Installer son propre serveur git – Partie 1

Github est aujourd'hui la référence pour ce qui est du "social coding", malheureusement il est un peu trop social à mon gout. En effet, il n'est pas possible d'avoir de dépôt privé à moins de payer 7$ par mois et de ne pas savoir ou seront stockés mes données que je veux garder pour moi.
Bref, il peut être intéressant de créer son propre serveur git personnel, pour contrôler parfaitement son code...

Vous l'aurez probablement compris, mais l'installation d'un serveur privé va se faire en plusieurs partie. Dans cette première partie, nous allons voir comment installer et configurer notre serveur Git sur un serveur Debian pour héberger ses propres dépôts git.
Après cette première partie, il sera possible de rapatrier du code sur sa machine locale, de le modifier et d'envoyer au serveur les modifications, par exemple :

```
git clone git@mon-serveur.fr:mon-depot.git
```

Ce tutoriel se veut simple et s'adresse aux personnes qui souhaitent être seules à pouvoir commiter du code sur le serveur.

› Installation de git

L'installation de git à proprement parler n'est pas compliquée puisqu'il est généralement disponible dans les gestionnaires de paquets des différentes distributions Linux et j'ose espérer qu'il est déjà installer sur votre machine.
Si ce n'est pas le cas, la commande à saisir pour installer git est la suivante :

```
sudo apt-get install git
```

Un peu de patience et voilà git installé sur votre serveur.

› Création de l'utilisateur git

Bon maintenant, il faut créer un utilisateur *git* qui permettra de se connecter sur la machine en SSH.
On va donc créer un utilisateur git ainsi que son groupe, l'utilisateur n'aura pas de

mot de passe et sa "home" se trouvera dans le répertoire `/var/git` :

```
sudo adduser --system --shell /bin/bash --group --disabled-password --home /var/gi
```

Normalement le répertoire `/var/git` existe déjà (il est créé lors de l'installation de git), donc ne vous inquiétez pas si des "warning" apparaissent lors de la création de l'utilisateur. Cependant, il va falloir changer le propriétaire du répertoire `/var/git/` pour l'utilisateur `git` :

```
sudo chown git:git /var/git
```

L'utilisateur `git` est désormais créé sur votre serveur, mais pour l'instant il n'est pas possible de se connecter en tant qu'utilisateur `git`.

› Configuration SSH du client

Puisque l'on ne peut pas se connecter à l'utilisateur, il va falloir configurer l'accès à cet utilisateur. Puisque l'on va y accéder à partir de machines distantes, on va utiliser SSH pour se connecter en tant qu'utilisateur git, ce qui nécessite un peu de configuration sur votre machine locale.

Pour configurer cet accès, il faut créer des clés SSH avec la commande suivante :

```
ssh-keygen -t rsa
```

Cette commande va générer une clef SSH privée et une clef publique qui se trouveront dans le répertoire `~/.ssh`. Avec la commande que nous venons de saisir, la clef privée s'appelle `id_rsa` et la clé publique `id_rsa.pub`. Ces clef permettront de s'authentifier sur votre serveur sans avoir à saisir d'identifiant. Visualisez le contenu de votre clé publique puisque c'est le contenu de la clé publique qui sera nécessaire pour la configuration SSH du serveur.

Avant de passer à l'étape suivante, veuillez vérifier les droits de votre configuration SSH :

```
chmod 755 ~/.ssh
chmod 600 ~/.ssh/id_rsa
chmod 644 ~/.ssh/id_rsa.pub
```

Je suppose que vous utilisez une machine SSH avec le port par défaut, sinon il faudra ajuster votre configuration SSH (http://www.sheevaboite.fr/article41/gerer-

cles-ssh-plusieurs-serveurs).

Si vous voulez accéder aux dépôts à partir de plusieurs machines, il faudra transférer votre clef SSH dans le répertoire `.ssh` sur votre seconde machine.

› Configuration SSH du serveur

Maintenant que votre machine de développement est configurée, il faut maintenant finir la configuration SSH sur votre serveur qui hébergera les dépôts git.
Voici les commandes à saisir pour terminer la configuration sur le serveur :

```
sudo mkdir /var/git/.ssh
sudo touch /var/git/.ssh/authorized_keys
```

Il faut maintenant écrire le contenu de votre clé publique que vous venez de générer dans le fichier `authorized_keys`. Faites cela avec la commande suivante :

```
echo "CONTENU_DE_LA_CLE" >> /var/git/.ssh/authorized_keys
```

Dernière étape, vérifier bien les droits de vos fichiers de configuration SSH :

```
chmod 755 ~/.ssh
chmod 644 ~/.ssh/authorized_keys
```

Et voila, la configuration du serveur git est terminée. Pour vérifier qu'il n'y a pas d'erreur, on peut vérifier que l'on peut se connecter sur le serveur en SSH avec l'utilisateur git :

```
ssh git@mon-serveur.fr
```

Normalement, vous devriez être connecté et vous pouvez naviguer dans l'arborescence de fichiers.

