Git, au travers d'HTTP/HTTPS sur Debian

par Olivier Thiébaut

Date de publication : 08 juillet 2011

Dernière mise à jour :

Cet article explique la mise en oeuvre de Git en mode serveur central, au dessus de HTTP grâce au protocole WebDAV.

I - Objectif

Dans le cadre du développement d'applications, il existe des bonnes pratiques, en équipe ou seul, l'utilisation d'outils dit "collaboratifs" est un incontournable. Les logiciels de gestion de versions (Software Configuration Management, SCM) comme CVS ou Subversion font parties de ces indispensables dans notre travail. Cependant après Subversion (commande svn), qui a introduit des améliorations notables par rapport à CVS, de nouvelles générations de SCM décentralisés sont apparues.

Leur point fort, ne plus être obligé d'avoir un serveur de dépôt centralisé. Git et Mercurial proposent ainsi cette possiblité. Même s'ils offrent de travailler déconnecté d'un système central, voir sans, se pose quand même le problème de disposer d'un serveur permettant de tout fusionner dans un dépôt central. C'est ce qu'on vous propose de découvrir dans ce tutoriel.

On vous invite vivement à lire, **why git is better than x** , car il tente d'expliquer les points forts de Git. L'objectif n'est pas de prouver qu'il est le meilleur. Mais son approche est constructive et permet de mieux appréhender la partie **organisationnelle** que peut proposer Git dans une projet.

II - Introduction

Dans la premier partie, installation et découverte d'un dépôt **Git** centralisé sur un serveur. Nous allons utiliser le protocole DAV couplé au serveur web Apache pour les accès distants. En effet, dans le cadre d'infrastructures sécurisées, le passage par le port 80 (HTTP) ou 443 (HTTPS) ne pose aucun problème ;) .
Il existe la possibilité de créer un dépôt via **ssh://... (port 22) ou git://... (port 9418)**.
. Ce dernier est optimisé en terme de performance.

Dans la seconde partie, éclairage sur **Gitweb** qui est une solution logiciel pour visualiser, exposer les sources au travers d'un navigateur, mot clef, simplicité.

Dans la troisième partie, on vous propose un introduction sur l'utilisation de cette configuration à travers un scénario initiateur.
Enfin, nous verrons comment utiliser cette installation pour la déclarer dans **Redmine**, qui est une solution complète de suivi de projet. Nous ne verrons pas son installation, un prochain article, pourquoi pas ...

Détail des logiciels et versions

- **Système d'exploitation, Debian Lenny**
- **GIT, Debian Lenny**
- **Apache 2.0, Debian Lenny**
- **Gitweb, Debian Lenny**
- **Redmine, version 1.0.2 stable**

Nous allons nous concentrer sur les points essentiels, pour des adaptations à la marge, nous vous renvoyons vers la documentation **man** ou les sites officielles.

III - Installation de GIT sur le serveur

III-A - git-core paramétrage et dépôt principal

L'installation de Git et son application web sous Debian se fait par la commande

```
apt-get install git-core git-web
```

La distribution Debian gère très bien les dépendances logiciels.
Pour les autres distributions, on utilise **yum** ou **rpm**. Sinon retour aux sources et compilation (sic :=( ).
Dans toute la suite de notre "expérience" Git, nous allons centraliser tous les dépôts (projets) sous la racine **/opt/git**.

```
mkdir /opt/git
cd /opt/git
mkdir MonProjet.git
cd MonProjet.git
git init --bare
chown -R www-data:www-data .
chmod a+x hooks/post-update
```

Sous Debian, l'utilisateur qui est propriétaire des processus web est **www-data**. C'est à travers cet dernier, que les fichiers vont être lus ou écrit en HTTP. D'ou la commande **chown**. Pour les autres distributions, voir qui est est le propriétaire de **www**, qui est la racine web de votre serveur web.
Pour Git, l'option **--bare** correspond à la création d'un dépôt vide. La dernière ligne correspond au fait que lors d'un push sur le serveur central, la commande interne **update-server-info** doit être lancée. Pour qu'elle se produise de façon automatique, il faut changer les droits du script **post-update** (confère doc officielle de git en ligne). Un message d'erreur récurrent **PUT error: curl result=22, HTTP code=403**, lors de mes tests, s'est affiché pendant des **push** vers le dépôt central. Il ne faut pas hésiter à utiliser la commande **chown -R www-data:www-data /opt/git/MonProjet.git**.

