Référentiels de code source

—

gestion de versions

—

SVN et GIT

Table des matières

I - SCM (Source Code Management)

Un gestionnaire de code source (SCM) permet de :

- travailler en équipe (référentiel de code partagé)
- mémoriser différentes versions (pour y revenir si besoin)

1.1. Principaux "SCM"

SCM	*époque*	*caractéristiques*
CVS (Concurrent version system)	Avant l'an 2000	Une des premières technologies "SCM" open-source
...		
SVN (subversion)	2002-2012 environ	Version améliorée de CVS
GIT	>= 2010 environ	Mode distribué , d'origine linux
Mercurial	>=2010 environ	ressemble à GIT
...		...

II - Essentiel de SVN (subversion)

1. SVN

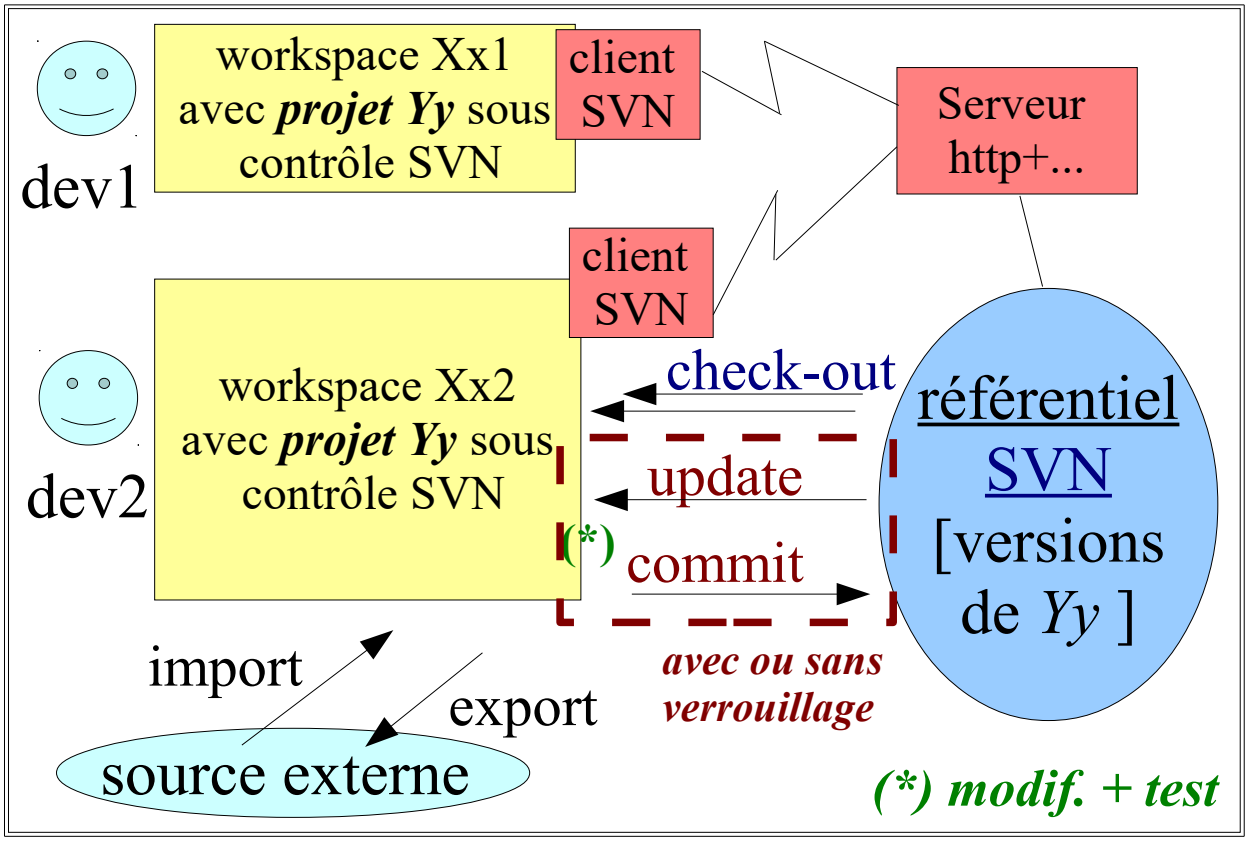
1.1. Terminologie & fonctionnement de CVS et SVN

CVS = **C**oncurrent **V**ersion **S**ystem , **SVN** = **S**ub**V**ersio**N**

CVS est un produit "Open Source" qui permet de **gérer différentes versions d'un ensemble de fichiers sources** lié au développement d'un certain module logiciel.
Différents programmeurs peuvent travailler en équipe sur des fichiers partagés au niveau d'un référentiel commun.

SVN se veut avant tout être une "***version améliorée de CVS"*** qui

- ne remet pas en cause les principes fondamentaux de CVS (référentiel commun et commit,update,...)
- a refondu l'implémentation du serveur et des référentiels (meilleur gestion des transactions, protocole d'accès plus simples, ...)



