Développer avec Subversion, Trac et Buildbot

version 1.1

Date: 8 juin 2006

Olivier Ramonat <olivier.ramonat@netelem.com>
Pascal Obry <pascal@obry.net>

Table des matières

1 Introduction

Ce document porte sur l'installation sur un serveur[1][2] d'une suite d'outils de gestion de projet :

- Subversion[3] pour le contrôle de version ;
- Trac[4] pour la gestion de projet : Wiki, système de gestion de tickets et suivi de l'évolution des sources (quand il est couplé à Subversion) ;
- et Buildbot pour l'automatisation du packaging, des compilations et des tests.

Ces trois outils peuvent interagir et permettent ensemble d'améliorer la qualité et la rapidité d'un développement logiciel. Installés sur un serveur, ils peuvent être partagés par différents projets très facilement.

Il est possible de réaliser l'installation de plusieurs manières. Ce document propose un modèle d'installation et d'administration qui se veut cohérent, simple et sécurisé.

2 Préparer le système

2.1 Installation sous GNU/Linux et MacOS

2.1.1 Créer les comptes d'utilisateurs

Trois comptes système doivent être créés sur le serveur. Cette opération est réalisée par l'utilisateur *root* avec la commande *adduser* :

```
$ adduser svn
$ adduser trac
$ adduser buildbot
```

2.2 Installation sous Windows

Il est possible d'installer Subversion, Trac et Buildbot sous Windows 2000/XP en utilisant l'environnement Cygwin[5].

[1] Testé sous GNU/Linux et MacOs

[2] Testé sous Windows/Cygwin, procédure à ajuster pour cette plateforme

[3] Subversion est le remplaçant de CVS, il est utilisé par de nombreux projets tels que Python, Apache, GCC ou KDE

[4] Trac est un équivalent de Sourceforge

[5] CYGWIN est une couche d'émulation POSIX pour Windows

2.2.1 Installation de Cygwin

L'installation est assez simple et automatisée :

- récupérer setup.exe depuis http://www.cygwin.com
- installer les modules suivants :
  - Base (sélectionné par défaut car obligatoire)
  - Admin - cygrunsrv
  - Devel - gcc-core
  - Devel - make
  - Devel - Subversion
  - Net - openssh
  - Python - python
  - Utils - rebase
- lancer une console Cygwin et exécuter rebaseall

2.2.2 Créer les comptes d'utilisateurs

Créer les comptes utilisateurs svn, trac, buildbot depuis l'interface standard de Windows.

Depuis une console Cygwin créer les fichiers '`/etc/passwd`' et '`/etc/group`' correspondant :

```
$ mkpasswd -l > /etc/passwd
$ mkgroup -l > /etc/group
```

2.2.3 Configurer SSH

Cette étape va permettre de se connecter à distance et de façon sécurisée à la machine Windows. C'est aussi le protocole recommandé ici pour Subversion.

```
$ ssh-host-config
```

Aux questions qui vous sont posées, il faut répondre :

- "privilege separation" : *yes*
- "create local user sshd" : *yes*
- "install sshd as a service" : *yes*
- "CYGWIN=" : *ntsec*

2.2.4 Installation de Trac

Cygwin ne fournit pas de paquet tout préparé pour Trac. L'installation est un peu délicate car elle requière plusieurs modules :

– Sqlite
Un paquet binaire '`sqlite-2.8.14-1.tar.bz2`' pour Cygwin est disponible ici : http://sourceforge.net/project/showfiles.php?group_id=99645&package_id=121049.

```
$ cd /
$ tar xfj sqlite-2.8.14-1.tar.bz2
```

– pysqlite (testé avec la version 2.1.12)
Installation en utilisant le module python '`setup.py`'.

```
$ python setup.py build
$ python setup.py install
```

– Clearsilver (testé avec la version 0.10.2)
Installation en utilisant la procedure standard suivante :

```
$ ./configure
$ make
$ make install
```

