Sytème de numérotation des versions

Nous avons deux contraintes à gérer :

- le marquage des versions majeures / mineures
- pouvoir prendre un snapshot du code à pérenniser.

Versions majeures / mineures.

Par convention, on note le plus souvent les numéros de version de la façon suivante :

```
<numéro majeur>.<numéro mineur>
```

Un changement de version mineure indique des modifications qui ne remettent pas en cause le fonctionnement précédent du logiciel. Par exemple : du debugging, l'ajout d'une option voir d'une feature.
Un changement de version majeure indique des modifications qui cassent une spécification précédente (ou bien l'ajout de features importantes).
Pour l'instant, nous restons dans une version majeure < 1.0 pour montrer que le package est toujours en version "test/beta/not fully tested".
Concernant la numérotation mineure, nous n'avons pas besoin de l'incrémenter à chaque fois qu'on commit des modifs. Ca concerne plutôt les distribs du code.

Pour produire une release "par défaut" (à la racine de pyhrf-free/trunk):

```
./snap_default.sh
```

Ce qui produit un archive de la forme : ./dist/pyhrf-0.1.tar.gz

Il arrive qu'on veuille sortir une release correspondant à une modification de la version courante sans pour autant vouloir changer le numéro de version (eg correction de bug). Cette version modifiée peut être marquée par la date et releasée via la commande (à la racine de pyhrf-free/trunk) :

```
./snap_default_tag_date.sh
```

Ce qui produit une archive de la forme : ./dist/pyhrf-0.1-20111026.tar.gz

Snapshot à pérenniser

Cette pratique est importante pour pouvoir sauvegarder et marquer l'état du code à un instant t et pouvoir s'y référer de façon sûre par la suite.
On a en effet besoin d'associer de façon sûre du code à des résultats.

Une bonne pratique serait qu'à la fin d'un projet (eg après avoir sorti les résultats d'un article), le code spécifique à ce projet soit mis de côté.
Ce "code" comprend :

- du code empaqueté/structuré dans des fonctions qui sont (potentiellement) testées et intégrées dans la lib (accessible via import pyhrf).
- du code contenu dans des scripts qui n'est pas inclu dans des tests et qui n'est pas intégré dans la lib.

Pour le code empaqueté, ce sont les tests qui doivent garantir son bon fonctionnement. Pour les scripts, leur bon fonctionnement est garanti seulement s'ils utilisent la bonne version de la lib. Il faut donc placer un garde-fou au début de chaque script pour forcer l'utilisation d'une version spécifique de la lib.

Voici une proposition de procédure pour garantir cette pérennisation :
1. démarrage du projet -> création de scripts dans script/WIP/
2. modification du code dans la lib et dans les scripts (work ... hack ... work ... svn commit ... rehack ...)
3. sortie des résultats pour le projet
4. finalisation de la production liée au projet (eg soumission d'un article), champagne !

5. nettoyage des scripts, puis extraction vers script/Projects/mon_projet des scripts qui produisent les résultats produits précédemment.
6. ajout des lignes suivantes au début des scripts forçant la version du code à utiliser :

```
import pkg_resources
pkg_resources.require('pyhrf==0.1mon_projet')
```

où 0.1 est la version "canonique" courante de pyhrf à laquelle est ajouté le suffixe "mon_projet" (voir étape 7.)
7. production du snapshot marqué par "mon_projet" en utilisant la commande suivante à la racine de pyhrf-free/trunk :

```
./snap_with_projects.sh mon_projet
```

Ce qui créera l'archive ./dist/pyhrf-0.1mon_projet.tar.gz

> Pour note:
> ce script bash contient la commande suivante :
>
> ```
> > cp MANIFEST.proj MANIFEST.in && python setup.py egg_info --tag-build=mon_projet sdist
> >
> ```
>
> Il y a manipulation du MANIFEST (fichier qui indique quels fichiers à inclure ou non dans le tarball distribué) car on ne voudrait pas par défaut que les scripts des projets soient distribués comme le reste (confidentialité). Nous pourrons revenir là dessus plus tard. En attendant, c'est plus safe de faire comme ça.

Par la suite, si on veut revisiter les résultats, il suffit de réinstaller au préalable la version du code qui va bien:

```
easy_install --prefix=~/local ./dist/pyhrf-0.1mon_projet.tar.gz
```

Si on veut reprendre le code du projet et faire des modifications (pour une review de papier), le mieux serait de considérer qu'on démarre un nouveau projet en partant d'une copie des scripts précédents. On créera donc un nouveau tag de projet "mon_projet_review1" et on utilisera un nouveau répertoire script/Projects/mon_projet_review1