Version Control Systems (VCS)

Vincent DANJEAN

Examples avec Git (et subversion)

Document dérivé de la présentation SVN faite à l'IUT de Villetaneuse
et de l'introduction à Git de Bart Trojanowski

16 décembre 2015

Plan

Travailler à plusieurs

fichier : `Essai.java`

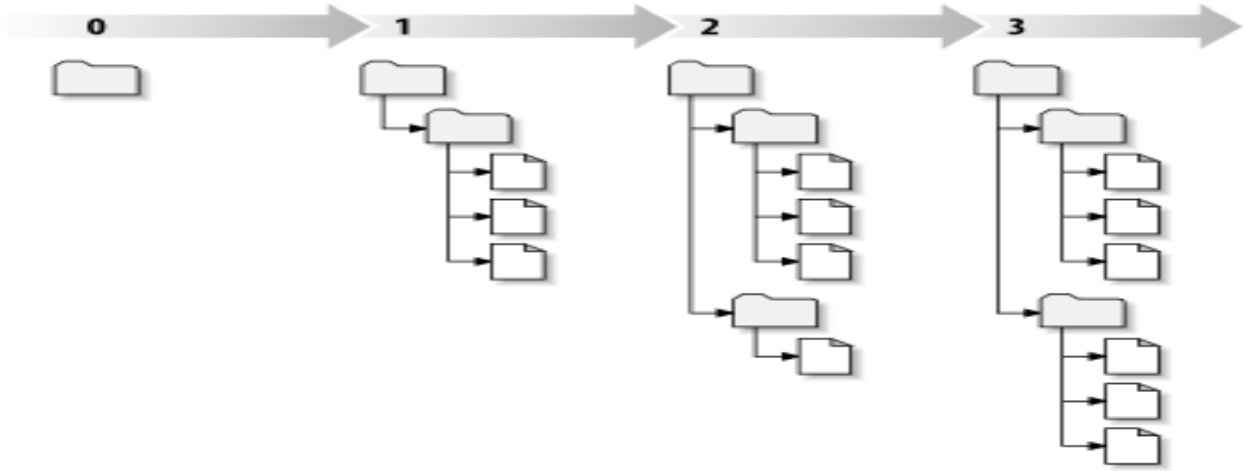
```
class Essai
{
int taille;
...
```

programmeur

programmeur

programmeur

Garder un historique



Savoir répondre aux questions

- Qui a modifié ce fichier ?
- Qui a écrit cette ligne ?
- Quelle était la version précédente de ce fichier ?
- Quels fichiers avait-on le 12 juin 2007 ?

Des outils d'aide au développement

Extensions indépendantes d'un même code

- version stable/de développement
- développement de fonctionnalités indépendantes

Intégration de développements multiples

- création/diffusion/intégration de patch
- fusion de différentes branches de développement

1, 2, 3, . . . , plein de VCS

VCS centralisés (un seul dépôt)

CVS historique, encore beaucoup utilisé
Subversion (SVN) remplaçant du précédent, mieux conçu

- multi-plateforme (Linux, MacOSX, Windows, . . .)
- intégration avec les IDE (plugins pour Éclipse, . . .)
- beaucoup utilisés (forges, . . .)

Mais aussi Codeville, Perforce (P4), . . .

