Centos: Serveur GIT sous Centos

Table of Contents

Mise en oeuvre du serveur GIT

Prérequis

Le serveur doit avoir :

- Un serveur SSH fonctionnel et correctement configuré (sécurisé) ;
- Un serveur HTTP(S) fonctionnel et correctement configuré (Apache ou Nginx).

Installation de GIT

Sur le serveur

Installer GIT-core sur le serveur

Seul le paquet « git-core » est nécessaire côté serveur.

```
# sur le serveur
$ yum install git-core
```

Pour rendre le serveur accessible via une adresse en git://, il faut également installer git-daemon

```
# sur le serveur
$ yum install git-daemon
```

Sur le client

Installer Git-core, par exemple sur debian:

```
$ sudo apt-get install git-core
```

Configurer les paramètres de Git (pour l'utilisateur root)

Pour éviter toute erreur de commit plus tard configurer l'utilisateur du client (remplacer 'gituser' et 'gituser@example.com' par ceux du compte réel)

```
$ git config --global user.name "gituser"
$ git config --global user.email "gittuser@example.com"
```

Git utilise la bibliothèque curl pour faire des opérations réseau sur HTTP, selon la séquence suivante :

- Recherche des crédits (mot de passe, utilisateur) dans le fichier .netrc file;
- Recherche de l'Hostname dans le cahe DNS
- Connection via le proxy système

Si le proxy bloque les requètes cURL, désactiver le proxy système pour les connexions de GIT en local :

```
$ git config --global http.proxy ""
```

Configuration

Serveur

Créer un dépôt (vide ou non) à destination du serveur

Concrètement, un dépôt GIT côté serveur est un dossier, nommé par convention my_repo.git, auquel on accède par la suite par SSH, HTTP(S), ... via le client git côté utilisateur.

Plusieurs solutions pour créer ce dossier : soit directement côté serveur (dépôt vide), soit côté client (dépôt vide ou non) qu'il faudra envoyer sur le serveur.

Créer un dépôt vide, côté serveur

Solution la plus simple et rapide : créer un dépôt vide sur le serveur. Ensuite, en local, après avoir fait un clone de ce dépôt vide, on y ajoutera au fur et à mesure les nouveaux fichiers. Après un (ou plusieurs) commit(s) en local, un push enverra le travail vers le serveur (c'est à dire les différentes versions commitées).

Si on déjà un dépôt GIT en local, avec un ensemble de révisions (ie de commits), et qu'on copie/colle les fichiers (tous sauf le dossier .git) dans le dossier du nouveau dépôt vide cloné, on perd les révisions antérieures ! Il est dans ce cas préférable d'utiliser la solution suivante

Créer un dépôt en local.

Sur le serveur, dans le dossier qui contient vos dépôts, créer un dossier nommé my_repo.git :

```
# sur le serveur
user@serveur:~$ mkdir -p /repos/my_repo.git
user@serveur:~$ cd repos/my_repo.git
```

Initialiser ce dépôt

Initialiser le nouveau dépôt Git vide dans /home/user/repos/my_repo.git/

```
# sur le serveur
user@serveur:~/repos/my_repo.git$ git init --bare
```

Explications :

- git init est la commande d'initialisation d'un dépôt GIT qui est utilisée en local (sur un client) et qui ne doit pas être utilisée seule pour initialiser un dépôt sur le serveur ;
- git init –bare permet d'initialiser un dépôt GIT sur le serveur : cela créé un ensemble de fichiers (config, description, HEAD) et de dossiers (branches, looks, info, objects, refs) utiles à GIT ;
- L'option –shared (facultative) permet de configurer automatiquement les droits des fichiers et dossiers du dépôt en écriture au groupe

Récupérer le dépôt sur le client en SSH

```
# sur le client, on clone le dépôt :
user@client:~$ git clone user@serveur:repos/my_repo.git .
user@client:~$ cd my_repo
# sur le client, on ajoute des fichiers :
user@client:~/my_repo$ cp path/to/my_file.txt .
# sur le client, on ajoute/commit les changements :
user@client:~/my_repo$ git add .
user@client:~/my_repo$ git commit -m "1er commit"
# sur le client, on push ce commit :
user@client:~/my_repo$ git push
```

