés et les valeurs d'une base de données Berkeley DB. Puisque les bases de données Berkeley DB ne sont pas portables d'une architecture de machine à une autre, ce format est utile pour transférer les bases de données entre deux machines, indépendamment de l'architecture et du système d'exploitation. Comme nous le décrivons plus loin dans ce chapitre, vous pouvez aussi utiliser **svnadmin dump** et **svnadmin load** pour faire la même chose, mais **db_dump** et **db_load** peuvent accomplir certaines tâches tout aussi bien et beaucoup plus vite. Ils peuvent aussi être utiles si un expert Berkeley DB, pour une raison ou pour une autre, doit manipuler les données directement dans la base de données d'un dépôt BDB (les utilitaires Subversion ne le vous permettent pas). De plus, l'utilitaire **db_stat** peut fournir des informations utiles sur l'état de votre environnement Berkeley DB, y compris des statistiques détaillées concernant les sous-systèmes de verrouillage et de stockage.

Pour davantage d'informations sur la suite d'outils Berkeley DB, consultez la documentation en ligne sur le site Internet d'Oracle, dans la section Berkeley DB : http://www.oracle.com/technology/documentation/berkeley-db/db/ (ce site est en anglais).

Correction des messages de propagation

Parfois un utilisateur se trompe dans son message de propagation (une faute d'orthographe ou une coquille, par exemple). Si le dépôt est configuré (en utilisant la procédure automatique `pre-revprop-change`, voir la section intitulée « Mise en place des procédures automatiques ») pour accepter les modifications de ce message après la fin de la propagation, l'utilisateur peut

corriger son message à distance en utilisant **svn propset** (voir svn propset). Cependant, en raison de la possibilité de perte d'information irrémédiable, les dépôts Subversion ne sont pas configurés, par défaut, pour autoriser les modifications de propriétés non suivies en versions — sauf de la part d'un administrateur.

Si un administrateur est amené à changer un message de propagation, il peut le faire avec **svnadmin setlog**. Cette commande change le message de propagation (la propriété `svn:log`) d'une révision donnée du dépôt, la nouvelle valeur étant lue dans un fichier.

```
$ echo "Voici le nouveau message de propagation, en version corrigée" >
nouveau-message.txt
$ svnadmin setlog mon-depot nouveau-message.txt -r 388
```

La commande **svnadmin setlog**, par défaut, possède les mêmes garde-fous pour empêcher de modifier des propriétés non suivies en versions qu'un client distant — les procédures automatiques `pre-` et `post-revprop-change` sont toujours activées et doivent donc être configurées afin d'accepter ce type de changement. Mais un administrateur peut contourner ces protections en passant l'option `--bypass-hooks` à la commande **svnadmin setlog**.

Souvenez-vous cependant que, en contournant les procédures automatiques, vous êtes susceptible de ne pas activer certaines actions telles que la notification par email du changement des propriétés, la sauvegarde par les systèmes qui surveillent les propriétés non suivies en versions, etc. En d'autres termes, faites particulièrement attention aux changements que vous apportez et à la manière dont vous le faites.

Gestion de l'espace disque

Bien que le coût de stockage ait diminué de manière drastique ces dernières années, l'utilisation de l'espace disque reste une des préoccupations de l'administrateur qui doit suivre en versions de grandes quantités de données. Chaque élément de l'historique de chaque donnée stockée dans un dépôt actif doit être sauvegardé ailleurs, peut-être même de nombreuses fois dans le cas de sauvegardes tournantes. Il est utile de savoir quelles données d'un dépôt Subversion doivent rester sur le site de production, lesquelles doivent être sauvegardées et lesquelles peuvent être supprimées sans risque.

Économie d'espace disque

Pour garder un dépôt petit, Subversion utilise la *différenciation* (ou « stockage différentiel ») à l'intérieur du dépôt lui-même. La différenciation implique l'encodage de la représentation d'un groupe de données sous la forme d'un ensemble de différences avec un autre groupe de données. Si les deux groupes de données sont très similaires, la différenciation économise de l'espace pour le groupe différencié — au lieu de prendre le même espace que les données originales, le groupe occupe juste l'espace nécessaire pour dire : « je ressemble à l'autre groupe de données là-bas, sauf pour les deux ou trois changements qui suivent ». Au final, l'espace occupé par l'ensemble des données du dépôt — c'est-à-dire le contenu des fichiers suivis en versions — est beaucoup plus petit que la représentation textuelle originale de ces données. Et pour les dépôts créés avec Subversion en version 1.4 ou plus, l'espace économisé est encore plus important — les représentations textuelles des fichiers sont à présent elles-mêmes compressées.

Comme toutes les données sujettes à différenciation dans un dépôt BDB sont stockées dans un unique fichier de la base de données, réduire la taille des données stockées ne réduit pas instantanément la taille du fichier de base de données lui-même. Le gestionnaire de base de données Berkeley DB garde néanmoins une trace des zones non-utilisées du fichier de base de données et utilise ces zones avant d'augmenter la taille du fichier de base de données. Ainsi, même si la différenciation n'économise pas immédiatement de la place, cela ralentit de façon drastique la croissance de la base de données.

Suppression des transactions mortes

Bien que rares, il y a des circonstances dans lesquelles le déroulement d'une propagation Subversion peut mal se terminer, laissant derrière elle dans le dépôt des restes de cette tentative de propagation : une transaction inachevée et toutes les modifications de fichiers et de répertoires associées. Il peut y avoir plusieurs raisons à cet échec : l'utilisateur a peut-être brutalement interrompu l'opération côté client ou bien une coupure réseau s'est peut-être produite au milieu de l'opération. Quoi qu'il en soit, des transactions mortes peuvent apparaître. Elles ne sont pas dangereuses mais elles consomment inutilement de l'espace disque. Un administrateur consciencieux se doit néanmoins de les supprimer.

Vous pouvez utiliser la commande **svnadmin lstxns** pour obtenir la liste des noms des transactions non encore réglées :

```
$ svnadmin lstxns mon-depot
19
3a1
a45
$
```

Chaque élément de la sortie de cette commande peut être passé en argument de **svnlook** (avec l'option `--transaction` (`-t`)) pour déterminer qui est à l'origine de la transaction, quand elle a eu lieu et quels types de changements ont été effectués — ces informations sont très utiles pour savoir si on peut supprimer la transaction sans arrière pensée ! Si vous décidez effectivement de supprimer la transaction, son nom peut être passé à **svnadmin rmtxns** qui fera le nettoyage adéquat. En fait, **svnadmin rmtxns** peut directement prendre en entrée la sortie de **svnadmin lstxns** !

```
$ svnadmin rmtxns mon-depot `svnadmin lstxns mon-depot`
$
```

Si vous utilisez ces deux sous-commandes ainsi, vous devriez envisager de rendre votre dépôt temporairement indisponible pour les clients. De cette manière, personne ne peut initier une transaction légitime avant que le nettoyage n'ait commencé. L'exemple Exemple 5.1, « txn-info.sh (lister les transactions inachevées) » contient quelques lignes de script shell qui peuvent produire les informations relatives à chaque transaction inachevée de votre dépôt.

Exemple 5.1. txn-info.sh (lister les transactions inachevées)

```
#!/bin/sh

### Produit les informations relatives à toutes les transactions
### inachevées d'un dépôt Subversion

DEPOT="${1}"
if [ "x$DEPOT" = x ] ; then
  echo "utilisation: $0 CHEMIN_VERS_LE_DEPOT"
  exit
fi

for TXN in `svnadmin lstxns ${DEPOT}`; do
  echo "---[ Transaction ${TXN} ]-------------------------------------------"
  svnlook info "${DEPOT}" -t "${TXN}"
done
```

La sortie produite par ce script est, en bref, la concaténation des différents groupes d'informations fournis par **svnlook info** (voir la section intitulée « svnlook ») et ressemble à ceci :

```
$ txn-info.sh mon-depot
---[ Transaction 19 ]-------------------------------------------
sally
2001-09-04 11:57:19 -0500 (mar. 04 sep. 2001)
0
---[ Transaction 3a1 ]-------------------------------------------
harry
2001-09-10 16:50:30 -0500 (lun. 10 sep. 2001)
39
Tentative de propagation dans un réseau pourri
---[ Transaction a45 ]-------------------------------------------
sally
2001-09-12 11:09:28 -0500 (mer. 12 sep. 2001)
0
```

```
$
```

Une transaction initiée depuis longtemps correspond en général à une propagation qui a été interrompue ou qui a échoué. L'horodatage de la transaction peut fournir des informations intéressantes — par exemple, quelle est la probabilité qu'une transaction commencée il y a neuf mois soit toujours active ?

En résumé, la décision de supprimer une transaction ne doit pas être prise à la légère. D'autres sources d'informations — comme les journaux d'Apache sur les erreurs et les accès, les journaux opérationnels de Subversion, l'historique des révisions Subversion, etc. — peuvent aider à la prise de décision. Et bien sûr, l'administrateur peut toujours entrer en contact (par email, par exemple) avec l'auteur d'une transaction qui semble abandonnée pour vérifier que c'est bien le cas.

Purge des fichiers de journalisation inutilisés de Berkeley DB

Jusqu'à il y a peu, les plus gros consommateurs d'espace disque pour les dépôts Subversion basés sur BDB étaient les fichiers de journalisation dans lesquels le gestionnaire Berkeley DB effectue les pré-écritures avant de modifier la base de données elle-même. Ces fichiers recensent toutes les actions menées pour modifier la base de données, étape par étape ; alors que les fichiers de la base de données, à un instant donné, ne reflètent qu'un état particulier, les fichiers de journalisation contiennent l'ensemble de tous les changements opérés *entre* chaque état successif. Ainsi, ils peuvent grossir assez rapidement.

Heureusement, à partir de la version 4.2 de Berkeley DB, l'environnement de la base de données est capable de supprimer ses propres fichiers non utilisés automatiquement. Tout dépôt créé en utilisant **svnadmin** compilé avec la version 4.2 de Berkeley DB (ou suivantes) est configuré pour supprimer automatiquement les fichiers de journalisation. Si vous ne voulez pas activer cette fonctionnalité, passez simplement l'option `--bdb-log-keep` à la commande **svnadmin create**. Si vous oubliez de le faire ou si vous changez d'avis plus tard, éditez simplement le fichier `DB_CONFIG` qui se trouve dans le répertoire `db` de votre dépôt, commentez la ligne qui contient la directive `set_flags DB_LOG_AUTOREMOVE` puis lancez **svnadmin recover** sur votre dépôt pour que le changement de configuration prenne effet. Reportez-vous à la section intitulée « Configuration de la base de données Berkeley DB » pour plus d'informations sur la configuration du gestionnaire de bases de données.

Sans suppression automatique des fichiers de journalisation, les journaux vont s'accumuler au fur et à mesure de l'utilisation de votre dépôt. Cela peut être considéré comme une fonctionnalité du gestionnaire de bases de données — vous devez être capable de recréer entièrement votre base de données en utilisant uniquement vos fichiers de journalisation, c'est pourquoi ceux-ci sont utiles pour le rétablissement de la base après une catastrophe. Mais en général, vous voudrez archiver les fichiers de journalisation qui ne sont plus utilisés par la base de données et ensuite les enlever du disque pour conserver de l'espace libre. Utilisez la commande **svnadmin list-unused-dblogs** pour avoir la liste des fichiers de journalisation inutilisés :

```
$ svnadmin list-unused-dblogs /var/svn/depot
/var/svn/depot/log.0000000031
/var/svn/depot/log.0000000032
/var/svn/depot/log.0000000033
…
$ rm `svnadmin list-unused-dblogs /var/svn/depot`
## espace disque récupéré !
```

Les dépôts BDB qui utilisent les fichiers de journalisation pour les sauvegardes ou les rétablissements après incident *ne doivent pas* activer la suppression automatique des fichiers de journalisation. La reconstruction des données d'un dépôt à partir des fichiers de journalisation ne peut être effectuée que si *tous* les fichiers de journalisation sont accessibles. Si quelques fichiers de journalisation sont supprimés du disque avant que le système de sauvegarde n'ait pu les copier ailleurs, l'ensemble incomplet des fichiers de journalisation est totalement inutile.

Rétablissement de bases de données Berkeley DB

Comme indiqué dans la section intitulée « Berkeley DB », un dépôt Berkeley DB peut se retrouver bloqué s'il n'est pas arrêté proprement. Quand cela arrive, un administrateur doit faire revenir la base de données en arrière jusque dans un état cohérent. Ceci ne concerne cependant que les dépôts BDB — si vous utilisez FSFS, vous n'êtes pas concerné. Et pour ceux qui utilisent Subversion 1.4 avec Berkeley DB version 4.4 ou plus, vous constaterez que Subversion est devenu beaucoup plus résilient face à ce type de problème. Certes, mais des plantages de dépôts Berkeley DB arrivent encore et un administrateur doit savoir comment réagir dans de telles circonstances.

Pour protéger les données du dépôt, le gestionnaire Berkeley DB utilise un mécanisme de verrouillage. Ce mécanisme s'assure que les éléments de la base de données ne sont pas modifiés en même temps par plusieurs utilisateurs et que chaque processus voit les données dans un état cohérent lors de la lecture de la base de données. Quand un processus a besoin de modifier quelque chose dans la base de données, il vérifie d'abord l'existence d'un verrou sur les données concernées. Si les données ne sont pas verrouillées, le processus les verrouille, effectue les changements qu'il veut puis déverrouille les données. Les autres processus sont obligés d'attendre que le verrou soit libéré avant d'être autorisés à accéder aux données de cette zone (ceci n'a rien à voir avec les verrous que vous, utilisateur, pouvez appliquer sur les fichiers suivis en versions dans le dépôt ; nous essayons de lever l'ambiguïté créée par l'emploi de cette terminologie commune dans l'encadré Les trois types de « verrous ».)

Au cours de l'utilisation de votre dépôt Subversion, des erreurs fatales ou des interruptions peuvent empêcher un processus de supprimer des verrous qu'il a placés dans la base de données. Cela conduit à des plantages du magasin de données. Lorsque cela arrive, toutes les tentatives d'accès au dépôt se soldent par un échec (puisque chaque nouvel arrivant attend que le verrou se libère, ce qui n'est pas prêt d'arriver).

Si cela arrive à votre dépôt, ne paniquez pas. Le système de fichiers Berkeley DB tire parti des transactions de la base de données, des points de contrôle et de la journalisation préalable à toute écriture pour garantir que seuls les événements les plus catastrophiques[8] soient à même de détruire définitivement un environnement de base de données. Un administrateur suffisamment paranoïaque conserve des sauvegardes des données du dépôt dans un endroit distinct, mais attendez un peu avant de vous diriger vers l'armoire de rangement des sauvegardes.

Appliquez plutôt la recette suivante pour tenter de « faire repartir » votre dépôt :

1. Assurez-vous qu'aucun processus n'accède au dépôt (ou ne tente de le faire). Pour les dépôts en réseau, cela implique d'arrêter le serveur HTTP Apache ou le démon svnserve.

2. Prenez l'identité de l'utilisateur qui possède et gère le dépôt. C'est important, puisque rétablir un dépôt avec un autre utilisateur peut modifier les droits d'accès des fichiers du dépôt de telle manière que votre dépôt soit toujours inaccessible même après la remise en service.

3. Lancez la commande **`svnadmin recover /var/svn/depot`**. Vous devriez obtenir une sortie du genre :

   ```
   Verrou du dépôt acquis.
   Patientez ; le rétablissement du dépôt peut être long...

   Fin du rétablissement.
   La dernière révision du dépôt est 19
   ```

   Cette commande peut durer plusieurs minutes.

4. Redémarrez le processus serveur.

Cette procédure fonctionne dans presque tous les cas de plantage. Faites attention à ce qu'elle soit lancée par l'utilisateur qui possède et gère la base de données, pas par `root`. La procédure de rétablissement peut impliquer de récréer en partant de zéro certains fichiers de la base de données (de la mémoire partagée, par exemple). Un rétablissement par `root` créerait ces fichiers avec `root` comme propriétaire, ce qui veut dire que même après que vous ayez rétabli l'accès à votre dépôt, les utilisateurs de base n'y auront pas accès.

Si la procédure précédente, pour une raison ou pour une autre, ne fait pas repartir votre dépôt, vous devez faire deux choses. D'abord, mettez de côté votre répertoire de dépôt cassé (par exemple en le renommant `depot.CASSE`) puis restaurez la dernière sauvegarde de votre dépôt. Ensuite, envoyez un email à la liste de diffusion des utilisateurs de Subversion (`<users@subversion.tigris.org>`) et décrivez votre problème en détail. L'intégrité des données fait partie des sujets à très haute priorité pour les développeurs Subversion.

Migration des données d'un dépôt

Un système de fichiers Subversion a ses données réparties dans les fichiers du dépôt d'une manière que seuls les développeurs Subversion eux-mêmes comprennent (et s'y intéressent). Il peut cependant y avoir des circonstances qui obligent à copier ou

[8] Par exemple, disque dur + gros aimant à côté = désastre.

déplacer l'ensemble (ou une partie) des données d'un dépôt à un autre.

Subversion fournit cette fonctionnalité par le biais des *flux de déchargement du dépôt*. Un flux de déchargement de dépôt (« fichier dump » ou *dump file* en anglais, quand il est stocké dans un fichier sur le disque) est un format de fichier portable, contenant des données brutes, qui décrit les différentes révisions de votre dépôt — ce qui a été modifié, par qui, quand, etc. Ce fichier dump est le principal mécanisme utilisé pour réorganiser des historiques de versions — en partie ou en totalité, avec ou sans modification — entre des dépôts. Et Subversion fournit les outils nécessaires à la création et au chargement de ces fichiers dump : les sous-commandes **svnadmin dump** et **svnadmin load** respectivement.

Bien que le format des fichiers dump de Subversion contienne des parties lisibles par les humains et une structure familière (elle ressemble au format décrit par la RFC 822, utilisé pour la plupart des emails), *ce n'est pas* un format de fichier purement textuel. C'est un format de fichier binaire, très sensible aux modifications faites à son contenu. Par exemple, de nombreux éditeurs de textes corrompent le fichier en convertissant les caractères de fin de ligne.

Il existe de nombreuses raisons de décharger et recharger les données d'un dépôt Subversion. Aux premiers temps de Subversion, la principale raison était l'évolution de Subversion lui-même. Au fur et à mesure que Subversion gagnait en maturité, des changements faits sur les schémas des magasins de données sous-jacents entraînaient des problèmes de compatibilité avec les versions précédentes du dépôt, ce qui obligeait les utilisateurs à décharger les données de leurs dépôts en utilisant la version précédente de Subversion puis à recharger ces données dans un dépôt tout neuf créé avec la nouvelle version de Subversion. Il n'y a pas eu de changement de schéma de ce type depuis la version 1.0 de Subversion et les développeurs ont promis de ne pas forcer les utilisateurs à décharger et recharger leurs dépôts lors du passage d'une version mineure à une autre (comme par exemple entre la version 1.3 et la version 1.4) de Subversion. Mais il existe néanmoins des raisons de décharger et recharger ses données, comme le redéploiement d'un dépôt Berkeley DB sur un nouveau système d'exploitation ou sur une architecture CPU différente, la migration du magasin de données de Berkeley DB à FSFS et réciproquement ou (comme nous le voyons dans ce chapitre à la section intitulée « Filtrage de l'historique d'un dépôt ») la purge de données suivies en version de l'historique du dépôt.

Le format de déchargement des dépôts Subversion ne décrit que l'évolution des éléments suivis en version. Il ne contient pas d'information sur les transactions inachevées, les verrous utilisateurs sur les chemins du système de fichiers, la configuration personnalisée du dépôt ou du serveur (y compris les procédures automatiques) et ainsi de suite.

Quelle que soit la raison pour laquelle vous voulez migrer votre historique de dépôt, l'utilisation des sous-commandes **svnadmin dump** et **svnadmin load** est simplissime. **svnadmin dump** affiche un intervalle de révisions du dépôt, chacune utilisant le format des fichiers dump Subversion. Le fichier dump est envoyé sur la sortie standard tandis que les messages d'information sont envoyés sur la sortie d'erreur. Ceci vous permet de rediriger le flux standard vers un fichier tout en visualisant ce qui se passe dans votre terminal. Par exemple :

```
$svnlook youngest mon-depot
26
$ svnadmin dump mon-depot > fichier-dump
* Révision 0 déchargée.
* Révision 1 déchargée.
* Révision 2 déchargée.
…
* Révision 25 déchargée.
* Révision 26 déchargée.
```

A la fin de la procédure, vous obtiendrez un fichier unique (`fichier-dump` dans l'exemple précédent) qui contient toutes les données stockées dans votre dépôt pour l'intervalle de révisions demandé. Notez que **svnadmin dump** lit les arborescences des révisions du dépôt de la même manière que tout autre processus « lecteur » (par exemple **svn checkout**), vous pouvez donc sans risque lancer cette commande à n'importe quel moment.

La commande jumelle, **svnadmin load**, recherche dans l'entrée standard la structure d'un fichier dump Subversion puis insère les révisions déchargées dans le dépôt de destination spécifié. Elle fournit elle aussi des informations sur le déroulement de l'opération, cette fois en utilisant la sortie standard :

```
$ svnadmin load nouveau-depot < fichier-dump
<<< Début d'une nouvelle transaction basée sur la révision 1
       * ajout de : A ... fait.
       * ajout de : A/B ... fait.
       …
  ------- Révision 1 propagée (commit) >>>

  <<< Début d'une nouvelle transaction basée sur la révision 2
       * édition de : A/mu ... fait.
       * édition de : A/D/G/rho ... fait.

  ------- Révision 2 propagée (commit) >>>

  …

  <<< Début d'une nouvelle transaction basée sur la révision 25
       * édition de : A/D/gamma ... fait.

  ------- Révision 25 propagée (commit) >>>

  <<< Début d'une nouvelle transaction basée sur la révision 26
       * ajout de : A/Z/zeta ... fait.
       * édition de : A/mu ... fait.

  ------- Révision 26 propagée (commit) >>>
```

Le résultat d'un chargement est l'ajout de nouvelle révisions à un dépôt — comme si vous faisiez des propagations vers ce dépôt avec un client Subversion classique. De la même manière que pour une propagation, vous pouvez utiliser les procédures automatiques pour effectuer des actions particulières avant et après chaque propagation faite par la procédure de chargement. En passant les options `--use-pre-commit-hook` et `--use-post-commit-hook` (respectivement) à **svnadmin load**, vous demandez à Subversion d'exécuter les procédures automatiques pré-propagation et post-propagation (respectivement) pour chaque révision chargée. Un exemple d'utilisation de ces options est de s'assurer que les révisions chargées passent par les mêmes étapes de validation qu'une propagation normale. Bien sûr, utilisez ces options avec prudence — si votre script post-propagation envoie des emails à une liste de diffusion pour chaque nouvelle propagation, vous ne voulez peut-être pas envoyer des centaines voire des milliers d'emails de notification à la suite vers cette liste ! Vous pouvez en apprendre davantage sur l'utilisation des procédures automatiques dans la section intitulée « Mise en place des procédures automatiques ».

Notez que puisque **svnadmin** utilise l'entrée et la sortie standards pour le déchargement et le rechargement, les administrateurs les plus intrépides peuvent tenter des choses du genre (peut-être même en utilisant différentes versions de **svnadmin** de chaque côté de la barre verticale `|`) :

```
$ svnadmin create nouveau-depot
$ svnadmin dump vieux-depot | svnadmin load nouveau-depot
```

Par défaut, le fichier dump prend beaucoup de place — beaucoup plus que le dépôt lui-même. C'est parce que, par défaut, chaque version de chaque fichier est écrite en entier dans le fichier dump. C'est le comportement le plus simple et le plus rapide et cela convient bien si vous redirigez le flux de données directement vers un autre processus (comme un programme de compression, de filtrage ou de chargement). Mais si vous créez un fichier dump dans une optique de stockage à long terme, vous voudrez sans doute économiser de l'espace disque en utilisant l'option `--deltas`. Avec cette option, les révisions successives des fichiers sont écrites en tant que différences binaires et compressées — de la même manière que pour le stockage des fichiers dans le dépôt. Cette option ralentit le processus mais le fichier résultant a une taille beaucoup plus proche de celle du dépôt original.

Nous avons mentionné auparavant que **svnadmin dump** affiche un intervalle de révisions. Pour spécifier une révision unique ou un intervalle à décharger, utilisez l'option `--revision` (`-r`). Si vous omettez cette option, toutes les révisions existantes sont affichées :

```
$ svnadmin dump mon-depot -r 23 > rev-23.fichier-dump
$ svnadmin dump mon-depot -r 100:200 > revs-100-200.fichier-dump
```

Au fur et à mesure que Subversion décharge chaque nouvelle révision, il n'affiche que le minimum d'informations nécessaire à un futur chargement pour re-générer la révision à partir de la précédente. En d'autres termes, pour n'importe quelle révision du fichier dump, seuls les éléments ayant subi une modification dans cette révision apparaissent dans le fichier dump. La seule exception à cette règle est la première révision qui est déchargée par la commande **svnadmin dump** courante.

Par défaut, Subversion n'exprime pas la première révision déchargée sous forme de différences à appliquer à la révision précédente. En effet, il n'y a pas de révision précédente dans le fichier dump ! Et puis Subversion ne peut pas connaître l'état du dépôt dans lequel les données vont être chargées (si jamais elles le sont). Pour s'assurer que la sortie de chaque exécution de **svnadmin dump** est suffisante, la première révision déchargée est, par défaut, une représentation complète de chaque répertoire, de chaque fichier et de chaque propriété de cette révision du dépôt.

Vous pouvez toujours modifier ce comportement par défaut. Si vous ajoutez l'option `--incremental` quand vous déchargez le dépôt, **svnadmin** compare la première révision déchargée à la révision précédente du dépôt — de la même manière qu'il traite toutes les autres révisions qui sont déchargées. Il affiche alors la première révision de la même manière que le reste des révisions dans l'intervalle demandé — en ne mentionnant que les changements contenus dans cette révision. L'avantage est que vous pouvez créer plusieurs petits fichiers dump qui peuvent être chargés les uns à la suite des autres au lieu d'un unique gros fichier. Par exemple :

```
$ svnadmin dump mon-depot -r 0:1000 > fichier-dump1
$ svnadmin dump mon-depot -r 1001:2000 --incremental > fichier-dump2
$ svnadmin dump mon-depot -r 2001:3000 --incremental > fichier-dump3
```

Ces fichiers dump peuvent maintenant être chargés dans un nouveau dépôt avec la séquence de commandes suivante :

```
$ svnadmin load nouveau-depot < fichier-dump1
$ svnadmin load nouveau-depot < fichier-dump2
$ svnadmin load nouveau-depot < fichier-dump3
```

Une autre astuce consiste à utiliser l'option `--incremental` pour ajouter un nouvel intervalle de révisions à un fichier dump existant. Par exemple, vous pouvez avoir une procédure automatique `post-commit` qui ajoute simplement à un fichier dump le contenu de la révision qui a déclenché la procédure. Ou alors vous pouvez avoir un script qui tourne la nuit pour ajouter à un fichier dump les données de toutes les révisions qui ont eu lieu depuis le dernier lancement du script. Ainsi, **svnadmin dump** est une manière de réaliser des sauvegardes des changements de votre dépôt au fil du temps, dans l'éventualité d'un plantage système ou de toute autre événement catastrophique.

Les fichiers dump peuvent aussi être utilisés pour fusionner le contenu de différents dépôts en un seul dépôt. En utilisant l'option `--parent-dir` de **svnadmin load**, vous pouvez spécifier un nouveau répertoire racine virtuel pour la procédure de chargement. Ainsi, si vous avez des fichiers dump pour trois dépôts — disons `fichier-dump-calc`, `fichier-dump-cal` et `fichier-dump-tab` — vous pouvez commencer par créer un nouveau dépôt pour les héberger tous :

```
$ svnadmin create /var/svn/projets
$
```

Ensuite, créez dans le dépôt les nouveaux répertoires qui vont encapsuler le contenu de chacun des trois dépôts précédents :

```
$ svn mkdir -m "Racines initiales des projets" \
      file:///var/svn/projets/calc \
      file:///var/svn/projets/calendrier \
      file:///var/svn/projets/tableur
Révision 1 propagée.
$
```

Enfin, chargez chaque fichier dump dans le répertoire correspondant du nouveau dépôt :

```
$ svnadmin load /var/svn/projets --parent-dir calc < fichier-dump-calc
…
$ svnadmin load /var/svn/projets --parent-dir calendrier < fichier-dump-cal
```

```
…
$ svnadmin load /var/svn/projets --parent-dir spreadsheet < fichier-dump-tab
…
$
```

Mentionnons une dernière façon d'utiliser les fichiers dump de Subversion — la conversion depuis un système de stockage différent ou depuis un autre système de gestion de versions. Comme le format des fichiers dump est, pour sa plus grande partie, lisible par un humain, il devrait être relativement facile de décrire des ensembles de modifications — chaque ensemble constituant une nouvelle révision — en utilisant ce format de fichier. En fait, l'utilitaire **cvs2svn** (voir la section intitulée « Conversion d'un dépôt CVS vers Subversion ») utilise le format dump pour représenter le contenu d'un dépôt CVS, de manière à pouvoir copier ce contenu dans un dépôt Subversion.

Filtrage de l'historique d'un dépôt

Puisque Subversion stocke votre historique du suivi de versions en utilisant, au minimum, des algorithmes de différenciation binaire et de la compression de données (le tout, en option, dans un système de gestion de bases de données complètement opaque). Il est maladroit, et en tous cas fortement déconseillé, d'essayer de le modifier manuellement, sachant qu'en plus c'est assez difficile. Et une fois que des données ont été stockées dans votre dépôt, Subversion ne fournit généralement pas de moyen simple pour enlever ces données[9]. Mais, inévitablement, il y a des cas où vous voulez manipuler l'historique de votre dépôt. Par exemple pour supprimer tous les occurrences d'un fichier qui a été accidentellement ajouté au dépôt (alors qu'il ne devrait pas y être)[10]. Ou bien lorsque vous avez plusieurs projets qui partagent le même dépôt et que vous décidez de leur attribuer chacun le leur. Pour accomplir ce genre de tâches, les administrateurs ont besoin d'une représentation des données de leurs dépôts plus souple et plus facile à gérer : les fichiers dump Subversion.

Comme indiqué précédemment dans la section intitulée « Migration des données d'un dépôt », le format des fichiers dump Subversion est une représentation lisible par les humains des modifications apportées au cours du temps aux données suivies en versions. Utilisez la commande **svnadmin dump** pour extraire les données et **svnadmin load** pour les charger dans un nouveau dépôt. Le gros atout de l'aspect « lisible par les humains » des fichiers dump est que, si vous y tenez, vous pouvez en inspecter le contenu et le modifier. Bien sûr, la contrepartie est que, si vous avez un fichier dump d'un dépôt actif depuis plusieurs années, cela vous prendra un certain temps pour en inspecter manuellement le contenu et le modifier, un temps certain même.

C'est là qu'intervient **svndumpfilter**. Ce programme agit comme un filtre sur les chemins pour les flux de déchargement/chargement d'un dépôt. Vous n'avez qu'à lui fournir une liste de chemins que vous voulez conserver ou une liste de chemins que vous voulez éliminer et ensuite rediriger le flux de dump de vos données à travers ce filtre. Vous obtenez un flux modifié qui ne contient que les données suivies en versions des chemins que vous avez demandés (explicitement ou implicitement).

Prenons un exemple concret d'utilisation de ce programme. Précédemment dans ce chapitre (voir la section intitulée « Stratégies d'organisation d'un dépôt »), nous avons décrit le processus de décision permettant de choisir l'organisation des données de votre dépôt (utiliser un dépôt par projet ou les combiner, comment organiser les répertoires au sein du dépôt, etc.). Mais il peut arriver qu'après un certain nombre de révisions vous repensiez votre organisation et vouliez la modifier. Une modification classique est de déplacer plusieurs projets qui partagent le même dépôt vers des dépôts propres à chacun d'eux.

Notre dépôt imaginaire contient trois projets : `calc`, `calendrier` et `tableur`. Ils se trouvent côte à côte comme ceci :

```
/
   calc/
      trunk/
      branches/
      tags/
   calendrier/
      trunk/
      branches/
      tags/
   tableur/
      trunk/
      branches/
      tags/
```

[9] C'est d'ailleurs pour cela que vous utilisez un système de gestion de versions, non ?

[10] Des suppressions délibérées et avisées de données suivies en versions peuvent effectivement être justifiées par des cas d'utilisation réels. C'est pourquoi une fonctionnalité « d'oblitération » est une des fonctionnalités les plus demandées pour Subversion et les développeurs de Subversion espèrent pouvoir la fournir bientôt.

Pour placer ces trois projets dans leur dépôts propres, nous commençons par décharger tout le dépôt :

```
$ svnadmin dump /var/svn/depot > fichier-dump-depot
* Révision 0 déchargée.
* Révision 1 déchargée.
* Révision 2 déchargée.
* Révision 3 déchargée.
…
$
```

Ensuite, nous passons ce fichier dump à travers le filtre, en n'incluant à chaque fois qu'un seul répertoire racine. Nous obtenons trois nouveaux fichiers dump :

```
$ svndumpfilter include calc < fichier-dump-depot > fichier-dump-calc
…
$ svndumpfilter include calendrier < fichier-dump-depot > fichier-dump-cal
…
$ svndumpfilter include tableur < fichier-dump-depot > fichier-dump-tab
…
$
```

C'est le moment de prendre une décision. Chacun de vos fichiers dump générera un dépôt valide, mais il conservera les chemins exactement comme ils étaient dans le dépôt original. Cela veut dire que même si vous obtenez un dépôt propre à votre projet `calc` ce dépôt aura toujours un répertoire racine `calc`. Si vous voulez que les répertoires `trunk`, `tags` et `branches` soient placés à la racine de votre dépôt, vous devez alors éditer les fichiers dump, en modifiant les en-têtes `Node-path` et `Node-copyfrom-path` pour qu'ils ne contiennent plus de référence au répertoire `calc/`. Vous devez également supprimer la section des données qui crée le répertoire `calc`. Elle ressemble à ceci :

```
Node-path: calc
Node-action: add
Node-kind: dir
Content-length: 0
```

Si vous envisagez d'éditer à la main le fichier dump pour enlever un répertoire à la racine, assurez-vous que votre éditeur n'est pas configuré pour convertir les caractères de fin de ligne vers le format natif (par exemple de `\r\n` vers `\n`), sinon le contenu ne serait plus conforme aux métadonnées. Cela corromprait le fichier dump de manière irréversible.

Tout ce qu'il reste à faire à présent, c'est de créer vos trois nouveaux dépôts et de charger chaque fichier dump dans le bon dépôt, en ignorant l'UUID contenu dans chaque flux dump :

```
$ svnadmin create calc
$ svnadmin load --ignore-uuid calc < fichier-dump-calc
<<< Début d'une nouvelle transaction basée sur la révision 1
     * ajout de : Makefile ... fait.
     * ajout de : bouton.c ... fait.
…
$ svnadmin create calendrier
$ svnadmin load --ignore-uuid calendrier < fichier-dump-cal
<<< Début d'une nouvelle transaction basée sur la révision 1
     * ajout de : Makefile ... fait.
     * ajout de : cal.c ... fait.
…
$ svnadmin create tableur
$ svnadmin load --ignore-uuid tableur < fichier-dump-tab
```

```
<<< Début d'une nouvelle transaction basée sur la révision 1
     * ajout de : Makefile ... fait.
     * ajout de : tableur.c ... fait.
…
$
```

Les deux sous-commandes **svndumpfilter** possèdent des options pour décider comment traiter les révisions « vides ». Si une révision donnée ne contient que des modifications concernant des chemins qui ont été filtrés, cette révision dorénavant vide peut être considérée comme inintéressante voire indésirable. Pour permettre à l'utilisateur de décider que faire de telles révisions, **svndumpfilter** propose les options suivantes :

`--drop-empty-revs`
: Ne générer aucune révision vide ; elles sont tout simplement ignorées.

`--renumber-revs`
: Si les révisions vides sont ignorées (avec l'option `--drop-empty-revs`), changer les numéros de révision restants pour qu'il n'y ait pas de trous dans la séquence de numérotation.

`--preserve-revprops`
: Si les révisions vides ne sont pas ignorées, garder les propriétés de la révision (message de propagation, auteur, date, propriétés personnalisées, etc.) pour ces révisions vides. Autrement les révisions vides ne contiennent que l'horodatage original et un message expliquant que c'est à cause de **svndumpfilter** que cette révision est vide.

Alors que **svndumpfilter** peut s'avérer très utile et permet de gagner énormément de temps, il est affublé malheureusement de deux chausse-trappes. D'abord, cet utilitaire est extrêmement sensible à la sémantique des chemins. Prêtez attention à la manière dont sont spécifiés les chemins dans votre fichier dump, avec ou sans barre oblique (`/`) initiale. Regardez pour cela les en-têtes `Node-path` et `Node-copyfrom-path`.

```
…
Node-path: tableur/Makefile
…
```

Si les chemins ont une barre oblique initiale, vous devez inclure des barres obliques au début de chaque chemin que vous indiquez à **svndumpfilter include** et **svndumpfilter exclude** (et s'ils n'en ont pas, n'incluez pas de barre oblique au début). Pour aller plus loin, si votre fichier dump contient à la fois des chemins avec et des chemins sans barre oblique initiale, pour quelque raison que ce soit[11], vous devrez probablement normaliser les chemins en adoptant une des deux conventions.

En outre, les chemins qui ont été copiés peuvent vous donner quelques soucis. Subversion supporte les opérations de copie dans le dépôt, c'est-à-dire quand un nouveau chemin est créé par la copie d'un autre chemin qui existe déjà. Il est possible qu'à un certain moment de la vie de votre dépôt, vous ayez copié un fichier ou un répertoire d'un endroit que **svndumpfilter** a exclu vers un endroit qui est inclus. Pour rendre les données du dump cohérentes, **svndumpfilter** doit bien inclure l'ajout du nouveau chemin — y compris le contenu de tous les fichiers créés par la copie — mais en tant que copie d'un chemin source qui n'existe pas dans le flux des données filtrées. Mais puisque le format dump de Subversion ne contient que ce qui a été modifié dans chaque révision, le contenu de la source de la copie risque de ne pas être disponible. Si vous êtes susceptible d'avoir la moindre copie de ce type dans votre dépôt, vous devrez peut-être repenser votre ensemble de chemins à inclure/exclure, pour y inclure aussi les chemins qui ont servi de sources à des opérations de copie qui vous posent problème.

Enfin, **svndumpfilter** effectue un filtrage des chemins pour le moins littéral. Si vous essayez de copier l'historique d'un projet dont la racine est `trunk/mon-projet` et de le déplacer dans son propre dépôt, vous utiliserez évidemment la commande **svndumpfilter include** pour conserver tous les changements dans et sous `trunk/mon-projet`. Mais le fichier dump résultant ne fait aucune hypothèse sur le dépôt dans lequel vous allez charger ces données. En particulier, les données déchargées peuvent commencer par la révision qui a ajouté le répertoire `trunk/mon-projet` mais *ne pas contenir* les directives pour créer le répertoire `trunk` lui-même (parce que `trunk` ne correspond pas au filtre utilisé). Vous devez vous assurer que tous les répertoires à la présence desquels le flux de données déchargées s'attend existent réellement dans le dépôt destination, avant d'essayer de charger le flux de données à l'intérieur.

[11] Bien que **svnadmin dump** ait une politique cohérente concernant la barre oblique initiale (aussi appelée « slash » — il ne l'inclut pas), d'autres programmes qui génèrent des fichiers dump sont susceptibles de ne pas être aussi cohérents.

Réplication d'un dépôt

Divers scénarios montrent l'intérêt d'avoir un dépôt Subversion dont l'historique des versions est exactement le même que celui d'un autre dépôt. Le plus évident est probablement celui de maintenir un dépôt de secours, utilisé quand le dépôt principal est inaccessible en raison d'un problème matériel, d'une coupure réseau ou de tout autre souci de ce type. D'autres scénarios comprennent le déploiement de dépôts redondants pour distribuer la charge sur plusieurs serveurs, les mises à niveau transparentes et d'autres encore.

Depuis la version 1.4, Subversion fournit un programme pour gérer de tels scénarios : **svnsync**. Il fonctionne essentiellement en demandant au serveur Subversion de « rejouer » les révisions, une par une. Il utilise ces informations sur les révisions pour répéter une propagation identique sur un autre dépôt. Aucun des deux dépôts n'a besoin d'être accessible localement sur la machine où **svnsync** tourne : ses paramètres sont des URL de dépôt et tout le travail est effectué via les interfaces d'accès au dépôt (*Repository Access* en anglais, ou *RA*) de Subversion. Tout ce dont il a besoin est un accès en lecture au dépôt source et un accès en lecture/écriture au dépôt de destination.

Quand vous utilisez **svnsync** sur un dépôt source distant, le serveur Subversion de ce dépôt doit être en version 1.4 ou supérieure.

Supposons que vous voulez réaliser un miroir d'un de vos dépôts ; il vous faut alors disposer d'un dépôt destination vide qui servira de miroir. Ce dépôt cible peut utiliser n'importe quel magasin de données disponible (voir la section intitulée « Choix du magasin de données »), mais il ne doit comporter aucun historique. Le protocole utilisé par **svnsync** pour transmettre les informations de révision est particulièrement sensible aux divergences entre les historiques suivies en versions de la source et de la destination. Pour cette raison, bien que **svnsync** ne puisse pas *exiger* que le dépôt destination soit en lecture seule[12], autoriser des modifications d'historique sur le dépôt destination par un mécanisme externe autre que le processus de réplication mène droit au désastre.

Ne modifiez pas le dépôt miroir de sorte que son historique de version diffère de celui du dépôt source. Les seules propagations et modifications de propriétés de révisions qui doivent avoir lieu sur ce dépôt miroir sont celles effectuées par l'outil **svnsync**.

Une autre exigence concernant le dépôt destination est que le processus **svnsync** doit être autorisé à modifier les propriétés de révision. Comme **svnsync** fonctionne dans le cadre du système des procédures automatiques du dépôt, l'état par défaut du dépôt (qui consiste à interdire les modifications des propriétés de révision, voir pre-revprop-change) n'est pas suffisant. Vous devez activer explicitement la procédure automatique `pre-revprop-change` et votre script doit autoriser **svnsync** à définir et à modifier les propriétés de révision. Une fois ces dispositions prises, vous êtes parés pour commencer la réplication des révisions du dépôt.

Il est de bon ton de mettre un place un contrôle d'accès pour autoriser le processus de réplication de votre dépôt à faire ce qu'il a à faire tout en interdisant aux autres utilisateurs de modifier le contenu de votre dépôt miroir.

Examinons maintenant l'utilisation de **svnsync** dans un scénario classique de réplication. Nous saupoudrons le discours de quelques recommandations pratiques que vous êtes libre d'ignorer si elles ne sont pas nécessaires ou pas applicables à votre environnement.

Pour rendre service aux excellents développeurs de notre système de gestion de versions favori, nous allons répliquer le dépôt public qui contient le code source de Subversion et mettre ce miroir à disposition sur Internet, sur une machine différente de celle qui héberge le dépôt original. Cet hôte distant possède une configuration globale qui autorise les accès anonymes en lecture mais requiert une authentification pour modifier les dépôts (pardonnez-nous de passer rapidement sur les détails de la configuration du serveur Subversion pour le moment, mais ces aspects sont traités en profondeur dans le Chapitre 6, *Configuration du serveur*.). Et pour rendre l'exemple plus intéressant, et uniquement pour cela, nous piloterons la réplication depuis une troisième machine — en l'occurrence, celle que nous sommes en train d'utiliser.

Dans un premier temps, nous allons créer le dépôt qui servira de miroir. Cette étape et les deux suivantes requièrent l'accès à la ligne de commande de la machine sur laquelle le miroir sera hébergé. Toutefois, une fois que ce dépôt sera complètement configuré, nous n'aurons plus besoin d'y avoir accès directement.

[12] En fait, le dépôt ne peut pas être complètement en lecture seule, sinon **svnsync** lui-même aurait du mal à y copier l'historique des révisions.

```
$ ssh admin@svn.exemple.com \
      "svnadmin create /var/svn/miroir-svn"
admin@svn.exemple.com's password: ********
$
```

À ce stade, nous disposons d'un dépôt et, en raison de la configuration de notre serveur, ce dépôt est accessible directement depuis Internet. Maintenant, puisque nous ne voulons pas que quoi que ce soit modifie notre dépôt en dehors du processus de réplication, nous devons trouver un moyen de distinguer ce processus des autres prétendants aux propagations. Pour ce faire, nous utilisons un identifiant d'utilisateur dédié à notre processus. Seules les propagations et les modifications de propriétés de révisions effectuées par l'identifiant spécial `id-sync` sont autorisées.

Nous allons utiliser le système de procédures automatiques du dépôt à la fois pour autoriser le processus de réplication à faire ce qu'il doit faire et pour garantir qu'il soit le seul à le faire. Nous implémentons donc deux des procédures automatiques du dépôt : `pre-revprop-change` et `start-commit`. Le script `pre-revprop-change` est présenté dans l'Exemple 5.2, « Procédure automatique pre-revprop-change du dépôt miroir » et, pour résumer, vérifie que l'utilisateur qui essaie de modifier les propriétés est bien notre utilisateur `id-sync`. Si c'est bien le cas, la modification est autorisée ; sinon, elle est refusée.

Exemple 5.2. Procédure automatique pre-revprop-change du dépôt miroir

```
#!/bin/sh

USER="$3"

if [ "$USER" = "id-sync" ]; then exit 0; fi

echo "Seul l'utilisateur id-sync est autorisé à modifier les propriétés de révision."
>&2
exit 1
```

Voilà pour les modifications des propriétés de révision. Maintenant nous devons nous assurer que seul l'utilisateur `id-sync` est autorisé à propager de nouvelles révisions dans le dépôt. Ce que nous allons faire en utilisant une procédure automatique `start-commit` telle que celle présentée dans l'Exemple 5.3, « Procédure automatique start-commit du dépôt miroir ».

Exemple 5.3. Procédure automatique start-commit du dépôt miroir

```
#!/bin/sh

USER="$2"

if [ "$USER" = "id-sync" ]; then exit 0; fi

echo "Seul l'utilisateur id-sync est autorisé à effectuer des propagations." >&2
exit 1
```

Après avoir installé nos procédures automatiques et s'être assuré qu'elles sont exécutables par le serveur Subversion, nous en avons terminé avec l'installation de notre dépôt miroir. Maintenant, nous allons effectivement lancer la réplication.

La première chose à faire avec **svnsync** est d'enregistrer dans notre dépôt destination le fait qu'il sera un miroir du dépôt source. Nous utilisons donc la sous-commande **svnsync initialize**. Nous fournissons des URL qui pointent vers les répertoires racines des dépôts destination et source, respectivement. Dans Subversion 1.4, c'est obligatoire — seule la réplication de dépôts complets est permise. Dans Subversion 1.5, cependant, vous pouvez aussi utiliser **svnsync** pour répliquer uniquement des sous-arborescences du dépôt.

```
$ svnsync help init
initialize (init): usage : svnsync initialize DEST_URL SOURCE_URL

Initialise un dépôt destination pour être synchronisé à partir
d'un autre dépôt.
…
$ svnsync initialize http://svn.exemple.com/miroir-svn \
                     http://svn.collab.net/repos/svn \
                     --sync-username id-sync --sync-password mdp-sync
Propriétés copiées pour la révision 0.
$
```

Notre dépôt destination se rappelle maintenant qu'il est un miroir du dépôt public du code source Subversion[13]. Notez que nous avons fourni un identifiant et un mot de passe en arguments à **svnsync** — c'était exigé par la procédure automatique `pre-revprop-change` de notre dépôt miroir.

Dans Subversion 1.4, les valeurs assignées aux options `--username` et `--password` de **svnsync** étaient utilisées pour l'authentification à la fois par le dépôt source et par le dépôt destination. Cela posait des problèmes quand ces valeurs n'étaient pas exactement les mêmes pour les deux dépôts, particulièrement quand le mode non-interactif était utilisé (avec l'option `--non-interactive`).

Ce problème a été résolu dans la version 1.5 de Subversion avec l'introduction de deux nouvelles paires d'options. Utilisez `--source-username` et `--source-password` pour vous authentifier auprès du dépôt source ; utilisez `--sync-username` et `--sync-password` pour vous authentifier auprès du dépôt destination (les vieilles options `--username` et `--password` existent encore pour assurer la compatibilité mais nous en déconseillons l'usage).

Abordons maintenant la partie amusante. En une seule sous-commande, nous pouvons demander à **svnsync** de copier toutes les révisions qui n'ont pas encore été répliquées du dépôt source vers le dépôt destination[14]. La sous-commande **svnsync synchronize** fouille dans les propriétés de révision spéciales du dépôt destination pour déterminer aussi bien de quel dépôt source il est le miroir que la dernière révision qui a été répliquée, en l'occurrence la révision 0. Ensuite, elle interroge le dépôt source pour savoir quelle est la dernière révision propagée dans ce dépôt. Enfin, elle demande au dépôt source de commencer à envoyer toutes les révisions entre la révision 0 et la dernière révision. Au moment où **svnsync** reçoit la réponse du dépôt source, elle commence la retransmission des révisions vers le dépôt destination en tant que nouvelles propagations.

```
$ svnsync help synchronize
synchronize (sync): usage : svnsync synchronize URL_DEST

Transfère toutes les révisions en attente vers la destination,
à partir de la source avec laquelle elle a été initialisée.
…
$ svnsync synchronize http://svn.exemple.com/miroir-svn
Transmission des données .....................................
Révision 1 propagée.
Propriétés copiées pour la révision 1.
Transmission des données ..
Révision 2 propagée.
Propriétés copiées pour la révision 2.
Transmission des données .....
Révision 3 propagée.
Propriétés copiées pour la révision 3.
…
Transmission des données ..
Révision 23406 propagée.
Propriétés copiées pour la révision 23406.
Transmission des données .
Révision 23407 propagée.
Propriétés copiées pour la révision 23407.
Transmission des données ....
```

[13] NdT : il s'agit ici de l'ancienne adresse du dépôt Subversion.

[14] Nous avertissons le lecteur que, bien qu'il lui suffise de quelques secondes pour lire ce paragraphe et l'exemple qui suit, le temps nécessaire pour réaliser une opération de réplication est, disons, un peu plus long.

```
Révision 23408 propagée.
Propriétés copiées pour la révision 23408.
$
```

Il est intéressant de noter ici que, pour chaque révision répliquée, il y a d'abord propagation de la révision dans le dépôt destination, puis des changements de propriétés ont lieu. C'est parce que la propagation initiale est effectuée par (et donc attribuée à) l'utilisateur `id-sync` et qu'elle est horodatée lors de la création de la nouvelle révision. Et aussi parce que les interfaces d'accès au dépôt Subversion n'autorisent pas la définition de propriétés de révision au sein d'une propagation. C'est pourquoi **svnsync** fait suivre la réplication par une série de modifications de propriétés qui copient dans le dépôt destination toutes les propriétés de révision trouvées dans le dépôt source pour cette révision. Cela a également pour effet de corriger l'auteur et l'horodatage de la révision pour être cohérent avec le dépôt source.

Notez également que **svnsync** documente tout ce qu'il fait en détail, afin de pouvoir être interrompu ou redémarré sans remettre en cause l'intégrité des données répliquées. Si une panne réseau survient pendant la réplication d'un dépôt, relancez simplement la commande **svnsync synchronize** et elle reprendra tranquillement là où elle s'était arrêtée. En fait, au fur et à mesure que de nouvelles révisions apparaissent dans le dépôt source, c'est précisément ce qu'il faut faire pour conserver votre miroir à jour.

Propriétés propres à svnsync

svnsync a besoin de pouvoir définir et modifier des propriétés de révision dans le dépôt miroir car ces propriétés font partie intégrante des données à répliquer. Au fur et à mesure que ces propriétés changent dans le dépôt source, ces changements doivent également être répliqués dans le dépôt miroir. Mais **svnsync** utilise aussi un ensemble de propriétés de révision qui lui sont propres — stockées dans la révision 0 du dépôt miroir — à ses propres fins de journalisation. Ces propriétés contiennent des informations telles que l'URL et l'UUID du dépôt source, ainsi que quelques informations supplémentaires sur l'état du miroir.

Une de ces informations sur l'état du miroir est un drapeau indiquant qu'« il y a une synchronisation en cours ». Il est utilisé pour éviter que de multiples processus **svnsync** entrent en collision en essayant de répliquer les données vers le même dépôt destination. Quoi qu'il en soit, vous n'aurez généralement pas à vous soucier de ces propriétés spéciales (elles commencent toutes par le préfixe `svn:sync-`). Cependant, il peut arriver qu'une synchronisation échoue accidentellement et que Subversion n'ait pas l'occasion d'enlever ce drapeau indiquant l'état de la synchronisation. Cela fait échouer toute nouvelle tentative de synchronisation, puisque le drapeau indique qu'une synchronisation est en cours, alors que ce n'est pas le cas. Heureusement, pour sortir de cette situation, il suffit de supprimer la propriété `svn:sync-lock` de la révision 0 du dépôt miroir (c'est le fameux drapeau) :

```
$ svn propdel --revprop -r0 svn:sync-lock http://svn.exemple.com/miroir-svn
Propriété 'svn:sync-lock' supprimée de la révision 0 du dépôt
$
```

Le fait que **svnsync** stocke l'URL du dépôt source dans une propriété qui lui est dédiée au sein du dépôt destination est la raison pour laquelle vous n'avez à renseigner cette URL qu'une seule fois, au moment de **svnsync init**. Les opérations suivantes de synchronisation de ce miroir se contentent de consulter la propriété `svn:sync-from-url` stockée dans le dépôt miroir lui-même pour déterminer la source de la synchronisation. Notez bien que cette valeur est utilisée telle quelle par le processus de synchronisation. Ainsi, depuis l'intérieur du réseau CollabNet vous pouvez peut-être accéder à notre URL source `http://svn/depot/svn` (parce que le premier `svn` se voit ajouter `.collab.net` par la magie du serveur DNS), mais, si vous devez mettre à jour le miroir plus tard depuis une machine en dehors du réseau CollabNet, la synchronisation risque d'échouer (parce que le nom de machine `svn` est ambigu). Pour cette raison, il est préférable d'utiliser le nom complet pour l'URL du dépôt source lors de l'initialisation de votre dépôt miroir plutôt que simplement le nom de machine ou l'adresse IP (qui peut varier au cours du temps). Là encore, si vous devez changer l'URL de la source (ayant le même contenu) d'un dépôt miroir existant, vous pouvez changer la propriété de journalisation qui stocke cette information :

```
$ svn propset --revprop -r0 svn:sync-from-url NOUVELLE-URL-SOURCE \
      http://svn.exemple.com/miroir-svn
Propriété 'svn:sync-from-url' définie à la révision 0 du dépôt
$
```

Un autre point intéressant concernant ces propriétés liées à la synchronisation est que **svnsync** n'essaie pas de répliquer

> ces propriétés s'il les trouve dans le dépôt source. La raison en est probablement évidente mais est due en résumé au fait que **svnsync** n'est pas capable de distinguer les propriétés spéciales qu'il n'a fait que copier à partir du dépôt source de celles qu'il doit consulter et tenir à jour pour ses propres besoins. Cette situation peut arriver si, par exemple, vous hébergez le miroir d'un miroir d'un dépôt. Quand **svnsync** voit ses propres propriétés de synchronisation dans la révision 0 du dépôt source, il les ignore purement et simplement.

Le procédé est cependant peu élégant. Comme les propriétés des révisions Subversion peuvent être modifiées n'importe quand dans la vie du dépôt et comme elles ne conservent pas de trace des modifications effectuées, les processus de réplication doivent faire particulièrement attention à ces propriétés. Si vous avez déjà répliqué les quinze premières révisions d'un dépôt et que quelqu'un modifie une propriété de révision concernant la révision 12, **svnsync** ne sait pas qu'il faut revenir en arrière et modifier la copie de la révision 12. Vous devez le lui indiquer manuellement en utilisant la sous-commande **svnsync copy-revprops** (ou à l'aide d'autres outils), ce qui re-réplique toutes les propriétés de révision pour une révision particulière ou un ensemble de révisions.

```
$ svnsync help copy-revprops
copy-revprops: usage: svnsync copy-revprops URL_DEST [REV[:REV2]]

Copie les propriétés de révision pour l'intervalle donné vers la destination
à partir de la source avec laquelle elle a été initialisée.
…
$ svnsync copy-revprops http://svn.exemple.com/miroir-svn 12
Propriétés copiées pour la révision 12.
$
```

C'en est fini pour la présentation rapide de la réplication de dépôt. Vous voudrez sûrement automatiser un certain nombre de choses autour de ce processus. Par exemple, alors que notre exemple présentait une mise en place « tirer-et-pousser », vous êtes susceptible de vouloir que ce soit le dépôt source qui pousse les modifications vers un ou plusieurs miroirs prédéfinis lors de l'exécution des procédures automatiques `post-commit` et `post-revprop-change`. Ceci permettrait au miroir d'être à jour presque en temps réel.

En outre, et bien que ce ne soit pas très utilisé, **svnsync** sait répliquer des dépôts pour lesquels l'identifiant qu'il utilise pour s'authentifier n'a que des droits partiels en lecture. Il copie simplement les parties du dépôt qu'il est autorisé à voir. Un tel miroir n'est clairement pas une bonne solution de sauvegarde.

Avec Subversion 1.5, **svnsync** a acquis la capacité de répliquer uniquement un sous-ensemble d'un dépôt plutôt que le dépôt entier. La procédure pour configurer et assurer la maintenance d'un tel miroir est exactement la même que pour répliquer un dépôt entier, excepté lors du passage de l'URL du dépôt source à **svnsync init** : vous spécifiez l'URL d'un sous-répertoire à l'intérieur du dépôt. La synchronisation du miroir ne copie que les modifications relatives à l'arborescence sous le répertoire indiqué. Notez quand même quelques restrictions sur cette fonction : premièrement, vous ne pouvez pas répliquer plusieurs sous-répertoires disjoints du dépôt source vers un unique dépôt miroir — vous devez dans ce cas répliquer un répertoire parent qui est commun à tous les répertoires que vous voulez répliquer ; deuxièmement, la logique de filtrage est entièrement basée sur le chemin d'accès donc, si le sous-répertoire que vous répliquez a été renommé par le passé, votre miroir ne contiendra que les révisions depuis lesquelles il a le nom indiqué dans l'URL que vous avez spécifiée. Et, de la même manière, si le sous-répertoire source est renommé dans le futur, le processus de synchronisation ne répliquera plus les données à partir du moment où l'URL que vous avez spécifiée ne sera plus valide.

En ce qui concerne les interactions entre les utilisateurs et les dépôts ainsi que les miroirs, *il est possible* d'avoir une seule copie de travail qui interagisse avec un dépôt et son miroir, mais vous devez faire quelques manipulations pour y arriver. D'abord, vous devez vous assurer que le dépôt source et le dépôt miroir ont bien le même identifiant unique UUID (ce qui n'est pas le cas par défaut). Reportez-vous à la section intitulée « Gestion des identifiants uniques (UUID) des dépôts » plus loin dans ce chapitre pour plus de détails à ce sujet.

Une fois que les deux dépôts ont le même identifiant unique, vous pouvez utiliser **svn switch** avec l'option `--relocate` pour faire pointer votre copie de travail vers le dépôt de votre choix ; cette procédure est décrite dans svn switch. Cette manœuvre est potentiellement dangereuse si le miroir et le dépôt original ne sont pas étroitement synchronisés : une copie de travail à jour, pointant vers le dépôt source, que l'on ferait pointer vers un miroir non à jour, est perdue devant l'absence soudaine de révisions qu'elle s'attend à trouver et elle renvoie des erreurs dans ce sens. Si cela arrive, vous pouvez refaire pointer votre copie de travail vers le dépôt original et soit attendre que le dépôt miroir se mette à jour, soit faire revenir en arrière votre copie de travail jusqu'à un numéro de révision dont vous savez qu'il est présent dans le dépôt miroir, puis retenter le changement de dépôt.

Enfin, soyez conscient que la réplication fournie par **svnsync**, basée sur les révisions, n'est rien de plus que de la simple réplication de révisions. Seules les informations incluses dans le format de fichier dump des dépôts Subversion peuvent être répliquées. Ainsi, **svnsync** possède les mêmes limitations que les flux de chargement/déchargement Subversion et n'inclut donc pas les procédures automatiques, la configuration du dépôt et du serveur, les transactions inachevées ni les informations relatives aux verrous posés par les utilisateurs sur les chemins du dépôt.

Sauvegarde d'un dépôt

En dépit des nombreux progrès de la technologie depuis l'avènement de l'informatique moderne, une chose reste certaine : le pire n'est jamais très loin. L'alimentation électrique tombe en panne, les réseaux subissent des coupures, les mémoires vives crament, les disques durs flanchent, et ce même pour le plus consciencieux des administrateurs. Nous en venons donc maintenant à un sujet très important : comment réaliser des copies de sauvegarde des données de votre dépôt.

Les administrateurs de dépôts Subversion disposent de deux méthodes de sauvegarde : la complète et l'incrémentale. Une sauvegarde complète d'un dépôt implique de récupérer d'un seul coup toutes les informations nécessaires pour reconstruire complètement ce dépôt dans le cas d'une catastrophe. Habituellement, cela consiste à dupliquer, littéralement, la totalité du répertoire du dépôt (ce qui inclut l'environnement Berkeley DB ou FSFS). Les sauvegardes incrémentales sont plus restreintes : elles ne sauvegardent que les données du dépôt qui ont changé depuis la sauvegarde précédente.

Pour ce qui concerne les sauvegardes complètes, l'approche naïve pourrait sembler satisfaisante. Mais à moins d'interdire temporairement tout accès au dépôt, une simple copie récursive du répertoire risque de créer une sauvegarde corrompue. Dans le cas d'une base de données Berkeley DB, la documentation décrit un certain ordre de copie des fichiers qui garantit une copie de sauvegarde valide. Un ordre similaire existe avec les données FSFS. Mais vous n'avez pas à implémenter ces algorithmes vous-même, puisque l'équipe de développement de Subversion l'a déjà fait pour vous. La commande **svnadmin hotcopy** prend soin de tout ça afin de réaliser une copie à chaud de votre dépôt. Et son invocation est aussi triviale que la commande Unix **cp** ou qu'une copie sous Windows :

```
$ svnadmin hotcopy /var/svn/depot /var/svn/depot-sauvegarde
```

La sauvegarde générée est un dépôt Subversion totalement fonctionnel, capable de prendre immédiatement la place de votre dépôt en production si les choses tournent au vinaigre.

Lors de la copie d'un dépôt Berkeley DB, vous pouvez même ordonner à **svnadmin hotcopy** de purger les fichiers de journalisation inutilisés (voir la section intitulée « Purge des fichiers de journalisation inutilisés de Berkeley DB ») du dépôt original une fois la copie terminée. Ajoutez simplement l'option `--clean-logs` à la ligne de commande :

```
$ svnadmin hotcopy --clean-logs /var/svn/depot-bdb /var/svn/depot-bdb-sauvegarde
```

Des outils additionnels existent également autour de cette commande. Le répertoire `tools/backup` du code source de Subversion contient le script **hot-backup.py**. Ce script ajoute de la gestion de sauvegardes par-dessus **svnadmin hotcopy**, permettant de ne garder que les dernières sauvegardes (le nombre de sauvegardes à conserver est configurable) de chaque dépôt. Il gère automatiquement les noms des répertoires sauvegardés pour éviter les collisions avec les précédentes sauvegardes et élimine par rotation les sauvegardes les plus anciennes. Même si vous réalisez aussi des sauvegardes incrémentales, cette commande vous servira peut-être régulièrement. Par exemple, vous pouvez utiliser **hot-backup.py** dans un outil permettant le lancement différé de commandes (tel que **cron** sur les systèmes Unix) afin de le lancer toutes les nuits (ou à tout autre intervalle de temps qui vous convient mieux).

Quelques administrateurs utilisent un autre mécanisme de sauvegarde basé sur la génération et le stockage de flux dump des dépôts. Nous avons décrit dans la section intitulée « Migration des données d'un dépôt » comment utiliser **svnadmin dump** avec l'option `--incremental` pour réaliser une sauvegarde incrémentale d'une révision ou d'un intervalle de révisions donné. Et bien sûr, vous pouvez réaliser une sauvegarde complète en omettant l'option `--incremental` dans la commande. Cette méthode comporte certains avantages, entre autres que le format de vos informations sauvegardées est flexible (il n'est pas lié à une plate-forme, à un type de magasin de données ou à une version particulière de Subversion ou de Berkeley DB). Mais cette flexibilité a un coût, à savoir le temps de restauration des données, qui en plus augmente avec chaque nouvelle révision propagée dans le dépôt. Aussi, comme c'est le cas pour un tas d'autres méthodes de sauvegarde, les modifications sur les propriétés de révision qui sont effectuées après une sauvegarde de ladite révision ne sont pas prises en compte par l'utilisation de flux de dump incrémentaux ne se chevauchant pas. C'est pourquoi nous recommandons de ne pas se reposer uniquement sur le type de sauvegarde basé sur les fichiers dump.

Comme vous pouvez le constater, chaque type de sauvegarde a ses avantages et ses inconvénients. La méthode la plus facile est de loin la sauvegarde à chaud complète, qui fournit toujours une copie conforme et fonctionnelle de votre dépôt. En cas d'accident sur votre dépôt de production, vous pouvez effectuer une restauration à partir de votre sauvegarde par une simple copie récursive de répertoire. Malheureusement, si vous maintenez plusieurs sauvegardes de votre dépôt, chaque sauvegarde consomme autant d'espace disque que votre dépôt en production. Les sauvegardes incrémentales, en revanche, sont plus rapides à réaliser et prennent moins de place. Mais le processus de restauration peut s'avérer pénible, avec souvent plusieurs sauvegardes incrémentales à appliquer. D'autres méthodes ont leurs propres bizarreries. Les administrateurs doivent trouver le juste milieu entre le coût des sauvegardes et le coût de la restauration.

Le programme **svnsync** (voir la section intitulée « Réplication d'un dépôt ») fournit en fait une approche médiane assez pratique. Si vous synchronisez régulièrement un miroir en lecture seule avec votre dépôt principal, ce miroir en lecture seule se révèle être un bon candidat pour prendre le relais du dépôt défaillant en cas de besoin. Le principal inconvénient de cette méthode est que seules les données suivies en versions sont synchronisées — les fichiers de configuration du dépôt, les verrous utilisateurs sur les chemins ou d'autres éléments stockés physiquement dans le répertoire du dépôt mais pas *dans le système de fichiers virtuel du dépôt* ne sont pas pris en charge par **svnsync**.

Quelle que soit la méthode de sauvegarde utilisée, les administrateurs doivent savoir comment les modifications des propriétés non suivies en versions des révisions sont prises en compte (ou pas). Puisque ces modifications elles-mêmes ne créent pas de nouvelles révisions, elles n'activent pas les procédures automatiques `post-commit` ni même éventuellement les procédures automatiques `pre-revprop-change` et `post-revprop-change`[15]. Et puisque vous pouvez modifier les propriétés de révision sans respecter l'ordre chronologique (vous pouvez changer n'importe quelle propriété de révision à n'importe quel moment), une sauvegarde incrémentale des dernières révisions pourrait ne pas intégrer la modification d'une propriété de révision qui faisait partie d'une sauvegarde précédente.

En règle générale, seuls les plus paranoïaques ont besoin de sauvegarder le dépôt entier, disons, à chaque propagation. Cependant, en considérant qu'un dépôt donné possède des mécanismes de redondance autres avec une certaine granularité (tels que des e-mails envoyés à chaque propagation ou des fichiers dumps incrémentaux), réaliser une copie à chaud de la base de données, dans le cadre des sauvegardes nocturnes quotidiennes des systèmes, est une bonne pratique d'administration. Ce sont vos données, protégez-les autant que vous le voulez.

Bien souvent, la meilleure approche de sauvegarde d'un dépôt consiste à ne pas mettre tous ses œufs dans le même panier, en utilisant une combinaison des méthodes décrites ici. Les développeurs Subversion, par exemple, sauvegardent le code source de Subversion chaque nuit en utilisant **hot-backup.py** et effectuent une copie distante par **rsync** de ces sauvegardes complètes ; ils conservent plusieurs archives de tous les emails de notification des propagations et des changements de propriétés ; ils ont également des miroirs maintenus par divers volontaires qui utilisent **svnsync**. Votre solution peut ressembler à cela, mais elle doit être adaptée à vos besoins et maintenir l'équilibre entre commodité et paranoïa. Et quoi que vous fassiez, vérifiez la validité de vos sauvegardes de temps en temps (à quoi servirait une roue de secours crevée ?). Bien que tout ceci n'empêche pas votre matériel de subir les affres du destin[16], cela vous aidera certainement à vous sortir de ces situations délicates.