III-B - Apache et DAV

WebDAV (Web-based Distributes Authoring and Versioning) est une extension du protocole HTTP RFC 4918. Il permet de récupérer, déposer ou synchroniser des fichiers distants facilement à travers le protocol HTTP et ou HTTPS. **le site officiel WebDAV** peut vous apporter d'autres informations sur le sujet. On doit vérifier qu'Apache, le serveur web supporte ce protocole.

```
a2enmod dav_fs
a2enmod dav
```

Si erreur il y a, il suffit d'installer le package adéquat.

```
#Pour les anciennes versions de Debian
apt-get install libapache-mod-dav
```

Pour permettre un accès WebDAV à la racine de notre projet MonProjet.git, nous créons un nouveau fichier en tant qu'utilisateur **root** :

```
touch /etc/apache2/sites-available/git-dav
```

Contenu a créé fichier git-dav

/etc/apache2/sites-available/git-dav
```
#On crée un alias http://mamachine/MonProjet.git
Alias /MonProjet.git /opt/git/MonProjet.git
#Déclaration des spécificités de l'alias précédent
<Location /MonProjet.git>
 #On authorise le protocole DAV
DAV on
#Type d'authentification
AuthType Basic
#AuthName
AuthName "Git, yop, pass ?"
#Fichier qui doit permettre l'authentification
AuthUserFile /etc/apache2/passwd.git
Require valid-user
</Location>
```

On doit activer le nouveau "site", avec l'instruction **a2ensite** soit :

```
#Utilisateur root
a2ensite git-dav
```

Il nous reste la déclaration de nos utilisateurs, avec création de mot de passe. pour l'ajout ou la modification du fichier passwd.git, faites un **man htpasswd**.

```
#Utilisateur root
htpasswd -c /etc/apache2/passwd.git <utilisateur>
```

Il faut systématiquement valider les étapes intermédiaires.
L'utilisation des outils **cadaver** ou **litmus** sont en mode console, sinon en mode graphique, avec **konqueror** et l'URL **webdav://...** Bien entendu, il existe moultes solutions.

```
#cadaver [-et] [-V] [-h] http://hostname[:port]/path
client>cadaver http://machine/MonProjet.git
Authentification required for Git tutoriel on server ...
client>Username: user
client>Password: motdepasse
dav:/tutotiel.git/> ls
Listing collection`/tutoriel.git/': succeeded.
Coll: branches ....
```

Sous windows XP, il faut exploiter le lien **Favoris réseau** -> clique droit -> **Connecter un lecteur réseau** dans la fenêtre, en bas il faut sélectionner **Ouvrir une session de stockage ou se connecter à un serveur distant**. En se laissant guider, il suffira de déclarer l'adresse en http://.../Monprojet.git/.

*Cela permet de tester le bon fonctionnement, cependant depuis le début, nous avons passé sous silence la notion de sécurité. Le protocole **HTTP** permet une mise au point simple (scan de trame réseau), mais pas "securisée" avec notamment le mot de passe en clair. Pour une solution d'entreprise, il faudra veiller à tout passer en **HTTPS**.*

Pour la partie cliente. lors de l'authentification distante avec **Git**. Le serveur web Apache demandera le login et le mot de passe, sous Linux, il faut crée un fichier **$HOME/.netrc** qui permettra le stockage de ces données.

```
olivier> cat > ~/.netrc
machine monserveur.com
login olivier
password monmotdepasse
^C
olivier> chmod 600 .netrc
```

