Projet Algo & Programmation 2017-2018
Poly#
Pour bien démarrer

Polytech Nantes – INFO3

9 octobre 2017

1 Préambule

Une archive `polyhash.tar.gz` contenant une version martyre du projet se trouve sur Madoc. Vous devez la télécharger et la décompresser dans un répertoire temporaire de travail. Elle sera utilisée au fur et à mesure de la séance pour alimenter la première version de votre projet.

Les instructions sont conçues pour fonctionner—au moins—sous linux/MacOSX.

Comme pré-requis, vous devez avoir constitué les équipes de 4 à 6 étudiant(e)s. Il est également chaudement recommandé de suivre ce tutoriel interactif git (15mn).

2 Initialisation du projet avec Gitlab

1. Créer 1 dépôt gitlab par équipe, nommé `polyhash`, sur le compte de l'un des participants.
2. Ouvrir les droits :
   - *Master* à tous les membres du projet
   - *Reporter* aux 6 intervenants de l'équipe « Projet A&P » :
     - `lehn-r`
     - `perreiradasilva-m`
     - `peter-p`
     - `picarougne-f`
     - `raschia-g`
     - `vigier-t`
3. Comme suggéré dans l'entête du projet, créer un premier fichier `README.md` dans lequel vous insérez le code suivant :

```
Projet Poly#
============

Une brève description ici.

Le fichier `README.md` est écrit en **Markdown**
et permet de soigner la _mise en forme_.


L'équipe
============

à compléter
```

La message de commit est : « Ajout du README »

4. Dans le projet, importer[1] directement par l'interface `gitlab/Files/+`, les fichiers et répertoires suivants :
   — `main.py`
   — `polyhash/`
   — `polyhash/__init__.py`
   — `polyhash/polyhmodel.py`
   — `polyhash/polyhsolver.py`

   À chaque commit, choisir un message simple et précis.

3 Pip et les environnements virtuels Python

Chaque participant au projet devra réaliser individuellement ce qui suit, pour préparer son environnement de projet. Les commandes sont à exécuter dans une fenêtre de terminal.

1. Dans votre répertoire de travail, créer un environnement virtuel python 3, sorte de bac à sable propre au projet :

   ```
   $ python3 -m venv polyhvenv
   ```

   De manière alternative, si la commande échoue, essayer :

   ```
   $ virtualenv -p python3 polyhvenv
   ```

   Attention aux maudites histoires de proxys :

   ```
   $ export https_proxy=http://cache.etu.univ-nantes.fr:3128
   ```

   Et si une erreur de `locale` persiste, essayer :

   ```
   $ export LC_ALL=C
   ```

2. Entrer dans l'environnement virtuel :

   ```
   $ source polyhvenv/bin/activate
   ```

   Il est possible qu'un message suggère que certaines variables d'environnement ne soient pas correctement initialisées. Ce n'est pas problématique.
   Pour sortir de l'environnement virtuel, une fois la session de travail terminée, il suffit d'invoquer :

   ```
   $ deactivate
   ```

   Il est ainsi possible d'aller et venir dans n'importe lequel de vos environnements virtuels python, au gré des projets. Pour la suite, retournez dans votre environnement virtuel si vous en êtes sorti.

3. Localiser la commande `python` pour vérifier qu'elle désigne bien un binaire de votre environnement plutôt que l'interpréteur python du système :

   ```
   $ which python
   ```

   Puis vérifier qu'il s'agit bien d'une version 3 (`python --version`).

1. Il existe une autre manière, plus élégante, de démarrer un projet git en reprenant du code existant via la séquence de commandes `git init/add/commit/push`.

4. Mettre à jour la commande `pip` de l'environnement virtuel :

```
$ pip install --upgrade pip
```

pip est un gestionnaire de paquets pour python qui deviendra rapidement votre meilleur ami dès lors que les projets se multiplieront, avec des dépendances spécifiques à telle ou telle librairie !

5. Installer 2 paquets python :

```
$ pip install flake8 pep8-naming
```

Ils serviront à tester la conformance à PEP8, guide des bonnes pratiques syntaxiques en python.