Gestion des identifiants uniques (UUID) des dépôts

Chaque dépôt Subversion possède un identifiant unique (*Universally Unique IDentifier* en anglais ou *UUID*). Cet UUID est utilisé par les clients Subversion pour vérifier l'identité d'un dépôt quand les autres formes de vérification ne sont pas satisfaisantes (telles que l'URL du dépôt qui peut varier avec le temps). La plupart des administrateurs de dépôt n'ont jamais (ou très rarement) à se préoccuper des UUID autrement que comme d'un détail d'implémentation bas niveau de Subversion. Parfois, cependant, il faut prêter attention à ce détail.

En règle générale, les UUID de vos dépôts de production doivent être uniques. C'est le but, après tout, des UUID. Mais il y a des cas où vous voulez que les UUID de deux dépôts soient identiques. Par exemple, quand vous faites une copie de sauvegarde d'un dépôt, vous voulez que cette sauvegarde soit une réplique exacte de l'original afin que dans le cas où vous restaureriez la sauvegarde pour remplacer le dépôt de production, ceci soit transparent pour les utilisateurs. Quand vous déchargez et chargez l'historique d'un dépôt (comme décrit précédemment dans la section intitulée « Migration des données d'un dépôt »), vous devez décider si vous incluez l'UUID dans le flux de données déchargées (le fichier dump) qui va dans le nouveau dépôt. Les circonstances vous imposeront la marche à suivre.

Il y a plusieurs façons de faire pour attribuer (ou modifier) un UUID à un dépôt, le cas échéant. Avec Subversion 1.5, il suffit d'utiliser la commande **svnadmin setuuid**. Si vous fournissez un UUID explicite à cette sous-commande, elle valide le format de l'UUID et fixe l'identifiant unique du dépôt à cette valeur. Si vous omettez l'UUID, un nouvel UUID est généré

[15] La commande **svnadmin setlog** peut être utilisée de manière à contourner les procédures automatiques.
[16] Vous savez, le fameux « concours de circonstances », celui auquel vous êtes arrivé premier.

automatiquement pour votre dépôt.

```
$ svnlook uuid /var/svn/depot
cf2b9d22-acb5-11dc-bc8c-05e83ce5dbec
$ svnadmin setuuid /var/svn/depot   # créer un nouvel UUID
$ svnlook uuid /var/svn/depot
3c3c38fe-acc0-11dc-acbc-1b37ff1c8e7c
$ svnadmin setuuid /var/svn/depot \
           cf2b9d22-acb5-11dc-bc8c-05e83ce5dbec  # restaure l'ancien UUID
$ svnlook uuid /var/svn/depot
cf2b9d22-acb5-11dc-bc8c-05e83ce5dbec
$
```

Pour ceux qui utilisent une version de Subversion antérieure à 1.5, ces tâches sont un peu plus compliquées. Vous pouvez attribuer explicitement un UUID en redirigeant le flux d'un fichier dump qui comporte le nouvel UUID avec la commande **svnadmin load --force-uuid *CHEMIN-DU-DEPOT***.

```
$ svnadmin load --force-uuid /var/svn/depot <<EOF
SVN-fs-dump-format-version: 2

UUID: cf2b9d22-acb5-11dc-bc8c-05e83ce5dbec
EOF
$ svnlook uuid /var/svn/depot
cf2b9d22-acb5-11dc-bc8c-05e83ce5dbec
$
```

Faire générer un nouvel UUID à une ancienne version de Subversion n'est pas aussi simple. La meilleure façon de faire est certainement de trouver un moyen de générer un UUID puis d'affecter explicitement cet UUID au dépôt.

Déplacement et suppression d'un dépôt

Les données d'un dépôt Subversion sont toutes contenues dans l'arborescence du répertoire du dépôt. Ainsi, vous pouvez déplacer un dépôt Subversion vers un autre endroit du disque, renommer un dépôt, copier un dépôt ou effacer un dépôt entier en utilisant les outils de votre système d'exploitation pour manipuler les répertoires — **mv**, **cp -a** et **rm -r** pour les plates-formes Unix ; **copy**, **move** et **rmdir /s /q** sous Windows ; cliquodrome avec de nombreux explorateurs graphiques, etc.

Bien sûr, il y a souvent d'autres choses à faire lors de ce genre de manipulations. Par exemple, vous voulez peut-être mettre à jour la configuration de votre serveur Subversion pour le faire pointer vers le nouveau chemin du dépôt déplacé ou pour supprimer les éléments de configuration relatifs à un dépôt ayant été effacé. Si vous avez des processus automatiques qui publient des informations à partir de vos dépôts, ou qui leur sont relatives, ils doivent sûrement être mis à jour. La configuration des procédures automatiques a peut-être besoin d'être modifiée. Les utilisateurs doivent être informés. La liste est longue, mais vous avez sûrement des procédures d'exploitation et des règles d'administration de votre dépôt Subversion qui vous indiquent ce qu'il faut faire.

Dans le cas d'un dépôt copié, vous devez aussi prendre en compte le fait que Subversion utilise l'identifiant unique UUID pour distinguer les dépôts. Si vous copiez un dépôt en utilisant la commande typique de copie récursive de votre interpréteur de commande, vous vous retrouvez avec deux dépôts absolument identiques, y compris en ce qui concerne l'UUID. Dans certains cas, c'est ce que l'on cherche. Dans d'autres cas, non. Vous devez alors générer un nouvel UUID pour l'un des deux dépôts identiques. Lisez la section intitulée « Gestion des identifiants uniques (UUID) des dépôts » pour plus d'informations sur la gestion des UUID.

Résumé

À présent vous devez avoir une compréhension de base de la création, la configuration et l'administration des dépôts Subversion. Nous vous avons présenté les différents outils qui vous assistent dans ces tâches. Tout au long de ce chapitre, nous avons identifié quelques chausse-trappes d'administration et proposé quelques solutions pour les éviter.

Tout ce qui vous reste à faire est de décider quelles données passionnantes vous allez héberger dans votre dépôt et, finalement, comment les mettre à disposition sur un réseau. Le prochain chapitre est entièrement consacré au travail en réseau.

Chapitre 6. Configuration du serveur

Un dépôt Subversion hébergé sur une machine donnée est accessible simultanément par des clients fonctionnant sur cette même machine en utilisant la méthode `file://`. Mais la configuration typique de Subversion consiste en une machine serveur unique à laquelle accèdent des clients tournant sur les ordinateurs de tout le bureau — ou, peut-être, de partout dans le monde.

Ce chapitre décrit comment rendre visible votre dépôt Subversion en dehors de la machine hôte pour qu'il soit utilisable par des clients distants. Nous couvrons les mécanismes serveurs disponibles actuellement pour Subversion, leur configuration et leur utilisation. À la fin de ce chapitre, vous devriez être apte à décider quelle configuration réseau convient à vos besoins et comprendre comment mettre en place cette configuration sur votre machine hôte.

Présentation générale

Subversion a été conçu avec une couche réseau abstraite. Cela signifie que l'on peut accéder de façon automatisée à un dépôt par toute sorte de protocoles client/serveur, et que l'interface (l'API) d'accès au dépôt du client permet aux programmeurs d'écrire des greffons implémentant les protocoles réseaux appropriés. En théorie, Subversion peut utiliser un nombre infini d'implémentations réseau. En pratique, il n'existe à ce jour que deux serveurs.

Apache est un serveur web extrêmement populaire ; en utilisant le module **mod_dav_svn**, Apache a accès au dépôt et peut le rendre accessible aux clients via le protocole WebDAV/DeltaV, qui est une extension d'HTTP. Comme Apache est un serveur très complet, il inclut un grand nombre de fonctionnalités « gratuites », telles que : communications chiffrées par SSL, journalisation, intégration avec bon nombre de systèmes d'authentification tiers et navigation web limitée des dépôts.

À l'opposé vous trouvez **svnserve** : un petit programme serveur, léger, qui communique via un protocole sur mesure avec les clients. Parce que ce protocole a été conçu spécialement pour Subversion et possède la notion d'états (à la différence d'HTTP), il offre des opérations significativement plus rapides, certes aux dépens de certaines fonctionnalités. Bien qu'il sache utiliser SASL pour offrir toute une gamme d'options d'authentification et de chiffrement, il ne possède ni journalisation ni navigation web intégrée. Il est néanmoins très facile à mettre en place et constitue souvent le meilleur choix pour de petites équipes débutant avec Subversion.

Une troisième option consiste à utiliser **svnserve** encapsulé dans une connexion SSH. Même si ce scénario utilise toujours **svnserve**, il diffère quelque peu, en termes de fonctionnalités, d'un déploiement **svnserve** traditionnel. SSH sert à chiffrer toutes les communications. Par ailleurs, l'authentification utilise exclusivement SSH, ce qui nécessite des comptes utilisateurs au niveau système sur le serveur hôte (à la différence de « svnserve simple », qui possède ses propres comptes utilisateurs privés). Enfin, avec ce type de déploiement, un processus **svnserve** temporaire privé est créé pour chaque utilisateur qui se connecte, ce qui équivaut (du point de vue des permissions) à permettre à un groupe d'utilisateurs locaux d'accéder au dépôt via des URL `file://`. Les autorisations d'accès basées sur des chemins n'ont donc aucun sens, puisque chaque utilisateur accède directement aux fichiers de la base de données du dépôt.

Le Tableau 6.1, « Comparaison des fonctionnalités des serveurs Subversion » présente un résumé rapide des trois types courants de déploiements de serveurs.

Tableau 6.1. Comparaison des fonctionnalités des serveurs Subversion

Fonctionnalité	Apache + mod_dav_svn	svnserve	svnserve sur SSH
Authentification	Authentification HTTP(S) de base, certificats X.509, LDAP, NTLM, ou tout autre mécanisme compatible avec Apache.	CRAM-MD5 par défaut ; LDAP, NTLM, ou tout autre mécanisme compatible avec SASL.	SSH.
Comptes utilisateurs	Fichiers privés ou tout autre mécanisme compatible avec Apache (LDAP, SQL, etc.).	Fichiers privés ou tout autre mécanisme compatible avec SASL (LDAP, SQL, etc.).	Comptes utilisateurs systèmes.
Contrôle d'accès	L'accès en lecture/écriture peut être autorisé sur le dépôt tout entier ou restreint à des chemins spécifiés.	L'accès en lecture/écriture peut être autorisé sur le dépôt tout entier ou restreint à des chemins spécifiés.	L'accès en lecture/écriture ne peut être autorisé que sur le dépôt tout entier.

Fonctionnalité	Apache + mod_dav_svn	svnserve	svnserve sur SSH
Chiffrement	Disponible via SSL (option).	Disponible via les fonctionnalités optionnelles de SASL.	Inhérent à toute connexion SSH.
Journalisation	Journaux Apache complets de chaque requête HTTP et, en option, la journalisation « de haut niveau » des opérations côté client.	Pas de journaux.	Pas de journaux.
Interopérabilité	Accessible par tout client WebDAV.	Ne peut communiquer qu'avec des clients svn.	Ne peut communiquer qu'avec des clients svn.
Accès via une interface Web	Support intégré limité ou via des outils tiers tels que ViewVC.	Uniquement via des outils tierces tels que ViewVC.	Uniquement via des outils tierces tels que ViewVC.
Réplication serveur de type maître-esclave	Possibilité d'un mandataire transparent pour l'écriture (de l'esclave vers le maître).	Possibilité de créer seulement des serveurs esclaves en lecture seule.	Possibilité de créer seulement des serveurs esclaves en lecture seule.
Vitesse	Relativement plus lent.	Relativement plus rapide.	Relativement plus rapide.
Mise en place	Relativement complexe.	Extrêmement simple.	Moyennement simple.

Choix d'une configuration serveur

Et alors, quel serveur vous faut-il ? Quel est le meilleur ?

Bien évidemment, il n'y a pas de bonne réponse à cette question. Chaque équipe a des besoins propres et les différents serveurs représentent des compromis différents. Le projet Subversion lui-même ne recommande pas un serveur plus qu'un autre, ni ne considère qu'un serveur est plus « officiel » qu'un autre.

Voici quelques argumentaires qui vous aideront à choisir entre un déploiement et un autre, ainsi que quelques raisons de *ne pas choisir* l'un d'entre eux.

Serveur svnserve

Les bonnes raisons de l'utiliser :

- facile et rapide à mettre en place ;
- le protocole réseau possède la notion d'états et est significativement plus rapide que WebDAV ;
- pas besoin de créer des comptes utilisateurs système sur le serveur ;
- le mot de passe ne circule pas sur le réseau.

Pourquoi l'éviter :

- par défaut, seule une méthode d'authentification est disponible, le protocole réseau n'est pas chiffré et le serveur enregistre les mots de passe en clair (tous ces éléments peuvent être modifiés en configurant SASL, mais cela requiert un peu plus de travail) ;
- aucune journalisation, de quelque sorte que ce soit, n'est disponible, ni même celle des erreurs ;
- pas de navigation web intégrée (il vous faut installer un serveur web séparé et un logiciel de navigation du dépôt pour ajouter cette fonctionnalité).

svnserve sur SSH

Les bonnes raisons de l'utiliser :

- le protocole réseau possède la notion d'états et est significativement plus rapide que WebDAV ;
- vous pouvez tirer parti des comptes SSH existants et de l'infrastructure déjà mise en place pour les utilisateurs ;
- tout le trafic réseau est chiffré.

Pourquoi l'éviter :

- il n'y a qu'un seul choix possible de méthode d'authentification ;
- aucune journalisation, de quelque sorte que ce soit, n'est disponible, ni même celle des erreurs ;
- les utilisateurs doivent être membres du même groupe système ou utiliser une clé SSH partagée ;
- mal utilisé, il peut aboutir à des problèmes de droits sur les fichiers.

Serveur HTTP Apache

Les bonnes raisons de l'utiliser :

- il permet à Subversion d'utiliser n'importe lequel des nombreux systèmes d'authentification déjà intégrés dans Apache ;
- pas besoin de créer des comptes système sur le serveur ;
- journalisation Apache complète disponible ;
- le trafic réseau peut être chiffré par SSL ;
- HTTP(S) fonctionne généralement même derrière des pare-feu d'entreprise ;
- la possibilité de parcourir le dépôt avec un navigateur Web est disponible nativement ;
- le dépôt peut être monté en tant que lecteur réseau à des fins de gestion de versions transparente (voir la section intitulée « Gestion de versions automatique »).

Pourquoi l'éviter :

- significativement plus lent que **svnserve**, car HTTP est un protocole sans état et nécessite plus d'allers et retours réseau ;
- la mise en place initiale peut s'avérer complexe.

Recommandations

En général, les auteurs de ce livre recommandent une installation ordinaire de **svnserve** aux petites équipes qui débutent avec Subversion ; c'est le plus simple à installer et celui qui pose le moins de problèmes de maintenance. Vous pouvez toujours passer au déploiement d'un serveur plus complexe au fur et à mesure que vos besoins évoluent.

Voici quelques recommandations et astuces générales, basées sur des années de support aux utilisateurs :

- si vous essayez de mettre en place le serveur le plus simple possible pour votre groupe, une installation ordinaire de **svnserve** est la voie la plus sûre et la plus rapide. Notez cependant que les données de votre dépôt circuleront en clair sur le réseau. Si votre déploiement se situe entièrement à l'intérieur du LAN ou du VPN de votre compagnie, ce n'est pas un problème. Si le dépôt est accessible sur Internet, il serait bon de vous assurer que son contenu n'est pas sensible (c'est-à-dire qu'il ne contient que du code open source), ou alors de faire l'effort supplémentaire de configurer SASL pour chiffrer les communications réseau ;

- si vous devez vous insérer au sein de systèmes d'authentification déjà en place (LDAP, Active Directory, NTLM, X.509, etc.), il vous faudra utiliser soit le serveur basé sur Apache, soit **svnserve** configuré avec SASL. Si vous avez un besoin absolu de journaux des erreurs côté serveur ou de journaux des activités côté client, le serveur basé sur Apache est le seul choix possible ;

- que vous ayez décidé d'utiliser Apache ou un banal **svnserve**, créez un utilisateur `svn` unique sur votre système et utilisez-le pour faire tourner le processus serveur. Assurez-vous également que la totalité du répertoire du dépôt est la propriété de cet utilisateur. Du point de vue de la sécurité, les données du dépôt restent ainsi englobées et protégées par l'application des droits gérés par le système de fichiers du système d'exploitation et sont modifiables uniquement par le processus serveur Subversion lui-même ;

- si vous avez une infrastructure existante basée principalement sur des comptes SSH et si vos utilisateurs possèdent déjà des comptes système sur la machine qui sert de serveur, il est logique de déployer une solution **svnserve** sur SSH. Dans le cas contraire, nous ne recommandons généralement pas cette option au public. Il est considéré comme plus sûr de donner l'accès au dépôt à vos utilisateurs au moyen de comptes (imaginaires) gérés par **svnserve** ou Apache plutôt que par de véritables comptes système. Si vous voulez vraiment chiffrer les communications, nous vous recommandons d'utiliser Apache avec chiffrement SSL ou **svnserve** avec chiffrement SASL à la place ;

- ne vous laissez pas séduire par l'idée très simple de donner l'accès à un dépôt à tous vos utilisateurs directement via des URL `file://`. Même si le dépôt est accessible à tous sur un lecteur réseau, c'est une mauvaise idée. Elle ôte toutes les couches de protection entre les utilisateurs et le dépôt : les utilisateurs peuvent corrompre accidentellement (ou intentionnellement) la base de données du dépôt, il est difficile de mettre le dépôt hors ligne pour inspection ou pour mise à niveau et vous risquez d'aboutir à de graves problèmes de droits sur les fichiers (voir la section intitulée « Accès au dépôt par plusieurs méthodes »). Remarquez que c'est aussi une des raisons pour laquelle nous déconseillons d'accéder aux dépôts via des URL `svn+ssh://` : du point de vue de la sécurité, c'est en fait comme si les utilisateurs locaux y accédaient via `file://`, ce qui peut entraîner exactement les mêmes problèmes si l'administrateur ne fait pas preuve de prudence.

svnserve, un serveur sur mesure

Le programme **svnserve** est un serveur léger, capable de dialoguer avec des clients sur un réseau TCP/IP en utilisant un protocole dédié avec gestion des états. Les clients contactent le serveur **svnserve** en utilisant une URL qui commence par `svn://` ou `svn+ssh://`. Cette section explique différentes mises en œuvre de **svnserve**, l'authentification des clients sur le serveur et la configuration d'un contrôle d'accès approprié pour vos dépôts.

Démarrage du serveur

Il existe différentes façons de démarrer le programme **svnserve** :

- lancer **svnserve** en tant que serveur autonome, à l'écoute de requêtes ;
- utiliser le démon Unix **inetd** pour lancer une instance temporaire de **svnserve** quand une requête arrive sur un port déterminé ;
- utiliser SSH pour lancer une instance temporaire de **svnserve** dans un tunnel chiffré ;
- lancer **svnserve** en tant que service Microsoft Windows.

svnserve en serveur autonome

Le plus facile est de démarrer **svnserve** en tant que serveur autonome. Pour ce faire, utilisez l'option `-d` (d pour « daemon » qui est l'appellation consacrée pour les serveurs Unix) :

```
$ svnserve -d
$              # svnserve est maintenant actif, en écoute sur le port 3690
```

Lorsque vous lancez **svnserve** en serveur autonome, vous pouvez utiliser les options `--listen-port` et `--listen-host` pour spécifier le port et l'adresse voulus pour « écouter ».

Une fois le serveur démarré de cette manière, tous les dépôts présents sur votre machine seront accessibles par le réseau. Un client doit spécifier un *chemin absolu* dans l'URL du dépôt. Par exemple, si un dépôt est situé sous `/var/svn/projet-1`, un client l'atteindra par `svn://hote.exemple.com/var/svn/projet-1`. Pour renforcer la sécurité, vous pouvez passer l'option `-r` à **svnserve** afin de restreindre l'export aux dépôts situés sous le chemin indiqué. Par exemple :

```
$ svnserve -d -r /var/svn
…
```

L'utilisation de l'option `-r` modifie le chemin que le serveur considère comme la racine du système de fichiers à exporter. Les clients utiliseront alors des URL ne comportant pas cette portion du chemin (ce qui rend les URL plus courtes et plus discrètes) :

```
$ svn checkout svn://hote.exemple.com/projet-1
…
```

svnserve via inetd

Si vous désirez que **inetd** lance le processus, il vous faudra passer l'option `-i` (`--inetd`). Dans l'exemple suivant, nous montrons le résultat de la commande `svnserve -i`, mais notez bien que ce c'est pas de cette manière que l'on démarre le serveur ; reportez-vous aux paragraphes qui suivent l'exemple pour savoir comment configurer **inetd** pour qu'il démarre **svnserve**.

```
$ svnserve -i
( success ( 1 2 ( ANONYMOUS ) ( edit-pipeline ) ) )
```

Quand on l'invoque avec l'option `--inetd`, **svnserve** tente de communiquer avec un client Subversion via l'entrée et la sortie standards (`stdin` et `stdout`) en utilisant un protocole spécifique. C'est le comportement habituel de tout programme lancé par **inetd**. L'IANA a réservé le port 3690 pour le protocole Subversion ; sur un système Unix vous pouvez donc ajouter au fichier `/etc/services` les lignes suivantes (si elles n'existent pas déjà) :

```
svn             3690/tcp   # Subversion
svn             3690/udp   # Subversion
```

Si votre système utilise un serveur **inetd** classique de type Unix, vous pouvez ajouter la ligne suivante à `/etc/inetd.conf` :

```
svn stream tcp nowait proprio-svn /usr/bin/svnserve svnserve -i
```

Assurez-vous que l'utilisateur « proprio-svn » possède des droits d'accès appropriés pour vos dépôts. Dès lors, quand une connexion client arrive sur le port 3690, **inetd** va créer un processus **svnserve** pour lui répondre. Bien sûr, vous pouvez également ajouter l'option `-r` à cette ligne de configuration, pour préciser quels dépôts doivent être exportés.

svnserve encapsulé dans un tunnel

Le mode tunnel est une troisième façon de lancer **svnserve**, via l'option `-t`. Ce mode présuppose qu'un programme de connexion à distance tel que **rsh** ou **ssh** a permis à un utilisateur de s'authentifier avec succès et lance alors un processus privé **svnserve** *pour le compte de cet utilisateur* (remarquez qu'en tant qu'utilisateur vous aurez rarement, sinon jamais, l'occasion de lancer **svnserve** avec l'option `-t` en ligne de commande ; c'est le serveur SSH qui le fait à votre place). Le programme **svnserve** se comporte alors normalement (utilisation des entrées/sorties `stdin` et `stdout`) et suppose que le trafic est redirigé automatiquement vers le client par un tunnel. Quand **svnserve** est lancé par un gestionnaire de tunnel comme ici, soyez sans crainte : l'utilisateur authentifié possédera les droits de lecture et d'écriture sur les fichiers de la base de données. C'est essentiellement la même chose que quand un utilisateur local accède au dépôt via des URL `file://`.

Cette option est décrite en détail plus loin dans ce chapitre, dans la section intitulée « Encapsulation de svnserve dans un tunnel

SSH ».

svnserve en tant que service Windows

Si votre système Windows est un descendant de Windows NT (2000, 2003, XP ou Vista), vous pouvez lancer **svnserve** en tant que service Windows. C'est généralement une méthode bien plus plaisante que de le lancer en démon indépendant via l'option `(-d)`. Utiliser le mode démon nécessite de lancer une console, de taper une commande et ensuite de laisser la fenêtre de la console tourner indéfiniment. Un service Windows, au contraire, tourne à l'arrière-plan, peut être lancé automatiquement au démarrage et peut être démarré ou arrêté à l'aide de la même interface d'administration que les autres services Windows.

Vous devrez définir le nouveau service en utilisant l'outil en ligne de commande **SC.EXE**. De façon analogue à la ligne de configuration **inetd**, il vous faudra fournir la commande de lancement précise de **svnserve** pour que Windows le lance au démarrage :

```
C:\> sc create svn
        binpath= "C:\svn\bin\svnserve.exe --service -r C:\depot"
        displayname= "Serveur Subversion"
        depend= Tcpip
        start= auto
```

Ceci définit un nouveau service Windows nommé « svn », qui exécute une commande particulière **svnserve.exe** quand il démarre (dont la racine est, dans ce cas, `C:\depot`). Il y a toutefois un certain nombre de précautions à prendre avec cet exemple.

Premièrement, remarquez que le programme **svnserve.exe** doit toujours être lancé avec l'option `--service`. Toute autre option de **svnserve** doit ensuite être spécifiée sur la même ligne, mais vous ne pouvez pas ajouter d'options qui seraient en conflit avec celle-ci, telles que `--daemon (-d)`, `--tunnel`, ou `--inetd (-i)`. D'autres options, comme `-r` ou `--listen-port` ne posent pas de problème. Deuxièmement, faites attention aux espaces quand vous tapez la commande **SC.EXE** : les groupes `clé= valeur` ne doivent pas comporter d'espace dans `clé=` et doivent comporter exactement une espace avant `valeur`. Enfin, faites attention aux espaces présentes dans la ligne de commande que vous indiquez. Si le nom d'un répertoire contient des espaces (ou tout autre caractère qui ait besoin d'être banalisé), placez l'ensemble du contenu de `binpath` entre guillemets, qui doivent eux-mêmes être banalisés :

```
C:\> sc create svn
        binpath= "\"C:\program files\svn\bin\svnserve.exe\" --service -r C:\depot"
        displayname= "Serveur Subversion"
        depend= Tcpip
        start= auto
```

Notez aussi que le terme `binpath` prête à confusion : sa valeur est une *ligne de commande*, pas le chemin d'accès à un exécutable. C'est pourquoi vous devez l'entourer de guillemets s'il contient des espaces.

Une fois que le service a été crée, il peut être arrêté, démarré ou interrogé à l'aide des outils standards de l'interface graphique (le programme « Services » des outils d'administration) ou de la ligne de commande :

```
C:\> net stop svn
C:\> net start svn
```

Le service peut aussi être désinstallé (c'est-à-dire supprimé) en effaçant sa définition : **`sc delete svn`**. Prenez soin d'arrêter le service auparavant ! Le programme **SC.EXE** possède de nombreuses autres sous-commandes ; tapez **`sc /?`** en ligne de commande pour en savoir plus.

Authentification et contrôle d'accès intégrés

Lorsqu'un client se connecte au processus **svnserve**, les choses suivantes se passent :

- Le client sélectionne un répertoire particulier.

- Le serveur analyse le fichier `conf/svnserve.conf` de ce dépôt et commence à suivre les politiques d'authentification et de contrôle d'accès qui y sont décrites.
- Selon les politiques définies, une des choses suivantes a lieu :
  - Le client est autorisé à lancer des requêtes anonymes, sans jamais recevoir le moindre défi d'authentification.
  - Le client peut recevoir un défi d'authentification à tout instant.
  - Si l'on est en mode tunnel, le client déclare lui-même avoir déjà satisfait à une authentification externe (généralement par SSH).

Le serveur **svnserve** ne sait envoyer, par défaut, que des défis d'authentification CRAM-MD5[1]. Plus précisément, le serveur envoie une petite quantité de données aux clients. Le client utilise l'algorithme de hachage MD5 pour créer une empreinte combinant les données et le mot de passe, puis renvoie l'empreinte en guise de réponse. Le serveur effectue le même calcul avec le mot de passe enregistré pour vérifier que le résultat est identique. *Le mot de passe ne circule ainsi jamais en clair sur le réseau.*

Si votre serveur **svnserve** a été compilé en incluant le support de SASL, non seulement il sait comment envoyer des défis CRAM-MD5, mais il connaît aussi probablement un grand nombre d'autres mécanismes d'authentification. Consultez la section intitulée « Utilisation de svnserve avec SASL » plus loin dans ce chapitre pour savoir comment configurer l'authentification et le chiffrement avec SASL.

Le client peut bien sûr aussi être authentifié en externe par un gestionnaire de tunnel tel que **ssh**. Dans ce cas, le serveur se contente de prendre l'identifiant par lequel il a été lancé et de s'en servir comme utilisateur authentifié. Pour plus de détails, reportez-vous plus loin à la section intitulée « Encapsulation de svnserve dans un tunnel SSH ».

Vous avez sûrement déjà deviné que le fichier `svnserve.conf` est le mécanisme central qui contrôle les politiques d'authentification et de contrôle d'accès. Ce fichier a le même format que d'autres fichiers de configuration (voir la section intitulée « Zone de configuration des exécutables ») : les noms de paragraphes sont entourés de crochets (`[` et `]`), les lignes de commentaires commencent par des dièses (#) et chaque paragraphe contient des variables spécifiques qui peuvent être définies (`variable = valeur`). Examinons ces fichiers et apprenons à les utiliser.

Création d'un fichier utilisateurs et d'un domaine d'authentification

Pour ce qui suit, la section `[general]` de `svnserve.conf` contient toutes les variables dont vous avez besoin. Commencez par modifier les valeurs de toutes les variables : choisissez un nom pour le fichier qui contiendra vos noms d'utilisateur ainsi que vos mots de passe et choisissez un domaine d'authentification :

```
[general]
password-db = fichier-utilisateurs
realm = exemple de domaine
```

Le domaine (`realm` dans le fichier de configuration) est un nom que vous définissez. Il indique aux clients à quelle sorte d'« espace de noms » ils se connectent ; le client Subversion l'affiche dans l'invite d'authentification et l'utilise comme clé (en combinaison avec le nom de machine et le port du serveur) pour mettre en cache les éléments d'authentification sur le disque (voir la section intitulée « Mise en cache des éléments d'authentification du client »). La variable `password-db` pointe vers un fichier séparé qui contient une liste de noms d'utilisateurs et de mots de passe, utilisant le même format usuel. Par exemple :

```
[users]
harry = motdepassemachin
sally = motdepassebidule
```

La valeur de `password-db` peut correspondre à un chemin absolu ou à un chemin relatif pour le fichier des utilisateurs. Pour de nombreux administrateurs, conserver le fichier dans la zone `conf/`, aux côtés de `svnserve.conf`, est une solution simple et facile. D'un autre côté, il se pourrait que deux dépôts, voire plus, doivent partager le même fichier ; dans ce cas, le fichier devrait sans doute être situé dans un répertoire plus accessible. Les dépôts partageant le même fichier utilisateurs

[1] Voir la RFC 2195.

devraient aussi être configurés de sorte qu'ils soient dans le même domaine, puisque la liste des utilisateurs définit, par essence, un domaine d'authentification. Quel que soit l'emplacement du fichier, faites attention à positionner les droits en lecture/écriture de façon appropriée. Si vous savez sous quel nom d'utilisateur **svnserve** fonctionnera, restreignez l'accès au fichier utilisateurs en conséquence.

Mise en place du contrôle d'accès

Il y a deux variables supplémentaires à définir dans le fichier `svnserve.conf` : elles déterminent ce que les utilisateurs non-authentifiés (anonymes) et les utilisateurs authentifiés ont le droit de faire. Les variables `anon-access` et `auth-access` peuvent contenir les valeurs `none`, `read`, ou `write`. Choisir la valeur `none` empêche à la fois lecture et écriture ; `read` autorise l'accès en lecture seule au dépôt et `write` autorise l'accès complet en lecture/écriture au dépôt. Par exemple :

```
[general]
password-db = fichier-utilisateurs
realm = exemple de domaine

# les utilisateurs anonymes ne peuvent accéder au dépôt qu'en lecture
anon-access = read

# les utilisateurs authentifiés peuvent à la fois lire et écrire
auth-access = write
```

Les lignes présentes dans le fichier contiennent en fait les valeurs par défaut des variables, au cas où vous oublieriez de les définir. Si vous voulez être encore plus prudent, vous pouvez complètement interdire les accès anonymes :

```
[general]
password-db = fichier-utilisateurs
realm = exemple de domaine

# les utilisateurs anonymes ne sont pas autorisés
anon-access = none

# les utilisateurs authentifiés peuvent à la fois lire et écrire
auth-access = write
```

Le processus serveur sait interpréter non seulement ce contrôle d'accès générique, mais aussi des restrictions d'accès plus fines associées à des fichiers et répertoires spécifiques du dépôt. Pour utiliser cette fonctionnalité, vous devez définir un fichier contenant des règles plus détaillées, puis faire pointer la variable `authz-db` vers ce fichier :

```
[general]
password-db = fichier-utilisateurs
realm = exemple de domaine

# Règles d'accès propres à certains emplacements
authz-db = fichier-authz
```

Nous étudions la syntaxe du fichier authz plus loin dans ce chapitre, dans la section intitulée « Contrôle d'accès basé sur les chemins ». Notez que la variable `authz-db` n'est pas mutuellement exclusive avec les variables `anon-access` et `auth-access` ; si toutes les variables sont définies en même temps, *toutes* les règles doivent être satisfaites pour que l'accès soit autorisé.

Utilisation de svnserve avec SASL

L'authentification par CRAM-MD5 suffit aux besoins de bon nombre d'équipes. Cependant, si votre serveur (et vos clients Subversion) ont été compilés avec la bibliothèque « Cyrus Simple Authentication and Security Layer » (SASL), vous avez à votre disposition un certain nombre d'options d'authentification et de chiffrement.

Qu'est-ce que SASL ?

Cyrus Simple Authentication and Security Layer (ce qui signifie « Couche d'authentification et de sécurité simple ») est un logiciel libre qui a été écrit par l'université Carnegie Mellon. Il ajoute des possibilités d'authentification et de chiffrement à n'importe quel protocole réseau et, à partir de la version 1.5 de Subversion, le serveur **svnserve** ainsi que le client **svn** sont tous les deux capables de se servir de cette bibliothèque. Elle peut ne pas être à votre disposition : si vous compilez Subversion vous-même, il vous faut au minimum la version 2.1 de SASL d'installée sur votre système et vous devez vous assurer qu'elle est bien prise en compte durant le processus de compilation de Subversion. Si vous utilisez un paquet binaire précompilé de Subversion, vous devez vérifier auprès de celui qui maintient ce paquet si SASL a été inclus à la compilation. SASL est fourni avec de nombreux modules optionnels qui représentent différents systèmes d'authentification : Kerberos (GSSAPI), NTLM, One-Time-Passwords (OTP), DIGEST-MD5, LDAP, Secure-Remote-Password (SRP) et d'autres encore. Certains mécanismes seront disponibles, d'autres pas ; pensez à vérifier quels modules sont inclus.

Vous pouvez télécharger Cyrus SASL (à la fois le code source et la documentation) à l'adresse http://asg.web.cmu.edu/sasl/sasl-library.html.

Normalement, quand un client Subversion se connecte à **svnserve**, le serveur envoie un message de bienvenue qui liste les fonctionnalités qu'il supporte et le client répond avec une liste similaire. Si le serveur est configuré pour exiger une authentification, il envoie ensuite un défi listant les mécanismes d'authentification disponibles ; le client répond en choisissant un des mécanismes et l'authentification se déroule ensuite par quelques échanges de messages. Même quand SASL n'est pas présent dans la liste, le client et le serveur sont capables d'utiliser les mécanismes CRAM-MD5 et ANONYMOUS (voir la section intitulée « Authentification et contrôle d'accès intégrés »). Si le serveur et le client ont été compilés pour inclure SASL, un certain nombre d'autres mécanismes d'authentification sont éventuellement disponibles. Néanmoins, vous devez configurer explicitement SASL sur le serveur pour qu'ils soient proposés.

Authentification par SASL

Pour activer les mécanismes spécifiques à SASL sur le serveur, il faut faire deux choses. D'abord, créez un paragraphe `[sasl]` dans le fichier `svnserve.conf` de votre dépôt avec le couple clé/valeur initial :

```
[sasl]
use-sasl = true
```

Ensuite, créez le fichier principal de configuration de SASL, appelé `svn.conf`, à un endroit où la bibliothèque SASL saura le trouver — généralement dans un répertoire où sont situés les greffons SASL. Vous devez localiser le répertoire des greffons de votre système, tel que `/usr/lib/sasl2/` ou `/etc/sasl2/` (notez qu'*il ne s'agit pas* là du fichier `svnserve.conf` qui réside dans votre dépôt !).

Sur un serveur Windows vous devez aussi éditer la base de registre (à l'aide d'un outil tel que **regedit**) pour indiquer à SASL les emplacements où chercher. Créez une clé nommée `[HKEY_LOCAL_MACHINE\SOFTWARE\Carnegie Mellon\Project Cyrus\SASL Library]` et placez-y deux clés : l'une appelée `SearchPath` (dont la valeur est le chemin du répertoire contenant les bibliothèques de greffons SASL `sasl*.dll`) et l'autre appelée `ConfFile` (dont la valeur est le chemin du répertoire parent contenant le fichier `svn.conf` que vous avez créé).

Parce que SASL fournit de très nombreux mécanismes d'authentification, il serait insensé (et bien au-delà du cadre de ce livre) d'essayer de décrire toutes les configurations serveurs possibles. Nous vous recommandons plutôt de lire la documentation fournie dans le sous-répertoire `doc/` du code source de SASL. Elle décrit en détail chaque mécanisme, ainsi que la manière de configurer le serveur correctement pour chacun d'entre eux. Dans ce paragraphe, nous nous contentons de donner un exemple simple de configuration du mécanisme DIGEST-MD5. Par exemple, si votre fichier `subversion.conf` (ou `svn.conf`) contient ce qui suit :

```
pwcheck_method: auxprop
auxprop_plugin: sasldb
sasldb_path: /etc/ma_bdd_sasl
mech_list: DIGEST-MD5
```

vous demandez à SASL de proposer le mécanisme DIGEST-MD5 aux clients et de comparer les mots de passe des utilisateurs à une base de mots de passe privée située à l'emplacement `/etc/ma_bdd_sasl`. Un administrateur système pourra ensuite

utiliser le programme **saslpasswd2** pour ajouter ou modifier les noms d'utilisateurs et les mots de passe contenus dans cette base de données :

```
$ saslpasswd2 -c -f /etc/ma_bdd_sasl -u domaine utilisateur
```

Quelques consignes de prudence : tout d'abord, l'argument « domaine » qui est passé à **saslpasswd2** doit correspondre au même domaine que celui que vous avez défini dans le fichier `svnserve.conf` ; s'ils ne correspondent pas, l'authentification échouera. En outre, à cause d'une limitation de SASL, ce domaine commun doit être une chaîne sans espace. Enfin, si vous décidez d'utiliser la base de données standard de mots de passe SASL, assurez-vous que le programme **svnserve** a accès en lecture à ce fichier (et éventuellement aussi en écriture, si vous utilisez un mécanisme tel que OTP).

Ceci est une manière simple de configurer SASL. De nombreux autres mécanismes d'authentification sont disponibles et les mots de passe peuvent être conservés dans des conteneurs différents, par exemple des annuaires LDAP ou des bases de données SQL. Reportez-vous à la documentation complète de SASL pour plus de détails.

Souvenez-vous que si vous configurez votre serveur pour qu'il n'autorise que certains mécanismes d'authentification SASL, tous les clients qui se connectent ont l'obligation de supporter SASL. Tout client Subversion compilé sans SASL (ce qui inclut tous les clients antérieurs à la version 1.5) est incapable de se connecter. D'un autre côté, ce type de restriction est peut-être exactement ce que vous recherchez (« Mes clients doivent tous utiliser Kerberos ! »). Néanmoins, si vous voulez permettre à des clients non-SASL de se connecter, pensez bien à inclure le mécanisme CRAM-MD5 dans les choix possibles. Tous les clients savent utiliser CRAM-MD5, qu'ils aient des fonctionnalités SASL ou pas.

Chiffrement SASL

SASL est également capable d'effectuer le chiffrement des données si un mécanisme particulier le supporte. Le mécanisme intégré CRAM-MD5 ne supporte pas le chiffrement, mais DIGEST-MD5 le supporte et d'autres mécanismes tels que SRP requièrent l'utilisation de la bibliothèque OpenSSL. Pour activer ou désactiver différents niveaux de chiffrement, vous pouvez définir deux variables dans le fichier `svnserve.conf` de votre dépôt :

```
[sasl]
use-sasl = true
min-encryption = 128
max-encryption = 256
```

Les variables `min-encryption` et `max-encryption` contrôlent le niveau de chiffrement exigé par le serveur. Pour désactiver complètement le chiffrement, mettez les deux valeurs à 0. Pour activer une simple somme de contrôle sur les données (par exemple pour empêcher toute manipulation douteuse et garantir l'intégrité des données sans chiffrement), mettez les deux valeurs à 1. Si vous voulez autoriser le chiffrement, sans que ce soit obligatoire, mettez 0 pour la valeur minimale et un nombre de bits donné pour la valeur maximale. Pour exiger un chiffrement inconditionnel, mettez les deux valeurs à un nombre plus grand que 1. Dans l'exemple précédent, nous obligeons les clients à utiliser au moins un chiffrement 128 bits et au plus un chiffrement 256 bits.

Encapsulation de svnserve dans un tunnel SSH

L'authentification intégrée (ainsi que SASL) peuvent être très pratiques, car ils évitent d'avoir à créer de véritables comptes systèmes. D'un autre côté, certains administrateurs ont déjà des systèmes d'authentification bien établis en place. Dans ce cas, tous les utilisateurs du projet possèdent déjà des comptes systèmes et peuvent se connecter au serveur par SSH.

Utiliser SSH en conjonction avec **svnserve** est facile. Le client utilise juste une URL `svn+ssh://` pour se connecter :

```
$ whoami
harry

$ svn list svn+ssh://hote.exemple.com/depot/projet
harryssh@hote.exemple.com's password:  *****

truc
machin
bidule
…
```

Dans cet exemple, le client Subversion lance un processus local **ssh**, se connecte à `hote.exemple.com`, s'authentifie en tant que `harryssh` (en accord avec la configuration SSH des utilisateurs) puis un processus **svnserve** privé est généré automatiquement sur la machine distante, processus dont le propriétaire est l'utilisateur `harryssh`. La commande **svnserve** est lancée en mode tunnel (`-t`) et son protocole réseau est encapsulé dans la connexion chiffrée par **ssh**, le gestionnaire de tunnel. Si le client effectue une propagation, l'auteur de la nouvelle révision sera l'utilisateur authentifié (`harryssh`).

Ce qu'il est important de comprendre ici est que le client Subversion *ne se connecte pas* à un serveur **svnserve** fonctionnant en permanence. Cette méthode d'accès ne requiert pas la présence d'un démon, ni ne vérifie s'il y en a un qui tourne. Elle se base entièrement sur la capacité de **ssh** à générer un processus **svnserve** temporaire, qui ensuite se termine une fois la connexion SSH close.

Quand vous utilisez des URL `svn+ssh://` pour accéder à un dépôt, souvenez-vous que c'est le programme **ssh** qui envoie l'invite d'authentification, *pas le client* **svn**. Ce qui signifie qu'il n'y a pas de mise en cache automatique de mots de passe (voir la section intitulée « Mise en cache des éléments d'authentification du client »). Le fonctionnement du client Subversion fait qu'il accède souvent au dépôt par des connexions multiples, bien que les utilisateurs ne s'en rendent habituellement pas compte grâce à la fonctionnalité de mise en cache du mot de passe. Lorsque vous utilisez des URL `svn+ssh://`, les utilisateurs risquent de trouver ça pénible que **ssh** demande le mot de passe de façon répétitive pour toute connexion vers l'extérieur. La solution est d'utiliser un outil séparé de mise en cache du mot de passe, tel que **ssh-agent** sur Unix ou **pageant** sur Windows.

Quand le trafic passe par un tunnel, les accès sont contrôlés principalement par les droits sur les fichiers de la base de données liés au système d'exploitation ; c'est quasiment pareil que si Harry accédait au dépôt directement via une URL `file://`. Si plusieurs utilisateurs systèmes accèdent au dépôt directement, il est de bon ton de les placer dans un même groupe et vous devez faire attention aux umasks (prenez soin de lire la section intitulée « Accès au dépôt par plusieurs méthodes » plus loin dans ce chapitre). Mais même dans le cas de l'encapsulation, vous pouvez toujours utiliser le fichier `svnserve.conf` pour bloquer l'accès, en spécifiant juste `auth-access = read` ou `auth-access = none`[2].

Vous vous attendez à ce que cette histoire d'encapsulation SSH se termine ici, mais ce n'est pas le cas. Subversion vous permet de créer des comportements d'encapsulation personnalisés dans votre fichier de configuration (voir la section intitulée « Zone de configuration des exécutables »). Par exemple, supposons que vous vouliez utiliser RSH au lieu de SSH[3]. Dans le paragraphe `[tunnels]` de votre fichier `config`, définissez-le comme ceci :

```
[tunnels]
rsh = rsh
```

À présent vous pouvez utiliser cette nouvelle définition d'encapsulation par le biais d'un schéma d'URL qui correspond au nom de votre nouvelle variable : `svn+rsh://hote/chemin`. Lorsqu'il utilise le nouveau type d'URL, le client Subversion lance en fait en arrière-plan la commande **`rsh hote svnserve -t`**. Si vous incluez un nom d'utilisateur dans l'URL (par exemple `svn+rsh://nomdutilisateur@hote/chemin`), le client va l'inclure dans sa commande (**`rsh nomdutilisateur@hote svnserve -t`**). Mais vous pouvez définir des schémas d'encapsulation bien plus évolués :

```
[tunnels]
joessh = $JOESSH /opt/alternate/ssh -p 29934
```

Cet exemple illustre plusieurs choses. D'abord, il indique comment faire pour que le client Subversion lance un exécutable d'encapsulation particulier (celui situé à l'emplacement `/opt/alternate/ssh`) avec des options particulières. Dans ce cas, se connecter à une URL `svn+joessh://` lance un exécutable SSH particulier avec les arguments `-p 29934` (utile si vous voulez que le programme d'encapsulation se connecte à un port non standard).

Ensuite, il indique comment définir une variable d'environnement personnalisée capable de remplacer le nom du programme d'encapsulation. Configurer la variable d'environnement `SVN_SSH` est un moyen simple de modifier le programme d'encapsulation par défaut. Mais s'il vous faut différents programmes d'encapsulation pour différents serveurs, chacun se connectant par exemple à un port différent ou passant des options différentes à SSH, vous pouvez utiliser le mécanisme illustré dans cet exemple. Concrètement, si nous donnons une valeur à la variable d'environnement `JOESSH`, cette valeur remplacera la totalité de la valeur de la variable d'encapsulation — **$JOESSH** serait exécuté au lieu de **`/opt/alternate/ssh -p 29934`**.

[2] Notez qu'utiliser le moindre contrôle d'accès avec **svnserve** ne sert strictement à rien ; l'utilisateur ayant déjà directement accès à la base de données.
[3] Nous ne le recommandons pas, car RSH est significativement moins sécurisé que SSH.

Astuces de configuration de SSH

Il est possible de contrôler non seulement la manière dont le client lance **ssh**, mais aussi le comportement de **sshd** sur votre machine. Dans ce paragraphe, nous indiquons comment contrôler la commande **svnserve** précise qui est exécutée par **sshd**, ainsi que comment faire pour que plusieurs utilisateurs partagent un même compte système.

Mise en œuvre initiale

Pour commencer, localisez le répertoire « home » du compte utilisateur que vous utilisez pour lancer **svnserve**. Assurez-vous que ce compte possède un bi-clé de clés publique/privée et que l'utilisateur peut se connecter en s'authentifiant par la méthode à clé publique. L'authentification par mot de passe ne fonctionnera pas, puisque toutes les astuces SSH qui suivent consistent à utiliser le fichier `authorized_keys`.

S'il n'existe pas déjà, créez le fichier `authorized_keys` (sur Unix, c'est généralement `~/.ssh/authorized_keys`). Chaque ligne de ce fichier décrit une clé publique autorisée à se connecter. Ces ligne sont généralement de la forme :

```
ssh-dsa AAAABtce9euch… utilisateur@exemple.com
```

Le premier champ décrit le type de clé, le second champ est la clé elle-même, encodée en base 64, et le troisième champ est un commentaire. Cependant, c'est un fait moins connu que la ligne toute entière peut être précédée par un champ `command` :

```
command="programme" ssh-dsa AAAABtce9euch… utilisateur@exemple.com
```

Quand le champ `command` est présent, le serveur SSH va lancer le programme indiqué en lieu et place de l'habituel **svnserve** en mode tunnel que le client Subversion a demandé. En découlent un certain nombre d'astuces côté serveur. Dans les exemples suivants, nous abrégeons les lignes du fichier par :

```
command="programme" TYPE CLÉ COMMENTAIRE
```

Contrôle de la commande à exécuter

Comme nous pouvons spécifier la commande exécutée côté serveur, il devient facile de désigner un exécutable **svnserve** spécifique et d'y passer des arguments supplémentaires :

```
command="/chemin/vers/svnserve -t -r /racine/virtuelle" TYPE CLÉ COMMENTAIRE
```

Dans cet exemple, `/chemin/vers/svnserve` peut être un script personnalisé construit autour de **svnserve** qui spécifie le umask à utiliser (voir la section intitulée « Accès au dépôt par plusieurs méthodes »). Il indique aussi comment « ancrer » **svnserve** dans un répertoire racine virtuel, ce qui est aussi très souvent utilisé quand **svnserve** fonctionne en tant que démon. Le but est soit de restreindre l'accès à certaines parties du système, soit simplement d'éviter que l'utilisateur ait à taper un chemin absolu dans l'URL `svn+ssh://`.

Il est également possible d'avoir plusieurs utilisateurs partageant un compte utilisateur unique. Au lieu de créer un compte système distinct pour chaque utilisateur, générez plutôt un bi-clé de clés publique/privée pour chaque personne. Placez ensuite chaque clé publique dans le fichier `authorized_users`, une par ligne, et utilisez l'option `--tunnel-user` :

```
command="svnserve -t --tunnel-user=harry" TYPE1 CLÉ1 harry@exemple.com
command="svnserve -t --tunnel-user=sally" TYPE2 CLÉ2 sally@exemple.com
```

Cet exemple permet à la fois à Harry et à Sally de se connecter au même compte utilisateur avec l'authentification via leur clé publique. Chacun d'eux doit exécuter une commande qui lui est propre ; l'option `--tunnel-user` signale à **svnserve** que l'argument fourni doit être considéré comme le nom de l'utilisateur authentifié. Sans `--tunnel-user`, toutes les propagations sembleraient venir du même compte utilisateur partagé.

Finalement, un dernier avertissement : autoriser l'accès d'un utilisateur au serveur via une clé publique dans un compte partagé n'empêche pas forcément d'autres formes d'accès SSH, même si vous avez spécifié une valeur pour le champ `command` dans le fichier `authorized_keys`. Par exemple, l'utilisateur aura toujours accès au shell via SSH, il sera capable d'effectuer de la redirection X11 ou de la redirection d'un port quelconque de votre serveur. Pour accorder le moins de droits possibles à l'utilisateur, vous pouvez spécifier des options de restriction immédiatement après la commande du champ `command` :

```
  command="svnserve -t --tunnel-user=harry",no-port-forwarding,no-agent-for
warding,no-X11-forwarding,no-pty TYPE1 CLÉ1 harry@exemple.com
```

Notez bien que tout ceci doit tenir sur une seule ligne, vraiment sur une seule ligne, car les fichiers SSH `authorized_keys` n'autorisent même pas le caractère habituel de continuation de ligne (\). L'unique raison pour laquelle nous avons rajouté une coupure est pour que cela tienne dans le format physique d'un livre.

httpd, le serveur HTTP Apache

Le serveur HTTP Apache est un serveur réseau à tout faire que Subversion sait exploiter. Via un module adapté, **httpd** rend les dépôts Subversion accessibles aux clients par le protocole WebDAV/DeltaV, qui est une extension de HTTP 1.1 (voir http://www.webdav.org/ pour plus d'informations). Ce protocole se base sur HTTP, le protocole omniprésent à la base du World Wide Web, lui ajoute des fonctionnalités d'écriture et, en particulier, d'écriture avec gestion de versions. Le résultat est un système robuste et standardisé qui est inclus dans le logiciel Apache 2.0, supporté par de nombreux systèmes d'exploitation et outils tiers, et qui ne demande pas aux administrateurs réseaux d'ouvrir un port réseau supplémentaire[4]. Bien qu'un serveur Apache-Subversion ait plus de fonctionnalités que **svnserve**, il est aussi plus difficile à mettre en place. La flexibilité a bien souvent pour contrepartie la complexité.

Une grande partie de ce qui va suivre fait référence à des directives de configuration d'Apache. Bien que l'utilisation de ces directives soit illustrée par quelques exemples, les décrire complètement va bien au-delà du sujet de ce chapitre. L'équipe Apache tient à jour une documentation excellente, disponible publiquement sur leur site web à l'adresse http://httpd.apache.org. Par exemple, le guide de référence complet des directives de configuration est situé à l'adresse http://httpd.apache.org/docs-2.0/mod/directives.html.

En outre, au fur et à mesure des changements que vous apporterez à votre configuration d'Apache, il est probable que vous commettrez des erreurs. Si vous n'êtes pas déjà familier avec le sous-système de journalisation d'Apache, vous devrez apprendre à le connaître. Dans votre fichier `httpd.conf`, des directives spécifient l'emplacement sur le disque des journaux d'accès et d'erreurs générés par Apache (les directives `CustomLog` et `ErrorLog` respectivement). Le module **mod_dav_svn** de Subversion utilise également l'interface de journalisation des erreurs d'Apache. Pensez à naviguer dans ces fichiers lorsque vous recherchez des informations susceptibles de vous aider à trouver l'origine d'un problème.

À propos d'Apache 2

Si vous êtes un administrateur système, il est très probable que vous utilisiez déjà le serveur web Apache et que vous en ayez une expérience préalable. À l'heure où ces lignes sont écrites, Apache 1.3 est la version la plus populaire d'Apache. Le passage de la communauté à Apache 2 est assez lent, pour diverses raisons : certains ont peur du changement, en particulier quand il s'agit de toucher à quelque chose d'aussi essentiel qu'un serveur web. D'autres dépendent de greffons qui ne fonctionnent qu'avec l'interface (l'API) de Apache 1.3 et attendent qu'ils soient portés vers Apache 2. Quelle qu'en soit la raison, beaucoup de gens commencent à s'inquiéter quand ils découvrent que le module Apache de Subversion a été écrit spécifiquement pour l'API d'Apache 2.

Mais ne vous inquiétez pas ! Il est facile de faire fonctionner Apache 1.3 et Apache 2 côte à côte ; il suffit de les installer dans des endroits séparés et d'utiliser Apache 2 en tant que serveur dédié fonctionnant sur un port autre que le port 80. Les clients peuvent dès lors accéder au dépôt en indiquant le numéro de port dans l'URL :

```
$ svn checkout http://hote.exemple.com:7382/depot/projet
```

Prérequis

[4] C'est une chose qu'ils détestent faire.

Pour mettre à disposition votre dépôt sur le réseau par HTTP, il vous faut quatre composants, disponibles dans deux paquets. Il vous faut Apache **httpd** 2.0, le module DAV **mod_dav** fourni avec, Subversion et le module **mod_dav_svn** implémentant le système de fichiers, qui est fourni avec Subversion. Une fois que vous avez tous ces composants, la procédure de mise en réseau de votre dépôt est aussi simple que :

- faire fonctionner httpd 2.0 avec le module **mod_dav** ;
- installer le module **mod_dav_svn** derrière **mod_dav** (**mod_dav_svn** utilise les bibliothèques Subversion pour accéder au dépôt) ;
- configurer le fichier `httpd.conf` pour exporter (ou « exposer ») le dépôt.

Vous pouvez accomplir les deux premières tâches ci-dessus soit en compilant **httpd** et Subversion à partir du code source, soit en installant les paquets binaires précompilés correspondants sur votre système. Les informations les plus récentes sur la façon de compiler Subversion dans le cadre d'une utilisation en conjonction avec le serveur HTTP Apache, sur la compilation et sur la configuration d'Apache lui-même dans cet objectif, sont consultables dans le fichier `INSTALL` situé à la racine de l'arborescence du code source de Subversion.

Configuration Apache de base

Une fois que les composants requis sont installés sur votre système, il ne reste plus qu'à configurer Apache au moyen de son fichier `httpd.conf`. Indiquez à Apache de charger le module **mod_dav_svn** grâce à la directive `LoadModule`. Cette directive doit précéder tout autre élément de configuration lié à Subversion. Si votre serveur Apache a été installé avec la configuration par défaut, votre module **mod_dav_svn** devrait avoir été installé dans le sous-répertoire `modules` du répertoire d'installation d'Apache (souvent `/usr/local/apache2`). La directive `LoadModule` a une syntaxe très simple, faisant correspondre un nom de module à l'emplacement sur le disque d'une bibliothèque partagée :

```
LoadModule dav_svn_module     modules/mod_dav_svn.so
```

Notez que si **mod_dav** a aussi été compilé sous forme de bibliothèque partagée (et non par une édition de liens statiques qui le place alors directement dans l'exécutable **httpd**), il vous faudra une directive `LoadModule` pour celui-ci. Assurez-vous qu'elle est placée avant la ligne **mod_dav_svn** :

```
LoadModule dav_module         modules/mod_dav.so
LoadModule dav_svn_module     modules/mod_dav_svn.so
```

Plus loin dans votre fichier de configuration, vous devez indiquer à Apache l'endroit où réside votre dépôt (ou vos dépôts) Subversion. La directive `Location` possède une syntaxe de type XML, qui commence par une balise de début, se termine par une balise de fin et contient diverses autres directives de configuration au milieu. Le sens de la directive `Location` est de faire faire à Apache quelque chose de spécial quand il traite les requêtes adressées à une URL donnée ou à une de ses filles. Dans le cas de Subversion, il faut qu'Apache fasse traiter par la couche DAV les URL pointant vers les ressources suivies en versions. Vous pouvez indiquer à Apache de déléguer le traitement de toutes les URL dont la partie chemin d'accès (la partie de l'URL qui suit le nom du serveur et le numéro de port optionnel) commence par `/depot/` à un gestionnaire de DAV dont le dépôt est situé à l'adresse `/var/svn/mon-depot` en utilisant la syntaxe `httpd.conf` suivante :

```
<Location /depot>
  DAV svn
  SVNPath /var/svn/mon-depot
</Location>
```

Si vous avez l'intention de gérer plusieurs dépôts Subversion résidant dans le même répertoire parent sur votre disque local, vous pouvez utiliser une directive alternative, `SVNParentPath`, pour faire état de ce répertoire parent commun. Par exemple, si vous savez que vous allez créer plusieurs dépôts Subversion dans le répertoire `/var/svn`, auxquels on accédera par des URL telles que `http://mon.serveur.com/svn/depot1`, `http://mon.serveur.com/svn/depot2`, etc., vous pouvez utiliser la syntaxe de configuration de `httpd.conf` de l'exemple suivant :

```
<Location /svn>
  DAV svn

  # à toute URL "/svn/truc" correspond un dépôt /var/svn/truc
  SVNParentPath /var/svn
</Location>
```

Par cette syntaxe, Apache va déléguer le traitement de toutes les URL dont le chemin d'accès commence par `/svn/` au gestionnaire DAV de Subversion, qui ensuite supposera que tous les éléments du répertoire spécifié dans la directive `SVNParentPath` sont en fait des dépôts Subversion. C'est une syntaxe particulièrement pratique dans le sens où, à la différence de celles utilisant la directive `SVNPath`, vous n'avez pas besoin de redémarrer Apache pour créer et mettre à disposition sur le réseau de nouveaux dépôts.

Vérifiez bien que, quand vous définissez votre nouvelle directive `Location`, elle n'interfère pas avec d'autres emplacements exportés. Par exemple, si votre `DocumentRoot` (c'est-à-dire le répertoire racine des fichiers qu'Apache expose sur le web) principal est exporté vers `/www`, n'exportez pas de dépôt Subversion vers `<Location /www/depot>`. Si une requête arrivait pour l'URI `/www/depot/machin.c`, Apache ne saurait pas s'il doit chercher un fichier `depot/machin.c` dans son `DocumentRoot` ou s'il doit déléguer à **mod_dav_svn** la tâche de renvoyer `machin.c` qui se trouve dans le dépôt Subversion. Ceci aboutit souvent à une erreur du serveur de la forme `301 Moved Permanently`.

Noms de serveur et requêtes COPY

Subversion se sert du type de requête `COPY` pour effectuer des copies de fichiers et de répertoires côté serveur. Les modules Apache vérifient que la source de la copie est bien située sur la même machine que la destination. Pour satisfaire cette exigence, vous aurez peut-être besoin de déclarer à **mod_dav** le nom d'hôte de votre serveur. En général, la directive `ServerName` du fichier `httpd.conf` peut être utilisée à cette fin.

```
ServerName svn.exemple.com
```

Si vous faites usage du support des hôtes virtuels d'Apache via la directive `NameVirtualHost`, la directive `ServerAlias` vous permettra de spécifier des noms additionnels désignant votre serveur. Encore une fois, reportez-vous à la documentation Apache pour plus de détails.

À présent, il nous faut examiner sérieusement la question des droits d'accès. Si vous utilisez Apache depuis un certain temps en tant que serveur web principal, vous hébergez certainement pas mal de contenu : des pages web, des scripts, etc. Ces éléments ont déjà été configurés avec un ensemble de droits qui leur permet de fonctionner avec Apache, ou plus exactement qui permet à Apache de travailler avec ces fichiers. Apache, quand on l'utilise en tant que serveur Subversion, a aussi besoin de droits d'accès corrects pour lire et écrire dans votre dépôt.

Vous devez mettre en place les droits d'accès qui satisfont les besoins de Subversion sans mettre à mal l'installation des pages web et scripts pré-existants. Ceci implique peut-être de modifier les droits d'accès de votre dépôt Subversion pour qu'ils correspondent à ceux utilisés par les autres éléments qu'Apache gère, ou bien utiliser les directives `User` et `Group` du fichier `httpd.conf` pour spécifier qu'Apache doit fonctionner avec l'identifiant et le groupe qui est propriétaire de votre dépôt Subversion. Il n'y a pas une façon unique de mettre en place les droits d'accès et chaque administrateur a ses propres raisons pour faire les choses d'une manière ou d'une autre. Soyez juste conscient que les problèmes liés aux droits d'accès sont peut-être le point le plus négligé lors de la configuration d'un dépôt avec Apache.

Options d'authentification

À ce stade, si vous avez configuré `httpd.conf` avec quelque chose du style :

```
<Location /svn>
  DAV svn
  SVNParentPath /var/svn
</Location>
```

votre dépôt est à présent accessible « anonymement » au reste du monde. Jusqu'à ce que configuriez des politiques d'authentification et de contrôle d'accès, les dépôts Subversion que vous rendez disponibles via la directive `Location` sont généralement accessibles à tous. En d'autres termes :

- N'importe qui peut utiliser un client Subversion pour extraire une copie de travail d'une URL du dépôt (ou de n'importe lequel de ses sous-répertoires).
- N'importe qui peut naviguer interactivement dans la dernière révision du dépôt rien qu'en allant avec un navigateur web à l'URL du dépôt.
- N'importe qui peut effectuer des propagations vers le dépôt.

Bien sûr, vous avez peut-être déjà mis en place une procédure automatique `pre-commit` pour empêcher les propagations (voir la section intitulée « Mise en place des procédures automatiques »). Mais en progressant dans la lecture de ce chapitre, vous verrez qu'il est également possible d'utiliser les méthodes intégrées dans Apache pour restreindre les accès de façon spécifique.

Mise en place de l'authentification HTTP

La manière la plus facile d'authentifier un client est le mécanisme d'authentification « Basic » de HTTP, qui utilise juste un nom d'utilisateur et un mot de passe pour vérifier que l'utilisateur est bien celui qu'il prétend être. Apache fournit un utilitaire **htpasswd** pour gérer la liste des noms d'utilisateurs et mots de passe acceptés. Accordons le droit de propager à Sally et à Harry. D'abord, il faut les ajouter au fichier des mots de passe :

```
$ ### Première fois : utilisez -c pour créer le fichier
$ ### Ajoutez -m pour MD5 afin de chiffrer le mot de passe et ainsi le rendre plus
sûr
$ htpasswd -cm /etc/fichier-auth-svn harry
New password: *****
Re-type new password: *****
Adding password for user harry
$ htpasswd -m /etc/fichier-auth-svn sally
New password: *******
Re-type new password: *******
Adding password for user sally
$
```

Ensuite, vous devez ajouter des directives dans le bloc `Location` du fichier `httpd.conf` pour indiquer à Apache comment se servir du nouveau fichier des mots de passe. La directive `AuthType` spécifie le type d'authentification que l'on veut utiliser. Dans notre cas, nous voulons utiliser le mode « Basic ». La directive `AuthName` permet de donner un nom arbitraire au domaine d'authentification. La plupart des navigateurs affichent ce nom dans la boîte de dialogue (pop-up) qui demande le nom d'utilisateur et le mot de passe. Enfin, utilisez la directive `AuthUserFile` pour spécifier le chemin du fichier des mots de passe que vous venez de créer avec la commande **htpasswd**.

Après avoir ajouté ces trois directives, votre bloc `<Location>` devrait ressembler à ceci :

```
<Location /svn>
  DAV svn
  SVNParentPath /var/svn
  AuthType Basic
  AuthName "Depot Subversion"
  AuthUserFile /etc/fichier-auth-svn
</Location>
```

Ce bloc `<Location>` n'est pas encore complet et il ne sert pas à grand chose pour l'instant. Il indique juste à Apache que, lorsqu'un contrôle d'accès est requis, Apache doit demander un nom d'utilisateur et un mot de passe au client Subversion. Ce qui manque ici, cependant, ce sont les directives qui indiquent à Apache *quelles sortes de requêtes client* requièrent un contrôle d'accès. Dès que le contrôle d'accès est demandé, Apache exige également l'authentification. La chose la plus simple à faire est de protéger toutes les requêtes. Ajouter `Require valid-user` signale à Apache que, pour toutes les requêtes, l'utilisateur

doit être authentifié :

```
<Location /svn>
  DAV svn
  SVNParentPath /var/svn
  AuthType Basic
  AuthName "Depot Subversion"
  AuthUserFile /etc/fichier-auth-svn
  Require valid-user
```

Prenez bien soin de lire la section intitulée « Options de contrôle d'accès » qui suit pour plus de détails sur la directive `Require` et sur les autres manières de mettre en œuvre des politiques de contrôle d'accès.

Un petit avertissement : les mots de passe de la méthode HTTP Basic Auth circulent pratiquement en clair sur le réseau et ne sont donc pas du tout sécurisés.

Une autre possibilité est de ne pas utiliser la méthode d'authentification `Basic` mais d'utiliser la méthode d'authentification `Digest` à la place. L'authentification `Digest` permet au server de vérifier l'identité du client *sans envoyer le mot de passe en clair* sur le réseau. En supposant que le client et le serveur connaissent tous les deux le mot de passe de l'utilisateur, ils peuvent vérifier que le mot de passe est le même en l'utilisant pour appliquer une fonction de hachage à une donnée créée pour l'occasion. Le serveur envoie au client une chaîne plus ou moins aléatoire de petite taille ; le client se sert du mot de passe de l'utilisateur pour créer un condensat (*hash* en anglais) de cette chaîne ; le serveur examine ensuite si le condensat correspond à ses attentes.

Configurer Apache pour la méthode d'authentification `Digest` est également assez facile et ne comporte qu'une petite variation par rapport à notre exemple précédent. Prenez soin de consulter la documentation complète d'Apache pour plus de détails.

```
<Location /svn>
  DAV svn
  SVNParentPath /var/svn
  AuthType Digest
  AuthName "Depot Subversion"
  AuthDigestDomain /svn/
  AuthUserFile /etc/fichier-auth-svn
  Require valid-user
</Location>
```

Si vous recherchez la sécurité maximale, la cryptographie à clé publique est la meilleure solution. Il est sans doute mieux d'utiliser du chiffrement SSL afin que les clients s'authentifient via `https://` au lieu de `http://` ; le minimum minimorum consiste alors à configurer Apache pour qu'il utilise un certificat serveur auto-signé[5]. Consultez la documentation Apache (et la documentation OpenSSL) pour savoir comment faire.

Gestion des certificats SSL

Les entreprises qui ont besoin de rendre leur dépôts disponibles au-delà du pare-feu périmétrique de leur société doivent être conscientes que des entités non-autorisées auront la possibilité d'« écouter » leur trafic réseau. SSL permet de diminuer le risque que cette écoute conduise à des fuites de données sensibles.

