Cours environnement de développement libre

Cogito, un système de gestion de versions décentralisé

Gauthier Quesnel
quesnel@lil.univ-littoral.fr

Laboratoire d'Informatique du Littoral

11 octobre 2006

Au sujet des gestionnaires de versions :

> *« Je ne peux pas imaginer programmer sans. . . Ce serait comme faire du parachutisme sans parachute. »*
>
> Brian Fitzpatrick

1. **Gestion de versions - principe de base**
   - diff & patch
   - Gestionnaire de versions
   - Centralisé, décentralisé
2. **Cogito**
   - Les bases
   - Les commandes
   - Exemples d'utilisation
3. **Git**
   - Caractéristiques
   - Les objets
   - Commandes
4. **Conclusion**

Plan

Plan

diff & patch

Les gestionnaires de versions s'appuient sur deux principes : les différences et l'opération d'appliquer une différence sur un fichier :

- diff : Comparaison de fichiers ligne par ligne
  - indique les lignes ajoutées ou supprimées ;
  - peut ignorer les casses, les tabulations, les espaces ;
  - option -u pour créer des patchs unifiés, avec plus d'informations.
- patch : Utilise la différence entre deux fichiers pour passer d'une version à l'autre.

Exemple

```
$ diff toto.c toto-orig.c > correction.patch
$ bzip2 correction.patch
$ bzcat correction.patch.bz2 | patch -p 0 toto.c
```

diff & patch

Les diff peuvent comparer des hiérarchies de fichiers (option -r) :

Exemple

```
$ cp -r linux-2.6.17 linux-2.6.17-orig
$ cd linux-2.6.17
$ vim [...]
$ cd ..
$ diff -r -u linux-2.6.17-orig linux-2.6.17 > \
      network-driver-b44.patch
$ gzip -9 network-driver-b44.patch

$ cd linux-2.6.17
$ zcat network-driver-b44.patch.gz | patch -p 1
```

L'option -p permet de laisser les chemins du patch au complet 0, en supprimant le premier dossier 1, etc.

Plan

Principe de base

Un gestionnaire de version, à partir des patchs, permet de :

- conserver toutes les versions de tous les fichiers ;
- conserver toutes les arborescences de fichiers ;
- permettre d'identifier une arborescence de version de fichiers ;
- fournir des outils pour gérer le tout.

Commandes générales

Les intitulés des commandes génériques de tous les gestionnaires de versions :

add : ajouter un fichier de texte de préférence pour facilement générer des patchs même si des diff de binaire sont possibles : `rdiff` ;

remove : supprimer un fichier mais pas son historique ;

log : historique d'un fichier ou d'une arborescence ;

diff : différence entre deux versions de fichier ;

commit : envoyer un patch au gestionnaire de versions ;

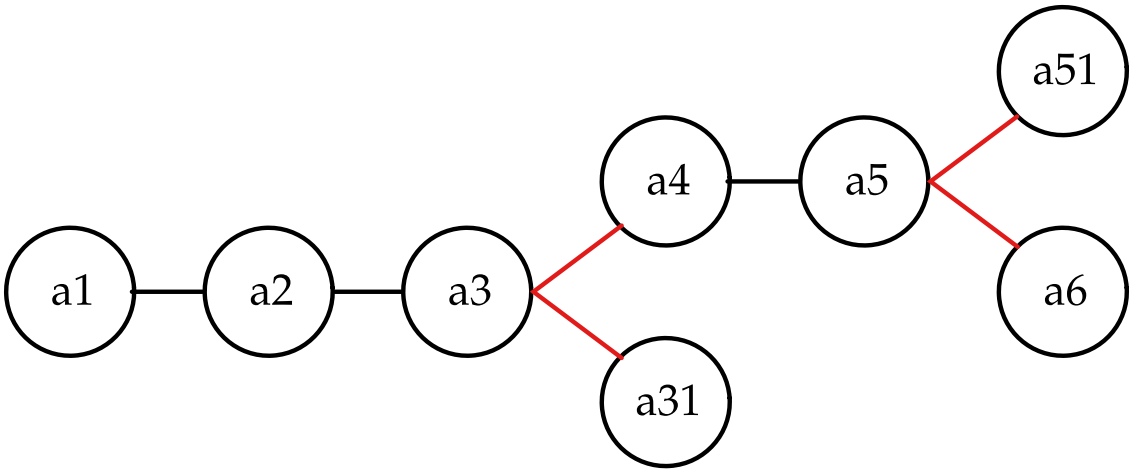
tag : poser un label sur un ensemble de version de fichiers ;

Branches

Définitions

Quelques définitions :

- un tronc représente la version centrale du développement ;
- une branche, une bifurcation du tronc ou d'une branche.

