Un aperçu de la gestion de version décentralisée avec GNU Arch

Ludovic Courtès <*ludovic.courtes@laas.fr*>

Toulibre, 10 janvier 2007

De quoi allons nous parler ?

Le sujet

- gestion de **versions** (à la CVS, Subversion, etc.)
- **décentralisée** (à la GNU Arch, GIT, Darcs, etc.)

Qui peut intéresser cette présentation ?

- un utilisateur d'outils **centralisés** (e.g., CVS, Subversion)
- n'importe qui ayant besoin de **gestion de version**, d'**archivage**, d'outils pour le **travail en groupe**, etc.
- un *geek* cherchant un moyen de **frimer**

- **Introduction à la gestion de version décentralisée, motivations**
  - Gestion de versions centralisée et décentralisée
  - Premier problème : travail « hors-ligne »
  - Deuxième problème : droits d'accès au dépôt
  - Troisième problème : coopération, travail en équipe
  - Pourquoi décentraliser ? Résumé des motivations
- Introduction à GNU Arch
- Utilisation basique de GNU Arch
- Coopération avec GNU Arch
- Repoussons les limites !
- Critique et discussion de la concurrence

Gestion de versions centralisée et décentralisée

Centralisée ?

- **un dépôt central** contenant toutes les données archivées
- impossibilité d'**échanger les données entre dépôts**
- exemples : CVS, Subversion, PRCS, etc.
- problème : utilisateurs **souvent contraints de « faire sans »**

Décentralisée ?

- utilisation d'un **réseau de dépôts**, tous équivalents
- approche « **pair-à-pair** »
- facilité d'**échange des données entre dépôts**
- s'abstrait de la **frontière géographique**

Premier problème : travail « hors-ligne »

Exemples

- travailler sur un projet **depuis son portable**
- **sans avoir accès au réseau**

Comment faire en centralisé ?

- **impossible d'accéder au dépôt** (ex. : serveur CVS/Subversion)
- donc impossible d'**enregistrer des modifications**
- on se retrouve à travailler **sans gestion de versions**
- ... avec les risques que ça comporte

Deuxième problème : droits d'accès au dépôt

En gestion de versions centralisée...

- mécanismes de **gestion des droits d'accès** *ad hoc* (ex. : CVS)
- un seul dépôt, donc **point singulier de défaillance**
- souvent : **limitation stricte des personnes ayant droit d'accès** (ex. : projet libre)

Conséquences pratiques non désirables

- gestion des droits d'accès **spécifique, non triviale**
- utilisateur **sans accès en écriture = utilisateur de seconde classe**
- on se retrouve à travailler **sans gestion de versions**...

Troisième problème : coopération, travail en équipe

Exemple : développement de logiciel (libre)

- Alice veut travailler sur la fonctionnalité F_0 et se moque du reste
- Bob veut travailler sur F_1 mais n'a **pas accès en écriture** au dépôt

Problèmes

- Bob doit constamment **mettre à jour sa copie locale** (`update`)
- Bob est donc **perturbé** par les changements relatifs à F_0

Pourquoi décentraliser ? Résumé des motivations

Techniquement...

- s'abstraire de la **séparation physique entre dépôts**
- fournir une **solution générale**
- faciliter **la création de branches**
- faciliter **l'intégration de changements faits dans d'autres branches**

« Philosophiquement »...

- faciliter la **coopération sur des projets libres**
- permettre réellement un **modèle de développement pair-à-pair**

- Introduction à la gestion de version décentralisée, motivations
- **Introduction à GNU Arch**
  - Remarques préliminaires
  - Historique rapide
  - Concepts de base, terminologie
  - Désignation
  - Désignation des révisions
- Utilisation basique de GNU Arch
- Coopération avec GNU Arch
- Repoussons les limites !
- Critique et discussion de la concurrence

Remarques préliminaires

J'entends dire...