Si un client Subversion est compilé pour utiliser OpenSSL, il obtient la possibilité de communiquer avec le serveur Apache via des URL `https://`. La bibliothèque Neon utilisée par le client Subversion est non seulement capable de vérifier les certificats du serveur, mais aussi de renvoyer des certificats clients quand on le lui demande. Une fois que le client et le serveur ont échangé des certificats SSL et se sont authentifiés mutuellement avec succès, tous les échanges qui s'ensuivent sont chiffrés par une clé de session.

Expliquer comment générer des certificats clients et serveurs ou comment configurer Apache pour les utiliser s'éloigne trop du sujet de ce livre. De nombreux autres livres, dont la documentation Apache elle-même, expliquent comment le faire. Mais nous *pouvons quand même* traiter ici la question de la gestion des certificats clients et serveurs à partir d'un client Subversion ordinaire.

[5] Bien que les certificats serveurs auto-signés soient vulnérables aux attaques du type *man-in-the-middle* (« attaque de l'homme du milieu » en français), une telle attaque est bien plus difficile à exécuter pour un observateur occasionnel que de juste espionner les mots de passe qui passent en clair sur le réseau.

Quand il communique avec Apache via `https://`, un client Subversion peut recevoir deux types d'informations différentes :

- un certificat serveur ;
- une demande de certificat client.

Si le client reçoit un certificat serveur, il doit vérifier que ce certificat est digne de confiance : le serveur est-il bien celui qu'il prétend être ? La bibliothèque OpenSSL effectue cette vérification en examinant le signataire du certificat serveur, aussi nommé *autorité de certification* (AC, ou *CA* en anglais). Si OpenSSL n'est pas capable de faire confiance automatiquement à l'autorité de certification, ou si un autre problème apparaît (tel que l'expiration du certificat ou des noms d'hôtes divergents), le client en ligne de commande de Subversion vous demande si vous voulez faire confiance au certificat serveur malgré tout :

```
$ svn list https://hote.exemple.com/depot/projet

Erreur de validation du certificat du serveur pour 'https://hote.exemple.com:443' :
 - Le certificat n'est pas signé pas une autorité de confiance.
   Valider le certificat manuellement !
Informations du certificat :
 - nom d'hôte : hote.exemple.com
 - valide de Jan 30 19:23:56 2004 GMT à Jan 30 19:23:56 2006 GMT
 - signataire : CA, exemple.com, Sometown, California, US
 - empreinte : 7d:e1:a9:34:33:39:ba:6a:e9:a5:c4:22:98:7b:76:5c:92:a0:9c:7b

(R)ejet, acceptation (t)emporaire ou (p)ermanente ?
```

Cette question devrait vous être familière ; c'est probablement la même que vous avez déjà dû voir posée par votre navigateur web (qui n'est rien d'autre qu'un client HTTP comme Subversion). Si vous choisissez l'option `(p)`ermanente, le certificat du serveur sera mis en cache dans votre zone privée `auth/` de la même façon que le nom d'utilisateur et le mot de passe y sont conservés (voir la section intitulée « Mise en cache des éléments d'authentification du client »). S'il est en cache, Subversion fera automatiquement confiance à ce certificat dans les échanges futurs.

Votre fichier `servers` dans la zone de configuration vous permet également d'indiquer au client Subversion qu'il doit automatiquement faire confiance à certaines autorités de certification spécifiques, soit globalement, soit en tenant compte de la machine hôte. Il suffit d'attribuer à la variable `ssl-authority-files` une liste de certificats d'autorités de certification au format PEM, séparés par des points-virgules :

```
[global]
ssl-authority-files = /chemin/vers/certAC-1.pem;/chemin/vers/certAC-2.pem
```

De nombreuses installations d'OpenSSL possèdent par défaut une liste prédéfinie d'autorités de certification auxquelles il est fait confiance de manière quasi-universelle. Pour que le client Subversion fasse confiance à ces autorités de certification automatiquement, mettez la variable `ssl-trust-default-ca` à `true`.

Quand il communique avec Apache, un client Subversion peut également recevoir une demande (un défi) de certificat client. Apache demande au client de s'authentifier : le client est-il bien celui qu'il prétend être ? Si tout se passe bien, le client Subversion renvoie un certificat (public) signé par une autorité de certification en laquelle Apache a confiance ainsi qu'une preuve que le client possède bien la clé privée associée au certificat (la réponse au défi). La clé privée, ainsi que le certificat public, sont habituellement enregistrés dans un conteneur (un fichier « .p12 », fichier au format PKCS#12) sur le disque, la clé privée étant chiffrée à l'aide d'un mot de passe. Quand Subversion reçoit ce défi, il vous demande le chemin d'accès au conteneur ainsi que le mot de passe qui protège la clé privée :

```
$ svn list https://hote.exemple.com/depot/projet

Domaine d'authentification :  https://hote.exemple.com:443
Fichier du certificat client : /chemin/vers/mon/cert.p12
Phrase de passe pour '/chemin/vers/mon/cert.p12' : ********
…
```

Pour utiliser un certificat client avec Subversion, il doit être dans un conteneur au format PKCS#12, qui est un standard portable. La plupart des navigateurs web sont déjà capables d'importer et d'exporter des certificats dans ce format. Une autre option est d'utiliser les outils en ligne de commande d'OpenSSL pour convertir les certificats existants en PKCS#12.

Encore une fois, le fichier `servers` de la zone de configuration vous permet d'automatiser la réponse à ces demandes en tenant compte de la machine hôte. L'une ou l'autre des informations, ou bien les deux, peuvent être décrites dans les variables de configuration :

```
[groups]
exemple-d-hote = hote.exemple.com

[exemple-d-hote]
ssl-client-cert-file = /chemin/vers/mon/cert.p12
ssl-client-cert-password = un-mot-de-passe
```

Une fois que vous avez défini les variables `ssl-client-cert-file` et `ssl-client-cert-password`, le client Subversion peut répondre automatiquement à une demande de certificat client sans la moindre interaction de votre part[6].

Options de contrôle d'accès

À ce stade, vous avez configuré l'authentification mais pas le contrôle d'accès. Apache est capable d'envoyer des défis aux clients afin de confirmer leur identité mais on ne lui a pas encore dit comment autoriser ou restreindre les accès en fonction de l'utilisateur. Ce paragraphe décrit deux stratégies pour contrôler les accès au dépôt.

Contrôle d'accès générique

La forme la plus simple de contrôle d'accès est d'autoriser des utilisateurs donnés à accéder à un dépôt en lecture seule ou en lecture/écriture.

Vous pouvez restreindre l'accès à toutes les opérations du dépôt en ajoutant la directive `Require valid-user` à votre bloc `<Location>`. Pour poursuivre dans la veine de notre exemple précédent, cela signifie que seuls les clients disant être `harry` ou `sally` et qui ont fourni le bon mot de passe pour leur nom d'utilisateur respectif ont l'autorisation d'effectuer des actions sur le dépôt Subversion :

```
<Location /svn>
  DAV svn
  SVNParentPath /var/svn

  # comment authentifier un utilisateur
  AuthType Basic
  AuthName "Depot Subversion"
  AuthUserFile /chemin/vers/fichier/utilisateurs

  # seuls les utilisateurs authentifiés ont le droit d'accès au dépôt
  Require valid-user
</Location>
```

Parfois vous n'aurez pas besoin d'être aussi strict. Par exemple, le propre dépôt du code source de Subversion, situé à l'adresse http://svn.apache.org/repos/asf/subversion/ autorise toute personne à effectuer des opérations de lecture du dépôt (comme extraire des copies de travail ou naviguer dans le dépôt avec un navigateur web) mais restreint toutes les opérations d'écriture aux utilisateurs authentifiés. Pour accomplir ce type de restriction sélective, vous pouvez utiliser les directives de configuration `Limit` et `LimitExcept`. Tout comme la directive `Location`, ces blocs doivent débuter et finir par des balises et vous devez les imbriquer dans votre bloc `<Location>`.

Les paramètres présents dans les directives `Limit` et `LimitExcept` sont des types de requêtes HTTP qui sont concernés par ce bloc. Par exemple, si voulez désactiver tout accès au dépôt excepté pour les opérations de lecture déjà supportées, utilisez la directive `LimitExcept`, en lui passant les paramètres de type de requêtes `GET`, `PROPFIND`, `OPTIONS` et `REPORT`. Ensuite

[6] Les administrateurs qui attachent le plus d'importance à la sécurité peuvent ne pas vouloir stocker le mot de passe du certificat client dans le fichier `servers` de la zone de configuration.

placez la directive `Require valid-user` mentionnée précédemment à l'intérieur du bloc `<LimitExcept>` au lieu de la mettre juste à l'intérieur du bloc `<Location>`.

```
<Location /svn>
  DAV svn
  SVNParentPath /var/svn

  # comment authentifier un utilisateur
  AuthType Basic
  AuthName "Depot Subversion"
  AuthUserFile /chemin/vers/fichier/utilisateurs

  # Pour toute autre opération que celles-ci, exiger un utilisateur authentifié.
  <LimitExcept GET PROPFIND OPTIONS REPORT>
    Require valid-user
  </LimitExcept>
</Location>
```

Ce ne sont là que quelques exemples simples. Pour des informations plus poussées sur le contrôle d'accès d'Apache et la directive `Require`, jetez un œil à la section `Security` du recueil des didacticiels de la documentation Apache à l'adresse http://httpd.apache.org/docs-2.0/misc/tutorials.html (site en anglais).

Contrôle d'accès par répertoire

On peut également mettre en place des droits d'accès avec une granularité plus fine en utilisant un second module Apache httpd, **mod_authz_svn**. Ce module se saisit des diverses URL opaques qui transitent entre le client et le serveur, demande à **mod_dav_svn** de les décoder et ensuite met éventuellement son veto à certaines requêtes en se basant sur les politiques d'accès définies dans un fichier de configuration.

Si vous avez compilé Subversion à partir du code source, **mod_authz_svn** est automatiquement inclus et installé aux côtés de **mod_dav_svn**. De nombreux exécutables distribués l'installent automatiquement aussi. Pour vérifier qu'il est installé correctement, assurez-vous qu'il vient juste après la directive `LoadModule` de **mod_dav_svn** dans le fichier `httpd.conf` :

```
LoadModule dav_module         modules/mod_dav.so
LoadModule dav_svn_module     modules/mod_dav_svn.so
LoadModule authz_svn_module   modules/mod_authz_svn.so
```

Pour activer ce module, vous devez configurer votre bloc `Location` pour qu'il utilise la directive `AuthzSVNAccessFile` ; elle spécifie quel fichier contient les politiques de contrôle d'accès aux chemins de vos dépôts (nous allons étudier le format de ce fichier).

Apache est flexible, vous avez donc la possibilité de configurer votre bloc selon une méthode à choisir parmi trois. Pour commencer, choisissez une des méthodes de configuration de base (les exemples suivants sont très simples ; reportez-vous à la documentation Apache qui contient beaucoup plus de détails sur les options d'authentification et de contrôle d'accès d'Apache).

Le bloc le plus simple consiste à donner l'accès à tout le monde. Dans ce scénario, Apache n'envoie jamais de défi d'authentification, tous les utilisateurs sont donc traités en tant qu'« anonymes » (voir l'Exemple 6.1, « Exemple-type de configuration : accès anonyme »).

Exemple 6.1. Exemple-type de configuration : accès anonyme

```
<Location /depot>
  DAV svn
  SVNParentPath /var/svn

  # notre politique de contrôle d'accès
  AuthzSVNAccessFile /chemin/vers/fichier/acces
</Location>
```

À l'opposé sur l'échelle de la paranoïa, vous pouvez configurer votre bloc de telle sorte qu'il exige que tout le monde s'authentifie. Ici, tous les clients doivent fournir un mot de passe pour prouver leur identité. Votre bloc exige l'authentification inconditionnelle via la directive `Require valid-user` et définit le moyen de s'authentifier (voir l'Exemple 6.2, « Exemple-type de configuration : accès authentifié »).

Exemple 6.2. Exemple-type de configuration : accès authentifié

```
<Location /depot>
  DAV svn
  SVNParentPath /var/svn

  # notre politique de contrôle d'accès
  AuthzSVNAccessFile /chemin/vers/fichier/acces

  # seuls les utilisateurs authentifiés ont accès au dépôt
  Require valid-user

  # comment authentifier un utilisateur
  AuthType Basic
  AuthName "Depot Subversion"
  AuthUserFile /chemin/vers/fichier/utilisateurs
</Location>
```

Une troisième méthode largement pratiquée consiste à autoriser à la fois des accès anonymes et des accès authentifiés. Par exemple, de nombreux administrateurs désirent autoriser les utilisateurs anonymes à lire certains répertoires dans les dépôts mais ne veulent voir que des utilisateurs authentifiés accéder en lecture (ou en écriture) dans des zones plus sensibles. Dans cette configuration, tous les utilisateurs commencent par accéder au dépôt de façon anonyme. Si votre politique de contrôle d'accès exige un véritable nom d'utilisateur à un moment ou à un autre, Apache demandera au client de s'authentifier. Pour ce faire, vous devez utiliser à la fois la directive `Satisfy Any` et la directive `Require valid-user` (voir l'Exemple 6.3, « Exemple-type de configuration : accès mixte authentifié/anonyme »).

Exemple 6.3. Exemple-type de configuration : accès mixte authentifié/anonyme

```
<Location /depot>
  DAV svn
  SVNParentPath /var/svn

  # notre politique de contrôle d'accès
  AuthzSVNAccessFile /chemin/vers/fichier/acces

  # essayer d'abord un accès anonyme,
  # ne demander une véritable authentification
  # que si nécessaire
  Satisfy Any
  Require valid-user

  # comment authentifier un utilisateur
  AuthType Basic
  AuthName "Depot Subversion"
  AuthUserFile /chemin/vers/fichier/utilisateurs
</Location>
```

Une fois l'un de ces modèles mis en place dans votre fichier `httpd.conf`, vous devez créer un fichier contenant les règles d'accès particulières pour chaque chemin de votre dépôt. Nous en parlons plus en détails plus loin dans ce chapitre, à la section intitulée « Contrôle d'accès basé sur les chemins ».

Désactivation du contrôle sur les chemins

Le module **mod_dav_svn** se donne beaucoup de peine pour assurer que les données que vous avez désignées comme « interdites » ne soient pas divulguées accidentellement. Cela veut dire qu'il doit surveiller étroitement tous les chemins d'accès et tous les contenus des fichiers renvoyés par des commandes telles que **svn checkout** et **svn update**. Si ces commandes tombent sur un chemin qui n'est pas lisible à cause d'une politique de contrôle d'accès, le chemin est généralement totalement omis. Dans le cas d'une recherche d'historique ou de renommage, par exemple une commande telle que **`svn cat -r ANCIENNE_REVISION truc.c`** sur un fichier qui a été renommé il y a longtemps, le suivi des renommages va simplement s'arrêter si un des anciens noms de l'objet se révèle être en accès restreint en lecture.

Ces contrôles sur les chemins peuvent parfois être assez coûteux, tout particulièrement dans le cas de **svn log**. Quand il demande une liste de révisions, le serveur examine chaque chemin modifié dans chaque révision et vérifie qu'il a le droit d'être lu. Si un chemin interdit est découvert, il est omis de la liste des chemins modifiés par la révision (obtenue habituellement par l'option `--verbose`) et le message de propagation est entièrement supprimé. Il va sans dire que ceci peut prendre un certain temps pour les révisions qui ont touché à un grand nombre de fichiers. C'est le coût de la sécurité : même si vous n'avez pas du tout configuré de module tel que **mod_authz_svn**, le module **mod_dav_svn** va quand même demander à **httpd** Apache de vérifier les droits d'accès pour chaque chemin. Le module **mod_dav_svn** n'est pas au courant de la liste des modules qui ont été installés, donc tout ce qu'il peut faire est de demander à Apache de lancer n'importe quel contrôle qui soit présent.

D'un autre côté, il existe une sorte d'échappatoire qui vous permet de faire un compromis entre les fonctions de sécurité et la vitesse. Si vous ne mettez pas en œuvre de contrôle d'accès sur les chemins (c'est-à-dire, si vous n'utilisez ni **mod_authz_svn** ni un module similaire), vous pouvez désactiver tous ces contrôles sur les chemins. Dans votre fichier `httpd.conf`, utilisez la directive `SVNPathAuthz` comme illustré dans l'Exemple 6.4, « Désactiver complètement les contrôles sur les chemins ».

Exemple 6.4. Désactiver complètement les contrôles sur les chemins

```
<Location /depot>
  DAV svn
  SVNParentPath /var/svn

  SVNPathAuthz off
</Location>
```

La directive `SVNPathAuthz` est activée (« on ») par défaut. Quand on la désactive (« off »), tous les contrôles d'accès basés sur les chemins sont désactivés ; **mod_dav_svn** arrête de contrôler chaque chemin qu'il traite.

Fonctionnalités bonus

Nous avons traité la plupart des options d'authentification et de contrôle d'accès pour Apache et **mod_dav_svn**. Mais Apache fournit quelques autres fonctionnalités sympathiques.

Navigation dans les dépôts

Un des avantages les plus frappants d'avoir une configuration Apache/WebDAV pour votre dépôt Subversion est que les révisions les plus récentes de vos fichiers et répertoires suivis en versions sont immédiatement consultables à l'aide d'un navigateur web classique. Puisque Subversion utilise des URL pour identifier les ressources suivies en versions, ces URL utilisées pour accéder aux dépôts via HTTP peuvent être tapées directement dans un navigateur web. Votre navigateur enverra une requête `HTTP GET` pour cette URL ; selon que cette URL représente ou non un fichier ou un répertoire suivi en versions, **mod_dav_svn** renverra soit la liste des éléments du répertoire, soit le contenu du fichier.

Comme les URL ne contiennent pas d'informations concernant la version de la ressource qui vous intéresse, **mod_dav_svn** renverra toujours la version la plus récente. Cette fonctionnalité a un merveilleux effet secondaire : vous pouvez partager avec vos pairs des URL Subversion en guise de références à des documents et ces URL pointeront toujours vers la dernière version des documents. Bien sûr, vous pouvez aussi utiliser ces URL en tant que liens hypertextes dans d'autres sites web.

Puis-je consulter d'anciennes révisions ?

Avec un navigateur web classique ? En un mot : non. Ou, du moins, pas avec **mod_dav_svn** pour seul outil.

Votre navigateur web ne sait communiquer qu'en langage HTTP ordinaire. Ce qui signifie qu'il sait seulement obtenir, par requête `GET`, des URL publiques, qui représentent les versions les plus récentes des fichiers et répertoires. D'après la spécification de WebDAV/DeltaV, chaque serveur définit une syntaxe privée d'URL pour les anciennes versions des ressources et cette syntaxe est incompréhensible pour les clients. Pour obtenir une ancienne version d'un fichier, un client doit suivre une procédure spécifique pour « découvrir » l'URL appropriée ; cette procédure nécessite d'envoyer une série de requêtes WebDAV `PROPFIND` et de comprendre les concepts DeltaV. C'est quelque chose que votre navigateur web n'est pas capable de faire.

Donc, pour répondre à la question, une méthode évidente permettant de consulter d'anciennes versions de fichiers et de répertoires est de passer l'argument `--revision` (`-r`) aux commandes **svn list** et **svn cat**. Pour naviguer dans les anciennes révisions avec votre navigateur web, vous pouvez utiliser des logiciels tiers. Un bon exemple en est ViewVC (http://viewvc.tigris.org/). ViewVC a été écrit à l'origine dans le but d'afficher des dépôts CVS sur le web[7] et les dernières versions sont également capables de travailler avec les dépôts Subversion.

Types MIME appropriés

Quand il consulte un dépôt Subversion, le navigateur web obtient un indice pour savoir comment rendre le contenu d'un fichier en examinant l'entête `Content-Type:` qui fait partie de la réponse envoyée par Apache à la requête HTTP `GET`. La valeur de cet entête est en quelque sorte un type MIME. Par défaut, Apache va indiquer aux navigateurs web que tous les fichiers du dépôt sont du type MIME par défaut, en général `text/plain`. Cela peut s'avérer assez frustrant, si un utilisateur désire visualiser les fichiers du dépôt de manière plus appropriée — par exemple, un fichier `truc.html` du dépôt sera bien plus lisible s'il est rendu dans le navigateur en tant que fichier HTML.

Pour rendre ceci possible, il suffit de vous assurer que vos fichiers portent bien la propriété `svn:mime-type`. Plus de détails sur ce sujet sont disponibles dans la section intitulée « Type de contenu des fichiers » et vous pouvez même configurer votre dépôt pour qu'il associe automatiquement la valeur de `svn:mime-type` appropriée aux fichiers qui arrivent dans le dépôt pour la première fois ; reportez-vous à la section intitulée « Configuration automatique des propriétés ».

Donc, dans notre exemple, si quelqu'un attribuait la valeur `text/html` à la propriété `svn:mime-type` du fichier `truc.html`, Apache indiquerait avec justesse à votre navigateur web de rendre le fichier comme une page HTML. On pourrait aussi associer des propriétés ayant des valeurs `image/*` appropriées aux fichiers d'images et, en fin de compte, faire qu'un site web entier soit consultable directement à travers un dépôt ! Ceci ne pose en général pas de problème, du moment que le site web ne possède pas de contenu généré dynamiquement.

Personnalisation de l'aspect

En général, vous utiliserez principalement des URL de fichiers suivis en versions ; après tout c'est là que le contenu intéressant réside. Mais vous aurez peut-être l'occasion de naviguer dans le contenu d'un répertoire Subversion et vous remarquerez rapidement que le code HTML généré pour afficher la liste des éléments du répertoire est très rudimentaire, et certainement pas conçu pour être agréable d'un point de vue esthétique (ni même intéressant). Afin d'activer la personnalisation de l'affichage de ces répertoires, Subversion fournit une fonctionnalité d'index XML. La présence d'une directive `SVNIndexXSLT` dans le bloc `Location` du fichier `httpd.conf` de votre dépôt conduira **mod_dav_svn** à générer un résultat en XML quand il affiche la liste des éléments d'un répertoire et à faire référence à la feuille de style XSLT de votre choix :

```
<Location /svn>
  DAV svn
  SVNParentPath /var/svn
  SVNIndexXSLT "/index-svn.xsl"
  …
</Location>
```

A l'aide de la directive `SVNIndexXSLT` et d'une feuille de style XSLT faisant preuve de créativité, vous pouvez adapter les

[7] A l'époque, il s'appelait ViewCVS.

listes de contenus de répertoires aux schémas de couleur et d'imagerie utilisés dans d'autres parties de votre site web. Ou, si vous préférez, vous pouvez utiliser les exemples de feuilles de style fournis dans le répertoire `tools/xslt/` du code source de Subversion. Gardez à l'esprit que le chemin d'accès fourni au répertoire `SVNIndexXSLT` est en fait une URL — les navigateurs de chemins doivent être capables de lire vos feuilles de style pour les utiliser !

Affichage de la liste des dépôts

Si vous desservez un ensemble de dépôts à partir d'une URL unique via la directive `SVNParentPath`, il est possible de faire afficher par Apache tous les dépôts disponibles dans un navigateur web. Il suffit d'activer la directive `SVNListParentPath` :

```
<Location /svn>
  DAV svn
  SVNParentPath /var/svn
  SVNListParentPath on
  …
</Location>
```

Si un utilisateur pointe son navigateur web à l'URL `http://hote.exemple.com/svn/`, il verra une liste de tous les dépôts Subversion situés dans `/var/svn`. Évidemment, ceci peut poser des problèmes de sécurité ; cette fonctionnalité est donc désactivée par défaut.

Journalisation Apache

Comme Apache est avant tout un serveur HTTP, il contient des fonctionnalités de journalisation d'une flexibilité fantastique. Détailler toutes les façons dont la journalisation peut être configurée sort du cadre de ce livre, mais nous voulons quand même vous faire remarquer que même le fichier `httpd.conf` le plus générique conduit Apache à créer deux fichiers de journalisation : `error_log` et `access_log`. Ces journaux peuvent apparaître à différents endroits, mais sont en général créés dans la zone de journalisation de votre installation Apache (sur Unix, ils résident souvent dans `/usr/local/apache2/logs/`).

Le fichier `error_log` décrit toutes les erreurs internes qu'Apache rencontre au cours de son fonctionnement. Le fichier `access_log` enregistre toutes les requêtes HTTP reçues par Apache. Ceci permet de voir facilement, par exemple, de quelles adresses IP les clients Subversion se connectent, à quelle fréquence un client donné utilise le serveur, quels utilisateurs se sont authentifiés correctement et quelles requêtes ont échoué ou réussi.

Malheureusement, parce qu'HTTP est un protocole sans notion d'état, même la plus simple opération du client Subversion génère plusieurs requêtes réseau. Il est donc très difficile d'examiner le fichier `access_log` et d'en déduire ce que le client faisait — la plupart des opérations se présentent sous la forme de séries de requêtes `PROPPATCH`, `GET`, `PUT` et `REPORT` énigmatiques. Pour couronner le tout, de nombreuses opérations du client envoient des séries de requêtes quasi-identiques, il est donc encore plus difficile de les distinguer.

Cependant, **mod_dav_svn** peut venir à votre rescousse. En activant la fonctionnalité de « journalisation opérationnelle », vous pouvez demander à **mod_dav_svn** de créer un fichier séparé décrivant quelles sortes d'opérations de haut niveau vos clients effectuent.

Pour ce faire, vous devez utiliser la directive `CustomLog` d'Apache (qui est expliquée en détail dans la documentation Apache). Prenez soin de placer cette directive *en dehors* de votre bloc `Location` de Subversion :

```
<Location /svn>
  DAV svn
  …
</Location>

CustomLog logs/journal-svn "%t %u %{SVN-ACTION}e" env=SVN-ACTION
```

Dans cet exemple, nous demandons à Apache de créer un fichier journal spécial, `journal-svn`, dans le répertoire habituel de journaux d'Apache (`logs`). Les variables `%t` et `%u` sont remplacées par l'horodatage et le nom d'utilisateur de la requête, respectivement. Les points importants sont les deux instances de `SVN-ACTION`. Quand Apache trouve cette variable, il lui substitue la valeur de la variable d'environnement `SVN-ACTION`, modifiée automatiquement par **mod_dav_svn** quand il détecte une action haut-niveau du client.

Ainsi, au lieu d'avoir à interpréter un fichier `access_log` traditionnel qui ressemble à :

```
[26/Jan/2007:22:25:29 -0600] "PROPFIND /svn/calc/!svn/vcc/default HTTP/1.1" 207 398
[26/Jan/2007:22:25:29 -0600] "PROPFIND /svn/calc/!svn/bln/59 HTTP/1.1" 207 449
[26/Jan/2007:22:25:29 -0600] "PROPFIND /svn/calc HTTP/1.1" 207 647
[26/Jan/2007:22:25:29 -0600] "REPORT /svn/calc/!svn/vcc/default HTTP/1.1" 200 607
[26/Jan/2007:22:25:31 -0600] "OPTIONS /svn/calc HTTP/1.1" 200 188
[26/Jan/2007:22:25:31 -0600] "MKACTIVITY
/svn/calc/!svn/act/e6035ef7-5df0-4ac0-b811-4be7c823f998 HTTP/1.1" 201 227
…
```

vous pouvez vous contenter de parcourir le fichier `journal-svn` qui est bien plus intelligible :

```
[26/Jan/2007:22:24:20 -0600] - get-dir /tags r1729 props
[26/Jan/2007:22:24:27 -0600] - update /trunk r1729 depth=infinity send-copyfrom-args
[26/Jan/2007:22:25:29 -0600] - status /trunk/machin r1729 depth=infinity
[26/Jan/2007:22:25:31 -0600] sally commit r1730
```

Pour une liste exhaustive de toutes les actions journalisées, reportez-vous à la section intitulée « Journalisation de haut niveau ».

Mandataire en écriture

Un des avantages notables d'Apache comme serveur Subversion est qu'il peut être configuré pour effectuer de la réplication. Par exemple, imaginez que votre équipe soit répartie sur quatre sites dans différents coins du globe. Le dépôt Subversion ne peut exister que sur un de ces sites, ce qui signifie que les trois autres sites n'auront pas un accès très satisfaisant — ils devront sans doute faire avec un trafic plus lent et des temps de réponse plus longs lors des mises à jour et des propagations. Une solution très puissante est de mettre en place un système constitué d'un serveur Apache *maître* et de plusieurs serveurs Apache *esclaves*. Si vous placez un serveur esclave sur chacun des sites, les utilisateurs peuvent extraire une copie de travail de l'esclave qui est le plus proche d'eux. Toutes les requêtes de lecture vont au serveur esclave local. Les requêtes d'écriture sont automatiquement routées vers l'unique serveur maître. Lorsque la propagation se termine, le maître « pousse » la nouvelle révision vers chaque serveur esclave en utilisant l'outil de réplication **svnsync**.

Cette configuration permet une immense amélioration de la vitesse perçue par les utilisateurs, car le trafic d'un client Subversion est généralement constitué à 80 - 90 % de requêtes de lecture. Et ces requêtes étant traitées par un serveur *local*, le gain est énorme.

Dans ce paragraphe, nous vous accompagnons dans la mise en place standard de ce système comportant un maître unique et plusieurs esclaves. Cependant, gardez à l'esprit que vos serveurs doivent faire tourner au moins Apache 2.2.0 (avec le module **mod_proxy** chargé) et Subversion 1.5 (**mod_dav_svn**).

Configuration des serveurs

Pour commencer, configurez le fichier `httpd.conf` de votre serveur maître de la façon habituelle. Mettez le dépôt à disposition à un certain emplacement URI et configurez l'authentification ainsi que le contrôle d'accès comme vous le souhaitez. Une fois que c'est fait, configurez chacun de vos serveurs « esclaves » exactement de la même manière, mais ajoutez la directive `SVNMasterURI` au bloc :

```
<Location /svn>
  DAV svn
  SVNPath /var/svn/depot
  SVNMasterURI http://maitre.exemple.com/svn
  …
</Location>
```

Cette nouvelle directive indique à un serveur esclave de rediriger toutes les requêtes d'écriture vers le maître (ce qui est accompli automatiquement par le module **mod_proxy** d'Apache). Les requêtes ordinaires de lecture, cependant, sont toujours traitées par les esclaves. Assurez-vous que vos serveurs maître et esclaves ont tous des configurations identiques d'authentification et de contrôle d'accès ; s'ils ne peuvent plus se synchroniser, cela peut engendrer beaucoup d'ennuis.

Ensuite nous devons nous occuper du problème de la récursion infinie. Avec la configuration actuelle, imaginez ce qui se va se passer quand un client Subversion va effectuer une propagation vers le serveur maître. Une fois la propagation terminée, le serveur utilise **svnsync** pour répliquer la nouvelle révision vers chaque esclave. Mais comme **svnsync** ne se présente que comme un simple client en train d'effectuer une propagation, l'esclave va immédiatement tenter d'envoyer vers le maître la requête d'écriture qui vient d'arriver ! Et là, patatras !

La solution consiste à s'arranger pour que le maître pousse les révisions vers un emplacement `<Location>` distinct au sein des dépôts esclaves. Cet emplacement est configuré *non pas pour servir de mandataire* pour les requêtes d'écriture mais pour accepter les propagations normales en provenance de l'adresse IP du maître (et seulement de lui) :

```
<Location /svn-proxy-sync>
  DAV svn
  SVNPath /var/svn/depot
  Order deny,allow
  Deny from all
  # Ne laisse que le serveur ayant l'adresse indiquée accéder à cet emplacement :
  Allow from 10.20.30.40
  …
</Location>
```

Mise en place de la réplication

Une fois que vous avez configuré les blocs `Location` du maître et des esclaves, vous devez configurer le maître pour que la réplication vers les esclaves fonctionne. Ceci se fait de la manière habituelle, en utilisant **svnsync**. Si vous n'êtes pas familier avec cet outil, reportez-vous à la section intitulée « Réplication d'un dépôt » pour plus de détails.

Tout d'abord, assurez-vous que chaque dépôt esclave a une procédure automatique `pre-revprop-change` qui autorise les modifications de propriétés de révisions à distance (cette étape fait partie de la procédure standard pour un serveur qui reçoit les révisions de **svnsync**). Ensuite, connectez-vous au serveur maître et configurez l'URI de chaque dépôt esclave pour qu'il reçoive les données en provenance du dépôt maître sur le disque local :

```
$ svnsync init http://esclave1.exemple.com/svn-proxy-sync file://var/svn/depot
Propriétés copiées pour la révision 0.
$ svnsync init http://esclave2.exemple.com/svn-proxy-sync file://var/svn/depot
Propriétés copiées pour la révision 0.
$ svnsync init http://esclave3.exemple.com/svn-proxy-sync file://var/svn/depot
Propriétés copiées pour la révision 0.

# Effectue la réplication initiale

$ svnsync sync http://esclave1.exemple.com/svn-proxy-sync
Transmission des données .....................................
Révision 1 propagée.
Propriétés copiées pour la révision 1.
Transmission des données ..
Révision 2 propagée.
Propriétés copiées pour la révision 2.
…

$ svnsync sync http://esclave2.exemple.com/svn-proxy-sync
Transmission des données .....................................
Révision 1 propagée.
Propriétés copiées pour la révision 1.
Transmission des données ..
Révision 2 propagée.
Propriétés copiées pour la révision 2.
…

$ svnsync sync http://esclave3.exemple.com/svn-proxy-sync
Transmission des données .....................................
Révision 1 propagée.
Propriétés copiées pour la révision 1.
Transmission des données ..
Révision 2 propagée.
```

```
Propriétés copiées pour la révision 2.
…
```

Une fois que c'est fait, nous configurons la procédure automatique post-propagation (`post-commit`) du serveur maître pour qu'elle lance **svnsync** sur chaque serveur esclave :

```
#!/bin/sh
# Procédure post-propagation pour répliquer les révisions nouvellement propagées vers
les esclaves

svnsync sync http://esclave1.exemple.com/svn-proxy-sync > /dev/null 2>&1
svnsync sync http://esclave2.exemple.com/svn-proxy-sync > /dev/null 2>&1
svnsync sync http://esclave3.exemple.com/svn-proxy-sync > /dev/null 2>&1
```

Les symboles en plus à la fin de chaque ligne ne sont pas nécessaires, mais constituent un moyen astucieux d'autoriser **svnsync** à lancer des commandes qui fonctionneront à l'arrière-plan, de telle sorte que le client Subversion ne se retrouvera pas à attendre indéfiniment que la propagation se termine. En plus de cette procédure post-propagation (`post-commit`), vous aurez également besoin d'une procédure `post-revprop-change` pour que, disons, quand un utilisateur modifie un message de propagation, les serveurs esclaves reçoivent aussi cette modification :

```
#!/bin/sh
# Procédure post-revprop-change pour répliquer les modifications
# des propriétés de révisions vers les esclaves

REV=${2}
svnsync copy-revprops http://esclave1.exemple.com/svn-proxy-sync ${REV} > /dev/null
2>&1
svnsync copy-revprops http://esclave2.exemple.com/svn-proxy-sync ${REV} > /dev/null
2>&1
svnsync copy-revprops http://esclave3.exemple.com/svn-proxy-sync ${REV} > /dev/null
2>&1
```

La seule chose que nous n'avons pas détaillé concerne les verrous. Comme les verrous sont gérés strictement par le serveur maître (le seul endroit où des propagations ont lieu), nous n'avons en théorie pas besoin de faire quoi que ce soit. De nombreuses équipes n'utilisent pas du tout les fonctionnalités de verrouillage de Subversion, il s'agit donc peut-être pour vous d'un faux problème. Cependant, si les modifications de verrous ne sont pas répliquées du maître vers les esclaves, cela signifie que les clients ne seront pas capables d'interroger l'état des verrous (par exemple, **`svn status -u`** ne donne aucune information sur les verrous du dépôt). Si cela vous embête, vous pouvez écrire des procédures automatiques `post-lock` et `post-unlock` qui lancent **svn lock** et **svn unlock** sur les serveurs esclaves, vraisemblablement à l'aide d'une méthode de connexion à distance telle que SSH. Nous laissons ceci au lecteur en guise d'exercice !

Pièges à éviter

Votre système de réplication maître/esclave doit à présent être prêt à l'emploi. Cependant, quelques consignes de prudence sont de mise. Souvenez-vous que la réplication n'est pas totalement robuste en ce qui concerne les plantages machine ou réseau. Par exemple, si l'une des commandes **svnsync** automatisées demeure inachevée, pour quelque raison que ce soit, les esclaves vont commencer à être décalés. Par exemple, vos utilisateurs distants verront qu'ils ont propagé la révision 100, mais quand ils exécuteront **svn update**, leur serveur local leur indiquera que la révision 100 n'existe pas encore ! Bien sûr, le problème se réglera automatiquement dès qu'une autre propagation aura lieu et que la commande **svnsync** qui s'ensuit aura fonctionné — cette synchronisation répliquera toutes les révisions en attente. Néanmoins, vous pouvez décider de mettre en place une surveillance des décalages, vérifiant le bon fonctionnement de la synchronisation et qui, en cas de problème, déclenche une nouvelle exécution de **svnsync**.

Pouvons-nous mettre en place la réplication avec svnserve ?

Si vous utilisez **svnserve** au lieu d'Apache comme serveur, vous pouvez tout à fait configurer les procédures automatiques de votre dépôt pour qu'elles lancent **svnsync** comme nous l'avons expliqué ici, lançant ainsi la réplication automatique du maître vers les esclaves. Malheureusement, à l'heure où nous écrivons ces lignes, il n'y a pas moyen de s'assurer que des serveurs esclaves **svnserve** envoient automatiquement les requêtes d'écriture vers le serveur maître.

Cela veut dire que vos utilisateurs ne pourraient pas extraire des copies de travail en lecture seule des serveurs esclaves. Il vous faudrait configurer vos serveurs esclaves pour qu'ils refusent complètement tout accès en écriture. Cela peut être utile pour créer des « miroirs » en lecture seule de projets open source populaires, mais il ne s'agit alors plus d'un système de mandataire d'écriture transparent.

Autres fonctionnalités d'Apache

Il y a également plusieurs fonctionnalités fournies par Apache, en tant que serveur web robuste, dont on peut tirer profit pour améliorer les fonctionnalités ou la sécurité de Subversion. Le client Subversion est capable d'utiliser SSL (Secure Socket Layer, le protocole de sécurisation des échanges sur internet, présenté auparavant). Si votre client Subversion a été compilé en incluant le support de SSL, il peut accéder à votre serveur Apache en utilisant des URL `https://` et bénéficier d'une session réseau avec un chiffrement de grande qualité.

D'autres fonctionnalités de la relation Apache/Subversion sont également tout aussi utiles, comme par exemple la possibilité de spécifier un port personnalisé (au lieu du port HTTP par défaut, 80) ou un nom de domaine virtuel par lequel accéder au dépôt Subversion ou la possibilité d'accéder au dépôt via un serveur mandataire HTTP.

Enfin, comme **mod_dav_svn** se sert d'un sous-ensemble du protocole WebDAV/DeltaV pour communiquer, il est possible d'accéder au dépôt depuis des clients DAV tiers. La possibilité de monter un serveur DAV en tant que « dossier partagé » réseau standard est intégrée dans la plupart des systèmes d'exploitation modernes (Win32, OS X et Linux). C'est un sujet compliqué, mais merveilleux une fois mis en place. Pour plus de détails, consultez l'Annexe C, *WebDAV et la gestion de versions automatique*.

Notez qu'il y a un certain nombre d'autres petits « bricolages » que l'on peut faire autour de **mod_dav_svn** qui sont trop obscurs pour être mentionnés dans ce chapitre. Pour voir la liste complète de toutes les directives `httpd.conf` auxquelles **mod_dav_svn** obéit, reportez-vous à la section intitulée « Directives ».

Contrôle d'accès basé sur les chemins

Apache et **svnserve** sont tous deux capables d'accorder ou de refuser l'accès aux utilisateurs. Généralement c'est géré globalement au niveau du dépôt : un utilisateur peut accéder (ou pas) au dépôt en lecture et il peut accéder (ou pas) au dépôt en écriture. Il est pourtant aussi possible de définir des règles d'accès possédant une granularité plus fine. Un ensemble d'utilisateurs peut alors obtenir le droit d'écrire dans certains répertoires du dépôt mais pas dans d'autres ; parallèlement, un autre répertoire peut très bien ne pas être accessible en lecture pour la majorité des utilisateurs.

Les deux types de serveurs utilisent un format de fichier commun pour décrire les règles d'accès basées sur les chemins. Dans le cas d'Apache, il faut charger le module **mod_authz_svn** puis ajouter la directive `AuthzSVNAccessFile` (dans le fichier `httpd.conf`) pointant vers votre propre fichier de règles (pour l'explication complète, voir la section intitulée « Contrôle d'accès par répertoire »). Si vous utilisez **svnserve**, vous devez faire pointer la variable `authz-db` (dans le fichier `svnserve.conf`) vers votre fichier de règles.

Avez-vous vraiment besoin d'un contrôle d'accès basé sur les chemins ?

De nombreux administrateurs qui mettent en place Subversion pour la première fois ont tendance à se lancer dans le contrôle d'accès basé sur les chemins sans trop y réfléchir. L'administrateur sait en général quelles équipes travaillent sur quel projet, il est dès lors facile de démarrer en accordant l'accès pour certains répertoires à certaines équipes et pas à d'autres. Ceci peut sembler assez naturel, et assouvir le désir de l'administrateur de contrôler le dépôt de très près.

Notez cependant qu'il y a souvent des coûts invisibles (et visibles !) associés à cette fonctionnalité. Dans la catégorie visible, le serveur doit faire beaucoup de travail pour s'assurer que l'utilisateur a le droit de lire ou d'écrire sur chaque chemin spécifié ; dans certaines situations, il y a une chute très significative des performances. Dans la catégorie invisible, réfléchissez à la culture que vous créez. La plupart du temps, même si certains utilisateurs ne devraient pas propager de modifications dans certaines parties du dépôt, ce contrat social n'a pas besoin de solution technologique pour être respecté. Les équipes peuvent parfois collaborer spontanément entre elles ; quelqu'un peut vouloir aider quelqu'un d'autre en effectuant une propagation dans une zone qui n'est pas celle où il travaille habituellement. En interdisant ce genre de choses au niveau du serveur, vous mettez en place une barrière à la collaboration. Vous créez aussi tout un tas de règles qui doivent être gérées au fur et à mesure que les projets se développent, que de nouveaux utilisateurs sont ajoutés, etc. C'est une quantité de travail supplémentaire à fournir.

Souvenez-vous que c'est un système de gestion de versions ! Même si quelqu'un propageait accidentellement une modification là où il n'aurait pas du, revenir en arrière reste très facile. Et si un utilisateur propage au mauvais endroit de façon intentionnelle, c'est un problème social qui doit être réglé, de toute manière, en dehors de Subversion.

Bref, avant que vous ne commenciez à restreindre les droits d'accès des utilisateurs, demandez-vous si cela correspond à un véritable besoin ou si c'est juste quelque chose qui « plaît » à l'administrateur. Demandez-vous si ça vaut la peine de sacrifier de la vitesse côté serveur et souvenez-vous que les risques associés sont très minimes ; ce n'est pas une bonne idée d'attendre de la technologie qu'elle résolve les problèmes sociaux[8].

En guise d'exemple à méditer, prenez le cas du projet Subversion lui-même, au sein duquel il a toujours été clairement défini quel utilisateur avait le droit d'effectuer des propagations à quel endroit, règles qui ont toujours été appliquées socialement. C'est un bon modèle de confiance dans la communauté, en particulier pour les projets open source. Bien sûr, il peut parfois y avoir de véritables besoins légitimant un contrôle d'accès basé sur les chemins ; dans les grandes entreprises, par exemple, certaines données *sont* sensibles et l'accès à ces données doit vraiment être restreint à un petit groupe de personnes.

Une fois que votre serveur sait où trouver votre fichier de règles, il est temps de définir ces règles.

La syntaxe du fichier est la même syntaxe que celle utilisée dans `svnserve.conf` et dans les fichiers de configuration. Les lignes commençant par un dièse (#) sont ignorées. Dans la forme la plus simple, chaque section désigne un dépôt et un chemin à l'intérieur de celui-ci et les noms d'utilisateurs authentifiés sont les noms des options à l'intérieur de chaque section. La valeur de chaque option décrit le niveau d'accès de l'utilisateur au chemin du dépôt : soit `r` (lecture seule), soit `rw` (lecture/écriture). Si l'utilisateur ne figure pas dans la section, l'accès n'est pas autorisé.

Plus précisément, la valeur des noms de section est soit de la forme `[nom-du-depot:chemin]`, soit de la forme `[chemin]`. Si vous utilisez la directive `SVNParentPath`, il est important de spécifier les noms des dépôts dans vos sections. Si vous les omettez, une section telle que `[/un/repertoire]` correspondra au chemin `/un/repertoire` de *tous les dépôts*. Cependant, si vous utilisez la directive `SVNPath`, ne définir que des chemins dans vos sections est acceptable — après tout, il n'y a qu'un seul dépôt.

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

Dans ce premier exemple, l'utilisateur `harry` a les droits d'accès complets en lecture/écriture au répertoire `/branches/calc/bogue-142` du dépôt `calc`, alors que l'utilisateur `sally` n'a que l'accès en lecture. Tous les autres utilisateurs sont bloqués et ne peuvent pas accéder à ce répertoire.

Bien évidemment, les droits d'accès sont hérités d'un répertoire parent à ses fils. Ce qui signifie que nous pouvons spécifier un sous-répertoire avec une politique d'accès différente pour Sally :

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

# donne à sally les droits d'écriture sur le sous-répertoire "test"
[calc:/branches/calc/bogue-142/test]
sally = rw
```

Maintenant Sally peut écrire dans le sous-répertoire `test` de la branche, mais ne peut toujours que lire les autres parties. Harry, en attendant, continue à avoir les droits d'accès complets en lecture écriture sur toute la branche.

Il est aussi possible d'interdire explicitement l'accès à quelqu'un grâce aux règles d'héritage, en attribuant la valeur vide à un nom d'utilisateur :

```
[calc:/branches/calc/bogue-142]
```

[8] Un thème récurrent dans ce livre !

```
harry = rw
sally = r

[calc:/branches/calc/bogue-142/secret]
harry =
```

Dans cet exemple, Harry a les droits de lecture/écriture sur l'arborescence `bogue-142` toute entière, mais n'a absolument pas accès au répertoire `secret` contenu dans celle-ci.

Ce qu'il faut retenir est que le chemin le plus spécifique est choisi en premier. Le serveur tente de trouver une correspondance avec le chemin lui-même, puis avec son chemin parent, puis avec le parent du parent, etc. Le résultat est que tout chemin spécifié dans le fichier des accès prendra le pas sur les droits hérités de ses répertoires parents.

Par défaut, personne n'a accès au dépôt. Cela signifie que si vous démarrez avec un fichier vide, vous voudrez probablement au moins donner les droits de lecture sur la racine du dépôt à tous les utilisateurs. Vous pouvez accomplir ceci en utilisant la variable astérisque (`*`), qui désigne « tous les utilisateurs » :

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

C'est une configuration très répandue ; notez qu'aucun nom de dépôt n'est mentionné dans le nom de la section. Ceci rend tous les dépôts accessibles en lecture à tous les utilisateurs. Une fois que tous les utilisateurs ont l'accès en lecture aux dépôts, vous pouvez accorder des droits d'écriture (`rw`) explicites à certains utilisateurs sur des sous-répertoires spécifiques à l'intérieur de dépôts spécifiques.

La variable astérisque (`*`) a aussi ceci de spécial qu'elle est le *seul* symbole qui puisse correspondre à un utilisateur anonyme. Si vous avez configuré votre serveur pour qu'il autorise un mélange d'accès anonymes et authentifiés, tous les utilisateurs peuvent commencer à y accéder anonymement. Le serveur cherche une valeur `*` définie pour le chemin d'accès demandé ; s'il n'en trouve pas, il demande au client de s'authentifier.

Le fichier des accès vous permet aussi de définir des groupes entiers d'utilisateurs, à la façon du fichier Unix `/etc/group` :

```
[groups]
developpeurs-calc = harry, sally, joe
developpeurs-paint = frank, sally, jane
tout-le-monde = harry, sally, joe, frank, sally, jane
```

Les droits d'accès peuvent être accordés aux groupes de la même façon qu'à de simples utilisateurs. Il faut juste les mettre en évidence par le préfixe « at » (`@`) :

```
[calc:/projets/calc]
@developpeurs-calc = rw

[paint:/projets/paint]
jane = r
@developpeurs-paint = rw
```

Un autre fait notable est que la première règle vérifiée est celle qui sera appliquée à l'utilisateur. Dans l'exemple précédent, même si `jane` est membre du groupe `développeurs-paint` (qui a les droits de lecture/écriture), la règle `jane = r` sera lue et vérifiée avant la règle du groupe, refusant ainsi à Jane l'accès en écriture.

Les groupes peuvent aussi contenir d'autres groupes :

```
[groups]
developpeurs-calc = harry, sally, joe
```

```
developpeurs-paint = frank, sally, jane
tout-le-monde = @developpeurs-calc, @developpeurs-paint
```

Subversion 1.5 introduit une autre fonctionnalité utile pour la syntaxe du fichier des accès : les alias. Certains systèmes d'authentification attendent et utilisent des noms d'utilisateurs relativement courts tels que ceux que nous avons décrits ici — `harry`, `sally`, `joe`, etc. Mais d'autres systèmes d'authentification, comme par exemple ceux qui utilisent des bases LDAP ou des certificats clients SSL, peuvent utiliser des noms d'utilisateurs beaucoup plus complexes. Par exemple, le nom d'utilisateur d'Harry dans un système protégé par LDAP pourrait très bien être : `CN=Harold Hacker,OU=Engineers,DC=red-bean,DC=com`. Avec des noms d'utilisateurs de ce type, le fichier des accès devient rapidement illisible, avec des noms d'utilisateurs longs ou obscurs qui peuvent facilement être mal orthographiés. Heureusement, les alias vous permettent de n'avoir à taper le nom d'utilisateur complexe entier qu'une seule fois, au sein d'un paragraphe qui lui attribue un alias bien plus digeste.

```
[aliases]
harry = CN=Harold Hacker,OU=Engineers,DC=red-bean,DC=com
sally = CN=Sally Swatterbug,OU=Engineers,DC=red-bean,DC=com
joe = CN=Gerald I. Joseph,OU=Engineers,DC=red-bean,DC=com
…
```

Une fois défini votre ensemble d'alias, vous pouvez faire référence à ces utilisateurs en d'autres endroits du fichier par leurs alias, partout là où vous auriez sinon entré leur véritables noms d'utilisateurs. Il faut juste ajouter une esperluette (&) juste avant l'alias pour le distinguer des noms d'utilisateurs classiques :

```
[groups]
developpeurs-calc = &harry, &sally, &joe
developpeurs-paint = &frank, &sally, &jane
tout-le-monde = @developpeurs-calc, @developpeurs-paint
```

Vous pouvez aussi choisir d'utiliser des alias si les noms de vos utilisateurs changent souvent. Ainsi vous n'aurez que la table des alias à mettre à jour quand des modifications de noms d'utilisateurs auront lieu, au lieu d'avoir à effectuer des opérations de recherches-et-remplacements-globaux sur la totalité du fichier.

Accès partiel en lecture et extractions

Si vous utilisez Apache en tant que serveur Subversion et que vous avez rendu certains sous-répertoires de votre dépôt inaccessibles en lecture à certains utilisateurs, vous devez être au courant d'un comportement potentiellement non-optimal de la commande **svn checkout**.

Quand le client lance une requête de mise à jour ou d'extraction via HTTP, il envoie une requête unique au serveur et reçoit du serveur une réponse unique (dont la taille peut être assez importante). Quand le serveur reçoit la requête, c'est la *seule* opportunité dont dispose Apache pour demander à l'utilisateur de s'authentifier. Ceci a des effets secondaires assez étonnants. Par exemple, si un certain sous-répertoire du dépôt n'est accessible en lecture qu'à l'utilisateur Sally et qu'Harry extrait un répertoire parent, son client répondra à la demande d'authentification initiale en tant que Harry. Au fur et à mesure que le serveur génère la réponse, il n'a aucun moyen de renvoyer un défi d'authentification quand il atteint le sous-répertoire spécial ; ainsi le sous-répertoire tout entier est omis, plutôt que de demander à l'utilisateur de se ré-authentifier en tant que Sally le moment venu. De même, si la racine du dépôt est accessible en lecture anonymement, l'extraction se fera entièrement sans authentification, omettant, encore une fois, le répertoire non-lisible, plutôt que d'envoyer une demande d'authentification au cours de l'opération.

Accès au dépôt par plusieurs méthodes

Vous venez de voir différentes méthodes d'accès à un dépôt. Est-il possible — et sans danger — d'accéder simultanément à un dépôt par différentes méthodes ? La réponse est oui, à condition d'être un petit peu prévoyant.

À un instant donné, les processus suivants peuvent avoir besoin de l'accès en lecture ou en écriture au dépôt :

- des utilisateurs classiques du système se connectant à l'aide d'un client Subversion à des URL `file://` ;
- des utilisateurs classiques du système se connectant à des processus **svnserve** privés générés par SSH (dont le propriétaire est l'utilisateur lui-même) et accédant au dépôt ;
- un processus **svnserve** — soit un serveur autonome, soit un processus lancé par **inetd** — dont le propriétaire est un utilisateur dédié ;
- un processus **httpd** Apache, dont le propriétaire est un utilisateur dédié.

Les problèmes les plus courants rencontrés par les administrateurs sont des problèmes de droits et de propriété pour le dépôt. Chaque processus de la liste précédente a-t-il les droits de lecture et d'écriture sur les fichiers sous-jacents du dépôt ? En supposant que vous ayez un système d'exploitation de type Unix, une approche naïve de ce problème serait de placer chaque utilisateur potentiel du dépôt dans un groupe `svn` unique et de faire posséder le dépôt tout entier par ce groupe. Mais cela ne suffit même pas, car un processus risque de modifier les fichiers de la base de données en utilisant un umask inadapté qui va interdire l'accès aux autres utilisateurs.

L'étape suivante consiste donc, après avoir mis en place un groupe commun pour les utilisateurs du dépôt, à forcer tout processus qui accède au dépôt à utiliser un umask correct. Pour les utilisateurs qui accèdent directement au dépôt, vous pouvez « envelopper » le programme svnserve dans un script (*wrapper* en anglais) qui commence par lancer la commande **`umask 002`** et qui, seulement ensuite, appelle le véritable programme client **svn**. Vous pouvez également écrire un script similaire pour le programme **svnserve** et ajouter la commande **`umask 002`** au script de démarrage d'Apache, `apachectl`. Par exemple :

```
$ cat /usr/bin/svn

#!/bin/sh

umask 002
/usr/bin/le-vrai-svn "$@"
```

Sur les systèmes de type Unix, on rencontre souvent un autre problème classique. Si vous avez un dépôt Berkeley DB, par exemple, il crée de temps en temps de nouveaux fichiers pour la journalisation. Même si le dépôt Berkeley DB est entièrement possédé par le groupe `svn`, ces nouveaux journaux ne seront pas nécessairement possédés par le même groupe, ce qui crée des problèmes de droits supplémentaires pour vos utilisateurs. Une bonne façon de contourner ce problème est d'activer le bit SUID du groupe sur le répertoire `db` du dépôt, ce qui a pour résultat que tous les nouveaux fichiers journaux créés ont le même propriétaire que le répertoire parent.

Une fois ces manipulations effectuées, vos dépôts devraient être accessibles par tous les processus nécessaires. Tout ceci peut sembler un petit peu confus et compliqué, mais les problèmes d'accès en écriture par plusieurs utilisateurs à des fichiers partagés sont des problèmes très classiques, qui ne sont pas souvent résolus avec élégance.

Heureusement, la plupart des administrateurs *n'auront jamais besoin* d'une configuration aussi complexe. Les utilisateurs qui désirent accéder aux dépôts résidant sur une même machine ne sont pas limités aux URL d'accès `file://` — ils peuvent généralement contacter le serveur http Apache ou le serveur **svnserve** en utilisant `localhost` comme nom de serveur dans leurs URL `http://` ou `svn://`. Et assurer la maintenance de plusieurs processus serveurs pour vos dépôts Subversion vous créera plus de soucis qu'autre chose. Nous vous recommandons de choisir un seul serveur (celui qui correspond le mieux à vos besoins) et de vous y tenir !

Serveur `svn+ssh://` : les points à vérifier

Partager un dépôt entre des utilisateurs qui ont des comptes SSH sans avoir de problème de droits d'accès peut être assez épineux. Si l'ensemble des tâches à mener par l'administrateur d'un système de type Unix est encore un peu confus pour vous, voici la liste des choses à vérifier qui récapitule les points abordés dans cette section :

- Tous vos utilisateurs SSH doivent être capables de lire et d'écrire dans le dépôt, donc mettez tous les utilisateurs SSH dans un même groupe ;

- Faites de ce dépôt l'entière propriété de ce groupe.
- Mettez les droits d'accès de ce groupe à lecture/écriture.
- Vos utilisateurs doivent utiliser un umask correct quand ils accèdent au dépôt, donc assurez-vous que **svnserve** (`/usr/bin/svnserve` ou le chemin vers lequel `$PATH` pointe) est en fait un script qui exécute **`umask 002`** avant de lancer le véritable exécutable **svnserve**.
- Prenez des mesures similaires quand vous utilisez **svnlook** et **svnadmin** : soit vous les lancez avec un umask correct, soit vous les « enveloppez » dans un script comme nous venons de l'expliquer.

Chapitre 7. Personnalisation de Subversion

La gestion de versions est un sujet complexe, au moins autant un art qu'une science, qui offre une myriade de façons d'accomplir chaque tâche. En lisant ce livre, vous avez expérimenté les sous-commandes Subversion en ligne de commande et les options pour modifier leur comportement. Dans ce chapitre, nous passons en revue les moyens de personnaliser le fonctionnement de Subversion : la configuration des options d'exécution, l'utilisation d'applications externes pour faciliter certains traitements, les interactions de Subversion avec la configuration des paramètres régionaux du système d'exploitation, etc.

Zone de configuration des exécutables

Subversion permet à l'utilisateur de contrôler finement son comportement. Beaucoup d'options ont vocation à s'appliquer à l'ensemble des opérations de Subversion. Ainsi, plutôt que de forcer les utilisateurs à se souvenir d'arguments en ligne de commande pour spécifier ces options et de les utiliser à chaque invocation, Subversion utilise des fichiers de configuration, tenus à l'écart dans une zone de configuration spécifique à Subversion.

La *zone de configuration* Subversion consiste en une hiérarchie à deux niveaux constituée de noms d'options et de leurs valeurs. Habituellement, cela se traduit par un répertoire dédié qui contient les *fichiers de configuration* (le premier niveau) : des fichiers texte au format standard INI (dont les « sections » constituent le deuxième niveau). Vous pouvez facilement éditer ces fichiers à l'aide de votre éditeur de texte favori (tel qu'Emacs ou vi). Ils contiennent des directives lues par le client Subversion afin de déterminer le comportement par défaut choisi par l'utilisateur.

Agencement de la zone de configuration

La première fois que le client **svn** en ligne de commande est exécuté, il crée une zone de configuration propre à l'utilisateur. Sur les systèmes de type Unix, cette zone est un répertoire nommé `.subversion` dans le répertoire personnel de l'utilisateur. Sur les systèmes Windows, Subversion crée un dossier nommé `Subversion`, généralement dans la zone `Application Data` du répertoire qui contient le profil de l'utilisateur (qui est habituellement, au passage, un répertoire caché). Cependant, sur cette plate-forme, l'emplacement exact du profil utilisateur varie d'un système à l'autre et est dicté par la base de registre Windows [1]. Nous nous référerons à cette zone de configuration propre à l'utilisateur en utilisant son nom Unix : `.subversion`.

En plus de la zone de configuration propre à l'utilisateur, Subversion reconnaît l'existence d'une zone de configuration globale pour le système. Cela permet aux administrateurs du système d'établir une configuration par défaut pour l'ensemble des utilisateurs d'une machine donnée. Notez que la zone de configuration globale seule ne fixe pas de politique définitive : les réglages de l'utilisateur sont prioritaires par rapport aux réglages globaux et les options de la ligne de commande ont toujours le dernier mot. Sur les plate-formes de type Unix, la zone de configuration globale doit se trouver dans le répertoire `/etc/subversion` ; sur les machines Windows, Subversion cherche un répertoire `Subversion` dans le dossier commun `Application Data` (là encore, l'endroit exact dépend de la base de registre Windows). Au contraire de la zone propre à l'utilisateur, le programme **svn** ne tente pas de créer la zone de configuration globale.

La zone de configuration propre à l'utilisateur contient actuellement trois fichiers : deux fichiers de configuration (`config` et `servers`) et un fichier `README.txt` qui décrit le format INI. Lors de leur création, ces fichiers contiennent les valeurs par défaut de toutes les options supportées par Subversion, généralement mises en commentaire et groupées avec une description textuelle de l'effet de la clé sur le fonctionnement de Subversion. Pour modifier un comportement précis, il suffit de charger le fichier de configuration dans un éditeur de texte et de changer la valeur de l'option correspondante. Si, par la suite, vous voulez rétablir les valeurs par défaut, vous n'avez qu'à supprimer ou renommer votre répertoire de configuration puis lancer une commande **svn** inoffensive, telle que **`svn --version`**. Un nouveau répertoire de configuration sera créé, qui contiendra les valeurs par défaut.

La zone de configuration propre à l'utilisateur contient également un cache pour les données d'authentification. Le répertoire `auth` héberge un ensemble de sous-répertoires qui contiennent des informations mises en cache, relatives aux différentes méthodes d'authentification utilisées par Subversion. Ce répertoire est créé de telle manière que seul l'utilisateur ait accès à son contenu.

[1] La variable d'environnement `APPDATA` pointe vers la zone `Application Data`, vous pouvez donc toujours faire référence à ce dossier en utilisant `%APPDATA%\Subversion`.

Configuration via la base de registre Windows

En plus de la zone de configuration classique contenant les fichiers INI, les clients Subversion qui tournent sur une plate-forme Windows peuvent aussi utiliser la base de registre Windows pour stocker leurs données de configuration. Les noms des options et leurs valeurs sont les mêmes que dans les fichiers INI. La hiérarchie « fichier/section » est également présente, bien que traitée de manière légèrement différente : dans ce cas, les fichiers et les sections sont juste des niveaux dans l'arborescence des clés de registres.

Subversion cherche les valeurs de configuration applicables à tout le système sous la clé `HKEY_LOCAL_MACHINE\Software\Tigris.org\Subversion`. Par exemple, l'option `global-ignores`, qui se trouve dans la section `miscellany` du fichier de configuration, est située dans `HKEY_LOCAL_MACHINE\Software\Tigris.org\Subversion\Config\Miscellany\global-ignores`. Les valeurs propres à un utilisateur doivent être stockées sous `HKEY_CURRENT_USER\Software\Tigris.org\Subversion`.

Les options de configuration de la base de registre sont analysées *avant* les options des fichiers INI ; elles sont donc supplantées par les valeurs trouvées dans les fichiers de configuration. En d'autres termes, Subversion cherche les options de configuration dans l'ordre suivant sur un système Windows (les plus prioritaires sont citées en premier) :

1. les options en ligne de commande ;
2. les fichiers INI propres à l'utilisateur ;
3. les valeurs de la base de registre propres à l'utilisateur ;
4. les fichiers INI applicables à l'ensemble du système ;
5. les valeurs de la base de registre applicables à l'ensemble du système.

Par ailleurs, la base de registre Windows ne supporte pas vraiment la notion de « mise en commentaire ». Cependant, Subversion ignorera toute clé dont le nom commence par le caractère dièse (#). Cela vous permet de mettre en commentaire efficacement une option Subversion sans avoir à effacer entièrement la clé de la base de registre, ce qui simplifie manifestement la procédure de restauration de l'option.

Le client en ligne de commande **svn** n'écrit jamais dans la base de registre et ne tentera pas d'y créer une zone de configuration par défaut. Vous pouvez créer les clés dont vous avez besoin en utilisant le programme **REGEDIT**. Une autre alternative consiste à créer un fichier `.reg` (tel que celui donné dans l'Exemple 7.1, « Exemple de fichier de modification de la base de registre (.reg) ») puis à double-cliquer sur l'icône de ce fichier dans l'explorateur Windows afin d'appliquer les modifications à votre base de registre.

Exemple 7.1. Exemple de fichier de modification de la base de registre (.reg)

```
REGEDIT4

[HKEY_LOCAL_MACHINE\Software\Tigris.org\Subversion\Servers\groups]

[HKEY_LOCAL_MACHINE\Software\Tigris.org\Subversion\Servers\global]
"#http-proxy-host"=""
"#http-proxy-port"=""
"#http-proxy-username"=""
"#http-proxy-password"=""
"#http-proxy-exceptions"=""
"#http-timeout"="0"
"#http-compression"="yes"
"#neon-debug-mask"=""
"#ssl-authority-files"=""
"#ssl-trust-default-ca"=""
"#ssl-client-cert-file"=""
"#ssl-client-cert-password"=""
```

```
[HKEY_CURRENT_USER\Software\Tigris.org\Subversion\Config\auth]
"#store-passwords"="yes"
"#store-auth-creds"="yes"

[HKEY_CURRENT_USER\Software\Tigris.org\Subversion\Config\helpers]
"#editor-cmd"="notepad"
"#diff-cmd"=""
"#diff3-cmd"=""
"#diff3-has-program-arg"=""

[HKEY_CURRENT_USER\Software\Tigris.org\Subversion\Config\tunnels]

[HKEY_CURRENT_USER\Software\Tigris.org\Subversion\Config\miscellany]
"#global-ignores"="*.o *.lo *.la #*# .*.rej *.rej .*~ *~ .#* .DS_Store"
"#log-encoding"=""
"#use-commit-times"=""
"#no-unlock"=""
"#enable-auto-props"=""

[HKEY_CURRENT_USER\Software\Tigris.org\Subversion\Config\auto-props]
```

L'exemple précédent présente le contenu d'un fichier `.reg` qui contient quelques unes des options les plus communément utilisées et leurs valeurs par défaut. Notez la présence de réglages propres à l'utilisateur (notamment l'éditeur de texte et le stockage des mots de passe) ainsi que de réglages applicables à l'ensemble du système (comme les options relatives au mandataire réseau). Notez également que toutes les options sont mises en commentaire. Il ne vous reste qu'à supprimer le caractère dièse (#) initial des noms d'options et à leur affecter la valeur que vous souhaitez.

Options de configuration

Dans cette section, nous allons nous intéresser aux options de configuration des programmes supportées par la version actuelle de Subversion.

Fichier `servers`

Le fichier `servers` contient les options de configuration relatives aux couches réseau. Il y a deux sections spéciales dans ce fichier : `groups` et `global`. La section `groups` n'est rien d'autre qu'un tableau de références croisées. Les clés de cette section sont les noms des autres sections du fichier ; ses valeurs sont des *globs* (des représentations textuelles qui peuvent contenir des caractères joker) qui sont comparés aux noms des machines auxquelles des requêtes Subversion sont envoyées.