Vocabulaire utilisé par CVS (puis SVN):

import	Créer un nouveau module en rapatriant dans le référentiel le code d'un répertoire existant [qui devient alors "l'ancienne source"].
check-out	Récupérer une copie à jour (vue locale à un programmeur) d'un module logiciel géré par SVN. Cette vue locale correspondra à un répertoire de travail qui sera sous le contrôle de SVN [NB: Ce répertoire contiendra un sous répertoire caché .svn]
update	Récupérer la dernière révision d'un fichier (pour lecture et/ou mise à jour)
commit	Enregistrer une nouvelle version d'un fichier (après modification et test unitaire). SVN incrémente alors automatiquement le numéro de révision.
export	Extraire depuis le référentiel une copie d'un module. Le répertoire externe alors créé ne sera plus sous le contrôle de SVN [Là est la principale différence avec un check-out classique]
...	

Vocabulaire (suite) utilisé par CVS et SVN:

module	module logiciel (Application, Librairie, ...) à placer sous le contrôle de SVN. Un module correspond à une arborescence de fichiers (sources, make).
révision	Version auto-incrémentée précise d'un module SVN (ou d'un fichier avec CVS) (ex: **1.1** , 1.2 , ... 1.9 , 1.10 , **1.11** , ...) NB: incrémentation automatique lors d'un "commit"
Release / Tag	version contrôlée / taguée d'un module logiciel complet (ex: "**rel-1-1**" , "**rel-1-2**" , ...) NB: un nom de release (ou **Tag)** doit avec CVS commencer par une lettre et ne doit pas contenir de point (.) ni de blanc .
...	

Principales différences entre SVN et CVS:

- Les numéros de révisions SVN sont liés à l'ensemble d'un module (projet) et non plus à un seul fichier (comme pour CVS).
- ....

2. Principales commandes de SVN

2.1. Récapitulatif des commandes SVN

Commandes SVN (avec syntaxe)	*utilités*
svnadmin create NomRepoProjet	Créer un nouveau dépôt pour un projet
svnadmin dump pathRepo > archiveSvn.db	Créer une sauvegarde du dépôt
svnadmin load pathDepot < archiveSvn.db	Recharge le contenu d'une archive
svn co *url_depot* *ou bien* **svn checkout url_depot**	Récupérer (sur poste développeur) un projet svn existant (sur un dépot)
svn add *./fichier_ou_dossier* **svn delete** *./fichier_ou_dossier*	Ajouter ou retire un nouveau ou ancien fichier ou dossier à ceux que SVN doit prendre en charge
svn commit *./fichier_ou_dossier* **-m** *message* **svn commit -m** *message*	Envoyer vers le dépôt une nouvelle version d'un fichier (ou bien de tout de qui a été modifié par défaut)
svn update **svn update** *./fichier_ou_dossier*	Récupérer la dernière version de l'ensemble ou d'un fichier particulier
svn info	Affiche les informations générales liées au projet (url_dépot , dernière modification, ...)
svn log **svn log -r** revi*sionX*:**r**revi*sionY*	Affiche la liste des modifications qui ont été apportées au projet
svn -h **ou** **svn --help** **svn -h** *cmdeSvn*	Obtenir une aide (liste des commandes svn) ou aide détaillée sur une commande
svn diff ./fichier	Affiche les différences entre version locale et du dépot
svn export *répertoireDestination* **svn import** *répertoireProjetNonSvnOrigine url_repo_svn*	Exporte une copie (non svn) du projet Ajoute/importe dans le référentiel SVN le contenu d'un projet initial (par encore sous le contrôle de SVN)
svn merge ...	Fusion de branches ou … (plusieurs variantes)
svn revert	Annule les modifications locales (inverse du commit)
svn copy , list , mkdir , move , status , ...	Autres commandes SVN

2.2. Exemples

Création d'un dépôt SVN « repo1 »(ici sur machine locale) :

dans /var/scm/svn ou ailleurs :
svnadmin create repo1

Ajouter le contenu d'un projet initial « p1 » (non svn) dans le dépôt « repo1 » :

svn -m "initial svn import" **import** p1/pom.xml.txt file:///var/scm/svn/repo1/trunk/p1/pom.xml.txt
svn -m "initial svn import" **import** p1/src file:///var/scm/svn/repo1/trunk/p1/src
#svn import --help

Récupérer (ailleurs) le contenu (sous le contrôle de SVN) du projet p1 du dépôt « repo1 » :

