Gestion de version

Version 1

Table des matières

Gestion de versions

Si vous avez déjà travaillé à plusieurs sur un projet informatique, vous avez certainement déjà entendu ou prononcé une de ces phrases :

- « Qui a modifié le fichier X, il marchait hier et maintenant ça bogue un max ! » ;
- « Kevin, tu peux mettre à jour le fichier X pendant que je m'occupe du fichier Y ? Attention, ne touche pas à Y faute de quoi je risque d'écraser tes modifications !» ;
- « Qui a touché au fichier Z ? Il y a du nouveau code qui ne sert strictement à rien ici…;
- « C'est quoi tous ces nouveaux fichiers ? Qui les a ajoutés au projet ? » ;
- « Rappelez-moi, le bug de la page qui se rafraîchissait en boucle, on avait résolu ça comment déjà ?? »

Si tel est le cas, un logiciel de gestion de version est ce qu'il vous faut.
La gestion de versions (en anglais *version control*, *revision control* ou *versionning*) consiste à gérer et à organiser l'ensemble des versions d'un ou plusieurs fichiers. Essentiellement utilisée dans le domaine de la création de logiciels, elle concerne surtout la gestion des codes source : un logiciel de gestion de version est capable de suivre l'évolution d'un fichier texte ligne de code par ligne de code.
Il existe un grand nombre de logiciels de gestion de versions qui, tout en partant d'un concept et d'un but identiques, possèdent chacun leur propre vocabulaire et leurs propres usages :

Outil	Type	Description
CVS	Centralisé	Un des plus anciens. Bien qu'il fonctionne et soit encore utilisé pour certains projets, mieux vaut employer SVN qui corrige un certain nombre de ses défauts.
SVN (Subversion)	Centralisé	Probablement l'outil le plus utilisé à l'heure actuelle. Assez simple d'utilisation, bien qu'il nécessite comme tous les outils du même type un certain temps d'adaptation. Il a l'avantage d'être bien intégré à Windows avec le programme Tortoise SVN, là où beaucoup d'autres logiciels s'utilisent surtout en ligne de commande.
Mercurial	Distribué	Plus récent, complet et puissant. Apparu quelques jours après le début du développement de Git, il lui est comparable sur bien des aspects.
Bazaar	Distribué	Complet et récent, sponsorisé par Canonical, l'éditeur de Ubuntu. Il se focalise sur la facilité d'utilisation et la flexibilité.
Git	Distribué	Très puissant et récent, créé par Linus Torvalds. Il se distingue par sa rapidité et sa gestion des branches qui permettent de développer en parallèle de nouvelles fonctionnalités.

Les produits de ce tableau sont tous des logiciels libres, mais il existe aussi des logiciels propriétaires : Perforce, BitKeeper, Visual SourceSafe de Microsoft, etc.

Définitions

Systèmes centralisés et décentralisés

Gestion de versions centralisée

Dans le cas d'une gestion de versions centralisée (comme CVS et Subversion), il n'existe qu'un seul dépôt des versions : un serveur conserve les anciennes versions des fichiers et les développeurs s'y connectent pour prendre connaissance des fichiers qui ont été modifiés par d'autres personnes et y envoyer leurs modifications. Cela simplifie la gestion des versions mais est contraignant pour certains usages comme le travail sans connexion au réseau, ou tout simplement lors du travail sur des branches expérimentales ou contestées.

Gestion de versions décentralisée ou distribuée

La gestion de versions décentralisée consiste à voir l'outil de gestion de versions comme un outil permettant à chacun de travailler à son rythme, de façon désynchronisée des autres, puis d'offrir un moyen à ces développeurs de s'échanger leur travaux respectifs. Il n'y a pas de serveur, chacun possède l'historique de l'évolution de chacun des fichiers. Les développeurs se transmettent directement entre eux les modifications, à la façon du peer-to-peer. Autrement dit, il existe plusieurs dépôts pour un même logiciel.
La gestion décentralisée permet ainsi entre autres choses de ne pas dépendre d'une seule machine, autorise le travail des contributeurs sans connexion au gestionnaire de version, facilite la gestion des autorisations de soumissions, accélère la plupart des opérations (réalisées en local sans accès réseau) etc..
En pratique, les logiciels distribués sont rarement utilisés selon le modèle distribué pur. Très souvent, un serveur va constituer un point de rencontre entre les développeurs. Le serveur connaît l'historique des modifications et permet l'échange d'informations entre les développeurs, qui eux possèdent également l'historique des modifications.

Modifications et ensemble de modifications

En principe, tout logiciel évolue en permanence depuis sa création jusqu'à son abandon. Chaque lot de modifications donne lieu à une nouvelle version (ou révision). Les différentes versions sont nécessairement liées à travers ces modifications (diff ou patch) : un ensemble d'ajouts, de modifications, et de suppressions de données.
Schématiquement, on passe d'une version N à une version N + 1 en appliquant une modification M. Un logiciel de gestion de versions applique ou retire ces modifications une par une pour fournir la version voulue du fichier. Il sait qui a effectué chaque modification de chaque fichier et pourquoi, et donc capable de dire qui a écrit chaque ligne de code de chaque fichier et dans quel but. Si deux personnes travaillent simultanément sur un même fichier, il est capable de fusionner leurs modifications et d'éviter que le travail d'une de ces personnes ne soit écrasé.