Il faudra simplement faire attention d'utiliser un mot de passe qui n'est pas le compte de votre machine, car c'est stocké en clair. C'est moyen comme solution. Cette manipulation vous évitera de saisir le nom d'utilisateur et le mot de passe dans l'URL.

```
#Sans fichier .netrc
olivier> git clone http://user:motdepasse@machine/depot.git/
#Avec fichier .netrc
olivier> git clone http://machine/depot.git/
```

III-C - gitweb

L'application **gitweb** est un script cgi. Son installation par la commande **apt-get** doit être complétée par un paramétrage. Il est écrit en Perl, le site officiel de **Gitweb**. Pour avoir un brève description des fichiers, on peut lancer la commande :

```
dpkg -L gitweb
```

Les fichiers *.png, *.css et autres sont installés dans l'arborescence **/usr/share**. le binaire cgi sous **/usr/lib/cgi-bin/gitweb.cgi**. Il suffit de modifier le fichier **/etc/gitweb.conf**, pour faire pointer l'application **gitweb** sur le repertoire qui va héberger tous les dépôts git soit **/opt/git**.

/etc/apache2/site-available/git-dav

```
#path to git projects (<project.git>.git)
$projectroot = "/opt/git";
```

Le chemin de tous les projets Git sur le serveur pointeront sur **/opt/git**.
Il faut paramétrer le serveur web Apache pour prendre en compte ce script en rajoutant dans le fichier **git-dav**.

/etc/apache2/site-available/git-dav

```
Alias /gitweb.css /usr/share/gitweb/gitweb.css
Alias /git-logo.png /usr/share/gitweb/git-logo.png
Alias /git-favicon.png /usr/share/gitweb/git-favicon.png
#URL http://nomdeserveur/gitweb
ScriptAlias /gitweb /usr/lib/cgi-bin/gitweb.cgi
```

Pour visualiser l'ensemble de nos projets, on fait pointer notre navigateur à l'adresse **http://nomduserveur/gitweb**.

Visualisation partielle de Gitweb

Si vous cliquez sur votre projet, gitweb, donnera le détail de votre projet. Dans la barre inférieur, le texte **Unnamed repository; edit this file to name it for gitweb** est disgracieux. Editer le fichier **description** sous /opt/git/Monprojet.git/description pour changer ce texte.
Dans notre cas, on rajoute **Gitweb Tutoriel pour developpez.com**.

IV - prise en main de GIT

IV-A - Les commandes de base

Le tableau ci-dessous, vous donne une perspective des commandes de base, avec un lien sur les options. Ils en existent bien d'autres.

Commande	type de commande	Description	options
git status	local	Etat du dépôt local, en fonction de la branche	, lien
git init	local	initialise un nouveau dépôt	--bare dépôt vide , lien
git add	local	ajoute un fichier ou un répertoire	, lien
git rm	local	efface un fichier ou un répertoire	-r de façon récursive,
git commit	local	valide les changements dans le dépôt	-m ajout du commentaire , lien
git reset	local	réinitialisation à l'état du dernier commit	--hard HEAD (exemple) , lien
git show	local	montre les informations sur un objet	, lien
git log	local	montre l'hitorique des commits	git log > changelog, création d'un fichier de log , lien
git repack	local	réorganise les objects du dépôt	, lien
git prune	local	nettoyage de la base	utilisation couplée de git repack && git prune , lien
git clone	distant	clone un dépôt distant dans un nouveau répertoire local	protocoles git://..., ssh://..., http://..., https://... , lien
git remote	distant	gestion des dépôts distants	add ajout, rm efface ... , lien
git pull	distant	récupération du dépôt distant	on peut préciser la branche , lien
git push	distant	poussera le dépôt courant vers le dépôt distant (défaut "origin")	git push projet v1, pousse la branche v1 vers distant git push [nom-distant] [nom-de-branche] , lien
git fetch	distant	récupération des objets et refs distantes	lien

Ce tableau n'est pas exhaustif.