Créer un dépôt vide, côté client

Ce dépôt sera à destination du serveur

Créer un dossier pour le dépôt

```
    # sur le client :
    user@client:~$ mkdir my_repo
    user@client:~$ cd my_repo/
```

Initialiser ce dépôt avec GIT(il faut donc que GIT soit installé sur la machine)

```
    # toujours dans le dossier my_repo
    user@client:~/my_repo$ git init
    Dépôt Git vide initialisé dans /home/user/my_repo/.git/
```

Dans ce cas, l'option –shared est inutile, car lors de la création de la version bare (ci-dessous), les droits ne sont pas conservés. Nous corrigerons les permissions manuellement plus tard du côté serveur.

Créer une version bare du dépôt my_repo.

Un bare repository est un dépôt qui ne contient que le nécessaire pour être placé sur le serveur (en l'occurence, qui ne contient pas les fichiers relatifs à la version de travail (ie working directory), mais qui grossièrement ne contient que le contenu du dossier .git à la racine du dépôt initialisé précédemment :

```
    # aller à la racine du dossier my_repo
    user@client:~/my_repo$ cd ..
    user@client:~$ git clone --bare my_repo/ my_repo.git
    Clonage dans le dépôt nu 'my_repo.git'
    warning: Vous semblez avoir cloné un dépôt vide.
    fait.
    user@client:~$ ls -l
```

```
    my_repo.git #la version 'bare', que l'on va placer sur le serveur
    my_repo #le dossier qui contient le dépôt que l'on a créé précédemment,
et qui ne nous intéresse plus, donc :
    user@client:~$ rm -rf my_repo
```

Déplacer la version bare du dépôt sur le serveur

```
    # exemple de transfert avec SCP :
    user@client:~$ scp -r my_repo.git  user@serveur:repos/
```

Si besoin, définir correctement le propriétaire, groupe et droits du dossier my_repo.git sur le serveur.

Push du dépôt depuis le client en SSH

```
    # sur le client, on clone le dépôt :
    user@client:~$ git clone user@serveur:repos/my_repo.git .
    user@client:~$ cd my_repo
    # sur le client, on ajoute des fichiers :
    user@client:~/my_repo$ cp path/to/my_file.txt .
    # sur le client, on ajoute/commit les changements :
    user@client:~/my_repo$ git add .
    user@client:~/my_repo$ git commit -m "1er commit"
    # sur le client, on push ce commit :
    user@client:~/my_repo$ git pull
```

Envoyer un dépôt local existant à destination du serveur

Supposons que vous ayez un dépôt GIT existant en local (/home/user/my_repo/). Vous avez dans le passé initialisé ce dossier, puis réalisé une série de commit.

Créer une version bare du dépôt my_repo.

Un bare repository est un dépôt qui ne contient que le nécessaire pour être placé sur le serveur (en l'occurence, qui ne contient pas les fichiers relatifs à la version de travail (ie working directory), mais qui grossièrement ne contient que le contenu du dossier .git à la racine du dépôt initialisé précédemment :

```
    # aller à la racine du dossier my_repo
    user@client:~/my_repo$ cd ..
    user@client:~$ git clone --bare my_repo/ my_repo.git
    Clonage dans le dépôt nu 'my_repo.git'
    fait.
    user@client:~$ ls -l
    my_repo.git  #la version 'bare', que l'on va placer sur le serveur
    my_repo      #le dossier qui contient le dépôt que l'on a créé
précédemment, et qui ne nous intéresse plus, donc :
    user@client:~$ rm -rf my_repo
```

Déplacer la version bare du dépôt sur le serveur

```
    # exemple de transfert avec SCP :
    user@client:~$ scp -r my_repo.git  user@serveur:repos/
```

Si besoin, définissez correctement le propriétaire, groupe et droits du dossier my_repo.git sur le serveur.