```
[groups]
serveurs-red-beans = *.red-bean.com
collabnet = svn.collab.net

[serveurs-red-beans]
…

[collabnet]
…
```

Quand vous utilisez Subversion en réseau, il essaie de faire correspondre le nom du serveur auquel il tente de se connecter avec un nom de groupe de la section `groups`. Si une correspondance existe, Subversion vérifie alors s'il existe dans le fichier `servers` une section dont le nom est le nom du groupe correspondant. Le cas échéant, il tire de cette section la configuration réseau à appliquer.

La section `global` contient la configuration à appliquer à tous les serveurs qui ne correspondent à aucun glob de la section `groups`. Les options disponibles dans cette section sont exactement les mêmes que pour les autres sections du fichier (exceptée bien sûr la section spéciale `groups`) et sont :

`http-proxy-exceptions`
: Cette option contient une liste de motifs (séparés par des virgules) de noms de machines qui doivent être contactées

directement, sans passer par le mandataire. La syntaxe des motifs est la même que celle utilisée par le shell Unix pour les noms de fichiers. L'accès aux dépôts d'une machine dont le nom correspond à l'un de ces motifs se fera sans passer par un mandataire.

`http-proxy-host`
Cette option contient le nom de la machine mandataire pour les requêtes HTTP de Subversion. La valeur par défaut est vide, ce qui signifie que Subversion ne tentera pas de faire passer ses requêtes par un mandataire mais essaiera de contacter la machine de destination directement.

`http-proxy-port`
Cette option contient le numéro du port à utiliser sur la machine mandataire. Par défaut, la valeur est vide.

`http-proxy-username`
Cette option contient le nom d'utilisateur à fournir à la machine mandataire. Par défaut, la valeur est vide.

`http-proxy-password`
cette option contient le mot de passe à fournir à la machine mandataire. Par défaut, la valeur est vide.

`http-timeout`
Cette option contient la durée maximum, en secondes, pendant laquelle Subversion attend la réponse du serveur. Si vous rencontrez des problèmes d'opérations Subversion qui expirent à cause d'une connexion réseau trop lente, vous devez augmenter cette valeur. Par défaut, la valeur est `0`, ce qui conduit la bibliothèque HTTP sous-jacente (Neon) à utiliser sa propre valeur par défaut.

`http-compression`
Cette option indique si oui ou non Subversion doit essayer de compresser les requêtes réseaux à destination de serveurs DAV. La valeur par défaut est `yes`. Notez que la compression ne sera effective que si la couche réseau a été compilée avec le support de la compression. Mettez `no` pour ne pas activer la compression, par exemple lorsque vous analysez les transmissions réseaux.

`http-library`
Subversion est fourni avec deux modules d'accès aux dépôts qui utilisent le protocole réseau WebDAV. Le module originel, fourni avec Subversion 1.0, est `libsvn_ra_neon` (bien qu'en ce temps-là, il s'appelait `libsvn_ra_dav`). Les nouvelles versions de Subversion fournissent également `libsvn_ra_serf`, qui utilise une implémentation sous-jacente différente et qui vise à supporter certains des concepts HTTP les plus récents.

Actuellement, `libsvn_ra_serf` est toujours considérée en version expérimentale, bien qu'elle semble fonctionner correctement dans les cas usuels. Afin d'inciter les gens à l'essayer, Subversion fournit l'option de configuration `http-library` pour permettre aux utilisateurs de définir (globalement ou par groupe de serveurs) quel module d'accès WebDAV ils veulent utiliser : `neon` ou `serf`.

`http-auth-types`
Cette option liste les méthodes d'authentification (séparées par des points-virgules) supportées par les module d'accès aux dépôts WebDAV basés sur Neon. Les valeurs valides sont : `basic`, `digest` et `negotiate`.

`neon-debug-mask`
Cette option contient un entier qui représente un masque que la bibliothèque HTTP sous-jacente (Neon) utilise pour choisir quel type d'affichage de débogage autoriser. La valeur par défaut est `0`, ce qui interdit tout affichage de débogage. Pour plus d'informations sur l'utilisation de Neon par Subversion, reportez-vous au Chapitre 8, *Intégration de Subversion*.

`ssl-authority-files`
Cette option contient une liste de chemins (séparés par des points-virgules) vers les fichiers contenant les certificats des autorités de certifications (AC) qui doivent être reconnues comme de confiance par le client Subversion lors des accès aux dépôts en HTTPS.

`ssl-trust-default-ca`
Mettez cette variable à `yes` si vous voulez que Subversion fasse automatiquement confiance à l'ensemble des autorités de certification livrées par défaut avec OpenSSL.

`ssl-client-cert-file`
Si un hôte (ou un ensemble d'hôtes) demande un certificat SSL au client, vous serez sollicité pour fournir le chemin de votre certificat. Dès que cette variable contient ce chemin, Subversion sera capable de trouver automatiquement votre certificat et ne vous sollicitera pas. Il n'existe pas d'emplacement standard pour stocker un certificat utilisateur sur le disque ; Subversion va le chercher à l'endroit que vous lui spécifiez.

`ssl-client-cert-password`
Si votre certificat client SSL est protégé par une phrase de passe, Subversion vous la demandera à chaque fois que le certificat est utilisé. Si vous trouvez cela pénible (et que cela ne vous dérange pas que cette phrase de passe soit stockée dans le fichier `servers`), vous pouvez placer dans cette variable la phrase de passe de votre certificat. Vous ne serez plus sollicité.

Fichier `config`

Le fichier `config` contient le reste des options du programme Subversion disponibles actuellement, c'est-à-dire celles qui ne se rapportent pas au réseau. Au moment où ces lignes sont écrites, seules quelques options sont utilisées, mais elles sont quand même regroupées en sections en prévision d'ajouts futurs.

La section `auth` contient les paramètres relatifs à l'authentification et au contrôle d'accès de Subversion pour les dépôts. Elle comprend les options suivantes :

`store-passwords`
Cette option demande à Subversion de garder en cache (ou non) les mots de passe qui sont tapés par l'utilisateur en réponse aux demandes d'authentification des serveurs. La valeur par défaut est `yes`. Remplacez cette valeur par `no` pour désactiver la mise en cache sur le disque. Vous pouvez outrepasser cette option pour un appel donné d'une commande **svn** en utilisant l'option de ligne de commande `--no-auth-cache` (pour les sous-commandes qui acceptent cette option). Pour plus d'informations, consultez la section intitulée « Mise en cache des éléments d'authentification du client ».

`store-auth-creds`
cette option est équivalente à `store-passwords` sauf qu'elle applique la mise en cache sur le disque (ou non) à *l'ensemble* des informations d'authentification : identifiants, mots de passe, certificats serveur et tout autre type d'élément pouvant être mis en cache.

La section `helpers` définit quelles sont les applications externes utilisées par Subversion pour accomplir ses tâches. Les options valides dans cette section sont :

`editor-cmd`
Cette option indique le programme utilisé par l'utilisateur auquel Subversion demande d'entrer des méta-données textuelles ou de résoudre les conflits de manière interactive. Consultez la section intitulée « Utilisation d'éditeurs externes » pour plus de détails sur l'utilisation d'un éditeur de texte externe avec Subversion.

`diff-cmd`
Cette option contient le chemin absolu du programme de comparaison qui est utilisé lorsque Subversion doit afficher à l'utilisateur plusieurs fichiers à comparer (par exemple lors de l'utilisation de la commande **svn diff**). Par défaut, Subversion utilise une bibliothèque interne de comparaison. Définir cette option le forcera à utiliser un programme externe pour effectuer cette tâche. Consultez la section intitulée « Utilisation des outils externes de comparaison et de fusion » pour plus de détails sur l'utilisation de tels programmes.

`diff3-cmd`
Cette option contient le chemin absolu d'un programme de comparaison à trois entrées. Subversion utilise ce programme pour fusionner les changements effectués par l'utilisateur avec ceux en provenance du dépôt. Par défaut, Subversion utilise une bibliothèque interne de comparaison. Définir cette option le forcera à utiliser un programme externe pour effectuer cette tâche. Consultez la section intitulée « Utilisation des outils externes de comparaison et de fusion » pour plus de détails sur l'utilisation de tels programmes.

`diff3-has-program-arg`
Ce drapeau doit être mis à `true` si le programme spécifié par l'option `diff3-cmd` accepte l'option `--diff-program` en ligne de commande.

`merge-tool-cmd`
Cette option contient le programme que Subversion utilise pour effectuer les opérations de fusion à trois sources sur vos fichiers suivis en versions. Consultez la section intitulée « Utilisation des outils externes de comparaison et de fusion » pour plus de détails sur l'utilisation de tels programmes.

La section `tunnels` vous permet de définir de nouveaux tunnels à utiliser avec les connexions clientes **svnserve** et `svn://`. Pour plus de détails, consultez la section intitulée « Encapsulation de svnserve dans un tunnel SSH ».

La section `miscellany` récolte tout ce qui n'a pas sa place ailleurs [2] Dans cette section, vous trouvez :

`global-ignores`
: Quand vous exécutez la commande **svn status**, Subversion affiche la liste des fichiers et répertoires non suivis en versions avec ceux qui sont suivis en versions, en les marquant d'un caractère `?` (voir la section intitulée « Avoir une vue d'ensemble des changements effectués »). Parfois, ces éléments inutiles et non suivis en version ne font que rendre l'affichage plus confus, par exemple dans le cas des fichiers objets générés par les compilations. L'option `global-ignores` contient une liste de globs séparés par des espaces qui décrivent les noms de fichiers et de répertoires que Subversion ne doit pas afficher, sauf s'ils sont suivis en versions. La valeur par défaut est `*.o *.lo *.la #*# .*.rej *.rej .*~ *~ .#* .DS_Store`.

  Tout comme **svn status**, les commandes **svn add** et **svn import** ignorent les fichiers qui entrent en correspondance avec la liste lors du parcours d'un répertoire. Vous pouvez bloquer ce comportement pour une instance donnée de ces commandes en spécifiant explicitement le nom de fichier ou en utilisant l'option `--no-ignore` en ligne de commande.

  Vous pouvez trouvez des informations pour contrôler plus finement les éléments ignorés dans la section intitulée « Occultation des éléments non suivis en versions ».

`enable-auto-props`
: Cette option demande à Subversion d'ajouter automatiquement des propriétés sur les fichiers nouvellement ajoutés ou importés. La valeur par défaut est `no`, vous devez donc lui affecter la valeur `yes` pour activer cette fonctionnalité. La section `auto-props` de ce fichier spécifie quelles propriétés doivent être définies sur quels fichiers.

`log-encoding`
: Cette option définit la valeur par défaut du codage des caractères des messages de propagation. C'est le pendant permanent de l'option `--encoding` (voir la section intitulée « Options de svn »). Le dépôt Subversion stocke les messages de propagation en UTF-8 et suppose que votre message est écrit en utilisant le codage défini par votre système d'exploitation. Vous devez spécifier un codage différent si vos messages de propagation sont rédigés avec un autre codage.

`use-commit-times`
: Normalement, les fichiers de votre copie de travail sont datés de manière à indiquer la dernière fois qu'ils ont été manipulés par n'importe quel processus, que ce soit votre éditeur de texte ou une sous-commande **svn**. C'est le comportement attendu généralement par les développeurs de logiciels, puisque les systèmes de compilation examinent souvent ces dates pour décider quels fichiers doivent être recompilés.

  Dans d'autres situations, cependant, il est préférable d'avoir une date qui indique la dernière fois que le fichier a été modifié dans le dépôt. La commande **svn export** marque « la date de la dernière propagation » sur les arborescences qu'elle produit. En mettant cette variable de configuration à `yes`, les commandes **svn checkout**, **svn update**, **svn switch** et **svn revert** marqueront également les fichiers qu'elles modifient avec la date de dernière propagation.

`mime-types-file`
: Cette option est apparue dans Subversion 1.5. Elle spécifie le chemin d'un fichier de correspondance pour les types MIME, de la même manière que le fichier `mime.types` fourni par le serveur HTTP Apache. Subversion utilise ce fichier pour associer des types MIME aux nouveaux fichiers ajoutés ou importés. Consultez la section intitulée « Configuration automatique des propriétés » et la section intitulée « Type de contenu des fichiers » pour plus d'informations sur la détection et l'utilisation des types de fichiers par Subversion.

`preserved-conflict-file-exts`
: La valeur de cette option est une liste d'extensions de fichiers (séparées par des espaces) que Subversion doit préserver quand il génère des noms de fichiers lors des conflits. Par défaut, cette liste est vide. C'est une nouvelle option apparue dans Subversion 1.5.

  Quand Subversion détecte des conflits dans les modifications effectuées sur un fichier, il soumet la résolution de ces conflits à l'utilisateur. Pour aider l'utilisateur à les résoudre, Subversion garde une copie originale des différentes versions en lice du fichier dans la copie de travail. Par défaut, ces fichiers ont des noms construits en ajoutant au nom de fichier original une extension particulière telle que `.mine` ou `.REV` (où *REV* est un numéro de révision). Ceci peut être gênant sur les systèmes d'exploitation qui utilisent les extensions de noms de fichiers pour déterminer l'application par défaut à

[2] En clair, c'est un joyeux fourre-tout.

utiliser pour ouvrir et éditer le fichier, le fichier avec la nouvelle extension n'étant plus automatiquement ouvert dans l'application prévue. Par exemple, si le fichier `NotesDeVersion.pdf` est en conflit, les fichiers générés risquent de s'appeler `NotesDeVersion.pdf.mine` ou `NotesDeVersion.pdf.r4231`. Bien que votre système soit peut-être configuré pour ouvrir les fichiers `.pdf` ave Acrobat Reader, il n'existe sûrement pas d'application préconfigurée pour ouvrir les fichiers avec l'extension `.r4231`.

Vous pouvez arranger cela en utilisant cette option de configuration. Pour les fichiers dont l'extension est spécifiée, Subversion ajoutera au nom des fichiers générés l'extension particulière liée au conflit, mais il rajoutera aussi à la suite l'extension originale. Dans l'exemple précédent, en supposant que `pdf` est une des extensions configurée dans la liste ci-dessus, les noms des fichiers générés pour `NotesDeVersion.pdf` vont être `NotesDeVersion.pdf.mine.pdf` et `NotesDeVersion.pdf.r4231.pdf`. Comme chaque fichier se termine par `.pdf`, l'application appropriée sera utilisée pour les visualiser.

`interactive-conflicts`
: Cette option est une option booléenne qui indique si Subversion doit essayer de résoudre les conflits de manière interactive. Si la valeur est `yes` (qui est la valeur par défaut), Subversion demandera à l'utilisateur comment gérer les conflits, comme indiqué dans la section intitulée « Résoudre les conflits (fusionner des modifications) ». Autrement, il marquera simplement qu'il existe un conflit et continuera l'opération en cours, remettant sa résolution à plus tard.

`no-unlock`
: Cette option booléenne correspond à l'option `--no-unlock` de **svn commit**. Elle indique à Subversion de ne pas libérer les verrous posés sur les fichiers que vous venez de propager. Si cette option est positionnée à `yes`, Subversion ne libérera jamais aucun verrou automatiquement, vous laissant le faire explicitement avec **svn unlock**. La valeur par défaut est `no`.

La section `auto-props` contrôle la possibilité par le client Subversion de positionner automatiquement certaines propriétés sur les fichiers qui sont ajoutés ou importés. Elle contient un nombre arbitraire de paires clé-valeur au format `MOTIF = NOM_PROPRIETE=VALEUR[;NOM_PROPRIETE=VALEUR ...]`, où `MOTIF` est un motif de nom de fichier qui correspond à un ou plusieurs noms de fichiers et le reste de la ligne est une liste d'affectations (séparées par des points-virgules) de valeurs à des propriétés. Si un nom de fichier correspond à plusieurs motifs, autant de propriétés seront positionnées ; cependant, il n'y a aucune garantie que les auto-props seront appliquées dans l'ordre dans lequel elles apparaissent dans le fichier `config` ; il ne faut donc pas définir de règles susceptibles d'en écraser d'autres. Vous pouvez trouver de nombreux exemples d'utilisation d'auto-props dans le fichier `config`. Enfin, n'oubliez pas de positionner `enable-auto-props` à `yes` dans la section `miscellany` si vous voulez activer auto-props.

Localisation

La *localisation* (aussi appelée *régionalisation*) consiste à modifier le comportement d'un programme pour qu'il agisse d'une manière propre à une région. Quand un programme formate les nombres et les dates d'une façon particulière pour votre région du monde ou quand il affiche des messages (ou accepte des entrées) dans votre langue maternelle, le programme est dit *localisé*. Cette section décrit les étapes qu'a franchi Subversion pour être localisable.

Généralités sur la localisation

La plupart des systèmes d'exploitation modernes intègrent la notion de « paramètres régionaux courants » (*current locale* en anglais), c'est-à-dire la région ou le pays dont les conventions sont appliquées. Ces conventions, généralement choisies par un mécanisme de configuration pour la durée du fonctionnement d'un programme sur l'ordinateur, affectent la façon dont les données sont présentées à l'utilisateur ainsi que la façon dont les entrées de l'utilisateur sont traitées.

Sur la majorité des systèmes de type Unix, vous pouvez vérifier la valeur des paramètres régionaux en cours en lançant la commande **locale** :

```
$ locale
LANG=
LC_COLLATE="C"
LC_CTYPE="C"
LC_MESSAGES="C"
LC_MONETARY="C"
LC_NUMERIC="C"
LC_TIME="C"
LC_ALL="C"
$
```

L'affichage comprend une liste de variables d'environnement relatives à des conventions locales et leur valeurs. Dans cet exemple, toutes les variables possèdent la valeur par défaut `C`, mais les utilisateurs peuvent modifier ces valeurs et leur affecter des valeurs propres à leur pays ou à leur langue. Par exemple, si quelqu'un fixe la valeur de la variable `LC_TIME` à `fr_CA`, les programmes sauront qu'il doivent afficher les dates et les heures conformément aux attentes des canadiens francophones. Et si quelqu'un fixe la valeur de la variable `LC_MESSAGES` à `zh_TW`, les programmes sauront qu'il doivent afficher les messages à destination de l'utilisateur en chinois traditionnel. Modifier la variable `LC_ALL` équivaut à donner à toutes les variables de paramètres régionaux la valeur choisie pour `LC_ALL`. La valeur de la variable `LANG` est utilisée par défaut pour toute variable de paramètre régional qui n'a pas de valeur attribuée. Pour voir la liste de toutes les variables de paramètres régionaux sur un système Unix, lancez la commande **`locale -a`**.

Sous Windows, la configuration des paramètres régionaux s'effectue par l'intermédiaire de l'élément « Options régionales, date, heure et langue » du panneau de configuration. Vous pouvez y voir et y sélectionner les valeurs de chacune des variables disponibles, voire personnaliser les conventions d'affichage de nombreux paramètres (à un niveau de détail presque maladif).

Utilisation des paramètres régionaux par Subversion

Le client Subversion, **svn**, utilise la configuration courante des paramètres régionaux de deux manières. D'abord, il prend en compte la valeur de la variable `LC_MESSAGES` et essaie d'afficher tous les messages dans la langue indiquée. Par exemple :

```
$ export LC_MESSAGES=de_DE
$ svn help cat
cat: Gibt den Inhalt der angegebenen Dateien oder URLs aus.
Aufruf: cat ZIEL[@REV]...
…
```

Ce comportement est identique sur les systèmes Unix et Windows. Notez cependant que, bien que votre système d'exploitation puisse supporter certaines valeurs de paramètres régionaux, Subversion ne parle peut-être pas toutes ces langues. Afin d'afficher ces messages localisés, des volontaires (humains) doivent fournir des traductions dans chaque langue. Les traductions sont écrites en utilisant le paquetage GNU gettext, ce qui produit des modules de traduction dont l'extension du nom de fichier est `.mo`. Par exemple, le fichier des traductions allemandes s'appelle `de.mo`. Ces fichiers de traductions sont installés quelque part sur votre système. Sous Unix, ils sont généralement placés dans `/usr/share/locale/`, alors que sous Windows on les trouve souvent dans le dossier `\share\locale\` de la zone d'installation de Subversion. Une fois installé, un module est renommé d'après le programme pour lequel il fournit des traductions. Par exemple, le fichier `de.mo` sera peut-être finalement installé en tant que `/usr/share/locale/de/LC_MESSAGES/subversion.mo`. En parcourant les fichiers `.mo` installés, vous pouvez voir quelles langues le client Subversion parle.

La prise en compte des paramètres régionaux se fait aussi au niveau de la façon dont **svn** interprète vos entrées. Le dépôt stocke tous les chemins, noms de fichiers et messages de propagation en Unicode, plus exactement en UTF-8. En ce sens, le dépôt est *internationalé*, c'est-à-dire que le dépôt peut accepter des entrées en n'importe quelle langue. Cela signifie cependant que le client Subversion ne doit envoyer vers le dépôt que des noms de fichiers et des messages de propagation en UTF-8. Pour ce faire, il doit convertir les données depuis les paramètres régionaux courants vers l'UTF-8.

Par exemple, supposons que vous créez un fichier nommé `caffè.txt` et qu'ensuite, lorsque vous propagez ce fichier, vous fournissez le message de propagation suivant : « Adesso il caffè è più forte ». Le nom du fichier et le message de propagation contiennent tous deux des caractères non-ASCII, mais puisque vos paramètres régionaux sont `it_IT`, le client Subversion sait qu'il doit les interpréter comme de l'italien. Il utilise alors un jeu de caractères italiens pour convertir ces données en UTF-8 avant de les envoyer au dépôt.

Remarquez que bien que le dépôt exige des noms de fichiers et des messages de propagation au format UTF-8, il *ne s'intéresse pas* au contenu du fichier. Subversion traite le contenu des fichiers comme des chaînes d'octets « opaques » ; ni le client ni le serveur ne tentent de comprendre le jeu de caractères ou le codage du contenu d'un fichier.

Erreurs de conversion des jeux de caractères

Lors de l'utilisation de Subversion, vous êtes susceptible d'être confronté à des erreurs de conversion des jeux de caractères :

```
svn: Impossible de convertir la chaîne de l'encodage natif vers 'UTF-8' :
…
svn: Impossible de convertir la chaîne de 'UTF-8' vers l'encodage natif :
…
```

De telles erreurs surviennent généralement quand un client Subversion reçoit une chaîne UTF-8 du dépôt et que certains caractères de cette chaîne ne peuvent pas être représentés dans le jeu de caractères local. Par exemple, si vos paramètres régionaux sont `en_US` et qu'un collaborateur a propagé un nom de fichier en japonais, alors il y a de grandes chances que cette erreur apparaisse lors de la réception du fichier pendant l'exécution de **svn update**.

La solution consiste soit à configurer vos paramètres régionaux de manière à *pouvoir* traiter n'importe quelles données au format UTF-8, soit à modifier le nom de fichier ou le message de propagation dans le dépôt (et n'oubliez pas de taper sur les doigts de votre collaborateur : les projets doivent décider en amont des langues à utiliser, de manière à ce que l'ensemble des collaborateurs utilisent les mêmes paramètres régionaux).

Utilisation d'éditeurs externes

La manière la plus évidente d'entrer des données dans Subversion consiste à ajouter des fichiers sous gestion de versions, propager des changements sur ces fichiers, etc. Mais d'autres informations existent dans le dépôt Subversion à côté des données constituées par les simples fichiers suivis en versions. Certaines de ces informations — les messages de propagation, les commentaires sur les verrous et les valeurs de certaines propriétés — ont naturellement tendance à être textuelles et sont explicitement fournies par les utilisateurs. La plupart de ces informations peuvent être fournies à Subversion à l'aide du client en ligne de commande en utilisant les options `--message` (`-m`) et `--file` (`-F`) dans le cadre des sous-commandes appropriées.

Chacune de ces options a des avantages et des inconvénients. Par exemple, quand vous effectuez une propagation, l'option `--file` (`-F`) fonctionne bien si vous avez déjà préparé un fichier texte qui contient votre message de propagation. Si vous ne l'avez pas fait, vous pouvez toujours utiliser l'option `--message` (`-m`) pour stipuler votre message en ligne de commande. Malheureusement, composer un message de plus d'une ligne en ligne de commande peut être assez délicat. Les utilisateurs veulent plus de flexibilité : ils veulent pouvoir écrire un message de propagation multi-ligne, sans contrainte de format et à la demande.

Subversion répond à cette attente en vous permettant de spécifier un éditeur de texte externe qui sera lancé au besoin, vous offrant ainsi un moyen plus puissant pour entrer des méta-données textuelles. Il y a plusieurs manières d'indiquer à Subversion quel éditeur vous voulez utiliser. Subversion vérifie les choses suivantes, dans l'ordre spécifié, avant de lancer un tel éditeur :

1. l'option en ligne de commande `--editor-cmd`
2. la variable d'environnement `SVN_EDITOR`
3. l'option de configuration `editor-cmd`
4. la variable d'environnement `VISUAL`
5. la variable d'environnement `EDITOR`
6. éventuellement, une valeur par défaut spécifiée dans une des bibliothèques de Subversion (non présente dans les distributions officielles)

La valeur de n'importe laquelle de ces options ou variables est le début de la ligne de commande qui sera exécutée par le shell. Subversion ajoute à cette ligne de commande un espace et le chemin vers un fichier temporaire à éditer. Ainsi, pour être utilisé avec Subversion, l'éditeur spécifié ou configuré doit pouvoir être appelé avec pour dernier argument de sa ligne de commande le fichier à éditer, il doit être capable de sauvegarder le fichier au même endroit et il doit retourner zéro comme code de retour, pour indiquer la réussite de l'opération.

Comme indiqué, les éditeurs externes peuvent être utilisés pour fournir les messages de propagation à n'importe quelle sous-commande de propagation (telle que **svn commit** ou **svn import**, **svn mkdir** ou **svn delete** quand vous fournissez une URL cible, etc.) et Subversion essaie de lancer l'éditeur automatiquement si vous ne spécifiez ni l'option `--message` (`-m`) ni

l'option `--file` (`-F`). La commande **svn propedit** est presque entièrement construite autour de l'utilisation d'un éditeur de texte externe. À partir de la version 1.5, Subversion utilise également l'éditeur de texte externe configuré quand l'utilisateur demande le lancement d'un éditeur lors de la résolution interactive de conflits. Bizarrement, il ne semble pas y avoir de moyen d'utiliser un éditeur externe pour fournir un commentaire de verrouillage de manière interactive.

Utilisation des outils externes de comparaison et de fusion

L'interface entre Subversion et les outils de comparaison (de deux ou trois fichiers) remonte à l'époque où Subversion, pour afficher les différences de manière contextuelle, utilisait directement la suite d'outils GNU de comparaison, plus précisément les utilitaires **diff** et **diff3**. Pour obtenir le comportement désiré par Subversion, l'appel à ces programmes se faisait avec une ribambelle d'options et de paramètres dont la plupart étaient spécifiques à ces utilitaires. Plus tard, Subversion s'est doté de sa propre bibliothèque de comparaison et, afin de conserver la possibilité de choisir, les options `--diff-cmd` et `--diff3-cmd` ont été ajoutées au client en ligne de commande pour que les utilisateurs allergiques à la nouvelle bibliothèque puissent indiquer facilement leur souhait de se servir des utilitaires GNU diff et diff3. Si ces options étaient utilisées, Subversion ignorait purement et simplement sa bibliothèque interne de comparaison et appelait alors ces programmes externes, avec leurs listes d'arguments longue comme un jour sans pain et leurs autres spécificités. Les choses sont restées en l'état.

Il n'a pas fallu longtemps pour que les adeptes de Subversion comprennent que si l'on pouvait utiliser les utilitaires GNU **diff** et **diff3** situés à un endroit précis du disque, on pouvait alors aussi utiliser ce mécanisme pour utiliser d'autres outils de comparaison. Après tout, Subversion ne vérifiait pas que les programmes qu'il lançait faisaient bien partie de la suite des outils GNU de comparaison. Mais la seule chose que l'on peut configurer dans l'utilisation de programmes externes est leur emplacement sur le disque ; pas les options, ni l'ordre des paramètres, ni le reste. Subversion continue à envoyer toutes les options des utilitaires GNU à votre programme de comparaison, indépendamment du fait que votre programme en tienne compte ou non. Et c'est là que la plupart des utilisateurs ne comprennent pas la logique de la chose.

La clé pour utiliser des outils externes de comparaison de deux ou trois fichiers (autres que les programmes GNU diff et diff3 bien sûr) avec Subversion est de créer des scripts d'interface qui convertissent les données fournies par Subversion en données compréhensibles par l'outil de comparaison que vous utilisez, puis de convertir les sorties de votre outil vers le format attendu par Subversion : le format que les outils GNU auraient utilisé. Les sections suivantes détaillent les spécificités de ce qu'attend Subversion.

La décision de lancer une comparaison entre deux ou trois fichiers dans le cadre d'une opération Subversion est entièrement du ressort de Subversion et est conditionnée, en autres, au fait que les fichiers soient lisibles par un humain comme indiqué par leur propriété `svn:mime-type`. Cela signifie, par exemple, que même si vous avez le plus formidable outil de l'univers, capable de comparer et fusionner des documents Microsoft Word, il ne sera jamais appelé par Subversion tant que le type MIME des documents Word suivis en versions indiquera qu'ils ne sont pas lisibles par un humain (tel que `application/msword`). Pour plus d'informations sur la configuration des types MIME, voir la section intitulée « Type de contenu des fichiers »

Subversion 1.5 introduit la résolution interactive des conflits (décrite dans la section intitulée « Résoudre les conflits (fusionner des modifications) ») et l'une des options proposées à l'utilisateur est de lancer un outil de fusion externe. Si cette action est choisie, Subversion consultera l'option `merge-tool-cmd` de la zone de configuration des exécutables, susceptible de contenir le nom d'un outil externe de fusion et, s'il en trouve un, lancera cet outil avec les fichiers appropriés en paramètres. Il y a deux différences notables avec l'outil de comparaison de trois fichiers. D'abord, l'outil de comparaison est toujours utilisé pour afficher les différences entre trois fichiers alors que l'outil de fusion est employé uniquement quand l'application de comparaison a détecté un conflit. Ensuite, l'interface est beaucoup plus propre — votre outil de fusion a seulement besoin d'accepter comme paramètres en ligne de commande les chemins des quatre fichiers suivants : le fichier de référence, le « leur » (qui contient les modifications en provenance du dépôt), le « mien » (qui contient les modifications locales) et le fichier dans lequel sera placé le contenu final à stocker.

Programme externe de comparaison

Subversion appelle les programmes externes de comparaison avec des paramètres compatibles avec la suite d'outils de comparaison GNU et s'attend à ce que le programme renvoie un code d'erreur signifiant la réussite de la comparaison. Pour la plupart des programmes de comparaison alternatifs, seuls les sixième et septième arguments, indiquant les chemins vers les fichiers à comparer, respectivement placés à gauche et à droite, sont intéressants. Notez que Subversion lance le programme de comparaison pour chaque fichier concerné par l'opération Subversion, ce qui peut vous amener à avoir plusieurs instances simultanément si votre programme fonctionne de manière asynchrone (ou s'il est placé en arrière-plan). Enfin, Subversion

s'attend à ce que votre programme retourne un code d'erreur de 1 s'il a détecté des différences ou 0 sinon. Tout autre code d'erreur est considéré comme une erreur fatale [3].

L'Exemple 7.2, « interface-diff.py » et l' Exemple 7.3, « interface-diff.bat » sont des modèles de scripts d'interface pour un outil externe de comparaison, respectivement en langage Python et en langage de script Windows :

Exemple 7.2. interface-diff.py

```
#!/usr/bin/env python
import sys
import os

# Indiquez ici le chemin de votre outil de comparaison favori.
DIFF = "/usr/local/bin/mon-diff-perso"

# Subversion fournit les chemins voulus dans les deux derniers arguments.
GAUCHE = sys.argv[-2]
DROITE = sys.argv[-1]

# Appelons la commande de comparaison (modifiez la ligne suivante
# en accord avec votre programme de comparaison).
cmd = [DIFF, '--left', GAUCHE, '--right', DROITE]
os.execv(cmd[0], cmd)

# Le code d'erreur renvoyé doit être :
# 0 si aucune différence n'est détectée,
# 1 s'il y a des différences.
# Tout autre code est traité comme une erreur fatale.
```

Exemple 7.3. interface-diff.bat

```
@ECHO OFF

REM Indiquez ici le chemin de votre outil de comparaison favori.
SET DIFF="C:\Program Files\Super Progs\Mon Diff Perso.exe"

REM Subversion fournit les chemins voulus dans les deux derniers arguments.
REM Ce sont les paramètres 6 et 7 (sauf si vous utilisez svn diff -x,
REM auquel cas tout est possible).
SET GAUCHE=%6
SET DROITE=%7

REM Appelons la commande de comparaison (modifiez la ligne suivante
REM en accord avec votre programme de comparaison).
%DIFF% --left %GAUCHE% --right %DROITE%

REM Le code d'erreur renvoyé doit être :
REM 0 si aucune différence n'est détectée,
REM 1 s'il y a des différences.
REM Tout autre code est traité comme une erreur fatale.
```

Programme externe de comparaison de trois fichiers

Subversion appelle les programmes externes de fusion avec des paramètres compatibles avec l'outil GNU diff3 et s'attend à ce

[3] Le manuel du diff GNU indique : « Un code de retour valant 0 signifie qu'aucune différence n'a été trouvée, 1 signifie que des différences sont apparues, 2 indique une erreur ».

que le programme externe retourne un code d'erreur correct et que le fichier résultant de l'opération de fusion soit envoyé vers la sortie standard (de façon à ce que Subversion puisse le rediriger vers le fichier suivi en versions approprié). Pour la plupart des programmes de fusion alternatifs, seuls les neuvième, dixième et onzième arguments (les chemins des fichiers qui contiennent les versions « mien », « original » et « leur » respectivement) sont intéressants. Notez que puisque Subversion utilise la sortie de votre programme de fusion, votre script d'interface ne doit pas se terminer avant que la sortie n'ait été fournie à Subversion. Quand il se termine, il doit retourner un code d'erreur de 0 si la fusion s'est correctement déroulée ou de 1 si des conflits non résolus persistent dans la sortie. Tout autre code d'erreur est considéré comme une erreur fatale.

L'Exemple 7.4, « interface-diff3.py » et l'Exemple 7.5, « interface-diff3.bat » sont des modèles pour des scripts d'interface vers un programme externe de fusion en langage Python et script Windows respectivement.

Exemple 7.4. interface-diff3.py

```
#!/usr/bin/env python
import sys
import os

# Indiquez ici le chemin de votre outil de fusion favori.
DIFF3 = "/usr/local/bin/mon-outil-de-fusion"

# Subversion fournit les chemins voulus dans les trois derniers arguments.
MIEN  = sys.argv[-3]
VIEUX = sys.argv[-2]
LEUR  = sys.argv[-1]

# Appelons la commande de fusion (modifiez la ligne suivante
# en accord avec votre outil de fusion).
cmd = [DIFF3, '--older', VIEUX, '--mine', MIEN, '--yours', LEUR]
os.execv(cmd[0], cmd)

# Après avoir effectué la fusion, le script doit envoyer le
# contenu du fichier résultant vers la sortie standard (stdout)
# Faites le à votre convenance.
# Le code d'erreur renvoyé doit être :
# 0 si la fusion a bien fonctionné,
# 1 s'il reste des conflits non résolus.
# Tout autre code est traité comme une erreur fatale.
```

Exemple 7.5. interface-diff3.bat

```
@ECHO OFF

REM Indiquez ici le chemin de votre outil de fusion favori.
SET DIFF3="C:\Program Files\Super Progs\Mon Outil De Fusion.exe"

REM Subversion fournit les chemins voulus dans les trois derniers arguments.
REM Ce sont les paramètres 9, 10 et 11. Mais nous n'avons accès qu'à
REM neuf paramètres en même temps, nous effectuons donc deux décalages
REM pour obtenir les paramètres manquants.
SHIFT
SHIFT
SET MIEN=%7
SET VIEUX=%8
SET LEUR=%9

REM Appelons la commande de fusion (modifiez la ligne suivante
REM en accord avec votre outil de fusion).
%DIFF3% --older %VIEUX% --mine %MIEN% --yours %LEUR%
```

```
REM Après avoir effectué la fusion, le script doit envoyer le
REM contenu du fichier résultant vers la sortie standard (stdout)
REM Faites le à votre convenance.
REM Le code d'erreur renvoyé doit être :
REM 0 si la fusion a bien fonctionné,
REM 1 s'il reste des conflits non résolus.
REM Tout autre code est traité comme une erreur fatale.
```

Résumé

Quelquefois il n'y a qu'une façon de bien faire les choses, quelquefois il en existe plusieurs. Les développeurs de Subversion comprennent que bien que le plus souvent son comportement est acceptable par la plupart des utilisateurs, il y a des fonctionnalités pour lesquelles il n'existe pas de consensus. Dans ces situations, Subversion permet à l'utilisateur de spécifier le comportement qu'*il* désire voir appliquer.

Dans ce chapitre, nous avons examiné le système de configuration des exécutables de Subversion ainsi que d'autres mécanismes permettant de contrôler le comportement de certaines actions. Si vous êtes développeur, le prochain chapitre vous emmènera un cran plus loin. Il décrit comment vous pouvez personnaliser davantage Subversion en écrivant vos propres programmes en remplacement des bibliothèques Subversion.

Chapitre 8. Intégration de Subversion

Subversion est conçu de manière modulaire : il est constitué d'un ensemble de bibliothèques écrites en langage C. Chaque bibliothèque a un but et une interface de programmation (API, *application programming interface* en anglais) bien définis ; cette interface est disponible non seulement pour le propre usage de Subversion mais aussi pour n'importe quel programme qui souhaite inclure ou piloter Subversion d'une manière ou d'une autre. De plus, l'API Subversion est non seulement disponible pour les programmes écrits en langage C, mais aussi pour les programmes écrits dans des langages de plus haut niveau tels que Python, Perl, Java et Ruby.

Ce chapitre est destiné à ceux qui souhaitent interagir avec Subversion au moyen de son API publique ou d'une de ses nombreuses interfaces avec d'autres langages. Si vous souhaitez écrire des scripts robustes qui encapsulent les fonctionnalités de Subversion afin de vous rendre la vie plus facile, si vous essayez de développer des intégrations plus poussées entre Subversion et d'autres logiciels ou si vous êtes juste intéressé par les nombreux modules de Subversion et ce qu'ils ont à offrir, ce chapitre est fait pour vous. Si, par contre, vous ne vous voyez pas participer à Subversion à ce niveau, vous pouvez sauter ce chapitre sans la moindre crainte pour vos compétences en tant qu'utilisateur de Subversion.

Organisation des bibliothèques en couches successives

Chaque bibliothèque au sein de Subversion peut être classée dans une des trois couches principales : la couche dépôt, la couche d'accès au dépôt (RA pour *Repository Access* en anglais) et la couche client (voir la Figure 1, « Architecture de Subversion » de la préface). Nous allons examiner ces trois couches rapidement mais, d'abord, passons brièvement en revue les différentes bibliothèques de Subversion. Pour des raisons de cohérence, nous nous référons à ces bibliothèques par leurs noms Unix sans extension (`libsvn_fs`, `libsvn_wc`, `mod_dav_svn`, etc.).

libsvn_client
: interface principale pour les programmes clients ;

libsvn_delta
: routines de recherche de différences pour les arborescences et les flux d'octets ;

libsvn_diff
: routines de recherche de différences et de fusions contextuelles ;

libsvn_fs
: chargeur de modules et outils communs pour le système de fichiers ;

libsvn_fs_base
: gestion du magasin de données Berkeley DB ;

libsvn_fs_fs
: gestion du magasin de données natif FSFS ;

libsvn_ra
: outils communs pour l'accès au dépôt et chargeur de modules ;

libsvn_ra_local
: module d'accès au dépôt en local ;

libsvn_ra_neon
: module d'accès au dépôt par WebDAV ;

libsvn_ra_serf
: autre module (expérimental) d'accès au dépôt par WebDAV ;

libsvn_ra_svn
: modèle d'accès au dépôt par le protocole Subversion ;

libsvn_repos

interface du dépôt ;

libsvn_subr
: diverses routines utiles ;

libsvn_wc
: bibliothèque pour la gestion de la copie de travail locale ;

mod_authz_svn
: module Apache d'authentification pour les accès aux dépôts Subversion par WebDAV ;

mod_dav_svn
: module Apache de correspondance entre les opérations WebDAV et les opérations Subversion.

Le fait que le mot « divers » n'apparaisse qu'une seule fois dans la liste précédente est plutôt un bon signe. L'équipe de développement de Subversion est particulièrement soucieuse de placer les fonctionnalités dans les couches et bibliothèques appropriées. Un des plus grands avantages de cette conception modulaire, du point de vue du développeur, est sûrement l'absence de complexité. En tant que développeur, vous pouvez vous forger rapidement une image mentale de cette architecture et ainsi trouver relativement facilement l'emplacement des fonctionnalités qui vous intéressent.

Un autre avantage de la modularité est la possibilité de remplacer un module par une autre bibliothèque qui implémente la même API sans affecter le reste du code. Dans un certain sens, c'est ce qui se passe déjà dans Subversion. Les bibliothèques `libsvn_ra_local`, `libsvn_ra_neon`, `libsvn_ra_serf` et `libsvn_ra_svn` implémentent toutes la même interface et fonctionnent comme des greffons pour `libsvn_ra`. Et toutes les quatre communiquent avec la couche dépôt — `libsvn_ra_local` se connectant directement au dépôt, les trois autres le faisant à travers le réseau. `libsvn_fs_base` et `libsvn_fs_fs` sont un autre exemple de bibliothèques qui implémentent les mêmes fonctionnalités de différentes manières — les deux sont des greffons pour la bibliothèque commune `libsvn_fs`.

Le client lui-même illustre également les avantages de la modularité dans l'architecture de Subversion. La bibliothèque `libsvn_client` est un point d'entrée unique pour la plupart des fonctionnalités nécessaires à la conception d'un client Subversion fonctionnel (voir la section intitulée « Couche client »). Ainsi, bien que la distribution Subversion fournisse seulement le programme en ligne de commande **svn**, de nombreux programmes tiers fournissent différents types d'IHM. Ces interfaces graphiques utilisent la même API que le client en ligne de commande fourni en standard. Depuis le début, cette modularité joue un rôle majeur dans la prolifération des différents clients Subversion, sous la forme de clients autonomes ou greffés dans des environnements de développement intégrés (*IDE* en anglais) et, par extension, dans l'adoption formidablement rapide de Subversion lui-même.

Couche dépôt

Quand nous faisons référence à la couche dépôt de Subversion, nous parlons généralement de deux concepts de base : l'implémentation du système de fichiers suivi en versions (auquel on a accès via `libsvn_fs` et qui est supporté par les greffons associés `libsvn_fs_base` et `libsvn_fs_fs`) et la logique du dépôt qui l'habille (telle qu'elle est implémentée dans `libsvn_repos`). Ces bibliothèques fournissent les mécanismes de stockage et de comptes-rendus pour les différentes révisions de vos données suivies en versions. Cette couche est connectée à la couche client via la couche d'accès au dépôt et est, du point de vue de l'utilisateur de Subversion, le « truc à l'autre bout de la ligne ».

Le système de fichiers Subversion n'est pas un système de fichiers de bas niveau que vous pourriez installer sur votre système d'exploitation (tels que NTFS ou ext2 pour Linux) mais un système de fichiers virtuel. Plutôt que de stocker les fichiers et répertoires comme des fichiers et des répertoires réels (du type de ceux dans lesquels vous naviguez avec votre navigateur de fichiers), il utilise un des deux magasins de données abstraits disponibles : soit le système de gestion de bases de données Berkeley DB, soit une représentation dans des fichiers ordinaires, dite « à plat » (pour en apprendre plus sur les deux magasins de données, reportez-vous à la section intitulée « Choix du magasin de données »). La communauté de développement Subversion a même exprimé le souhait que les futures versions de Subversion puissent utiliser d'autres magasins de données, peut-être à travers un mécanisme tel que ODBC (Open Database Connectivity , standard ouvert de connexion à des bases de données). En fait, Google a fait quelque chose de semblable avant de lancer le service « Google Code Project Hosting » (Hébergement de code source de projets) : ils ont annoncé mi-2006 que les membres de leur équipe open source avaient écrit un nouveau greffon propriétaire de système de fichiers pour Subversion, qui utilisait leur base de données « Google ultra-scalable Bigtable » comme magasin de données.

L'API du système de fichiers, mise à disposition par `libsvn_fs`, contient les fonctionnalités que vous pouvez attendre de n'importe quel autre système de fichiers : vous pouvez créer et supprimer des fichiers et des répertoires, les copier et les déplacer, modifier le contenu d'un fichier, etc. Elle possède également des caractéristiques peu communes comme la capacité

d'ajouter, modifier et supprimer des méta-données (« propriétés ») sur chaque fichier ou répertoire. En outre, le système de fichiers Subversion est un système de fichiers suivi en versions, ce qui veut dire que si vous faites des modifications dans votre arborescence, Subversion se souvient de l'état de votre arborescence avant les modifications. Et il se souvient aussi de l'état avant les modifications précédentes, et de l'état encore antérieur, et ainsi de suite. Vous pouvez ainsi remonter le temps (c'est-à-dire les versions) jusqu'au moment où vous avez commencé à ajouter des éléments dans le système de fichiers.

Toutes les modifications faites sur l'arborescence ont pour contexte les transactions de propagation de Subversion. Ce qui suit est la démarche générale simplifiée de modification du système de fichiers :

1. commencer une transaction de propagation de Subversion ;
2. effectuer les modifications (ajouts, suppressions, modifications de propriétés, etc.) ;
3. clore la transaction.

Une fois que la transaction est terminée, les modifications du système de fichiers sont stockées de façon permanente en tant qu'éléments de l'historique. Chacun de ces cycles génère une nouvelle révision de l'arborescence et chaque révision est accessible pour toujours sous la forme d'un cliché, immuable, de l'état de l'arborescence à un moment précis.

Digression sur les transactions

La notion de transaction Subversion peut être facilement confondue avec la notion de transaction concernant le magasin de données sous-jacent, en particulier à cause de la proximité du code des transactions Subversion dans `libsvn_fs_base` et du code du gestionnaire de bases de données Berkeley DB. Ces deux types de transactions assurent l'atomicité et l'isolation. En d'autres termes, les transactions vous permettent d'effectuer un ensemble d'actions avec une logique tout-ou-rien (soit toutes les actions de l'ensemble se terminent avec succès, soit c'est comme si aucune n'avait eu lieu), ce qui permet de ne pas interférer avec les autres processus qui travaillent sur les données.

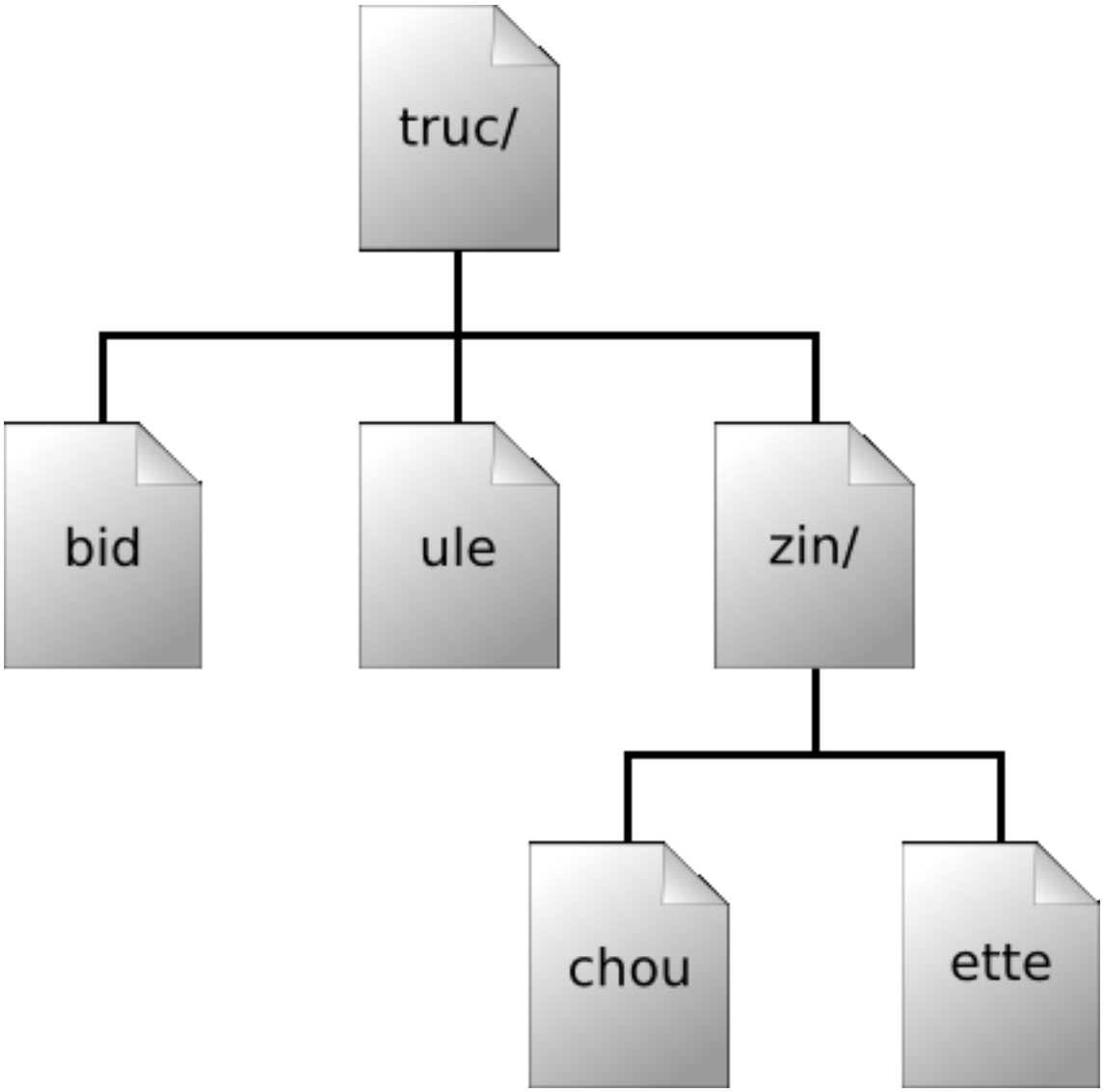
Les transactions dans les bases de données comprennent généralement de petites opérations relatives à la modification de données dans la base elle-même (comme changer le contenu d'une ligne dans une table). Les transactions Subversion ont un champ d'action plus large, elles comprennent des opérations de plus haut niveau telles que modifier un ensemble de fichiers et de répertoires qui doivent être stockés dans la prochaine révision de l'arborescence suivie en versions. Pour ajouter à la confusion, Subversion utilise une transaction de base de données pendant la création d'une transaction Subversion (ainsi, si la création de la transaction Subversion échoue, la base de données sera telle que si la demande de création n'avait jamais eu lieu) !

Heureusement pour les utilisateurs de l'API du système de fichiers, la notion de transaction du système de gestion de bases de données lui-même est presque entièrement masquée (comme on peut s'y attendre dans une architecture modulaire bien construite). C'est seulement si vous commencez à fouiller dans l'implémentation du système de fichiers que de telles choses deviennent visibles (ou intéressantes).

La majeure partie des fonctionnalités offertes par l'interface du système de fichiers traite d'actions relatives à un chemin unique du système de fichiers. C'est-à-dire que, vu de l'extérieur du système de fichiers, le mécanisme de base pour décrire et accéder à une révision donnée d'un fichier ou d'un répertoire utilise des chemins classiques tels que `/machin/bidule`, de la même manière que quand vous indiquez un fichier ou un répertoire dans votre interface en ligne de commande favorite. Vous ajoutez de nouveaux fichiers ou répertoires en passant leur « futur » chemin à la fonction idoine de l'API. Vous faites des requêtes sur ces éléments avec le même mécanisme.

Cependant, contrairement à la plupart des systèmes de fichiers, le chemin n'est pas une information suffisante pour identifier un fichier ou un répertoire dans Subversion. Représentez-vous l'arborescence des répertoires comme un système à deux dimensions, où l'on atteint les frères d'un nœud en se déplaçant horizontalement, à droite ou à gauche, et où la navigation dans les sous-répertoires de ce nœud peut être assimilée à un mouvement vers le bas. La Figure 8.1, « Fichiers et répertoires en deux dimensions » illustre ce concept pour une arborescence classique.

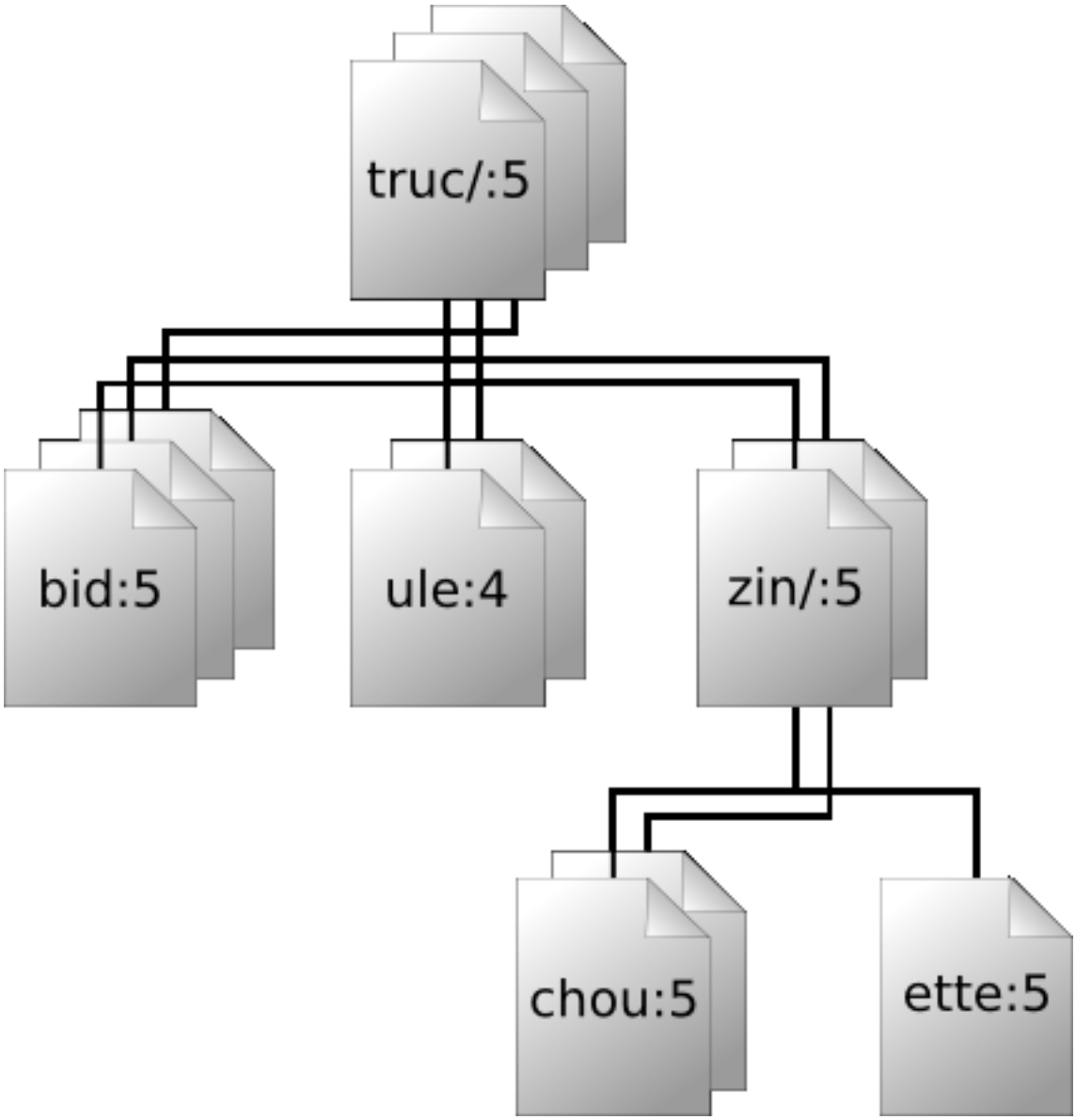
Figure 8.1. Fichiers et répertoires en deux dimensions



Ici, la différence est que le système de fichiers Subversion possède une élégante troisième dimension que la plupart des systèmes de fichiers n'ont pas : le temps[1]. Dans l'interface du système de fichiers, presque chaque fonction qui demande un argument de type `chemin` attend également un argument de type `racine` (dénommé en fait `svn_fs_root_t`). Cet argument décrit soit une révision, soit une transaction (qui est en fait la genèse d'une révision) et fournit la troisième dimension, l'élément de contexte indispensable pour différencier `/machin/bidule` dans la révision 32 et le même chemin dans la révision 98. La Figure 8.2, « Prise en compte du temps — la troisième dimension de la gestion de versions ! » présente l'historique des révisions comme une dimension supplémentaire de l'univers du système de fichiers Subversion.

Figure 8.2. Prise en compte du temps — la troisième dimension de la gestion de versions !

[1]Nous comprenons que cela puisse être un choc énorme pour les amateurs de science-fiction, qui ont longtemps cru que le Temps était en fait la quatrième dimension. Nous nous excusons pour le traumatisme psychologique causé par l'affirmation de cette théorie divergente.



Comme nous l'avons déjà mentionné, l'API de `libsvn_fs` ressemble à s'y méprendre à celle de n'importe quel autre système de fichiers, sauf qu'on y a ajouté la formidable capacité de gestion des versions. Elle a été conçue pour être utilisable par n'importe quel programme ayant besoin d'un système de fichiers suivi en versions. Et ce n'est pas un hasard si Subversion lui-même est intéressé par une telle fonctionnalité. Mais, bien que cette API soit suffisante pour effectuer une gestion de versions basique des fichiers et des répertoires, Subversion en demande plus, et c'est là que `libsvn_repos` entre en scène.

La bibliothèque du dépôt Subversion (`libsvn_repos`) se situe (logiquement parlant) au-dessus de l'API `libsvn_fs` et elle fournit des fonctionnalités supplémentaires allant au-delà de la logique sous-jacente du système de fichiers suivi en versions. Elle ne masque pas entièrement chaque fonction du système de fichiers — seules certaines étapes importantes dans le cycle général de l'activité du système de fichiers sont encapsulées par l'interface du dépôt. Parmi les fonctions encapsulées, on peut citer la création et la propagation des transactions Subversion et la modification des propriétés de révisions. Ces actions particulières sont encapsulées par la couche dépôt parce qu'elles ont des procédures automatiques associées. Le système des procédures automatiques du dépôt n'est pas strictement concomitant à l'implémentation d'un système de fichiers suivi en versions, c'est pourquoi il réside dans la bibliothèque d'encapsulation du dépôt.

Le mécanisme des procédures automatiques n'est pas l'unique raison qui a conduit à séparer logiquement la bibliothèque du dépôt du reste du code du système de fichiers. L'API de `libsvn_repos` fournit à Subversion un certain nombre d'autres possibilités intéressantes. Parmi elles, on peut citer :

- créer, ouvrir, détruire et effectuer des actions de restauration sur un dépôt Subversion et le système de fichiers inclus dans ce

dépôt ;

- décrire les différences entre deux arborescences ;
- obtenir les messages de propagation associés à toutes les révisions (ou certaines) qui ont modifié un ensemble de fichiers du système de fichiers ;
- générer des images (« dumps ») du système de fichiers lisibles par l'utilisateur — ces images étant des représentations complètes des révisions du système de fichiers ;
- analyser ces images et les charger dans un autre dépôt Subversion.

Comme Subversion continue à évoluer, la bibliothèque du dépôt grandit avec la bibliothèque du système de fichiers pour offrir davantage de fonctionnalités et des options configurables.

Couche d'accès au dépôt

Si la couche Dépôt de Subversion est « à l'autre bout de la ligne », la couche d'accès au dépôt (RA pour *repository access* en anglais) est la ligne en tant que telle. Chargée d'organiser les données entre les bibliothèques client et le dépôt, cette couche inclut la bibliothèque de chargement du module `libsvn_ra`, les modules RA eux-mêmes (qui incluent à l'heure actuelle `libsvn_ra_neon`, `libsvn_ra_local`, `libsvn_ra_serf` et `libsvn_ra_svn`) et toute bibliothèque supplémentaire requise par un ou plusieurs de ces modules RA (par exemple, le module Apache `mod_dav_svn` ou le serveur de `libsvn_ra_svn`, **svnserve**).

Comme Subversion utilise les URL pour identifier les dépôts à contacter, la partie de l'URL qui indique le protocole (habituellement `file://`, `http://`, `https://`, `svn://` ou `svn+ssh://`) est utilisée pour déterminer quel module RA gère les communications. Chaque module indique la liste des protocoles qu'il connaît afin que le chargeur RA puisse déterminer, à l'exécution, quel module utiliser pour la tâche en cours. Vous pouvez obtenir la liste des modules RA disponibles pour votre client Subversion en ligne de commande, ainsi que les protocoles qu'ils prennent en charge, en lançant la commande **`svn --version`** :

```
$ svn --version
svn, version 1.5.0 (r31699)
    compilé Jun 18 2008, 09:57:36

Copyright (C) 2000-2008 CollabNet.
Subversion est un logiciel libre, cf http://subversion.tigris.org/
Il inclut du logiciel développé par CollabNet (http://www.Collab.Net/).