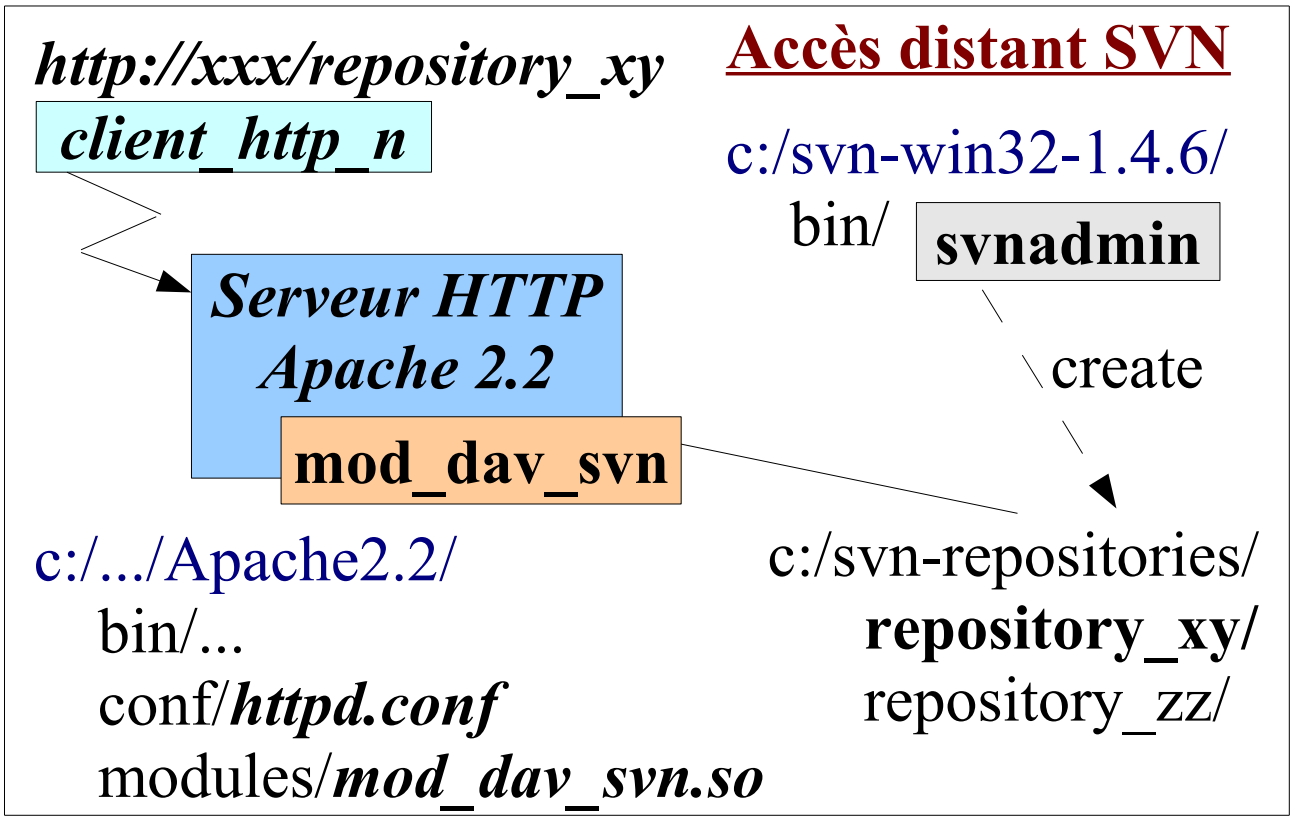
svn **checkout** file:///var/scm/svn/repo1/trunk/p1

Envoyer les modifications effectuées dans la branche src du projet local « p1 » :

svn **commit** -*m "yet another commit*" p1/**src/***

3. Configuration de SVN , accès webdav/http

SVN coté serveur (avec configuration d'un accès webdav/http):



3.1. installer SVN et créer un référentiel

- **télécharger SVN** alias subversion (site tigris ou collabnet ou ...)
  - version "bin for win32 build with apache2.2" ou ...

- **installer le code du serveur SVN** en dézippant vers c:
  --> *C:\svn-win32-1.4.6*
  Attention : ne surtout pas dézipper (sous windows) le contenu de SVN...zip dans un répertoire contenant des caractères blancs (ex : c:/program files/) car beaucoup de commandes ou produits d'origine « unix/linux » ne fonctionnent pas bien dans ce cas.

*NB : sous un **linux** de type « debian » (ex : Ubuntu) , l'installation de svn s'effectue simplement via « sudo **apt-get install svn** »*

- **créer un répertoire qui accueillera les référentiels**
  mkdir c:\svn-repositories (ou /var/scm/svn ou .....)
  et ce placer dedans (**cd**)

- **créer un référentiel vide**:
  set SVN_HOME=c:\svn-win32-1.4.6
  "%SVN_HOME%/bin/svnadmin" create ***myrepository***

  Sous linux :
  cd ***/var/scm/svn***
  svnadmin create ***myrepository***

#nb www-data est le groupe de apache2
sudo chgrp -R www-data *myrepository*

3.2. paramétrer un accès distant HTTP/DAV depuis Apache2.2

- **installer si besoin apache2.2**
- **installer le module d'accès distant dans apache2.2**:
  recopier les fichiers "***mod_dav_svn.so***" et "***mod_authz_svn.so***"
  de %SVN_HOME%/bin vers %Apache2_HOME%/modules
  recopier également les ***dll*** de %SVN_HOME%/bin
  vers "%Apache2_HOME%/bin *(sans écraser celles qui sont déjà présentes !!!)*

  et ajouter ceci dans **conf/http.conf** de apache2.2:

  LoadModule dav_module modules/mod_dav.so
  LoadModule dav_fs_module modules/mod_dav_fs.so
  LoadModule dav_svn_module modules/mod_dav_svn.so
  LoadModule authz_svn_module modules/mod_authz_svn.so

  ....

  <Location /myrepository>
  DAV svn
  SVNPath C:/svn-repositories/myrepository
  </Location>

  rédémarrer ensuite Apache2 (stop/start) et vérifier l'accès distant au référentiel via l'url suivante: *http://localhost/myrepository*

Configuration équivalente sous linux :

Apache 2.2 généralement déjà installé (sinon installation via **apt-get**).
Arborescence de la configuration de apache2 sous linux ubuntu :
/etc/apache2
httpd.conf *(quasiment vide sous linux car basé sur une liste de sous fichiers)*
mods-available/*xxx.**load*** et *xxx.**conf***
mods-enabled/*lien_vers_xxx.**load*** et *lien_vers_xxx.**conf***

Pour activer un module optionnel « yyy » de apache (tel que « dav » ou « dav_svn ») , il faut d'une manière ou d'une autre ajouter dans le répertoire mods-enabled des liens symboliques vers les fichiers «yyy.conf» et «yyy.load» du répertoire mods-available.