Push du dépôt depuis le client en SSH

```
    # sur le client, on clone le dépôt :
    user@client:~$ git clone user@serveur:repos/my_repo.git .
    user@client:~$ cd my_repo
    # sur le client, on ajoute des fichiers :
    user@client:~/my_repo$ cp path/to/my_file.txt .
    # sur le client, on ajoute/commit les changements :
    user@client:~/my_repo$ git add .
    user@client:~/my_repo$ git commit -m "1er commit"
    # sur le client, on push ce commit :
    user@client:~/my_repo$ git push
```

Activer le protocole git

Pour rendre accessible git via son protocole (git://) configurer le fichier suivant `/etc/sv/git-daemon/run`:

```
    #!/bin/sh
    exec 2>&1
    echo 'git-daemon starting.'
    exec chpst -ugitdaemon \
      "$(git --exec-path)"/git-daemon --verbose --base-path=/var/cache
/var/cache/gi
```

Ensuite pour chacun de vos projets que l'on veut rendre accessible depuis l'extérieur, il faudra placer un fichier 'git-daemon-export-ok' :

```
touch /var/cache/git/myproject.git/git-daemon-export-ok
```

Relancer le deamon et c'est maintenant accessible sur le port 9418.

Client

S'assurer que l'utilisateur puisse commiter

```
    $ git config --global user.email "clientuser@exemple.fr"
    $ git config --global user.name "clientuser"
```

Ce qui devrait donner ceci dans le fichier `~/.gitconfig`:

```
    [user]
       email = clientuser@exemple.fr
       name = clientuser
```

Pour utiliser des identifiants différents pour un repository différent, il faudra retirer **global** (en se plaçant dans le repository en question) :

```
git config user.email "clientuser@exemple.fr"
git config user.name "clientuser"
```

Récupérer le projet sur le serveur git

Récupérer le projet sur le serveur git avec ss (mais git sait également fonctionner sur rsync, http, https et git-daemon).

Avec SSH

```
git clone ssh://username@host/var/cache/git/myproject

    Initialized empty Git repository in /var/cache/git/myproject/.git/
    Password:
    remote: Counting objects: 1, done.
    remote: Total 1 (delta 0), reused 0 (delta 0)
    Receiving objects: 100% (1/1), done.
```

Pour les mises à jour du dépôt, il faut utiliser l'option push. (Les modifications devront être effectuées en locale)

```
git push ssh://username@host/var/cache/git/myproject.git
```

ou

```
git push ssh://username@host/var/cache/git/myproject.git master
```

'master' ici correspond au nom de la branche qui nous intéresse (voir plus bas pour l'utilisation des branches).

Pour ajouter un fichier à un dépôt SSH :

```
git remote add test ssh://username@host/var/cache/git/myproject.git
git commit -a
git push test
```

A l'avenir, afin de renouveler la partie SSH des commandes, ajouter ceci dans le fichier `~/.gitconfig` :

```
   [remote "myproject"]
      url = ssh://username@host/var/cache/git/myproject/
```

Avec Git-daemon

Git refusera de se synchroniser si le dossier en question ne contient pas un fichier appeller 'git-daemon-export-ok'. Une fois ce fichier créer, le dossier en question sera accessible par tout le monde :

```
git clone git://serveur.git/git/myproject
```

ou

```
git clone git://serveur.git/git/myproject.git
```

Nous pouvons utiliser les options suivantes :

- **–export-all** : cette option n'oblige plus à créer le fichier 'git-daemon-export-ok'
- **–user-path=gitexport** : cette option va permettre d'utiliser des URL sur les dossiers d'accueil des utilisateurs. Ainsi `%%git://deimos-laptop/~deimos/myproject.git` va pointer sur `/home/deimos/gitexport/myproject.git%%`

Avec HTTP (avec Nginx)

Configurer `/etc/nginx/sites-available/git.deimos.fr`

```
    server {
        listen 80;
        listen 443 ssl;
        ssl_certificate /etc/nginx/ssl/deimos.fr/server-unified.crt;
        ssl_certificate_key /etc/nginx/ssl/deimos.fr/server.key;
        ssl_session_timeout 5m;
        server_name git.deimos.fr;
        root /usr/share/gitweb/;
        access_log /var/log/nginx/git.deimos.fr_access.log;
```

```
        error_log /var/log/nginx/git.deimos.fr_error.log;
        index gitweb.cgi;
        # Drop config
        include drop.conf;
        # Git over https
            location /git/ {
            alias /var/cache/git/;
            if ($scheme = http) {
                rewrite ^ https://$host$request_uri permanent;
            }
        }
        # Gitweb
        location ~ gitweb\.cgi {
            fastcgi_cache mycache;
            fastcgi_cache_key $request_method$host$request_uri;
            fastcgi_cache_valid any 1h;
            include fastcgi_params;
            fastcgi_pass  unix:/run/fcgiwrap.socket;
        }
    }
```