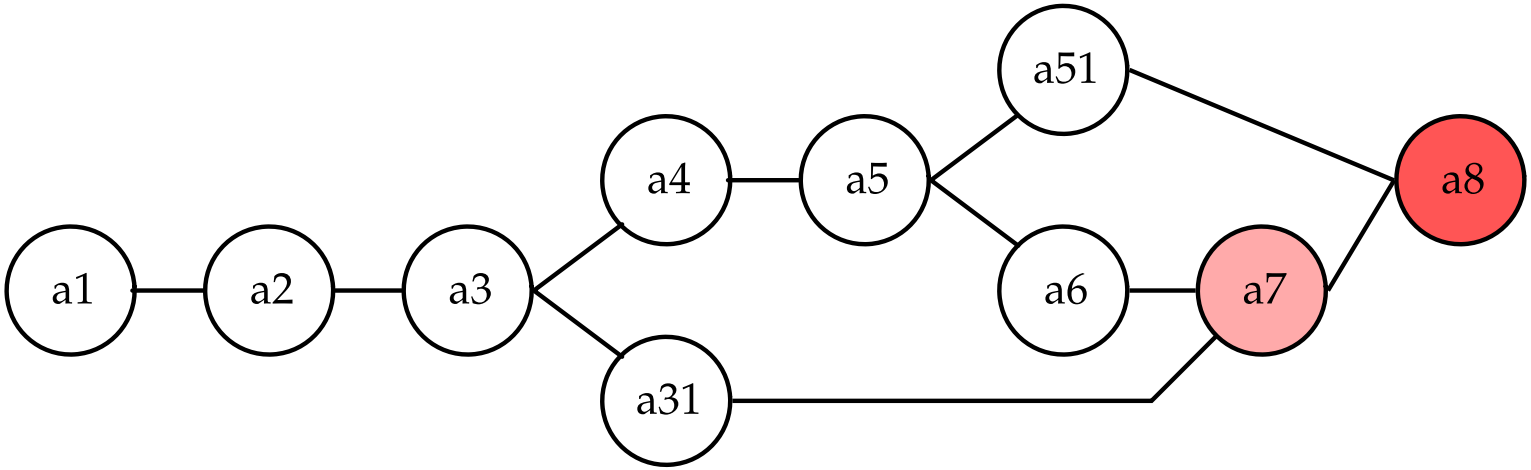


Branches

Fusion ou merge de branches

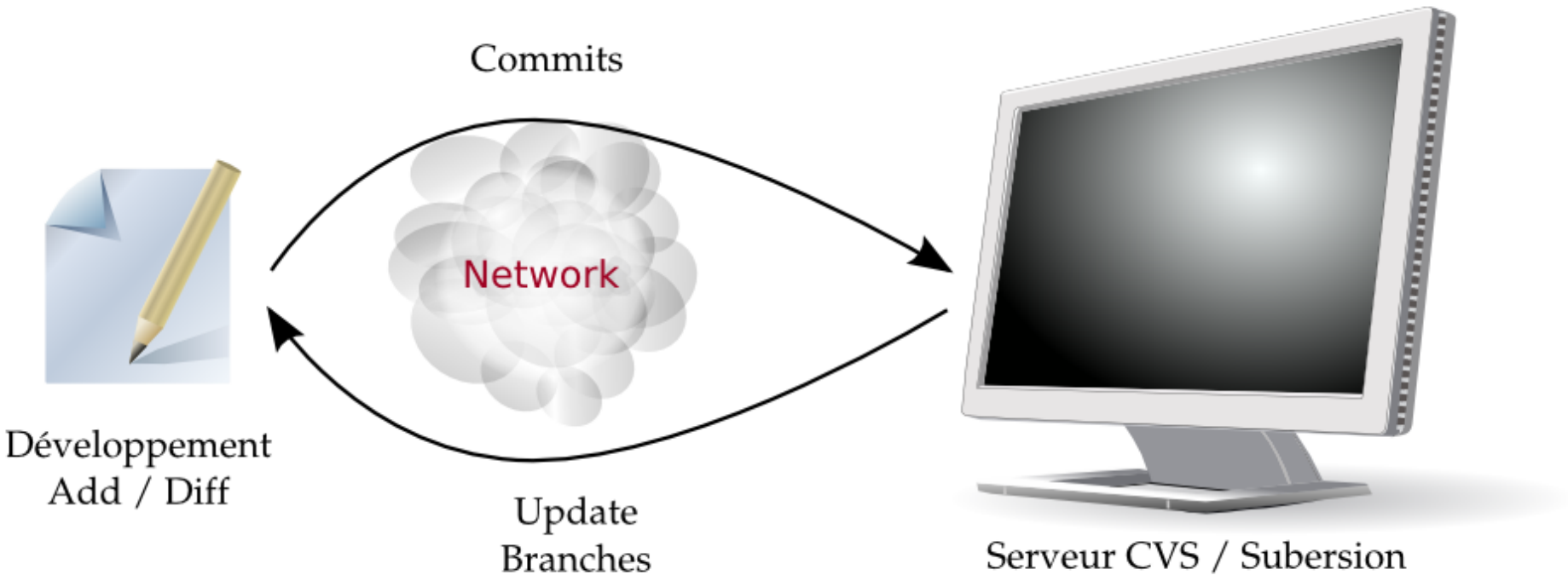
Quelques définitions :

- fusionner des patchs de plusieurs branches.