Ceci peut s'effectuer rapidement grâce à la commande unix **a2enmod** *(Available to ENabled MODule).*

Les modules nécessaires pour SVN sont « **dav** » , « **dav_svn** » et «**authz_svn** »
et les commandes unix à déclencher sont donc les suivantes :
sudo a2enmod dav
sudo a2enmod dav_svn
sudo a2enmod authz_svn

configuration à paramétrer dans **dav_svn.conf** :

```
...
<Location /svn>
  DAV  svn
  # Set this to the path to your repository
  #SVNPath /var/lib/svn/repo1
  # Alternatively, use SVNParentPath if you have multiple repositories under
  # under a single directory (/var/lib/svn/repo1, /var/lib/svn/repo2, ...).
  # You need either SVNPath and SVNParentPath, but not both.
  SVNParentPath /var/scm/svn
</Location>
…
```

Redémarrage du service apache2 :

service apache2 restart

3.3. Sécuriser l'accès distant au référentiel

Ajouter quelqueques paramètres au bloc <Location> de **httpd.conf** (ou dav_svn.conf) du serveur Apache2.2:

```
<Location /myrepository>
DAV svn
SVNPath C:/svn-repositories/myrepository
AuthType Basic
AuthName "SVN Repository"
AuthUserFile C:/svn-repositories/conf/dav_svn.passwd
Require valid-user
AuthzSVNAccessFile C:/svn-repositories/conf/authz.txt
</Location>
```

NB1:
Le fichier ***dav_svn.passwd*** comportera les ***mots de passe cryptés pour Apache2/Dav/Svn*** .

Pour créer (ou complémenter) ce fichier, il faut déclencher l'instruction suivante:
"%APACHE2_HOME%/bin/htpasswd" -c -m conf/dav_svn.passwd ***svnuser***
puis saisir le mot de passe (ex: ***svnpwd***) lorqu'il sera demandé

NB2:
Le fichier ***authz.txt*** comportera les ***droits d'accès au référentiel SVN***.
En voiçi un exemple:

```
[groups]
dev = svnuser, svnuser2
[/]
anonymous = r
@dev = rw
[/trunk]
@dev = rw
```

```
[/branches]
@dev = rw
[/tags]
@dev = rw
```

NB3:
Toute modification de **httpd.conf** nécessite un redémarrage de Apache2.2.
Si le serveur HTTP refuse de démarrer , on peut débugger en mettant temporairement en commentaire certaines lignes de httpd.conf (ajout d'un # en début de ligne) : ceci permet de situer rapidement le problème.

4. Plugin SVN pour eclipse

Au sein de l'I.D.E. eclipse , il existe des perspectives et des vues dédiée à CVS ou SVN.

4.1. Connexion au référentiel CVS / SVN depuis eclipse

Depuis la **perspective "CVS Repository"** ou bien "**SVN Repositories**" ,
click droit / **New repository Location**

- pour SVN ==> **url=http://machine_avec_apache2_2_et_svn/myrepository**
  + compte SVN (***svnuser,svnpwd***)

4.2. Intégrer un embryon de nouveau projet dans un référentiel CVS/SVN (import SVN)

1) Sur un seule poste de développement , préparer un nouveau projet
2) Paramétrer sa structure (répertoire , packages , ...)
3) Depuis la perspective Java ordinaire , click droit / **Team / Share Project**

4.3. Charger ce nouveau projet sur les autres postes de l'équipe de développement depuis un référentiel CVS/SVN (checkout)

- Depuis un workspace vide , click droit / ***Import/ Checkout CVS project***
  ou bien **Import/ Other.../ Checkout SVN project**

4.4. Gérer les révisions CVS/SVN depuis eclipse

Depuis perspective java , click droit , **Team** , ...
... **commit** ==> pour créer une nouvelle révision (après changement dans le code + tests)
.... **show in Resource History** ==> pour obtenir la liste des révisions (et par click droit sur une ancienne révision on peut obtenir la liste des différences vis à vis de la version actuelle)

4.5. Gérer les versions / releases

Depuis perspective java , click droit , **Team** ...
.... **Tag as version (ex: Rel_1_1)** [effectuer un Refresh dans la vue ressource History]

4.6. Autres commandes utiles

- **Team / disconnect** pour se déconnecter totalement du référentiel CVS/SVN [opération inverse = import / checkout complet]
- **Replace with** => pour remplacer la révision courante par une autre (éventuellement plus ancienne)
- **Compare with** => pour afficher les différences entre plusieurs révisions

4.7. Remarques importantes

- Après avoir ***supprimé un fichier*** définitivement inutile (depuis le workspace eclipse), il faudra penser à déclencher un "Team/**commit**" sur le **répertoire parent** de façon à ce que le référentiel prenne lui aussi en compte la suppression du fichier devenu inutile.
  Et un "Team/update" sur ce même répertoire parent fera disparaître le fichier effacé sur les autres postes de développement connectés au même référentiel.
- Après avoir ajouté un nouveau fichier , on peut activer le menu "***Team/Add to Version Control***" pour que ce nouveau fichier soit vu par SVN (il fera plus tard l'objet d'un "commit").
- Si suite à une tentative de commit via SVN, une erreur survient (conflit/merge) car un autre développeur a modifié ce fichier et a déjà effectué un commit il faut alors procéder de la façon suivante:
  a) effectuer un "**Team/update**" (ce qui conduit à une vue locale de type "merge")
  b) résoudre si possible les conflits en éditant le fichier (fusionner variante, supprimer variante, ...) soit manuellement , soit via "**Team/edit conflicts**".
  c) marquer les conflits comme résolus via "**Team/Mark Resolved**" puis tester et effectuer un "**Team/commit**" ou bien si conflits trop importants effectuer un "**Team/Revert**".
- Le menu "***Team/Synchronized with Repository***" permet de visualiser l'ensemble des différences entre la vue locale et le référentiel (avant les commit & update).

