JDHP Skeletons Howto

Auteur: Jérémie DECOCK

Contact: jd.jdhp@gmail.com

Version: 0.1

Date: 07/07/2016

Sommaire

1 Réflexion préalable sur les référentiels Git "skeletons"

Méthodes en concurrence:

1.1 Git submodules (jdhp-docs-shared)

- Points faibles:
    - Compliqué (?)

Dans tous les cas, c'est surement une bonne solution pour les images-samples dans les snippets

1.2 Liens symboliques (jdhp-docs-shared)

- Points forts:
    - Met à jour tous les documents d'un coup
    - Le précédent point fort est à nuancer car si je veux *exporter/publier* (sur jdhp, ...) les mises à jours, il faudra bien le faire document par document !!! (la méthode permet seulement d'éviter un "git pull upstream")
- Points faibles:
    - Les squelettes ne peuvent pas être personnalisés (à moins de les ignorer et de les copier au cas par cas mais on perd alors les mises à jours communes)
    - Facile à appliquer mais compliqué à expliquer sur le README.md (méthodes peu courante qui ressemble plus à un hack qu'à une méthode sérieuse)

1.3 Helpers/Builders (jdhp-docs-builders)

Deux possibilités:

1. les Helpers (ou builders ou ...) sont des script installés sur le système pour générer des nouveaux documents vides (comme pour Sphinx)
2. s'inspirer (voir utiliser) des autotools : des fichiers .in (ex: README.md.in, etc.) qui servent de fichier de configuration + des commandes pour générer les fichiers de sortie (ex: README.md)

Problèmes:

- les mises à jours...
- plus long et difficile à mettre en place (la 1ere solution nécessite d'écrire des scripts plus ou moins sophistiqués et la 2e solution nécessite de bien comprendre les outils comme autotools, M4, etc.

1.4 Fork (jdhp-skeletons)

Référentiel père dont tous les autres dérivent (fork) et mis à jours avec des "git merge" qur les fichiers communs

- Points forts:
    - Plus complet (permet de faire de fichier templates "à trous" à compléter (ex: titre, etc. dans un fichier .tex))
    - Permet de gérer plusieurs versions de squelettes (via des branches git) plus faciles à merger (par exemple: une branche pour la version anglaise et une branche pour la version française du squelette...)

    - L'utilisateur (ie les autres personnes que moi qui veulent cloner mes référentiels) n'a rien à faire, il n'y a rien à lui expliquer (pas d'étapes de post clonage contrairement à la méthode git submodule...)
    - Pas besoin de faire des découpages bizarres dans arborescence du squelette avec des liens symboliques déroutants (pour le Makefile, le README, ...)
- Points faibles:
    - Plus difficile de faire remonter les modifs du squelette faites depuis un document "à trous" déjà complété (nécessite d'utiliser des cherrypick ou des patchs ?)
    - Nécessite de mettre à jours manuellement chaque doc (c'est un faux problème car de toutes façons il faut republier les documents uns par uns...)
    - Nécessite de bien connaitre git branches, git merge, git cherrypick, ... (ce qui peux aussi être vu comme un avantage car ça permet de gagner en expérience...)

2 Skeleton HOWTO

2.1 Créer un document basé sur un squelette

1. Créer le référentiel sur github (le laisser vide pour le moment)
2. Cloner le squelette avec la branche par défaut et renommer le remote (utiliser de préférence HTTPS pour que le squelette soit en lecture seule et éviter de pousser les mises à jours vers le squelette par erreur):

```
git clone https://github.com/jdhp-skeletons/SKELETON_NAME.git NEW_DOCUMENT_DIRECTORY
cd NEW_DOCUMENT_DIRECTORY
git remote rename origin skeleton
```

3. Déclarer le remote github:

```
git remote add origin git@github.com:jdhp-docs/DOCUMENT_NAME.git
```

4. Pousser la branche par défaut (`english-version`, `french-version` ou `master` suivant le squelette) sur github:

```
git push -u origin english-version
```

ou

```
git push -u origin french-version
```

ou

```
git push -u origin master
```

Vérifier avec:

```
git branch -vv -a
```

2.2 Créer un document bilingue basé sur un squelette

TODO: tester

1. créer le référentiel sur github (le laisser vide pour le moment) 2. cloner le squelette:

```
git clone -o skeleton git@github.com:jdhp-skeletons/rst-skeleton.git git-volab-workflow
cd git-volab-workflow
```

3. cloner les deux branches:

```
git checkout -b english-version skeleton/english-version
git checkout -b french-version skeleton/french-version
```

4. déclarer le remote github et pusher:

```
git remote add origin git@github.com:jdhp-docs/git-volab-workflow.git
git checkout master
git push -u origin master
git checkout english-version
git push -u origin english-version
git checkout french-version
git push -u origin french-version
```

Vérifier avec:

```
git branch -vv -a
```

2.3 Attacher un squelette à un référentiel existant

Declare the skeleton:

```
git remote add skeleton git@github.com:jdhp-skeletons/SKELETON_NAME.git
git fetch skeleton
```

Declare the origin (if needed):

```
git remote add origin git@github.com:jdhp-docs/DOCUMENT_NAME.git
```

Push branches on origin (if needed):

```
git checkout master
git push -u origin master

git checkout english-version
git push -u origin english-version

git checkout french-version
git push -u origin french-version
```

TODO: supprimer la branche master (http://matthew-brett.github.io/pydagogue/gh_delete_master.html) puis supprimer et reclonner le référentiel local pour éviter les problèmres de références erronées.