Cependant il n'est pas possible de compiler le fichier '`neo_date.c`' avec le compilateur Cygwin. Un autre problème vient du linker sous Cygwin qui est sensible à l'ordre des bibliothèques. Le patch suivant permet de corriger ces problèmes :

```
*** util/neo_date.c.orig          Mon Mar  6 15:02:23
    2006
--- util/neo_date.c       Mon Mar  6 15:00:17 2006
***************
*** 83,89 ****
--- 83,93 ----
      return ttm->tm_gmtoff;
  #elif defined(HAVE_TZNAME)
      long tz;
+ #ifndef _CYGWIN_
      tz = - timezone;
+ #else
+     tz = - _timezone;
+ #endif
      if(ttm->tm_isdst)
        tz += 3600;
      return tz;
*** python/setup.py.orig          Mon Mar  6 15:38:03
    2006
--- python/setup.py       Mon Mar  6 15:38:45 2006
***************
*** 5,11 ****

  VERSION = "0.10.2"
  INC_DIRS = ["../"]
```

```
! LIBRARIES = ["neo_cgi", "neo_cs", "neo_utl"]
  LIB_DIRS = ["../libs"]
  CC = "gcc"
  LDSHARED = "gcc -shared"
--- 5,11 ----

  VERSION = "0.10.2"
  INC_DIRS = ["../"]
! LIBRARIES = ["neo_cgi", "neo_cs", "neo_utl", "z"]
  LIB_DIRS = ["../libs"]
  CC = "gcc"
  LDSHARED = "gcc -shared"
```

– Trac (testé avec la version 0.9.3)
  Installation en utilisant le module python '`setup.py`'.

```
$ python setup.py build
$ python setup.py install
```

Il est important de vérifier que le compilateur GCC/C de Cygwin est le premier dans le PATH. Ce compilateur est utilisé lors de la phase de construction de Clearsilver.

2.2.5 Installation de Buildbot

Cygwin ne fournit pas de paquet tout préparé pour Buildbot. Cependant l'installation est simple. Buildbot comporte un document nommé INSTALL qui décrit comment procéder.

Généralement la procédure est la suivante :

– récupérer l'archive source
– décompresser l'archive
– construire le(s) module(s)

```
$ python setup.py build
```

– installer le(s) module(s)

```
$ python setup.py install
```

Il est important de vérifier que le compilateur GCC/C de Cygwin est le premier dans le PATH. Ce compilateur est utilisé lors de la phase de construction de Twisted et Buildbot.

2.2.6 Création de services Windows

Il est possible d'installer Trac et Buildbot comme services Windows. Ceci a l'avantage que ces applications seront lancées automatiquement au redémarrage du système.

La première étape est d'ajouter c :\cygwin\bin au PATH du système via l'interface standard de Windows. La variable système CYGWIN positionnée avec "tty ntsec" doit aussi être ajoutée.

Pour créer un service on utilise l'outil Cygwin *cygrunsrv*. Ceci doit être fait après l'intallation de Trac et Buildbot comme décrit dans la suite de ce document.

```
$ cygrunsrv -I buildbot -c /home/buildbot/buildroot -a "--
    no_save -y buildbot.tac" -e CYGWIN="tty ntsec" -p /usr/
    bin/twistd -u buildbot -w buildbot_pwd
```

Pour démarrer le service :

```
$ cygrunsrv --start buildbot
```

Pour obtenir une description de tous les paramètres :

```
$ cygrunsrv --help
```

2.2.7 Note finale

L'installation d'outils du monde libre, développés principalement pour GNU/Linux, sous Windows en utilisant Cygwin est possible et assez simple. Il faut cependant garder en tête que quelques ajustements peuvent être nécessaires en fonction de la configuration de la machine.

Cette section a pour principal objectif de guider l'administrateur dans son installation en présentant les étapes propres à l'environnement Cygwin.

Le reste du document est indépendant de la plate-forme.

3 Paramétrer Subversion

Toutes les opérations se feront sur le compte *svn*.

3.1 Créer un dépôt Subversion

Pour créer le dépôt *my_proj*, utiliser la commande

```
$ svnadmin create --fs-type=fsfs my_proj
```

Subversion peut stoquer sa base de données en deux formats différents :
- Berkeley DB
- FSFS (développé par Subversion et fondé sur le système de fichiers)

Il est conseillé d'utiliser FSFS qui est plus stable en cas d'erreur de Subversion.

3.2 Accès au dépôt en ssh

Pour utiliser le protocol *svn+ssh* il faut disposer d'un compte utilisateur sur le serveur.