III - Essentiel de GIT

1. Présentation de GIT

GIT est un **système de gestion du code source** (avec prise en charge des différentes **versions**) qui fonctionne en **mode distribué** .

GIT est moins centralisé que SVN . Il existe deux niveaux de référentiel GIT (local et distant).
Un référentiel GIT est plus compact qu'un référentiel SVN.

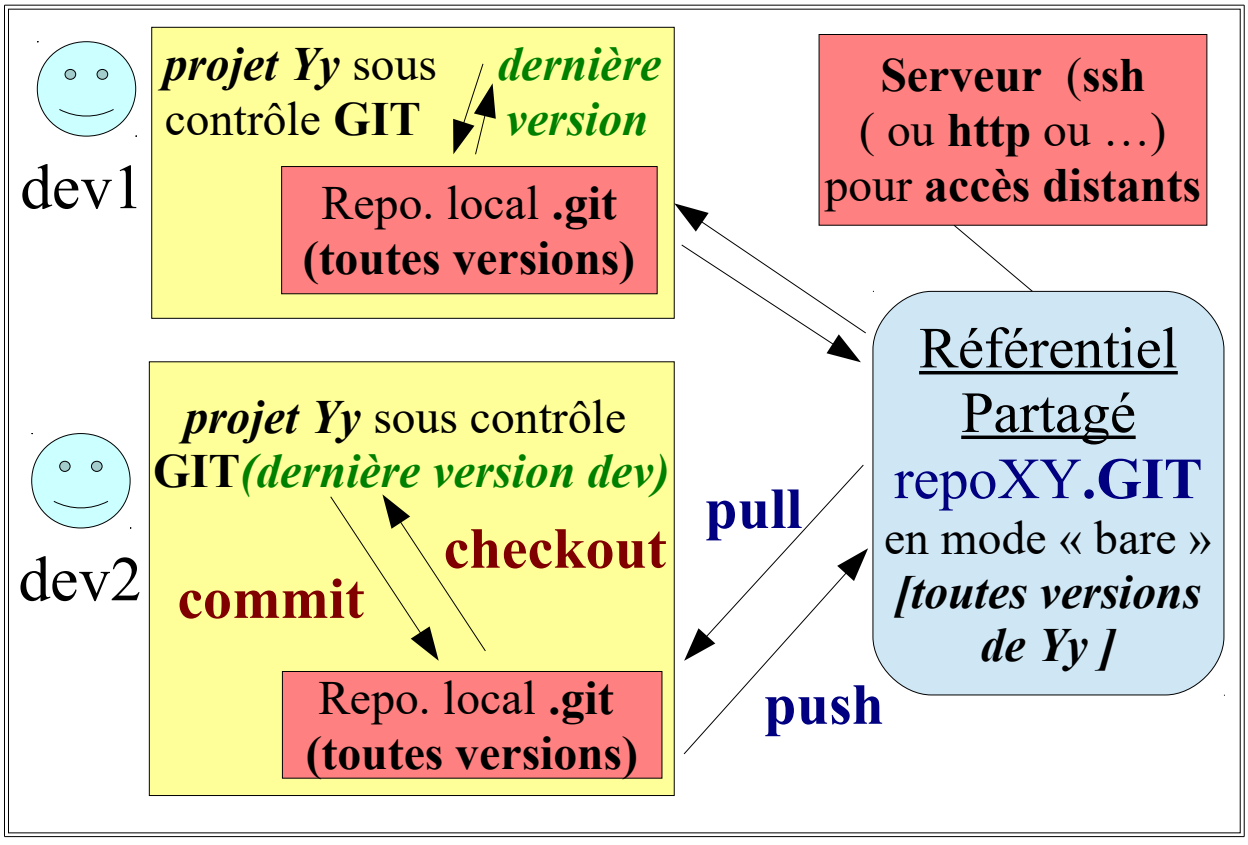
GIT a été conçu par **Linus Torvalds** (l'inventeur de **linux**) .

Un produit concurrent de GIT s'appelle « **Mercurial** » et offre à peu près les mêmes fonctionnalités.

1.1. Mode distribué de GIT

Dans un système « scm » centralisé (tel que CVS ou SVN) , le référentiel central comporte toutes les versions des fichiers et chaque développeur n'a (en général) sur son poste que les dernières versions des fichiers.

Dans un système « scm » distribué (tel que GIT ou Mercurial) , le référentiel central ne sert que pour échanger les modifications et chaque développeur a (potentiellement) sur son poste toutes les versions des fichiers.



En bref, les commandes «**commit** » et «**checkout**» de **GIT** permettent de gérer le référentiel **local** (propre à un certain développeur) et les commandes « **push** » et « **pull** » de **GIT** permettent d'effectuer des **synchronisations** avec le **référentiel partagé distant** .