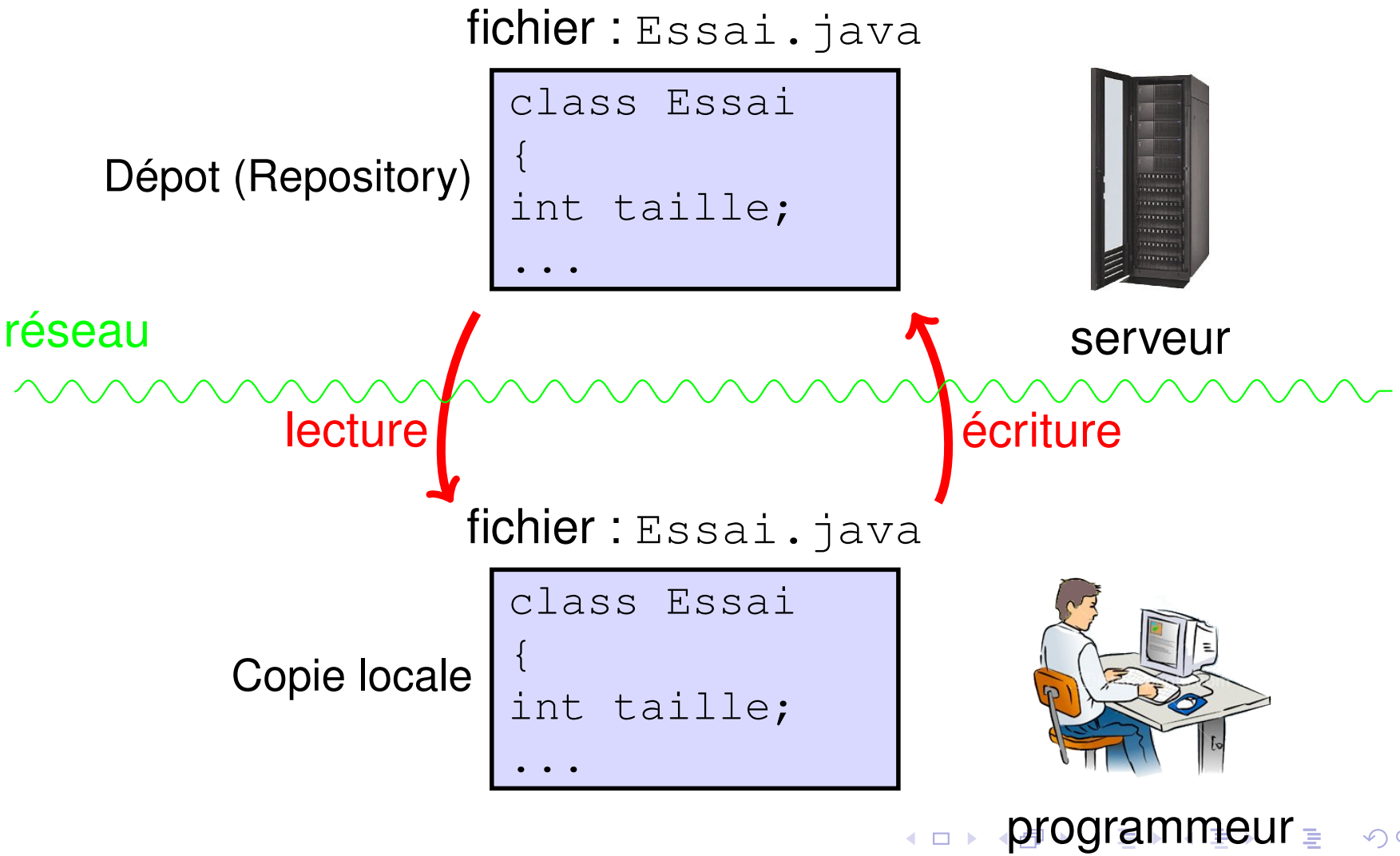
VCS décentralisés (plusieurs dépôts)

libres Git, Mercurial (hg), Darc, Bazaar (bzr), GNU arch, Codeville, Monotone, SVK, . . .

Propriétaires BitKeeper, Code Co-op, . . .

Plan

Principe : dépôt et copie locale



Concepts et opérations

dépôt lieu central (unique) avec tout l'historique, format interne

copie de travail répertoire de travail avec une version des fichiers (éventuellement modifiés localement)

checkout récupération d'une copie de travail depuis un dépôt

update mise à jour de la copie de travail avec les nouvelles versions dans le dépôt. Conflits possibles avec les modifications locales.

commit envoie des modifications locale sous forme d'un 'commit' dans le dépôt.

Conflits

Apparition des conflits

commit copie de travail pas à jour : update nécessaire

update modifications locale **et** dans le dépôt : conflit à résoudre par le programmeur

Unité de suivi : le fichier

- aucun problème (pour le SCM) si les fichiers modifiés sont différents

La fusion automatique est possible si :

- il s'agit d'un fichier texte
- ET les modifications sont à des endroits éloignées les unes des autres (quelques lignes)

Plan

Subversion

Le dépôt

- contient toutes les versions (révisions) de tous les fichiers
- n'est JAMAIS manipulé directement sauf pour
  - le créer (évidemment)
  - éventuellement pour changer les permissions
- est stocké sous forme non manipulable directement
- toute révision est associée à un auteur, un commentaire, ...