IV-B - Les autres solutions clientes

Elles peuvent se décliner en fonction du système d'exploitation.
Les unix Sous Linux, Unix ou BSD, il existe souvent un package git, sinon il existe la possiblité de compiler les sources. Les solutions graphiques :

- git-gui

- gitk

Sous windows :

- **Git for Windows**, émulateur Unix, complet
- **Tortoisegit**, client graphique non complet, ne supporte pas encore la notion de **submodule**
- **Cygwin**, émulateur Unix contenant un package Git

Enfin, pour les **macs**, on vous renvoie vers **le lien Git officiel sur les clients** . On ne doute pas, qu 'il existe encore d'autres solutions.

IV-C - Les fichiers de configuration

Comme tout SCM, il faut configurer en local son espace de travail pour que Git puisse retrouver votre nom, prénom, email et autres petits détails de conforts personnels.

```
git config --global user.name="olivier thiébaut"
git config --global user.email="momail@domain.fr
```

L'option **--global** va engendrer la création d'un fichier dans **$HOME/.gitconfig**. Sans l'option, il faut se placer dans un dépôt, les options seront donc mémoriser dans le fichier **MonDepot/.git/config**. Cette commande peut aussi servir de configuration à la création de raccourcis, quelques exemples :

```
git config --global alias.st=status
git config --global alias.ci=commit
git config --global alias.co=checkout
git config --global alias.br=branch
#Choix de l'éditeur, lors des commits, exemple vim
git config --global core.editor "vim"
```

IV-D - Cloner son premier dépôt en tant que développeur

L'objectif est de "cloner" un espace de développement et de s'initer à quelques commandes de base. On considère que le dépôt central est créé. Nous allons nous placer comme développeur, nous avons une URL distante de la forme, **http://nomduserveur/depot.git**, un nom d'utilisateur et un mot de passe fourni par l'administrateur du dépôt central. Examinons un cas pratique.

```
olivier> mkdir test
olivier> cd test
olivier> git init
olivier> git clone http://olivier:monmotdepasse@serveurdedepot/tutoriel.git tutoriel
olivier> cd tutoriel
olivier> git branch
* master
olivier> git remote
origin
olivier> git remote -v
origin http://olivier:monmotdepasse@serveurdedepot/tutoriel.git
olivier> git branch -a
* master
  origin/HEAD <- pointeur sur la branche active
  origin/master <- branche principale distante
  origin/develop <- branche de développement
  origin/bugs <- branche pour fixer les corrections
olivier> ls -a
. .. .git install.txt readme.txt
olivier> ls -a .git
.  branches description hooks info objects
.. config   HEAD        index logs refs
```

Il faut souligner, que lorsqu'on **clone** un projet son nom **distant** est par défaut **origin** et son nom **local** est **master**. Mais il est possible d'avoir un autre nom ou alias qui pourrait pointer sur le même serveur ou sur un autre serveur distant, voir le dépôt d'un autre collègue.
Ainsi, on ajoute un autre dépôt distant avec la commande suivante :

```
olivier> git remote add tuto http://gary:motdepasse@serveurdegary/tutoriel.git
olivier> git branch -a
```

IV-E - Ignorer des fichiers ou répertoires

Quel que soit le langage, il existe souvent un répertoire **log** et ou **cache** que nous ne voulons pas enregistrer dans le dépôt. Si nous voulons ne pas prendre en compte leur contenus respectifs, il faut créer à la racine du projet un fichier **.gitignores**.

```
cat > .gitignore
# Les ligne commençant par '#' sont des commentaires.
# Ignorer tous les fichiers nommés readme.txt
readme.txt
# Ignorer tous les fichiers zipper
*.zip
# à l'exception de projet.zip
!projet.zip
# Ignorer les objets et les archives
*.[oa]
# Ignorer le répertoire cache/ et log/
cache/
log/
# Exclure, la configuration de son éditeur favori, au hasard Netbeans
nbproject/
```