Git over https qui fonctionne (le http est redirigé vers https) et giitweb également puisque tout ce qui est gitweb.cgi est matché. Maintenant, pour la partie git, il faut devoir autoriser les repository que nous voulons. Pour celà, il va falloir renommer un fichier dans le repository et lancer une commande :

```
cd /var/cache/git/myrepo.git
hooks/post-update{.sample,}
su - www-data -c 'cd /var/cache/git/myrepo.git && /usr/lib/git-core/git-
update-server-info'
```

Remplacer www-data par l'utilisateur qui a les droits sur le repository. Utiliser www-data pour que ce soit nginx qui ait les droits. Ensuite, cloner :

```
git clone http://www.deimos.fr/git/deimosfr.git deimosfr
```

Configurer un proxy

Pour passer à travers un proxy sur Git:

```
git config --global http.proxy http://proxy:8080
```

Pour vérifier les paramètres actuels :

```
git config --get http.proxy
```

Pour retirer cette configuration :

```
git config --global --unset http.proxy
```

Gestion des utilisateurs

Solution client/serveur SSH mono-utilisateur

Si on a pas besoin de partager l'accès du dépôt avec d'autres utilisateurs.

Sur le serveur, créer logiquement dans le /home/user un dossier nommé repos (attention aux droits si le home est « ouvert »).

Il suffit ensuite de placer la version bare du dépôt (exemples ci-dessus avec SCP) dans ce dossier /home/user/repos.

Exemple d'utilisation :

```
# sur le client, on clone le dépôt :
user@client:~$ git clone user@serveur:repos/my_repo.git .
user@client:~$ cd my_repo
# sur le client, on ajoute des fichiers :
user@client:~/my_repo$ cp path/to/my_file.txt .
# sur le client, on ajoute/commit les changements :
user@client:~/my_repo$ git add .
user@client:~/my_repo$ git commit -m "1er commit"
# sur le client, on push ce commit :
user@client:~/my_repo$ git push
```

Par la suite, en local, à chaque git push, git pull (ou autre commande nécessitant une connexion au dépôt stocké sur le serveur), le login/pwd sera demandé, à moins de configurer SSH pour utiliser votre clé SSH (sur le serveur, .ssh/authorized_keys contient votre clé publique)

Solution client/serveur SSH multi-utilisateurs

Pour travailler à plusieurs sur le même dépôt stocké sur le serveur, il est plus interessant de créer un utilisateur et un groupe git sur le serveur, pour gérer les accès SSH via /home/git/.ssh/authorized_keys

Sur le serveur, créer un utilisateur git :

```
# sur le serveur
root@serveur:$ adduser git
Ajout de l'utilisateur git ...
Ajout du nouveau groupe git ...
Ajout du nouvel utilisateur git avec le groupe git ...
Création du répertoire personnel /home/git ...
Copie des fichiers depuis /etc/skel ...
```

```
Entrez le nouveau mot de passe UNIX : ******
Retapez le nouveau mot de passe UNIX : ******
passwd: le mot de passe a été mis à jour avec succès
```

Sur le serveur, se connecter avec le compte git :

```
# sur le serveur
root@serveur:$ su git
```

Sur le serveur, connecté en tant que git, créer un dossier racine pour les dépôts, et une éventuelle arborescence :

```
# sur le serveur
git@serveur:$ mkdir repo
```

avec cette solution, si on crée un dépôt vide, l'utilisation de l'option –shared lors de la commande git clone –bare a pour effet de créer un dépôt partagé en écriture aux membres du groupe git.

```
# sur le serveur
git@serveur:~$ mkdir -p repos/my_repo.git
git@serveur:~$ cd repos/my_repo.git
git@serveur:~/repos/my_repo.git$ git init --bare --shared
Dépôt Git *partagé* vide initialisé dans /home/git/repos/my_repo.git/
git@serveur:~/repos/my_repo.git$ ls -l
total 32K
drwxrwsr-x 2 git git branches
-rwxrw-r-- 1 git git config
-rw-rw-r-- 1 git git description
-rw-rw-r-- 1 git git HEAD
drwxrwsr-x 2 git git hooks
drwxrwsr-x 2 git git info
drwxrwsr-x 4 git git objects
drwxrwsr-x 4 git git refs
```