Copie de travail

- Une (au moins) par utilisateur
- Contient une révision particulière du dépôt avec éventuellement des modifications locales
- La commande `svn` permet de la manipuler

Les commandes principales

`svnadmin create` : créer un nouveau dépôt

`svn import` : créer un nouveau répertoire/projet
`svn checkout` : lire tout un projet
`svn update` : lire/mettre à jour depuis le dépôt
`svn commit` : écrire/modifier le dépôt (nouvelle révision)
`svn status` : état de la copie locale
`svn add` : ajouter un fichier
`svn rm` : enlever un fichier
`svn help` *cmd* : obtenir de l'aide sur *cmd*

Gérer le dépôt

Un seul dépôt pour plusieurs projets

- utiliser des répertoires différents pour séparer les projets
- URL pour le désigner
  - `file:///chemin/complet/vers/le/dépôt`
  - `svn+ssh://login@host/chemin/complet/dépôt`
  - `svn://host/chemin/relatif/au/serveur`

  L'URL peut désigner une sous-partie du dépôt (ie un projet particulier)

Manipulations directes

- création : `admin create` *`répertoire`*
- permissions : qui a le droit d'écrire dans ce dépôt (voir plus loin)

Créé un projet dans le dépôt

À utiliser une seule fois par projet

```
svn import -m "commentaires" répertoire URL
```

commentaires obligatoire, pour expliquer ce qui arrive dans le dépôts

répertoire le répertoire local (existant) à importer

URL l'URL du dépôt

Récupérer une copie locale d'un projet existant

À utiliser une seule fois par copie locale

```
svn checkout URL [répertoire]
```

URL : l'URL du dépôt

répertoire : répertoire dans lequel sera crée la copie locale (pas défaut, le dernier composant de l'URL)

Mise à jour de la copie locale

À utiliser souvent

`svn update [fichier|répertoire]`

	par défaut agit sur le répertoire courant
répertoire	met à jour ce répertoire (et sous répertoires récursivement)
fichier	met à jour ce fichier uniquement

comportement classique de beaucoup de commandes SVN

Écriture des modifications locales

À utiliser souvent

```
svn commit -m "correction ..."
[fichier|répertoire]
```

commentaire obligatoire, expliquez vos modifications, soyez clairs : c'est très utile

Crée une nouvelle révision dans le dépôt

Autres commandes très utiles (1/2)

`svn status [fichier|répertoire]`

état de la copie locale (fichiers modifiés, . . .)

`svn diff [fichier|répertoire]`

différences entre l'état actuel et la révision du dépôt (ie vos modifications non commitées)

`svn add fichier|répertoire`

ajoute un fichier ou un répertoire

`svn rm fichier|répertoire`

supprime un fichier ou un répertoire (on peut toujours retrouver les versions commitées avant)

Autres commandes très utiles (2/2)

`svn mkdir répertoire`

crée un nouveau répertoire

`svn log [fichier|répertoire]`

voir la liste des révisions avec les commentaires associés

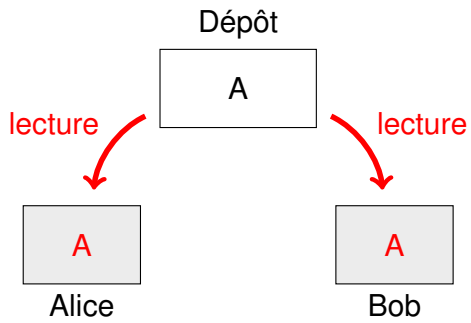
`svn mv fichier|répertoire fichier|répertoire`

renomme un fichier ou un répertoire (sans perdre son historique)

`svn cp fichier fichier`

copie un fichier AVEC son historique

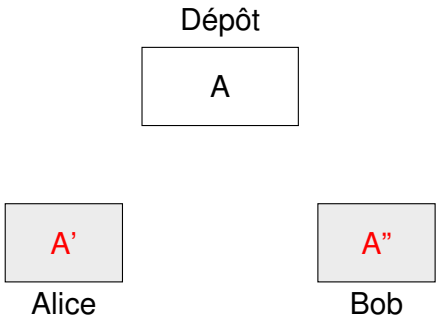
Exemple d'utilisation




```
[alice@mandelbrot ~/projet]$ svn update
U sources/Compteur.java
U sources/Personnage.java
U sources/Essai.java
U images/fond.png
Updated to revision 124.
[alice@mandelbrot ~/projet]$
```