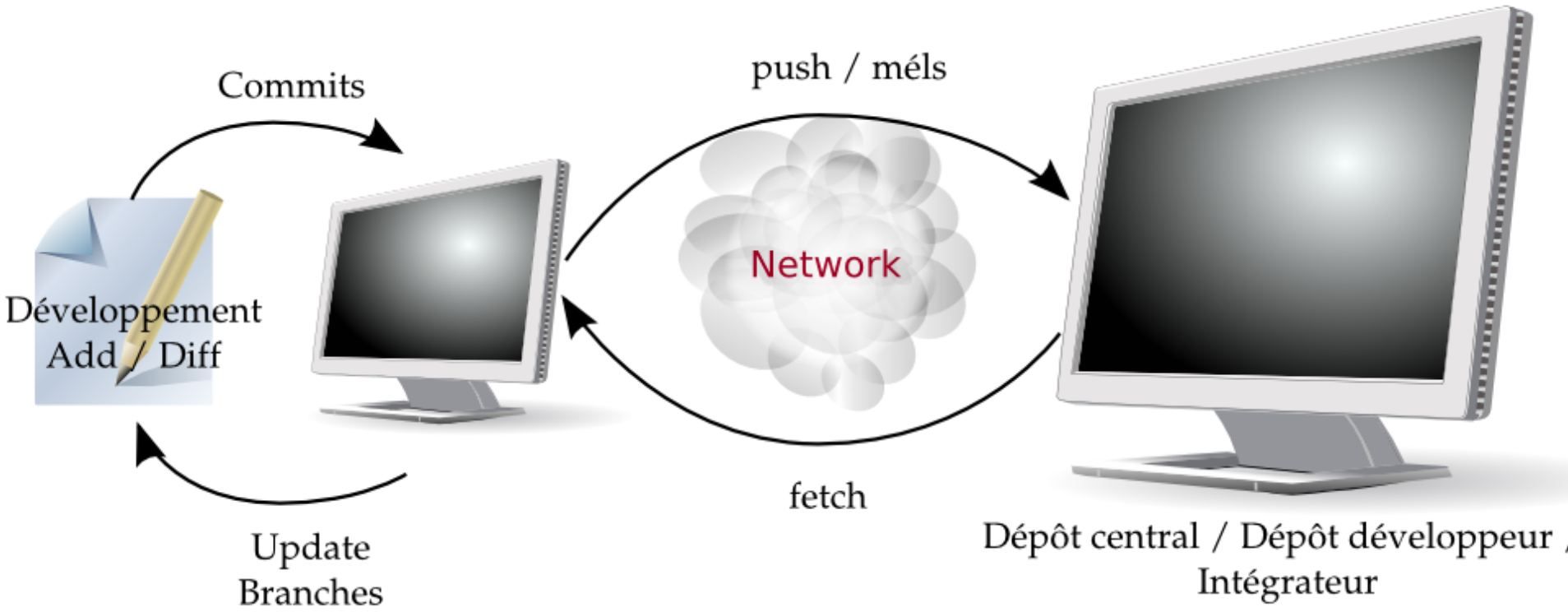


Plan

Dépôts centralisés



Dépôts centralisés



Dépôts centralisés

CVS ou Subversion

- Toutes les versions des fichiers sont entreposées dans un unique dépôt accessible aux développeurs (et contributeurs si autorisé) ;
- Les clients ne travaillent que sur une partie des données, en général, une simple branche ;
- Impossibilité de travailler « Off-line », sans faire des masses de commit ;
- Inclure un contributeur nécessite d'ouvrir le dépôt en écriture → besoins d'étendre le cercle de confiance pour un nouveau développeur sur le projet ;
- Chaque changement de branches nécessite un téléchargement de l'ensemble des données de la branche.

Mais pourquoi l'utiliser ?

Que des inconvénients → méthode historique des gestionnaires de versions.

Dépôts décentralisés

Arch, Bazar-Ng, Git, monotone etc.

- Chaque client possède l'ensemble des fichiers dans son dépôt local ;
  - On peut travailier « Off-Line » (à la plage, à la montagne, . . .)
- Chaque client possède l'ensemble des fichiers dans son dépôt local ;
- Aucune action du client ne nécessitent un accès au dépôt distant, sauf :
  - la mise à jour du dépôt local depuis l'extérieur ;
  - l'envoi d'information.
- Le changement de branche est rapide et est donc, une des méthodes de développement : utiliser les branches ;
