Les VCS Distribués, Nouvelle Génération de Clients SVN

Michel Jouvin

Journées Informatiques IN2P3/IRFU
17 Mai 2010
Aussois

Agenda

- Pourquoi les VCS distribués
- Principales caractéristiques des VCS distribués
- Les VCS distribués et SVN
- Branches, merge et tags
- Git et Mercurial
- Git SVN
- 1 expérience : Quattor GRIF
- Intégration dans les environnements de développement
- Conclusions

Disclaimer

- Ce n’est pas une présentation sur Git ou Mercurial
  - Ma principale expérience VCS distribué est Git
  - Ma connaissance de Mercurial limitée à l’évaluation initiale que j’en avais fait début 2010
  - Une grande partie de cette présentation pourrait sans doute s’appliquer à d’autres VCS distribués
- La mention de fonctionnalités Git ne signifie pas nécessairement qu’elles n’existent pas dans Mercurial…
- Quelques acronymes…
  - VCS : Version Control System
  - DVCS : VCS distribué
  - WC : working copy (espace de travail local)

VCS Centralisés

- Le canal historique du versioning …
- 2 implémentations (open-source) majeures
  - CVS : le père
    - Limitations nombreuses : ne gère que les fichiers, pas de commit historique, gestion complexes des branches, peu de clients
  - Subversion (SVN) : commandes proches de CVS pour un outil totalement différent
    - + : tous les défauts de CVS… + protocole client/server
    - - : confusion namespace/branches/tags
- Un repository unique + 1 working copy par contributeur
  - Synchronisation bidirectionnelle (commit + update)
  - 1 working copy ne contient qu'une révision du repository
    - Impossible de parcourir l'historique si déconnecté
    - SVN contient des méta-données qui permettent de garder l'image du repository même en cas de modification
    - Directory .svn rend difficile l'utilisation de grep, find…

VCS Distribués, Pourquoi ?

- Modèle centralisé inadéquat aux grands projets open-source
  - Trop lourd pour permettre de gérer de grands nombres de contributeurs occasionnels
  - 2 principaux DVCS (Git et Mercurial) issus des besoins pour le développement du kernel Linux
- Absence de l'historique dans la *working copy* est une limitation importante au travail « déconnecté »
  - Difficulté pour les situations de mobilité
- Absence de versioning pour le développement local
  - Conduit à encombrer l'historique central des étapes intermédiaires du développement
- Difficulté de travailler sur plusieurs choses à la fois dans le même projet
  - Ou multiplication des working copies
- Lourdeur et manque de flexibilité des merges

VCS Distribués

- Pas de repository central mais « coopération » entre des repositories personnels
  - Coopération volontaire et bidirectionnelle (clone/push/pull)
    - Distinction workflow personnel et les contributions exportées
  - 1 projet peut décider qu'un des repositories est la référence centrale
- 1 seule WC associée à chaque repository « personnel »
  - Pas possible (ou difficile) d'avoir plusieurs working copies d'un même repository, sur plusieurs machines par ex.
- Gestion très simple des branches et des merges
  - Branches distinctes du namespace
  - Comparé à SVN : merge très rapide, entièrement local
- Repository très compact (< working copy SVN)
- Pas de gestion des directories en tant que tel
  - 1 directory n'existe que s'il contient quelque chose

VCS Distribués et SVN

- Les VCS distribués peuvent s'utiliser comme un client SVN
  - Le repository SVN est utilisé comme le repository de référence cloné par le repository du DVCS
    - Remplace le working copy SVN : pas d'utilisation possible des commandes SVN
  - Généralement aussi possible d'importer des projets CVS
    - Interaction bidirectionnelle généralement pas disponible
      - Pour certains DVCS, aussi possible avec d'autres DVCS
- Le repository DVCS contient tout l'historique SVN
  - Taille DVCS + working copy < working copy SVN
- Certaines fonctionnalités des DVCS limitées par SVN
  - Gestion des branches et du merge : merge tracking récent dans SVN
  - Linéarité de l'historique
- Et vice-versa : pas de support direct des externals et des properties

DVCS Repositories et WC

- Différence avec SVN : pas de possibilité de checkout d'un sous ensemble du repository
  - Branches distinctes du namespace (directories)
  - Toujours le checkout d'une branche particulière
- Eviter un « super repository » : plutôt un repository par sous ensemble d'un « gros » projet
  - Possibilité de références inter-repository dans la plupart des DVCS (sub-repositories/submodules)
- WC ne contient que le code du projet
  - Pas de metadata (.svn) dans les directories
- Par défaut, le repository est stocké dans un répertoire se trouvant à la racine du WC
  - .git pour Git, .hg pour Mercurial