U : fichier mis à jour

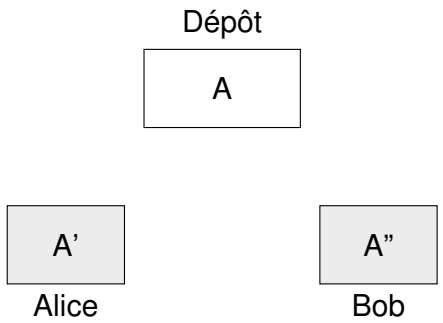
Exemple d'utilisation




```
[alice@mandelbrot ~/projet]$ emacs sources/*.java
[alice@mandelbrot ~/projet]$
```

```
[bob@portable ~/projet]$ emacs sources/*.java
[bob@portable ~/projet]$
```

Exemple d'utilisation



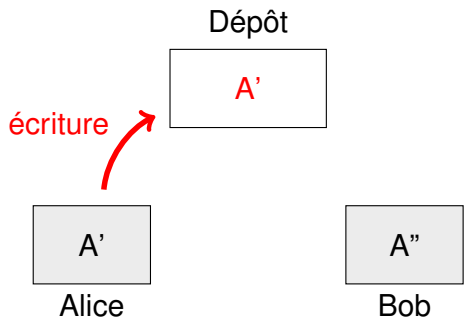

```
[alice@mandelbrot ~/projet]$ svn status
? sources/OutilsSon.java
M sources/Toto.java
M sources/Essai.java
[alice@mandelbrot ~/projet]$
```

? : fichier inconnu dans le dépôt

M : fichier modifié dans la copie locale

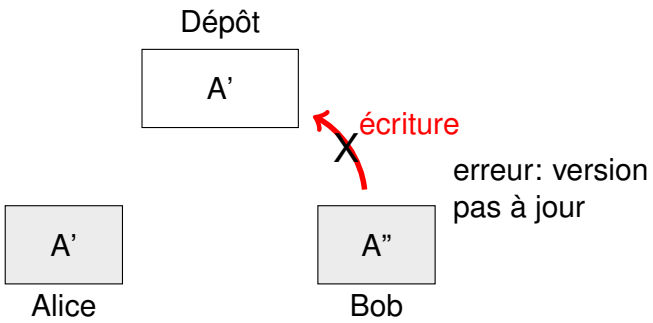
`svn help status` pour connaître tous les codes

Exemple d'utilisation



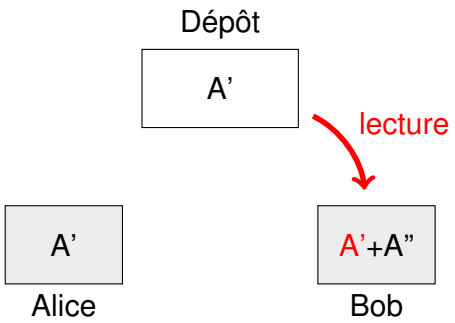

```
[alice@mandelbrot ~/projet]$ svn commit -m "bug fix:.."
Sending source/Toto.java
Sending source/Essai.java
Transmitting file data .
Committed revision 125.
[alice@mandelbrot ~/projet]$
```

Exemple d'utilisation




```
[bob@portable ~/projet]$ svn commit -m "mon bug..."
Sending source/Essai.java
svn: Commit failed (details follow):
svn: Out of date: 'source/Essai.java' in transaction '1
26-124'
[bob@portable ~/projet]$
```

Exemple d'utilisation

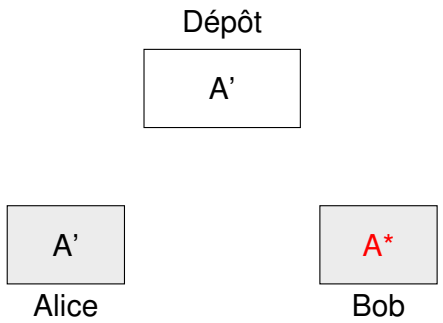



```
[bob@portable ~/projet]$ svn update
C sources/Essai.java
G source/Toto.java
Updated to revision 125.
[bob@portable ~/projet]$
```

G : fusion automatique

C : conflit à régler manuellement

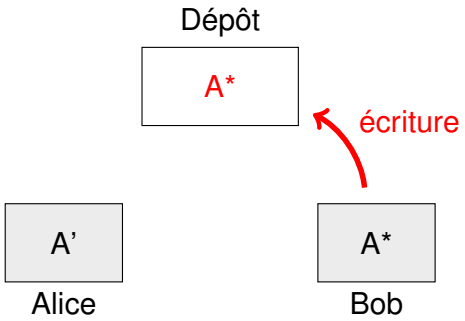
Exemple d'utilisation




```
[bob@portable ~/projet]$ emacs sources/Essai.java
[bob@portable ~/projet]$ svn resolved sources/Essai.java
Resolved conflicted state of 'sources/Essai.java'
[bob@portable ~/projet]$
```