- Les dépôts se gèrent par méls par un intégrateur, par un ensemble de dépôts, ou un dépôt centralisé.

Mais pourquoi est ce si peu utilisé ?

Bonne question :)

Branches distantes

Arch, Bazar-Ng, Git, monotone etc.

Un projet décentralisé possède deux types de branches :

- distante :
  - Pointe sur des dépôts distants en lecture et/ou écriture ;
  - Les dépôts distants peuvent être attaqués par une ou plusieurs personnes ;
- locale :
  - Peut être fusionné avec une branche distante pour envoyer des données.

Conflits

Quand ?

Les conflits apparaissent sur CVS/Subversion :

- au téléchargement des modifications `cvs update` ;
- au commit avec un refus de patcher.

Sur Git et les autres :

- Uniquement lors des fusions de branches : locale/locale, locale/distante ou distante/locale.

Plan

Plan

« Ahhhhhhhhhh de la porcelaine »

Guybrush Threepwood – The Curse of Monkey Island

Cogito ?

Informations générales

- un système de contrôle de versions utilisant Git comme couche bas-niveau → Cogito est compatible Git ;
- ensemble de scripts bash au-dessus des commandes Git pour simplifier son utilisation ;
- développé dans le but de fournir les mêmes commandes que CVS, SVN, BitKeeper etc.

Toutes les commandes sont préfixées par « cg- » ou « cg » :

Exemple

```
$ cg-add toto.cc
$ cg-log -scr master..bugs
$ cg clone http://lil.univ-littoral.fr/~quesnel/coursgit.git
```

Configuration

Elle hérite de Git...