Mais on on peut aussi exclure fichiers et repertoires à l'intérieur de l'arborescence.

```
olivier>pwd
/home/olivier/projetgit/
olivier>ls -a
. .. .git install.txt readme.txt
olivier> mkdir src; cd src
olivier> touch a.txt b.txt c.txt <- On crée trois fichier que l on rempli
olivier> for line in `ls -A` ; do echo "Fichier $line" > $line ; done
olivier> echo "a.txt" > .gitignore <- On ignore le fichier a.txt
olivier> cd .. <- retour à la racine du projet
olivier> git add src <- On ajoute le repertoires et ses fichiers !
olivier> git status
# On branch develop
# Your branch is ahead of 'origin/develop' by 1 commit.
#
# Changes to be commited:
#   (Use "git reset HEAD <file>..." to unstage)
#
#  new file: src/.gitignore
#  new file: src/b.txt
#  new file: src/c.txt
```

On constate que le fichier **a.txt**, n'a pas été pris en compte. Il est cependant préférable de tout concentrer dans le fichier **.gitignore** à la racine du projet en précisant l'arborescence du fichier à exclure.

*Remarque importante, on prend en considération dans le **git add** le fichier **.gitignore** dans le dépôt.*

IV-F - Ajouter, supprimer, renommer et annuler

Pour ajouter fichiers et répertoires :

```
git add fichier.txt repertoire
```

Pour renommer, un fichier ou un repertoire :

```
git mv fichier_origine.xt fichier_new.txt
git mv src_origin/ src_new/
```

pour supprimer un fichier ou un répertoires de façon récursive :

```
git rm fichier.txt
git rm --r src/
```