› Création d'un dépôt

Toujours sur le serveur, il faut maintenant créer un dépôt git sur le serveur et l'initialiser, rien de bien compliqué puisqu'il suffit de saisir les commandes suivantes :

```
sudo mkdir /var/git/mon-depot.git
sudo cd /var/git/mon-depot.git
sudo git init --bare
sudo chown -R git:git /var/git/mon-depot.git
```

La ligne `giti init --bare` permet d'initialiser le dépôt. C'est terminé pour de bon de la configuration du serveur.

› Clone et commit

Rendez-vous sur votre machine locale, nous allons rapatrier le projet `on-depot.git` sur la machine. Toujours très simple, un petit clone suffit :

```
git clone git@mon-serveur.fr:mon-depot.git
```

Vous avez maintenant un répertoire local nommé *mon-depot.git*.Normalement, un warning vous indique que vous venez de cloner un repository vide, ce qui est le cas donc pas de panique. Il ne vous reste plus qu'à coder et commiter...

```
echo "console.log('Hello NodeJS');" > helloworld.js
git add helloworld.js
git commit -m "Initial commit" helloworld.js
git push origin master
```

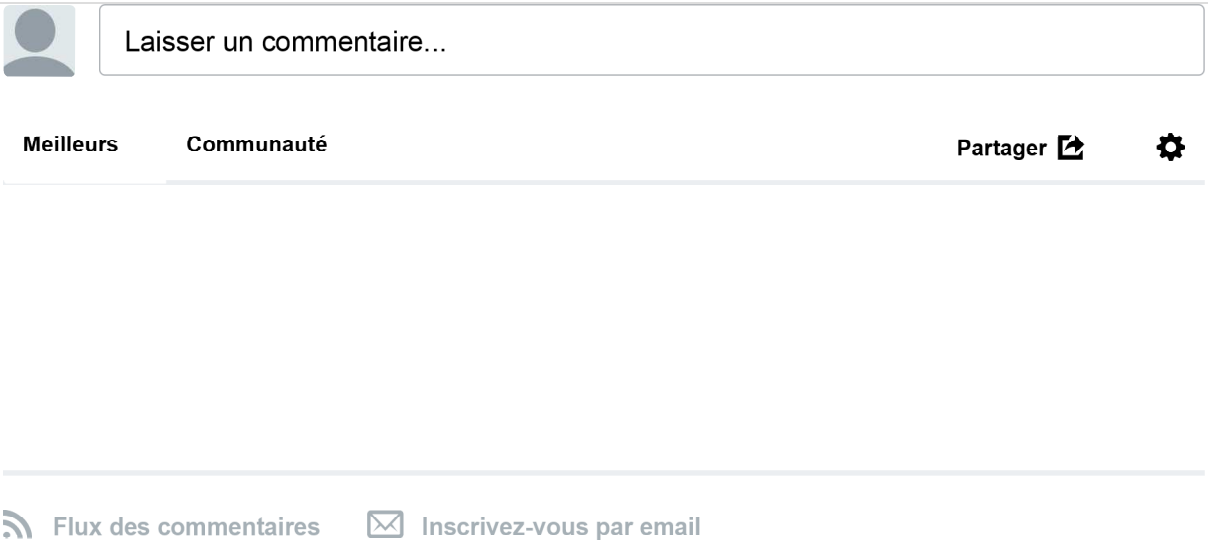
La dernière ligne est importante puisque sans elle vous ne pourrez pas commiter votre code! En effet, nous venons d'installer un dépôt vide, donc git ne parvient pas à faire de synchro entre les fichiers locaux et les fichiers sur le serveur. La commande `git push origin master` permet de pousser les premières modifications sur le serveur git.
Par la suite, vous pourrez comme d'habitude utiliser `git push` pour commiter votre code.

› Conclusion

Voilà la première partie est terminée, la mise en place de ce service est peu longue je le concède. Mais le fait d'avoir ses propres dépôt stockés sur son propre serveur est quelque chose de très agréable lorsque l'on ne veut pas envoyer ses données sur Github.

Dans la seconde partie, nous verrons comment importer des dépôts Github sur notre nouveau serveur puis comment mettre en place une web interface sur les dépôts...

› *Publié le 30 Octobre 2011* ▪ *Auto-hébergement (/archives/auto-hebergement)*



Blog auto-hébergé sur mon petit serveur (/a-propos) – Propulsé par Jekyll (http://jekyllrb.com/) – HumansTxt (/humans.txt)