- Définir son rôle :

À définir, dans `.bashrc`, `.zshrc`, `.tcshrc` etc.

```
export GIT_AUTHOR_NAME="Gauthier␣Quesnel"
export GIT_COMMITTER_NAME="Gauthier␣Quesnel"
export GIT_AUTHOR_EMAIL="quesnel@lil.univ-littoral.fr"
export GIT_COMMITTER_EMAIL="quesnel@lil.univ-littoral.fr"
```

- Avoir de la couleur dans les diff et les log :

Configuration

```
$ echo log -c > ~/.cgrc
$ echo diff -c >> ~/.cgrc
```

Plan

Gestion des sources

La gestion des fichiers sources, quelques commandes...

cg-add : ajout un dossier ou un fichier dans le dépôt (nécessite un cg-commit) ;

cg-mv : renomme ou déplace des fichiers du dépôt Git (demande un commit). Git ne sauvegarde pas les renommages et les déplacements. `cg-mv` = `cg-rm` + `cg-add` ;

cg-rm : supprime un fichier du dépôt Git. (demande un commit). L'option `-f` permet de supprimer physiquement le fichier de l'arbre ;

cg-restore : restaure un fichier de l'arbre courant à l'état du dernier commit ou celui précisé.

Gestion de l'historique

Historique, l'ensemble des versions des objets Git

cg-diff : faire une différence entre deux trees pour un ou plusieurs fichiers ;

cg-log : gère l'affichage de l'historique de tous les arbres, fichiers et modifications.

cg-admin-ls : liste l'ensemble des fichiers contenus dans un arbre (les objets tree de Git) ;

cg-admin-cat : détaille le contenu d'un fichier d'un arbre, courant ou précisé ;

cg-admin-lsobj : liste l'ensemble de tous les objets du dossier Git, blob, tree, commit et tag.

Gestion du dépôt local

Les scripts de gestion du dépôt local, quelques commandes de plus...

cg-commit : envoie les modifications locales dans le dépôt ;

cg-init : initialisation d'un dépôt local → `.git` ;

cg-status : affiche un ensemble d'information sur le dépôt local ;

cg-cancel : remise à zéro de l'état de l'arbre courant, c'est l'opposé de cg-commit ;

cg-clean : supprime tous les fichiers inconnus du dépôt Git c-à-d, tous les fichiers ignorés par la commande `cg-status` ;

cg-admin-uncommit : défait un commit ou un ensemble de commits. Très dangereux donc ;

cg-tag : associe un tag à un commit ;

cg-tag-ls : liste l'ensemble des tags d'un dépôt.

Gestion du dépôt distant

Les commandes utiles pour gérer le dépôt distant

cg-admin-setuprepo : commande définissant un dépôt publique, c'est-à-dire le contenu du dossier `.git` → dossier du projet ;

cg-push : envoie les commits de la branche courante ou d'une branche spécifiée au dépôt distant ;

cg-fetch : met à jour, en téléchargeant les commits, une branche distante du dépôt local ;

cg-merge : fusionne deux branches locales ou distances ;

cg-update : l'ensemble cg-fetch + cg-merge.

Remarque

`cg-admin-setuprepo` est une commande à lancer sur la machine du dépôt distant.

Plan

Initialisation d'un dépôt Git

- Initialisation à partir d'un dossier vide :

Exemple

```
$ mkdir monprojet
$ cd monprojet
$ cg-init
```

- Clone d'un dépôt existant :

Exemple

```
$ cg-clone http://lil.univ-littoral.fr/~quesnel/cours-git.git
```

Initialisation d'un dépôt Git

- Initialisation à partir d'un projet existant :

Exemple

```
$ tar zxf gaim-2.0.0-beta.tar.gz
$ cd gaim
$ cg-init
$ cg-add .
$ cg-commit
```

- Initialisation à partir des autres gestionnaires de versions :

Exemple

```
$ CVSROOT=:pserver:anonymous@vle.cvs.sf.net:/cvsroot/vle
$ MODULE=vle
$ INSTALLDIR=vlecvs
$ git-cvsimport -p x -v -k -d $CVSROOT -C $INSTALLDIR $MODULE
```

Gestion des tags

Un tag marque par un nom un certain commit

- cg-tag : pose un tag sur un commit :

Exemple

```
$ cg-tag 0.3.0 master
```

- cg-tag-ls : comme son nom l'indique, liste tous les tags enregistrés dans le dépôt Git.