Remarque : Pour un fichier, on préfère généralement le terme « révision » : le terme version s'applique plutôt à un logiciel, et plus exactement à sa distribution sous forme "finie", c'est-à-dire souvent binaire.

Une modification correspond donc à une évolution entre deux versions. La différence entre deux versions tient uniquement aux modifications ayant amené à la nouvelle version.
La gestion de versions s'applique le plus souvent à un ensemble de fichiers, constitutifs d'un projet. De toute évidence, une modification est appliquée à un seul fichier, alors que le projet subit un ensemble de modifications.

Pour illustrer, prenons l'exemple d'un logiciel nommé « Toto ». Il est constitué des fichiers A, B et C. À la version 1.0 de « Toto » correspondent les versions 1.0 de chacun des fichiers. Admettons que l'ajout d'une fonctionnalité à « Toto » impose la modification de A et de C. Présentons la situation à l'aide d'un tableau.

versions de « Toto »	versions de A	versions de B	versions de C
1.0	1.0	1.0	1.0
1.1	1.1		1.1

Du point de vue du projet, les modifications apportées à A et à C font partie du même ensemble.
Au sens de gestion de version, une modification du code source est un ou plusieurs changements qui permet soit de régler un bug, soit d'ajouter une fonctionnalité. Cela peut aussi bien correspondre à une ligne changée dans un fichier que 50 lignes changées dans un fichier A et 25 lignes dans le fichier C. Pour enregistrer une modification dans un logiciel de gestion de version, vous procédez à un commit : celui-ci représente donc généralement un ensemble de changements.

Dépôt et copies locales

Le but du logiciel de gestion de versions est de mettre les fichiers versionnés à disposition sur un espace de stockage facilement accessible, nommé dépôt.
Pour pouvoir effectuer des modifications, le développeur doit d'abord faire une copie locale des fichiers qu'il souhaite modifier, ou de tout le dépôt. Selon les systèmes de gestion de version, certains fichiers peuvent être verrouillés ou protégés en écriture pour tout le monde, ou pour certaines personnes.
Le développeur fait ces modifications et effectue ses premiers tests localement, indépendamment des modifications faites sur le dépôt du fait du travail simultané d'autres développeurs. Il doit ensuite faire un commit (une soumission), c'est-à-dire soumettre ses modifications, afin qu'elles soient enregistrées sur le dépôt. C'est là que peuvent apparaître des conflits entre ce que le développeur souhaite soumettre et les modifications effectuées depuis la dernière copie locale effectuée. Ces conflits doivent être résolus (merge) pour que le patch soit accepté sur le dépôt.

Branches

Lorsque des modifications divergentes interviennent hors conflit, il se crée des branches. Le fait de vouloir rassembler deux branches est une fusion de branches.
Les branches sont utilisées pour permettre :

- la maintenance d'anciennes versions du logiciel (sur les branches) tout en continuant le développement des futures versions (sur le tronc) ;
- le développement parallèle de plusieurs fonctionnalités volumineuses sans bloquer le travail quotidien sur les autres fonctionnalités.

Les correctifs de la dernière version doivent être faites sur le tronc principal (*trunk*).

Conflit de modifications

Dans le cas d'un développement en équipes, surtout si elles sont réparties dans le monde entier, il est nécessaire de partager une base commune de travail, et c'est tout l'intérêt des systèmes de gestion de version. Mais, il faut aussi veiller à coordonner les équipes de développement grâce à des outils de communication, un logiciel de suivi de problèmes, un générateur de documentation et/ou un logiciel de gestion de projets.

Il n'est pas rare que certaines modifications soient contradictoires (par exemple lorsque deux personnes ont apporté des modifications différentes à la même partie d'un fichier). On parle alors de conflit de modifications car le logiciel de gestion de versions n'est pas en mesure de savoir laquelle des deux modifications il faut appliquer.
Le contrôle de la concurrence, pour éviter ces conflits de modifications, est un problème classique en informatique : on le retrouve par exemple dans les systèmes de gestion de base de données ou en programmation système. Il peut être géré de deux manières différentes, qui ont toutes deux été appliquées à la gestion de version :

- Le contrôle de concurrence pessimiste impose à chaque utilisateur de demander un verrou avant de modifier une ressource ; ce verrou lui garantit qu'il sera le seul à modifier la ressource. Ce modèle s'impose quand on considère que le coût de résolution des conflits de modification (coût unitaire pondéré par leur probabilité d'occurrence) est plus important que celui de la gestion du verrou. En gestion de version, il correspond au modèle "verrouiller-modifier-déverrouiller" qui était utilisé par les systèmes les plus anciens. Il s'est avéré que la gestion manuelle des verrous par les utilisateurs n'était pas toujours satisfaisante : les outils de diff/merge s'améliorant, il est progressivement devenu moins pénalisant de corriger les conflits d'éditions simultanées que de traiter les problèmes de fichiers verrouillés en écriture.

- Le contrôle de concurrence optimiste permet à chaque utilisateur de modifier les données sans contrainte. Au moment d'appliquer ces modifications le système vérifie si un autre utilisateur n'a pas déjà posté des modifications pour ces mêmes données : il demande alors à l'utilisateur de résoudre le conflit avant de re-soumettre ses données. En gestion de configuration, c'est le modèle "copier-modifier-fusionner" qui a été popularisé par CVS. Il est devenu le paradigme de base des systèmes de gestion décentralisés.