2. Configuration locale de GIT

Installation de GIT sous linux :

sudo **apt-get install git-core**

Configuration locale:

git **config --global user.name** "Nom Prénom"
git **config --global user.email** "poweruser@ici_ou_la.fr"
#...

pour voir ce qui est configuré :
git **config --list**

3. Principales commandes de GIT (en mode local)

Commandes GIT (locales)	**Utilités**
git init	Initialise un référentiel local git (sous répertoire caché « .git ») au sein d'un projet neuf/originel.
git clone *url_referentiel_git*	Récupère une copie locale (sous le contrôle de GIT et avec toutes les versions des fichiers) d'un référentiel git existant (souvent distant)
git status **git diff** *fichier*	Affiche la liste des fichiers avec des changements (pas encore enregistrés par un commit) et git diff affiche les détails (lignes en + ou -) dans un certain fichier.
git add *liste_de_fichiers*	Ajoute un répertoire ou un fichier dans la liste des éléments qui seront pris en charge par git (lors du prochain commit).
git commit *-m message [-a]ou [liste_fichiers]*	Enregistre les derniers fichiers modifiés dans le référentiel git local (ceux précisés ou tous ceux ajoutés par *add* et affichés par *status* si -a)
git checkout *liste_de_fichiers*	Récupère les dernières versions depuis le référentiel local (sorte d'équivalent local du update de SVN , utile après un pull distant)
git **--help** git *cmde* **--help**	Obtention d'une aide (liste des commandes ou bien aide précise sur une commande)
git **log --stat** *ou git log -p*	Affiche l'historique des mises à jour

	-p : avec détails , --stat : résumé
git branch , **git checkout** *nomBranche*, **git merge**	Travailler (localement et …) sur des branches
git grep texte_a_rechercher	Recherche la liste des fichiers contenant un texte
git tag *NomTag IdCommit*	Associer un **tag** parlant(ex: ***v1.3***) à un id de commit .
git tag -l	Visualiser la liste des tags existants
git checkout tags/*NomTag*	Récupère la version identifiée par un tag

Exemples :

#initialisation

```
cd p1; git init
```

#affichage des éléments non enregistrés

```
cd p1; git status
```

→ *affiche:*

```
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#         modified:   src/f1.txt
#         modified:   src/f3-renamed.txt
no changes added to commit (use "git add" and/or "git commit -a")
```

commit all already tracked/added :

```
cd p1
#  -a  pour tous les fichiers listés dans git status
git commit -a  -m "my commit message"
```

#commit all (with all new and deleted) :

```
cd p1
git add pom.xml.txt   src/*
git status
# git commit gère tous les fichiers ajoutés (et supprimera de l'index ceux qui
# n'existent plus si option -a)
git commit -m "my commit message" -a
```

#checkout like local update

```
cd p1
git status
git checkout *
```

#historique des dernières mises à jour :

```
cd p1;  git log –stat
```

---> *affiche*:

```
commit 93446a0f2194089d83c941a63768f212eb96e0f8
Author: developpeur fou <moi@ici_ou_la.everywhere>
Date:   Wed Dec 12 18:36:40 2012 +0100
    my commit message
 pom.xml.txt          |   2 ++
 src/f1.txt           |   1 +
 src/f3-renamed.txt   |   1 +
 src/f4-renamed.txt   |   1 +
 src/p/pf2-renamed.txt |   2 ++
 5 files changed, 7 insertions(+)
```

4. Commandes de GIT pour le mode distant

Commandes GIT (mode distant)	Utilités
git init --bare	Initialisation d'un nouveau référentiel vide de type «nu» ou «serveur». (à alimenter par un push depuis un projet originel)
git clone --bare ***url_repo_existant***	Idem mais via un clonage d'un référentiel existant
git clone ***url_repo_sur_serveur.git***	Création d'une copie du projet sur un poste de développement (c'est à ce moment qu'est mémorisée l'url du référentiel « serveur » pour les futurs push et pull)
git pull	Rapatrie les dernières mises à jour du serveur distant (de référence) vers le référentiel local. (NB: *git pull* revient à déclencher les deux sous commandes *git fetch* et *git merge*)
git push	Envoie les dernières mises à jour vers le serveur distant (de référence) Attention: ***le push est irréversible*** et ***personne ne doit avoir effectuer un push depuis votre dernier pull !***
...	

Exemples:

#script de création d'un nouveau référentiel GIT (coté serveur) dans /var/scm/git ou ailleurs:

```
mkdir p0.git
cd p0.git
git init --bare
```

```
git  update-server-info
mv hooks/post-update.sample hooks/post-update

#nb www-data est le groupe de apache2
cd ..
sudo chgrp -R www-data p0.git

# ce repository initial et vide pourra être alimenté par un push depuis un projet "original"
# depuis ce projet original , on pourra lancer git config remote.p0.url http://localhost/git/p0.git
#                              puis   git push p0 master
echo "fin ?"; read fin
```

ou bien

```
# construira p1.git
git clone --bare  file:///home/formation/Bureau/tp/tmp-test-git/original/p1
cd p1.git
git update-server-info
```

#récupération d'une copie du projet sur un poste de développement

```
git clone http://localhost/git/p1.git
```

#pull from serv:

```
cd p1
git pull
```

#push to serv:

```
cd p1
git push
```

5. Gestion des branches avec GIT

Tout projet commence avec une seule branche «**master**» .