Exemple

```
$ cg-tag-ls
    0.3.0       cc5517b4ea4134c296d4ce2b1d82700c44200c1e
$ cd ~
$ cg-clone coursgit.git#0.3.0 coursgit-0.3.0
```

Gestion des branches locales

On démarre avec une branche distante, origin, et une locale, master. La commande `cg-switch` change de branche et avec l'option `-r`, créer une branche avant de changer :

Exemple

```
$ cg-status
  R origin       1b1abc007fb383cd3f364267f5104a4060f1b92e
   >master       5f07bd0a436bd28531bd15a940309b22c551f69a

$ cg-switch -r origin bugs
$ cg-status
  R origin       1b1abc007fb383cd3f364267f5104a4060f1b92e
    master       5f07bd0a436bd28531bd15a940309b22c551f69a
   >bugs         1b1abc007fb383cd3f364267f5104a4060f1b92e
```

Gestion des branches distantes

Les branches distantes permettent de déposer ou de récupérer des commits.

- `cg-branch-add` : ajoute les branches distantes,
- `cg-branch-ls` : liste toutes les branches distantes,
- `cg-branch-chg` : change l'url d'une branche distante.

Remarque

Les noms et URL des branches sont stockées dans des fichiers situés dans : `monprojet/.git/branches`

Gestion des branches distantes

Il existe plusieurs méthodes pour désigner une branche distante :

- locale : /home/gauthier/toto.git
- rsync : deprécié
- http : http://vle.univ-littoral.fr/toto.git
- ssh : git+ssh://gauthier@193.59.192.126/home/gauthier/toto.git
- démon git : git://vle.univ-littoral.fr/toto.git

Remarque

Seules les méthodes ssh et du démon git permettent d'envoyer des informations dans le dépôt.

Gestion par dépôt type CVS

Tout est géré par les droits du système d'exploitation

- Création du dépôt public sur la machine distante :

Exemple

```
$ addgroup git
$ adduser toto git && adduser titi git
$ mkdir -p /var/git
$ chown :git /var/git
$ su toto
$ cg-admin-setuprepo -g git monprojet
```

- Les utilisateurs peuvent « pusher » à distance par leurs comptes ssh

toto

```
$ cg-clone ...
$ cg-switch -r tacheA
$ edit/new/mv/rm && cg-commit
$ cg-push machinedistante
```

titi

```
$ cg-clone ...
$ cg-switch -r tacheB
$ edit/new/mv/rm && cg-commit
$ cg-push machinedistante
```

Gestion par méls

Tout se gère en deux commandes : `cg-mkpatch` et `cg-patch`

- Création du lot de patchs :

Exemple

```
$ cg-mkpatch -r master..bugs > cgbugs123.patch
$ bzip2 cgbugs123.patch
$ sylpheed --compose toto@tata.fr --attach bugs123.patch.bz2
```

- Applique les patchs dans la branche courante :

Exemple

```
$ cg-switch -r bugs123
$ bzcat bugs123 | cg-patch -c
```

Plan

Plan

– *« But, we have CVS! . . . Right ? »*

– *« CVS is not the answer, CVS is the question. No is the answer. »*

Theodore Ts'o

L'avant Git

Historique :

- 2001 - Linux est développé sur CVS ;
- 2002 à 2005 - Linux est développé avec un logiciel non-libre, Bitkeeper mais celui-ci devient non utilisable pour Linux en 2004 ;
- 2005 - Création de Git par Linus Thorvalds ;
- avril 2005 - Utilisation de Git pour le développement de Linux.

Quelques précisions :

- Certainement le gestionnaire de versions le plus rapide à appliquer des patchs ;
- Git est distribué sous licence GNU GPL 2 ;
- Portable sur la plupart des Unix et Linux, MS Windows (moins !).

Git

Quelques phrases clés

- Rôle de Git : stocker des objets et les identifier par un clé Sha1
- Git stocke les fichiers au complet par les diff.
- Git est un ensemble de programmes (120 environ en version 1.4.2.1) :
  - gérer les objets d'assez bas niveau ;
  - compacter les objets en paquet ;
  - envoyer et recevoir des objets de plusieurs manières ;
  - créer et manger des patchs ;
  - interroger les données, `diff`, `grep` et `log` ;
  - chercher les bugs ou plutôt quand et qui l'a introduit (si si) ;
  - importer les projets des autres gestionnaires de versions.
- `http://www.kernel.org/pub/software/scm/git/docs/everyday.html`
- `http://www.kernel.org/pub/software/scm/git/docs/`
- pour le reste, les manpages sont là `:)`

SHA-1 ?