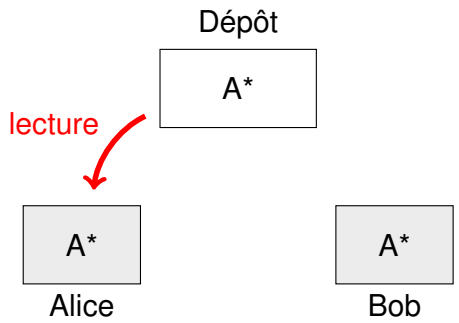
La résolution des conflits est détaillée ensuite

Exemple d'utilisation




```
[bob@portable ~/projet]$ svn commit -m "bug fix:.."
Sending source/Toto.java
Sending source/Essai.java
Transmitting file data .
Committed revision 126.
[bob@portable ~/projet]$
```

Exemple d'utilisation




```
[alice@mandelbrot ~/projet]$ svn update
U sources/Toto.java
U sources/Essai.java
Updated to revision 126.
[alice@mandelbrot ~/projet]$
```

Résolution des conflits (1/2)

Fichier avec conflit : `Essai.java`

```
class Essai
{
<<<<<<< .mine
  int tailleXYZ;
=======
  int tailleABC;
>>>>>>> .r125
String nom;
...
```

+ trois fichiers créés

- `Essai.java.r124` : version avant mes modifs
- `Essai.java.r125` : version actuelle du dépôt
- `Essai.java.mine` : ma version (avant fusion)

Résolution des conflits (2/2)

1. éditer le fichier `Essai.java` en rectifiant
2. effacer les trois fichiers créés
   - manuellement
   - ou avec `svn resolved file`
3. commiter

Plan

Retour sur les composants des VCS

Composants des dépôts

- objects / blobs / diffs / deltas / patches
- commits / changesets / revisions
- ancestry / history
- tags / labels
- branches / heads

Répertoire de travail

- fichiers
- liste de fichiers à ajouter/enlever

Les limitations des VCS centralisés

- impossible de commiter dans le TGV
- difficultés (sociales) pour obtenir les droits de commit
- diffusion immédiate de chaque modification commitée

L'apport des DVCS

- chaque copie contient tout l'historique
- les commits sont locaux
- deux nouvelles opérations :

  clone duplication d'un dépôt

  push/pull/fetch envoi/réception des nouvelles modifications d'un dépôt vers/dans un autre

Les opérations classiques des VCS centralisés

Checkout/Update

- crée le répertoire de travail
- obtient l'état courant
- MAJ du répertoire de travail

Commit

- regroupe les modifications
- envoie les modifications

Diff/Log

- obtient les infos du serveur
- affichage (diff/objets/historique)

Les opérations classiques des VCS décentralisés

Checkout/Update

- obtient l'état courant
- MAJ du répertoire de travail

Commit

- regroupe les modifications
- sauvegarde dans le dépôt

Diff/Log

- obtient les infos du dépôt local
- affichage (diff/objets/historique)

Clone

- création (dépôt **et** répertoire de travail)
- récupération de l'historique

Fetch/Pull

- transfert dépôt extérieur vers dépôt local
- parfois, MAJ rep de travail

Push

- transfert dépôt local vers autre dépôt

Les atouts des DVCS

- micro-commits non intrusifs
- opérations déconnectées
- pas de point unique critique
- sauvegardes triviales à effectuer
- multiples branches de développement en parallèle naturellement

Fusion des branches

- le point difficile pour les concepteurs de DVCS
- le système doit se souvenir de ce qu'il a déjà inclus
- "numérotation" unique des changeset nécessaire

Petite histoire de la naissance de Git

2002

- Linus utilise BitKeeper pour développer Linux
- BK s'améliore (retour d'expérience et de bugs)
- Le développement de Linux passe mieux à l'échelle

6 avril 2005

- BitMover retire la licence gratuite de BK
- Linus écrits son propre VCS : Git

18 avril 2005

- Git est capable de faire des fusions

16 juin 2005

- Git est officiellement utilisé pour gérer Linux

Identification unique des objets avec SHA1

Un identifiant SHA1 peut désigner

un fichier : contenu du fichier

un répertoire : liste des identifiants SHA1 des fichiers/répertoires avec leurs métadonnées (mode, ...)

un commit : répertoire + métadonnées (message, auteur, ...) + commit(s) parent(s)

Monotone et Mercurial utilisent ce même principe.
BitKeeper utilise des renumérotations des changesets.