Sans cette option, les membres du groupe git ont juste accès en lecture.

```
# sur le serveur
git@serveur:~/repos/my_repo.git$ git init --bare
Dépôt Git vide initialisé dans /home/git/repos/my_repo.git/
git@serveur:~/repos/my_repo.git$ ls -l
total 32
drwxr-sr-x 2 git git branches
-rwxr--r-- 1 git git config
```

```
    -rw-r--r-- 1 git git description
    -rw-r--r-- 1 git git HEAD
    drwxr-sr-x 2 git git hooks
    drwxr-sr-x 2 git git info
    drwxr-sr-x 4 git git objects
    drwxr-sr-x 4 git git refs
```

Gestion des accès au serveur

Plutôt que de communiquer le login/mot de passe du compte git aux autres utilisateurs, il est préférable d'utiliser la fonctionnalité d'authentification par clé de SSH : ajoutez la clé publique des utilisateurs au fichier `/home/git/.ssh/authorized_keys`

Une autre solution consiste à créer un compte pour chaque utilisateur sur le serveur, et d'ajouter ces utilisateurs au groupe git, en ajustant bien sûr les droits lecture/écriture/exécution du dossier (et sous-dossiers) /home/git/repos.

Interfaces graphiques

L'environnement natif de Git est le terminal. Les nouvelles fonctionnalités y apparaissent en premier. Mais le texte pur n'est pas toujours le meilleur choix pour toutes les tâches ; quelques fois, une représentation visuelle est préférable et certains utilisateurs sont beaucoup plus à l'aise avec une interface pointer-cliquer.

Il faut noter qu'il n'y a rien que les outils graphiques ne fassent que l'interface en ligne de commande ne puisse faire ; la ligne de commande reste l'interface qui donne le plus de puissance et de contrôle sur les dépôts.

Côté clients

Git, fournit des outils visuels, gitk et git-gui.

gitk

gitk est l'outil de visualisation graphique d'historique. C'est interface GUI puissante par-dessus git log et git grep. C'est l'outil à utiliser pour trouver un événement passé ou de visualiser l'historique des projets.

Gitk est à invoquer depuis la ligne de commande. Se positionner simplement dans le dépôt Git et taper :

```
gitk [options de git log]
```

Gitk accepte de nombreuses options de ligne de commande, dont la plupart sont passées directement à la commande git log sous-jacente. L'une des plus intéressantes est probablement d'ajouter l'option –all qui indique à gitk de montrer tous les commits joignables depuis n'importe quelle référence, et pas seulement HEAD.

Dans la partie supérieure, une zone ressemble à la sortie de git log –graph. Chaque point représente un commit, les lignes représentent les liens de parenté et les références apparaissent dans des rectangles colorés. Le point jaune représente HEAD et le point rouge représente les modifications qui ne sont pas validées. Dans la partie basse, on visualise le commit sélectionné : les commentaires et le patch sur la gauche et une vue en résumé sur la droite. Au milieu se trouve un ensemble de composants graphiques utilisés pour rechercher dans l'historique.

git-gui

git-gui, par contre est un outil permettant de ciseler les commits. Lui aussi est à invoquer en ligne de commande :

```
git gui
```

Sur la gauche, il y a l'index ; les modifications non indexées sont en haut, les modifications indexées en bas. Vous pouvez déplacer des fichiers entiers entre les deux états en cliquant sur leurs icônes ou vous pouvez sélectionner un fichier à visualiser en cliquant sur son nom.

La vue diff en haut à droite montre les modifications pour le fichier sélectionné. Vous pouvez indexer des sections individuelles (ou des lignes individuelles) en cliquant-droit dans cette zone.