Secure Hash Algorithm

- SHA-1 est une fonction de hachage cryptographique de la NSA ;
- fonction de hachage → grand ensemble de données en un ensemble plus petit et unique (à quelques 2^{80} clés différentes) ;
- Génère un « hash » de 160 bits.

Exemple

```
$ echo a > toto
$ sha1sum toto
3f786850e387550fdab836ed7e6dc881de23001b  toto
$ echo a >> toto
$ sha1sum toto
d7c8127a20a396cff08af086a1c695b0636f0c29  toto
```

Plan

Les types

Rôle de Git : stocker des objets et les identifier par un clé Sha1

Git utilise quatre types d'objets :

- Blobs :
  - un blob représente le contenu d'un fichier ;
  - un blob par révision du fichier ;
  - pas de relation entre le nom ou l'emplacement du fichier et le blob ;
  - si un fichier est renommé, pas de changement dans le blob.
- Trees :
  - les trees sont des ensembles de pointeurs vers des blobs ;
  - un tree associe les noms des fichiers et les pointeurs de blobs ;
  - un tree décrit l'état d'une hiérarchie de dossiers à un moment donné ;
  - un tree peut pointer vers d'autres trees afin de mémoriser l'arborescence de fichiers.

Les types

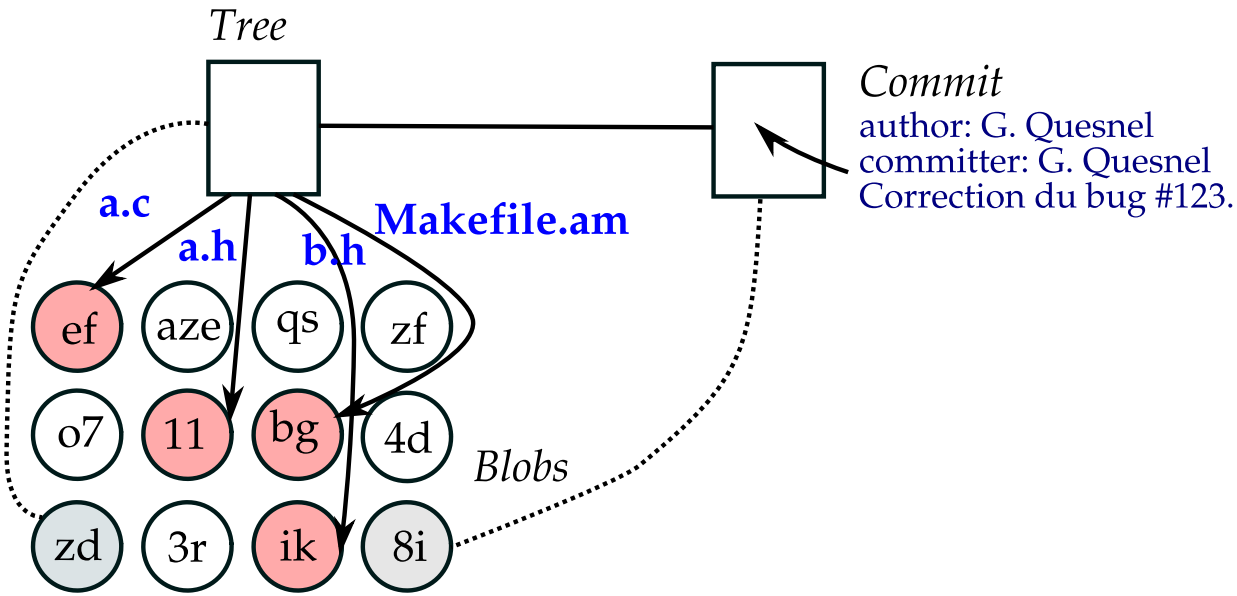
Rôle de Git : stocker des objets et les identifier par un clé Sha1

- Commits :
  - un commit pointe vers un arbre dont on souhaite sauver l'état ;
  - un commit pointe vers un ou plusieurs autres commits pour constituer l'historique et les branches ;
  - un commit est associé à une chaîne de caractères décrivant son action ;
  - un commit a un auteur et un committer.
- Tags :
  - le tag pointe vers un des objets précédents et porte un nom et une signature.