Les modules d'accès à un dépôt (RA) suivants sont disponibles :

* ra_neon : Module d'accès à un dépôt via le protocole WebDAV avec Neon.
  - gère le schéma d'URL 'http'
  - gère le schéma d'URL 'https'
* ra_svn : Module d'accès à un dépôt avec le protocole réseau propre de svn.
  - avec authentification Cyrus SASL
  - gère le schéma d'URL 'svn'
* ra_local : Module d'accès à un dépôt sur un disque local.
  - gère le schéma d'URL 'file'
* ra_serf : Module d'accès à un dépôt via le protocole WebDAV avec serf.
  - gère le schéma d'URL 'http'
  - gère le schéma d'URL 'https'
```

L'API publique exportée par la couche RA contient les fonctionnalités nécessaires pour envoyer des données suivies en versions vers le dépôt et pour en recevoir. Chacun des greffons RA disponibles est capable d'effectuer ces tâches en utilisant un protocole particulier : `libsvn_ra_dav` utilise le protocole HTTP/WebDAV (avec chiffrement SSL en option) pour communiquer avec un serveur HTTP Apache sur lequel tourne le module serveur Subversion `mod_dav_svn` ; `libsvn_ra_svn` utilise un protocole réseau propre à Subversion pour communiquer avec le programme **svnserve**, et ainsi de suite.

Ceux qui désirent accéder à un dépôt Subversion en utilisant un autre protocole comprendront rapidement pourquoi la couche d'accès au dépôt est modulaire ! Les développeurs peuvent tout simplement écrire une nouvelle bibliothèque qui implémente

l'interface RA d'un côté et qui communique avec le dépôt de l'autre. Votre nouvelle bibliothèque peut utiliser des protocoles réseaux existants ou vous pouvez en inventer de nouveaux. Vous pouvez ainsi utiliser les communications inter-processus (IPC pour *interprocess communication* en anglais) ou même, soyons fou, implémenter un protocole basé sur l'email. Subversion apporte les API, à vous d'apporter la créativité.

Couche client

Côté client, tout se passe dans la copie de travail Subversion. Le gros des fonctionnalités implémentées par les bibliothèques client existe dans le seul but de gérer les copies de travail locales — des répertoires pleins de fichiers et d'autres sous-répertoires qui sont une sorte de copie locale et modifiable d'un ou plusieurs dépôts — et de propager les changements vers et depuis la couche d'accès au dépôt.

La bibliothèque de Subversion pour la copie de travail, `libsvn_wc`, est directement responsable de la gestion des données dans les copies de travail. Pour ce faire, la bibliothèque stocke dans un sous-répertoire spécial des données d'administration concernant chaque répertoire suivi en versions. Ce sous-répertoire, nommé `.svn`, est présent dans chaque répertoire d'une copie de travail ; il contient tout un tas de fichiers et de répertoires qui enregistrent l'état du répertoire suivi en versions et fournit un espace privé pour les actions d'administration. Pour les habitués de CVS, ce sous-répertoire `.svn` a des objectifs similaires aux répertoires administratifs `CVS` que l'on trouve dans les copies de travail CVS. Pour plus d'informations sur la zone d'administration `.svn`, reportez-vous à la section intitulée « Au cœur de la zone d'administration de la copie locale » plus loin dans ce chapitre.

La bibliothèque client de Subversion, `libsvn_client`, est celle qui a le plus de responsabilités : son rôle est de mélanger les fonctionnalités de la bibliothèque de la copie de travail avec celles de la couche d'accès au dépôt (RA) afin de fournir l'API de plus haut niveau, utilisable par n'importe quelle application qui voudrait effectuer des actions générales de gestion de versions. Par exemple, la fonction `svn_client_checkout()` prend une URL en argument. Elle passe cette URL à la couche RA et ouvre une session authentifiée avec le dépôt concerné. Elle demande ensuite au dépôt l'arborescence requise, envoie cette arborescence à la bibliothèque de la copie de travail, qui écrit alors une copie de travail complète sur le disque (les répertoires `.svn` et tout le reste).

La bibliothèque client est conçue pour être utilisée par n'importe quelle application. Alors que le code source de Subversion inclut un client standard en ligne de commande, le but recherché est qu'il soit très facile d'écrire un nombre quelconque de clients dotés d'un environnement graphique (*GUI* en anglais) par-dessus cette bibliothèque client. Il n'y a pas de raison que les nouveaux environnements graphiques (ou les nouveaux clients en fait) pour Subversion ne soient que des sur-couches au client en ligne de commande : ils ont un accès total, via l'API `libsvn_client`, aux mêmes fonctionnalités, données et autres mécanismes que le client en ligne de commande utilise. En fait, le code source de Subversion contient un petit programme en C (que vous pouvez trouver dans `tools/examples/minimal_client.c`) qui montre comment utiliser en pratique l'API Subversion pour créer un programme client simple.

Un mot sur la pertinence d'utiliser directement les bibliothèques

Pourquoi utiliser directement `libsvn_client` pour votre interface graphique plutôt que d'encapsuler le programme en ligne de commande ? Non seulement c'est plus efficace, mais c'est aussi plus pertinent. Un programme en ligne de commande (tel que celui fourni avec Subversion) qui utilise la bibliothèque client a besoin de traduire effectivement des requêtes et des réponses contenues dans des variables en C en un affichage lisible par l'utilisateur. Ce type de traduction peut induire des pertes. C'est-à-dire que le programme n'affiche peut-être pas l'ensemble des informations qu'il a obtenues de l'API ou qu'il combine peut-être certaines informations pour obtenir une représentation plus compacte.

Si vous encapsulez le programme en ligne de commande avec un autre programme, cette sur-couche n'a accès qu'à des informations déjà interprétées (et, comme nous venons de le mentionner, potentiellement incomplètes) et elle doit une nouvelle fois traduire ces informations vers son propre format de représentation des données. À chaque couche d'encapsulation supplémentaire, l'intégrité des données originales s'effrite un peu plus, à la manière d'une copie de copie (de copie …) d'une cassette audio ou vidéo.

Mais l'argument décisif quant à l'utilisation directe des API plutôt que d'encapsuler d'autres programmes est que le projet Subversion assure la compatibilité vis-à-vis de ses API. Lors des changements de version mineure des API (comme par exemple entre la version 1.3 et 1.4), aucun prototype de fonction ne change. En d'autres termes, vous n'êtes pas forcé de mettre à jour le code source de votre programme simplement parce que vous avez mis à jour votre version de Subversion. Certaines fonctions seront peut-être obsolètes, mais elles fonctionneront toujours. Ainsi, cela vous laisse de la marge pour éventuellement adopter les nouvelles API. Ce type de promesse de compatibilité n'existe pas pour les sorties du programme Subversion en ligne de commande, qui sont susceptibles de changer à chaque version.

Au cœur de la zone d'administration de la copie locale

Comme nous l'avons déjà mentionné, chaque répertoire d'une copie de travail Subversion contient un sous-répertoire spécial nommé `.svn` qui héberge les données administratives concernant le répertoire de la copie de travail. Subversion utilise `.svn` pour gérer des informations telles que :

- à quel emplacement du dépôt les fichiers et les sous-répertoires du répertoire de la copie de travail font référence ;
- quelle révision de chacun de ces fichiers et répertoires est présente dans la copie de travail ;
- toute propriété définie et associée par l'utilisateur à ces fichiers et répertoires ;
- une copie locale originale des fichiers de la copie de travail.

L'agencement et le contenu de la zone d'administration de la copie de travail Subversion sont considérés comme des détails de l'implémentation non-destinés à être exploités par les utilisateurs. Nous encourageons les développeurs à utiliser les API publiques ou les outils fournis avec Subversion pour accéder aux données de la copie de travail et pour les manipuler, plutôt que de lire et modifier directement ces fichiers. Les formats de fichiers employés par la bibliothèque de la copie de travail pour gérer les données administratives changent de temps en temps et l'API publique réalise un gros travail pour que l'utilisateur moyen ne s'en rende pas compte. Dans cette section, nous abordons crûment certains détails de l'implémentation pour satisfaire votre insatiable curiosité.

Le fichier « entries »

Le fichier `entries` est sûrement LE fichier le plus important du répertoire `.svn`. Il contient l'essentiel des informations administratives concernant les éléments suivis en versions du répertoire de la copie de travail. Ce fichier contient l'URL du dépôt, le numéro de révision original, les sommes de contrôle des fichiers, les horodatages des copies originales et des propriétés, les informations d'état sur les conflits et les actions planifiées, les dernières informations connues sur les propagations (auteur, numéro de révision et horodatage) et l'historique de la copie locale : pratiquement tout ce qui intéresse un client Subversion à propos d'un élément suivi (ou à suivre) en versions !

Les lecteurs familiers avec les répertoires administratifs de CVS auront tout de suite reconnu que le fichier `.svn/entries` a les mêmes objectifs, entre autres choses, que l'ensemble des fichiers `CVS/Entries`, `CVS/Root` et `CVS/Repository` combinés.

Le format du fichier `.svn/entries` a changé au cours du temps. Au départ, c'était un fichier XML ; aujourd'hui, il utilise un format personnalisé mais toujours lisible par l'utilisateur. Le choix d'XML était particulièrement adapté pour les premiers développeurs de Subversion qui déboguaient fréquemment le contenu du fichier (et le comportement associé de Subversion). Cependant, le besoin de débogage a diminué au fur et à mesure que Subversion devenait plus mature et le besoin de performance au profit de l'utilisateur a alors pris le dessus. Soyez conscient que la bibliothèque Subversion de la copie de travail met automatiquement à niveau les copies de travail d'un format à un autre — elle comprend les vieux formats et utilise pour l'écriture le nouveau format — ce qui vous épargne l'effort d'extraire une nouvelle copie de travail mais peut aussi compliquer certaines situations : lorsque différentes versions de Subversion essaient de partager la même copie de travail.

Copies originales et propriétés des fichiers

Comme mentionné auparavant, le répertoire `.svn` contient aussi les copies originales (avant modification locale) des fichiers : les « versions de base » (*text-base versions* en anglais). Vous pouvez les trouver dans le répertoire `.svn/text-base`. Ces copies originales offrent tout un tas d'avantages — identification des différences et des modifications, retour en arrière sur les fichiers modifiés ou supprimés, tout cela sans faire appel au réseau ; échanges plus performants avec le serveur — mais au prix d'avoir chaque fichier stocké au moins en double sur le disque. De nos jours, c'est pratiquement négligeable pour la majorité des fichiers. Cependant, la situation se détériore à mesure que vos fichiers suivis en versions grossissent. Nous étudions dorénavant la possibilité de rendre optionnelle la présence de ces « versions de base ». Cependant, c'est justement quand vos fichiers suivis en versions grossissent que l'existence des « versions de base » devient cruciale : qui a envie de transmettre un énorme fichier à travers le réseau juste parce qu'il veut propager une petite modification ?

Dans le même esprit que les fichiers « versions de base », nous avons les fichiers de propriétés qui, eux aussi, possèdent leurs

copies originales « propriétés en version de base », situés respectivement dans `.svn/props` et `.svn/prop-base`. Comme les répertoires aussi peuvent avoir des propriétés, il existe également des fichiers `.svn/dir-props` et `.svn/dir-prop-base`.

Utiliser les API

Développer des applications utilisant les API des bibliothèques Subversion est plutôt simple. Subversion est d'abord un ensemble de bibliothèques en langage C, avec des fichiers d'en-têtes (`.h`) situés dans le répertoire `subversion/include` de l'arborescence des sources. Ces en-têtes sont copiés dans votre arborescence système (par exemple `/usr/local/include`) quand vous compilez et installez Subversion à partir des sources. Ces en-têtes contiennent l'ensemble des fonctions et des types censés être accessibles aux utilisateurs des bibliothèques Subversion. La communauté des développeurs Subversion apporte beaucoup d'attention à la disponibilité et la qualité de la documentation des API publiques — reportez-vous directement aux fichiers d'en-têtes pour cette documentation.

Quand vous examinez les fichiers d'en-tête publics, la première chose que vous remarquez est que les types de données et les fonctions ont un espace de nommage réservé. Cela veut dire que tous les noms de symboles Subversion publics commencent par `svn_`, suivi d'un code indiquant la bibliothèque dans laquelle le symbole est défini (par exemple `wc`, `client`, `fs`, etc.), suivi d'un unique caractère souligné (_) puis du reste du nom du symbole. Les fonctions semi-publiques (utilisées par plusieurs fichiers au sein d'une bibliothèque mais pas par du code extérieur à cette bibliothèque, on peut les trouver au sein des répertoires de la bibliothèque) suivent une règle de nommage légèrement différente dans le sens où, au lieu d'un unique caractère souligné après le code indiquant la bibliothèque, elles utilisent deux caractères souligné consécutifs (_ _). Les fonctions qui sont propres à un fichier source (c'est-à-dire privées) n'ont pas de préfixe particulier et sont déclarées avec le mot-clé `static`. Bien sûr, un compilateur n'a que faire de ces conventions de nommage, mais elles sont une aide précieuse pour clarifier la portée d'une fonction ou d'un type de données particuliers.

Une autre bonne source d'informations sur la programmation avec les API Subversion est constituée par les bonnes pratiques de programmation au sein du projet lui-même, que vous pouvez trouver à l'adresse suivante http://subversion.apache.org/docs/community-guide/ (pages en anglais). Ce document contient des informations particulièrement utiles qui, bien que destinées aux développeurs (ou aux personnes désireuses de le devenir) de Subversion lui-même, peuvent également servir à tous ceux qui développent des applications utilisant Subversion comme bibliothèque tierce [2].

APR, la bibliothèque Apache de portabilité des exécutables

À côté des types de données propres à Subversion, vous trouverez de nombreuses références à des types de données qui commencent par `apr_` : ce sont les symboles de la bibliothèque pour la portabilité d'Apache (*Apache Portable Runtime* en anglais, soit APR). APR est un jeu de bibliothèques Apache, originellement extraites du code source du serveur pour essayer de séparer ce qui dépendait du système d'exploitation de ce qui n'en dépendait pas. Au final, on obtient une bibliothèque qui fournit une API permettant d'effectuer des opérations qui changent un peu (ou beaucoup) en fonction du système d'exploitation. Alors que le serveur HTTP Apache était le premier utilisateur (et pour cause) de la bibliothèque APR, les développeurs Subversion ont immédiatement perçu les avantages qu'il y a à utiliser APR. Cela signifie qu'il n'y a pratiquement aucun code spécifique à un système d'exploitation dans Subversion en tant que tel. Cela veut aussi dire que le client Subversion peut être compilé et exécuté partout où un serveur Apache peut l'être. Actuellement, cette liste comprend toutes les variantes d'Unix, Win32, BeOS, OS/2 et Mac OS X.

En plus de fournir des implémentations fiables des appels systèmes qui diffèrent d'un système d'exploitation à l'autre [3], APR fournit à Subversion un accès direct à de nombreux types de données personnalisés tels que les tableaux dynamiques et les tables de hachage. Subversion utilise abondamment ces types de données et le type de données APR le plus utilisé, que l'on retrouve dans presque tous les prototypes de l'API Subversion, est `apr_pool_t` — le réservoir de mémoire (*memory pool* en anglais) APR. Subversion utilise les réservoirs de mémoire en interne pour tous ses besoins d'allocation mémoire (à moins qu'une bibliothèque externe ne requière un autre mécanisme de gestion de la mémoire pour les données transmises via son API) [4] et, bien qu'une personne qui utilise l'API Subversion ne soit pas obligée d'en faire autant, elle doit fournir des réservoirs aux fonctions de l'API qui en ont besoin. Cela implique que les utilisateurs de l'API Subversion *doivent* également inclure l'APR lors de l'édition de liens, doivent appeler `apr_initialize()` pour initialiser le sous-système APR et doivent ensuite créer et gérer des réservoirs de mémoire pour les appels à l'API Subversion, généralement en utilisant `svn_pool_create()`, `svn_pool_clear()` et `svn_pool_destroy()`.

[2] Après tout, Subversion utilise aussi les API Subversion.
[3] Subversion utilise les appels système et les types de données ANSI autant que possible.
[4] Neon et Berkeley DB par exemple.

Programmer avec les réservoirs de mémoire

Presque tous les développeurs qui ont essayé le langage C se sont heurtés à la tâche dantesque de gestion de la mémoire. Allouer suffisamment de mémoire pour l'exécution, garder une trace de ces allocations, libérer la mémoire quand elle n'est plus utilisée — ces tâches peuvent devenir particulièrement complexes. Et, bien sûr, si cette gestion est mal faite, cela peut conduire à un plantage du programme, voire de l'ordinateur.

Les langages de plus haut niveau, quant à eux, soit vous débarrassent complètement de cette tâche, soit vous laissent jouer avec uniquement quand vous faites des optimisations particulièrement pointues de votre programme. Des langages tels que Java ou Python utilisent un *ramasse-miettes* (*garbage collector* en anglais) qui alloue de la mémoire aux objets en cas de besoin et la libère automatiquement quand l'objet n'est plus utilisé.

APR fournit une approche à mi-chemin appelée *gestion de mémoire par réservoir*. Cela permet au développeur de contrôler l'utilisation de la mémoire à une résolution plus faible — par morceau (dit « réservoir ») de mémoire au lieu d'une gestion par objet. Plutôt que d'utiliser `malloc()` et compagnie pour allouer la mémoire à un objet donné, vous demandez à APR d'allouer de la mémoire à l'intérieur d'un réservoir de mémoire. Quand vous avez fini d'utiliser les objets que vous avez créés dans un réservoir, vous détruisez le réservoir tout entier, ce qui libère effectivement la mémoire consommée par *tous* les objets alloués. Ainsi, plutôt que de gérer individuellement la mémoire qui doit être allouée et libérée pour chaque objet, votre programme n'a plus qu'à se préoccuper de la durée de vie globale des objets et alloue ces objets dans un réservoir dont la durée de vie (le temps entre la création et la suppression du dit réservoir) correspond aux besoins des objets.

Prérequis pour les URL et les chemins

Subversion a été conçu pour effectuer à distance des opérations de gestion de versions. À ce titre, les possibilités d'internationalisation (i18n) ont fait l'objet d'une attention toute particulière. Après tout, « à distance » peut vouloir dire depuis un ordinateur situé « dans le même bureau », mais aussi « à l'autre bout de la planète ». Pour faciliter cette prise en compte, toutes les interfaces publiques de Subversion qui acceptent des chemins comme argument s'attendent à ce que ces chemins soient rendus canoniques — la façon la plus facile de le faire étant de les passer en argument à la fonction `svn_path_canonicalize()` — et codés dans le format UTF-8. Cela signifie, par exemple, que tout nouveau programme client qui pilote l'interface `libsvn_client` doit d'abord convertir les chemins depuis le codage local vers UTF-8 avant de fournir ces chemins à la bibliothèque Subversion, puis doit reconvertir tout chemin renvoyé par Subversion vers le codage local avant d'utiliser ce chemin à des fins externes à Subversion. Heureusement, Subversion fournit un ensemble de fonctions (voir `subversion/include/svn_utf.h`) que tout programme peut utiliser pour réaliser ces conversions.

De plus, les API Subversion demandent que toutes les URL passées en paramètres respectent le format URI. Ainsi, au lieu de désigner par `file:///home/utilisateur/Mon fichier.txt` l'URL d'un fichier nommé `Mon fichier.txt` situé dans le répertoire `home/utilisateur`, vous devez utiliser `file:///home/utilisateur/Mon%20fichier.txt`. Là encore, Subversion fournit des fonctions utiles à votre application — `svn_path_uri_encode()` et `svn_path_uri_decode()` pour coder et décoder, respectivement, des URI.

Utiliser d'autres langages que C et C++

Si vous désirez utiliser les bibliothèques Subversion à partir d'un autre langage que le C (par exemple un programme Python ou Perl), Subversion offre cette possibilité via le générateur simplifié d'interface et d'encapsulation (*Simplified Wrapper and Interface Generator* ou SWIG en anglais). Les interfaces SWIG de Subversion sont situées dans le répertoire `subversion/bindings/swig`. Elles sont toujours en cours d'évolution mais sont utilisables. Elles vous permettent d'appeler les fonctions de l'API Subversion indirectement, en utilisant des interfaces qui traduisent les types de données natifs de votre langage de programmation vers les types de données utilisés par les bibliothèques C de Subversion.

Des efforts significatifs ont été fournis pour produire des interfaces SWIG pleinement fonctionnelles pour Python, Perl et Ruby. D'une certaine manière, le travail effectué pour réaliser les interfaces vers ces langages est réutilisable pour produire des interfaces vers d'autres langages supportés par SWIG (ce qui inclut, entre autres, des versions de C#, Guile, Java, MzScheme, OCaml, PHP et Tcl). Cependant, vous aurez besoin d'un peu de programmation supplémentaire pour aider SWIG à faire les traductions entre les langages pour les API complexes. Pour plus d'informations sur SWIG lui-même, visitez le site Web du projet à l'adresse suivante : http://www.swig.org/ (site en anglais).

Subversion fournit également une interface vers le langage Java. L'interface javahl (située dans `subversion/`

`bindings/java` dans l'arborescence des sources Subversion) n'est pas basée sur SWIG mais est un mélange de Java et de JNI codé à la main. Javahl couvre le plus gros des API du client Subversion et se destine principalement aux développeurs d'environnements de développement intégrés (IDE) et de clients Subversion en Java.

Les interfaces Subversion vers les langages de programmation ne sont pas suivies avec le même niveau d'exigence que les modules du cœur de Subversion, mais peuvent généralement être utilisées en production. De nombreuses applications, de nombreux scripts, des clients graphiques alternatifs et des outils tiers utilisent aujourd'hui sans problème les interfaces vers les langages de programmation afin d'intégrer les fonctionnalités de Subversion.

Veuillez tout de même noter qu'il existe d'autres options pour s'interfacer avec Subversion dans d'autres langages : les interfaces pour Subversion qui ne sont pas fournies par la communauté de développement Subversion. Vous pouvez trouver des liens vers ces interfaces alternatives sur la page de liens externes du projet Subversion (à l'adresse http://subversion.tigris.org/links.html) et, en particulier, nous accordons une mention spéciale à deux d'entre elles. D'abord, l'interface PySVN de Barry Scott (http://pysvn.tigris.org/) est une interface reconnue vers Python. PySVN se targue d'une interface plus « pythonique » que les API « orientées C » fournies par l'interface standard de Subversion vers Python. Et si vous recherchez une implémentation 100 % Java de Subversion, jetez un œil à SVNKit (http://svnkit.com/), qui est une ré-écriture complète de Subversion en Java.

SVNKit ou javahl ?

En 2005, une petite entreprise du nom de TMate annonçait la sortie de la version 1.0.0 de JavaSVN — une implémentation 100 % Java de Subversion. Depuis, le projet a été renommé en SVNKit (disponible sur le site http://svnkit.com/) et connaît un grand succès en étant intégré dans de nombreux clients Subversion, IDE ou autres outils tiers.

La bibliothèque SVNKit est intéressante dans le sens où, contrairement à la bibliothèque javahl, elle ne se contente pas d'encapsuler les bibliothèques officielles du cœur de Subversion. En fait, elle ne partage aucun code avec Subversion. Cependant, bien qu'il soit facile de confondre SVNKit et javahl, et même encore plus facile de ne pas savoir laquelle de ces bibliothèques vous utilisez, vous devez être conscient que SVNKit diffère de javahl sur certains points particulièrement importants. D'abord, SVNKit n'est pas un logiciel libre et il semble qu'il ne soit développé que par une équipe de quelques personnes. La licence de SVNKit est aussi plus restrictive que celle de Subversion. Enfin, en voulant être une bibliothèque Subversion écrite uniquement en Java, SVNKit est limité dans sa capacité à cloner les fonctionnalités de Subversion au fur et à mesure de la sortie de nouvelles versions de ce dernier. Ce problème est déjà apparu une fois : SVNKit ne peut pas accéder à des dépôts Subversion utilisant une base de données BDB via le protocole `file://` car il n'existe pas d'implémentation 100 % Java de Berkeley DB qui soit compatible avec le format de fichier de l'implémentation native de cette bibliothèque.

Ceci dit, SVNKit est unanimement reconnu pour sa fiabilité. Et une solution 100 % Java est beaucoup plus robuste vis-à-vis des erreurs de programmation : un bogue dans SVNKit génère une exception Java que vous pouvez intercepter, tandis qu'un bogue dans une bibliothèque du cœur de Subversion utilisée par javahl peut mettre par terre tout votre environnement d'exécution Java. En conclusion, pesez le pour et le contre avant de choisir une implémentation en Java de Subversion.

Exemples de code

L'Exemple 8.1, « Utilisation de la couche dépôt » contient un bout de code (écrit en C) qui illustre plusieurs concepts que nous venons d'aborder. Il utilise à la fois l'interface du dépôt et celle du système de fichiers (comme dénoté par les préfixes `svn_repos_` et `svn_fs_` des noms de fonctions) pour créer une nouvelle révision dans laquelle un répertoire est ajouté. Vous pouvez y observer l'utilisation du réservoir de mémoire APR qui est utilisé pour les besoins d'allocation mémoire. En outre, le code révèle le côté obscur de la gestion des erreurs de Subversion : toutes les erreurs Subversion doivent être explicitement prises en compte pour éviter des fuites de mémoire (et dans certains cas, le plantage de l'application).

Exemple 8.1. Utilisation de la couche dépôt

```
/* Convertit une erreur Subversion en un simple code d'erreur booléen
 *
 * NOTE:  Les erreurs Subversion doivent être effacées (en utilisant
 *        svn_error_clear()) parce qu'elles sont allouées depuis le
 *        réservoir global, sinon cela produit une fuite de mémoire.
```

```
 */
#define INT_ERR(expr)                                   \
  do {                                                  \
    svn_error_t *__temperr = (expr);                    \
    if (__temperr)                                      \
      {                                                 \
        svn_error_clear(__temperr);                     \
        return 1;                                       \
      }                                                 \
    return 0;                                           \
  } while (0)

/* Crée un nouveau répertoire NOUVEAU_REP dans le dépôt Subversion
 * situé à CHEMIN_DEPOT. Effectue toutes les allocations mémoire dans
 * RESERVOIR. Cette fonction créera une nouvelle révision pour l'ajout
 * de NOUVEAU_REP. Elle retourne zéro si l'opération se termine
 * correctement, une valeur différente de zéro sinon.
 */
static int
cree_nouveau_rep(const char *chemin_depot,
                 const char *nouveau_rep,
                 apr_pool_t *reservoir)
{
  svn_error_t *err;
  svn_repos_t *depot;
  svn_fs_t *fs;
  svn_revnum_t derniere_rev;
  svn_fs_txn_t *transaction;
  svn_fs_root_t *racine_transaction;
  const char *chaine_conflit;

  /* Ouvre le dépôt situé à chemin_depot.
   */
  INT_ERR(svn_repos_open(&depot, chemin_depot, reservoir));

  /* Obtient un pointeur sur l'objet du système de fichiers qui est
   * stocké dans CHEMIN_DEPOT.
   */
  fs = svn_repos_fs(depot);

  /* Demande au système de fichiers de nous fournir le numéro de la
   * révision la plus récente.
   */
  INT_ERR(svn_fs_youngest_rev(&derniere_rev, fs, reservoir));

  /* Commence une nouvelle transaction qui est basée sur DERNIERE_REV.
   * Nous aurons moins de chance de voir notre propagation rejetée pour
   * cause de conflit si nous effectuons toujours nos changements à partir du
   * dernier instantané de l'arborescence du système de fichiers.
   */
  INT_ERR(svn_repos_fs_begin_txn_for_commit2(&transaction, depot,
                                             derniere_rev,
                                             apr_hash_make(reservoir),
                                             reservoir));

  /* Maintenant qu'une nouvelle transaction Subversion est commencée,
   * obtient l'objet racine qui représente cette transaction.
   */
  INT_ERR(svn_fs_txn_root(&racine_transaction, transaction, reservoir));

  /* Crée un nouveau répertoire sous la racine de la transaction, au
   * chemin NOUVEAU_REP.
   */
  INT_ERR(svn_fs_make_dir(racine_transaction, nouveau_rep, reservoir));

  /* Propage la transaction, créant une nouvelle révision du système de
   * fichiers incluant le nouveau répertoire.
   */
  err = svn_repos_fs_commit_txn(&chaine_conflit, depot,
```

```
                                &derniere_rev, transaction, reservoir);
  if (! err)
    {
      /* Pas d'erreur ? Excellent ! Indique brièvement la réussite
       * de l'opération.
       */
      printf("Le répertoire '%s' a été ajouté en tant que nouvelle "
             "révision '%ld'.\n", nouveau_rep, derniere_rev);
    }
  else if (err->apr_err == SVN_ERR_FS_CONFLICT)
    {
      /* Oh-oh. La propagation a échoué pour cause de conflit (il semble
       * que quelqu'un d'autre a effectué des changements dans la même
       * zone du système de fichiers que celle que nous avons essayé de
       * modifier). Affiche un message d'erreur.
       */
      printf("Un conflit s'est produit pour le chemin '%s' lors de"
             " l'ajout du répertoire '%s' au dépôt '%s'.\n",
             chaine_conflit, nouveau_rep, chemin_depot);
    }
  else
    {
      /* Une autre erreur s'est produite. Affiche un message d'erreur.
       */
      printf("Une erreur s'est produite lors de l'ajout du "
             "répertoire '%s' au dépôt '%s'.\n",
             nouveau_rep, chemin_depot);
    }

  INT_ERR(err);
}
```

Notez que dans l'Exemple 8.1, « Utilisation de la couche dépôt », le code aurait tout aussi bien pu propager la transaction en utilisant `svn_fs_commit_txn()`. Mais l'API du système de fichiers ignore tout des mécanismes de procédures automatiques de la bibliothèque du dépôt. Si vous voulez que votre dépôt Subversion effectue automatiquement certaines tâches externes à Subversion chaque fois qu'une transaction est propagée (par exemple envoyer un mail qui décrit les changements effectués dans la transaction à la liste de diffusion des développeurs), vous devez utiliser la version de la fonction encapsulée dans `libsvn_repos` qui ajoute la fonctionnalité d'activation des procédures automatiques : `svn_repos_fs_commit_txn()` (pour davantage d'informations sur les procédures automatiques des dépôts Subversion, consultez la section intitulée « Mise en place des procédures automatiques »).

Maintenant, changeons de langage. L'Exemple 8.2, « Utilisation de la couche dépôt en Python » est un programme de démonstration qui utilise l'interface SWIG vers Python pour parcourir récursivement la dernière révision du dépôt et afficher les différents chemins trouvés lors de ce parcours.

Exemple 8.2. Utilisation de la couche dépôt en Python

```
#!/usr/bin/python

"""Parcourir un dépôt en affichant les chemins des objets suivis en
versions."""

import sys
import os.path
import svn.fs, svn.core, svn.repos

def parcourir_rep_systemedefichiers(racine, repertoire):
    """Parcourt récursivement le REPERTOIRE situé sous RACINE dans le
       système de fichiers. Renvoie la liste de tous les chemins sous et
       de REPERTOIRE."""

    # Affiche le nom de ce chemin.
```

```
    print repertoire + "/"

    # Obtient les entrées du répertoire REPERTOIRE.
    entrees = svn.fs.svn_fs_dir_entries(racine, repertoire)

    # Pour chaque entrée
    noms = entrees.keys()
    for nom in noms:
        # Calcule le chemin complet de l'entrée.
        chemin_complet = repertoire + '/' + nom

        # Si l'entrée est un répertoire, effectue une récursion. La
        # récursion retournera une liste comprenant l'entrée et tous ses
        # enfants, que l'on ajoutera à notre liste.
        if svn.fs.svn_fs_is_dir(racine, chemin_complet):
            parcourir_rep_systemedefichiers(racine, chemin_complet)
        else:
            # Sinon, c'est un fichier donc l'afficher maintenant.
            print chemin_complet

def parcourir_la_plus_recente_revision(chemin_depot):
    """Ouvre le dépôt situé à CHEMIN_DEPOT et effectue un parcours
       récursif de la révision la plus récente."""

    # Ouvre le dépôt situé à CHEMIN_DEPOT et obtient une référence de
    # son système de fichiers suivi en versions.
    objet_depot = svn.repos.svn_repos_open(chemin_depot)
    objet_fs = svn.repos.svn_repos_fs(objet_depot)

    # Obtient la révision la plus récente (HEAD).
    rev_la_plus_recente = svn.fs.svn_fs_youngest_rev(objet_fs)

    # Ouvre un objet racine représentant la révision la plus récente.
    objet_racine = svn.fs.svn_fs_revision_root(objet_fs,
                                               rev_la_plus_recente)

    # Effectue le parcours récursif.
    parcourir_rep_systemedefichiers(objet_racine, "")

if __name__ == "__main__":
    # Vérifie que l'on est appelé correctement.
    if len(sys.argv) != 2:
        sys.stderr.write("Usage: %s CHEMIN_DEPOT\n"
                         % (os.path.basename(sys.argv[0])))
        sys.exit(1)

    # Transforme la chaîne en chemin canonique.
    chemin_depot = svn.core.svn_path_canonicalize(sys.argv[1])

    # Et c'est parti !
    parcourir_la_plus_recente_revision(chemin_depot)
```

Le même programme en C aurait besoin de faire appel aux réservoirs de mémoire d'APR. Mais Python gère l'utilisation de la mémoire automatiquement et l'interface Subversion vers Python se plie à cette convention. En C, vous auriez utilisé des types de données personnalisés (tels que ceux fournis par la bibliothèque APR) pour représenter la table de hachage des entrées et la liste des chemins, mais Python sait gérer nativement les tables de hachage (appelés « dictionnaires ») ainsi que les listes et possède une riche collection de fonctions pour travailler sur ces types de données. C'est pourquoi SWIG (avec l'aide de la couche d'interface vers les langages de programmation de Subversion, un peu modifiée) prend soin de faire correspondre ces types de données personnalisés aux types de données natifs du langage cible. On obtient ainsi une interface plus intuitive pour les utilisateurs de ce langage.

L'interface de Subversion vers Python peut également être utilisée pour effectuer des opérations dans la copie de travail. Dans la section précédente de ce chapitre, nous avons mentionné l'interface `libsvn_client` et le fait qu'elle a été conçue dans le seul but de faciliter l'écriture d'un client Subversion. L'Exemple 8.3, « Une version de status en Python » est un court exemple d'utilisation de cette bibliothèque via l'interface Python SWIG pour re-créer une version à petite échelle de la commande **svn status**.

Exemple 8.3. Une version de status en Python

```
#!/usr/bin/env python

"""Parcourir un répertoire d'une copie de travail en affichant les
informations d'état."""

import sys
import os.path
import getopt
import svn.core, svn.client, svn.wc

def generer_code_etat(etat):
    """Traduit la valeur d'état vers un code à un caractère en
    utilisant la même logique que le client Subversion en ligne de
    commande."""
    association_etat = { svn.wc.svn_wc_status_none        : ' ',
                         svn.wc.svn_wc_status_normal      : ' ',
                         svn.wc.svn_wc_status_added       : 'A',
                         svn.wc.svn_wc_status_missing     : '!',
                         svn.wc.svn_wc_status_incomplete  : '!',
                         svn.wc.svn_wc_status_deleted     : 'D',
                         svn.wc.svn_wc_status_replaced    : 'R',
                         svn.wc.svn_wc_status_modified    : 'M',
                         svn.wc.svn_wc_status_merged      : 'G',
                         svn.wc.svn_wc_status_conflicted  : 'C',
                         svn.wc.svn_wc_status_obstructed  : '~',
                         svn.wc.svn_wc_status_ignored     : 'I',
                         svn.wc.svn_wc_status_external    : 'X',
                         svn.wc.svn_wc_status_unversioned : '?',
                       }
    return association_etat.get(etat, '?')

def trouver_etat(chemin_copie_travail, verbeux):
    # Construit le "bâton" de contexte client.
    ctx = svn.client.svn_client_ctx_t()

    def _status_callback(path, etat):
        """Une fonction de renvoi ("callback") pour svn_client_status."""

        # Affiche le chemin, moins la partie déjà présente
        # dans la racine du parcours.
        text_status = generer_code_etat(etat.text_status)
        prop_status = generer_code_etat(etat.prop_status)
        print '%s%s  %s' % (text_status, prop_status, path)

    # Effectue le parcours des états, en utilisant _status_callback()
    # comme fonction de renvoi ("callback").
    revision = svn.core.svn_opt_revision_t()
    revision.type = svn.core.svn_opt_revision_head
    svn.client.svn_client_status2(chemin_copie_travail, revision,
                                  _status_callback,
                                  svn.core.svn_depth_infinity, verbeux,
                                  0, 0, 1, ctx)

def utilisation_et_sortie(code_erreur):
    """Affiche le message d'utilisation et sort avec CODE_ERREUR."""
    stream = code_erreur and sys.stderr or sys.stdout
    stream.write("""Usage: %s OPTIONS CHEMIN_COPIE_TRAVAIL
Options:
  --help, -h    : Affiche ce message d'aide.
  --verbose, -v : Affiche l'état de tous les objets, sans exception.
""" % (os.path.basename(sys.argv[0])))
    sys.exit(code_erreur)

if __name__ == '__main__':
```

```
# Analyse les options de la ligne de commande.
try:
    opts, args = getopt.getopt(sys.argv[1:], "hv", ["help", "verbose"])
except getopt.GetoptError:
    utilisation_et_sortie(1)
verbeux = 0
for opt, arg in opts:
    if opt in ("-h", "--help"):
        utilisation_et_sortie(0)
    if opt in ("-v", "--verbeux"):
        verbeux = 1
if len(args) != 1:
    utilisation_et_sortie(2)

# Transforme le chemin en chemin canonique.
chemin_copie_travail = svn.core.svn_path_canonicalize(args[0])

# Et c'est parti !
try:
    trouver_etat(chemin_copie_travail, verbeux)
except svn.core.SubversionException, e:
    sys.stderr.write("Erreur (%d): %s\n" % (e.apr_err, e.message))
    sys.exit(1)
```

Comme dans le cas de l'Exemple 8.2, « Utilisation de la couche dépôt en Python », ce programme voit sa mémoire gérée automatiquement et utilise en grande partie les types de données classiques de Python. L'appel de `svn_client_ctx_t()` est un peu trompeur parce que l'API publique de Subversion ne possède pas de telle fonction — la génération automatique de fonctions de SWIG (une sorte d'usine à fonctions pour transformer des structures C complexes vers un équivalent en Python) est à la peine. Notez également que le chemin passé au programme (tout comme dans le programme précédent) est mouliné par `svn_path_canonicalize()` car, *dans le cas contraire*, on s'expose à un arrêt rapide et brutal du programme par la bibliothèque C Subversion sous-jacente qui effectue des tests de conformité.

Résumé

L'une des plus formidables caractéristiques de Subversion n'est pas accessible avec le client en ligne de commande ou via d'autres outils. C'est le fait que Subversion a été conçu pour être modulaire et fournir une API publique stable de manière à ce que des développeurs tiers — tel que vous, peut-être — puissent écrire des logiciels qui pilotent les fonctionnalités du cœur de Subversion.

Dans ce chapitre, nous avons approfondi notre vision de l'architecture de Subversion, examiné ses couches logiques et décrit son API publique, celle-là même qu'utilisent les propres couches de Subversion pour communiquer entre elles. De nombreux développeurs ont imaginé des utilisations intéressantes de l'API Subversion, de la simple procédure automatique jusqu'à des systèmes de gestion de versions complètement différents, en passant par l'intégration de Subversion dans d'autres applications. Et *vous*, quelle utilisation originale en tirerez-vous ?

Chapitre 9. Références complètes de Subversion

Ce chapitre contient les références complètes de Subversion. Il comprend les références du client en ligne de commande (**svn**) et toutes ses sous-commandes, ainsi que des programmes d'admninistration du dépôt (**svnadmin** et **svnlook**) et leurs sous-commandes respectives.

Le client Subversion en ligne de commande : svn

Pour utiliser le client en ligne de commande, tapez **svn** puis la sous-commande que vous souhaitez utiliser[1] et toutes les options ou cibles que vous souhaitez traiter — la sous-commande et les options dosivent être indiquées dans un ordre bien particulier. Par exemple, voici quelques commandes valides utilisant **svn status** :

```
$ svn -v status
$ svn status -v
$ svn status -v mon-fichier
```

Vous trouverez beaucoup d'autres exemples relatifs aux commandes du client dans le Chapitre 2, *Utilisation de base* et relatifs aux propriétés dans la section intitulée « Propriétés ».

Options de svn

Bien que Subversion accepte différentes options en fonction des sous-commandes, toutes les options gardent leur signification quelle que soit la sous-commande avec laquelle elles sont utilisées — elles font partie d'un seul « espace de noms ». Par exemple, l'option `--verbose` (`-v`) signifie toujours « être plus bavard », quelle que soit la sous-commande utilisée.

Le client en ligne de commande **svn** renvoie généralement rapidement une erreur si vous lui indiquez une option qui ne s'applique pas à la sous-commande spécifiée. Cependant, depuis la version 1.5 de Subversion, plusieurs options sont acceptées par toutes les sous-commandes, même si elles n'ont pas d'effet dans certains cas. Elles sont regroupées dans la rubrique « options globales » des messages d'explication des commandes. Nous avons modifié le comportement antérieur afin de faciliter le travail des rédacteurs de scripts qui utilisent le client en ligne de commande. Ces options globales sont :

`--config-dir` *`REP`*
: Lire la configuration dans le répertoire spécifié plutôt qu'à l'endroit par défaut (`.subversion` dans le répertoire de l'utilisateur).

`--no-auth-cache`
: *Ne pas conserver* les éléments d'authentification (par exemple l'identifiant et le mot de passe) dans les répertoires de configuration de Subversion.

`--non-interactive`
: Pas de demande interactive. Les demandes interactives peuvent concerner, par exemple, les éléments d'authentification ou la résolution de conflits. Cette option est utile quand vous lancez Subversion dans un script totalement automatique et qu'il est plus pertinent de faire échouer Subversion plutôt que d'attendre une réponse interactive.

`--password` *`MDP`*
: Précise le mot de passe MDP à utiliser pour s'authentifier auprès du serveur Subversion. Si cette option n'est pas fournie ou si elle ne permet pas de s'authentifier correctement, Subversion vous demande, en tant que de besoin, le mot de passe de manière interactive.

`--username` *`NOM`*
: Précise le nom d'utilisateur NOM à utiliser pour s'authentifier auprès du serveur Subversion. Si cette option n'est pas fournie ou si elle ne permet pas de s'authentifier correctement, Subversion vous demande, en tant que de besoin, le nom d'utilisateur de manière interactive.

[1] En fait, l'option `--version` ne nécessite pas de sous-commande, nous verrons cela dans quelques minutes.

Le reste des options ne s'applique, et n'est reconnu, que par un sous-ensemble de commandes. Les voici :

`--accept` *`ACTION`*
Précise l'attitude à adopter pour la résolution automatique de conflits. Les valeurs possibles sont : `postpone`, `base`, `mine-full`, `theirs-full`, `edit` et `launch`.

`--auto-props`
Active les propriétés automatiques, remplaçant alors la directive `enable-auto-props` dans le fichier de configuration `config`.

`--change` (`-c`) *`ARG`*
Utilisée pour faire référence à un « changement » particulier (c-à-d. une révision). Cette option est du sucre syntaxique pour spécifier « -r ARG-1:ARG ».

`--changelist` *`ARG`*
N'opérer que sur les membres de la liste de changements nommée*`ARG`*. Vous pouvez utiliser cette option plusieurs fois pour spécifier un ensemble de listes de changements.

`--cl` *`ARG`*
Un alias pour l'option `--changelist`.

`--depth` *`ARG`*
Limiter l'opération à cette profondeur. *`ARG`* peut prendre les valeurs suivantes : `empty`, `files`, `immediates` ou `infinity`.

`--diff-cmd` *`CMD`*
Utiliser un programme externe pour afficher les différences entre fichiers. Quand **svn diff** est invoquée sans cette option, elle utilise le moteur interne de Subversion qui fournit un format unifié par défaut. Si vous voulez utiliser un programme externe, ajouter l'option `--diff-cmd`. Vous pouvez spécifier des options à ce programme externe à l'aide de l'option `--extensions` (voir plus loin dans cette section).

`--diff3-cmd` *`CMD`*
Spécifie une commande externe pour réaliser la fusion de fichiers.

`--dry-run`
Effectuer toutes les étapes de la commande, mais sans écrire effectivement les changements — que ce soit sur le disque local ou sur le dépôt.

`--editor-cmd` *`CMD`*
Spécifie un programme externe pour éditer l'entrée du journal de propagation ou une valeur de propriété. Voir la section `editor-cmd` dans la section intitulée « Fichier `config` » pour savoir comment spécifier un éditeur par défaut.

`--encoding` *`ENC`*
L'entrée du journal de propagation est encodée avec le jeu de caractères ENC. La valeur par défaut est l'encodage utilisé par votre système d'exploitation conformément à la régionalisation active. Vous devez spécifier explicitement l'encodage si votre entrée du journal de propagation utilise un autre encodage que la valeur par défaut.

`--extensions` (`-x`) *`ARGS`*
Spécifie un ou plusieurs arguments à passer à la commande diff externe. Cette option n'est valide qu'avec les commandes **svn diff** ou **svn merge**, et avec l'option `--diff-cmd`. Si vous voulez spécifier plusieurs arguments, vous devez mettre entre guillemets ces arguments (par exemple, **`svn diff --diff-cmd /usr/bin/diff -x "-b -E"`**).

`--file` (`-F`) *`NOM_FICHIER`*
Utiliser le contenu du fichier NOM_FICHIER pour la commande spécifiée, le traitement exact variant suivant les sous-commandes. Par exemple, **svn commit** utilise le contenu comme entrée du journal de propagation alors que **svn propset** l'utilise comme valeur de la propriété.

`--force`
Force l'exécution de l'opération. Subversion interdit certaines opérations en temps normal mais vous pouvez passer outre et indiquer à Subversion : « oui, je sais ce que je fais et les conséquences que cela peut avoir, alors laisse-moi faire ». En utilisant une métaphore, on pourrait dire que cela revient à travailler sur un circuit électrique sous tension ; si vous ne savez pas ce que vous faites, vous prendrez sûrement une bonne décharge.

`--force-log`
Valide la source de l'entrée du journal de propagation passée avec l'option `--message` (`-m`) ou `--file` (`-F`). Par défaut, Subversion renvoie une erreur si les paramètres de ces options ressemblent à des cibles de la sous-commande. Par exemple, si vous passez le chemin d'un fichier suivi en versions à l'option `--file` (`-F`), Subversion considère que vous avez fait une erreur, c'est-à-dire que le chemin indiqué devait plutôt être la cible de la commande et que vous avez oublié de fournir le nom d'un fichier — non suivi en versions — comme source pour l'entrée du journal de propagation. Pour confirmer votre intention et passer outre cette erreur, passez l'option `--force-log` aux sous-commandes qui prennent des entrées du journal de propagation comme arguments.

`--help` (`-h` ou `-?`)
Si elle est utilisée avec une ou plusieurs autres sous-commandes, affiche l'aide intégrée pour chacune d'elles. Utilisée seule, elle affiche l'aide générale relative au client en ligne de commande.

`--ignore-ancestry`
Ignorer l'héritage pour évaluer les différences (ne se fier qu'au chemin en tant que tel).

`--ignore-externals`
Ignorer les définitions de références externes ainsi que les copies de travail associées.

`--incremental`
Produire un affichage avec un format compatible pour la concaténation.

`--keep-changelists`
Ne pas détruire les listes de changements après la propagation.

`--keep-local`
Garder une copie locale d'un fichier ou répertoire (utilisée avec la commande **svn delete**).

`--limit` (`-l`) *NUM*
Afficher seulement les *NUM* premières entrées du journal de propagation.

`--message` (`-m`) *MESSAGE*
Indique que vous spécifiez, sur la ligne de commande à la suite de cette option, soit une entrée du journal de propagation, soit un commentaire de verrouillage. Par exemple :

```
$ svn commit -m "They don't make Sunday."
```

`--new` *ARG*
Utiliser *ARG* comme nouvelle cible (à utiliser avec la commande **svn diff**).

`--no-auto-props`
Désactive les propriétés automatiques, remplaçant alors la directive `enable-auto-props` du fichier de configuration `config`.

`--no-diff-deleted`
Ne pas afficher les différences pour les fichiers supprimés. Le comportement par défaut de **svn diff** est d'afficher les différences comme si vous aviez toujours le fichier et que celui-ci était vide.

`--no-ignore`
Afficher les fichiers qui seraient normalement omis dans la liste de statut en raison de la correspondance avec un motif de noms de fichiers présent dans l'option de configuration `global-ignores` ou dans la propriété `svn:ignore`. Voir la section intitulée « Fichier `config` » et la section intitulée « Occultation des éléments non suivis en versions » pour plus d'information.

`--no-unlock`
Ne pas déverrouiller les fichiers : le comportement par défaut de la propagation (**svn commit**) est de déverrouiller tous les fichiers propagés. Voir la section intitulée « Verrouillage » pour plus d'information.

`--non-recursive` (`-N`)
Dépréciée. Interdit à une sous-commande de descendre dans les sous-répertoires. La plupart des sous-commandes descend par défaut dans les sous-répertoires (comportement récursif), mais quelques unes — généralement celles qui peuvent supprimer ou annuler des modifications locales — ne le font pas.

`--notice-ancestry`
Prendre en compte l'héritage pour évaluer les différences.

`--old` *`ARG`*
Utiliser *`ARG`* comme ancien répertoire (utilisée avec la commande **svn diff**).

`--parents`
Créer (s'ils n'existent pas) et ajouter (s'ils ne sont pas suivis en versions) les sous-répertoires parents à une copie de travail locale ou à un dépôt dans le cadre d'une opération. Cette option est particulièrement utile pour créer automatiquement plusieurs répertoires quand aucun n'existe. Si elle concerne une URL, tous les répertoires seront créés par une seule et même propagation.

`--quiet` (`-q`)
N'afficher que ce qui est essentiel pendant une opération.

`--record-only`
Marquer les révisions comme fusionnées (pour une utilisation avec l'option `--revision`).

`--recursive` (`-R`)
Rend la sous-commande récursive. Beaucoup de sous-commandes sont récursives par défaut.

`--reintegrate`
Utilisée avec la sous-commande **svn merge**, fusionner tous les changements de l'URL source dans la copie de travail locale. Voir la section intitulée « Comment garder une branche synchronisée » pour les détails.

`--relocate` *`[CHEMIN...]`*
Utilisée avec la sous-commande **svn switch**, changer l'emplacement du dépôt auquel votre copie de travail fait référence. Ceci est utile si l'emplacement de votre dépôt a changé et que vous avez une copie de travail que vous voulez continuer à utiliser. Voir **svn switch** pour un exemple d'utilisation.

`--remove` *`ARG`*
Effacer l'association de *`ARG`* à une liste de changements.

`--revision` (`-r`) *`REV`*
Indique que vous allez fournir une révision (ou un intervalle de révisions) pour une opération particulière. Vous pouvez fournir des numéros de révisions, des mots-clés de révision ou des dates (encadrées par des accolades) comme arguments à l'option `revision`. Si vous voulez spécifier un intervalle de révisions, vous devez fournir deux révisions séparées par le symbole deux points (`:`). Par exemple :

```
$ svn log -r 1729
$ svn log -r 1729:HEAD
$ svn log -r 1729:1744
$ svn log -r {2001-12-04}:{2002-02-17}
$ svn log -r 1729:{2002-02-17}
```

Voir la section intitulée « Mots-clés de révision » pour plus d'information.

`--revprop`
Opérer sur une propriété de révision au lieu d'opérer sur une propriété de fichier ou de répertoire. Cette option requiert que vous passiez également une révision avec l'option `--revision` (`-r`).

`--set-depth` *`ARG`*
fixe (de manière permanente) la profondeur de la copie de travail à partir de ce répertoire à l'une des valeurs suivantes : `empty`, `files`, `immediates` ou `infinity`.

`--show-revs` *`ARG`*
Utilisée pour indiquer à **svn mergeinfo** de n'afficher qu'un ensemble de révisions, soit : `merged` ou `eligible`.

`--show-updates` (`-u`)
Afficher les informations relatives aux fichiers de la copie de travail qui ne sont plus à jour. Cela ne met pas à jour les fichiers — cela ne fait qu'afficher le nom des fichiers qui seront mis à jour lors de la prochaine commande **svn update**.

`--stop-on-copy`
Indique à une sous-commande Subversion qui parcourt l'héritage d'une ressource suivie en versions de s'arrêter lorsque une copie — c'est-à-dire un emplacement dans l'héritage où la ressource a été copiée depuis un autre endroit dans le dépôt — est trouvée.

`--strict`
Utiliser une sémantique stricte, notion plutôt vague sauf pour certaines sous-commandes spécifiques (pour ne pas la citer, **svn propget**).

`--targets` *`NOM_FICHIER`*
Prendre la liste de fichiers cibles pour l'opération dans le fichier NOM_FICHIER au lieu de prendre les fichiers fournis dans la ligne de commande.

`--use-merge-history` (`-g`)
Utiliser ou afficher des informations supplémentaires sur l'historique de la fusion.

`--verbose` (`-v`)
Afficher autant d'informations que possible durant l'exécution de la sous-commande. Cela se traduit par l'affichage de champs supplémentaires, d'informations détaillées à propos de chaque fichier ou d'informations complémentaires relatives aux actions menées.

`--version`
Afficher les informations sur la version du client. Ces informations comprennent le numéro de version du client ainsi que la liste de tous les modules d'accès à un dépôt disponibles. Avec l'option `--quiet` (`-q`), elle ne fait qu'afficher le numéro de version dans un format compact.

`--with-all-revprops`
Utilisée avec l'option `--xml` de **svn log**, récupèrer et afficher toutes les propriétés de révisions.

`--with-revprop` *`ARG`*
Quand elle est utilisée avec une commande qui écrit dans le dépôt, fixer une propriété de révision, en utilisant le format *`NOM=VALEUR`*, en affectant *`VALEUR`* à *`NOM`*. Quand elle est utilisée avec **svn log** dans le mode `--xml`, afficher la valeur de *`ARG`*.

`--xml`
Produire un affichage au format XML.

Sous-commandes svn

Voici les différentes sous-commandes du programme **svn**. Afin de rester concis, nous passons sous silence les options globales (expliquées dans la section intitulée « Options de svn ») dans les descriptions suivantes.

Nom

svn add — Ajouter des fichiers, répertoires et liens symboliques.

Synopsis

```
svn add CHEMIN...
```

Description

Prévoit l'ajout au dépôt des fichiers, répertoires et liens symboliques de la copie de travail. Ils seront transférés et ajoutés au dépôt lors de la prochaine propagation. Si vous ajoutez quelque chose et que vous changez d'avis avant de faire une propagation, vous pouvez annuler l'ajout en utilisant **svn revert**.

Noms alternatifs

Aucun.

Modifie

Copie de travail.

Accède au dépôt

Non.

Options

```
--auto-props
--depth ARG
--force
--no-auto-props
--no-ignore
--parents
--quiet (-q)
--targets NOM_FICHIER
```

Exemples

Pour ajouter un fichier à votre copie de travail :

```
$ svn add machin.c
A         machin.c
```

Lors de l'ajout d'un répertoire, le comportement par défaut de **svn add** est récursif :

```
$ svn add rep-test
A         rep-test
A         rep-test/a
A         rep-test/b
A         rep-test/c
A         rep-test/d
```

Vous pouvez ajouter un répertoire sans inclure son contenu :

```
$ svn add --depth=empty autre-rep
```

```
A         autre-rep
```

Normalement, la commande **svn add *** ignore tous les répertoires qui sont déjà suivis en versions. Parfois, cependant, vous souhaitez ajouter tous les objets non encore suivis en versions de la copie de travail, y compris ceux qui sont cachés au plus profond des sous-répertoires. L'option `--force` oblige alors la commande **svn add** à explorer les répertoires récursivement :

```
$ svn add * --force
A         machin.c
A         un-rep/bidule.c
A  (bin)  autre-rep/docs/truc.doc
…
```

Nom

svn blame — Afficher les informations de révisions et d'auteurs en plus du contenu pour les fichiers ou URL spécifiés.

Synopsis

```
svn blame CIBLE[@REV]...
```

Description

Affiche les informations de révisions et d'auteur en plus du contenu pour les fichiers ou URL spécifiés. Chaque ligne de texte est annotée en début par l'identifiant de l'auteur et le numéro de révision pour le dernier changement correspondant à la ligne.

Noms alternatifs

praise, **annotate**, **ann**

Modifie

Rien.

Accède au dépôt

Oui.

Options

```
--extensions (-x) ARG
--force
--incremental
--revision (-r) ARG
--use-merge-history (-g)
--verbose (-v)
--xml
```

Exemples

Si vous voulez voir les auteurs de chaque ligne de votre fichier source `lisezmoi.txt` de votre dépôt test :

```
$ svn blame http://svn.red-bean.com/repos/test/lisezmoi.txt
     3      sally Ceci est un fichier LISEZMOI.
     5      harry Vous devez le lire.
```

Même si **svn blame** dit que c'est Harry qui a modifié le fichier `lisezmoi.txt` dans la révision 5, vous devez examiner exactement les changements apportés par la révision pour être sûr que Harry a modifié le *contenu* de la ligne — peut-être n'a-t-il fait que modifier des espaces.

Si vous utilisez l'option `--xml`, vous obtenez une sortie XML décrivant les informations d'auteurs et de révisions mais pas le contenu des lignes lui-même :

```
$ svn blame --xml http://svn.red-bean.com/repos/test/lisezmoi.txt
<?xml version="1.0"?>
<blame>
<target
   path="lisezmoi.txt">
<entry
   line-number="1">
<commit
```

```
   revision="3">
<author>sally</author>
<date>2008-05-25T19:12:31.428953Z</date>
</commit>
</entry>
<entry
   line-number="2">
<commit
   revision="5">
<author>harry</author>
<date>2008-05-29T03:26:12.293121Z</date>
</commit>
</entry>
</target>
</blame>
```

Nom

svn cat — Afficher le contenu des fichiers ou URL spécifiés.

Synopsis

```
svn cat CIBLE[@REV]...
```

Description

Affiche le contenu des fichiers ou URL spécifiés. Pour afficher la liste du contenu des répertoires, voir **svn list** plus loin dans ce chapitre.

Noms alternatifs

Aucun.

Modifie

Rien.

Accède au dépôt

Oui.

Options

```
--revision (-r) REV
```

Exemples

Si vous voulez voir le contenu de `lisezmoi.txt` dans votre dépôt sans pour autant l'extraire dans votre copie de travail :

```
$ svn cat http://svn.red-bean.com/repos/test/lisezmoi.txt
Ceci est un fichier LISEZMOI.
Vous devez le lire.
```

Si la copie de travail n'est pas à jour (ou si vous avez des modifications locales) et que vous voulez voir la révision `HEAD` d'un fichier de la copie de travail, **svn cat -r HEAD *NOM_FICHIER*** récupère automatiquement la révision `HEAD` du chemin spécifié :

```
$ cat machin.c
Ce fichier est dans la copie de travail locale
et j'y ai apporté des modifications.

$ svn cat -r HEAD machin.c
Dernière version toute fraîche en provenance du dépôt!
```

Nom

svn changelist — Associer (ou dissocier) des fichiers d'une liste de changements.

Synopsis

```
changelist NOM_CL CIBLE...

changelist --remove CIBLE...
```

Description

Utilisée pour répartir les fichiers d'une copie de travail dans des listes de changements (groupes logiques dotés d'un nom) dans le but de faciliter le travail de l'utilisateur travaillant sur plusieurs ensembles de fichiers dans une seule copie de travail.

Noms alternatifs

cl

Modifie

Copie de travail.

Accède au dépôt

Non.

Options

```
--changelist ARG
--depth ARG
--quiet (-q)
--recursive (-R)
--remove
--targets ARG
```

Exemple

Edite trois fichiers et les ajoute à une liste de changements. Puis propage uniquement les fichiers associés à cette liste de changements :

```
$ svn cl pb1729 machin.c bidule.c truc.c
Le chemin 'machin.c' fait maintenant partie de la liste de changement 'pb1729'.
Le chemin 'bidule.c' fait maintenant partie de la liste de changement 'pb1729'.
Le chemin 'truc.c' fait maintenant partie de la liste de changement 'pb1729'.

$ svn status
A      un-autre-fichier.c
A      test/un-test.c

---  Liste de changements 'pb1729' :
A      machin.c
A      bidule.c
A      truc.c

$ svn commit --changelist pb1729 -m "Règle le problème 1729."
Envoi         bidule.c
Envoi         truc.c
Envoi         machin.c
```

```
Transmission des données .
Révision 2 propagée.

$ svn status
A      un-autre-fichier.c
A      test/un-test.c
```

Notez que seuls les fichiers associés à la liste de changements *pb1729* ont été propagés.

Nom

svn checkout — Extraire une copie de travail à partir d'un dépôt.

Synopsis

```
svn checkout URL[@REV]... [CHEMIN]
```

Description

Extrait une copie de travail à partir d'un dépôt. Si *`CHEMIN`* n'est pas spécifié, le nom de fichier de l'URL (*basename*) est utilisé comme destination. Si plusieurs URL sont fournies, chacune est extraite dans un sous-répertoire de *`CHEMIN`*, avec comme nom de sous-répertoire le nom de fichier de l'URL.

Noms alternatifs

co

Modifie

Crée une copie de travail.

Accède au dépôt

Oui.

Options

```
--depth ARG
--force
--ignore-externals
--quiet (-q)
--revision (-r) REV
```

Exemples

Extrait une copie de travail dans le répertoire `mien` :

```
$ svn checkout file:///var/svn/depot/test mien
A  mien/a
A  mien/b
A  mien/c
A  mien/d
Révision 20 extraite.
$ ls
mien
```

Extrait deux répertoires différents vers deux copies de travail séparées :

```
$ svn checkout file:///var/svn/depot/test  file:///var/svn/depot/quizz
A  test/a
A  test/b
A  test/c
A  test/d
Révision 20 extraite.
A  quizz/l
A  quizz/m
Révision 13 extraite.
```

```
$ ls
quizz  test
```

Extrait deux répertoires différents vers deux copies de travail séparées, mais les place toutes les deux dans un répertoire appelé `copies-de-travail` :

```
$ svn checkout file:///var/svn/depot/test  file:///var/svn/depot/quiz
copies-de-travail
A  copies-de-travail/test/a
A  copies-de-travail/test/b
A  copies-de-travail/test/c
A  copies-de-travail/test/d
Révision 20 extraite.
A  copies-de-travail/quizz/l
A  copies-de-travail/quizz/m
Révision 13 extraite.
$ ls
copies-de-travail
```

Si vous interrompez l'extraction (ou si n'importe quoi interromp l'extraction, comme la perte du réseau par exemple), vous pouvez recommencer l'extraction en invoquant exactement la même commande ou en mettant à jour la copie incomplète :

```
$ svn checkout file:///var/svn/depot/test mien
A  mien/a
A  mien/b
^C
svn: The operation was interrupted
svn: caught SIGINT

$ svn checkout file:///var/svn/depot/test mien
A  mien/c
^C
svn: The operation was interrupted
svn: caught SIGINT

$ svn update mien
A  mien/d
Révision 20 extraite.
```

Si vous voulez extraire une révision qui n'est pas la plus récente, vous pouvez le faire à l'aide de l'option `--revision` (`-r`) de la commande **svn checkout** :

```
$ svn checkout -r 2 file:///var/svn/depot/test mien
A  mien/a
Révision 2 extraite.
```

Nom

svn cleanup — Nettoyer récursivement la copie de travail.

Synopsis

```
svn cleanup [CHEMIN...]
```

Description

Nettoie récursivement la copie de travail en enlevant les verrous et en reprenant les opérations en cours. Si jamais vous obtenez une erreur `copie de travail verrouillée` (`working copy locked` si l'affichage de votre Subversion est en anglais), lancez cette commande pour supprimer les verrous et remettre votre copie de travail dans un état utilsable.

Si, pour quelque raison que ce soit, une commande **svn update** échoue suite à un problème de programme diff externe (par exemple en raison d'une entrée utilisateur incorrecte ou d'un problème réseau), passez l'option `--diff3-cmd` pour autoriser **cleanup** à terminer l'opération de fusion avec votre programme diff externe. Vous pouvez aussi spécifier un répertoire de configuration particulier avec l'option `--config-dir` , mais vous ne devriez avoir à utiliser ces options qu'extrêmement rarement.

Noms alternatifs

Aucun.

Modifie

Copie de travail.

Accède au dépôt

Non.

Options

```
--diff3-cmd CMD
```

Exemples

Il n'y a pas vraiment matière à exemple ici, la commande **svn cleanup** ne produisant aucun affichage. Si vous ne fournissez pas de paramètre *`CHEMIN`*, alors « `.` » est utilisé :

```
$ svn cleanup
$ svn cleanup /var/svn/copie-de-travail
```

Nom

svn commit — Envoyer les modifications de la copie de travail vers le dépôt.

Synopsis

```
svn commit [CHEMIN...]
```

Description

Envoie les modifications de la copie de travail vers le dépôt. Si vous ne fournissez pas d'entrée du journal avec votre propagation en utilisant soit l'option `--file`, soit l'option `--message`, **svn** lance votre éditeur de texte pour que vous en rédigiez une. Lisez le paragraphe relatif à la liste `editor-cmd` dans la section intitulée « Fichier `config` ».

svn commit propage tous les jetons de verrouillage qu'il trouve et déverrouille tous les verrous sur les *`CHEMIN`*s propagés (récursivement) à moins que l'option `--no-unlock` ne soit spécifiée.

Si vous commencez une propagation et que subversion lance votre éditeur de texte pour rédiger l'entrée du journal de propagation, vous pouvez toujours abandonner la propagation. Si vous voulez l'abandonner, quitter simplement l'éditeur sans sauvegarder le message de propagation ; Subversion vous demande alors si vous voulez abandonner, continuer sans rien écrire dans le journal de propagation ou éditer à nouveau le message.

Noms alternatifs

ci (raccourci pour *check in* ; pas **co**, qui est un alias pour la sous-commande **checkout**).

Modifie

Copie de travail ; dépôt.

Accède au dépôt

Oui.

Options

```
--changelist ARG
--depth ARG
--editor-cmd ARG
--encoding ENC
--file (-F) FICHIER
--force-log
--keep-changelists
--message (-m) TEXTE
--no-unlock
--quiet (-q)
--targets NOM_FICHIER
--with-revprop ARG
```

Exemples

Propage une simple modification sur un fichier avec l'entrée du journal de propagation indiquée dans la ligne de commande et la cible implicite étant le répertoire courant (« `.` ») :

```
$ svn commit -m "ajout de la Foire Aux Questions."
Envoi        a
Transmission des données .
```

```
Révision 3 propagée.
```

Propage une modification sur le fichier `machin.c` (spécifié explicitement sur la ligne de commande) avec l'entrée du journal de propagation dans le fichier nommé `msg` :

```
$ svn commit -F msg machin.c
Envoi        machin.c
Transmission des données .
Révision 5 propagée.
```

Si vous voulez utiliser un fichier suivi en versions pour votre entrée du journal de propagation avec l'option `--file`, vous devez également spécifier l'option `--force-log` :

```
$ svn commit --file fichier-versionné.txt machin.c
svn: Le fichier de l'entrée du journal est versionné; forcer avec '--force-log'

$ svn commit --force-log --file fichier-versionné.txt machin.c
Envoi        machin.c
Transmission des données .
Révision 6 propagée.
```

Pour propager un fichier à supprimer :

```
$ svn commit -m "Fichier 'c' supprimé."
Suppression    c

Révision 7 propagée.
```

Nom

svn copy — Copier un ou plusieurs fichiers ou répertoires dans une copie de travail ou dans le dépôt.

Synopsis

```
svn copy SRC[@REV]... DST
```

Description

Copie un ou plusieurs fichiers ou répertoires dans une copie de travail ou dans le dépôt. Lors de la copie de plusieurs sources, elles seront ajoutées en tant que fils de *DST*, qui doit être un répertoire. *SRC* et *DST* peuvent être dans la copie de travail(WC) ou une URL :

WC # WC
: Copie et prévoit pour ajout un élément (avec reprise de l'historique).

WC # URL
: Propage immédiatement une copie de WC vers URL.

URL # WC
: Extrait l'URL dans WC et le prévoit pour ajout.

URL # URL
: Copie complète côté serveur. Utilisé habituellement pour créer des branches ou des étiquettes.

S'il n'y pas de piquet de révision (c-à-d. *@REV*) spécifié, la révision BASE est utilisée par défaut pour les fichiers copiés depuis la copie de travail et la révision HEAD est utilisée par défaut pour les fichiers copiés depuis une URL.

Vous ne pouvez copier que des fichiers provenant d'un même dépôt. Subversion ne supporte pas les copies inter-dépôts.

Noms alternatifs

cp

Modifie

Le dépôt si la destination est une URL ; la copie de travail si la destination est un chemin WC.

Accède au dépôt

Oui, si la source ou la destination sont dans le dépôt ou s'il y a besoin de déterminer le numéro de révision de la source.

Options

```
--editor-cmd EDITEUR
--encoding ENC
--file (-F) FICHIER
--force-log
--message (-m) TEXTE
--parents
--quiet (-q)
--revision (-r) REV
--with-revprop ARG
```

Exemples

Copie un élément dans la copie de travail (la copie est prévue pour ajout ; rien n'est écrit dans le dépôt avant que vous n'effectuyez une propagation) :

```
$ svn copy machin.txt bidule.txt
A         bidule.txt
$ svn status
A  +   bidule.txt
```

Copie plusieurs fichiers de la copie de travail dans un sous-répertoire :

```
$ svn cp bat.c truc.c qux.c src
A         src/bat.c
A         src/truc.c
A         src/qux.c
```

Copie la révision 8 de `bat.c` dans la copie de travail sous un autre nom :

```
$ svn cp -r 8 bat.c un-vieux-bat.c
A         un-vieux-bat.c
```

Copie un élément de la copie de travail vers une URL dans le dépôt (ceci provoque immédiatement une propagation, c'est pourquoi vous devez fournir une entrée dans le journal de propagation) :

```
$ svn copy proche.txt file:///var/svn/depot/test/tres-loin.txt -m "Copie distante."

Révision 8 propagée.
```

Copie un élément du dépôt vers la copie de travail (ceci ne fait que prévoir la copie — rien ne se passe sur le dépôt tant que vous ne faites pas de propagation) :

```
$ svn copy file:///var/svn/depot/test/tres-loin -r 6 pres-d-ici
A         pres-d-ici
```

C'est la méthode recommandée pour ressuciter un fichier mort dans le dépôt !

Et enfin, copie entre deux URL :

```
$ svn copy file:///var/svn/depot/test/tres-loin \
           file:///var/svn/depot/test/la-bas -m "copie distante."

Révision 9 propagée.



$ svn copy file:///var/svn/depot/test/trunk \
           file:///var/svn/depot/test/tags/0.6.32-pre-officielle \
           -m "création d'une étiquette"

Révision 12 propagée.
```

C'est la méthode la plus facile pour « étiqueter » une révision dans le dépôt — faites juste **svn copy** de cette révision (généralement, c'est HEAD) vers votre répertoire `tags`.

Et ne vous inquietez pas si vous avez oublié de créer votre étiquette — vous pouvez toujours indiquer une vieille révision et étiqueter quand bon vous semble :

```
$ svn copy -r 11 file:///var/svn/depot/test/trunk \
           file:///var/svn/depot/test/tags/0.6.32-pre-officielle \
           -m "Oublié de créer l'étiquette à la révision 11"

Révision 13 propagée.
```

Nom

svn delete — Supprimer un élément de la copie de travail ou du dépôt.

Synopsis

```
svn delete CHEMIN...

svn delete URL...
```

Description

Les éléments spécifiés par *`CHEMIN`* sont prévus pour être supprimés lors de la prochaine propagation. Les fichiers (et répertoires) qui n'ont pas été propagés sont immédiatement supprimés de la copie de travail à moins que l'option `--keep-local` ne soit spécifiée. La commande ne supprime ni ne modifie aucun élément qui n'est pas suivi en versions ; utilisez l'option `--force` pour passer outre ce comportement.

Les éléments sous forme d'URL sont supprimés du dépôt par une propagation immédiate. Si plusieurs URL sont passées en paramètre, la propagation est atomique.

Noms alternatifs

del, **remove**, **rm**

Modifie

Copie de travail si l'opération porte sur un chemin local ; le dépôt si l'opération porte sur des URL.

Accède au dépôt

Seulement si l'opération porte sur des URL

Options

```
--editor-cmd EDITEUR
--encoding ENC
--file (-F) FICHIER
--force
--force-log
--keep-local
--message (-m) TEXTE
--quiet (-q)
--targets NOM_FICHIER
--with-revprop ARG
```

Exemples

Utiliser **svn** pour supprimer un fichier de la copie de travail supprime la copie locale du fichier mais, surtout, cela prévoit de supprimer le fichier du dépôt. Lors de la prochaine propagation, le fichier est supprimé du dépôt.