Commandes GIT (branches)	Utilités
git branch	Affiche la liste des branches et précise la branche courante (*) .
git branch *nomNouvelleBranche*	Créer une nouvelle branche (qui n'est pas automatiquement la courante)
git checkout *nomBrancheExistante*	Changement de branche (avec mise à jour « checkout » des fichiers pour refléter le changement de branche) .
git checkout *master* **git merge** *autreBranche_a_fusionner*	Modifie la branche courante (ici «master») en fusionnant le contenu d'une autre branche
git branch -d *ancienneBrancheAsupprimer*	Supprime une ancienne branche (avec -d : vérification préalable fusion, avec -D : pas de verif , pour forcer la perte d'une branche morte)

...	

6. Gérer plusieurs référentiels distants

s_list_remote_git_url.bat

```
git remote -v
pause
```

s_set_git_remote_origin.bat

```
git remote set-url origin Z:\TP\tp_angular1.git
git remote -v
pause
```

s_push_to_remote_origin.bat

```
git push -u origin master
pause
```

s_push_to_gitHub.bat

```
git remote add GitHubMyContribOrigin https://github.com/didier-mycontrib/tp_angular.git
REM didier-mycontrib / gh14.....sm..x / didier@d-defrance.fr
git push -u GitHubMyContribOrigin master
pause
```

7. Configuration d'accès distant à un référentiel Git

Si Répertoire "réseau" partagé (via NFS ou autre) , URL possible en file:///
(ex: file:///var/git/project.git)

7.1. Accès distant (non sécurisé) via git

URL de type **git://nomMachineOuDomaine/xxxx/projetYy.git**
où *xxx* est le chemin menant au référentiel git sur la machine "serveur" (ex: **/var/git/**)
(Alias possible dans .gitconfig)

7.2. Accès distant sécurisé via git+ssh

URL de type **ssh://user@hostXx/var/git/projectYy.git**

git+ssh est un tunneling sécurisé ssh pour le protocole git

Les clefs (publiques et privées) ssh sont placés dans le répertoire **$HOME/.ssh**
Celles ci se génèrent via la commande **ssh-keygen** .
Lors de la génération des clefs , un mot de passe (*passphrase*) à retenir est demandé .
Ce mot de passe peut éventuellement être vide (sécurité alors que via la clef publique) .
La **clef publique** (à envoyer par email ou ….) correspond au fichier ***id_dsa.pub*** (ou bien ***id_rsa.pub***) .

Si la partie "serveur" de ssh n'est pas encore installée , on peut alors lancer "**sudo apt-get install openssh-server**" puis éventuellement "***sudo service ssh start***' .
NB : sur certaines versions de Linux Ubuntu , la commande apt-get install ne fonctionne pas bien avec openssh-server et l'on peut dans ce cas installer alternativement openssh-server de la façon suivante :

1) télécharger le fichier ***openssh-server_5.9p1-5ubuntu1_i386.deb*** (via une recherche google)
2) lancer **sudo dpkg --install ./openssh-server_5.9p1-5ubuntu1_i386.deb**
3) redémarrer linux (ou …) pour que le service "ssh" soit activé

7.3. Accès distant en lecture seule via http (sans webdav)

Configurer un accès de apache2 vers un répertoire correspondant à un référentiel git et activer le hook "post-update" en renommant post-update.sample en post-update .

```
cd xy.git
mv hooks/post-update.sample hooks/post-update
```

7.4. Accès distant en lecture/browsing via gitweb

En configurant sur le poste serveur , l'extension "gitweb"(pour apache2 et git) , on peut alors parcourir toute l'arborescence d'un projet GIT via un simple navigateur.

http://localhost/**gitweb**/?p=p0.git

Activation et configuration du site web "gitweb":

```
cd /var/www;
sudo mkdir gitweb;
cd gitweb;
sudo cp /usr/share/gitweb/* . ;
sudo cp /usr/share/gitweb/static/* .
```

Il faut également fixer la variable **$projectroot = "/var/scm/git/"**
dans le fichier **/etc/gitweb.conf**

/etc/gitweb.conf

```
$projectroot="/home/formation/scm/git";
# directory to use for temp files
$git_temp="/tmp";
# html text to include at home page
$home_text="indextext.html";
# file with project list; by default, simply scan the projectroot dir.
$projects_list=$projectroot;
# stylesheet to use
# I took off the prefix / of the following path to put these files inside gitweb directory directly
$stylesheet="gitweb.css";
# logo to use
$logo="git-logo.png";
# the 'favicon'
$favicon="git-favicon.png";
```

D'autre part, le module " RewriteEngine" d'apache2 doit être activé.
Si ce n'est pas encore le cas, on l'active via la commande "**sudo a2enmod rewrite**"

Créer et configurer un nouveau fichier pour configurer gitweb sous apache2 **:**

/etc/apche2/conf.d/git.conf

```
...
<Directory /var/www/gitweb >
SetEnv  GITWEB_CONFIG  /etc/gitweb.conf
DirectoryIndex gitweb.cgi
```

```
Allow from all
AllowOverride all
Order allow,deny
Options +ExecCGI
AddHandler cgi-script .cgi
<Files gitweb.cgi>
  SetHandler cgi-script
</Files>
RewriteEngine on
RewriteRule ^[a-zA-Z0-9_-]+.git/?(\?.)?$ /gitweb.cgi%{REQUESTURI} [L,PT]
</Directory>
….
```