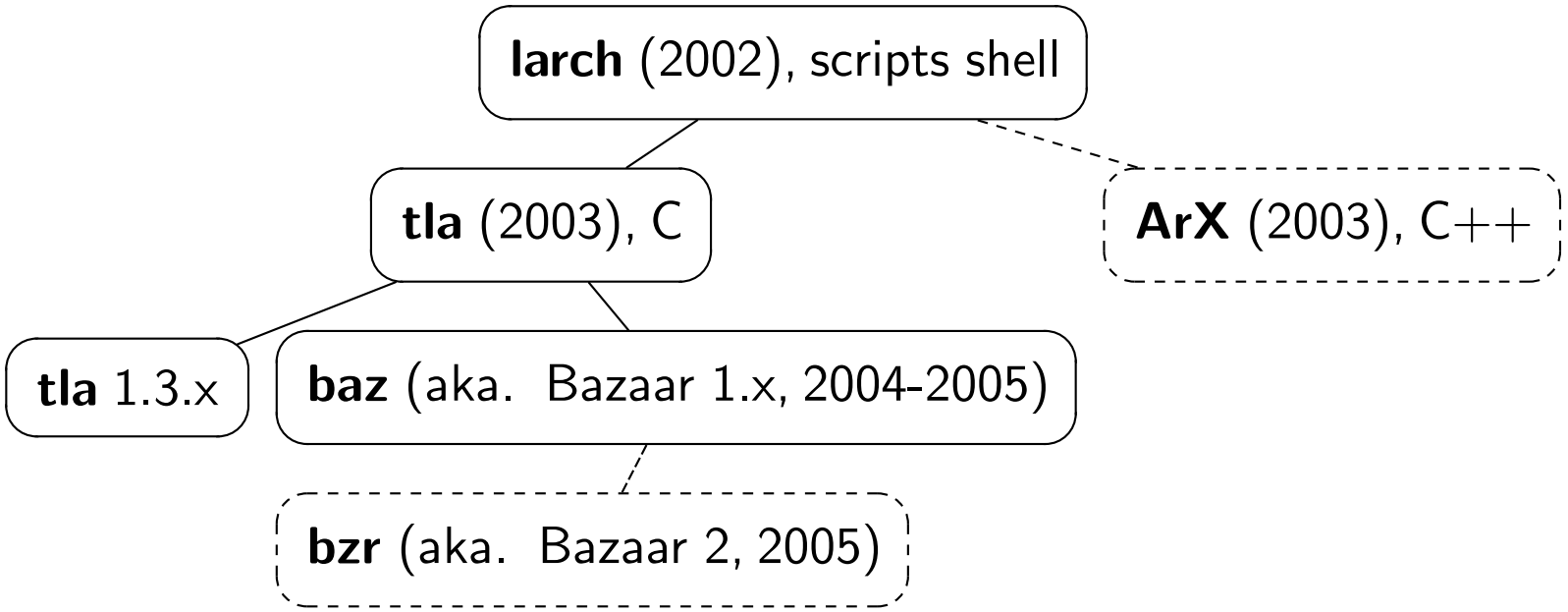
- c'est bizarre
- c'est „`vraiment ++bizar--re`
- c'est lent
- c'est intrusif
- c'est passé de mode
- mais pourquoi pas plutôt Bazaar/Darcs/GIT/Mercurial/Monotone/... ?

Parce que :

- c'est celui que j'utilise ;-)
- ça marche
- c'est robuste et mûr
- dispose de fonctionnalités non disponibles ailleurs

Historique rapide

- **premier outil (libre)** de gestion de révisions décentralisé
- initialement écrit par Tom Lord
- débuté en 2002
- a influencé d'autres outils : Darcs, Bazaar, Aegis, etc.



Concepts de base, terminologie

archive (= dépôt)
: endroit où sont stockées les différentes versions des données

branche
: identification **logique** d'un ensemble de travaux cohérent :

- « la branche de Linus T. »
- « la branche 2.6 du noyau »
- « la branche de développement de la fonctionnalité F »

révision
: contenu d'une branche tel qu'admis (*committed*) par l'utilisateur à un moment précis.

Désignation

Attention, ne plaît pas à tout le monde ! ;-)

archive/dépôt

- objectif : **nom globalement unique** sur Internet (autant que possible)
- format (et convention) : `ludo@chbouib.org--bidule-2007`

branche

- identifiant divisé en **trois éléments** : « catégorie », « branche » et « version »
- format : `categorie--branche--X.Y`
- exemples : `linux--torvalds--2.6`, `linux--andrew--2.2`, `emacs--devel--2007`
- nom complet, globalement unique : `linus@osdl.org/linux--torvalds--2.6`