La zone de message et d'action est en bas à droite. Taper les messages dans la boîte à texte et cliquer « Commiter » pour réaliser une action similaire à git commit. On peut aussi choisir de corriger le commit précédent en sélectionnant le bouton radio « Corriger dernier commit », ce qui met à jour la zone « Modifs. indexées » avec le contenu du dernier commit. Ensuite, on peut simplement indexer ou désindexer certaines modifications, modifier le message de validation et cliquer à nouveau sur le bouton « Commiter » pour remplacer l'ancien commit par le nouveau.

Côté serveur

Gitweb

Gitweb est une interface web permettant de voir tous les commits, les commentaires etc… pour git.

Installation

```
yum install gitweb
```

Configuration

Préferer Apache à Nginx car il faut un module complémentaire pour ce dernier qui n'existe pas dans Centos : fcgiwrap

Configurer Apache

```
vi /etc/apache2/conf.d/gitweb

    <VirtualHost *:80>
        ServerName git.example.org
        DocumentRoot /pub/git
        SetEnv  GITWEB_CONFIG   /etc/gitweb.conf
        RewriteEngine on
        # make the front page an internal rewrite to the gitweb script
        RewriteRule ^/$  /cgi-bin/gitweb.cgi
        # make access for "dumb clients" work
        RewriteRule ^/(.*\.git/(?!/?(HEAD|info|objects|refs)).*)?$ /cgi-
bin/gitweb.cgi%{REQUEST_URI}  [L,PT]
    </VirtualHost>

Ou bien, :

    Alias /gitweb /usr/share/gitweb
    <Directory /usr/share/gitweb>
      Options FollowSymLinks +ExecCGI
      AddHandler cgi-script .cgi
    </Directory>
```

Redémarrer ou reloader ensuite le serveur apache.

Adapter le fichier gitweb.conf

```
    $ vi /etc/gitweb.conf

    # path to git projects (<project>.git)
    $projectroot = "/var/lib/git";
    # directory to use for temp files
    $git_temp = "/tmp";
    # target of the home link on top of all pages
    #$home_link <nowiki>= $my_uri || "/";
    # 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
    $stylesheet = "/gitweb.css";
    # logo to use
    $logo = "/git-logo.png";
```

```
    # the 'favicon'
    $favicon = "/git-favicon.png";

    # change default git logo url
    $logo_url = "http://[adresse_serveur]/gitweb";
    $logo_label = "web Git Repository";
    # This prevents gitweb to show hidden repositories
    $export_ok = "git-daemon-export-ok";
    $strict_export = 1;
    # This lets it make the URLs you see in the header
    @git_base_url_list = ( 'git://[adresse_serveur]/git' );
```

Il est possible de désactiver l'accès à certains repository. Pour ce faire, activer ceci dans la configuration de gitweb :

```
vi /etc/gitweb.conf

    $export_ok = "gitweb-export-ok";
```

Puis créer ce fichier dans chaque repository à afficher :

```
touch /var/cache/git/git_deimosfr.git/gitweb-export-ok
```

Seuls les repository disposant de ce fichier seront affichés

Pour changer le header de gitweb, créer un fichier indextext.html à l'emplacement des cgi et insérer du code HTML :

```
vi /usr/lib/cgi-bin/indextext.html

    <h2>Web Git</h2>
    Welcome on my Gitweb. I store here my configurations that I usually use
every days.<br /><br />
```

Customisation de l'Interface web

Il existe d'autres thèmes que celui par défaut de gitweb.

Télécharger le thème puis remplacer l'ancien thème (gitweb.css) :

```
tar -xzvf kogakure-gitweb-theme.tar.gz
cd kogakure-gitweb-theme
mv /usr/share/gitweb/gitweb.css /usr/share/gitweb/gitweb-old.css
cp gitweb.css /usr/share/gitweb/
cp -Rf img /usr/share/gitweb/
chown -Rf www-data. /usr/share/gitweb/
```