Redémarrage du service apache2:
service apache2 restart

7.5. Accès distant "rw" via http/https (webdav)

Activer les ***modules apache2*** **"dav"** , **"dav_fs"** et **"dav_lock"**

sudo a2enmod dav
sudo a2enmod dav_fs
sudo a2enmod dav_lock

Créer et configurer un nouveau fichier pour configurer git sous apache2 **:**

/etc/apche2/conf.d/git.conf

```
...
Alias /git /var/scm/git/

<Location /git>
    DAV on
    #AuthType Basic
    #AuthName "Git"
    #AuthUserFile /etc/apache2/dav_git.passwd
    #Require valid-user
</Location>
...
```

Redémarrage du service apache2:
service apache2 restart

+ si besoin paramétrage d'autres détails (sécurité , …) :

```
<Directory "/var/scm/git/">
Options Indexes FollowSymLinks MultiViews ExecCGI
#DirectoryIndex index
AllowOverride None
```

```
Order allow,deny
allow from all
</Directory>
```

Eventuel paramétrage (facultatif et très délicat) pour optimiser les transferts (par paquets) via HTTP:

```
# Git-Http-Backend (for smart http push , useful with egit )
SetEnv GIT_PROJECT_ROOT /var/scm/git/
SetEnv GIT_HTTP_EXPORT_ALL
#ScriptAlias /git/     /usr/lib/git-core/git-http-backend/
ScriptAliasMatch \
        "(?x)^/git/(.*/(HEAD | \
                        info/refs | \
                        objects/(info/[^/]+ | \
                                [0-9a-f]{2}/[0-9a-f]{38} | \
                                pack/pack-[0-9a-f]{40}\.(pack|idx)) | \
                        git-(upload|receive)-pack))$" \
                        "/usr/lib/git-core/git-http-backend/$1"

<LocationMatch "^/git/.*/git-receive-pack$">
        #AuthType Basic
        #AuthName "Git Access"
        #Require group committers
</LocationMatch>
```

8. Plugin eclipse pour GIT (EGIT)

Le plugin eclipse pour GIT s'appelle **EGIT** .

8.1. Actions basiques (commit , checkout , pull , push)

→ Se laisser guider par la perspective "GIT" et via le menu "Team"

8.2. Résolution de conflits

1) déclencher "**Team / pull**" pour récupérer (en tâche de fond) la dernière version (partagée / de

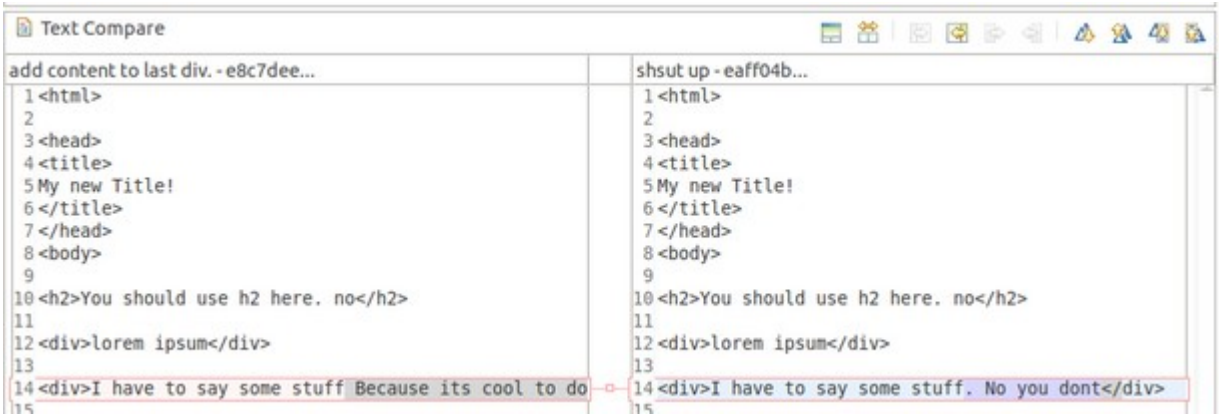
l'équipe). Le plugin EGIT va alors tenter un "***auto-merge***" ("git fetch FETCH_HEAD" suivi par "git merge").

2) En cas de conflit (non résoluble automatiquement) , les fichiers en conflit seront marqués d'un point rouge.

```
<<<<<<< HEAD
<div>I have to say some stuff Because its cool to do so.</div>
=======
<div>I have to say some stuff. No you dont</div>
>>>>>>> branch 'master' of /var/data/merge-issue.git
```

Sur chacun des fichiers en conflit , on pourra déclencher le *menu contextuel* "**Team / Merge tool**" . (laisser par défaut la configuration de "Merge Tool" : use HEAD).

3) ***Saisir , changer ou supprimer alors au moins un caractère dans la zone locale (à gauche) + save*** :



… au cas par cas ….

4) Déclencher le menu contextuel "**Team / add to index**" pour ajouter le fichier modifié dans la liste de ceux à gérer (staging).
5) Effectuer un "**Team / commit**" local .
6) Effectuer un "**Team / push to upstream** …" pour mettre à jour le référentiel distant/partagé .

ANNEXES

IV - Annexe – Bibliographie, Liens WEB + TP

1. Bibliographie et liens vers sites "internet"

2. TP