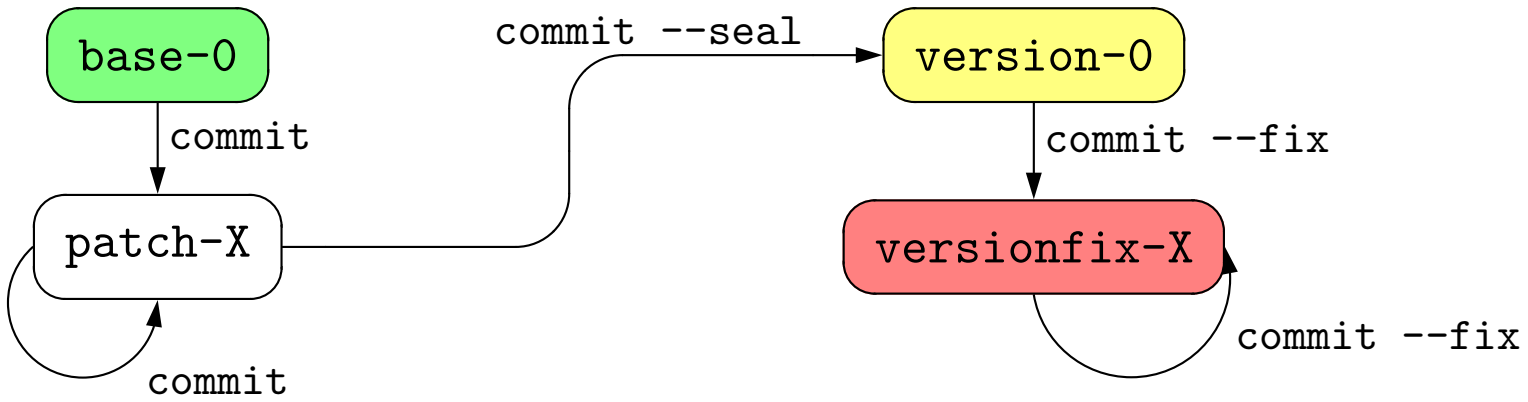
Désignation des révisions

Les bases

- la première (`import`) : `base-0`
- les suivantes : `patch-1`, `patch-2`, etc.
- nom complet : `l@c-bidule/c--b--v--patch-5`

Cycle de vie d'une branche

- possibilité de **sceller une branche**



- Introduction à la gestion de version décentralisée, motivations
- Introduction à GNU Arch
- **Utilisation basique de GNU Arch**
  - Commencer avec GNU Arch (`tla`)
  - Inventaire des fichiers
  - Inventaire : avantages et inconvénients
  - Bon, et comment on « rajoute » un fichier ?
  - Et ensuite ?
- Coopération avec GNU Arch
- Repoussons les limites !
- Critique et discussion de la concurrence

Commencer avec GNU Arch (`tla`)

Création d'une archive

- protocoles possibles : système de fichiers, FTP, SFTP, HTTP
- commande : `make-archive`

Accès à une archive existante

- commande : `register-archive`
- établit la **correspondance entre le nom et l'URL**

Création d'une branche

- nouvelle branche : `archive-setup`
- à partir d'une branche existante : `tag -S`

Inventaire des fichiers

Catégorie	Effaçable ?	Archivé ?
`junk`, `backup`	oui	non
`precious`	non	non
`source`	non	oui
`unrecognized`	?	

Classification des fichiers par nom

- à partir de **regexps**
- **spécifié par branche**, dans `{arch}/=tagging-method`

Inventaire : avantages et inconvénients

Inconvénients

- **intrusif**
- parfois **redondant** avec un Makefile

Intérêts

- plus **précis/complet** qu'un `.cvsignore`
- pratique **pour l'import** de nombreux fichiers
- évite **d'oublier des fichiers**
- ... ou d'en rajouter (commande `lint`)

Bon, et comment on « rajoute » un fichier ?

Comme souvent, il y a plusieurs façons...

Principe : chaque fichier a un identifiant

- quand le fichier est **renommé**, l'identifiant **ne change pas**
- permet de **suivre les renommages**

La méthode `explicit`

- **comme avec CVS, SVN, etc.** : `tla add` *`fichier`*

La méthode `tagline`

- on peut encore utiliser `tla add`
- ou **ajouter une ligne** `arch-tag` **dans le(s) fichier(s)**

Et ensuite ?

La suite est habituelle...

- `tla commit`
- `tla changes`
- `tla diff`
- `tla undo`
- `tla redo`
- `tla revisions`
- `tla changelog`
- ...

- Introduction à la gestion de version décentralisée, motivations
- Introduction à GNU Arch
- Utilisation basique de GNU Arch
- **Coopération avec GNU Arch**
  - Voir ce qu'ont fait les autres
  - Fusion et intégration de modifications
  - Bonnes pratiques pour faciliter la coopération
  - Intégrité et authenticité
  - Coopération en pratique dans le libre
- Repoussons les limites !
- Critique et discussion de la concurrence

Voir ce qu'ont fait les autres

- **lister les révisions** : `revisions`
- lister **les révisions qui nous manquent** : `missing`

Fusion et intégration de modifications

Intégration d'un changement

- *cherry-picking* : on choisit spécifiquement des modifications à « rejouer »
- commande `replay`
- toujours **indépendamment de la localisation de l'autre branche**

Fusion de changements

- avec `star-merge`
- applique **tous les changements faits depuis la dernière fusion** dans l'autre branche
- conserve les **changements locaux**
- **voir les fusions** : commande `merges` ou interface ArchWay

Bonnes pratiques pour faciliter la coopération

- objectif : **faciliter la fusion entre développeurs**
- **créer des branches** pour identifier le travail sur une fonctionnalité, etc.
- enregistrer des **ensembles de changements cohérents** (*clean changesets*)
- les modifications introduites par une révision **doivent toutes avoir un rapport**
- ne pas **fusionner *et* modifier** en même temps