Les types

Petit exemple

Par exemple, pour un petit projet de quelques blobs :



Remarque

Les trees et les commits sont contenus dans les blobs.

Plan

Modes de développement

Git permet aux développeurs de gérer leurs sources de trois manières :

- un dépôt centralisé à la CVS, SVN tout en conservant les avantages de la conservation du dépôt local ;
- un dépôt pour chaque développeur, chacun se synchronise chez les autres, méthode traditionnelle de Arch ;
- une gestion par méls de dépôts, méthode de développement du noyau Linux et de son équipe de maintenance.

Commandes basiques du dépôt

Les commandes de tout le monde

git-init-db : initialisation d'un dépôt ;

git-clone : copie d'un dépôt existant (local ou par http, ftp, etc.) ;

git-fsck-object : pour valider un dépôt ;

git-repack : fait des paquets de blobs pour l'efficacité ;

git-prune : supprime les blobs uniques mais existants dans un paquet.

Création d'un nouveau dépôt

La création d'un dépôt, une commande simple :

Exemple

```
$ mkdir monprojet
$ cd monprojet
$ git-init-db
defaulting to local storage area
```

Un petit exemple pour la création d'un dépôt Git sur un projet existant.

Exemple

```
$ cd monprojet
$ git-init-db
$ git-add .
$ git-commit -a
defaulting to local storage area
```

Commandes développeurs

git-show-branch : montre les branches et leurs commits ;

git-log : montre tous les commits ;

git-whatchanged : montre les commits et les différences introduites ;

git-checkout : change ou crée des branches locales ;

git-branch : change de branches ;

git-add : ajoute un fichier ;

git-update-index : ajoute un fichier dans l'index ;

git-diff : fait des diffs entre des commits, trees ;

git-status : montre le status des branches ;

git-commit : avant le développement ;

git-reset : pour supprimer des changements (modifie HEAD) ;

git-pull : met à jour une branche distante et fusion avec la branche courante ;

git-tag : pour poser des tags.

Développeur individuel

Pas d'accès à un dépôt du projet → méls

git-clone : clone un projet distant ;

git-pull : met à jour une branche distante et fusionne avec la courante ;

git-fetch : met à jour la branche distante ;

git-push : envoie des données sur une branche distante ;

git-format-patch : prépare l'envoi d'un mail.

Exemple

```
$ git-clone git://git.kernel.org/pub/linux-2.6 lx26
$ édite / compile / teste
$ git-commit
$ git-format-patch origin
```

L'intégrateur

Développeur + reçoit les méls de patchs

git-am : pour appliquer des méls d'un contributeur sur une branche locale ;

git-pull : pour fusionner des branches avec d'autres collègues intégrateur ;

git-format-patch : pour préparer et envoyer des patchs aux contributeurs ;

git-revert : pour supprimer l'application d'un patch ;

git-push : pour publier sur le dépôt.

Exemple

```
$ mailx
$ s 6 7 2 9 ./appliquer.mbox
$ q
$ git-am ./appliquer.mbox
```

Plan

Utiliser Git ou Cogito ?

- Cogito
  - Fournit une interface utilisateur simple et intuitive
  - On garde toute la puissance du travail par branche avec `cg-switch`
- Git
  - C'est l'interface bas niveau à connaître pour optimiser
  - Pour des utilisations avancées (`git-rebase`)
  - Comprendre les index

Outils externes

Quelques outils fonctionnels

- `gitk` : interface graphique pour l'affichage des logs, commits, branches, tags, etc.
- `qgit` : affichage, création et application des patchs.
- `gitweb` : interface web (script perl) pour l'affichage d'un dépôt Git ;

Exemple

```
$ firefox http://vle.univ-littoral.fr/gitweb
$ firefox http://www.kernel.org/git/
```

Références

- http://download.ikaaro.org/doc/git/200607-ols.pdf
- http://kernel.org/pub/software/scm/cogito/
- http://git.or.cz/
- http://git.or.cz/gitwiki/
- http://www.kernel.org/pub/software/scm/git/docs/everyday.html
- http://www.kernel.org/pub/software/scm/git/docs/

fin

Licence

Cours Git / Cogito

quesnel@lil.univ-littoral.fr