Pour revenir à votre version initiale :

```
git reset --hard
```

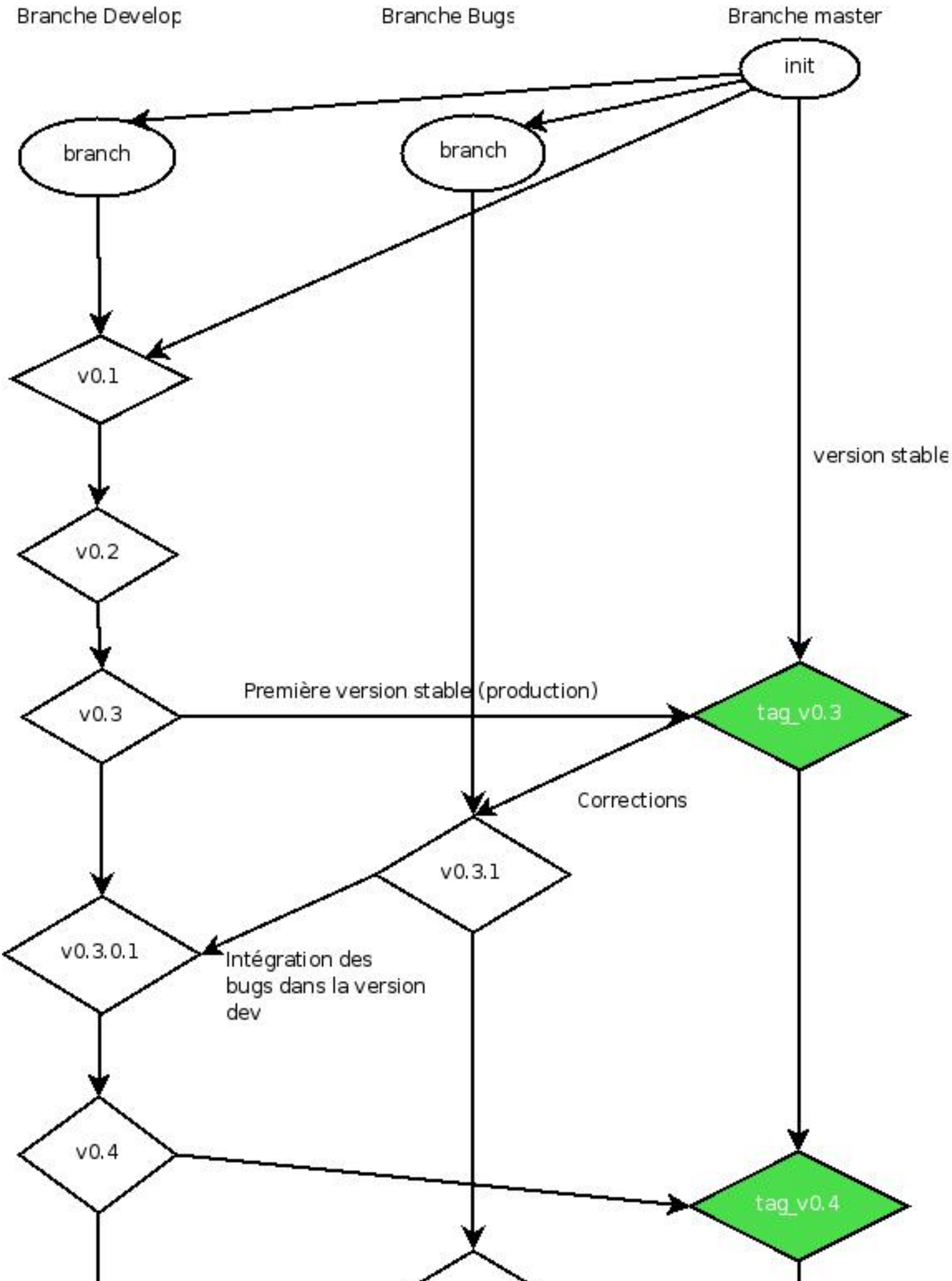
IV-G - Scénario de projet

IV-D-1 - Workflow

Dans la section III-C, dans l'image de gitweb, on s'aperçoit qu'il y a sur le serveur **trois branches**. Mais pour l'instant si vous avez suivi le tutoriel , il n'existe sur le dépôt qu'une branche dite **master** en local et **origin** sur le serveur distant. Nous devons aboutir à :

- **master** : la branche principale
- **develop** : la branche de développement
- **bugs** : la branche de correction des bugs

On se place dans un cycle de vie applicatif. Sur des architectures standards, il n'est pas rare d'avoir plusieurs branches qui cohéxistent. Il est fréquent de travailler sur une version de développement, fournir une version de recette pour valider les spécifications fonctionnelles de la maîtrise d'ouvrage et déployer une version en production.
Cela implique de maintenir plusieurs branches, le modèle que nous allons présenter est tirer de l'exemple du site **nvie.com**.


Branche Develop
Branche Bugs
Branche master
init
branch
branch
v0.1
version stable
v0.2
v0.3
Première version stable (production)
tag_v0.3
Corrections
v0.3.1
v0.3.0.1
Intégration des bugs dans la version dev
v0.4
tag_v0.4

On se situe sur le serveur, à la racine du dépôt principal. Nous allons créer les deux branches qui nous manquent. C'est à dire la branche **develop** qui sera servira la branche développement et **bugs** qui sera la branche d'intégration des correctifs.

```
#Il faut être www-data donc su, puis su www-data, car il n'a pas de compte
www-data@serveur> cd /opt/git/MonProjet.git
www-data@serveur> git branch develop
www-data@serveur> git branch bugs
www-data@serveur> git branch -l
 bugs
 develop
*master
```

IV-D-3 - Mise en pratique

Le scénario est le suivant, nous avons à disposition une équipe. Nous prenons volontairement un exemple concret pour vous permettre d'assimiler les rôles et les concepts.
Sur le serveur de l'entreprise, existe notre projet MonProjet.git, avec trois branches.
Les acteurs :

Nom	Rôle	Symbole
bob	chef de projet	CP
gary	développeur	DEV
patrick	développeur	DEV
carlos	admministrateur réseau	ADR

bob décide de rappatrier le projet, avec toutes les branches, il fait une modification sans s'en rendre compte sur la branche **master**. A chaque fois que l'on veut changer de branche, il faut faire un **checkout**. Ce qui est intéressant, c'est la ligne

```
git branch --track develop origin/develop
```