Intégrité et authenticité

Arch garantit l'intégrité de chaque révision

- avec des sommes MD5/SHA1 stockées dans l'archive

Garantir l'authenticité

- permet de **certifier l'origine d'une modification**
- avec des **signatures GPG**
- **signature d'un changement**
- ou **signature d'une révision complète**

Coopération en pratique dans le libre

Par envoi de demandes de fusions

- chacun travaille **séparément**
- lorsqu'on est prêt, **envoi d'une demande de fusion** au mainteneur/auteur (*merge request*)
- inconvénient : **impossible d'envoyer les changements par courriel** (le *diff*)

En utilisant un « gestionnaire de file de modifications » (PQM)

- le PQM **lit les courriels** qui lui sont adressés
- lorsqu'on est prêt, **envoi d'une demande de fusion au PQM**
- le PQM **intègre la fusion dans sa branche**...
- ... **sauf** si il y a conflit, échec des suites de tests, etc.

- Introduction à la gestion de version décentralisée, motivations
- Introduction à GNU Arch
- Utilisation basique de GNU Arch
- Coopération avec GNU Arch
- **Repoussons les limites !**
  - Création de miroirs d'archives
  - Mécanisme des « configurations »
  - Exemple de configuration
  - Mais c'est un peu lent tout ça !
  - Un cache dans l'archive
- Critique et discussion de la concurrence

Création de miroirs d'archives

- **mécanisme intégré** à `tla`
- création de **miroirs *push* ou *pull*** (commande `make-archive`)
- **mise à jour des miroirs** en une commande : `archive-mirror`

Mécanisme des « configurations »

Les configurations

- exprime **les dépendances** (d'un logiciel) **de manière précise** (branche voire révision)
- précise **le répertoire où stocker chaque dépendance**

Utilisation

- projet **séparé en plusieurs branches**
- sous-projets désignés par une **configuration dans le projet principal**
- `build-config` récupère les sous-projets et les met **dans les bons répertoires**

Exemple de configuration

```
# Une configuration du noyau Linux…

# répertoire     <-->     branche à stocker
arch/i386                    l@c/linux--i386--2.6
arch/sparc                   l@c/linux--sparc--2.6
drivers                      l@c/drivers--main--2.6
```

Mais c'est un peu lent tout ça !

Problème de performance

- révision obtenue **par application des modifications** depuis `base-0`
- si en plus les révisions sont **distantes**...

Solution : créer un cache de révisions

- révisions **récemment utilisées disponibles instantanément**
- commandes `my-library`, `library-config`

Un cache dans l'archive

Problème : performance du premier `get`

- obtention d'une branche **pour la première fois**
- besoin d'**appliquer beaucoup de changements**, donc lent

Solution : un cache dans l'archive

- `cacherev` stocke **une révision complète** dans l'archive
- un `get` devient **rapide**

- Introduction à la gestion de version décentralisée, motivations
- Introduction à GNU Arch
- Utilisation basique de GNU Arch
- Coopération avec GNU Arch
- Repoussons les limites !
- **Critique et discussion de la concurrence**
  - Critique de GNU Arch
  - Et la concurrence ?
  - Les avantages des « nouveaux »
  - Les soucis des « nouveaux »

Critique de GNU Arch

Problèmes

- **performance**
- difficilement utilisable **avec des très gros projets** (ex. : Linux)
- relative **inefficacité du stockage**

Avantages

- **fonctionnement transparent**
- **format d'archive transparent**, basé sur des simples fichiers
- simple à **déployer**
- **puissant**, robuste
- en un mot : **cool**

Et la concurrence ?

- Bazaar 2.x (bzr)
- Darcs
- GIT/Cogito
- Mercurial

Les avantages des « nouveaux »

(comparaison grossière)

- **meilleures performances** (sauf Darcs (?))
- **stockage plus efficace** (sauf Darcs (?))
- **plus de notion d'archive/dépôt** : un répertoire est un dépôt
- **facilité de prise en main**, interactivité (surtout Darcs)

Les soucis des « nouveaux »

(comparaison grossière, voire partiale…)

- **désignation des objets** incompréhensible à l'être humain : `aed90ea99f…` (sauf Darcs)
- conséquence : **format d'archive** « opaque », inaccessible
- peu ou pas de **gestion des renommages** (sauf Darcs et bzr)
- **protocoles réseaux spécifiques** ou pas toujours disponibles (ex. : `rsync`)

Conclusion

- **GNU Arch, c'est bien.**
- (mais ça pourrait être encore mieux) ;-)

Fin !

Questions ?

http://www.gnu.org/software/gnu-arch/

https://gna.org/projects/xtla-el

http://www.nongnu.org/archway/

http://better-scm.berlios.de/