```
$ svn delete mon-fichier
D         mon-fichier

$ svn commit -m "Supprimé le fichier 'mon-fichier'."
Suppression     mon-fichier
Transmission des données .
Révision 14 propagée.
```

Supprimer une URL, en revanche, est immédiat. C'est pourquoi vous devez fournir une entrée dans le journal de propagation :

```
$ svn delete -m "Suppression du fichier 'ton-fichier'" \
             file:///var/svn/depot/test/ton-fichier

Révision 15 propagée.
```

Voici un exemple montrant comment forcer la suppression d'un fichier qui comporte des modifications locales :

```
$ svn delete la-bas
svn: Utiliser --force pour passer cette restriction
svn: 'la-bas' a des modifications locales

$ svn delete --force la-bas
D         la-bas
```

Nom

svn diff — Afficher les différences entre deux révisions ou chemins.

Synopsis

```
diff [-c M | -r N[:M]] [CIBLE[@REV]...]

diff [-r N[:M]] --old=VCIBLE[@VREV] [--new=NCIBLE[@NREV]] [CHEMIN...]

diff VURL[@VREV] NURL[@NREV]
```

Description

- Affiche les différences entre deux chemins. Vous pouvez utiliser **svn diff** selon différentes formes :
- Tapez juste **svn diff** pour afficher les modifications faites localement à un fichier.
- Affiche les modifications faites à *`CIBLE`* entre deux révisions telles qu'elles apparaissent lors de la révision *`REV`*. Les *`CIBLE`*s peuvent faire référence à des chemins de la copie de travail ou à des *`URL`*. Si les *`CIBLE`*s font référence à des chemins de la copie de travail, alors *`N`* vaut par défaut BASE et *`M`* à la copie de travail ; si les *`CIBLE`*s font référence à des *`URL`*, alors *`N`* doit être spécifié et *`M`* vaut par défaut HEAD. L'option `-c M` est équivalente à `-r N:M` avec `N = M-1`. L'utilisation de `-c -M` correspond à l'inverse : `-r M:N` avec `N = M-1`.
- Affiche les différences entre *`VCIBLE`* vue en *`VREV`* et *`NCIBLE`* vue en *`NREV`*. Les *`CHEMIN`*s, s'ils sont spécifiés, sont relatifs à *`VCIBLE`* et *`NCIBLE`* et restreignent l'affichage des différences à ces chemins. *`VCIBLE`* et *`NCIBLE`* peuvent faire référence à des chemins de la copie de travail ou des *`URL[@REV]`*. *`NCIBLE`* vaut par défaut *`VCIBLE`* si elle n'est pas spécifiée. `-r N` entraine que *`VREV`* vaut par défaut `N` ; `-r N:M` entraine que *`VREV`* vaut par défaut *`N`* et *`NREV`* vaut par défaut *`M`*.

`svn diff VURL[@VREV] NURL[@NREV]` est un raccourci pour **`svn diff --old=VURL[@VREV] --new=NURL[@NREV].`**

`svn diff -r N:M URL` est un raccourci pour **`svn diff -r N:M --old=URL --new=URL`**.

`svn diff [-r N[:M]] URL1[@N] URL2[@M]` est un raccourci pour **`svn diff [-r N[:M]] --old=URL1 --new=URL2`**.

Si *`CIBLE`* est une URL, alors les révisions `N` et `M` peuvent être passées via l'option `--revision` ou en utilisant la notation « @ » décrite précédemment.

Si *`CIBLE`* est un chemin de la copie de travail, le comportement par défaut (sans option `--revision`) est d'afficher les différences entre les versions « de base » et la copie de travail de *`CIBLE`*. Si l'option `--revision` est spécifiée dans cette forme, cela correspond à :

`--revision N:M`
: Le serveur compare *`CIBLE@N`* et *`CIBLE@M`*.

`--revision N`
: Le client compare *`CIBLE@N`* à la copie de travail.

Si la syntaxe alternative est utilisée, le serveur compare *`URL1`* et *`URL2`* aux révisions *`N`* et *`M`*, respectivement. Si l'une ou l'autre des révisions *`N`* et *`M`* n'est pas spécifiée, la valeur HEAD est prise par défaut.

Par défaut, **svn diff** ignore l'héritage des fichiers et ne fait que comparer le contenu des fichiers spécifiés. Si vous utilisez l'option `--notice-ancestry`, l'héritage des chemins en question sera pris en compte lors de la comparaison des révisions

(c-à-d. que si vous lancez **svn diff** sur deux fichiers dont le contenu est identique mais qui ont des héritages différents, l'affichage correspond à un fichier supprimé puis ajouté de nouveau).

Noms alternatifs

di

Modifie

Rien.

Accède au dépôt

Pour obtenir les différences relatives à toutes les révisions, sauf BASE dans la copie de travail.

Options

```
--change (-c) ARG
--changelist ARG
--depth ARG
--diff-cmd CMD
--extensions (-x) "ARGS"
--force
--new ARG
--no-diff-deleted
--notice-ancestry
--old ARG
--revision (-r) ARG
--summarize
--xml
```

Exemples

Compare BASE et la copie de travail (c'est l'usage le plus fréquent de **svn diff**):

```
$ svn diff COMMITTERS
Index: COMMITTERS
===================================================================
--- COMMITTERS (révision 4404)
+++ COMMITTERS (copie de travail)
```

Regardons ce qui a changé dans le fichier COMMITTERS depuis la révision 9115 :

```
$ svn diff -c 9115 COMMITTERS
Index: COMMITTERS
===================================================================
--- COMMITTERS (révision 3900)
+++ COMMITTERS (copie de travail)
```

Regardons ce qui change dans la copie de travail par rapport aux anciennes révisions :

```
$ svn diff -r 3900 COMMITTERS
Index: COMMITTERS
===================================================================
--- COMMITTERS (révision 3900)
+++ COMMITTERS (copie de travail)
```

Compare la révision 3000 à la révision 3500 en utilisant la syntaxe « @ » :

```
$ svn diff http://svn.collab.net/repos/svn/trunk/COMMITTERS@3000 \
           http://svn.collab.net/repos/svn/trunk/COMMITTERS@3500
Index: COMMITTERS
===================================================================
--- COMMITTERS (révision 3000)
+++ COMMITTERS (révision 3500)
…
```

Compare la révision 3000 à la révision 3500 en utilisant la notation pour les intervalles (on ne passe qu'une seule fois l'URL dans ce cas) :

```
$ svn diff -r 3000:3500 http://svn.collab.net/repos/svn/trunk/COMMITTERS
Index: COMMITTERS
===================================================================
--- COMMITTERS (révision 3000)
+++ COMMITTERS (révision 3500)
```

Compare la révision 3000 à la révision 3500 de tous les fichiers dans `trunk` en utilisant la notation pour les intervalles :

```
$ svn diff -r 3000:3500 http://svn.collab.net/repos/svn/trunk
```

Compare la révision 3000 à la révision 3500 de seulement trois fichiers dans `trunk` en utilisant la notation pour les intervalles :

```
$ svn diff -r 3000:3500 --old http://svn.collab.net/repos/svn/trunk \
           COMMITTERS README HACKING
```

Si vous avez une copie de travail, vous pouvez obtenir ces différences sans taper de longues URL :

```
$ svn diff -r 3000:3500 COMMITTERS
Index: COMMITTERS
===================================================================
--- COMMITTERS (révision 3000)
+++ COMMITTERS (révision 3500)
```

Utilisez l'option `--diff-cmd` *`CMD`* `-x` pour passer directement des paramètres au programme diff externe :

```
$ svn diff --diff-cmd /usr/bin/diff -x "-i -b" COMMITTERS
Index: COMMITTERS
===================================================================
0a1,2
> This is a test
>
```

Pour finir, vous pouvez utiliser l'option `--xml` avec l'option `--summarize` pour afficher un document XML décrivant les modifications apportées entre les révisions, mais pas le contenu des différences en tant que tel :

```
$ svn diff --summarize --xml http://svn.red-bean.com/repos/test@r2 \
           http://svn.red-bean.com/repos/test
<?xml version="1.0"?>
<diff>
<paths>
<path
```

```
   props="none"
   kind="file"
   item="modified"&gt;http://svn.red-bean.com/repos/test/sandwich.txt&lt;/chemin&gt;
<path
   props="none"
   kind="file"
   item="deleted"&gt;http://svn.red-bean.com/repos/test/burrito.txt&lt;/chemin&gt;
<path
   props="none"
   kind="dir"
   item="added"&gt;http://svn.red-bean.com/repos/test/snacks&lt;/chemin&gt;
</chemins>
</diff>
```

Nom

svn export — Créer une copie non versionnée d'une arborescence.

Synopsis

```
svn export [-r REV] URL[@PEGREV] [CHEMIN]

svn export [-r REV] CHEMIN1[@PEGREV] [CHEMIN2]
```

Description

La première forme exporte une copie non versionnée d'un dépôt spécifié par *`URL`* — à la révision *`REV`* si elle est spécifiée ; sinon, à `HEAD`, vers *`CHEMIN`*. Si *`CHEMIN`* est omis, le nom de fichier (*basename*) de *`URL`* est utilisé comme nom du répertoire local.

La deuxième forme exporte une copie non versionnée de la copie de travail spécifiée par *`CHEMIN1`* vers *`CHEMIN2`*. Toutes les modifications locales sont préservées mais les fichiers qui ne sont pas suivis en versions ne sont pas copiés.

Noms alternatifs

Aucun.

Modifie

Disque local.

Accède au dépôt

Seulement si l'export concerne une URL.

Options

```
--depth ARG
--force
--ignore-externals
--native-eol EOL
--quiet (-q)
--revision (-r) REV
```

Exemples

Exporte depuis la copie de travail (n'affiche pas tous les fichiers et répertoires) :

```
$ svn export a-wc mon-export
Fin d'exportation.
```

Exporte directement depuis le dépôt (affiche chaque fichier et répertoire) :

```
$ svn export file:///var/svn/depot mon-export
A  mon-export/test
A  mon-export/quizz
…
Exporté à la révision 15.
```

Lorsque vous produisez une archive spécifique à un système d'exploitation donné, il peut être utile de faire l'export en utilisant

le caractère de fin de ligne correspondant au système d'exploitation. L'option `--native-eol` est prévue à cet effet, mais elle ne s'applique qu'aux fichiers qui possèdent la propriété `svn:eol-style =native`. Par exemple, pour exporter une arborescence avec le marqueur de fin de ligne CRLF (convient pour une archive `.zip` Windows), tapez :

```
$ svn export file:///var/svn/depot mon-export --native-eol CRLF
A  mon-export/test
A  mon-export/quizz
…
Exporté à la révision 15.
```

Vous pouvez spécifier `LR`, `CR` ou `CRLF` comme marqueur de fin de ligne à l'option `--native-eol`.

Nom

svn help — À l'aide !

Synopsis

```
svn help [SOUS-COMMANDE...]
```

Description

C'est votre meilleure amie quand vous utilisez Subversion et que ce livre n'est pas à portée de main !

Noms alternatifs

?, **h**

Les options `-?`, `-h` et `--help` ont le même effet que l'utilisation de la sous-commande **help**.

Modifie

Rien.

Accède au dépôt

Non.

Options

Nom

svn import — Propager un fichier ou une arboresence non versionnée dans un dépôt.

Synopsis

```
svn import [CHEMIN] URL
```

Description

Propage récursivement une copie de *`CHEMIN`* vers *`URL`*. Si *`CHEMIN`* est omis, « `.` » est la valeur par défaut. Les répertoires parents sont créés dans le dépôt autant que nécessaire. Les éléments non suivis en versions tels que les descripteurs de périphériques et les tubes de communication (*pipe*) sont ignorés même si l'option `--force` est spécifiée.

Noms alternatifs

Aucun.

Modifie

Dépôt.

Accède au dépôt

Oui.

Options

```
--auto-props
--depth ARG
--editor-cmd EDITEUR
--encoding ENC
--file (-F) FICHIER
--force
--force-log
--message (-m) TEXTE
--no-auto-props
--no-ignore
--quiet (-q)
--with-revprop ARG
```

Exemples

Import du répertoire local `mon-projet` vers `trunk/misc` dans le dépôt. Le répertoire `trunk/misc` n'a pas besoin d'exister avant l'import — **svn import** créera récursivement les répertoires pour vous.

```
$ svn import -m "Nouvel import" mon-projet \
             http://svn.red-bean.com/repos/trunk/misc
Ajout        mon-projet/echantillon.txt
…
Transmission des données .........
Révision 16 propagée.
```

Attention, cette commande *ne créera pas* de répertoire `mon-projet` dans le dépôt. Si vous voulez le faire, ajoutez simplement `mon-projet` à la fin de l'URL :

```
$ svn import -m "Nouvel import" mon-projet \
            http://svn.red-bean.com/repos/trunk/misc/mon-projet
```

```
Ajout       mon-projet/echantillon.txt
…
Transmission des données .........
Révision 16 propagée.
```

Après avoir importé des données, notez que l'arborescence originale *n'est pas* placée en suivi de versions. Pour commencer à travailler, vous devez toujours extraire une copie de travail de l'arborescence à l'aide de la commande **svn checkout**.

Nom

svn info — Afficher les informations sur des éléments locaux ou distants.

Synopsis

```
svn info [CIBLE[@REV]...]
```

Description

Affiche les informations sur des éléments de la copie de travail ou du dépôt. Les informations susceptibles d'être affichées sont, pour les deux formess :

- Chemin
- Nom
- URL
- Racine du dépôt
- UUID du dépôt
- Révision
- Type de nœud
- Auteur de la dernière modification
- Révision de la dernière modification
- Date de la dernière modification
- Nom de verrou
- Propriétaire du verrou
- Verrou créé (date)
- Verrou expire (date)

Les informations disponibles pour les chemins faisant référence à la copie de travail sont :

- Tâche programmée
- Copié de l'URL
- Copié de la révision
- Texte mis à jour
- Properties last updated
- Somme de contrôle
- Conflict previous base file
- Conflict previous working file
- Conflict current base file

- Conflict properties file

Noms alternatifs

Aucun.

Modifie

Rien.

Accède au dépôt

Seulement si l'opération porte sur des URL.

Options

```
--changelist ARG
--depth ARG
--incremental
--recursive (-R)
--revision (-r) REV
--targets NOM_FICHIER
--xml
```

Exemples

svn info affiche toutes les informations pertinentes qu'elle trouve pour les éléments dans la copie de travail. Elle affiche les informations concernant les fichiers :

```
$ svn info machin.c
Chemin : machin.c
Nom : machin.c
URL : http://svn.red-bean.com/repos/test/machin.c
Racine du dépôt : http://svn.red-bean.com/repos/test
UUID du dépôt : 5e7d134a-54fb-0310-bd04-b611643e5c25
Révision: 4417
Type de nœud : fichier
Tâche programmée : normale
Auteur de la dernière modification : sally
Révision de la dernière modification : 20
Date de la dernière modification : 2003-01-13 16:43:13 -0600 (lun. 13 jan 2003)
Texte mis à jour : 2003-01-16 21:18:16 -0600 (Thu, 16 Jan 2003)
Propriétés mises à jour : 2003-01-13 21:50:19 -0600 (lun. 13 jan 2003)
Somme de contrôle : d6aeb60b0662ccceb6bce4bac344cb66
```

Elle affiche aussi des informations sur les répertoires :

```
$ svn info vendors
Chemmin : vendors
URL : http://svn.red-bean.com/repos/test/vendors
Racine du dépôt : http://svn.red-bean.com/repos/test
UUID du dépôt : 5e7d134a-54fb-0310-bd04-b611643e5c25
Révision: 19
Type de nœud : répertoire
Tâche programmée : normale
Auteur de la dernière modification : harry
Révision de la dernière modification : 19
Date de la dernière modification : 2003-01-16 23:21:19 -0600 (jeu. 16 jan 2003)
Properties mises à jour : 2003-01-16 23:39:02 -0600 (jeu. 16 jan 2003)
```

svn info traite aussi les URL (notez que dans cet exemple, le fichier `lisezmoi.doc` est verrouillé et donc l'information relative à ce verrouillage est fournie) :

```
$ svn info http://svn.red-bean.com/repos/test/lisezmoi.doc
Chemin : lisezmoi.doc
Nom : lisezmoi.doc
URL : http://svn.red-bean.com/repos/test/lisezmoi.doc
Racine du dépôt : http://svn.red-bean.com/repos/test
UUID du dépôt : 5e7d134a-54fb-0310-bd04-b611643e5c25
Révision: 1
Type de nœud : fichier
Tâche programmée : normale
Auteur de la dernière modification : sally
Révision de la dernière modification : 42
Date de la dernière modification : 2003-01-14 23:21:19 -0600 (mar. 14 jan 2003)
Nom de verrou : opaquelocktoken:14011d4b-54fb-0310-8541-dbd16bd471b2
Propriétaire du verrou : harry
Verrou créé : 2003-01-15 17:35:12 -0600 (mer. 15 jan 2003)
Lock Comment (1 line):
My test lock comment
```

Pour finir, l'affichage de **svn info** est disponible au format XML en spécifiant l'option `--xml` :

```
$ svn info --xml http://svn.red-bean.com/repos/test
<?xml version="1.0"?>
<info>
<entry
   kind="dir"
   path="."
   revision="1">
<url>http://svn.red-bean.com/repos/test</url>
<repository>
<root>http://svn.red-bean.com/repos/test</root>
<uuid>5e7d134a-54fb-0310-bd04-b611643e5c25</uuid>
</repository>
<wc-info>
<schedule>normal</schedule>
<depth>infinity</depth>
</wc-info>
<commit
   revision="1">
<author>sally</author>
<date>2003-01-15T23:35:12.847647Z</date>
</commit>
</entry>
</info>
```

Nom

svn list — Lister le contenu de répertoires dans un dépôt.

Synopsis

```
svn list [CIBLE[@REV]...]
```

Description

Liste chaque fichier *`CIBLE`* et le contenu de chaque répertoire *`CIBLE`* comme ils existent dans le dépôt. Si *`CIBLE`* est un chemin de la copie de travail, alors l'URL correspondante sur le dépôt est utilisée.

Par défaut, la *`CIBLE`* est « `.` », c'est-à-dire l'URL du dépôt pour le répertoire courant de la copie de travail.

Avec l'option `--verbose`, **svn list** affiche les champs suivants pour chaque élément :

- Numéro de révision de la dernière propagation ;
- Auteur de la dernière propagation
- Si l'élément est verrouillé, la lettre « O » (lisez la section svn info pour plus de détails) ;
- Taille (en octets) ;
- Date et heure de la dernière propagation.

Avec l'option `--xml`, l'affichage est au format XML format (avec une entête et un élément document qui encadre le tout à moins que l'option `--incremental` soit aussi spécifiée). Toute l'information disponible est affichée, l'option `--verbose` n'est pas acceptée.

Noms alternatifs

ls

Modifie

Rien.

Accède au dépôt

Oui.

Options

```
--depth ARG
--incremental
--recursive (-R)
--revision (-r) REV
--verbose (-v)
--xml
```

Exemples

svn list est particulièrement utile quand vous voulez visualiser des fichiers dans le dépôt sans les télécharger dans votre copie de travail :

```
$ svn list http://svn.red-bean.com/repos/test/support
LISEZMOI.txt
INSTALL
exemples/
…
```

Vous pouvez passer l'option `--verbose` pour obtenir des informations complémentaires, un peu comme la commande Unix **ls -l** :

```
$ svn list --verbose file:///var/svn/depot
     16 sally         28361 Jan 16 23:18 LISEZMOI.txt
     27 sally             0 Jan 18 15:27 INSTALL
     24 harry               Jan 18 11:27 exemples/
```

Vous pouvez obtenir un affichage au format XML avec la commande **svn list** en spécifiant l'option `--xml` :

```
$ svn list --xml http://svn.red-bean.com/repos/test
<?xml version="1.0"?>
<lists>
<list
   path="http://svn.red-bean.com/repos/test">
<entry
   kind="dir">
<name>exemples</name>
<size>0</size>
<commit
   revision="24">
<author>harry</author>
<date>2008-01-18T06:35:53.048870Z</date>
</commit>
</entry>
…
</list>
</lists>
```

For further details, see the earlier section la section intitulée « svn list ».

Nom

svn lock — Verrouiller des chemins ou des URL dans le dépôt, afin qu'aucun autre utilisateur ne puisse propager (**commit**) des modifications les concernant.

Synopsis

```
svn lock CIBLE...
```

Description

Verrouille chaque *`CIBLE`*. Si une *`CIBLE`* est déjà verrouillée par un autre utilisateur, affiche un avertissement et continue de verrouiller les autres *`CIBLE`*s. Utilisez l'option `--force` pour voler un verrou à un autre utilisateur ou à une autre copie de travail.

Noms alternatifs

Aucun.

Modifie

Copie de travail, dépôt.

Accède au dépôt

Oui.

Options

```
--encoding ENC
--file (-F) FICHIER
--force
--force-log
--message (-m) TEXTE
--targets NOM_FICHIER
```

Exemples

Verrouille deux fichiers de la copie de travail :

```
$ svn lock arbre.jpg maison.jpg
'arbre.jpg' verrouillé par l'utilisateur 'harry'.
'maison.jpg' verrouillé par l'utilisateur 'harry'.
```

Verrouille un fichier dans la copie de travail qui est déjà verrouillé par un autre utilisateur :

```
$ svn lock arbre.jpg
svn: avertissement : Chemin '/arbre.jpg' déjà verrouillé par \
  l'utilisateur 'sally' dans le système de fichiers '/var/svn/depot/db'

$ svn lock --force arbre.jpg
'arbre.jpg' verrouillé par l'utilisateur 'harry'.
```

Verrouille un fichier sans copie de travail :

```
$ svn lock http://svn.red-bean.com/repos/test/arbre.jpg
```

```
'arbre.jpg' verrouillé par l'utilisateur 'harry'.
```

Pour davantage de détails, référez-vous à la section intitulée « Verrouillage ».

Nom

svn log — Afficher les entrées du journal de propagation.

Synopsis

```
svn log [CHEMIN]

svn log URL[@REV] [CHEMIN...]
```

Description

Affiche les entrées du journal de propagation du dépôt. Si aucun argument n'est fourni, **svn log** affiche les entrées du journal de propagation pour tous les fichiers et répertoires dans (et y compris) le répertoire courant de la copie de travail. Vous pouvez affiner votre requête en spécifiant un chemin, une ou plusieurs révisions, ou une combinaison des deux. L'intervalle de révisions par défaut pour un chemin local est `BASE:1`.

Si vous spécifiez une URL simple, la commande affiche les entrées du journal pour tout ce que contient l'URL. Si vous ajoutez des chemins après l'URL, seules les entrées relatives aux chemins spécifiés seront affichées. L'intervalle de révisions par défaut pour une URL est `HEAD:1`.

Avec l'option `--verbose`, **svn log** affiche également tous les chemins modifiés pour chaque entrée du journal. Avec l'option `--quiet`, **svn log** n'affiche pas le corps de l'entrée du journal (cette option est compatible avec l'option `--verbose`).

Chaque entrée du journal est affichée seulement une fois, même si, parmi les chemins explicitement demandés, plus d'un a été modifié pour cette révision. Les entrées du journal suivent l'historique de la copie de travail par défaut. Utilisez l'option `--stop-on-copy` pour désactiver ce comportement, ce qui peut être utile pour déterminer à quel moment une branche a été créée.

Noms alternatifs

Aucun.

Modifie

Rien.

Accède au dépôt

Oui.

Options

```
--change (-c) ARG
--incremental
--limit (-l) NUM
--quiet (-q)
--revision (-r) REV
--stop-on-copy
--targets NOM_FICHIER
--use-merge-history (-g)
--verbose (-v)
--with-all-revprops
--with-revprop ARG
--xml
```

Exemples

Vous pouvez visualiser les entrées du journal pour tous les chemins qui ont été modifiés dans la copie de travail en lançant la

commande **svn log** depuis le répertoire racine :

```
$ svn log
------------------------------------------------------------------------
r20 | harry | 2003-01-17 22:56:19 -0600 (ven. 17 jan 2003) | 1 ligne

Peaufinage.
------------------------------------------------------------------------
r17 | sally | 2003-01-16 23:21:19 -0600 (jeu. 16 jan 2003) | 2 lignes
…
```

Examine toutes les entrées du journal pour un fichier particulier de la copie de travail :

```
$ svn log machin.c
------------------------------------------------------------------------
r32 | sally | 2003-01-13 00:43:13 -0600 (lun. 13 jan 2003) | 1 ligne

Ajouté des définitions.
------------------------------------------------------------------------
r28 | sally | 2003-01-07 21:48:33 -0600 (mar. 07 jan 2003) | 3 lignes
…
```

Si vous n'avez pas de copie de travail sous la main, vous pouvez examiner les entrées du dépôt :

```
$ svn log http://svn.red-bean.com/repos/test/machin.c
------------------------------------------------------------------------
r32 | sally | 2003-01-13 00:43:13 -0600 (lun. 13 jan 2003) | 1 ligne

Ajouté des définitions.
------------------------------------------------------------------------
r28 | sally | 2003-01-07 21:48:33 -0600 (mar. 07 jan 2003) | 3 lignes
…
```

Si vous voulez spécifier plusieurs chemins distincts sous la même URL, vous pouvez utiliser la syntaxe `URL [CHEMIN...]` :

```
$ svn log http://svn.red-bean.com/repos/test/ machin.c bidule.c
------------------------------------------------------------------------
r32 | sally | 2003-01-13 00:43:13 -0600 (lun. 13 jan 2003) | 1 ligne

Ajouté des définitions.
------------------------------------------------------------------------
r31 | harry | 2003-01-10 12:25:08 -0600 (ven. 10 jan 2003) | 1 ligne

Ajouté le nouveau fichier bidule.c
------------------------------------------------------------------------
r28 | sally | 2003-01-07 21:48:33 -0600 (mar. 07 jan 2003) | 3 lignes
…
```

L'option `--verbose` demande à **svn log** d'inclure les informations relatives aux chemins modifiés pour chaque révision affichée. Ces chemins sont affichés, un chemin par ligne, avec le code d'action qui indique quel changement a été effectué sur le chemin.

```
$ svn log -v http://svn.red-bean.com/repos/test/ machin.c bidule.c
------------------------------------------------------------------------
r32 | sally | 2003-01-13 00:43:13 -0600 (lun. 13 jan 2003) | 1 ligne
Chemins modifiés :
   M /machin.c
```

```
Ajouté des définitions.
------------------------------------------------------------------------
r31 | harry | 2003-01-10 12:25:08 -0600 (ven. 10 jan 2003) | 1 ligne
Chemins modifiés :
   A /bidule.c

Ajouté le nouveau fichier bidule.c
------------------------------------------------------------------------
r28 | sally | 2003-01-07 21:48:33 -0600 (mar. 07 jan 2003) | 3 lignes
…
```

svn log n'utilise qu'une poignée de codes d'action, les mêmes que ceux utilisés par la commande **svn update** :

`A`
: L'élément a été ajouté.

`D`
: L'élément a été supprimé (*Deleted* en anglais).

`M`
: Les propriétés ou le contenu de l'élément ont été modifiés.

`R`
: L'élément a été remplacé par un autre au même endroit.

En plus des codes d'action qui précèdent les chemins modifiés, **svn log** avec l'option `--verbose` indique si le chemin a été ajouté ou remplacé par le fait d'une opération de copie. Il le signale en affichant (de *COPIE-DE-CHEMIN*:*COPIE-DE-REV*) après le chemin.

Quand vous concaténez la sortie de plusieurs commandes **svn log**, il peut être judicieux d'utiliser l'option `--incremental`. **svn log** affiche normalement une ligne de tirets avant chaque entrée de journal, après chaque entrée conséquente et après la derniere entrée du journal. Si vous lancez **svn log** sur un intervalle de deux révisions, vous obtenez la sortie suivante :

```
$ svn log -r 14:15
------------------------------------------------------------------------
r14 | …

------------------------------------------------------------------------
r15 | …

------------------------------------------------------------------------
```

Cependant, si vous voulez rassembler, dans un fichier, deux entrées du journal qui ne se suivent pas, vous lancerez des commandes comme ceci :

```
$ svn log -r 14 > mon-journal
$ svn log -r 19 >> mon-journal
$ svn log -r 27 >> mon-journal
$ cat mon-journal
------------------------------------------------------------------------
r14 | …

------------------------------------------------------------------------
------------------------------------------------------------------------
r19 | …

------------------------------------------------------------------------
------------------------------------------------------------------------
r27 | …

------------------------------------------------------------------------
```

Vous pouvez éviter l'inconvénient des doubles lignes de tirets dans votre fichier en utilisant l'option `--incremental` :

```
$ svn log --incremental -r 14 > mon-journal
$ svn log --incremental -r 19 >> mon-journal
$ svn log --incremental -r 27 >> mon-journal
$ cat mon-journal
------------------------------------------------------------------------
r14 | …

------------------------------------------------------------------------
r19 | …

------------------------------------------------------------------------
r27 | …
```

L'option `--incremental` fournit le même type de fonctionnalité quand elle est utilisée avec l'option `--xml` :

```
$ svn log --xml --incremental -r 1 lisezmoi.txt
<logentry
   revision="1">
<author>harry</author>
<date>2008-06-03T06:35:53.048870Z</date>
<msg>Initial Import.</msg>
</logentry>
```

Quelquefois, quand vous lancez la commande **svn log** sur un chemin spécifique et une révision spécifique, vous ne voyez aucune entrée de journal, comme dans le cas suivant :

```
$ svn log -r 20 http://svn.red-bean.com/pas-modifié.txt
------------------------------------------------------------------------
```

Cela signifie simplement que le chemin n'a pas été modifié lors de la révision spécifiée. Pour obtenir l'entrée de journal correspondante à cette révision, soit lancez l'opération de log sur l'URL racine du dépôt, soit sur un chemin dont vous savez qu'il a été modifié lors de cette révision :

```
$ svn log -r 20 modifié.txt
------------------------------------------------------------------------
r20 | sally | 2003-01-17 22:56:19 -0600 (ven. 17 jan 2003) | 1 ligne

Effectué un changement.
------------------------------------------------------------------------
```

Nom

svn merge — Appliquer les différences entre deux sources à une copie de travail.

Synopsis

```
svn merge sourceURL1[@N] sourceURL2[@M] [CHEMIN]

svn merge sourceCHEMIN1@N sourceCHEMIN2@M [CHEMIN]

svn merge [[-c M]... | [-r N:M]...] [SOURCE[@REV] [CHEMIN]]
```

Description

Avec la première forme, les URL sources sont spécifiées en précisant les révisions *N* et *M*. Ce sont les deux sources à comparer. Les valeurs par défaut de ces révisions sont HEAD si elles sont omises.

Avec la deuxième forme, les URL correspondantes aux chemins dans la copie de travail définissent les sources à comparer. Les révisions doivent être spécifiées.

Avec la troisième forme, *SOURCE* peut être une URL ou un chemin de la copie de travail (dans ce cas, l'URL correspondante est utilisée). Si elle n'est pas spécifiée, *SOURCE* est identique à *CHEMIN*. *SOURCE* dans la révision *REV* est comparée telle que vue entre les révisions *N* et *M* pour chaque intervalle de révisions fourni. Si *REV* n'est pas spécifiée, HEAD est la valeur par défaut.

`-c M` est équivalent à `-r <M-1>:M` et `-c -M` correspond à l'inverse : `-r M:<M-1>`. Si aucun intervalle de révisions n'est spécifié, l'intervalle par défaut `1:HEAD` est utilisé. Plusieurs instances de `-c` ou `-r` peuvent être spécifiées et le mélange d'intervalles croissants et décroissant est autorisé — les intervalles sont réduits en interne à leur représentation minimum avant d'effectuer la fusion proprement dite (ce qui peut conduire à une opération ne faisant rien).

CHEMIN est le chemin de la copie de travail qui reçoit les modifications. Si *CHEMIN* est omis, la valeur par défaut est « . », à moins que les sources aient des noms de fichiers (*basename*) identiques qui correspondent à un fichier dans « . ». Dans ce cas, les différences sont appliquées à ce fichier.

Subversion garde une trace en interne des métadonnées relatives aux fusions opérées seulement si les deux sources sont parentes l'une de l'autre, dans un sens ou dans l'autre. Cela est garanti si la troisième forme est utilisée. Au contraire de **svn diff**, la commande **merge** prend en compte l'héritage d'un fichier pour effectuer la fusion. Ceci est particulièrement important quand vous fusionnez les modifications d'une branche dans une autre et que vous avez renommé un fichier dans une branche mais pas l'autre.

Noms alternatifs

Aucun.

Modifie

Copie de travail.

Accède au dépôt

Seulement si l'opération porte sur des URL

Options

```
--accept ARG
--change (-c) REV
--depth ARG
--diff3-cmd CMD
--dry-run
```

```
--extensions (-x) ARG
--force
--ignore-ancestry
--quiet (-q)
--record-only
--reintegrate
--revision (-r) REV
```

Exemples

Fusionne une branche (issue du tronc) dans le tronc (considère que vous avez une copie de travail à jour du tronc) :

```
$ svn merge --reintegrate \
            http://svn.example.com/depot/calc/branches/ma-branche-calc
--- Fusion des différences des URLs du dépôt vers '.':
U    bouton.c
U    entier.c
U    Makefile
 U   .

$ # compilation, tests, vérifications ...

$ svn commit -m "Ré-intègre ma-branche-calc dans le tronc !"
Envoi        .
Envoi        bouton.c
Envoi        entier.c
Envoi        Makefile
Transmission des données .
Révision 391 propagée.
```

Pour fusionner les modifications dans un seul fichier :

```
$ cd mon-projet
$ svn merge -r 30:31 thhgttg.txt
U  thhgttg.txt
```

Nom

svn mergeinfo — Afficher les informations liées aux fusions. Voir la section intitulée « Mergeinfo et aperçus » pour les détails.

Synopsis

```
svn mergeinfo URL_SOURCE[@REV] [CIBLE[@REV]...]
```

Description

Affiche les informations liées aux fusions (ou fusions potentielles) entre *`SOURCE-URL`* et *`CIBLE`*. Si l'option `--show-revs` n'est pas fournie, affiche les révisions qui ont été fusionnées depuis *`SOURCE-URL`* vers *`CIBLE`*. Sinon, affiche soit `merged`, soit `eligible` comme spécifié par l'option `--show-revs`.

Noms alternatifs

Aucun.

Modifie

Rien.

Accède au dépôt

Oui.

Options

```
--revision (-r) REV
```

Exemples

Trouve quels ensembles de modifications le tronc (`trunk/`) a déjà reçu et quels ensembles il est prêt (« eligible ») à recevoir :

```
$ svn mergeinfo branches/test
Path: branches/test
  Source path: /trunk
    Merged ranges: r2:13
    Eligible ranges: r13:15
```

Nom

svn mkdir — Créer un nouveau répertoire et le placer en gestion de versions.

Synopsis

```
svn mkdir CHEMIN...

svn mkdir URL...
```

Description

Crée un répertoire avec le nom fourni comme dernière composante (*basename*) de *`CHEMIN`* ou *`URL`*. Un répertoire spécifié par un *`CHEMIN`* dans la copie de travail est prévu pour ajout lors de la prochaine propagation. Un répertoire spécifié par une URL est directement créé dans le dépôt par une propagation immédiate. Si plusieurs URL sont spécifiées, les répertoires sont créés atomiquement. Dans les deux cas, les répertoires intermédiaires doivent déjà exister sauf si l'option `--parents` est spécifiée.

Noms alternatifs

Aucun.

Modifie

Copie de travail ; Dépôt si l'opération concerne des URL.

Accède au dépôt

Seulement si l'opération concerne des URL.

Options

```
--editor-cmd EDITEUR
--encoding ENC
--file (-F) FICHIER
--force-log
--message (-m) TEXTE
--parents
--quiet (-q)
--with-revprop ARG
```

Exemples

Crée un répertoire dans la copie de travail :

```
$ svn mkdir nouveau-rep
A         nouveau-rep
```

En crée un dans le dépôt (cela entraine une propagation, il faut donc fournir une entrée de journal) :

```
$ svn mkdir -m "Création d'un nouveau répertoire." \
            http://svn.red-bean.com/repos/nouveau-rep

Révision 26 propagée.
```

Nom

svn move — Déplacer un fichier ou un répertoire.

Synopsis

```
svn move SRC... DST
```

Description

Déplace des fichiers ou des répertoires dans la copie de travail ou dans le dépôt.

Cette commande est équivalent à une **svn copy** suivi d'une **svn delete**.

Quand plusieurs sources sont déplacées, elles sont ajoutées en tant que fils de *`DST`*, qui doit être un répertoire.

Subversion n'accepte pas les déplacements entre les copies de travail et les URL. De plus, vous ne pouvez déplacer des fichiers qu'à l'intérieur d'un même dépôt — Subversion n'accepte pas les déplacements inter-dépôts. Subversion accepte les déplacements selon les formes suivantes à l'intérieur d'un dépôt :

WC # WC
: Déplace et prévoit pour ajout (avec historique) le fichier ou répertoire.

URL # URL
: renommage complet côté serveur.

Noms alternatifs

mv, **rename**, **ren**

Modifie

Copie de travail ; dépôt si l'opération concerne une URL.

Accède au dépôt

Seulement si l'opération concerne une URL.

Options

```
--editor-cmd EDITEUR
--encoding ENC
--file (-F) FICHIER
--force
--force-log
--message (-m) TEXTE
--parents
--quiet (-q)
--revision (-r) REV
--with-revprop ARG
```

Exemples

Déplace un fichier de la copie de travail :

```
$ svn move machin.c bidule.c
A         bidule.c
D         machin.c
```

Déplace plusieurs fichiers de la copie de travail vers un sous-répertoire :

```
$ svn move truc.c bat.c qux.c src
A         src/truc.c
D         truc.c
A         src/bat.c
D         bat.c
A         src/qux.c
D         qux.c
```

Déplace un fichier dans le dépôt (cela entraine une propagation, il faut donc fournir une entrée de journal) :

```
$ svn move -m "Déplace un fichier" http://svn.red-bean.com/repos/machin.c \
                                   http://svn.red-bean.com/repos/bidule.c

Révision 27 propagée.
```

Nom

svn propdel — Supprimer une propriété d'un élément.

Synopsis

```
svn propdel PROP_NOM [CHEMIN...]

svn propdel PROP_NOM --revprop -r REV [CIBLE]
```

Description

Supprime les propriétés des fichiers, des répertoires ou des révisions. La première forme supprime les propriétés suivies en version de votre copie de travail et la deuxième forme supprime à distance les propriétés non suivies en version d'une révision du dépôt (*`CIBLE`* permet juste de préciser le dépôt à utiliser).

Noms alternatifs

pdel, **pd**

Modifie

Copie de travail ; dépôt si l'opération concerne une URL.

Accède au dépôt

Seulement si l'opération concerne une URL.

Options

```
--changelist ARG
--depth ARG
--quiet (-q)
--recursive (-R)
--revision (-r) REV
--revprop
```

Exemples

Supprime une propriété d'un fichier de votre copie de travail :

```
$ svn propdel svn:mime-type un-script
Propriété 'svn:mime-type' supprimée de 'un-script'.
```

Supprime une propriété de révision :

```
$ svn propdel --revprop -r 26 date-de-publication
Propriété 'date-de-publication' supprimée de la révision 26 du dépôt
```

Nom

svn propedit — Modifier une propriété d'un ou plusieurs éléments gérés en version. Voir svn propset plus loin dans ce chapitre.

Synopsis

```
svn propedit PROP_NOM CIBLE...

svn propedit PROP_NOM --revprop -r REV [CIBLE]
```

Description

Modifie une ou plusieurs propriétés en utilisant votre éditeur préféré. La première forme modifie les propriétés suivies en version de votre copie de travail et la deuxième forme modifie à distance des propriétés non suivies en version d'une révision d'un dépôt (*`CIBLE`* permet juste de préciser le dépôt à utiliser).

Noms alternatifs

pedit, **pe**

Modifie

Copie de travail ; dépôt si l'opération concerne une URL.

Accède au dépôt

Seulement si l'opération concerne une URL.

Options

```
--editor-cmd EDITEUR
--encoding ENC
--file (-F) ARG
--force
--force-log
--message (-m) ARG
--revision (-r) REV
--revprop
--with-revprop ARG
```

Exemples

Avec **svn propedit** il est très facile de modifier des propriétés contenant plusieurs valeurs :

```
$ svn propedit svn:keywords  machin.c
    <svn lance votre éditeur favori en prenant en entrée le
    contenu actuel de la propriété svn:keywords. Pour ajouter
    plusieurs valeurs, en placer une par ligne.>
Nouvelle valeur définie pour la propriété 'svn:keywords' sur 'machin.c'
```

Nom

svn propget — Afficher la valeur d'une propriété.

Synopsis

```
svn propget PROP_NOM [CIBLE[@REV]...]

svn propget PROP_NOM --revprop -r REV [URL]
```

Description

Affiche la valeur d'une propriété définie sur des fichiers, des répertoires ou des révisions. #a première forme affiche la valeur d'une propriété suivie en version d'un ou plusieurs éléments de votre copie de travail et la deuxième forme affiche des propriétés suivies en version d'une révision d'un dépôt. Voir la section intitulée « Propriétés » pour de plus amples informations sur les propriétés.

Noms alternatifs

pget, **pg**

Modifie

Copie de travail ; dépôt si l'opération concerne une URL.

Accède au dépôt

Seulement si l'opération concerne une URL.

Options

```
--changelist ARG
--depth ARG
--recursive (-R)
--revision (-r) REV
--revprop
--strict
--xml
```

Exemples

Examine une propriété d'un fichier de votre copie de travail :

```
$ svn propget svn:keywords machin.c
Author
Date
Rev
```

De même pour une propriété de révision :

```
$ svn propget svn:log --revprop -r 20
Démarré le journal de bord.
```

Enfin, il est possible d'obtenir la sortie de **svn propget** au format XML avec l'option `--xml` :

```
$ svn propget --xml svn:ignore .
<?xml version="1.0"?>
<properties>
<target
   path="">
<property
   name="svn:ignore">*.o
</property>
</target>
</properties>
```

Nom

svn proplist — Lister toutes les propriétés.

Synopsis

```
svn proplist [CIBLE[@REV]...]

svn proplist --revprop -r REV [CIBLE]
```

Description

Liste toutes les propriétés existentes sur les fichiers, répertoires ou révisions. La première forme liste les propriétés suivies en version de votre copie de travail et la deuxième forme liste les propriétés définies sur une révision d'un dépôt (*`CIBLE`* permet juste de préciser le dépôt à utiliser).

Noms alternatifs

plist, **pl**

Modifie

Copie de travail ; dépôt si l'opération concerne une URL.

Accède au dépôt

Seulement si l'opération concerne une URL.

Options

```
--changelist ARG
--depth ARG
--quiet (-q)
--recursive (-R)
--revision (-r) REV
--revprop
--verbose (-v)
--xml
```

Exemples

proplist peut être utilisé pour obtenir la liste des propriétés d'un élément de votre copie de travail :

```
$ svn proplist machin.c
Propriétés sur 'machin.c':
  svn:mime-type
  svn:keywords
  proprietaire
```

Grâce à l'option `--verbose`, **svn proplist** peut s'avérer très utile car il affiche alors en plus les valeurs des propriétés :

```
$ svn proplist --verbose machin.c
Propriétés sur 'machin.c':
  svn:mime-type : text/plain
  svn:keywords : Author Date Rev
  proprietaire : sally
```

Enfin, il est possible d'obtenir la sortie de **svn proplist** au format XML avec l'option `--xml` :

```
$ svn proplist --xml
<?xml version="1.0"?>
<properties>
<target
   path=".">
<property
   name="svn:ignore"/>
</target>
</properties>
```

Nom

svn propset — Associer à la propriété *PROP_NOM* la valeur *PROP_VAL* pour des fichiers, répertoires ou révisions.

Synopsis

```
svn propset PROP_NOM [PROP_VAL | -F FICHIER_VAL] CHEMIN...

svn propset PROP_NOM --revprop -r REV [PROP_VAL | -F FICHIER_VAL] [CIBLE]
```

Description

Associe à *PROP_NOM* la valeur *PROP_VAL* sur des fichiers, répertoires ou révisions. Le premier exemple définit la modification locale de la propriété suivie en version dans la copie de travail et la deuxième forme définit un changement à distance d'une propriété de révision du dépôt non suivie en version (*CIBLE* permet juste de préciser le dépôt à utiliser).

Subversion possède un certain nombre de propriétés « spéciales » permettant de modifier la façon dont il fonctionne. Voir la section intitulée « Propriétés dans Subversion » plus loin dans ce chapitre pour plus d'informations sur ces propriétés.

Noms alternatifs

pset, *ps*

Modifie

Copie de travail ; dépôt si l'opération concerne une URL.

Accède au dépôt

Seulement si l'opération concerne une URL.

Options

```
--changelist ARG
--depth ARG
--encoding ENC
--file (-F) FICHIER
--force
--quiet (-q)
--recursive (-R)
--revision (-r) REV
--revprop
--targets NOM_FICHIER
```

Exemples

Définit le type MIME d'un fichier :

```
$ svn propset svn:mime-type image/jpeg truc.jpg
Propriété 'svn:mime-type' définie sur 'truc.jpg'
```

Sur un système de type Unix, pour qu'un fichier devienne exécutable :

```
$ svn propset svn:executable ON un-script
```

```
Propriété 'svn:executable' définie sur 'un-script'
```

Vous disposez peut-être de règles internes consistant à définir certaines propriétés utiles à vos collègues :

```
$ svn propset proprietaire sally machin.c
Propriété 'proprietaire' définie sur 'machin.c'
```

Si une erreur a été commise dans le message de propagation d'une révision donnée et que vous désirez la corriger, utilisez l'option `--revprop` et affectez à `svn:log` le contenu du nouveau message :

```
$ svn propset --revprop -r 25 svn:log "Compte-rendu du voyage à New York."
Propriété 'svn:log' définie à la révision du dépôt 25
```

Ou bien si vous ne disposez pas d'une copie de travail, vous pouvez fournir une URL :

```
$ svn propset --revprop -r 26 svn:log "Jour sans." \
              http://svn.red-bean.com/repos
Propriété 'svn:log' définie à la révision du dépôt 26
```

Enfin, vous pouvez indiquer à **propset** de lire le contenu de la propriété dans un fichier. Vous pourriez même utiliser ceci pour donner une valeur binaire à cette propriété :

```
$ svn propset icone-du-proprietaire -F sally.jpg bidule.c
Propriété 'icone-du-proprietaire' définie sur 'bidule.c'
```

Par défaut il n'est pas possible de modifier les propriétés de révision d'un dépôt Subversion. L'administrateur du dépôt doit explicitement activer la modification des propriétés de révision en créant une procédure automatique appelée `pre-revprop-change`. Consulter la section intitulée « Mise en place des procédures automatiques » pour plus d'informations sur les procédures automatiques.

Nom

svn resolve — Résoudre les conflits sur les fichiers et répertoires de la copie de travail.

Synopsis

```
svn resolve CHEMIN...
```

Description

Résoud les fichiers et répertoires marqués « en conflit » dans la copie de travail. Cette commande ne résoud pas sémantiquement les marques de conflit ; Cependant, elle remplace *`CHEMIN`* avec la version spécifiée par l'argument de l'option `--accept` puis supprime les fichiers créés lors de la signalisation du conflit. Cela permet à *`CHEMIN`* d'être à nouveau propagé, c'est-à-dire que Subversion considère que les conflits ont été « résolus ». Vous pouvez passer les arguments suivants à l'option `--accept`, en fonction du résultat attendu :

`base`
: Choisit le fichier qui était dans la révision BASE avant de mettre à jour la copie de travail. C'est-à-dire le fichier que vous avez extrait avant de faire vos dernières modifications.

`working`
: Suppose que vous avez géré manuellement la résolution du conflit et choisit la version du fichier tel qu'il est actuellement dans votre copie de travail.

`mine-full`
: Utilise les versions de fichiers tels qu'ils étaient immédiatement avant que vous ne lanciez la commande **svn update**.

`theirs-full`
: Utilise les versions de fichiers tels qu'ils ont été extraits depuis le serveur quand vous avez lancé la commande **svn update**.

reportez-vous à la section intitulée « Résoudre les conflits (fusionner des modifications) » pour une description en profondeur de la gestion des conflits.

Noms alternatifs

Aucun.

Modifie

Copie de travail.

Accède au dépôt

Non.

Options

```
--accept ARG
--depth ARG
--quiet (-q)
--recursive (-R)
--targets NOM_FICHIER
```

Exemples

Voici un exemple où, après avoir reporté un conflit durant la mise à jour, **svn resolve** remplace tous les conflits du fichier

`machin.c` par vos modifications :

```
$ svn up
Conflit découvert dans 'machin.c'.
Sélectionner : (p) report, (df) diff complet, (e) édite,
        (h) aide pour plus d'options :p
C    machin.c
Actualisé à la révision 5.

$ svn resolve --accept mine-full machin.c
Conflit sur 'machin.c' résolu
```

Nom

svn resolved — *Obsolète*. Supprimer les marques « en conflit » des fichiers et répertoires de la copie de travail.

Synopsis

```
svn resolved CHEMIN...
```

Description

Cette commande est obsolète et remplacée par **`svn resolve --accept working`** ***`CHEMIN`***. Reportez-vous à svn resolve dans la section précédente pour plus de détails.

Supprimer les marques « en conflit » sur les fichiers et répertoires de la copie de travail. Cette commande ne résoud pas sémantiquement les conflits ; elle supprime simplement les fichiers créés lors de la signalisation du conflit et permet à ***`CHEMIN`*** d'être à nouveau propagé. C'est-à-dire qu'elle indique à Subversion que les conflits ont été résolus. Reportez-vous à la section intitulée « Résoudre les conflits (fusionner des modifications) » pour une description en profondeur de la gestion des conflits.

Noms alternatifs

Aucun.

Modifie

Copie de travail.

Accède au dépôt

Non.

Options

```
--depth ARG
--quiet (-q)
--recursive (-R)
--targets NOM_FICHIER
```

Exemple

Si une mise à jour détecte un conflit, votre copie de travail comporte trois nouveaux fichiers :

```
$ svn update
C  machin.c
Actualisé à la révision 31.
$ ls
machin.c
machin.c.mine
machin.c.r30
machin.c.r31
```

Une fois que vous avez résolu les conflits et que `machin.c` est prêt à être propagé, lancez **svn resolved** pour indiquer à votre copie de travail que vous avez pris les choses en main.

Vous *pouvez* supprimer les fichiers associés au conflit et effectuer une propagation, mais **svn resolved** met à jour des données de suivi dans la zone d'administration en plus de supprimer les fichiers associés au conflit. C'est pourquoi nous recommandons d'utiliser cette commande.

Nom

svn revert — Restaurer l'état initial.

Synopsis

```
svn revert CHEMIN...
```

Description

Restaure l'état des fichiers et répertoires en annulant les modifications apportées localement et supprime les marques de conflit. **svn revert** restaure non seulement le contenu des éléments de la copie de travail, mais également les valeurs des propriétés. En fait, vous pouvez l'utiliser pour annuler toute opération planifiée que vous avez initiée (pa exemple, l'ajout ou la suppression de fichiers peuvent être « déplanifiés »).

Noms alternatifs

Aucun.

Modifie

Copie de travail.

Accède au dépôt

Non.

Options

```
--changelist ARG
--depth ARG
--quiet (-q)
--recursive (-R)
--targets NOM_FICHIER
```

Exemples

Annuler les changements apportés à un fichier :

```
$ svn revert machin.c
'machin.c' réinitialisé
```

Si vous voulez restaurer l'état initial de tout un répertoire, utilisez l'option `--depth=infinity` :

```
$ svn revert --depth=infinity .
'nouveau-rep/un-fichier' réinitialisé
'machin.c' réinitialisé
'bidule.txt' réinitialisé
```

Enfin, vous pouvez annuler toute opération planifiée :

```
$ svn add betise.txt oulala
A         betise.txt
A         oulala
A         oulala/opla.c
```

```
$ svn revert betise.txt oulala
'betise.txt' réinitialisé
'oulala' réinitialisé

$ svn status
?      betise.txt
?      oulala
```

svn revert est dangereux par nature puisque sa finalité est de supprimer des données (vos modifications non propagées). Une fois lancée, Subversion ne peut *en aucun cas* récuperer les modifications non propagées.

Si vous ne spécifiez aucune cible à **svn revert**, elle ne fait rien — ce comportement est destiné à vous protéger d'une perte accidentelle de données de votre copie de travail. Vous devez fournir au moins une cible à **svn revert**.

Nom

svn status — Afficher l'état des fichiers et des répertoires de la copie de travail.

Synopsis

```
svn status [CHEMIN...]
```

Description

Affiche l'état des fichiers et des répertoires de la copie de travail. Sans arguments, elle affiche simplement les éléments modifiés (pas d'accès au dépôt). Avec l'option `--show-updates`, elle affiche la révision en cours et les éléments obsolètes. Avec l'option `--verbose`, elle affiche les informations de révision complètes pour chaque élément. Avec l'option `--quiet`, elle affiche uniquement des informations résumées sur les éléments modifiés localement.

Les six premières colonnes de l'affichage mesurent un caractère de large et chaque colone apporte des informations complémentaires sur chaque élément de la copie de travail.

La première colonne indique si un élément est ajouté, supprimé ou modifié :

`' '`
: Pas de modification.

`'A'`
: Élement prévu d'être ajouté.

`'D'`
: Élément prévu d'être supprimé (*Deletion*).

`'M'`
: Élément modifié.

`'R'`
: Élément remplacé dans la copie de travail. Cela veut dire que le fichier a été prévu d'être supprimé puis un nouveau fichier avec le même nom a été prévu d'être ajouté à sa place.

`'C'`
: Le contenu (par opposition aux propriétés) de l'élément est en conflit avec les mises à jour reçues depuis le dépôt.

`'X'`
: L'élément fait partie d'une définition externe.

`'I'`
: Élément ignoré (par exemple en raison d'une propriété `svn:ignore`).

`'?'`
: Élément non géré en versions.

`'!'`
: Élément manquant (par exemple si vous l'avez déplacé ou effacé sans utiliser la commande **svn**). Cela indique également qu'un répertoire n'est pas complet (une extraction ou une mise à jour a été interrompue).

`'~'`
: Élément géré en tant qu'un certain type d'objet (fichier, répertoire, lien) mais qui a été remplacé par un objet de type différent.

La deuxième colonne indique l'état des propriétés des fichiers ou répertoires :

`' '`

Pas de modification.

`'M'`
Les propriétés de l'élément ont été modifiées.

`'C'`
Les propriétés de cet élément sont en conflit avec les mises à jour des propriétés reçues depuis le dépôt.

La troisième colonne indique si le répertoire de la copie de travail est verrouillé (voir la section intitulée « Parfois, il suffit de faire le ménage ») :

`' '`
Élément non verrouillé.

`'L'`
Élément verrouillé (*Locked*).

La quatrième colonne indique si la prochaine propagation de l'élément comportera une reprise de l'historique :

`' '`
Pas de reprise de l'historique prévue.

`'+'`
Ajout avec reprise de l'historique prévue.

La cinquième colonne indique si l'élément est ré-aiguillé par rapport à son parent (voir la section intitulée « Parcours des branches ») :

`' '`
L'élément est un fils du répertoire parent.

`'S'`
L'élément a été ré-aiguillé (*Switched*).

La sixième colonne fournit les informations de verrouillage :

`' '`
Quand `--show-updates` est spécifiée, le fichier n'est pas verrouillé. Si `--show-updates` *n'est pas spécifiée* , cela veut simplement dire que le fichier n'est pas verrouillé dans la copie de travail.

K
Fichier verrouillé par cette copie de travail (*locked*).

O
Fichier verrouillé soit par un autre utilisateur, soit par une autre copie de travail. Ceci n'est possible qu'en utilisant l'option `--show-updates`.

T
Fichier verrouillé par cette copie de travail mais le verrou a été « volé » et n'est plus valide (*stolen*). Le fichier est verrouillé dans le dépôt. Ceci n'est possible qu'en utilisant l'option `--show-updates`.

B
Fichier verrouillé par cette copie de travail mais le verrou a été « cassé » et n'est plus valide (*broken*). Le fichier n'est plus verrouillé. Ceci n'est possible qu'en utilisant l'option `--show-updates`.

Les informations relatives à l'obsolescence apparaissent dans la septième colonne (uniquement si l'option `--show-updates`

est spécifiée) :

`' '`
: L'élément de la copie de travail est à jour.

`'*'`
: Une nouvelle version de l'élément existe sur le serveur.

Les champs restants sont de largeur variable et séparés par des espaces. Le numéro de la révision de travail si l'option `--show-updates` ou `--verbose` est spécifiée.

Si l'option `--verbose` est spécifiée, le numéro et l'auteur de la dernière révision propagée sont affichés.

Le chemin de la copie de travail est toujours le dernier champ affiché et peut inclure des espaces.

Noms alternatifs

stat, **st**

Modifie

Rien.

Accède au dépôt

Seulement si l'option `--show-updates` est utilisée.

Options

```
--changelist ARG
--depth ARG
--ignore-externals
--incremental
--no-ignore
--quiet (-q)
--show-updates (-u)
--verbose (-v)
--xml
```

Exemples

Voici la manière la plus facile de déterminer les modifications que vous avez apportées à votre copie de travail :

```
$ svn status wc
 M     wc/bidule.c
A  +   wc/qax.c
```

Si vous voulez trouver quels fichiers dans votre copie de travail sont obsolètes, spécifiez l'option `--show-updates` (cela *ne modifiera pas* votre copie de travail). Dans l'exemple suivant, vous pouvez voir que `wc/machin.c` a été modifié dans le dépôt depuis la dernière mise à jour dans notre copie de travail :

```
$ svn status --show-updates wc
 M           965    wc/bidule.c
       *     965    wc/machin.c
A  +         965    wc/qax.c
État par rapport à la révision 981
```

`--show-updates` *ne fait que* afficher une astérisque pour les éléments qui sont obsolètes (c'est-à-dire les éléments qui seront mis à jour par le dépôt lors de la prochaine commande **svn update**). `--show-updates` *ne modifie pas* le numéro de version affiché pour l'élément en indiquant le numéro de version de l'élément dans le dépôt (bien que vous pouvez voir le numéro de révision dans le dépôt en spécifiant l'option `--verbose`).

Le maximum d'informations que vous pouvez obtenir à l'aide de la sous-commande **status** est le suivant :

```
$ svn status --show-updates --verbose wc
 M           965       938 sally        wc/bidule.c
       *     965       922 harry        wc/machin.c
A  +         965       687 harry        wc/qax.c
             965       687 harry        wc/zig.c
État par rapport à la révision 981
```

Enfin, vous pouvez demander à **svn status** d'afficher les résultats au format XML en spéficiant l'option `--xml` :

```
$ svn status --xml wc
<?xml version="1.0"?>
<status>
<target
   path="wc">
<entry
   path="qax.c">
<wc-status
   props="none"
   item="added"
   revision="0">
</wc-status>
</entry>
<entry
   path="bidule.c">
<wc-status
   props="normal"
   item="modified"
   revision="965">
<commit
   revision="965">
<author>sally</author>
<date>2008-05-28T06:35:53.048870Z</date>
</commit>
</wc-status>
</entry>
</target>
</status>
```

Pour voir beaucoup plus d'exemples utilisant la commande **svn status**, lisez la section intitulée « Avoir une vue d'ensemble des changements effectués ».

Nom

svn switch — Mettre à jour la copie de travail à partir d'une nouvelle URL.

Synopsis

```
svn switch URL[@PEGREV] [CHEMIN]

switch --relocate FROM TO [CHEMIN...]
```

Description

La première forme de cette sous-commande (sans l'option `--relocate`) actualise votre copie de travail en la faisant pointer sur une nouvelle URL — généralement une URL qui partage un ancêtre commun avec votre copie de travail, bien que cela ne soit pas obligatoire. C'est la manière de faire avec Subversion pour utiliser une nouvelle branche pour votre copie de travail. Si *PEGREV* est spécifié, cela indique dans quelle révision chercher pour déterminer la cible. Reportez-vous à la section intitulée « Parcours des branches » pour une description en profondeur de la gestion des branches.

Si l'option `--force` est utilisée, les chemins non gérés en versions de la copie de travail qui font obstacle à l'ajout, par la commande de ré-aiguillage, de chemins ayant le même nom ne font pas échouer automatiquement l'opération. Si le chemin faisant obstacle est du même type (fichier ou répertoire) que le chemin correspondant dans le dépôt, il passe sous gestion de versions mais son contenu est laissé tel quel dans la copie de travail. Cela veut dire que les enfants d'un répertoire non géré en versions peuvent également faire obstacle et se retrouver gérés en versions. Pour les fichiers, toute différence entre l'obstacle et le dépôt est gérée comme une modification locale à la copie de travail. Toutes les propriétés du dépôt sont appliquées au chemin qui fait obstacle.

Comme pour la plupart des sous-commandes, vous pouvez limiter le périmètre d'action de l'opération de ré-aiguillage à une profondeur de l'arborescence en utilisant l'option `--depth`. De la même manière, vous pouvez utiliser l'option `--set-depth` pour définir un nouveau niveau de récursion associé à la cible de ré-aiguillage. Actuellement, la profondeur de récursion d'un répertoire d'une copie de travail ne peut qu'être augmenté (aller plus profond) ; vous ne pouvez pas faire diminuer la profondeur d'un réperoire.

L'option `--relocate` indique à **svn switch** d'agir différemment : elle actualise votre copie de travail pour pointer vers *le même* répertoire du dépôt, mais sur une URL différente (typiquement lorsque l'administrateur a déplacé le dépôt sur un autre serveur ou à une autre URL sur le même serveur).

Noms alternatifs

sw

Modifie

Copie de travail.

Accède au dépôt

Oui.

Options

```
--accept ARG
--depth ARG
--diff3-cmd CMD
--force
--ignore-externals
--quiet (-q)
--relocate
--revision (-r) REV
--set-depth ARG
```

Exemples

Si vous pointez vers le répertoire `vendors`, qui a fait l'objet d'un déplacement vers le répertoire `vendors-with-fix`, et que vous voulez que votre copie de travail pointe vers cette nouvelle branche :

```
$ svn switch http://svn.red-bean.com/repos/branches/vendors-with-fix .
U  mon-projet/machin.txt
U  mon-projet/bidule.txt
U  mon-projet/truc.c
U  mon-projet/qux.c
À la révision 31.
```

Pour revenir sur la branche initiale, vous n'avez qu'à fournir l'emplacememnt dans le dépôt à partir duquel vous avez extrait votre copie de travail :

```
$ svn switch http://svn.red-bean.com/repos/trunk/vendors .
U  mon-projet/machin.txt
U  mon-projet/bidule.txt
U  mon-projet/truc.c
U  mon-projet/qux.c
À la révision 31.
```

Vous pouvez ne ré-aiguiller qu'une partie de votre copie de travail vers une branche si vous ne voulez pas basculer l'ensemble de votre copie de travail.

Parfois, il arrive qu'un administrateur change l'emplacement (ou l'emplacement virtuel) de votre dépôt. En d'autres termes, le contenu du dépôt ne change pas mais l'URL racine du dépôt est modifiée. Par exemple, le nom de machine peut changer, le schéma de l'URL ou n'importe quelle partie de l'URL qui conduit au dépôt lui-même peut changer. Plutôt que d'extraire une copie de travail entièrement nouvelle, vous pouvez utiliser la commande **svn switch** pour « ré-écrire » les méta-données administrative de votre copie de travail pour prendre en compte le nouvel emplacement. Si vous passez l'option `--relocate` à **svn switch**, Subversion contacte le dépôt pour valider la demande de déplacement (il vérifie que le dépôt est bien à la nouvelle URL) puis il modifie les méta-données d'admninistration. Aucun contenu de fichier ne sera modifié par cette opération, elle ne concerne que les méta-données de la copie de travail.