Branches, Merge et Tags…

- Branches distinctes du namespace
  - Retour au modèle CVS…
  - Beaucoup plus de possibilités, merge plus simple
  - 1 *head* principal par branche
    - Git: aussi un *fetch head* et un *merge head*
  - Checkout du repository sans copies multiples à cause des branches contenues dans le namespace (SVN)
  - Créer des branches n'a pas d'impact sur la taille du repository
- Merge est une opération simple et peu couteuse
  - Fusion d'historique et non de révisions
    - Chaque révision ajoutée dans la branche + 1 révision merge (Git)
  - Merge est extrêmement rapide
  - cherry-pick : permet le merge d'une sélection de révisions
    - Géré par le merge tracking en cas de merge ultérieur
  - Git implémente plusieurs stratégies de merge pour différents contexte

...Branches, Merge et Tags

- Conflits
  - Pas de tentative de résoudre les conflits par des algorithmes évolués (mais couteux)
    - Marquage des conflits avec les marqueurs CVS/SVN
    - Accès 3 versions classiques du fichier : parent, version courante dans le repository/branche local, version causant le conflit
  - Possibilité d'utiliser des outils externes pour la résolution de conflit (kdiff3, vimdiff, tortoisemerge...)

- Tags : nom symbolique attaché à une révision dans une branche
  - Peut être utilisé en remplacement d'une révision dans toutes les commandes

Git et Mercurial...

- Tous 2 issus des besoins pour le kernel Linux
  - Même concepts mais implémentation/interface très différents
    - Git = Linus Torvald...
- Mercurial (Hg)
  - 1 module (presque pure) Python
  - Commandes/vocabulaire assez proches de SVN (cmd=hg)
    - Sélection explicite et permanente des fichiers gérés par Hg
    - Manipulation de fichiers (cp, mv, rm...) via la commande hg
  - Gestions des branches déroutantes
    - Une branche n'existe que si elle contient des révisions
    - Suppression (difficile) d'une révision référencée par plusieurs branches la supprime de toutes les branches (pas de suppression par branche)
  - Difficile de réécrire l'historique
    - Module rebase maintenant disponible
    - Module additionnel (Queues) relativement lourd à employer
  - Numérotation relative des révisons (à la SVN)

... Git et Mercurial

- Git
  - Vocabulaire et commandes très différents de SVN
    - Parfois inconsistant avec Hg (ex: pull vs. fetch)
  - Des dizaines de commandes : *plomberie* et *porcelaine*
    - Porcelaine = interface utilisateur, généralement en Perl (cmd=git)
    - Plomberie = gestion du repository, généralement C
  - Pas de numérotation relative des révisions
    - Possibilité d'utiliser numéro abrégé tant que non ambigu
  - Pas de sélection permanente des fichiers gérés par Git
    - Doit faire un add des différences à enregistrer à chaque commit
    - Content-based difference : 1 différence enregistrée une fois, éventuellement référencée par plusieurs fichiers
  - Possibilité d'inclure des patches reçus par mail
  - Réécriture possible de l'historique d'une branche
    - *Amend* du dernier commit
    - *Rebase* : compactage de révisions (squash), suppression de révision
  - Suggestions des actions à faire, mode interactif...

Git/Hg et Accès Distant

- Modèle distribué = accès distant requis…
- … mais pas d'ACL sur le namespace
  - Même accès pour tous le repository
  - Droit d'écriture lié aux permissions des répertoires/fichiers du repository
    - Pas de partage des working copies
- Mercurial : mode serveur http
  - Très simple à mettre en œuvre
- Git : principal protocole est ssh
  - Possibilité d'utiliser des clés ou tout autre méthode supportée par ssh
  - Aussi un mode http mais peu utilisé et aucune expérience
    - git http-push / git http-fetch
  - Echange de patch entre repositories par mail
    - git format-patch / git am

Git SVN

- *git svn* : commande permettant d'utiliser Git comme un client SVN
  - Equivalent Hg: hg subversion (module optionel)
- Les principales commandes git-svn
  - clone (init + fetch) : svn checkout
  - rebase : svn update
    - Utilise l'édition d'historique pour remettre les révisions SVN au bon endroit dans l'historique
    - Fait un *rebase* des modifications locales survenues depuis le dernier rebase
    - Peut occasionner des conflits
  - dcommit : enregistrement des commits Git dans SVN
    - Chaque commit Git génère 1 commit SVN
    - Le message du SVN commit **est** celui du commit Git
    - Implique un rebase car pas de possibilité de réécrire l'historique SVN
- Permet de récupérer la structure trunk/branches/tags
  - Jamais réussi à utiliser avec succès...