Les trois entités pour le travail avec Git

Le dépôt (repository)

- contient l'historique des commits, les tags et les branches (avec leur tête)

L'index

- contient les changements qui seront commitées
- au plus une nouvelle version d'un fichier (sauf en cas de fusion)

Le répertoire de travail

- la version de travail des fichiers

Utilisation de GIT

- Basics
  - Creating a repository
  - Creating commits
  - Looking at the history
- Tagging & Branching
  - Creating
  - Merging
  - Rebase
- Patches
  - Generating
  - Accepting

- Decentralized
  - Cloning
  - Fetch / Pull
  - Push
  - Daemon
  - Remotes
- Tools
  - Working with other SCMs
  - Housekeeping
  - Visualization
- Tricks

Commandes

format

```
$ git <options> <command> <cmd-options>
```

list of commands

```
$ git help
```

usage info

```
$ git init -h
usage: git init [--template=...] [--shared]
```

detailed help

```
$ git help init
$ git init --help
$ man git-init
```

Configuration

global configuration is in $HOME/.gitconfig

```
$ git config --global --list
user.name=Vincent Danjean
user.email=Vincent.Danjean@imag.fr
color.branch=auto
color.pager=true
```

repository configuration is in repo/.git/config

```
$ git config --list
section.variable=value
...
```

initial settings

```
$ git config --global user.name "Vincent Danjean"
$ git config --global user.email danjean@imag.fr
```

GUIs

Creating commits

```
$ git gui
$ gitg
$ qgit
$ git cola
```

Managing branch and history

```
$ gitk --all
```

Création d'un dépôt sur une forge

- Un dépôt créé par projet automatiquement (au plus)
- Forge Inria : notion de clones personnels désormais
- L'accès shell permet de créer des dépôts
  - non visibles dans l'interface web de la forge

`git-init-gforge`

- outil maison (patches welcome)
- paquet Debian/Ubuntu dans mon dépôt perso
- `git clone git+ssh://login@shell.gforge.inria.fr/home/users/vdanjean/public_git/software/tools/git-init-gforge.git`

Plate-formes dédiés à Git

github

- `https://github.com/`
- très populaire, très agréable à utiliser
- code non libre, dépôts privés payants

gitorious

- `http://gitorious.org/`
- un peu moins de fonctionnalités
- projet libre (GNU AGPL), peut être installé localement
- dépôts privés payants sur l'instance "officielle"

Utiliser un dépôt subversion avec Git

- commits locaux, utilisations de branches
- linéarisation des commits, un seul dépôt Git

Bi-directionnal exchanges

```
$ git svn clone svn+ssh://host/project/trunk project
$ cd project
... work work work ...
$ git commit
...
$ git svn fetch
$ git svn rebase
$ git svn dcommit
```

Passer de Subversion à Git

- facile si on utilise un seul dépôt central (+ dépôts locaux)
- attention: après un commit, faire un push
- Git : modifs, commit, fetch, merge puis push
- Subversion : modifs, update, merge puis commit

Être raisonnable avec Git

- Git : outil très puissant (réécriture historique, etc.)
- un peu comme root sur une machine unix
- d'autres SCM distribués sont plus restrictifs (mercurial, etc.)

Workflow et commits

- Git n'impose rien (subversion un peu plus)
- workflow à définir avec ses collaborateurs
- Git peut servir à transférer des fichiers entre machine. Est-ce vraiment une bonne utilisation ?

Conclusion

De nombreux outils disponibles

- subversion (svn)
- Git : boîte à outil très puissante
  - git-rebase, git-svn, git-bisect, ...
  - à utiliser avec ou sans modération
- mais aussi quilt (gestion de patches), ...

Outils bons ou mauvais : cela dépend de l'utilisateur

Les méta-données sont très importantes

- faire des commits "logiques"
  - ils peuvent être nombreux et petits
  - git permet de "retravailler" des commits (locaux)
- mettre un message de commit correct

Liens utiles

Autres présentations générales

- `http://www-verimag.imag.fr/~moy/cours/formation-git/`