```
$ svn checkout file:///var/svn/depot test
A  test/a
A  test/b
…

$ mv depot nouvel-emplacement
$ cd test/

$ svn update
svn: Impossible d'ouvrir une session ra_local pour l'URL
svn: Le dépôt 'file:///var/svn/depot' n'a pu être ouvert

$ svn switch --relocate file:///var/svn/depot file:///tmp/nouvel-emplacement .
$ svn update
À la révision 3.
```

Soyez prudent dans l'utilisation de l'option `--relocate`. Si vous vous trompez dans les arguments, vous pouvez vous retrouver avec des URL sans aucun sens dans votre copie de travail qui peuvent rendre l'ensemble de votre espace de travail inutilisable et particulièrement difficile à réparer. Il est aussi important de comprendre exactement quand utiliser ou ne pas utiliser l'option `--relocate`. Voici une règle « au doigt mouillé » :

- Si la copie de travail doit reflèter un nouveau répertoire *à l'intérieur* du dépôt, utilisez simplement **svn switch**.

- Si la copie de travail reflète toujours le même répertoire du dépôt, mais c'est l'emplacement du dépôt lui-même qui a changé, utilisez **svn switch** avec l'option `--relocate`.

Nom

svn unlock — Déverrouiller des chemins de la copie de travail ou des URL.

Synopsis

```
svn unlock CIBLE...
```

Description

Déverrouille chaque *`CIBLE`*. Si une *`CIBLE`* est verrouillée par un autre utilisateur ou s'il n'existe pas de jeton de verrouillage valide dans la copie de travail, affiche un avertissement et continue à déverrouiller le reste des *`CIBLE`*s. Utilisez l'option `--force` pour casser un verrou appartenant à un autre utilisateur ou à une autre copie de travail.

Noms alternatifs

Aucun.

Modifie

Copie de travail, dépôt.

Accède au dépôt

Oui.

Options

```
--force
--targets NOM_FICHIER
```

Exemples

Déverrouille deux fichiers de votre copie de travail :

```
$ svn unlock arbre.jpg maison.jpg
'/arbre.jpg' déverrouillé.
'/maison.jpg' déverrouillé.
```

Déverrouille un fichier dans votre copie de travail qui est verrouillé par un autre utilisateur :

```
$ svn unlock arbre.jpg
svn: 'arbre.jpg' n'est pas verrouillé dans cette copie de travail
$ svn unlock --force arbre.jpg
'arbre.jpg' déverrouillé.
```

Déverrouille un fichier sans copie de travail :

```
$ svn unlock http://svn.red-bean.com/repos/test/arbre.jpg
'arbre.jpg' déverrouillé.
```

Pour plus de détails, reportez-vous à la section intitulée « Verrouillage ».

Nom

svn update — Mettre à jour la copie de travail.

Synopsis

```
svn update [CHEMIN...]
```

Description

svn update actualise la copie de travail par rapport au dépôt. Si aucune révision n'est spécifiée, elle actualise par rapport à la révision `HEAD`. Sinon, elle synchronise la copie de travail à la révision spécifiée par l'option `--revision`. La procédure de synchronisation suivie par **svn update** inclut la suppression des verrous dépassés (voir la section intitulée « Parfois, il suffit de faire le ménage ») trouvés dans la copie de travail.

Pour chaque élément mis à jour, elle affiche une ligne avec un caractère indiquant l'action effectuée. Ces caractères ont la signification suivante :

`A`
: Ajouté

`B`
: Verrou cassé (troisième colonne seulement — *Broken*)

`D`
: Effacé (*Deleted*)

`U`
: Mis à jour (*Updated*)

`C`
: En conflit

`G`
: Fusionné (*Merged*)

`E`
: Existant

Un caractère dans la première colonne signifie une mise à jour du fichier existant, alors que les mises à jour des propriétés de fichiers apparaissent dans la deuxième colonne. Les informations de verrouillage sont affichées dans la troisième colonne.

Comme pour la plupart des sous-commandes, vous pouvez limiter le périmètre d'action de l'opération de mise à jour à une profondeur de l'arborescence en utilisant l'option `--depth`. De la même manière, vous pouvez utiliser l'option `--set-depth` pour définir un nouveau niveau de récursion associé à la cible de mise à jour. Actuellement, la profondeur de récursion d'un répertoire d'une copie de travail ne peut qu'être augmenté (aller plus profond) ; vous ne pouvez pas faire diminuer la profondeur d'un réperoire.

Noms alternatifs

up

Modifie

Copie de travail.

Accède au dépôt

Oui.

Options

```
--accept ARG
--changelist
--depth ARG
--diff3-cmd CMD
--editor-cmd ARG
--force
--ignore-externals
--quiet (-q)
--revision (-r) REV
--set-depth ARG
```

Exemples

Récupère les modifications apportées au dépôt depuis la dernière mise à jour :

```
$ svn update
A  nouveau-rep/toggle.c
A  nouveau-rep/disclose.c
A  nouveau-rep/launch.c
D  nouveau-rep/LISEZ-MOI
Actualisé à la révision 32.
```

Vous pouvez aussi « mettre à jour » votre copie de travail vers une vieille révision (Subversion ne connait pas le concept de fichier « sticky » défini dans CVS ; reportez-vous à Annexe B, *Guide Subversion à l'usage des utilisateurs de CVS*) :

```
$ svn update -r30
A  nouveau-rep/LISEZ-MOI
D  nouveau-rep/toggle.c
D  nouveau-rep/disclose.c
D  nouveau-rep/launch.c
U  machin.c
Actualisé à la révision 30.
```

Si vous voulez examiner un seul fichier dans une révision antérieure, vous pouvez préférer l'utilisation de **svn cat** — cela ne modifiera pas votre copie de travail.

svnadmin

svnadmin est l'outil d'administration pour surveiller et réparer votre dépôt Subversion. Pour des informations détaillées sur l'administration d'un dépôt, reportez-vous à la section intitulée « svnadmin ».

Comme **svnadmin** fonctionne par un accès direct au dépôt (et ne peut ainsi être utilisé que sur la machine qui héberge le dépôt), il fait référence au dépôt par un chemin et non par une URL.

svnadmin Options

Les options de **svnadmin** sont globales, de la même manière que pour **svn** :

`--bdb-log-keep`
: (spécifique au magasin de données Berkeley DB) désactiver la suppression des fichiers de journalisation automatique de la base de données. Disposer de ces fichiers de journalisation peut être appréciable si vous devez effectuer une restauration à l'issue d'une panne catastrophique sur le dépôt.

`--bdb-txn-nosync`

(spécifique au magasin de données Berkeley DB) désactiver fsync lors des validations des transactions de la base de données. Utilisée avec la commande **svnadmin create** pour créer un dépôt avec magasin de données Berkeley DB dont l'option `DB_TXN_NOSYNC` est activée (cela améliore la vitesse du dépôt mais comporte certains risques).

`--bypass-hooks`
Ne pas utiliser le système des procédures automatiques du dépôt.

`--clean-logs`
Supprimer les fichiers de journalisation inutiles de la base de données Berkeley DB.

`--force-uuid`
Par défaut, lorsque des données sont introduites dans un dépôt qui contient déjà des révisions, **svnadmin** ignore l'`UUID` du flux dump. Cette option force le dépôt à adopter l'`UUID` indiqué dans le flux dump.

`--ignore-uuid`
Par défaut, lorsque des données sont introduites dans un dépôt qui contient déjà des révisions, **svnadmin** ignore l'`UUID` du flux dump. Cette option force le dépôt à ignorer l'`UUID` du flux (utile pour surcharger le fichier de configuration si celui-ci a l'option `--force-uuid` activée).

`--incremental`
Lors de la décharge d'une révision, ne produit que la différence avec la révision précédente plutôt que le texte complet habituel.

`--parent-dir` *`REPERTOIRE`*
Lors du chargement d'un flux dump, prend *`REPERTOIRE`* comme racine de l'arborescence plutôt que `/`.

`--pre-1.4-compatible`
Lors de la création d'un nouveau dépôt, utilise un format compatible avec les versions de Subversion antérieures à Subversion 1.4.

`--pre-1.5-compatible`
Lors de la création d'un nouveau dépôt, utilise un format compatible avec les versions de Subversion antérieures à Subversion 1.5.

`--revision` (`-r`) *`ARG`*
Spécifie une révision particulière sur laquelle s'effectue l'opération.

`--quiet`
Ne pas afficher le déroulement de l'opération — afficher uniquement les erreurs.

`--use-post-commit-hook`
Lors du chargement d'un fichier dump, exécute la procédure automatique `post-commit` du dépôt après chaque nouvelle révision chargée.

`--use-post-revprop-change-hook`
Lors du changement d'une propriété de révision, exécute la procédure automatique `post-revprop-change` du dépôt après la modification de la propriété.

`--use-pre-commit-hook`
Lors du chargement d'un fichier dump, exécute la procédure automatique `pre-commit` du dépôt avant de conclure chaque nouvelle révision chargée. Si la procédure automatique échoue, la propagation est abandonnée et la procédure de chargement s'arrête.

`--use-pre-revprop-change-hook`
Lors du changement d'une propriété de révision, exécute la procédure automatique `pre-revprop-change` du dépôt avant de modifier la propriété. Si la procédure automatique échoue, la modification est annulée et le programme s'arrête.

svnadmin Subcommands

voici les différentes sous-commandes du programme **svnadmin**.

Nom

svnadmin crashtest — Simuler l'arrêt brutal d'un processus.

Synopsis

```
svnadmin crashtest CHEMIN_DÉPÔT
```

Description

Ouvre le dépôt situé à *`CHEMIN_DÉPÔT`*, puis s'arrête. Cela simule un processus qui se termine brutalement alors qu'il détient un descripteur ouvert sur le dépôt. Cette sous-commande est utilisée pour tester le rétablissement automatique du dépôt (une nouvelle fonctionnalité de Berkeley DB 4.4). Il est peu probable que vous ayez besoin de lancer cette sous-commande.

Options

Aucun.

Exemple

```
$ svnadmin crashtest /var/svn/depot
Abandon
```

Excitant, n'est-ce pas ?

Nom

svnadmin create — Créer un nouveau dépôt vide.

Synopsis

```
svnadmin create CHEMIN_DÉPÔT
```

Description

Crée un nouveau dépôt vide à l'emplacement spécifié. Si le répertoire spécifié n'existe pas, il est créé pour vous[1]. Depuis Subversion 1.2, **svnadmin** crée les nouveaux dépôts en utilisant le magasin de données FSFS par défaut.

Bien que **svnadmin create** crée le répertoire de base pour un nouveau dépôt, il ne créera pas les répertoires intermédiaires. Par exemple, si vous avez un répertoire vide appelé `/var/svn`, créer `/var/svn/depot` fonctionne, alors que la tentative de créer `/var/svn/sous-repertoire/depot` échoue avec une erreur.

Options

```
--bdb-log-keep
--bdb-txn-nosync
--config-dir REPERTOIRE
--fs-type TYPE
--pre-1.4-compatible
--pre-1.5-compatible
```

Exemples

Créer un nouveau dépôt est aussi simple que ça :

```
$ svnadmin create /var/svn/depot
```

Dans Subversion 1.0, un dépôt est toujours créé avec un magasin de données Berkeley DB. Dans Subversion 1.1, le magasin de données par défaut est Berkeley DB, mais un magasin de données FSFS peut être demandé *via* l'option `--fs-type` :

```
$ svnadmin create /var/svn/depot --fs-type fsfs
```

[1] Rappelez-vous que **svnadmin** ne fonctionne qu'avec des *chemins* sur votre système de fichiers local, pas avec des *URL*.

Nom

svnadmin deltify — Ne stocker que les différences dans un intervalle de révisions.

Synopsis

```
svnadmin deltify [-r BAS[:HAUT]] CHEMIN_DÉPÔT
```

Description

svnadmin deltify n'existe dans les versions actuelles de Subversion que pour des raisons historiques. Cette commande est désormais obsolète.

Elle date du temps où Subersion offrait un contrôle plus fin à l'administrateur sur les stratégies de compression des données dans le dépôt. Cela s'est avéré être beaucoup de complexité pour un gain *très* faible, c'est pourquoi cette « fonctionnalité » a disparu.

Options

```
--quiet (-q)
--revision (-r) REV
```

Nom

svnadmin dump — Décharger le contenu du système de fichiers vers `stdout`.

Synopsis

```
svnadmin dump CHEMIN_DÉPÔT [-r BAS[:HAUT]] [--incremental]
```

Description

Décharge le contenu du système de fichiers vers `stdout` dans le format portable « dump », L'avancement de l'opération est envoyé vers `stderr`. Les révisions de *BAS* jusqu'à *HAUT* sont déchargées. Si aucune révision n'est spécifiée, toutes les révisions sont déchargées. Si seule *BAS* est spécifiée, décharge uniquement cette révision. Reportez-vous à la section intitulée « Migration des données d'un dépôt » pour la description pratique de l'utilisation de cette commande.

Par défaut, le flux dump Subversion ne contient qu'une seule révision (la première dans l'intervalle demandé) dans laquelle tous les fichiers et les répertoires sont indiqués comme ayant été ajoutés d'un seul coup ; les autres révisions (spécifiées par l'intervalle de révisions) ne contiennent que les fichiers et les répertoires qui ont été modifiés par ces révisions. Pour un fichier modifié, le contenu entier du fichier (en représentation plein-texte), ainsi que ses propriétés, est présent dans le fichier dump. Pour un répertoire, toutes les propriétés sont présentes.

Deux options modifient le comportement par défaut du générateur de flux dump. La première est l'option `--incremental` qui force la première révision à ne contenir que les fichiers et répertoires modifiés lors de cette révision (au lieu de l'ensemble de l'arborescence) au même format que les autres révisions d'un flux dump. Cette option est utile pour générer un fichier dump relativement petit pour être chargé dans un autre dépôt qui est déjà au courant des révisions précédentes du dépôt original.

L'autre option est `--deltas`. Cette option indique à **svnadmin dump** de, au lieu de générer des représentations en plein-texte du contenu des fichiers et la liste des propriétés, générer uniquement le delta des éléments concernés par rapport à leur version antérieure. Cette option réduit (dans certains cas drastiquement) la taille du fichier dump produit par **svnadmin dump**. Il y a cependant des inconvénients à utiliser cette option : la production de fichiers dump delta est plus consommatrice de puissance CPU, ces fichiers ne peuvent pas être traités par **svndumpfilter** et ils se révèlent être moins facilement compressés par des outils tiers tels que **gzip** ou **bzip2**.

Options

```
--deltas
--incremental
--quiet (-q)
--revision (-r) REV
```

Exemples

Décharge complète d'un dépôt :

```
$ svnadmin dump /var/svn/depot > depot-complet.dump
* Révision 0 déchargée.
* Révision 1 déchargée.
* Révision 2 déchargée.
…
```

Décharge incrémentale d'une seule transaction de votre dépôt :

```
$ svnadmin dump /var/svn/depot -r 21 --incremental > increment.dump
* Révision 21 déchargée.
```

Nom

svnadmin help — À l'aide !

Synopsis

```
svnadmin help [SOUS-COMMANDE...]
```

Description

Cette sous-commande est utile si vous êtes abandonné sur une île déserte sans connexion Internet ni copie de ce livre.

Noms alternatifs

?, **h**

Nom

svnadmin hotcopy — Effectuer une copie à chaud d'un dépôt.

Synopsis

```
svnadmin hotcopy CHEMIN_DÉPÔT NOUVEAU_CHEMIN_DÉPÔT
```

Description

Cette commande effectue une sauvegarde « à chaud » de votre dépôt, y compris les procédures automatiques, les fichiers de configuration et, bien sûr, les fichiers de la base de données. Si vous spécifiez l'option `--clean-logs`, **svnadmin** effectue la copie à chaud de votre dépôt et supprime ensuite les journaux inutiles de la base de données Berkeley DB du dépôt original. Vous pouvez lancer cette commande à n'importe quel moment pour effectuer une sauvegarde de votre dépôt, indépendamment de l'activité concernant votre dépôt.

Options

`--clean-logs`

Comme indiqué dans la section intitulée « Berkeley DB », les dépôts Berkeley DB copiés à chaud *ne sont pas* portables d'un système d'exploitation à l'autre, ni entre des machines dont l'architecture CPU est différente (ordonnancement différent des bits de poids fort et de poids faible ou « *endianness* »).

Nom

svnadmin list-dblogs — Demander à la base de données Berkeley DB quels sont les fichiers de journalisation existants pour un dépôt Subversion donné (s'applique uniquement aux dépôts dont le magasin de données est du type `bdb`).

Synopsis

```
svnadmin list-dblogs CHEMIN_DÉPÔT
```

Description

Le gestionnaire de bases de données Berkeley DB crée des fichiers de journalisation de tous les changements apportés au dépôt, ce qui permet un rétablissement si une catastrophe survient. À moins d'activer l'option `DB_LOG_AUTOREMOVE`, les fichiers de journalisation s'accumulent, bien que la plupart ne soit plus utilisée et puisse être supprimée pour récupérer de l'espace disque. Reportez-vous à la section intitulée « Gestion de l'espace disque » pour davantage d'information.

Nom

svnadmin list-unused-dblogs — Demander au gestionnaire Berkeley DB quels fichiers de journalisation peuvent être effacés (s'applique uniquement aux dépôts dont le magasin de données est du type `bdb`).

Synopsis

```
svnadmin list-unused-dblogs CHEMIN_DÉPÔT
```

Description

Le gestionnaire de bases de données Berkeley DB crée des fichiers de journalisation de tous les changements apportés au dépôt, ce qui permet un rétablissement si une catastrophe survient. À moins d'activer l'option `DB_LOG_AUTOREMOVE`, les fichiers de journalisation s'accumulent, bien que la plupart ne soit plus utilisée et puisse être supprimée pour récupérer de l'espace disque. Reportez-vous à la section intitulée « Gestion de l'espace disque » pour davantage d'information.

Exemple

Supprime tous les fichiers de journalisation inutiles dans le dépôt :

```
$ svnadmin list-unused-dblogs /var/svn/depot
/var/svn/depot/log.0000000031
/var/svn/depot/log.0000000032
/var/svn/depot/log.0000000033

$ svnadmin list-unused-dblogs /var/svn/depot | xargs rm
## je fais de la place !
```

Nom

svnadmin load — Charger un flux dump à partir de `stdin`.

Synopsis

```
svnadmin load CHEMIN_DÉPÔT
```

Description

Charge un flux dump depuis `stdin`, propageant de nouvelles révisions dans le système de fichiers du dépôt. L'avancement de l'opération est affiché sur `stdout`.

Options

```
--force-uuid
--ignore-uuid
--parent-dir
--quiet (-q)
--use-post-commit-hook
--use-pre-commit-hook
```

Exemples

Cet exemple montre le début de chargement d'un dépôt à partir d'un fichier de sauvegarde (créé, bien sûr, avec la commande **svnadmin dump**) :

```
$ svnadmin load /var/svn/restauré < sauvegarde-depot
<<< Début d'une nouvelle transaction basée sur la révision 1
     * ajout de : test ... fait.
     * ajout de : test/a ... fait.
…
```

Ou, si vous voulez le charger dans un sous-répertoire :

```
$ svnadmin load --parent-dir nouveau/sous-repertoire/pour/projet \
                /var/svn/restauré < sauvegarde-depot
<<< Début d'une nouvelle transaction basée sur la révision 1
     * ajout de : test ... fait.
     * ajout de : test/a ... fait.
…
```

Nom

svnadmin lslocks — Afficher la description de tous les verrous.

Synopsis

```
svnadmin lslocks CHEMIN_DÉPÔT [CHEMIN-DANS-DEPOT]
```

Description

Affiche la description de tous les verrous dans le dépôt *`CHEMIN_DÉPÔT`* sous le chemin *`CHEMIN-DANS-DEPOT`*. Si *`CHEMIN-DANS-DEPOT`* n'est pas fourni, la valeur par défaut est la racine du dépôt.

Options

Aucun.

Exemple

Cet exemple affiche l'unique fichier verrouillé dans le dépôt situé dans `/var/svn/depot` :

```
$ svnadmin lslocks /var/svn/depot
Chemin : /arbre.jpg
Chaîne UUID : opaquelocktoken:ab00ddf0-6afb-0310-9cd0-dda813329753
Propriétaire : harry
Créé : 2005-07-08 17:27:36 -0500 (ven. 08 Jul 2005)
Expire :
Commentaire (1 ligne):
Retouche sur la branche la plus haute du cyprès plat au premier plan.
```

Nom

svnadmin lstxns — Afficher le nom de toutes les transactions mortes.

Synopsis

```
svnadmin lstxns CHEMIN_DÉPÔT
```

Description

Affiche le nom de toutes les transactions inachevées. Reportez-vous à la section intitulée « Suppression des transactions mortes » pour savoir comment des transactions peuvent être mortes et ce que vous devriez faire avec.

Exemple

Liste toutes les transactions en cours dans le dépôt :

```
$ svnadmin lstxns /var/svn/depot/
1w
1x
```

Nom

svnadmin recover — Remettre la base de données d'un dépôt dans un état consistant (s'applique uniquement aux dépôts utilisant un magasin de données de type `bdb`). En complément, si le fichier `depot/conf/passwd` n'existe pas, un fichier de mots de passe par défaut est créé.

Synopsis

```
svnadmin recover CHEMIN_DÉPÔT
```

Description

Lancez cette commande si vous obtenez une erreur indiquant que votre dépôt doit être rétabli.

Options

```
--wait
```

Exemple

Rétablit un dépôt planté :

```
$ svnadmin recover /var/svn/depot/
Verrou du dépôt acquis.
Patientez ; le rétablissement du dépôt peut être long...

Fin du rétablissement.
La dernière révision du dépôt est 34
```

Rétablir la base de données nécessite d'obtenir un verrou exclusif sur le dépôt (cela ressemble à « verrou de base de doonées »; lisez l'encadré Les trois types de « verrous ».) Si un autre processus est en train d'accéder au dépôt, **svnadmin recover** se termine avec l'erreur :

```
$ svnadmin recover /var/svn/depot
svn: Échec de l'obtention de l'accès exclusif au dépôt ; peut-être
processus tel 'httpd', 'svnserve' ou 'svn' a-t-il ouvert le dépôt ?

$
```

L'option `--wait`, force **svnadmin recover** à attendre que les autres processus se déconnectent :

```
$ svnadmin recover /var/svn/depot --wait
Attente du verrou sur le dépôt ; un autre processus le tient-il ?

### le temps passe…

Verrou du dépôt acquis.
Patientez ; le rétablissement du dépôt peut être long...

Fin du rétablissement.
La dernière révision du dépôt est 34
```

Nom

svnadmin rmlocks — Effacer inconditionnellement un ou plusieurs verrous du dépôt.

Synopsis

```
svnadmin rmlocks CHEMIN_DÉPÔT CHEMIN_VERROUILLÉ...
```

Description

Supprime un ou plusieurs verrous de chaque *`CHEMIN_VERROUILLÉ`*.

Options

Aucun.

Exemple

Cet exemple supprime les verrous posés sur `arbre.jpg` et `maison.jpg` dans le dépôt situé dans `/var/svn/depot`:

```
$ svnadmin rmlocks /var/svn/depot arbre.jpg maison.jpg
'/arbre.jpg' déverrouillé.
'/maison.jpg' déverrouillé.
```

Nom

svnadmin rmtxns — Supprimer des transactions d'un dépôt.

Synopsis

```
svnadmin rmtxns CHEMIN_DÉPÔT TXN_NAME...
```

Description

Supprime les transactions en cours dans le dépôt. Cette fonctionnalités est décrite en détail dans la section intitulée « Suppression des transactions mortes ».

Options

```
--quiet (-q)
```

Exemples

Supprime les transactions dont les noms sont fournis :

```
$ svnadmin rmtxns /var/svn/depot/ 1w 1x
```

Les choses étant bien faites, la sortie de **lstxns** convient bien comme entrée de **rmtxns** :

```
$ svnadmin rmtxns /var/svn/depot/  `svnadmin lstxns /var/svn/depot/`
```

Cet exemple a supprimé toutes les transactions inachevées de votre dépôt.

Nom

svnadmin setlog — Définir l'entrée du journal pour une révision.

Synopsis

```
svnadmin setlog CHEMIN_DÉPÔT -r REVISION FICHIER
```

Description

Définit l'entrée du journal pour la révision *`REVISION`* au contenu du fichier *`FICHIER`*.

Cela revient à utiliser **svn propset** avec l'option `--revprop` pour définir la propriété `svn:log` de la révision, sauf que vous pouvez également utiliser l'option `--bypass-hooks` pour éviter les procédures automatiques pre-commit et post-commit, ce qui est utile si la procédure automatique `pre-revprop-change` n'autorise pas les modifications de propriétés de révision.

Les propriétés de révision ne sont pas gérées en versions, donc cette commande écrase de manière irréversible la précédente entrée dans le journal.

Options

```
--bypass-hooks
--revision (-r) REV
```

Exemple

Cet exemple définit l'entrée du journal pour la révision 19 au contenu du fichier `msg`:

```
$ svnadmin setlog /var/svn/depot/ -r 19 msg
```

Nom

svnadmin setrevprop — Définir une propriété de révision.

Synopsis

```
svnadmin setrevprop CHEMIN_DÉPÔT -r REVISION NOM FICHIER
```

Description

Définit la propriété *`NOM`* de la révision *`REVISION`* au contenu du fichier *`FICHIER`*. Utilisez l'option `--use-pre-revprop-change-hook` ou `--use-post-revprop-change-hook` pour activer les procédures automatiques relatives aux propriétés de révision (par exemple si vous voulez notifier la modification aux utilisateurs par la procédure automatique `post-revprop-change-hook`).

Options

```
--revision (-r) ARG
--use-post-revprop-change-hook
--use-pre-revprop-change-hook
```

Exemple

Cet exemple définit la propriété `photo-du-depot` au contenu du fichier `sandwich.png` :

```
$svnadmin setrevprop /var/svn/depot -r 0 photo-du-depot sandwich.png
```

Comme vous pouvez le constater, **svnadmin setrevprop** n'affiche rien en cas de succès.

Nom

svnadmin setuuid — Redéfinir l'identifiant unique (UUID) du dépôt.

Synopsis

```
svnadmin setuuid CHEMIN_DÉPÔT [NOUVEAU_UUID]
```

Description

Redéfinit l'identifiant unique (UUID) du dépôt situé dans *`CHEMIN_DÉPÔT`*. Si *`NOUVEAU_UUID`* est fourni, il est utilisé comme nouvel identifiant ; sinon, un nouvel identifiant est généré pour le dépôt.

Options

Aucun.

Exemple

Si vous avez synchronisé (avec **svnsync**) `/var/svn/depot` vers `/var/svn/nouveau-depot` et que vous voulez utiliser `nouveau-depot` comme dépôt par défaut, vous voudrez sûrement définir l'identiant unique de `nouveau-depot` à la même valeur que l'identifiant de `depot` ainsi vos utilisateurs n'auront pas à extraire une nouvelle copie de travail pour s'adapter à la nouvelle configuration :

```
$ svnadmin setuuid /var/svn/nouveau-depot 2109a8dd-854f-0410-ad31-d604008985ab
```

Comme vous pouvez le constater, **svnadmin setuuid** n'affiche rien en cas de succès.

Nom

svnadmin upgrade — Mettre le dépôt à la dernière version supportée du schéma.

Synopsis

```
svnadmin upgrade CHEMIN_DÉPÔT
```

Description

Met le dépôt situé à *`CHEMIN_DÉPÔT`* à la dernière version supportée du schéma.

Cette fonctionnalité est un « petit plus » offert aux administrateurs qui veulent profiter des nouvelles fonctionnalités de Subversion sans avoir à passer par une opération coûteuse de déchargement et chargement complet du dépôt. Ainsi, **svnadmin upgrade** ne fait que le strict minimum pour effectuer la mise à jour, tout en gardant l'intégrité du dépôt. Alors qu'un déchargement suivi d'un chargement (**svnadmin dump** puis **svnadmin load**) garantissent un état optimisé du dépôt, **svnadmin upgrade** ne le garantit pas.

Vous devriez *toujours* effectuer une sauvegarde avant **upgrade**.

Options

Aucun.

Exemple

Met le dépôt `/var/depot/svn` à la dernière version du schéma :

```
$ svnadmin upgrade /var/depot/svn
Verrou du dépôt acquis.
Patientez ; la mise à jour du dépôt peut être longue...

Fin de la mise à jour.
```

Nom

svnadmin verify — Vérifier les données stockées dans le dépôt.

Synopsis

```
svnadmin verify CHEMIN_DÉPÔT
```

Description

Lancez cette commande si vous désirez vérifier l'intégrité de votre dépôt. Elle parcourt toutes les révisions du dépôt et les décharge en interne sans se préoccuper du flux produit — c'est une bonne idée de lancer régulièrement ce programme pour prévenir les pannes latentes de disque dur et le vieillissement lié au stockage sur supports magnétiques (« bitrot »). Si cette commande échoue (ce qu'elle fait au moindre problème), cela signifie qu'au moins une révision de votre dépôt est corrompue ; vous devriez alors restaurer la révision corrompue depuis une sauvegarde (vous avez bien fait une sauvegarde, n'est-ce pas ?).

Options

```
--quiet (-q)
--revision (-r) ARG
```

Exemple

Cet exemple vérifie un dépôt planté :

```
$ svnadmin verify /var/svn/depot/
* Révision 1729 vérifiée.
```

svnlook

svnlook est un utilitaire en ligne de commande pour examiner le contenu d'un dépôt Subversion. Il n'effectue aucune modification sur le dépôt, se contentant juste de « jeter des coups d'œil ». **svnlook** est utilisé typiquement par les procédures automatiques, mais un administrateur de dépôt peut aussi y trouver un intérêt à des fins de diagnostic.

Comme **svnlook** fonctionne par un accès direct au dépôt (et ne peut ainsi être utilisé que sur la machine qui héberge le dépôt), il fait référence au dépôt par un chemin et non par une URL.

Si aucune révision ou transaction n'est spécifiée, **svnlook** utilise par défaut la révision la plus jeune (c'est-à-dire récente) du dépôt.

Options de svnlook

Les options de **svnlook** sont globales, de la même manière que pour **svn** et **svnadmin** ; Cependant, la plupart des options ne s'applique qu'à une seule sous-commande puisque le périmètre des fonctionnalités de **svnlook** est (intentionnellement) limité :

`--copy-info`
: Indique à **svnlook changed** de détailler les informations relatives aux sources de copies.

`--no-diff-deleted`
: Empêche **svnlook diff** d'afficher les différences pour les fichiers effacés. Le comportement par défaut pour un fichier effacé dans une transaction/révision est d'afficher les différences que vous verriez si le fichier était toujours présent mais avec un contenu vide.

`--no-diff-added`
: Empêche **svnlook diff** d'afficher les différences pour les fichiers ajoutés. Le comportement par défaut pour un fichier

ajouté est d'afficher les différences que vous verriez si vous aviez ajouté tout le contenu au fichier auparavant vide.

`--revision` (`-r`)
Spécifie le numéro de révision que vous voulez examiner.

`--revprop`
Agit sur une propriété de révision au lieu d'une propriété de fichier ou de répertoire. Cette option nécessite qu'un numéro de révision soit fourni à l'option `--revision` (`-r`).

`--transaction` (`-t`)
Spécifie un identifiant de transaction particulière que vous voulez examiner.

`--show-ids`
Affiche les identifiants des nœuds de révision du système de fichiers pour chaque chemin dans l'arborescence du système de fichiers.

svnlook Subcommands

Voici les différentes sous-commandes du programme **svnlook**.

Nom

svnlook author — Afficher l'auteur.

Synopsis

```
svnlook author CHEMIN_DÉPÔT
```

Description

Affiche l'auteur d'une révision ou transaction dans le dépôt.

Options

```
--revision (-r) REV
--transaction (-t) TXN
```

Exemple

svnlook author est pratique, mais pas très excitant :

```
$ svnlook author -r 40 /var/svn/depot
sally
```

Nom

svnlook cat — Afficher le contenu d'un fichier.

Synopsis

```
svnlook cat CHEMIN_DÉPÔT CHEMIN_DANS_DÉPÔT
```

Description

Affiche le contenu d'un fichier.

Options

```
--revision (-r) REV
--transaction (-t) TXN
```

Exemple

La commande suivante affiche le contenu du fichier `/trunk/LISEZ-MOI` pour la transaction `ax8` :

```
$ svnlook cat -t ax8 /var/svn/depot /trunk/LISEZ-MOI

               Subversion, a version control system.
               =====================================

$LastChangedDate: 2003-07-17 10:45:25 -0500 (Thu, 17 Jul 2003) $

Contents:

     I. A FEW POINTERS
    II. DOCUMENTATION
   III. PARTICIPATING IN THE SUBVERSION COMMUNITY
…
```

Nom

svnlook changed — Afficher les chemins modifiés.

Synopsis

```
svnlook changed CHEMIN_DÉPÔT
```

Description

Affiche les chemins qui ont été modifiés par une révision ou une transaction particulière, avec les mêmes codes lettres que **svn status** dans les deux premières colonnes :

`'A '`
: Élément ajouté au dépôt.

`'D '`
: Élément supprimé du dépôt.

`'U '`
: Le contenu du fichier a été modifié.

`'_U'`
: Les propriétés de l'élément ont été modifiées ; notez le caractère « souligné » en tête.

`'UU'`
: Le contenu du fichier et les propriétés ont été modifiés.

Les fichiers et les répertoires peuvent être distingués par le fait que les noms de répertoires sont suivis d'une barre oblique « `/` ».

Options

```
--copy-info
--revision (-r) REV
--transaction (-t) TXN
```

Exemples

Cet exemple montre la liste de toutes les modifications apportées aux fichiers et aux répertoires par la révision 39 sur un dépôt de test. Notez que le premier élément est un répertoire comme l'indique la barre oblique (`/`) finale :

```
$ svnlook changed -r 39 /var/svn/depot
A   trunk/magasin/epicerie/
A   trunk/magasin/epicerie/chips.txt
A   trunk/magasin/epicerie/sandwich.txt
A   trunk/magasin/epicerie/vinaigre.txt
U   trunk/magasin/boulangerie/petit-pain.txt
_U  trunk/magasin/boulangerie/croissant.txt
UU  trunk/magasin/boulangerie/chausson-aux-pommes.txt
D   trunk/magasin/boulangerie/baguette.txt
```

Voici un exemple qui montre une révision dans laquelle un fichier a été renommé :

```
$ svnlook changed -r 64 /var/svn/depot
A   trunk/magasin/boulangerie/toast.txt
```

```
D   trunk/magasin/boulangerie/pain.txt
```

Malheureusement, rien dans l'affichage précédent n'indique la relation entre le fichier supprimé et le fichier ajouté. Utilisez l'option `--copy-info` pour faire apparaître cette relation :

```
$ svnlook changed -r 64 --copy-info /var/svn/depot
A + trunk/magasin/boulangerie/toast.txt
    (from trunk/magasin/boulangerie/pain.txt:r63)
D   trunk/magasin/boulangerie/pain.txt
```

Nom

svnlook date — Afficher la date de dernière modification.

Synopsis

```
svnlook date CHEMIN_DÉPÔT
```

Description

Affiche la date de la révision ou de la transaction dans le dépôt.

Options

```
--revision (-r) REV
--transaction (-t) TXN
```

Exemple

Cet exemple montre la date de la révision 40 d'un dépôt test :

```
$ svnlook date -r 40 /var/svn/depot/
2003-02-22 17:44:49 -0600 (sam. 22 févr. 2003)
```

Nom

svnlook diff — Afficher les différences pour les fichiers et les propriétés modifiés.

Synopsis

```
svnlook diff CHEMIN_DÉPÔT
```

Description

Affiche les différences des fichiers et propriétés modifiés au style GNU.

Options

```
--diff-copy-from
--no-diff-added
--no-diff-deleted
--revision (-r) REV
--transaction (-t) TXN
```

Exemple

Cet exemple montre un fichier nouvellement ajouté (vide), un fichier effacé et un fichier copié :

```
$ svnlook diff -r 40 /var/svn/depot/
Copié: oeuf.txt (from rev 39, trunk/magasin/epicerie/vinaigre.txt)

Ajouté: trunk/magasin/epicerie/soda.txt
===================================================================

Modifié: trunk/magasin/epicerie/sandwich.txt
===================================================================
--- trunk/magasin/epicerie/sandwich.txt (original)
+++ trunk/magasin/epicerie/sandwich.txt 2003-02-22 17:45:04.000000000 -0600
@@ -0,0 +1 @@
+N'oublies pas la mayo !

Modifié: trunk/magasin/epicerie/logo.jpg
===================================================================
(Binary files differ)

Effacé: trunk/magasin/epicerie/chips.txt
===================================================================

Effacé: trunk/magasin/epicerie/vinaigre.txt
===================================================================
```

Si un fichier possède une propriété `svn:mime-type` d'un type non textuel, les différences ne sont pas affichées explicitement.

Nom

svnlook dirs-changed — Afficher les répertoires modifiés.

Synopsis

```
svnlook dirs-changed CHEMIN_DÉPÔT
```

Description

Affiche les répertoires eux-même modifiés (édition de propriétés) ou dont les fichiers contenus ont été changés.

Options

```
--revision (-r) REV
--transaction (-t) TXN
```

Exemple

Cet exemple montre les répertoires qui ont été modifiés par la révision 40 sur notre dépôt de test :

```
$ svnlook dirs-changed -r 40 /var/svn/depot
trunk/magasin/epicerie/
```

Nom

svnlook help — À l'aide !

Synopsis

```
Aussi svnlook -h et svnlook -?.
```

Description

Affiche le message d'aide de **svnlook**. Cette commande, comme sa consœur **svn help**, est votre amie, même si vous ne l'appelez plus jamais et que vous avez oublié de l'inviter à votre dernière soirée.

Options

Aucun.

Noms alternatifs

?, **h**

Nom

svnlook history — Afficher l'historique d'un chemin dans le dépôt (ou de la racine du dépôt si aucun chemin n'est spécifié).

Synopsis

```
svnlook history CHEMIN_DÉPÔT [CHEMIN_DANS_DÉPÔT]
```

Description

Affiche l'historique d'un chemin dans le dépôt (ou de la racine du dépôt si aucun chemin n'est spécifié).

Options

```
--limit (-l) ARG
--revision (-r) REV
--show-ids
```

Exemple

Cet exemple affiche l'historique pour le chemin `/branches/bookstore` pour la révision 13 dans notre dépôt de test :

```
$ svnlook history -r 13 /var/svn/depot /branches/bookstore --show-ids
REVISION   CHEMIN <ID>
--------   ---------
      13   /branches/bookstore <1.1.r13/390>
      12   /branches/bookstore <1.1.r12/413>
      11   /branches/bookstore <1.1.r11/0>
       9   /trunk <1.0.r9/551>
       8   /trunk <1.0.r8/131357096>
       7   /trunk <1.0.r7/294>
       6   /trunk <1.0.r6/353>
       5   /trunk <1.0.r5/349>
       4   /trunk <1.0.r4/332>
       3   /trunk <1.0.r3/335>
       2   /trunk <1.0.r2/295>
       1   /trunk <1.0.r1/532>
```

Nom

svnlook info — Afficher l'auteur, la date, la taille et le contenu de l'entrée du journal.

Synopsis

```
svnlook info CHEMIN_DÉPÔT
```

Description

Affiche l'auteur, la date, la taille (en octets) et le contenu de l'entrée du journal, suivis d'un caractère de fin de ligne.

Options

```
--revision (-r) REV
--transaction (-t) TXN
```

Exemple

Cet exemple affiche les informations relatives à la révision 40 dans notre dépôt de test :

```
$ svnlook info -r 40 /var/svn/depot
sally
2003-02-22 17:44:49 -0600 (sam. 22 févr. 2003)
16
Réorganisé le déjeuner.
```

Nom

svnlook lock — Décrire le verrou sur le chemin dans le dépôt, s'il existe.

Synopsis

```
svnlook lock CHEMIN_DÉPÔT CHEMIN_DANS_DÉPÔT
```

Description

Affiche tous les informations disponibles sur le verrou appliqué sur *`CHEMIN_DANS_DÉPÔT`*. Si *`CHEMIN_DANS_DÉPÔT`* n'est pas verrouillé, n'affiche rien.

Options

Aucun.

Exemple

Cet exemple affiche la description du verrou posé sur le fichier `arbre.jpg`:

```
$ svnlook lock /var/svn/depot arbre.jpg
Chaîne UUID : opaquelocktoken:ab00ddf0-6afb-0310-9cd0-dda813329753
Propriétaire du verrou : harry
Verrou créé: 2005-07-08 17:27:36 -0500 (ven. 08 jul 2005)
Expire :
Commentaire (1 ligne):
Retouche sur la branche la plus haute du cyprès plat au premier plan.
```

Nom

svnlook log — Afficher l'entrée du journal.

Synopsis

```
svnlook log CHEMIN_DÉPÔT
```

Description

Affiche l'entrée du journal, suivie par un caractère de fin de ligne.

Options

```
--revision (-r) REV
--transaction (-t) TXN
```

Exemple

Cet exemple affiche l'entrée du journal pour la révision 40 dans notre dépôt de test :

```
$ svnlook log /var/svn/depot/
Réorganisé le déjeuner.
```

Nom

svnlook propget — Afficher la valeur brute de la propriété pour un chemin du dépôt.

Synopsis

```
svnlook propget CHEMIN_DÉPÔT PROP_NOM [CHEMIN_DANS_DÉPÔT]
```

Description

Affiche la valeur brute de la propriété pour un chemin du dépôt.

Noms alternatifs

pg, **pget**

Options

```
--revision (-r) REV
--revprop
--transaction (-t) TXN
```

Exemple

Cet exemple affiche la valeur de la propriété « assaisonnement » du fichier `/trunk/sandwich` pour la révision `HEAD` :

```
$ svnlook pg /var/svn/depot assaisonnement /trunk/sandwich
moutarde
```

Nom

svnlook proplist — Afficher les noms et valeurs des propriétés d'un fichier ou d'un répertoire.

Synopsis

```
svnlook proplist CHEMIN_DÉPÔT [CHEMIN_DANS_DÉPÔT]
```

Description

Liste les propriétés d'un chemin dans le dépôt. Avec l'option `--verbose`, affiche également les valeurs afférentes.

Noms alternatifs

pl, **plist**

Options

```
--revision (-r) REV
--revprop
--transaction (-t) TXN
--verbose (-v)
```

Exemples

Cet exemple affiche le nom des propriétés du fichier `/trunk/LISEZ-MOI` pour la révision `HEAD` :

```
$ svnlook proplist /var/svn/depot /trunk/LISEZ-MOI
  original-author
  svn:mime-type
```

Voici la même commande que dans l'exemple précédent, mais cette fois en affichant aussi les valeurs des propriétés :

```
$ svnlook --verbose proplist /var/svn/depot /trunk/LISEZ-MOI
  original-author : harry
  svn:mime-type : text/plain
```

Nom

svnlook tree — Afficher l'arborescence.

Synopsis

```
svnlook tree CHEMIN_DÉPÔT [CHEMIN_DANS_DÉPÔT]
```

Description

Affiche l'arborescence à partir de *CHEMIN_DANS_DÉPÔT* (ou de la racine du dépôt si non précisé). Affiche aussi, en option, les identifiants de nœuds de révision.

Options

```
--full-paths
--non-recursive (-N)
--revision (-r) REV
--show-ids
--transaction (-t) TXN
```

Exemple

Cet exemple affiche l'arborescence (avec les identifiants de nœuds) pour la révision 13 dans notre dépôt de test :

```
$ svnlook tree -r 13 /var/svn/depot --show-ids
/ <0.0.r13/811>
 trunk/ <1.0.r9/551>
  bouton.c <2.0.r9/238>
  Makefile <3.0.r7/41>
  entier.c <4.0.r6/98>
 branches/ <5.0.r13/593>
  bookstore/ <1.1.r13/390>
   bouton.c <2.1.r12/85>
   Makefile <3.0.r7/41>
   entier.c <4.1.r13/109>
```

Nom

svnlook uuid — Afficher l'`UUID` (identifiant unique) du dépôt.

Synopsis

```
svnlook uuid CHEMIN_DÉPÔT
```

Description

Affiche l'`UUID` (identifiant unique — *universal unique identifier*) du dépôt. Le client Subversion utilise cet identifiant pour distinguer les dépôts entre eux.

Options

Aucun.

Exemple

```
$ svnlook uuid /var/svn/depot
e7fe1b91-8cd5-0310-98dd-2f12e793c5e8
```

Nom

svnlook youngest — Afficher le numéro de la révision la plus récente.

Synopsis

```
svnlook youngest CHEMIN_DÉPÔT
```

Description

Affiche le numéro de la révision la plus récente d'un dépôt.

Options

Aucun.

Exemple

Cet exemple affiche le numéro de la révision la plus récente de notre dépôt de test :

```
$ svnlook youngest /var/svn/depot/
42
```

svnsync

svnsync est l'outil de réplication de dépôt à distance de Subversion. En clair, il vous permet de rejouer les propagations d'un dépôt sur un autre dépôt.

Dans tout scénario de réplication, il y a deux dépôts : le dépôt source et le dépôt miroir (ou « destination »). Le dépôt source est le dépôt à partir duquel **svnsync** lit les révisions. Le dépôt miroir est le dépôt sur lequel **svnsync** applique les propagations lues sur le dépôt source. Chacun des dépôts peut être un dépôt local ou distant (ils sont toujours uniquement désignés par leur URL).

Le processus **svnsync** requiert un accès uniquement en lecture sur le dépôt source ; il ne tente jamais aucune modification sur celui-ci. En revanche, bien évidemment, **svnsync** a besoin d'un accès en lecture et écriture sur le dépôt miroir.

svnsync est particulièrement sensible aux modifications faites sur le dépôt miroir qui ne sont pas issues d'une opération de réplication. Pour éviter ce genre d'ennui, il est recommandé que **svnsync** soit le seul processus autorisé à modifier le dépôt miroir.

svnsync Options

Les options de **svnsync** sont globales, de même que pour **svn** et **svnadmin** :

`--config-dir` *`REPERTOIRE`*
: Indique à Subversion de lire les informations de configuration dans le répertoire spécifié plutôt qu'à l'emplacement par défaut (`.subversion` dans le répertoire de l'utilisateur).

`--no-auth-cache`
: *Ne pas conserver* les éléments d'authentification (par exemple l'identifiant et le mot de passe) dans les répertoires de configuration de Subversion.

`--non-interactive`
: Dans le cas d'un échec d'authentification ou de droits insuffisants, ne demande pas d'éléments d'authentification (par exemple identifiant et mot de passe) de manière interactive. Cette option est utile quand vous lancez Subversion dans un

script totalement automatique et qu'il est plus pertinent de faire échouer Subversion plutôt que d'attendre une réponse interactive.

`--quiet` (`-q`)
N'afficher que ce qui est essentiel pendant l'opération.

`--source-password` *`MDP`*
Précise le mot de passe à utiliser pour s'authentifier auprès du serveur Subversion source. Si cette option n'est pas fournie ou si elle ne permet pas de s'authentifier correctement, Subversion vous demande, en tant que de besoin, le mot de passe de manière interactive.

`--source-username` *`NOM`*
Précise le nom d'utilisateur à utiliser pour s'authentifier auprès du serveur Subversion source. Si cette option n'est pas fournie ou si elle ne permet pas de s'authentifier correctement, Subversion vous demande, en tant que de besoin, le nom d'utilisateur de manière interactive.

`--sync-password` *`MDP`*
Précise le mot de passe à utiliser pour s'authentifier auprès du serveur Subversion destination. Si cette option n'est pas fournie ou si elle ne permet pas de s'authentifier correctement, Subversion vous demande, en tant que de besoin, le mot de passe de manière interactive.

`--sync-username` *`NOM`*
Précise le nom d'utilisateur à utiliser pour s'authentifier auprès du serveur Subversion destination. Si cette option n'est pas fournie ou si elle ne permet pas de s'authentifier correctement, Subversion vous demande, en tant que de besoin, le nom d'utilisateur de manière interactive.

svnsync Subcommands

Voici les différentes sous-commandes du programme **svnsync**.

Nom

svnsync copy-revprops — Copier toutes les propriétés de révision pour une révision donnée (ou un intervalle de révisions) du dépôt source vers le dépôt miroir.

Synopsis

```
svnsync copy-revprops URL_DEST [REV[:REV2]
```

Description

Comme les propriétés de révision Subversion peuvent être modifiées à n'importe quel moment, il est possible que des propriétés d'une révision donnée soient modifiées après que la révision a été répliquée sur le dépôt miroir. Comme la commande **svnsync synchronize** n'opère que sur un intervalle de révisions qui n'ont pas encore été répliquées, elle ne remarquera pas la modification d'une propriété en dehors de cet intervalle. Si rien n'est fait pour contrer ce phénomène, cela entraine une divergence entre les valeurs des propriétés de révision du dépôt source et du dépôt miroir. **svnsync copy-revprops** répond à ce problème : utilisez-la pour resynchroniser les propriétés de révision pour une révision donnée ou pour un intervalle de révisions.

Noms alternatifs

Aucun.

Options

```
--config-dir REPERTOIRE
--no-auth-cache
--non-interactive
--quiet (-q)
--source-password ARG
--source-username ARG
--sync-password ARG
--sync-username ARG
```

Exemple

Re-synchronise les propriétés de révision pour une seule révision :

```
$ svnsync copy-revprops file:///var/svn/depot-miroir 6
Propriétés copiées pour la révision 6.
$
```

Nom

svnsync help — À l'aide !

Synopsis

```
svnsync help
```

Description

Cette sous-commande est utile quand vous vous retrouvez prisonnier dans une prison à l'étranger sans connexion Internet ni exemplaire de ce livre, mais que vous avez un hotspot WiFi à portée et que vous voulez répliquer votre dépôt vers le serveur de sauvegarde de Marcel Au Couteau situé dans une cellule du bloc D.

Noms alternatifs

Aucun.

Options

Aucun.

Nom

svnsync initialize — Initialiser un dépôt miroir pour une synchronisation à partir d'un dépôt source.

Synopsis

```
svnsync initialize URL_MIROIR URL_SOURCE
```

Description

svnsync initialize vérifie que le dépôt répond aux exigences d'un dépôt miroir vierge (il n'y a pas d'historique et la modification des propriétés de révision est autorisée), puis enregistre les informations administratives initiales qui associent le dépôt miroir au dépôt source. C'est la première opération **svnsync** que vous lancez sur un dépôt miroir « en devenir ».

Noms alternatifs

init

Options

```
--config-dir REPERTOIRE
--no-auth-cache
--non-interactive
--quiet (-q)
--source-password ARG
--source-username ARG
--sync-password ARG
--sync-username ARG
```

Exemples

Échoue dans l'initialisation d'un dépôt miroir en raison de l'incapacité à modifier les propriétés de révision :

```
$ svnsync initialize file:///var/svn/depot-miroir http://svn.exemple.com/depot
svnsync: Le dépôt n'est pas configuré pour accepter les modifications de
propriétés de révision ; parler à l'administrateur de la procédure
automatique (hook) pre-revprop-change
$
```

Initialise un dépôt en tant que miroir, après avoir créé une procédure automatique `pre-revprop-change` qui autorise les modifications des propriétés de révision :

```
$ svnsync initialize file:///var/svn/depot-miroir http://svn.exemple.com/depot
Propriétés copiées pour la révision 0.
$
```

Nom

svnsync synchronize — Transfèrer toutes les révisions en attente depuis le dépôt source vers le dépôt miroir.

Synopsis

```
svnsync synchronize URL_DEST
```

Description

La commande **svnsync synchronize** effectue le gros du travail de réplication. Après avoir consulté le dépôt miroir pour connaître les révisions déjà copiées, elle commence la copie des révisions qui n'ont pas encore été copiées depuis le dépôt source.

svnsync synchronize peut être interrompue et redémarrée sans souci.

Depuis Subversion 1.5, vous pouvez limiter **svnsync** à un sous-répertoire du dépôt source en spécifiant le sous-répertoire dans l'*URL_SOURCE*.

Noms alternatifs

sync

Options

```
--config-dir REPERTOIRE
--no-auth-cache
--non-interactive
--quiet (-q)
--source-password ARG
--source-username ARG
--sync-password ARG
--sync-username ARG
```

Exemple

Copie les révisions en attente du dépôt source vers le dépôt miroir :

```
$ svnsync synchronize file:///var/svn/depot-miroir
Transmission des données .............................................
Révision 1 propagée.
Propriétés copiées pour la révision 1.
Révision 2 propagée.
Propriétés copiées pour la révision 2.
Révision 3 propagée.
Propriétés copiées pour la révision 3.
…
Révision 45 propagée.
Propriétés copiées pour la révision 45.
Transmission des données .
Révision 46 propagée.
Propriétés copiées pour la révision 46.
Transmission des données .
Révision 47 propagée.
Propriétés copiées pour la révision 47.
$
```

svnserve

La commande **svnserve** permet les accès aux dépôts Subversion en utilisant le protocole réseau sur mesure de Subversion.

Vous pouvez faire tourner **svnserve** en tant que serveur autonome (pour les clients qui utilisent la méthode d'accès `svn://`) ; Vous pouvez aussi avoir un démon tel que **inetd** ou **xinetd** qui le lance pour vous à la demande (encore pour `svn://`) ou vous pouvez avoir **sshd** qui le lance à la demande pour les clients qui utilisent la méthode d'accès `svn+ssh://`.

Quelle que soit la méthode d'accès, une fois que le client a choisi un dépôt en spécifiant son URL, **svnserve** lit le fichier `conf/svnserve.conf` dans le répertoire du dépôt pour déterminer les réglages spécifiques au dépôt tels que la base de données d'authentification à utiliser ou quelle politique de contrôle d'accès appliquer. Reportez vous à la section intitulée « svnserve, un serveur sur mesure » pour les détails relatifs au fichier `svnserve.conf`.

Options de svnserve

Au contraire des commandes décrites précédemment, **svnserve** ne possède pas de sous-commande, il est entièrement contrôlé par les options.

`--daemon` (`-d`)
: Lance **svnserve** en mode démon. **svnserve** passe en arrière-plan et répond aux connexions TCP/IP sur le port `svn` (3690 par défaut).

`--foreground`
: Quand elle est utilisée avec l'option `-d`, indique à **svnserve** de rester en avant-plan. Cette option est principalement utilisée à des fins de débogage.

`--inetd` (`-i`)
: Indique à **svnserve** d'utiliser l'entrée standard (`stdin`) et la sortie standard (`stdout`), comme requis pour une utilisation avec **inetd**.

`--help` (`-h`)
: Affiche un court descriptif du programme et sort.

`--listen-host=`*`HOTE`*
: Indique à **svnserve** d'écouter sur l'interface spécifiée par *`HOTE`*, qui peut être soit une adresse IP soit un nom d'hôte.

`--listen-once` (`-X`)
: Indique à **svnserve** d'accepter une connexion sur le port `svn`, d'y répondre puis de se terminer. Cette option est principalement utilisée à des fins de débogage.

`--listen-port=`*`PORT`*
: Indique à **svnserve** d'écouter le port *`PORT`* quand il fonctionne en tant que démon (les démons FreeBSD n'écoutent par défaut que sur les adresses IPv6 — Cette option indique d'écouter également sur les adresses IPv4).

`--pid-file` *`NOM_FICHIER`*
: Indique à **svnserve** d'écrire son identifiant de processus dans *`NOM_FICHIER`* ; l'utilisateur sous lequel **svnserve** tourne doit avoir le droit d'écrire dans ce fichier.

`--root=`*`RACINE`* (`-r=`*`RACINE`*)
: Définit la racine virtuelle pour les dépôts accessibles par **svnserve**. Les chemins dans les URL fournies par le client seront interprétés relativement à cette racine et le client ne pourra pas sortir de cette arborescence.

`--threads` (`-T`)
: Quand il fonctionne en mode démon, indique à **svnserve** de créer un processus léger (*thread*) plutôt qu'un nouveau processus pour chaque connexion (par exemple lors d'un fonctionnement sous Windows). Le processus **svnserve** passe toujours en arrière-plan au démarrage.

`--tunnel` (`-t`)
: Indique à **svnserve** de fonctionner en mode tunnel, qui est le même que le mode de fonctionnement que **inetd** : les deux modes répondent aux connexions sur l'entrée et la sortie standards (`stdin`/`stdout`) puis terminent, sauf que la connexion est considérée comme déjà authentifiée (l'identifiant correspond à l'UID courant). Ce drapeau est passé automatiquement pour vous par le client quand il utilise un tunnel tel que le programme **ssh**. Cela signifie que vous aurez rarement le besoin de passer cette option *vous-même* à **svnserve**. Aussi, si vous vous surprenez à taper `svnserve --tunnel` sur une ligne de commande et que vous vous demandez quoi faire par la suite, reportez vous à la section

intitulée « Encapsulation de svnserve dans un tunnel SSH ».

`--tunnel-user NOM`
Utilisée en conjonction avec l'option `--tunnel`, indique à **svnserve** que *`NOM`* est l'utilisateur authentifié, plutôt que l'UID du processus **svnserve**. Cette option est utile pour les utilisateurs qui souhaitent partager un compte unique pour SSH, mais qui veulent continuer à avoir des identités différentes pour les propagations.

`--version`
Affiche les informations de version ainsi que la liste des modules d'accès aux dépôts disponibles, puis se termine.

svndumpfilter

svndumpfilter est un utilitaire en ligne de commande pour supprimer une partie de l'historique dans un fichier dump, soit en excluant, soit en ne gardant que, des chemins commençant par une ou plusieurs racines spéficiées. Pour plus de détails, référez-vous à la section intitulée « svndumpfilter ».

svndumpfilter Options

Les options de **svndumpfilter** sont globales, de la même manière que pour les commandes **svn** et **svnadmin** :

`--drop-empty-revs`
En cas de filtrage, entraine la suppression de toutes les révisions vides (c'est-à-dire celles qui ne modifient pas le dépôt) du flux dump résultant.

`--renumber-revs`
Renumérote les révisions qui restent après le filtrage.

`--skip-missing-merge-sources`
Omet les sources fusionnées qui ont été supprimées par le filtrage. Sans cette option, **svndumpfilter** sort avec une erreur si l'origine d'une fusion a été supprimée alors que le chemin fusionné a été gardé par le filtrage.

`--preserve-revprops`
Si tous les noeuds d'une révision sont supprimés par le filtrage et que l'option `--drop-empty-revs` n'est pas spécifiée, le comportement par défaut de **svndumpfilter** consiste à supprimer toutes les propriétés de révision sauf la date et l'entrée du journal (qui indiquera simplement que la révision est vide). Spécifier cette option préserve les propriétés de révision (ce qui peut avoir du sens ou pas puisque les modifications afférentes ne figureront plus dans le flux dump).

`--quiet`
N'affiche pas les statistiques de filtrage.

svndumpfilter Subcommands

Voici les différentes sous-commandes du programme **svndumpfilter**.

Nom

svndumpfilter exclude — Exclure du flux dump résultant les chemins spécifiés.

Synopsis

```
svndumpfilter exclude PREFIXE...
```

Description

Cette commande exclut du flux dump les chemins qui commencent par le ou les *`PREFIXE`* passés en paramètres.

Options

```
--drop-empty-revs
--preserve-revprops
--quiet
--renumber-revs
--skip-missing-merge-sources
```

Exemples

Si nous avons un fichier dump issu d'un dépôt qui contient un certain nombre de répertoires concernant des piques-niques et que nous voulons garder tout *sauf* la partie relative à `sandwichs` de ce dépôt, nous allons exclure uniquement ce chemin :

```
$ svndumpfilter exclude sandwichs < fichier-dump > fichier-dump-filtré
Exclusion des préfixes :
   '/sandwichs'

Révision 0 propagée en 0
Révision 1 propagée en 1
Révision 2 propagée en 2
Révision 3 propagée en 3
Révision 4 propagée en 4

1 nœud éliminé :
   '/sandwichs'
```

Nom

svndumpfilter include — Exclure du flux dump résultant tous les chemins qui ne sont pas spécifiés.

Synopsis

```
svndumpfilter include PREFIXE...
```

Description

Cette commande ne retient dans le flux dump que les chemins qui commencent par le ou les *`PREFIXE`* passés en paramètres (excluant de fait tous les autres chemins).

Options

```
--drop-empty-revs
--preserve-revprops
--quiet
--renumber-revs
--skip-missing-merge-sources
```

Exemple

Si nous avons un fichier dump issu d'un dépôt qui contient un certain nombre de répertoires concernant des piques-niques et que nous voulons garder *uniquement* la partie relative à `sandwichs` de ce dépôt, nous allons inclure uniquement ce chemin :

```
$ svndumpfilter include sandwichs < fichier-dump > fichier-dump-filtré
Inclusion des préfixes :
   '/sandwichs'

Revision 0 propagée en 0
Revision 1 propagée en 1
Revision 2 propagée en 2
Revision 3 propagée en 3
Revision 4 propagée en 4

3 nœuds éliminés :
   '/boissons'
   '/encas'
   '/fournitures'
```

Nom

svndumpfilter help — À l'aide !

Synopsis

```
svndumpfilter help [SOUS-COMMANDE...]
```

Description

Affiche le message d'aide pour **svndumpfilter**. Contrairement aux autres commandes d'aide documentées dans ce chapitre, il n'y a pas de commentaire spirituel pour cette commande. Les auteurs de ce livre sont vraiment désolés pour cette grave lacune.

Options

Aucun.

svnversion

Nom

svnversion — Produire un numéro de révision compact pour la copie de travail.

Synopsis

```
svnversion [OPTIONS] [CT_CHEMIN [FIN_URL]]
```

Description

svnversion est un programme qui produit un numéro de révision (ou un intervalle) à partir des révisions utilisées et des modifications effectuées sur la copie de travail. Le numéro de révision, ou l'intervalle, produit est écrit sur la sortie standard.

Il est d'usage d'utiliser ce résultat dans la chaîne de compilation pour définir le numéro de version d'un programme.

`FIN_URL`, si elle est spécifiée, est la partie de la fin de l'URL à utiliser pour déterminer si *`CT_CHEMIN`* a été ré-aiguillé (normalement, la détection de ré-aiguillage pour *`CT_CHEMIN`* ne prend pas en compte *`FIN_URL`*).

Quand *`CT_CHEMIN`* n'est pas spécifié, le répertoire courant est utilisé. *`FIN_URL`* ne peut pas être spécifiée si *`CT_CHEMIN`* ne l'est pas.

Options

De même que pour **svnserve**, **svnversion** n'a pas de sous-commande, seulement des options :

`--no-newline` (`-n`)
: Pas de fin de ligne habituel en fin d'affichage.

`--committed` (`-c`)
: Utilise la dernière révision propagée plutôt que les révisions courantes (c'est-à-dire les plus récentes disponibles localement).

`--help` (`-h`)
: Affiche l'aide du programme.

`--version`
: Affiche la version de **svnversion** et termine sans erreur.

Exemples

Si la copie de travail est issue d'une unique révision (par exemple, immédiatement après une mise à jour par **svn update**), alors le numéro de révision correspondant est affiché :

```
$ svnversion
4168
```

Vous pouvez ajouter *`FIN_URL`* pour être sûr que la copie de travail n'a pas été ré-aiguillée ailleurs que ce que vous pensez. Notez que *`CT_CHEMIN`* est indispensable dans ce cas :

```
$ svnversion . /var/svn/trunk
4168
```

Pour une copie de travail qui utilise des révisions multiples, l'intervalle des révisions utilisées localement est affiché :

```
$ svnversion
4123:4168
```

Si la copie de travail contient des modifications locales, un "M" est ajouté à la fin :

```
$ svnversion
4168M
```

Si la copie de travail a été re-aiguillée (avec la sous-commande **svn switch**), un "S" (pour *switched*) est ajouté à la fin :

```
$ svnversion
4168S
```

Ainsi, voici le résultat pour une copie de travail issues de plusieurs révisions, re-aiguillée et qui contient des modifications locales :

```
$ svnversion
4212:4168MS
```

Si la commande **svnversion** est invoquée sur un répertoire qui n'est pas une copie de travail, elle considère que c'est un export d'une copie de travail et affiche "exporté" :

```
$ svnversion
exporté
```

mod_dav_svn

Nom

Directives de configuration de mod_dav_svn — Directives de configuration Apache pour les dépôts Subversion accessibles via un serveur HTTP Apache.

Description

Cette section décrit brièvement chaque directive de configuration Apache pour Subversion. Pour une description plus approfondie, veuillez vous référer à la section intitulée « httpd, le serveur HTTP Apache ».)

Directives

Voici les directives du fichier `httpd.conf` qui s'appliquent au module **mod_dav_svn** :

`DAV svn`
: Doit être incluse dans chaque bloc `Directory` ou `Location` pour un dépôt Subversion. Elle indique au serveur **httpd** d'utiliser le dorsal `mod_dav` pour gérer les requêtes.

`SVNAllowBulkUpdates On|Off`
: Bascule le support pour les réponses complètes aux requêtes de mises à jour de type `REPORT`. Les clients Subversion utilisent les requêtes `REPORT` pour obtenir des informations relatives aux extractions et mises à jour de la part de **mod_dav_svn**. Ils peuvent demander au serveur d'envoyer cette information sous deux formes : soit la totalité des informations concernant l'arborescence dans une seule réponse, soit un *skelta* (une représentation squelettique des modifications de l'arborescence) qui contient les informations juste nécessaires au client pour déterminer quelles données *supplémentaires* demander au serveur. Quand cette directive est incluse avec la valeur `Off`, **mod_dav_svn** ne répond aux requêtes `REPORT` qu'avec des réponses skelta, quelle que soit la forme demandée par le client.

  La plupart des lecteurs n'aura jamais à utiliser cette directive. Elle existe uniquement pour les administrateurs qui désirent (pour des raisons de sécurité ou d'audit) forcer les clients à retirer individuellement les fichiers et répertoires nécessaires aux extractions et mises à jour, et ainsi laisser des traces de toutes les requêtes `GET` et `PROPFIND` dans les journaux d'Apache. La valeur par défaut de cette directive est `On`.

`SVNAutoversioning On|Off`
: Quand la valeur est `On`, elle autorise les requêtes en écriture de la part des clients WEBDAV pour générer des propagations automatiques. Une entrée de journal générique est créée automatiquement et associée à chaque révision. Si vous autorisez la gestion de versions automatique, vous voudrez sûrement spécifier `ModMimeUsePathInfo On` de manière à ce que `mod_mime` puisse définir `svn:mime-type` au type MIME correct automatiquement (pour autant que `mod_mime` sache le faire, bien évidemment). Pour plus d'informations, voir l'Annexe C, *WebDAV et la gestion de versions automatique*. La valeur par défaut de cette directive est `Off`.

`SVNPath` *`chemin-vers-repertoire`*
: Spécifie le chemin de la racine des fichiers du dépôt Subversion dans le système de fichiers. Dans un bloc de configuration d'un dépôt Subversion, soit cette directive est présente, soit `SVNParentPath` doit l'être, mais pas les deux.

`SVNSpecialURI` *`composant`*
: Spécifie l'URI (espace de noms) pour les ressources spéciales de Subversion. La valeur par défaut est `!svn`, et la majorité des administrateurs n'utilisera jamais cette directive. Ne la définissez que si vous avez un besoin impérieux d'avoir un fichier dont le nom est `!svn` dans votre dépôt. Si vous changez cette valeur dans un serveur déjà en fonction, cela endommagera toutes les copies de travail déjà diffusées. Vos utilisateurs vous poursuivront alors avec des fourchettes et des torches.

`SVNReposName` *`nom`*
: Spécifie le nom du dépôt Subversion à utiliser dans les réponses aux requêtes `HTTP GET`. Cette valeur est ajoutée en tête du titre de chaque listing de répertoire (qui sont affichés quand vous naviguez dans un dépôt Subversion avec un navigateur Web). Cette directive est optionnelle.

`SVNIndexXSLT` *`chemin-vers-repertoire`*
: Spécifie l'URI d'une transformation XSL pour les index de répertoires. Cette directive est optionnelle.

`SVNParentPath` *`chemin-vers-repertoire`*
: Spécifie le chemin dans le système de fichiers du répertoire racine dont les sous-répertoires sont des dépôts Subversion.

Dans un bloc de configuration d'un dépôt Subversion, soit cette directive est présente, soit `SVNPath` l'est, mais pas les deux.

`SVNPathAuthz On|Off|short_circuit`
Définit le contrôle d'accès basé sur les chemins en autorisant les sous-requêtes (`On`) ou en les interdisant (`Off` ; Voir la section intitulée « Désactivation du contrôle sur les chemins ») ou en interrogeant directement **mod_authz_svn** (`short_circuit`). La valeur par défaut de cette directive est `On`.

`SVNListParentPath On|Off`
Quand cette directive est définie à `On`, elle autorise les requêtes `GET` sur `SVNParentPath`, ce qui produit l'affichage du listing de tous les dépôts sous le chemin considéré. La valeur par défaut est `Off`.

`SVNMasterURI` *`url`*
Spécifie l'URI d'un dépôt maître Subversion (utilisée dans le cadre d'un serveur mandataire en écriture).

`SVNActivitiesDB` *`chemin-vers-repertoire`*
Spécifie le chemin dans le système de fichiers où les activités de la base de données doivent être stockées. Par défaut, **mod_dav_svn** crée et utilise un répertoire dans le dépôt appelé `dav/activities.d`. Le chemin spécifié par cette directive doit être un chemin absolu.

Si un chemin est spécifié à l'aide de `SVNParentPath`, **mod_dav_svn** ajoute le nom de fichier (basename) du dépôt au chemin fourni dans cette directive. Par exemple :

```
<Location /svn>
  DAV svn

  # toute URL de type "/svn/truc" est transformée en
  # /net/svn.nfs/depots/truc
  SVNParentPath         "/net/svn.nfs/depots"