On fait une copie locale de la branche distante "origin/develop", avec le nom local "develop". Ce qui va permettre de travailler dessus avec un **checkout**, puis de mettre à jour le version distante avec un **git push**. Effectivement, la branche locale "develop" est reliée à celle distante "origin/develop".

```
olivier>~/test$ git clone http://www.xxxxxxxx.com/tutoriel.git
Initialized empty Git repository in /home/olivier/test/tutoriel/.git/
Getting alternates list for http://www.xxxxxxxx.com/tutoriel.git
Getting pack list for http://www.xxxxxxxxx.com/tutoriel.git
Getting index for pack 40e0b475f53287da14ba2a0b2de79083ba3be80b
Getting pack 40e0b475f53287da14ba2a0b2de79083ba3be80b
 which contains 95cef2dcf9e5c5f129b49b210e02bbe138aa63ed
walk 95cef2dcf9e5c5f129b49b210e02bbe138aa63ed
walk 34e3771587e60ba5d01819f2c8deec70fd977061
walk 658f9e8e7b4d1220e14e12513a78c0564d729af6
walk f79c548b68b13bbdeca4185bf8113f8146aaa1ac
walk 9eb5adb410b4b18624aed47cef18110957c9429f
walk 6b394c1ff73dba1998d5a31f9c00250a9fec3809
olivier>cd tutoriel/
olivier>~/test/tutoriel$ git branch -l
* master
olivier>~/test/tutoriel$ git branch -r
  origin/HEAD
  origin/bugs
  origin/develop
  origin/master
olivier>~/test/tutoriel$ git branch --track develop origin/develop
Branch develop set up to track remote branch refs/remotes/origin/develop.
olivier>~/test/tutoriel$ vi install.txt
olivier>~/test/tutoriel$ git commit -a
```

```
Created commit aae35a4: modifications
 1 files changed, 1 insertions(+), 1 deletions(-)
olivier>~/test/tutoriel$ git push
Fetching remote heads...
  refs/
  refs/heads/
  refs/tags/
'refs/heads/develop': up-to-date
updating 'refs/heads/master'
  from 95cef2dcf9e5c5f129b49b210e02bbe138aa63ed
  to   aae35a4a9e9258bad354e548747032e4364e8ad0
    sending 3 objects
    done
Updating remote server info
olivier>~/test/tutoriel$ git push develop
fatal: 'develop': unable to chdir or not a git archive
fatal: The remote end hung up unexpectedly
olivier>~/test/tutoriel$ git branch -a
  develop
* master
  origin/HEAD
  origin/bugs
  origin/develop
  origin/master
olivier>~/test/tutoriel$ git checkout develop
Switched to branch "develop"
olivier>~/test/tutoriel$ vi install.txt
olivier>~/test/tutoriel$ git merge master
Updating 95cef2d..aae35a4
Fast forward
 install.txt |    2 +-
 1 files changed, 1 insertions(+), 1 deletions(-)
olivier>~/test/tutoriel$ vi install.txt
olivier>~/test/tutoriel$ git push
Fetching remote heads...
  refs/
  refs/heads/
  refs/tags/
updating 'refs/heads/develop'
  from 95cef2dcf9e5c5f129b49b210e02bbe138aa63ed
  to   aae35a4a9e9258bad354e548747032e4364e8ad0
    done
'refs/heads/master': up-to-date
Updating remote server info
```

Ce début de travail se repète pour tous les acteurs du projet. Chaqu'un récupère un copie afin de pouvoir travailler dessus. Il faut en règle général "éditer" un document sur les bonnes pratiques du dépôt central. Une sorte de **manager**, doit orchestrer le fonctionnement du dépôt central.