Intégration Piwik

Pour intégrer piwik dans witgeb, appliquer le patch suivant sur le code javascript dans la page gitweb.cgi

```
    *** gitweb.old  2011-04-05 14:05:06.120951481 +0200
    --- gitweb.cgi  2011-04-05 14:04:41.913944817 +0200
    ***************
    *** 3612,3617 ****
    --- 3612,3633 ----
                  qq!</script>\n!;
        }
    +
    +   print <<PIWIK;
    + <!-- Piwik -->
    + <script type="text/javascript">
    + var pkBaseURL = (("https:" == document.location.protocol) ?
"https://www.deimos.fr/piwik/" : "http://www.deimos.fr/piwik/");
    + document.write(unescape("%3Cscript src='" + pkBaseURL + "piwik.js'
type='text/javascript'%3E%3C/script%3E"));
    + </script><script type="text/javascript">
    + try {
    + var piwikTracker = Piwik.getTracker(pkBaseURL + "piwik.php", 10);
    + piwikTracker.trackPageView();
    + piwikTracker.enableLinkTracking();
    + } catch( err ) {}
    + </script><noscript><p><img
src="http://www.deimos.fr/piwik/piwik.php?idsite=10" style="border:0" alt=""
/></p></noscript>
    + <!-- End Piwik Tracking Code -->
    + PIWIK
    +
        print "\n</body>\n" .
              "</html>";
      }
    ***************
    *** 7033,7038 ****
    --- 7049,7066 ----
        }
        print <<XML;
      </outline>
    + <!-- Piwik -->
    + <script type="text/javascript">
    + var pkBaseURL = (("https:" == document.location.protocol) ?
"https://www.deimos.fr/piwik/" : "http://www.deimos.fr/piwik/");
    + document.write(unescape("%3Cscript src='" + pkBaseURL + "piwik.js'
type='text/javascript'%3E%3C/script%3E"));
    + </script><script type="text/javascript">
    + try {
```

```
    + var piwikTracker = Piwik.getTracker(pkBaseURL + "piwik.php", 10);
    + piwikTracker.trackPageView();
    + piwikTracker.enableLinkTracking();
    + } catch( err ) {}
    + </script><noscript><p><img
src="http://www.deimos.fr/piwik/piwik.php?idsite=10" style="border:0" alt=""
/></p></noscript>
    + <!-- End Piwik Tracking Code -->
      </body>
      </opml>
      XML
```

Gogs

Gogs également connu sous le nom de Go Git Server est un serveur Git auto-hébergé multiplateforme open source écrit en Golang, similaire à GitLab qui est écrit en Ruby.

Il est facile à installer et à configurer et offre beaucoup d'options de personnalisation lors de l'installation, il permet de choisir entre MySQL, SQLite et PostgreSQL en tant que backend de base de données.

C'est aussi l'une des solutions serveur Git auto-hébergées les plus légères et les plus rapides.

Récupérer le rpm sur packager.io et installer de Gogs:

```
wget https://packager.io/gh/pkgr/gogs/builds/669/download/centos-6
yum install gogs
```

Installer MySQL:

```
yum install mysql-server -y
```

Installer NGINX

Il est nécessaire d'utiliser une reverse proxy pour accéder à Gogs.

```
yum install nginx -y
```

Se connecter à la console MySQL pour créer la base de données gogs

```
mysql -u root -p
CREATE DATABASE gogs;
quit;
```

Modifier de la configuration NGINX `/etc/nginx/conf.d/default.conf` pour faire pointer vers le port d'écoute local 6000

```
vi /etc/nginx/conf.d/default.conf
```

```
    server {
      listen          80;
      server_name     ${HOSTNAME};
      location / {
        proxy_pass      http://localhost:3000;
      }
   }
```

Ou pour configurer un alias derrière le reverse-proxy NGINX

```
    location /git/ {

      proxy_set_header        Host $host;
      proxy_set_header        X-Real-IP $remote_addr;
      proxy_set_header        X-Forwarded-For $proxy_add_x_forwarded_for;
      proxy_set_header        X-Forwarded-Proto $scheme;
      rewrite                 ^/git/(.*) /$1 break;
      rewrite                 ^/git$ /$1 break;
      proxy_pass          http://gogs.ip:3000;
      proxy_read_timeout  90;
    }
```

Redémarrer NGINX.

```
service nginx restart
```

Accéder au serveur Gogs à l'adresse http://127.0.0.1/, le configurer et créer un nouvel utilisateur, pour stocker les projets sur ce serveur Git privé.