Remarques sur le Rebase

- *rebase* est un outil extrêmement précieux pour réagencer un historique avant son exportation…
  - En particulier, dans Git, le rebase interactif : git rebase –i
  - Aussi *git merge --squash*
- Mais doit être utilisé à bon escient
  - Règle d'or : ne **jamais** faire de rebase de qqc qui a été exporté sinon génèrera des conflits lors des merges ultérieurs
    - Seule résolution possible : appliquer le même rebase aux autres repositories
- Le rebase n'est pas une alternative au merge
  - Merge créée une nouvelle révision qui peut donc être propagée ultérieurement
    - Utiliser *git revert* pour créer un « merge inverse »
- Particulièrement vrai pour git svn puisqu'il est **impossible** de réécrire un historique SVN

Expérience : Quattor GRIF

- Quattor : gestion de configuration avec descriptions versionnées dans SVN
  - GRIF: 7300 fichiers, 60 MB
  - SVN : WC=190 MB, 35K révisions, repository=1,5 GB
    - Dans SVN impossible de supprimer des révisions
  - Impossible de migrer à Git ou autre DVCS
- Utilisation de Git comme client SVN
  - Taille repository + WC = 130 MB
    - Contient toutes les révisions SVN...
  - 1 branche principale pour la synchronisation avec SVN et les développements quotidiens
  - Des branches pour les développements au « long cours »
  - Principalement Windows : Git + Eclipse/EGit
- Git transparent pour les autres utilisateurs
  - Seuls les commits exportés sont visibles : identique à un commit SVN

Expérience : Quattor GRIF

- Clonage SVN et update (rebase) relativement long
  - Lié au fait de récupérer toutes les révisions
  - Probablement le principal inconvénient de la solution DVCS
  - Pour le clonage initial, maintenir un clone de référence
    - Voir *git help svn*
- Commit message des merges
  - Par défaut « Merge branch... » : pas très parlant, -m ou --log
- Eclipse : mauvaise gestion des symlinks
- Besoin de gérer les SVN externals : utilisation d'une solution basée sur les submodules
  - 1 remote-tracking branch par SVN external
  - Clone du repository Git sur lui-même dans un submodule avec partage des metadata entre repository et submodules
    - Clone ~= 0 byte
  - Update des externals via Git fetch/pull
    - Pb sur Windows du fait de l'absence de symlink

Git: Autres Fonctionnalités

- Stash : possibilité de *mettre de coté* des modifications en cours sans en faire de commit
  - Les *stash* sont empilés : possibilité d'en faire plusieurs
  - Possibilité de les lister et de les réintégrer dans le working copy ou de les détruire
  - Commande : git (help) stash
- Remote-tracking branches : l'association avec un autre repository se fait par la création d'une branche spécifique
  - Possible d'en faire un checkout mais pas de commit
  - Import des modifications en 2 opérations : *fetch* + *merge*
    - Possible de les faire séparément ou en une fois (*pull*)
  - *git log* permet de visualiser les divergences (avec .. ou ...)
- Git : spécification relative des révisions/trees
  - Beaucoup de possibilité : *git help rev-parse*

Intégration dans les IDE

- DVCS: pas vraiment de distinction client/serveur contrairement à SVN
  - Modèle fondamentalement server-less
- Mercurial : uniquement une API Python
  - Pour les autres environnements, utilisation de wrappers
    - Exemple : Eclipse
  - Intégration simple dans les outils Python
    - Exemple : Trac
- Git : une API pour (presque) tous les langages
  - Sauf Java...
  - Intégration simple dans la plupart des environnements... sauf Eclipse
    - Exemple : plugin Trac
  - En cours : Egit/Jgit pour le support Eclipse
    - Jgit : implémentation Java de Git (et non pas une API)
  - GUI Tk partie intégrante de Git : gitk
    - Intéressant pour visualiser des historiques complexes

Conclusions

- DVCS : fonctionnalités attractives mais utilisation plus complexe que SVN
  - Intéressant de pouvoir mixer avec SVN pour permettre une plus large participation
- Il est possible de bénéficier de ces fonctionnalités sans que tout le projet utilise un DVCS
  - Le DVCS remplace le client SVN : transparent pour le reste du projet
  - Toujours possible d'avoir une working copy SVN à coté du DVCS clonant le repository SVN…
- Utiliser un DVCS en remplacement d'un client SVN demande un effort initial significatif
  - Malgré les ressemblances, beaucoup de différences subtiles
  - Prendre le temps de comprendre le vocabulaire et ne pas se laisser piéger par les faux amis…
- Ne pas sous estimer la réécriture d'historique…
  - Aide précieuse pour la traçabilité à long terme

Liens Utiles...

- Git
  - Site Git : http://git-scm.com
  - Documentation en ligne très complète (version html)
    - Tutorial : *git help tutorial*
  - SVN crash course : http://git.or.cz/course/svn.html
  - Git SVN man page : *git help svn*
  - Externals: http://kerneltrap.org/mailarchive/git/2007/5/1/245002
  - Page Quattor contenant un exemple et des références
    - https://trac.lal.in2p3.fr/Quattor/wiki/Doc/SCDB/Git
- Mercurial
  - Site principal : http://mercurial.selenic.com
  - Hg/SVN : http://mercurial.selenic.com/wiki/WorkingWithSubversion