Pour automatiser la connexion à l'aide de script, une solution simple est de créer une clé RSA sans mot de passe (pass phrase) et de l'utiliser pour s'authentifier sur le serveur.

Création de la clé rsa :

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

Envoi de la clé sur le serveur *server_addr* :

```
$ ssh-copy-id -i /home/user/.ssh/id_rsa.pub
    user@server_addr
```

L'outil ssh-copy-id n'existe pas pour Cygwin il est donc nécessaire de d'ajouter manuallement le contenu du fichier '`/home/user/.ssh/id_rsa.pub`' au fichier '`user@server_addr:.ssh/authorized_keys`'.

Récupération des sources du projet *my_proj* :

```
$ svn co svn+ssh://user@server_addr/home/svn/my_proj
```

Il n'y a pas besoin de founir de mot de passe.

3.3 Hook Subversion

Il est possible de faire exécuter automatiquement des scripts par Subversion, en les plaçant dans le répertoire hook du dépôt.

3.3.1 pre-commit hook

Le script `hooks/pre-commit` est lancé à chaque *commit*[6]. Il est donc possible de vérifier certaines conditions sur les fichiers envoyés avant de les accepter. Par exemple, il peut être utile de tester le bon respect des règles de style élémentaires et des règles de syntaxe des fichiers lorsque le compilateur le permet[7].

Il est possible aussi d'analyser le log de la transaction afin d'imposer qu'un numéro de ticket Trac soit renseigné. Ceci permet de garantir une traçabilité complète entre les modifications effectuées dans le code et le système de gestion de ticket. La traçabilité est un point important pour l'assurance qualité logiciel.

[6] Un commit est un envoi de modification des sources du projet

[7] Par exemple l'option -gnatc du compilateur GNAT pour le langage Ada

3.3.2 post-commit hook

Le script `hooks/post-commit` est couramment utilisé pour l'envoi de notification par mail. Il peut aussi servir à l'intégration du log de la transaction dans Trac (section 4.3) ou pour exécuter des compilations automatiques avec Buildbot (section 5.1.4).

Dans la suite de ce document il est supposé que le dépôt Subversion des projets utilise une structure standard.

```
my_proj
  |
  | - trunk
  |
  | - branches
  |     |
  |     | - release_1
  |     | - release_2
```

4 Trac

Toutes les opérations se feront sur le compte *trac*.

4.1 Créer un projet sous Trac

La création du projet *my_proj* sous Trac se fait à l'aide de l'outil *trac-admin*.

Création du projet sous le répertoire '`/home/trac/rootenv`'

```
$ trac-admin rootenv/my_proj initenv
Project Name [My Project]> My Project
Database connection string [sqlite:db/trac.db]>
Path to repository [/var/svn/test]> /home/svn/my_proj
Templates directory [/opt/local/share/trac/templates]>
```

La seule chose à faire est donc de nommer le projet et d'indiquer l'adresse du dépôt Subversion (qui doit être sur le même serveur).

4.2 Démarrer Trac

La commande pour lancer Trac est :

```
$ tracd -a *,users.htdigest,TracRealm -p 8000 -e /home/trac
  /rootenv/
```

Où users.htdigest est créé par la commande :

```
$ htdigest -c users.htdigest TracRealm user
```

Utiliser l'authentification permet de garantir que les modifications sur le Wiki Trac ne soient pas anonymes, toujours pour garantir une bonne traçabilté des informations.

8000 est le numéro du port.

Votre server Trac hébergé sur le serveur *my_server* doit maintenant être accessible à l'adresse : http ://my_server :8000

4.3 De Subversion aux tickets de Trac

Un ticket sur Trac peut être un rapport de bug, une demande pour une nouvelle fonctionnalité... Trac attribue automatiquement un numéro à chaque ticket.

Pour relier automatiquement les messages de logs d'une transaction Subversion à un ticket émis sur Trac, il faut créer un hook Subversion.

Le script python `scripts/trac-post-commit-hook` repère, dans le log Subversion, les chaînes *refs #11, #12* ou *fixes #14* et ajoute, dans l'historique des tickets correspondants sous Trac, le log de la transaction.

Lorsque le log contient *fixes*, le billet sous Trac est automatiquement fermé.

Pour plus de contrôle, on peut utiliser le script '`trac-pre-commit-hook`' pour rejeter toute transaction Subversion ne contenant pas un numéro de ticket.