Utilisateur	Action	machine	Description
ADR	mkdir /opt/git /opt/git/projet.git cd /opt/git/projet git init --bare voir III	serveur	initialisation d'un projet vide
CP	git clone http://cp:cppassword@serveur/projet.git	serveur	initialisation d'un projet vide

V - Redmine, intégration de Git

Redmine est outil de gestion de projet l'objectif n'est pas de voir son installation, mais de montrer le couplage possible entre différents outils libres. La flexibilité de Git, et sa possiblité d'exister de façon couper du serveur central va bien nous servir.
Dans un cadre idéal, budget et sécurité, Git et Redmine sont installés sur deux machines différentes, le problème étant que Redmine ne peut déclarer qu'un dépôt sur le même serveur.

Il suffit de cloner le serveur principal, et de mettre à jour régulièrement le dépôt avec une tâche **cron**.

VI - Passage au HTTPS

Nous allons passer à la version sécurisée de notre dispositif Git en accès **HTTPS**. Si vous êtes dans une configuration ou votre certificat est "auto-signé", vous devrez rajouter la variable d'environnement suivante :

```
env GIT_SSL_NO_VERIFY=true
```

Ce qui donne pour une un clone de projet en HTTPS

```
env GIT_SSL_NO_VERIFY=true git clone https://monserveur/mondepot.git/
```

L'installation du mode **SSL** se fait sans difficulté sous Debian.
Il crée normalament un fichier certificat auto-signé. Il vous demandera différents paramètres à enregistrer DNS, nom, pays etc ... . Cependant nous allons voir la manipulation pour le faire or refaire nous même. Du sur mesure.

```
root>a2enmod ssl
See /usr/share/doc/apache2.2-common/README.Debian.gz on how to configure SSL and create self-signed
certificates.
...
root> mkdir /etc/apache2/ssl
root> openssl req $@ -new -x509 -days 365 -nodes -out /etc/apache2/ssl/apache.pem -keyout /etc/
apache2/ssl/apache.key
root> chmod 600 /etc/apache2/ssl/*
root> touch /etc/apache2/sites-available/securessl
```

Le fichier securessl, va contenir les informations du site SSL du dépôt central.

/etc/apache2/sites-available/securessl

```
#On crée un alias https://mamachine/*.git
<VirtualHost *:443>
 ServerAdmin webmaster@localhost
 Document root /opt/git

 SSL Engine on
 SSLCertificateFile /etc/apache2/ssl/apache.pem
 SSLCertificateKeyFile /etc/apache2/ssl/apache.key

AliasMatch ^/git/(.*) /opt/git/$1
#Déclaration des spécificités de l'alias précédent
<Location /git>
  #On authorise le protocole DAV
 DAV on
 #Type d'authentification
 AuthType Basic
 #AuthName
 AuthName "Git, yop, pass ?"
 #Fichier qui doit permettre l'authentification
 AuthUserFile /etc/apache2/passwd.git
 Require valid-user
</Location>
</VirtualHost>
```

Il faut prendre en compte le nouveau site, redémarrer Apache et tester

```
root> a2ensite securessl
root> apachectl restart
```

VII - Conclusion

Ce tutoriel représente une synthèse de mon travail, l'objectif était de regrouper dans un article toutes les problématiques ou questions qui me sont passés par la tête. J'ai essayé d'avoir une démarche pédagogique. Sachant que c'est un vaste sujet et que je n'ai abordé qu'un partie.

Cela se résume à démystifier la complexité au profit d'un évident sens pratique du concepteur (Linus Torvalds, Linux). Git n'est pas "compliqué", comme on peut l'entendre dire, mais **riche** en fonctionnalités, le côté décentralisé, perturbe, mais cela est d'un confort évident. Là ou cela devient puissant, c'est la facilité avec laquelle il est aisé de gérer branches et tags de versions d'un projet.

Là ou les autres posent parfois problèmes (notions avancées), ce dernier semble beaucoup plus facile d'utilisation.

VIII - Ressources et remerciements

Les ressources :

- **setup-git-server-over-http.txt de Rutger Nijlunsing**
- **Git documentation**
- **Synchronisation Redmine et Git**
- **Modèles de branche Git (merci à ovh)**
- **Git the basics (en ligne, à regarder)**