2.4 Appliquer les mises à jour du squelette à un document / resynchroniser les branches d'un squelette

En supposant que le squelette est dans `skeleton/english-version` (adapter s'il est dans `skeleton/french-version` ou `skeleton/master`).

2.4.1 Synchroniser le dépôt local avec le dépôt "squelette" distant

```
git fetch skeleton
```

2.4.2 Préparer la fusion

Préparer le terrain en effectuant les changements "lourds" hors du futur merge (fichiers/répertoires déplacés, fichiers/répertoires renommés, etc.)

Pour avoir une vue d'ensemble des différences:

```
git difftool -d skeleton/BRANCH_NAME
```

ou simplement:

```
git diff skeleton/BRANCH_NAME
```

Par exemple:

```
git difftool -d skeleton/english-version
```

ou:

```
git diff skeleton/english-version
```

Pour mettre à jour un fichier donné avec un *difftool* externe

```
git difftool skeleton/master FILENAME
```

Une fois les changements effectués:

```
git add .
git commit -m "Prepare a merge with skeleton/master."
```

2.4.3 Fusionner et résoudre les conflits fichier par fichier

```
git merge skeleton/BRANCH_NAME
git status
git mergetool FILENAME1
git mergetool FILENAME2
...
```

Il se peut que git refuse de fusionner deux branches qui n'ont aucun commit en commun: "refus de fusionner des historiques sans relation" ("refusing to merge unrelated histories" en anglais). Dans ce cas, il faut ajouter l'option `--allow-unrelated-histories` à `git merge`

```
git merge --allow-unrelated-histories skeleton/BRANCH_NAME
...
```

Cf. http://stackoverflow.com/questions/27641380/git-merge-commits-into-an-orphan-branch pour plus d'informations.

Corriger d'éventuelles erreurs dans la résolution des conflits ~~~~~~-

Si un fichier a migré dans l'index par erreur (i.e. dans un mauvais état) annuler et recommencer la résolution des conflits pour ce fichier

```
git checkout -m FILENAME
git mergetool FILENAME
```

2.4.4 *Vérifier et commiter*

```
git commit
```

Puis supprimer les fichiers `.orig`.

3 Brouillon

3.1 Créer un document basé sur un squelette (ancienne méthode)

1. Créer le référentiel sur github (le laisser vide pour le moment)
2. Cloner le squelette avec une des deux branches et renommer le remote:

```
git clone -b french-version git@github.com:jdhp-skeletons/rst-skeleton.git git-volab
git remote rename origin skeleton
```

ou (TODO: essayer)

```
git clone -b french-version -o skeleton git@github.com:jdhp-skeletons/rst-skeleton.git g
```

3. Renommer la branche french-version -> master

```
git branch -m french-version master
```

4. Déclarer le remote github et pusher:

```
git remote add origin git@github.com:jdhp-docs/git-volab-workflow.git
git push -u origin master
```

Vérifier avec:

```
git branch -vv -a
```

http://stackoverflow.com/questions/4950725/how-do-i-get-git-to-show-me-which-branches-are-tracking-what/16879922#16879922

3.2 Misc

Ce qui était prévu à l'origine:

1. cloner le squelette dans jdhp-docs sur github
2. renommer le référentiel nouvellement créé dans jdhp-docs sur github

Mais en fait on ne peut créer qu'un seul fork par "organisation" d'un référentiel donné sur Github !!!

Du coup, la nouvelle procédure est la suivante:

- clonner:

```
git clone git@github.com:jdhp-skeletons/rst-skeleton.git git-volab-workflow
```

- renommer le remote:

```
git remote rename origin skeleton
```

- récupérer les branches du squelette:

```
git checkout -b skeleton-french-version skeleton/french-version
```

http://stackoverflow.com/questions/2862590/how-to-replace-master-branch-in-git-entirely-from-another-branch

Alternative à étudier:

```
git merge -s recursive -X theirs skeleton/french-version
```

Ou, dans le cas d'un document bilingue:

```
git checkout -b skeleton-english-version skeleton/english-version
git checkout -b skeleton-french-version skeleton/french-version
```

- déclarer le remote github et pusher:

```
git remote add origin git@github.com:jdhp-docs/git-volab-workflow.git
git push -u origin master

mkdir git-volab-workflow
cd git-volab-workflow/
touch README.md
git init
git add README.md
git commit -m "Initial commit."
git remote add origin git@github.com:jdhp-docs/git-volab-workflow.git
git push -u origin master
```

- ajouter le remote upstream (le squelette):

```
git remote add upstream git@github.com:jdhp-skeletons/rst-skeleton.git

git checkout english-version
...
git add .
```

```
git commit -m "..."
git push origin english-version
git checkout master
git merge english-version
git push
```

Utiliser le squelette sur un document déjà existant:

```
git clone ...
git remote origin ...
git remote upstream ...
```

4 Licence