5 Automatiser avec Buildbot

5.1 Configurer Buildbot

5.1.1 Pré-requis

- Twisted http ://twistedmatrix.com/trac/wiki/TwistedProject
- Buildbot http ://sourceforge.net/projects/buildbot/

Récupérer l'archive TwistedSumo puis installer :

- ZopeInterface
- Twisted
- TwistedWeb
- TwistedMail (optionnel)

Sous debian :

```
$ apt-get install python-twisted python-twisted-web
```

Récupérer Buildbot et l'installer.

La version de Buildbot testée, Buildbot-0.7.3, ne permet pas de partager un serveur Buildbot entre différents projets. Il est néanmoins possible de modifier ce comportement en surchargeant une classe (code à placer dans master.cfg, voir section 5.1.5). Cependant, il y aura certaines limitations. Il ne sera pas, par exemple, possible de créer une page de suivi par projet.

5.1.2 Configurer le serveur

En tant qu'utilisateur Buildbot, vous devez exécuter :

```
$ mkdir /home/buildbot/buildroot
$ buildbot master /home/buildbot/buildroot
$ cd /home/buildbot/buildroot
$ cp Makefile.sample makefile
```

Il faut récuperer le fichier master.cfg et l'éditer suivant votre configuration.

Définition des paramètres de connexion : *svnserver* doit être accessible de tous les clients Buildbot ici à l'adresse *my_server*.

```
svnserver = 'svn+ssh://buildbot@my_server/home/svn/'

port = 9989                                    # for slaves
wport = 8010                                   # for web interface
```

Déclaration des clients Buildbot : Il faut ensuite définir le nom et le mot de passe de tous les clients Buildbot qui se connecteront au serveur. Ici bot1 et bot2 :

```
# bot-machines used for the builds

c['bots'] = [("bot1", "bot_passwd"),
             ("bot2", "bot_passwd")]
```

Configuration d'un projet :

Compilation à chaque modification des sources : Ici, la classe *RepositoryScheduler* est utilisée à la place de la classe *Scheduler*. Un paramètre supplémentaire *repository* est défini pour que Buildbot gère de multiples projets (section 5.1.5).

```
s_trunk=RepositoryScheduler(name="trunk-only",
                            branch="trunk",
                            treeStableTimer=1*60,
                            builderNames=["test-linux",
                                          "test-windows"],
                            repository='/home/svn/test')

s_b1=RepositoryScheduler(name="release-1",
                         branch="branches/release-1",
                         treeStableTimer=1*60,
                         builderNames=["test-linux",
                                       "test-windows"],
                         repository='/home/svn/test')

c['schedulers'].append(s_trunk)
c['schedulers'].append(s_b1)
```

Le projet est reconstruit lorsqu'une modification a été faite sous Subversion, et qu'aucun autre changement n'a eu lieu depuis *treeStableTimer*. Il est important de noter que Buildbot gère parfaitement les différentes branches d'un projet. Ci-dessus un scheduler a été défini pour la branche principale du projet (*trunk*) et pour la branche *branches/release-1*.

Commandes à exécuter : Dans le cas où le projet est récupéré à partir de l'url *baseURL*, où on utilise un *./configure*, et où la compilation se fait par la commande *make* et les tests par la commande *make check*, il faudra définir :

```
test_steps = [s(step.SVN, mode="update", baseURL=svnserver
    + 'test/'),
              s(step.Configure),
              s(step.Compile),
              s(step.Test, command=["make", "check"])]

test_f = factory.BuildFactory(test_steps)
```

Affectation des clients pour ce projet : Les deux clients Buildbot qui construiront le projet *test* sont *bot1* et *bot2*. Ils apparaîtront sous *test-linux* et *test-windows* dans l'interface web du serveur Buildbot.

```
bot1_builder = {'name': "test-linux",
```

```
        'slavename': "bot1",
        'builddir': "test-linux",
        'factory': test_f,
        }

bot2_builder = {'name': "test-windows",
        'slavename': "bot2",
        'builddir': "test-windows",
        'factory': test_f,
        }

c['builders'].append (bot1_builder)
c['builders'].append (bot2_builder)
```

Tests journaliers : Un test supplémentaire aura lieu tous les soirs à 23H45 sur la branche principale.

```
test_nightly=Nightly('test-nightly', ['test-linux'],
                            hour=23, minute=45, branch='trunk')
c['schedulers'].append (test_nightly)
```

Notification : Un mail sera envoyé à urladmin@my_server pour l'informer du résultat.

```
c['status'].append(mail.MailNotifier
                    (builders=['test-linux', 'test-windows'
                        ],
                     fromaddr="buildbot@my_server",
                     extraRecipients=["admin@my_server"])
```

Lancer le serveur :

```
$ make start
```

5.1.3 Créer un client Buildbot

L'installation d'un client Buildbot ne nécessite pas le module *python-twisted-web*.