  # toute URL de type "/svn/machin" est transformée en base de données
  # activités dans /var/db/svn/activites/truc
  SVNActivitiesDB       "/var/db/svn/activites"
</Location>
```

Journalisation de haut niveau

Voici la liste des messages de haut niveau produits par le mécanisme de journalisation Apache pour les actions Subversion. Un exemple est fourni pour chaque type de message. Reportez-vous à la section intitulée « Journalisation Apache » pour les détails relatifs à la journalisation.

Extraction ou export
```
checkout-or-export /chemin r62 depth=infinity
```

Propagation
```
commit harry r100
```

Différences
```
diff /chemin r15:20 depth=infinity ignore-ancestry

diff /chemin1@15 /chemin2@20 depth=infinity ignore-ancestry
```

Parcours d'un répertoire
```
get-dir /trunk r17 text
```

Parcours d'un fichier
```
get-file /chemin r20 props
```

Parcours des révisions d'un fichier
```
get-file-revs /chemin r12:15 include-merged-revisions
```

Parcours des informations de fusion
```
get-mergeinfo (/chemin1 /chemin2)
```

Verrouillage
```
lock /chemin steal
```

Entrées du journal
```
log (/chemin1,/chemin2,/chemin3) r20:90 discover-changed-paths revprops=()
```

Rejeu de révisions (svnsync)
```
replay /chemin r19
```

Changements de propriétés de révisions
```
change-rev-prop r50 nom-de-propriete
```

Liste des propriétés de révision
```
rev-proplist r34
```

Status
```
status /chemin r62 depth=infinity
```

Re-aiguillage
```
switch /cheminA /cheminB@50 depth=infinity
```

Déverrouillage
```
unlock /chemin break
```

Mise à jour
```
update /chemin r17 send-copyfrom-args
```

mod_authz_svn

Nom

Directives de configuration de mod_authz_svn — Directives de configuration Apache pour configurer les contrôles sur les chemins dans les dépôts Subversion accessibles via un serveur HTTP Apache.

Description

Cette section décrit brièvement chaque directive de configuration Apache gérée par le module **mod_authz_svn**. Pour une description approfondie de l'utilisation des contrôles sur les chemins dans Subversion, reportez-vous à la section intitulée « Contrôle d'accès basé sur les chemins »).

Directives

Voici les directives du fichier `httpd.conf` qui s'appliquent au module **mod_authz_svn** :

`AuthzSVNAccessFile` *`chemin/vers/fichier`*
: Indique à Apache de consulter le fichier *`chemin/vers/fichier`* pour obtenir les règles décrivant les droits d'accès des chemins du dépôt Subversion.

`AuthzSVNAnonymous On|Off`
: Mettez `Off` pour interdire deux comportements particuliers de ce module : l'interaction avec la directive `Satisfy Any` et l'application d'une politique d'autorisation par défaut quand aucune directive `Require` n'est présente. La valeur par défaut de cette directive est `On`.

`AuthzSVNAuthoritative On|Off`
: Mettez `Off` pour déléguer le contrôle d'accès à des modules de plus bas niveau. La valeur par défaut de cette directive est `On`.

`AuthzSVNNoAuthWhenAnonymousAllowed On|Off`
: Mettez `On` pour supprimer les demandes d'authentification et les droits d'accès pour les requêtes autorisées aux utilisateurs anonymes. La valeur par défaut de cette directive est `On`.

Propriétés dans Subversion

Avec Subversion, les utilisateurs peuvent définir des propriétés au nom arbitraire, gérées en versions, sur les fichiers et les répertoires, ainsi que des propriétés non gérées en versions sur les révisions. La seule restriction concerne les propriétés dont le nom commence par `svn:` (elles sont réservées pour l'usage propre de Subversion). Bien que ces propriétés puissent être définies par les utilisateurs pour modifier le comportement de Subversion, ceux-ci ne devraient pas définir de nouvelles propriétés `svn:`.

Propriétés gérées en versions

Voici les propriétés gérées en versions utilisées par Subversion pour son usage propre :

`svn:executable`
: Si elle est définie sur un fichier, le client active le bit d'exécution du fichier dans les copies de travail sur les machines de type Unix. Voir la section intitulée « Fichiers exécutables ou non ».

`svn:mime-type`
: Si elle est définie sur un fichier, sa valeur indique le type MIME du fichier. Ceci permet au client de décider si la fusion contextuelle basée sur les lignes est pertinente au cours des mises à jour ; cela peut aussi affecter l'affichage du fichier par un navigateur Web. Voir la section intitulée « Type de contenu des fichiers ».

`svn:ignore`
: Si elle est définie sur un répertoire, sa valeur est la liste des motifs de fichiers *non gérés en versions* qui doivent être ignorés par les commandes **svn status** et les autres sous-commandes. Voir la section intitulée « Occultation des éléments non suivis en versions ».

`svn:keywords`
Si elle est définie sur un fichier, sa valeur indique au client comment substituer certains mots-clés à l'intérieur du fichier. Voir la section intitulée « Substitution de mots-clés ».

`svn:eol-style`
Si elle est définie sur un fichier, sa valeur indique au client quelle convention adopter pour les marqueurs de fin de ligne dans la copie de travail et dans les exports d'arborescences. Voir la section intitulée « Caractères de fin de ligne » et svn export plus en amont dans ce chapitre.

`svn:externals`
Si elle est définie sur un répertoire, sa valeur est une liste, chaque élément sur une ligne, de chemins et d'URL à extraire par le client. Voir la section intitulée « Définition de références externes ».

`svn:special`
Si elle est définie sur un fichier, elle indique que le fichier n'est pas un fichier ordinaire mais un lien symbolique ou un autre objet spécial[1].

`svn:needs-lock`
Si elle est définie sur un fichier, indique au client de marquer ce fichier en lecture seule dans la copie de travail, de façon à ce que cela rappelle que ce fichier devrait être vérouillé avant édition. Voir la section intitulée « Communication par l'intermédiaire des verrous ».

`svn:mergeinfo`
Cette propriété est utilisée par Subversion pour suivre les données fusionnées. Voir la section intitulée « Mergeinfo et aperçus » pour plus de détails. Vous ne devriez jamais modifier cette propriété à moins de savoir *vraiment* ce que vous faîtes.

Propriétés non gérées en versions

Voici les propriétés non gérées en versions que Subversion réserve à son usage propre :

`svn:author`
Si elle est définie, la valeur est l'identifiant (authentifié) de l'utilisateur qui a créé la révision. Si elle n'est pas définie, la révision a été propagée de manière anonyme.

`svn:date`
La valeur de cette propriété est la date et l'heure UTC de création de la révision, au format ISO 8601. La valeur est issue de l'horloge du *serveur*, pas de celle du client.

`svn:log`
La valeur est l'entrée du journal pour la révision.

`svn:autoversioned`
Si cette propriété est définie, la révision a été créée par un mécanisme de gestion de versions automatique. Voir la section intitulée « Gestion de versions automatique ».

Procédures automatiques du dépôt

Subversion met à votre disposition les procédures automatiques suivantes sur le dépôt :

[1] Au moment de l'écriture de ce livre, les liens symboliques sont en fait les seuls objets « spéciaux ». Mais il est envisageable qu'il y ait d'autres objets dans les futures versions de Subversion.

Nom

start-commit — Notification du début d'une propagation.

Description

La procédure automatique `start-commit` est activée avant que la transaction de propagation ne soit créée. Typiquement, elle est utilisée pour décider si l'utilisateur possède les droits de propager une nouvelle révision.

Si le code de retour de la procédure automatique `start-commit` est non nul, la propagation est arrêtée avant même la création de la transaction de propagation et tout ce qui a été écrit vers `stderr` est renvoyé vers le client.

Paramètres en entrée

Les arguments de la ligne de commande passés à la procédure automatique sont, dans l'ordre :

1. Chemin du dépôt ;
2. identifiant (authentifié) de l'utilisateur qui initie la propagation ;
3. liste, dont les éléments sont séparés par des virgules, de capacités que le client passe au serveur, dont `depth`, `mergeinfo` et `log-revprops` (nouveau dans Subversion 1.5).

Utilisations principales

Contrôle d'accès (par exemple, interdiction temporaire d'effectuer des propagations pour telle ou telle raison).

Un moyen de n'autoriser l'accès qu'à des clients qui possèdent certaines capacités.

Nom

pre-commit — Notification juste avant la fin de la propagation.

Description

La procédure automatique `pre-commit` est activée juste avant que la transaction de propagation ne génère une nouvelle révision. Cette procédure automatique est typiquement utilisée pour protéger le dépôt vis-à-vis de propagations qui ne respectent pas certaines règles relatives au contenu ou au chemin (par exemple, votre dépôt peut imposer que les propagations sur une certaine branche incluent un numéro de ticket de l'outil de gestion de suivi des bogues ou alors que l'entrée du journal de propagation ne soit pas vide).

Si le code de retour de la procédure automatique `pre-commit` est non nul, la propagation est annulée, la transaction de propagation supprimée et tout ce qui a été écrit vers `stderr` est renvoyé vers le client.

Paramètres en entrée

Les arguments de la ligne de commande passés à la procédure automatique sont, dans l'ordre :

1. Chemin du dépôt ;
2. Nom de la transaction de propagation.

Utilisation principale

Contrôle et validation des modifications.

Nom

post-commit — Notification d'une propagation réussie.

Description

La procédure automatique `post-commit` est activée après que la transaction ait été validée et que la nouvelle révision soit créée. La plupart des administrateurs utilisent cette procédure automatique pour envoyer des e-mails décrivant la propagation ou pour notifier à d'autres outils (tels que les outils de gestion pour le suivi de bogues) qu'une nouvelle propagation a eu lieu. Certains utilisent cette procédure automatique pour déclencher une sauvegarde.

Si le code de retour de la procédure automatique `post-commit` est non nul, la propagation *a bien lieu* puisqu'elle est déjà terminée. Cependant, tout ce qui a été écrit vers `stderr` est renvoyé vers le client, afin de trouver plus facilement la raison de l'échec de la procédure automatique.

Paramètres en entrée

Les arguments de la ligne de commande passés à la procédure automatique sont, dans l'ordre :

1. Chemin du dépôt ;
2. Numéro de révision créée par la propagation.

Utilisations principales

Notification de propagation ; intégration avec d'autres outils.

Nom

pre-revprop-change — Notification avant le changement d'une propriété de révision.

Description

La procédure automatique `pre-revprop-change` est activée juste avant la modification d'une propriété de révision quand elle n'a pas lieu dans le cadre d'une propagation normale. Contrairement aux autres procédures automatiques, la configuration par défaut de cette procédure automatique est d'interdire l'action demandée. La procédure automatique doit exister effectivement et le code de retour doit être nul pour autoriser la modification d'une propriété de révision.

Si la procédure automatique `pre-revprop-change` n'existe pas, ne possède pas les droits d'exécution ou si le code de retour n'est pas nul, la propriété n'est pas modifiée et tout ce qui a été écrit vers `stderr` est renvoyé vers le client.

Paramètres en entrée

Les arguments de la ligne de commande passés à la procédure automatique sont, dans l'ordre :

1. Chemin du dépôt ;
2. Révision dont une propriété va être modifiée ;
3. identifiant (authentifié) de l'utilisateur qui demande la modification de la propriété ;
4. Nom de la propriété à modifier ;
5. Description de la modification : `A` (ajout), `D` (suppression) ou `M` (modification).

Par ailleurs, Subversion fournit la nouvelle valeur de la propriété à la procédure automatique *via* l'entrée standard.

Utilisations principales

Contrôle d'accès ; Contrôle et validation des modifications.

Nom

post-revprop-change — Notification d'une modification de propriété de révision réussie.

Description

La procédure automatique `post-revprop-change` est activée immédiatement après la modification d'une propriété de révision, quand elle n'a pas lieu dans le cadre d'une propagation normale. Comme vous pouvez le deviner à la lecture de la description de son homologue, la procédure automatique `pre-revprop-change`, cette procédure automatique ne peut être activée que si `pre-revprop-change` est implémentée. Cette procédure automatique est principalement utilisée pour envoyer des e-mails de notification qu'une propriété a été modifiée.

Si le code de retour de la procédure automatique `post-revprop-change` est non nul, la modification *a bien lieu* puisqu'elle est déjà terminée. Cependant, tout ce qui a été écrit vers `stderr` est renvoyé vers le client, afin de trouver plus facilement la raison de l'échec de la procédure automatique.

Paramètres en entrée

Les arguments de la ligne de commande passés à la procédure automatique sont, dans l'ordre :

1. Chemin du dépôt ;
2. Révision dont une propriété a été modifiée ;
3. identifiant (authentifié) de l'utilisateur qui a effectué la modification de la propriété ;
4. Nom de la propriété modifiée ;
5. Description de la modification : `A` (ajout), `D` (suppression) ou `M` (modification).

Par ailleurs, Subversion fournit la valeur précédente de la propriété à la procédure automatique *via* l'entrée standard.

Utilisation principale

Notification de la modification d'une propriété.

Nom

pre-lock — Notification d'une demande de verrouillage d'un chemin.

Description

La procédure automatique `pre-lock` est activée lorsque quelqu'un demande à verrouiller un chemin. Elle peut être utilisée pour empêcher tout verrouillage ou pour définir une politique plus complexe où tels utilisateurs sont autorisés à verrouiller tels chemins. Si la procédure automatique détecte un verrou pré-existant, elle peut aussi décider si l'utilisateur est autorisé à « voler » ce verrou pré-existant.

Si le code de retour de la procédure automatique `pre-lock` est non nul, le verrouillage est annulé et tout ce qui a été écrit vers `stderr` est renvoyé vers le client.

Paramètres en entrée

Les arguments de la ligne de commande passés à la procédure automatique sont, dans l'ordre :

1. Chemin du dépôt ;
2. Chemin géré en versions qui va être verrouillé ;
3. identifiant (authentifié) de l'utilisateur qui demande le verrouillage.

Utilisation principale

Contrôle d'accès.

Nom

post-lock — Notification d'un verrouillage de chemin réussi.

Description

La procédure automatique `post-lock` est activée après qu'un ou plusieurs chemins aient été verrouillés. Elle est utilisée typiquement pour envoyer des e-mails de notification signalant ce verrouillage.

Si le code de retour de la procédure automatique `post-lock` est non nul, le verrouillage *a bien lieu* puisqu'il est déjà effectif. Cependant, tout ce qui a été écrit vers `stderr` est renvoyé vers le client, afin de trouver plus facilement la raison de l'échec de la procédure automatique.

Paramètres en entrée

Les arguments de la ligne de commande passés à la procédure automatique sont, dans l'ordre :

1. Chemin du dépôt ;
2. identifiant (authentifié) de l'utilisateur qui a effectué le verrouillage.

Par ailleurs, la liste des chemins verrouillés est fournie à la procédure automatique *via* l'entrée standard, un chemin par ligne.

Utilisation principale

Notification de verrouillage.

Nom

pre-unlock — Notification d'une demande de déverrouillage d'un chemin.

Description

La procédure automatique `pre-unlock` est activée lorsque quelqu'un demande à supprimer un verrou sur un fichier. Elle peut être utilisée pour définir des politiques qui spécifient que tels utilisateurs sont autorisés à déverrouiller tels chemins. Ceci est particulièrement important dans le cadre de votre politique globale de verrouillage. Si un utilisateur A verrouille un fichier, est-ce que l'utilisateur B est autorisé à casser ce verrou ? Qu'en est-il si le verrou est âgé de plus d'une semaine ? Ce type de politique peut être implémenté par la procédure automatique.

Si le code de retour de la procédure automatique `pre-unlock` est non nul, le déverrouillage n'a pas lieu et tout ce qui a été écrit vers `stderr` est renvoyé vers le client.

Paramètres en entrée

Les arguments de la ligne de commande passés à la procédure automatique sont, dans l'ordre :

1. Chemin du dépôt ;
2. Chemin géré en version qui va être déverrouillé ;
3. identifiant (authentifié) de l'utilisateur qui demande le déverrouillage.

Utilisation principale

Contrôle d'accès.

Nom

post-unlock — Notification d'un déverrouillage de chemin réussi.

Description

La procédure automatique `post-unlock` est activée quand un ou plusieurs chemins ont été déverrouillés. Elle est utilisée typiquement pour envoyer des e-mails notifiant le déverrouillage.

Si le code de retour de la procédure automatique `post-unlock` est non nul, le déverrouillage *a bien lieu* puisqu'il est déjà effectif. Cependant, tout ce qui a été écrit vers `stderr` est renvoyé vers le client, afin de trouver plus facilement la raison de l'échec de la procédure automatique.

Paramètres en entrée

Les arguments de la ligne de commande passés à la procédure automatique sont, dans l'ordre :

1. Chemin du dépôt ;
2. identifiant (authentifié) de l'utilisateur qui a effectué le déverrouillage.

Par ailleurs, la liste des chemins déverrouillés est passée à la procédure automatique *via* l'entrée standard, un chemin par ligne.

Utilisation principale

Notification de déverrouillage.

Annexe A. Guide de démarrage rapide avec Subversion

Si vous êtes pressé d'installer et d'utiliser Subversion (et que vous aimez apprendre en expérimentant), cette annexe vous indiquera comment créer un dépôt, importer votre code puis comment en extraire une copie de travail. Tout au long de l'exposé, nous vous fournirons des liens vers les chapitres correspondants de ce livre.

Si vous débutez avec les concepts de suivi de versions ou avec le modèle « Copier-Modifier-Fusionner » utilisé par CVS et Subversion, nous vous conseillons de lire le Chapitre 1, *Notions fondamentales* avant d'aller plus loin.

Installer Subversion

Subversion est construit sur une couche de portabilité appelée APR (« Apache Portable Runtime » en anglais, pour bibliothèque Apache de portabilité des exécutables). Cette bibliothèque APR fournit toutes les interfaces dont Subversion a besoin pour fonctionner sur différents systèmes d'exploitation : accès aux disques, au réseau, gestion de la mémoire, et bien d'autres choses encore. Bien que Subversion soit capable d'utiliser Apache comme serveur réseau, la dépendance envers APR *ne signifie pas* qu'Apache soit un composant indispensable. APR est une bibliothèque autonome, utilisable par n'importe quelle application. Cela signifie en revanche que, tout comme Apache, les clients et serveurs Subversion fonctionnent sur n'importe quel système d'exploitation sur lequel fonctionne le serveur Apache **httpd** : Windows, Linux, tous les systèmes BSD, Mac OS X, NetWare entre autres.

La manière la plus simple d'obtenir Subversion est de télécharger un programme précompilé pour votre système d'exploitation. Le site Web de Subversion (http://subversion.tigris.org) tient à disposition de nombreux paquets produits par des volontaires. Le site contient généralement des exécutables avec une interface graphique d'installation pour les utilisateurs de systèmes Microsoft. Si votre système d'exploitation est de type Unix, vous pouvez utiliser le gestionnaire de paquets fourni avec votre distribution (RPM, DEB, l'arbre des ports, etc.) pour obtenir Subversion.

Sinon, vous pouvez aussi compiler Subversion directement à partir des sources, bien que ce ne soit pas toujours facile (si vous n'avez pas l'habitude de compiler vos logiciels, choisissez plutôt de télécharger un paquet précompilé pour votre distribution). Sur le site Web de Subversion, téléchargez la dernière version du code source. Puis, après l'avoir décompacté, suivez les instructions fournies dans le fichier `INSTALL` pour la compilation. Notez que le fichier contenant le code source n'inclut pas forcément tout ce dont vous avez besoin pour construire un client en ligne de commande apte à communiquer avec un dépôt distant. Depuis Subversion 1.4, les bibliothèques dont dépend Subversion (apr, apr-util et neon) sont distribuées dans un paquet source distinct suffixé par `-deps`. Ces bibliothèques sont maintenant tellement courantes qu'elles sont peut-être déjà installées sur votre système. Sinon, vous devrez décompacter le paquet des dépendances à l'endroit où vous avez décompacté le code source de Subversion. Indépendamment de ces paquets obligatoires, vous voudrez peut-être également installer d'autres bibliothèques optionnelles telles que Berkeley DB et Apache **httpd**. Si vous voulez effectuer une compilation complète, assurez-vous bien d'avoir l'ensemble des paquets documentés dans le fichier `INSTALL`.

Si vous êtes de ceux qui aiment avoir la toute dernière version des logiciels, vous pouvez aussi obtenir le code source de Subversion depuis le dépôt Subversion. Évidemment, il faudra pour y parvenir que vous disposiez déjà d'un client Subversion… Mais, si c'est le cas, vous pouvez extraire une copie de travail du dépôt contenant le code source de Subversion à l'adresse http://svn.collab.net/repos/svn/trunk/ : [1]

```
$ svn checkout http://svn.collab.net/repos/svn/trunk subversion
A    subversion/HACKING
A    subversion/INSTALL
A    subversion/README
A    subversion/autogen.sh
A    subversion/build.conf
…
```

La commande précédente crée une copie de travail de la dernière version (non officielle) du code source de Subversion dans un

[1] Notez que l'URL que l'on extrait dans cet exemple ne se termine pas par `svn`, mais par un sous-répertoire nommé `trunk`. Reportez-vous à notre discussion sur le modèle de gestion des branches de Subversion pour en comprendre la raison.

sous-répertoire appelé `subversion` de votre répertoire de travail courant. Vous pouvez modifier le dernier argument à votre convenance. Indépendamment du nom que vous donnez au répertoire contenant la nouvelle copie de travail, une fois cette opération terminée, vous aurez à votre disposition le code source de Subversion. Bien sûr, il vous faudra encore récupérer quelques autres bibliothèques (apr, apr-util, etc.)—consultez le fichier `INSTALL` dans le répertoire racine de la copie de travail pour plus de détails.

Tutoriel rapide

> « Vérifiez que le dossier de votre siège est relevé, que votre ceinture est correctement bouclée et que la tablette devant vous est rangée et verrouillée. Personnel de cabine, attention au décollage… »

Ce qui suit est un bref tutoriel qui couvre quelques opérations élémentaires, ainsi que la configuration de base de Subversion. Une fois que vous l'aurez terminé, vous devriez avoir une compréhension globale de la façon dont Subversion peut être utilisé.

Les exemples utilisés dans cette annexe supposent que vous disposez de **svn** (le client en ligne de commande de Subversion) et de **svnadmin** (l'outil d'administration) prêts à l'emploi sur un système de type Unix (ce tutoriel fonctionne également en ligne de commande sous Windows, sous réserve de quelques adaptations triviales). Nous supposons également que vous utilisez la version 1.2 ou ultérieure de Subversion (tapez **`svn --version`** pour vous en assurer).

Subversion stocke toutes les données suivies en version dans un dépôt central. Pour commencer, créez un nouveau dépôt :

```
$ svnadmin create /var/svn/depot
$ ls /var/svn/depot
conf/  dav/  db/  format  hooks/  locks/  README.txt
```

Cette commande crée un nouveau répertoire, `/var/svn/depot`, qui contient un dépôt Subversion. Ce nouveau répertoire contient (entre autres choses) un ensemble de fichiers constituant une base de données. Vous ne verrez pas vos fichiers suivis en version si vous examinez le contenu de ces fichiers. Pour plus d'informations sur la création et la maintenance des dépôts, consultez le Chapitre 5, *Administration d'un dépôt*.

Dans Subversion, il n'existe pas de concept de « projet ». Le dépôt est juste un système de fichiers virtuel suivi en versions, une arborescence qui peut contenir tout ce que vous voulez. Certains administrateurs préfèrent ne stocker qu'un seul projet par dépôt, d'autres préfèrent stocker plusieurs projets par dépôt en les plaçant dans des répertoires distincts. Les mérites de chacune de ces approches sont discutés dans la section intitulée « Stratégies d'organisation d'un dépôt ». De toute façon, le dépôt ne fait que gérer des fichiers et des répertoires, c'est donc aux humains de faire le lien entre répertoires et « projets ». Ainsi, bien que vous trouverez mention de projets dans ce livre, gardez en mémoire que nous ne parlons jamais que d'un répertoire (ou d'un ensemble de répertoires) du dépôt.

Dans cet exemple, nous supposons que vous avez déjà une sorte de projet (c'est-à-dire un ensemble de fichiers et de répertoires) que vous voulez importer dans votre dépôt Subversion tout neuf. Commencez par organiser vos données dans un répertoire unique appelé `monprojet` (ou quoi que ce soit d'autre). Pour des raisons que nous expliquons au Chapitre 4, *Gestion des branches*, la structure de votre arborescence doit contenir trois répertoires à la racine : `branches`, `tags`, et `trunk`. Le répertoire `trunk` doit contenir toutes vos données et les répertoires `branches` et `tags` doivent être vides :

```
/tmp/monprojet/branches/
/tmp/monprojet/tags/
/tmp/monprojet/trunk/
                    Makefile
                    machin.c
                    truc.c
                    …
```

Les sous-répertoires `branches`, `tags` et `trunk` ne sont pas réellement requis par Subversion. Ils font simplement partie des conventions d'utilisation que vous voudrez certainement suivre par la suite.

Une fois l'arborescence de vos données prête, importez-la dans le dépôt avec la commande **svn import** (reportez-vous à la

section intitulée « Enregistrement de données dans votre dépôt ») :

```
$ svn import /tmp/monprojet file:///var/svn/depot/monprojet \
             -m "Import initial"
Ajout          /tmp/monprojet/branches
Ajout          /tmp/monprojet/tags
Ajout          /tmp/monprojet/trunk
Ajout          /tmp/monprojet/trunk/Makefile
Ajout          /tmp/monprojet/trunk/truc.c
Ajout          /tmp/monprojet/trunk/machin.c
…
Révision 1 propagée.
$
```

À présent, le dépôt contient cette arborescence de données. Comme indiqué précédemment, vous ne verrez pas vos fichiers directement en regardant dans le dépôt : ils sont stockés dans un magasin de données. Mais le système de fichiers imaginaire du dépôt contient désormais un répertoire racine appelé `monprojet`, qui à son tour contient vos données.

Notez que le répertoire original `/tmp/monprojet` n'a pas été modifié ; Subversion ignore tout de son existence (en fait, vous pouvez même le supprimer si vous voulez). Pour commencer à manipuler les données du dépôt, vous devez créer une nouvelle « copie de travail » des données, une sorte d'espace de travail privé. Demandez à Subversion de vous « extraire » une copie de travail du répertoire `monprojet/trunk` du dépôt :

```
$ svn checkout file:///var/svn/depot/monprojet/trunk monprojet
A  monprojet/Makefile
A  monprojet/machin.c
A  monprojet/truc.c
…
Révision 1 extraite.
```

À présent, vous disposez d'une copie personnelle d'une partie du dépôt, située dans un nouveau répertoire appelé `monprojet`. Vous pouvez éditer les fichiers dans votre copie de travail puis propager ces changements vers le dépôt.

- Entrez dans le répertoire de votre copie de travail et éditez le contenu d'un fichier ;
- lancez la commande **`svn diff`** pour obtenir la liste des différences que vos modifications ont engendrée ;
- lancez la commande **`svn commit`** pour propager la nouvelle version de votre fichier vers le dépôt ;
- lancez la commande **`svn update`** pour « mettre à jour » votre copie de travail à partir du dépôt.

Pour une description complète de ce que vous pouvez faire avec votre copie de travail, reportez-vous au Chapitre 2, *Utilisation de base*.

Dès lors, vous pouvez mettre votre dépôt à disposition sur le réseau. Consultez le Chapitre 6, *Configuration du serveur* pour découvrir les différents serveurs disponibles et la manière de les configurer.

Annexe B. Guide Subversion à l'usage des utilisateurs de CVS

Cette annexe est un guide pour les utilisateurs de CVS qui découvrent Subversion. Il est essentiellement constitué d'une liste de différences, « au doigt mouillé », entre les deux systèmes. Pour chaque section, nous fournissons autant que possible les références des chapitres pertinents.

Bien que le but de Subversion soit de s'emparer de la communauté des utilisateurs actuels et futurs de CVS, de nouvelles fonctionnalités et des changements conceptuels étaient nécessaires pour corriger certains comportements « malencontreux » de CVS. Cela signifie que, en tant qu'utilisateur de CVS, vous devrez vous défaire de certaines habitudes — celles dont vous avez oublié combien elles vous semblaient bizarres au début.

Les numéros de révisions sont différents

Dans CVS, les numéros de révisions sont associés à un fichier. Ceci est du au fait que CVS stocke ses données dans des fichiers RCS ; à chaque fichier est associé un fichier RCS dans le dépôt et la structure du dépôt correspond plus ou moins à l'arborescence de votre projet.

Dans Subversion, le dépôt ressemble à un système de fichiers unique. Chaque propagation conduit à un système de fichiers entièrement nouveau ; au fond, le dépôt est un tableau d'arborescences de fichiers. Chacune de ces arborescences est étiquetée avec un numéro de révision. Quand quelqu'un parle de la « révision 54 », il parle d'une arborescence particulière (et indirectement, de l'état du système de fichiers après la cinquante-quatrième propagation).

Techniquement, il n'est pas correct de parler de « la révision 5 du fichier `machin.c` ». À la place, on devrait dire « `machin.c` tel qu'il était en révision 5 ». Soyez également prudent quand vous faites des suppositions sur les évolutions d'un fichier. Dans CVS, les révisions 5 et 6 de `machin.c` sont toujours différentes. Dans Subversion, le plus probable est que `machin.c` *n'a pas changé* entre les révisions 5 et 6.

De la même manière, dans CVS, une étiquette et une branche sont des annotations sur un fichier ou sur l'information de version de ce fichier. En revanche, dans Subversion, une branche ou une étiquette sont des copies complètes d'une arborescence (situées respectivement par convention dans les répertoires `/branches` et `/tags` qui se trouvent à la racine de l'arborescence du dépôt, à côté de `/trunk`). Dans l'ensemble du dépôt, plusieurs versions de chaque fichier peuvent être visibles : la dernière version à l'intérieur de chaque branche, la version au sein de chaque étiquette et, bien sûr, la dernière version dans le tronc lui-même (sous `/trunk`). Ainsi, pour être vraiment précis, il convient de dire « `machin.c` tel qu'il était dans `/branches/IDEE1` à la révision 5 ».

Pour plus de détails sur ce sujet, consultez la section intitulée « Révisions ».

Suivi de versions des répertoires

Subversion assure le suivi de l'arborescence entière, pas seulement des fichiers. C'est une des raisons majeures pour lesquelles Subversion a été créé, dans le but de se substituer à CVS.

Voilà ce que cela implique, du point de vue d'un ancien utilisateur de CVS :

- Les commandes **svn add** et **svn delete** fonctionnent désormais aussi sur les répertoires, de la même manière que sur les fichiers. Idem pour **svn copy** et **svn move**. Cependant, ces commandes n'ont pas d'effet immédiat sur le dépôt ; en effet, elles ne font que *planifier* l'ajout ou la suppression des éléments concernés. Aucun changement n'a lieu dans le dépôt tant que vous n'effectuez pas de propagation (commande **`svn commit`**).
- Les répertoires ne sont plus de simples conteneurs ; ils possèdent un numéro de révision tout comme les fichiers (ou pour être plus précis, il faut parler du « répertoire `machin/` tel qu'il était à la révision 5 »).

Approfondissons ce dernier point. La gestion de versions d'un répertoire est un problème difficile ; comme nous voulons autoriser des copies de travail à cheval sur plusieurs révisions, des limitations apparaissent quand on essaie de pousser le modèle trop loin.

D'un point de vue théorique, nous définissons « la révision 5 du répertoire `machin` » comme un ensemble d'entrées et de propriétés du répertoire. Maintenant, supposons que nous ajoutons et supprimons des fichiers de `machin`, puis que nous propageons ces modifications. Dire que nous avons toujours la révision 5 de `machin` est un mensonge. Cependant, si nous changeons le numéro de révision de `machin` après la propagation, c'est aussi un mensonge ; il peut y avoir d'autres changements sur `machin` que nous n'avons pas encore reçus parce que nous n'avons pas encore effectué de mise à jour (commande **svn update**).

Subversion traite ce problème en conservant discrètement, dans la zone `.svn`, le détail des ajouts et des suppressions propagés. Par la suite, quand vous lancez **`svn update`**, ces informations sont prises en compte et combinées avec celles du dépôt et le nouveau numéro de révision du répertoire est alors positionné correctement. Ainsi, *c'est seulement après une mise à jour que vous pouvez affirmer, sans risque de vous tromper, que vous disposez d'une révision « parfaite » d'un répertoire.* La plupart du temps, votre copie de travail contient des répertoires « imparfaitement » synchronisés.

De la même manière, un problème survient si vous essayez de propager des modifications de propriétés sur un répertoire. Normalement, la propagation devrait ajuster le numéro de révision du répertoire de la copie de travail locale. Mais là encore, ce serait un mensonge puisqu'il peut y avoir des ajouts et des suppressions que le répertoire n'a pas encore reçu en raison de la mise à jour qui n'a pas encore eu lieu. *En conséquence, il n'est pas permis de propager des changements sur les propriétés d'un répertoire sans que ce répertoire ne soit préalablement mis à jour.*

Pour plus d'informations sur les limitations de la gestion de versions des répertoires, reportez-vous à la section intitulée « Copies de travail mixtes, à révisions mélangées ».

Davantage d'opérations en mode déconnecté

Ces dernières années, l'espace de stockage sur disques est devenu outrageusement bon marché et abondant, alors que ce n'est pas le cas pour la bande passante disponible sur le réseau. En conséquence, les copies de travail Subversion ont été optimisées pour économiser la ressource la plus rare.

Le répertoire administratif `.svn` a le même objectif que le répertoire `CVS`, à la différence près qu'il stocke également des copies « originales » en lecture seule de vos fichiers. Ceci permet beaucoup d'opérations sans connexion réseau :

svn status
: liste les modifications locales que vous avez apportées (voir la section intitulée « Avoir une vue d'ensemble des changements effectués ») ;

svn diff
: donne le détail de vos modifications (voir la section intitulée « Voir en détail les modifications que vous avez effectuées ») ;

svn revert
: supprime vos modifications locales (voir la section intitulée « Annuler des changements sur la copie de travail »).

Par ailleurs, les fichiers originaux en cache permettent au client Subversion de n'envoyer que les différences au moment de la propagation, ce que CVS ne sait pas faire.

La dernière sous-commande de la liste (**svn revert**) est nouvelle. Elle supprime non seulement les modifications locales mais aussi annule les opérations planifiées telles que les ajouts et les suppressions. Bien que supprimer le fichier puis lancer **`svn update`** fonctionne toujours, cette méthode détourne la mise à jour de sa vocation. Et puis, tant que nous y sommes…

Distinction entre les commandes status et update

Subversion essaie de dissiper la confusion qui règne entre les commandes **cvs status** et **cvs update**.

La commande **cvs status** a deux objectifs : d'abord, lister pour l'utilisateur les modifications locales de la version de travail et, ensuite, indiquer à l'utilisateur quels fichiers ne sont plus à jour. Malheureusement, en raison de l'affichage peu lisible de la commande **`cvs status`**, beaucoup d'utilisateurs de CVS n'utilisent plus cette commande. À la place, ils ont pris l'habitude de lancer **`cvs update`** ou **`cvs -n update`** pour visualiser leurs changements rapidement. Si les utilisateurs oublient d'utiliser l'option `-n`, cela a pour effet de bord de fusionner des changements du dépôt qu'ils ne sont pas forcément prêts à prendre en compte.

Subversion supprime ce cafouillage en facilitant la lecture de **svn status** à la fois pour les humains et pour les programmes d'analyse de texte. De plus, **svn update** n'affiche que les informations relatives aux fichiers qui ont été mis à jour côté dépôt, *pas les modifications locales*.

svn status

La commande **svn status** liste tous les fichiers qui ont des modifications locales. Par défaut, le dépôt n'est pas contacté. Bien que cette sous-commande accepte un bon nombre d'options, voici les plus utilisées :

`-u`
: contacte le dépôt pour déterminer et lister les informations de mise à jour ;

`-v`
: liste *tous les éléments* suivis en versions ;

`-N`
: exécution non récursive (ne pas descendre dans les sous-répertoires).

La commande **svn status** possède deux formats de sortie. Dans le mode « court », mode par défaut, les modifications locales sont présentées comme ceci :

```
$ svn status
M      machin.c
M      truc/bidule.c
```

Si vous spécifiez l'option `--show-updates` (`-u`), un format d'affichage plus long sera utilisé :

```
$ svn status -u
M            1047   machin.c
       *     1045   tetes.html
       *            dessin.png
M            1050   truc/bidule.c
État par rapport à la révision  1066
```

Dans ce cas, deux nouvelles colonnes font leur apparition. La deuxième colonne contient une astérisque si le fichier ou le répertoire n'est plus à jour. La troisième colonne contient le numéro de révision de la copie de travail pour l'élément considéré. Dans l'exemple précédent, l'astérisque indique que `tetes.html` serait modifié lors d'une mise à jour et que `dessin.png` est un nouveau fichier dans le dépôt (l'absence d'un numéro de révision pour `dessin.png` indique qu'il n'existe pas dans la copie de travail locale).

Vous devriez maintenant jeter un œil à la liste des codes d'état possibles de svn status. Voici quelques codes d'état courants que vous rencontrerez :

```
A    L'Ajout de l'élément est planifié
D    La suppression de l'élément est planifiée
M    L'élément a été Modifié localement
C    L'élément est en Conflit (les changements n'ont pas été complètement
       fusionnés entre la version du dépôt et la version de la copie de travail)
X    L'élément est eXterne à cette copie de travail (il vient peut-être
       d'un autre dépôt). Voir la section intitulée « Définition de références
externes »
?    L'élément n'est pas suivi en versions
!    L'élément est manquant ou incomplet (supprimé par un moyen autre que
       Subversion)
```

Pour plus de détails sur la commande **svn status**, consultez la section intitulée « Avoir une vue d'ensemble des changements effectués ».

svn update

La commande **svn update** met à jour votre copie de travail et n'affiche que les informations relatives aux fichiers qui ont été mis à jour.

Subversion combine les codes CVS `P` et `U` dans le seul code `U`. Quand une fusion ou un conflit apparaît, Subversion se contente d'afficher `G` ou `C`, plutôt qu'une phrase complète, pour indiquer ce qui se passe.

Pour plus d'informations sur **svn update**, reportez-vous à la section intitulée « Mettre à jour votre copie de travail ».

Branches et étiquettes

Subversion ne fait pas de différence entre l'espace du système de fichiers et l'espace des branches ; les branches et les étiquettes sont de simples répertoires du système de fichiers. C'est probablement l'obstacle psychologique le plus important que les utilisateurs de CVS doivent franchir. Pour tout savoir sur ce sujet, rendez-vous au Chapitre 4, *Gestion des branches*.

Puisque Subversion traite les branches et les étiquettes comme de simples répertoires, les différentes lignes de développement de votre projet sont probablement hébergées dans des sous-répertoires du répertoire du projet principal. En conséquence, n'oubliez pas de faire vos extractions en spécifiant l'URL du sous-répertoire qui contient la ligne de développement que vous désirez et pas l'URL racine du projet. Sinon, il y a de grandes chances pour que vous récupériez une copie complète de votre projet, y compris toutes les branches et toutes les étiquettes[1].

Propriétés des méta-données

Une nouvelle fonctionnalité de Subversion est la possibilité d'affecter des méta-données arbitraires (ou « propriétés ») aux fichiers et répertoires. Les propriétés sont des couples nom/valeur arbitraires associés aux fichiers et répertoires de votre copie de travail locale.

Pour définir ou récupérer un nom de propriété, utilisez les sous-commandes **svn propset** et **svn propget**. Pour obtenir la liste de toutes les propriétés d'un objet, utilisez **svn proplist**.

Pour plus d'informations, reportez-vous à la section intitulée « Propriétés ».

Résolution des conflits

CVS signale les conflits par des « marqueurs de conflits » insérés directement dans les fichiers puis affiche un `C` pendant l'opération de mise à jour ou de fusion. Historiquement, ce comportement a généré beaucoup de problèmes, car CVS ne finit pas son travail. Beaucoup d'utilisateurs oublient (ou ne voient pas) le `C` une fois qu'il disparaît de l'écran. Ils oublient aussi souvent que les marqueurs de conflit sont toujours présents et, accidentellement, propagent des fichiers contenant ces marqueurs de conflit.

Subversion corrige ce problème de deux façons. D'abord, quand un conflit est détecté sur un fichier, Subversion enregistre le fait que le fichier est en conflit et refuse de propager des modifications concernant ce fichier tant que vous ne résolvez pas explicitement le conflit. Ensuite, Subversion 1.5 propose une résolution interactive des conflits, ce qui vous permet de résoudre les conflits au moment où ils apparaissent, plutôt que d'avoir à revenir dessus une fois que la fusion ou la mise à jour est terminée. Consultez la section intitulée « Résoudre les conflits (fusionner des modifications) » pour plus d'informations sur la résolution des conflits avec Subversion.

Fichiers binaires et conversions

En général, Subversion se débrouille mieux avec les fichiers binaires que CVS. Comme CVS utilise RCS, il est seulement capable de stocker successivement des copies entières d'un fichier binaire modifié. Subversion, en revanche, utilise un algorithme de différenciation binaire, indépendant du contenu textuel ou binaire des fichiers, pour déterminer les différences entre les fichiers. Cela veut dire que tous les fichiers sont stockés de manière différentielle (compressée) dans le dépôt.

[1] Tout du moins si vous avez suffisamment d'espace disque pour pouvoir terminer l'extraction.

Les utilisateurs de CVS doivent marquer les fichiers binaires avec l'indicateur `-kb` pour empêcher que les données ne soient corrompues (par l'expansion des mots-clés et les conversions de fins de lignes). Ils oublient quelquefois de le faire.

Subversion utilise une approche plus paranoïaque. Premièrement, il ne fait aucune substitution de mot-clé ou de fin de ligne à moins qu'on ne le lui demande explicitement (voir la section intitulée « Substitution de mots-clés » et la section intitulée « Caractères de fin de ligne » pour plus de détails). Par défaut, Subversion considère que les fichiers de données sont des suites littérales d'octets et les fichiers sont toujours stockés dans le dépôt dans un état « non-converti ».

Deuxièmement, Subversion conserve une notion interne pour le contenu de chaque fichier : « texte » ou « binaire ». Mais *cette notion ne s'applique qu'à la copie de travail locale*. Lors d'un **svn update**, Subversion effectue des fusions contextuelles sur les fichiers texte modifiés localement mais ne tente pas d'en faire autant pour les fichiers binaires.

Pour déterminer si une fusion contextuelle est possible, Subversion examine la propriété `svn:mime-type`. Si le fichier ne possède pas la propriété `svn:mime-type` ou si le type MIME est textuel (par exemple `text/*`), Subversion considère que c'est du texte. Sinon, Subversion considère que le fichier est binaire. Subversion aide également les utilisateurs en incluant l'exécution d'un algorithme de détection des fichiers binaires dans les commandes **svn import** et **svn add**. Ces commandes tentent de deviner puis affectent (éventuellement) un type binaire à la propriété `svn:mime-type` du fichier ajouté (si Subversion se trompe dans la détection, l'utilisateur peut toujours supprimer ou éditer à la main la propriété).

Gestion de versions des modules

Contrairement à CVS, une copie de travail locale de Subversion sait qu'elle est l'extraction d'un module. Cela signifie que si quelqu'un change la définition du module (par exemple ajoute ou supprime des composants), un appel à **svn update** met à jour la copie de travail correctement, en ajoutant et supprimant les composants concernés.

Subversion définit les modules comme une liste de répertoires formant une propriété d'un répertoire ; voir la section intitulée « Définition de références externes ».

Authentification

Avec le pserver de CVS, vous devez vous connecter au serveur (en utilisant la commande **cvs login**) avant n'importe quelle opération de lecture ou d'écriture — parfois, vous devez même vous authentifier pour des opérations en mode anonyme. Avec un dépôt Subversion utilisant Apache **httpd** ou **svnserve**, vous n'avez pas besoin de vous authentifier a priori — si une opération nécessite que vous vous authentifiez, le serveur vous demande de le faire (que ce soit par identifiant et mot de passe, certificat client ou les deux). Ainsi, si votre dépôt est accessible en lecture pour tous, vous n'avez pas besoin de vous authentifier pour les opérations de lecture.

Comme CVS, Subversion met en cache sur le disque vos éléments d'authentification (dans votre répertoire `~/.subversion/auth/`) à moins que vous ne lui spécifiez le contraire avec l'option `--no-auth-cache`.

Ce comportement possède une exception : l'accès à un serveur **svnserve** via un tunnel SSH, en utilisant les URL de type `svn+ssh://`. Dans ce cas, le programme **ssh** vous demande toujours de vous authentifier avant d'ouvrir le tunnel.

Conversion d'un dépôt CVS vers Subversion

La meilleure façon d'habituer les utilisateurs CVS à Subversion est certainement de les laisser continuer à travailler sur leurs projets en utilisant le nouveau système. Et, bien que cela puisse être fait par un import « à plat » dans le dépôt Subversion d'un dépôt CVS exporté, la solution la plus aboutie implique de transférer d'un système à l'autre non seulement la dernière version des données mais aussi tout l'historique qui va avec. C'est un problème particulièrement ardu ; cela implique, parmi d'autres complications, de déterminer les modifications qui vont ensemble, en l'absence d'atomicité et de mécanisme de conversion entre les politiques totalement contradictoires de gestion des branches des deux systèmes. Néanmoins, il existe quelques outils qui se prétendent capables de convertir, au moins en partie, des dépôts CVS en dépôts Subversion.

L'outil le plus populaire (et le plus abouti) est cvs2svn (http://cvs2svn.tigris.org/ site en anglais), un programme en Python créé à l'origine par des membres de la communauté de développement Subversion elle-même. Cet outil n'est censé être lancé qu'une seule fois : il analyse votre dépôt CVS en plusieurs passes et essaie d'en déduire des propagations, des branches et des étiquettes autant qu'il le peut. Au final, le résultat est soit un dépôt Subversion soit un fichier dump Subversion représentant l'historique du code. Consultez le site web pour le détail des instructions et les précautions d'usage.

Annexe C. WebDAV et la gestion de versions automatique

WebDAV est une extension de HTTP qui devient de plus en plus courante comme standard pour le partage de fichiers. De nos jours, les systèmes d'exploitation sont de plus en plus « connectés » et beaucoup supportent maintenant nativement le montage de « partages » mis à disposition par des serveurs WebDAV.

Si vous utilisez Apache comme serveur réseau pour Subversion, on peut dire que d'une certaine manière vous faites aussi tourner un serveur WebDAV. Cette annexe donne quelques notions générales sur ce protocole, la manière dont Subversion l'utilise et l'état actuel de l'interopérabilité entre Subversion et d'autres logiciels compatibles WebDAV.

À propos de WebDAV

DAV signifie *Distributed Authoring and Versioning*, que l'on pourrait traduire par « Écriture distribuée et gestion de versions ». La RFC 2518 définit un ensemble de concepts et d'extensions sur la base du protocole HTTP 1.1 afin de faire du Web un périphérique de lecture/écriture plus universel. L'idée est qu'un serveur Web compatible WebDAV peut être considéré comme un serveur de fichiers générique ; les clients peuvent « monter » des répertoires partagés, au-dessus d'une couche HTTP, et ces répertoires se comportent pratiquement comme les autres systèmes de fichiers en réseau (tels que NFS ou SMB).

Là où le bât blesse, c'est que, malgré l'acronyme, les spécifications de la RFC ne décrivent en fait aucun façon d'assurer la gestion de versions. Les clients et serveurs WebDAV de base considèrent que chaque fichier ou répertoire n'existe qu'en une seule version, qui peut être ré-écrite autant de fois que l'on veut.

Comme la RFC 2518 a ignoré les concepts de gestion de versions, un autre groupe de travail a hérité de la responsabilité d'écrire la RFC 3253 quelques années plus tard. La nouvelle RFC ajoute le concept de gestion de versions à WebDAV, en rendant sa signification au « V » de « DAV », d'où le terme « DeltaV ». Les clients et serveurs WebDAV/DeltaV sont souvent appelés « DeltaV » tout court, puisque DeltaV implique obligatoirement la prise en compte de WebDAV.

Le standard WebDAV initial a connu un grand succès. Tous les systèmes d'exploitation modernes ont un client WebDAV intégré (nous les verrons en détail plus tard) et de nombreux logiciels sont également capables de communiquer via ce protocole : Microsoft Office, Dreamweaver, Photoshop pour n'en citer que quelques uns. Côté serveur, le serveur HTTP Apache dispose de services WebDAV depuis 1998 et est considéré *de facto* comme la référence en matière de logiciel libre. Il existe plusieurs autres serveurs WebDAV commerciaux, dont le propre serveur de Microsoft, IIS.

Malheureusement, DeltaV n'a pas connu autant de succès. Il est très difficile de trouver des clients ou des serveurs DeltaV. Les rares qui existent sont des serveurs commerciaux plus ou moins inconnus ; l'interopérabilité est donc très difficile à tester. Il n'est pas évident de trouver pourquoi DeltaV n'a pas percé. Certains mettent en cause des spécifications trop complexes. D'autres arguent du fait que, contrairement à WebDAV, qui est une technologie de masse (même les utilisateurs les moins férus d'informatique aiment à partager des fichiers en réseau), ses fonctionnalités de gestion de versions intéressent peu ou ne sont pas nécessaires à la majorité des gens. Enfin, certains sont persuadés que DeltaV n'intéresse pas grand monde parce qu'il n'existe aucun serveur libre qui l'implémente correctement.

Quand Subversion était encore en phase de conception, l'utilisation d'Apache comme serveur réseau paraissait une très bonne idée. Il possédait déjà un module pour fournir des services WebDAV et DeltaV était une spécification relativement jeune. L'idée était que le module serveur de Subversion (**mod_dav_svn**) évoluerait pour devenir l'implémentation libre de référence de DeltaV. Malheureusement, DeltaV possède un modèle de gestion de versions très particulier, qui n'est pas vraiment compatible avec le modèle de Subversion. Certains concepts pouvaient être adaptés, mais d'autres non.

Quelles conséquences en tirer ?

Premièrement, le client Subversion n'est pas un client DeltaV complet. Il a besoin d'informations de la part du serveur que DeltaV est incapable de lui fournir, ce qui implique qu'il dépend en grande partie de requêtes HTTP `REPORT` spécifiques à Subversion que seul **mod_dav_svn** sait interpréter.

Deuxièmement, **mod_dav_svn** n'implémente pas toutes les fonctionnalités d'un serveur DeltaV. De nombreux éléments des spécifications du protocole DeltaV ne sont pas pertinents dans le cas de Subversion et n'ont donc pas été implémentés.

Le débat est toujours d'actualité au sein de la communauté des développeurs pour savoir si cela vaut la peine de combler ces lacunes. Il serait irréaliste de modifier l'architecture de Subversion pour rallier celle de DeltaV, et donc le client ne sera sans

doute jamais capable d'obtenir toutes les informations nécessaires d'un serveur DeltaV. D'un autre côté, **mod_dav_svn** *pourrait* être complété pour intégrer les fonctionnalités manquantes de DeltaV, mais il est difficile de se motiver pour le faire : il n'existe pratiquement aucun client DeltaV avec qui communiquer.

Gestion de versions automatique

Bien que le client Subversion ne soit pas un client DeltaV complet et que le serveur Subversion n'implémente pas toutes les fonctionnalités d'un serveur DeltaV, il faut se féliciter de l'existence d'une petite lueur d'intéropérabilité WebDAV : la *gestion de versions automatique*.

La gestion de versions automatique est une fonctionnalité optionnelle définie dans le standard DeltaV. Un serveur DeltaV classique n'autorisera pas un client WebDAV non compatible à effectuer des opérations `PUT` sur un fichier suivi en versions. Pour modifier un tel fichier, le serveur exige un enchaînement précis de requêtes de gestion de versions : quelque chose comme `MKACTIVITY`, `CHECKOUT`, `PUT`, `CHECKIN` (c'est-à-dire : créer une activité, extraire le fichier suivi en versions, renvoyer le fichier modifié et assortir cette modification d'un commentaire). Mais si le serveur DeltaV supporte la fonctionnalité de gestion de versions automatique, les requêtes en écriture des clients WebDAV ordinaires sont acceptées. Le serveur agit *comme si* le client avait envoyé l'enchaînement de requêtes approprié, en faisant une propagation « sous le manteau ». En d'autres termes, la gestion de versions automatique permet à un serveur DeltaV de communiquer avec des clients WebDAV ordinaires qui ne disposent pas de fonctionnalités de gestion de versions.

Comme beaucoup de systèmes d'exploitation ont des clients WebDAV intégrés, cette fonctionnalité est particulièrement intéressante pour les administrateurs qui travaillent avec des utilisateurs non techniciens. Imaginez un bureau avec des utilisateurs « ordinaires » sous Microsoft Windows ou Mac OS. Chaque utilisateur « monte » le dépôt Subversion qui apparaît comme un lecteur réseau classique. Ils utilisent le partage réseau comme ils l'ont toujours fait : ils ouvrent les fichiers, les modifient et les sauvegardent. Pendant ce temps, le serveur assure automatiquement la gestion de versions. L'administrateur (ou tout autre utilisateur sachant le faire) peut toujours utiliser un client Subversion pour effectuer des requêtes sur l'historique des fichiers ou récupérer une vieille version.

Ce scénario n'est pas de la science-fiction : c'est du concret, qui fonctionne depuis la version 1.2 de Subversion. Pour activer la gestion de versions automatique dans **mod_dav_svn**, utilisez la directive `SVNAutoversioning` dans le bloc `Location` du fichier `httpd.conf`, comme dans l'exemple suivant :

```
<Location /depot>
  DAV svn
  SVNPath /var/svn/depot
  SVNAutoversioning on
</Location>
```

Quand la gestion de versions automatique de Subversion est active, les requêtes en écriture des clients WebDAV sont automatiquement transformées en propagations. Un message de propagation générique est créé et associé automatiquement à chaque révision.

Cependant, avant d'activer cette fonctionnalité, comprenez bien dans quoi vous vous engagez. Les clients WebDAV ont tendance à effectuer *beaucoup* de requêtes en écriture, ce qui engendre un nombre astronomique de propagations automatiques. Par exemple, lors d'une sauvegarde d'un fichier, beaucoup de clients effectuent un `PUT` pour un fichier de 0 octets (pour signifier qu'ils réservent le nom) suivi par un autre `PUT` avec les données effectives du fichier. La simple écriture d'un fichier entraîne ainsi deux propagations distinctes. Tenez également compte du fait que de nombreuses applications effectuent des sauvegardes automatiques régulièrement, toutes les cinq minutes par exemple, qui se traduisent par autant de propagations.

Si vous avez une procédure automatique qui envoie un e-mail après chaque propagation (`post-commit`), il est conseillé de désactiver cet envoi soit complètement soit au moins pour certaines parties du dépôt, selon que ces e-mails vous semblent apporter une plus-value ou pas. De plus, une procédure automatique post-commit bien pensée peut distinguer une propagation générée par la gestion de versions automatique d'une propagation classique. L'astuce consiste à examiner la propriété de révision dénommée `svn:autoversioned`. Si elle est présente, la propagation est issue d'un client WebDAV quelconque.

Une autre caractéristique utile et complémentaire de la gestion de versions automatique de Subversion est fournie par le module `mod_mime` d'Apache. Si un client WebDAV ajoute un nouveau fichier au dépôt, l'utilisateur n'a pas l'occasion de lui adjoindre la propriété `svn:mime-type`. Dans ce cas, il se peut que, lors de la navigation dans un répertoire partagé WebDAV, l'icône du fichier soit générique et qu'aucune application ne soit associée à ce fichier. Une solution peut être qu'un administrateur système (ou toute autre personne sachant utiliser Subversion) extraie une copie de travail et définisse

manuellement la propriété `svn:mime-type` sur les fichiers concernés. Mais c'est un peu comme tenter de remplir le tonneau des Danaïdes, alors qu'il suffit de placer la directive `ModMimeUsePathInfo` dans le bloc `<Location>` de Subversion.

```
<Location /depot>
  DAV svn
  SVNPath /var/svn/depot
  SVNAutoversioning on

  ModMimeUsePathInfo on

</Location>
```

Cette directive autorise `mod_mime` à déduire automatiquement le type MIME des nouveaux fichiers qui entrent dans le dépôt de la gestion de versions automatique. Ce module examine l'extension du nom de fichier et éventuellement le contenu de celui-ci ; si certains motifs sont repérés, la propriété `svn:mime-type` est automatiquement renseignée.

Interopérabilité des clients

Les clients WebDAV peuvent être classés en trois catégories : applications autonomes, greffons pour explorateurs de fichiers et implémentations de systèmes de fichiers. Ces catégories définissent grosso modo les types de fonctionnalités WebDAV offertes aux utilisateurs. Le Tableau C.1, « Principaux clients WebDAV » contient notre répartition en catégories et fournit une brève description des principaux logiciels compatibles WebDAV. Vous trouverez plus d'informations sur ces logiciels, ainsi que sur les catégories auxquelles ils appartiennent, dans les sections à suivre.

Tableau C.1. Principaux clients WebDAV

Logiciel	Type	Windows	Mac	Linux	Description
Adobe Photoshop	Application WebDAV autonome	X			Logiciel de retouche d'images, capable d'accéder directement à des URL WebDAV, en lecture et en écriture
cadaver	Application WebDAV autonome		X	X	Client WebDAV en ligne de commande, supportant des opérations de transfert de fichiers, d'arborescences et de verrouillage
DAV Explorer	Application WebDAV autonome	X	X	X	Interface graphique en Java dont le but est de parcourir des partages WebDAV
Adobe Dreamweaver	Application WebDAV autonome	X			Logiciel de création Web, capable d'accéder directement à des URL WebDAV, en lecture et en écriture
Microsoft Office	Application WebDAV autonome	X			Suite bureautique dont plusieurs composants sont capables d'accéder directement à des URL WebDAV, en

Logiciel	Type	Windows	Mac	Linux	Description
					lecture et en écriture
Dossiers Web de Microsoft	Greffon WebDAV pour explorateur de fichiers	X			Explorateur de fichiers avec interface graphique capable d'effectuer des opérations sur les arborescences de partages WebDAV
GNOME Nautilus	Greffon WebDAV pour explorateur de fichiers			X	Explorateur de fichiers avec interface graphique capable d'effectuer des opérations sur les arborescences de partages WebDAV
KDE Konqueror	Greffon WebDAV pour explorateur de fichiers			X	Explorateur de fichiers avec interface graphique capable d'effectuer des opérations sur les arborescences de partages WebDAV
Mac OS X	Implémentation d'un système de fichiers WebDAV		X		Système d'exploitation capable de monter des partages WebDAV nativement
Novell NetDrive	Implémentation d'un système de fichiers WebDAV	X			Logiciel qui permet d'affecter des partages réseaux WebDAV à des lecteurs réseaux Windows
SRT WebDrive	Implémentation d'un système de fichiers WebDAV	X			Logiciel de transfert de fichiers qui, entre autres choses, permet d'affecter des partages réseaux WebDAV à des lecteurs réseaux Windows
davfs2	Implémentation d'un système de fichiers WebDAV			X	Pilote de système de fichiers Linux qui permet de monter des partages WebDAV

Applications WebDAV autonomes

Une application WebDAV est un programme qui communique avec un serveur WebDAV en utilisant les protocoles WebDAV. Nous allons passer en revue les programmes les plus populaires dans cette catégorie.

Microsoft Office, Dreamweaver, Photoshop

Sous Windows, plusieurs applications bien connues intègrent nativement un client WebDAV, comme par exemple Microsoft Office[1], Photoshop d'Adobe et Dreamweaver. Ils sont capables d'accéder à des URL WebDAV, à la fois en lecture et en écriture, et ont tendance à faire un usage intensif des verrous WebDAV lors de l'édition d'un fichier.

Notez que bien que beaucoup de ces programmes fonctionnent également sous Mac OS X, ils ne semblent pas directement supporter WebDAV sur cette plate-forme. En fait, sous Mac OS X, la boîte de dialogue Fichier#Ouvrir ne permet d'entrer ni un chemin ni une URL. Il est probable que le support de WebDAV ait été délibérément laissé de côté sur les versions Macintosh de ces programmes, puisque le système de fichiers de bas niveau d'OS X est déjà lui-même hautement compatible avec WebDAV.

cadaver, DAV Explorer

cadaver (cadavre en français) est un programme Unix rudimentaire en ligne de commande pour parcourir et modifier des partages WebDAV. Comme le client Subversion, il utilise la bibliothèque HTTP neon — ce qui n'est pas surprenant puisque `neon` et **cadaver** sont écrits par le même auteur. **cadaver** est un logiciel libre (licence GPL) disponible à l'adresse http://www.webdav.org/cadaver/ (site en anglais).

L'utilisation de **cadaver** est similaire à l'utilisation d'un programme FTP en ligne de commande et se révèle donc extrêmement utile pour effectuer du débogage sur WebDAV. Il peut être utilisé pour transférer rapidement plusieurs fichiers du serveur vers son ordinateur ou de son ordinateur vers le serveur, pour examiner les propriétés et pour copier, déplacer, verrouiller ou déverrouiller les fichiers :

```
$ cadaver http://hote/depot
dav:/depot/> ls
Listing collection `/depot/': succeeded.
Coll: > machintruc                              0  May 10 16:19
      > auteur.el                            2864  May  4 16:18
      > preuve-en-poeme.txt                  1461  May  5 15:09
      > cote-d-azur.jpg                     66737  May  5 15:09

dav:/depot/> put LISEZMOI
Uploading LISEZMOI to `/depot/LISEZMOI':
Progress: [==============================>] 100.0% of 357 bytes succeeded.

dav:/depot/> get preuve-en-poeme.txt
Downloading `/depot/preuve-en-poeme.txt' to preuve-en-poeme.txt:
Progress: [==============================>] 100.0% of 1461 bytes succeeded.
```

DAV Explorer est un autre client WebDAV autonome, écrit en Java. Il est sous une licence libre de type Apache et est disponible à l'adresse http://www.ics.uci.edu/~webdav/ (site en anglais). Il peut faire tout ce que fait **cadaver** et a l'avantage d'être portable, ainsi que d'avoir une interface graphique plus conviviale. C'est aussi un des premiers clients à supporter le nouveau protocole « WebDAV Access Control Protocol » (RFC 3744).

Bien sûr, le fait que DAV Explorer supporte les listes de contrôle d'accès (ACL) n'a aucun intérêt pour nous, puisque **mod_dav_svn** ne les supporte pas. Le support limité de certaines commandes DeltaV par **cadaver** et DAV Explorer n'est pas particulièrement utile non plus puisqu'ils n'autorisent pas les requêtes `MKACTIVITY`. Mais cela n'est pas pertinent de toute manière ; nous considérons que tous ces clients se connectent à des dépôts avec gestion de versions automatique.

Greffons WebDAV pour explorateur de fichiers

Certains explorateurs de fichiers avec interface graphique bien connus disposent de greffons pour WebDAV qui permettent à l'utilisateur de parcourir un partage DAV comme si c'était un répertoire sur l'ordinateur local et d'effectuer des opérations de base sur l'arborescence partagée. Par exemple, Windows Explorer est capable de parcourir un serveur WebDAV en tant qu'« emplacement réseau ». Les utilisateurs peuvent glisser-déposer des fichiers depuis et vers le bureau, ou peuvent renommer, copier ou effacer des fichiers comme d'habitude. Mais comme cette fonctionnalité est propre à l'explorateur de fichiers, le partage DAV n'est pas visible par les applications ordinaires. Toutes les interactions DAV doivent passer par l'intermédiaire de l'interface de l'explorateur.

[1]Microsoft a décidé de supprimer le support WebDAV d'Access mais il existe toujours pour le reste de la suite Office.

Dossiers Web de Microsoft

Microsoft faisait partie du groupe de travail sur les spécifications WebDAV et a commencé à livrer un client WebDAV avec Windows 98, sous le nom de « Dossiers Web ». Ce client était également livré avec Windows NT 4.0 et Windows 2000.

Le client originel « Dossiers Web » était une extension de l'explorateur Windows, la principale interface utilisée pour parcourir le système de fichiers. Il fonctionne plutôt bien. Sous Windows 98, l'extension doit être explicitement installée si « Dossiers Web » n'est pas visible dans « Mon Ordinateur ». Sous Windows 2000, ajoutez simplement un nouveau « Favori réseau », entrez l'URL et le partage WebDAV apparaît, prêt à être parcouru.

Avec la sortie de Windows XP, Microsoft a commencé à livrer une nouvelle version de « Dossiers Web », connue sous le nom de mini-redirecteur WebDAV. Cette nouvelle implémentation est un client permettant au niveau du système de fichiers de monter les partages WebDAV en tant que lecteurs réseau. Malheureusement, cette implémentation est incroyablement boguée. Le client essaie généralement de convertir les URL HTTP (`http://hote/depot`) en notation UNC (`\\hote\depot`) ; il essaie aussi souvent d'utiliser l'authentification de domaine Windows pour répondre aux défis d'authentification de la méthode basic-auth d'HTTP en envoyant les identifiants sous la forme `HOTE\identifiant`. Ces problèmes d'interopérabilité sont graves et sont décrits un peu partout sur le Web, à la grande frustration de beaucoup d'utilisateurs. Même Greg Stein, l'auteur initial du module Apache WebDAV, affirme sèchement que les « Dossiers Web » de Windows XP ne peuvent tout simplement pas fonctionner avec un serveur Apache.

La version initiale des « Dossiers Web » de Windows Vista semble être pratiquement la même que celle de Windows XP et les problèmes sont donc les mêmes. Avec de la chance, Microsoft corrigera ces problèmes dans un Service Pack de Vista.

Cependant, il semble qu'il existe des solutions de contournement, à la fois pour XP et pour Vista, qui permettent aux « Dossiers Web » de fonctionner avec un serveur Apache. Les témoignages des utilisateurs de ces solutions sont majoritairement positifs, c'est pourquoi nous les signalons ici.

Avec Windows XP, vous avez deux options : la première, chercher sur le site Web de Microsoft le correctif KB907306, « Mise à jour de logiciels pour les dossiers Web ». Ceci devrait résoudre vos problèmes. Si ce n'est pas le cas, il semble que la version originale pré-XP des « Dossiers Web » soit toujours fournie avec le système. Vous pouvez y accéder en allant sur « Favoris Réseau » et en ajoutant un nouveau « Favori réseau ». Quand le système vous demande d'entrer une URL, saisissez l'URL du dépôt *en incluant un numéro de port* dans cette URL. Par exemple, vous devez entrer **`http://hote:80/depot`** au lieu de **`http://hote/depot`**. Répondez ensuite avec vos identifiants Subversion à toute demande d'authentification.

Avec Windows Vista, le même correctif KB907306 devrait faire l'affaire. Mais il est possible qu'il reste d'autres problèmes. Certains utilisateurs rapportent que Vista considère toute connexion `http://` comme non sécurisée et fait donc échouer toute tentative d'authentification avec Apache à moins que la connexion n'utilise `https://`. Si vous ne pouvez pas vous connecter à un dépôt Subversion en SSL, vous pouvez modifier la base de registre du système pour inhiber ce comportement. Changez uniquement la valeur de la clé `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\WebClient\Parameters\BasicAuthLevel` de **`1`** à **`2`**. Et finalement, en guise d'avertissement : assurez-vous de faire pointer votre favori réseau vers le répertoire racine du dépôt (`/`), plutôt que vers un sous-répertoire comme `/trunk`. Les « Dossiers Web » de Vista semblent ne fonctionner qu'avec les répertoires racines des dépôts.

En général, et bien que ces astuces puissent vous être utiles, vous serez peut-être mieux servi avec un client WebDAV tiers tel que WebDrive ou NetDrive.

Nautilus, Konqueror

Nautilus est l'explorateur/navigateur de fichiers officiel du bureau GNOME (http://www.gnome.org) et Konqueror est l'explorateur/navigateur de fichiers officiel du bureau KDE (http://www.kde.org). Ces deux applications disposent d'un client WebDAV intégré dans l'explorateur de fichiers et elles peuvent se connecter sans problème à un dépôt utilisant la gestion de versions automatique.

Dans Nautilus (GNOME), choisissez l'élément de menu Fichier#Ouvrir un emplacement et entrez l'URL dans la boîte de dialogue qui s'affiche. Le dépôt s'affiche alors comme n'importe quel autre système de fichiers.

Dans Konqueror (KDE), vous devez utiliser la syntaxe `webdav://` pour entrer une URL dans la barre d'adresse. Si vous entrez une URL en `http://`, Konqueror se comporte comme un navigateur classique. Vous risquez alors de voir le contenu du répertoire dans une page HTML classique produite par **mod_dav_svn**. Si vous entrez **`webdav://hote/depot`** à la place de **`http://hote/depot`**, Konqueror devient un client WebDAV et affiche le dépôt comme un système de fichiers.

Implémentations de WebDAV en système de fichiers

L'implémentation en système de fichiers peut être considérée avec raison comme le meilleur type de client WebDAV. Il est alors implanté en tant que module bas-niveau, généralement au sein du noyau du système d'exploitation. Cela signifie qu'un partage DAV est monté comme tout autre système de fichiers en réseau, de la même manière qu'un partage NFS est monté sous Unix ou qu'un lecteur réseau est affecté à un partage SMB sous Windows. Au final, ce type de client fournit l'accès WebDAV en lecture/écriture de manière transparente à tous les programmes. Les applications ne sont même pas conscientes des requêtes WebDAV qu'elles génèrent.

WebDrive, NetDrive

WebDrive et NetDrive sont tous deux d'excellents produits commerciaux qui permettent, sous Windows, d'affecter un lecteur réseau à un partage WebDAV. Vous pouvez ainsi effectuer des opérations sur ces pseudo-lecteurs WebDAV aussi facilement et de la même manière que sur un disque dur local. Vous pouvez vous procurer WebDrive auprès de South River Technologies (http://www.southrivertech.com). NetDrive de Novell est disponible gratuitement en ligne mais exige d'avoir une licence Netware.

Mac OS X

Le système d'exploitation OS X d'Apple possède un client WebDAV intégré sous forme de système de fichiers. Depuis Finder (l'explorateur de fichiers), choisissez l'élément de menu Go#Connect to Server. Entrez l'URL WebDAV et elle apparaît en tant que disque sur le bureau, comme tout autre volume monté. Vous pouvez également monter un partage WebDAV avec la commande **mount** depuis le terminal Darwin, en utilisant le type de système de fichiers `webdav` :

```
$ mount -t webdav http://svn.exemple.com/depot/projet /mon/point/de/montage
$
```

Notez que si votre module **mod_dav_svn** est plus ancien que la version 1.2, OS X refuse de monter le partage en lecture/écriture ; il apparaît en lecture seule. C'est parce que OS X exige le verrouillage pour les partages en lecture/écriture, et la possibilité de verrouiller des fichiers n'est apparue que dans Subversion 1.2.

Par ailleurs, le client WebDAV d'OS X peut de temps en temps se montrer très sensible aux redirections HTTP. Si OS X est incapable de monter le dépôt, vous devrez peut-être activer la directive `BrowserMatch` dans le fichier de configuration `httpd.conf` du serveur Apache :

```
BrowserMatch "^WebDAVFS/1.[012]" redirect-carefully
```

davfs2 (Linux)

davfs2 est un module de système de fichiers du noyau Linux. Le projet est hébergé à l'adresse http://savannah.nongnu.org/projects/davfs2. Une fois davfs2 installé, vous pouvez monter un partage réseau WebDAV en utilisant la commande **mount** habituelle de Linux :

```
$ mount.davfs http://hote/depot /mnt/dav
```

Annexe D. Copyright

unmodified form, along with a number of other contributions, constituting separate and independent works in themselves, are assembled into a collective whole. A work that constitutes a Collective Work will not be considered a Derivative Work (as defined below) for the purposes of this License.

b. "Derivative Work" means a work based upon the Work or upon the Work and other pre-existing works, such as a translation, musical arrangement, dramatization, fictionalization, motion picture version, sound recording, art reproduction, abridgment, condensation, or any other form in which the Work may be recast, transformed, or adapted, except that a work that constitutes a Collective Work will not be considered a Derivative Work for the purpose of this License. For the avoidance of doubt, where the Work is a musical composition or sound recording, the synchronization of the Work in timed-relation with a moving image ("synching") will be considered a Derivative Work for the purpose of this License.

c. "Licensor" means the individual or entity that offers the Work under the terms of this License.

d. "Original Author" means the individual or entity who created the Work.

e. "Work" means the copyrightable work of authorship offered under the terms of this License.

f. "You" means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.

2. Fair Use Rights. Nothing in this license is intended to reduce, limit, or restrict any rights arising from fair use, first sale or other limitations on the exclusive rights of the copyright owner under copyright law or other applicable laws.

3. License Grant. Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

a. to reproduce the Work, to incorporate the Work into one or more Collective Works, and to reproduce the Work as incorporated in the Collective Works;

b. to create and reproduce Derivative Works;

c. to distribute copies or phonorecords of, display publicly, perform publicly, and perform publicly by means of a digital audio transmission the Work including as incorporated in Collective Works;

d. to distribute copies or phonorecords of, display publicly, perform publicly, and perform publicly by means of a digital audio transmission Derivative Works.

e.

For the avoidance of doubt, where the work is a musical composition:

i. Performance Royalties Under Blanket Licenses. Licensor waives the exclusive right to collect, whether individually or via a performance rights society (e.g. ASCAP, BMI, SESAC), royalties for the public performance or public digital performance (e.g. webcast) of the Work.

ii. Mechanical Rights and Statutory Royalties. Licensor waives the exclusive right to collect, whether individually or via a music rights agency or designated agent (e.g. Harry Fox Agency), royalties for any phonorecord You create from the Work ("cover version") and distribute, subject to the compulsory license created by 17 USC Section 115 of the US Copyright Act (or the equivalent in other jurisdictions).

f. Webcasting Rights and Statutory Royalties. For the avoidance of doubt, where the Work is a sound recording, Licensor waives the exclusive right to collect, whether individually or via a performance-rights society (e.g. SoundExchange), royalties for the public digital performance (e.g. webcast) of the Work, subject to the compulsory license created by 17 USC Section 114 of the US Copyright Act (or the equivalent in other jurisdictions).

The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats. All rights not expressly granted by Licensor are hereby reserved.

4. Restrictions.The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:

a. You may distribute, publicly display, publicly perform, or publicly digitally perform the Work only under the terms of this License, and You must include a copy of, or the Uniform Resource Identifier for, this License with every copy or phonorecord of the Work You distribute, publicly display, publicly perform, or publicly digitally perform. You may not offer or impose any terms on the Work that alter or restrict the terms of this License or the recipients' exercise of the rights granted hereunder. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties. You may not distribute, publicly display, publicly perform, or publicly digitally perform the Work with any technological measures that control access or use of the Work in a manner inconsistent with the terms of this License Agreement. The above applies to the Work as incorporated in a Collective Work, but this does not require the Collective Work apart from the Work itself to be made subject to the terms of this License. If You create a Collective Work, upon notice from any Licensor You must, to the extent practicable, remove from the Collective Work any reference to such Licensor or the Original Author, as requested. If You create a Derivative Work, upon notice from any Licensor You must, to the extent practicable, remove from the Derivative Work any reference to such Licensor or the Original Author, as requested.

b. If you distribute, publicly display, publicly perform, or publicly digitally perform the Work or any Derivative Works or Collective Works, You must keep intact all copyright notices for the Work and give the Original Author credit reasonable to the medium or means You are utilizing by conveying the name (or pseudonym if applicable) of the Original Author if supplied; the title of the Work if supplied; to the extent reasonably practicable, the Uniform Resource Identifier, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work; and in the case of a Derivative Work, a credit identifying the use of the Work in the Derivative Work (e.g., "French translation of the Work by Original Author," or "Screenplay based on original Work by Original Author"). Such credit may be implemented in any reasonable manner; provided, however, that in the case of a Derivative Work or Collective Work, at a minimum such credit will appear where any other

comparable authorship credit appears and in a manner at least as prominent as such other comparable authorship credit.

5. Representations, Warranties and Disclaimer

UNLESS OTHERWISE MUTUALLY AGREED TO BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTIBILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

6. Limitation on Liability. EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. Termination

   a. This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Derivative Works or Collective Works from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.

   b. Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. Miscellaneous

   a. Each time You distribute or publicly digitally perform the Work or a Collective Work, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.

   b. Each time You distribute or publicly digitally perform a Derivative Work, Licensor offers to the recipient a license to the original Work on the same terms and conditions as the license granted to You under this License.

   c. If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

   d. No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.

   e. This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no

understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.

Creative Commons is not a party to this License, and makes no warranty whatsoever in connection with the Work. Creative Commons will not be liable to You or any party on any legal theory for any damages whatsoever, including without limitation any general, special, incidental or consequential damages arising in connection to this license. Notwithstanding the foregoing two (2) sentences, if Creative Commons has expressly identified itself as the Licensor hereunder, it shall have all rights and obligations of Licensor.

Except for the limited purpose of indicating to the public that the Work is licensed under the CCPL, neither party will use the trademark "Creative Commons" or any related trademark or logo of Creative Commons without the prior written consent of Creative Commons. Any permitted use will be in compliance with Creative Commons' then-current trademark usage guidelines, as may be published on its website or otherwise made available upon request from time to time.

Creative Commons may be contacted at http://creativecommons.org/.

==========

Index