Installer Buildbot sur un autre poste et créer un client :

Créer un répertoire *buildroot*, indiquer l'adresse du serveur Buildbot *server_addr* (le port par défaut est 9989) ainsi que le nom et le mot de passe du client.

```
$ buildbot slave buildroot server_addr:9989 <name> <passwd>
```

ATTENTION : Pour que le client Buildbot puisse récupérer la dernière version Subversion par le protocole svn+ssh la connexion doit être automatique. Vérifier que l'utilisateur qui lance le client peut bien se connecter automatiquement (utilisation de clé rsa cf section 3.2 page 8).

5.1.4 De Subversion à Buildbot

Le script python `scripts/trac-post-commit-hook` doit être appelé par Subversion depuis `hooks/post-commit`. Après chaque commit, le serveur Buildbot ajoute une reconstruction du projet (dans la branche où le changement a eu lieu) dans sa file d'attente.

5.1.5 Projets multiples

Pour étendre la gestion de Buildbot à plusieurs projets, il est possible de créer une nouvelle classe *RepositoryScheduler* héritant de la classe *Scheduler* de Buildbot.

Lorsqu'un hook Subversion notifie au serveur un changement dans un projet, seul le projet concerné est reconstruit.

```
class RepositoryScheduler(Scheduler):
    """Extend Scheduler to allow multiple projects"""
        def __init__(self, name, branch, treeStableTimer,
            builderNames,
                     repository, fileIsImportant=None):
            """
            Override Scheduler.__init__

            Add a new parameter : repository
            """
                Scheduler.__init__(self, name, branch,
                    treeStableTimer,
                                   builderNames,
                                       fileIsImportant)
                self.repository = repository

        def addChange(self, change):
            """Call Scheduler.addChange only if the
                repository is modified"""
                # check to make sure the repository matches
                    !
                cs = change.comments.split(' ')
                if len(cs) > 0:
```

```
                repo = cs[0]
                log.msg('checking %s vs %s' %
                        (repo, self.repository))
                if repo != self.repository:
                        log.msg("%s ignoring
                            repository %s" %
                                (self, repo))
                        return
        # call our parent since this on the correct
            repository
        Scheduler.addChange(self, change)
```

6 Annexe

6.1 Subversion hooks

De nombreuses fonctions pour les scripts *pre-commit* et *post-commit* sont fournies dans le script `scripts/svn-hook-support.sh`.

6.1.1 pre-commit

Le script `scripts/svn-hook-support.sh` contient les différentes fonctions qui peuvent être employées dans les hooks de Subversion.

Fonction check_style permet de vérifier quelques règles de syntaxe et de style sur les fichiers modifiés. Nécessite Style_Checker (`http://www.obry.net/contrib.html`).

Fonction log_not_empty teste si le message de log a bien été rempli.

Fonction trac_pre_commit_record_log teste si le message de log contient une référence vers un ticket de Trac. Cette fonction utilise le script python `scripts/trac-pre-commit-hook`.

6.1.2 post-commit

Fonction trac_post_commit_record_log insère le message de log dans l'historique du ticket de Trac. Cette fonction utilise le script python `scripts/trac-post-commit-hook`.

Fonction send_mail_post_commit envoie un mail à chaque transaction.

Fonction buildbot_post_commit notifie le serveur Buildbot qu'une transaction a eu lieu. Cette fonction utilise le script python `scripts/svn_buildbot.py`.

6.1.3 Exemples

Deux exemples de scripts hooks sont fournis :

- `hooks/pre-commit`
- `hooks/post-commit`

6.2 Buildbot

Un exemple de configuration pour le serveur de Buildbot est aussi présent : `scripts/buildbot/master.cfg`.