4 Git : les premiers pas

git est la machinerie qui se cache derrière la plate-forme gitlab de l'université de Nantes, et sert à faciliter (?) le travail en groupe, principalement sur des projets de développement logiciel. C'est également le socle d'une autre plate-forme de projets collaboratifs, GitHub.

La documentation git est abondante sur la toile. Un échantillon vous est proposé sur Madoc.

4.1 Préambule

1. Configurer git localement :

```
$ git config --global user.name "John Doe"
$ git config --global user.email john.doe@etu.univ-nantes.fr
```

en remplaçant John Doe par vos nom et prénom…

2. Générer un couple (clé publique, clé privée) en laissant vide tous les champs proposés :

```
$ ssh-keygen
```

puis copier la clé *publique* qui se trouve par défaut dans `~/.ssh/id_rsa.pub` sur gitlab via le menu `Profile Settings/SSH Keys` de l'interface web.

Retenez que la clé dépend du compte qui l'a généré et vous permettra une communication transparente (sans authentification systématique) sur gitlab.

4.2 Scénario d'entraînement avec git [optionnel]

Consignes :

— utiliser les *Issues* gitlab pour la répartition des tâches.

1. Créer une copie locale de travail à partir du dépôt principal (voir les instructions dans gitlab) :

```
$ git clone https://... <dir>
```

Attention encore au proxy :

```
$ export https_proxy=http://cache.etu.univ-nantes.fr:3128
```

ou directement et durablement dans la configuration git :

```
$ git config --global https.proxy http://cache.etu.univ-nantes.fr:3128
```

2. Répartir les tâches suivantes entre les membres de l'équipe et déclarer la répartition dans des *Issues* :
   1. dans la branche `master` : modifier le `README.md` pour qu'il présente l'objectif du projet ainsi que l'équipe.
   2. dans une branche `check-pep8`, vérifier via `flake8 <nom_de_fichier>` et corriger si nécessaire la validité des fichiers :
      — `main.py`
      — `polyhash/polyhsolver.py`
      — `polyhash/polyhmodel.py`
   3. ajouter au dépôt dans la branche master le sous-répertoire `polyhutils/` et ses fichiers.

   Chacune des trois tâches ci-dessus doit être réalisée localement puis le résultat partagé sur le dépôt.
3. Fusionner la branche `check-pep8` dans la branche master en ayant déclenché une *Merge Request* et obtenu l'accord de chaque membre de l'équipe.
4. Tagger la version obtenue dans la branche master en `v0.1` (une seule fois, par l'un des membres de l'équipe).

Dorénavant, il est indispensable de maintenir un `README.md` à jour, avec les informations clefs du projet. De même, un fichier texte `CHANGELOG` permet de synthétiser les changements majeurs qui surviennent d'une version à une autre (l'histoire condensée des `commit`).

5 Annexe : la pratique git au quotidien

```
# marquer une modification
$ git add .

# valider une modification
$ git commit -m "message de commit"

# concilier sa copie de travail avec les progrès éventuels
# de la version partagée sur le dépôt principal
$ git pull origin <branch>

# partager ses modifications avec les autres membres du projet
$ git push origin <branch>
```

Adopter le workflow GitHub pour le développement collaboratif

La branche `master` porte la « version de production », et est le lieu des corrections de bugs critiques.

Le long de la branche `master`, il faut désigner certains `commit` comme des *releases*, c-à-d. des nouvelles versions apportant des changements significatifs.

```
$ git tag v1.2
```

Au-delà de ces corrections de bugs, il faut créer une branche par fonctionnalité, avec un nom très explicite :

```
# création de la branche
$ git branch lecture-csv

# déplacement dans la nouvelle branche
$ git checkout lecture-csv

# vérification
$ git status
```

Il faut proposer une *Merge Request* pour annoncer l'intégration d'une fonctionnalité à la branche `master`. Une *Merge Request* peut également avoir pour seul motif la consultation des membres de l'équipe sur le développement en cours de la fonctionnalité (donne un droit de regard et de commentaire). La fusion ne devient effective qu'*après* une revue par l'ensemble des membres du projet. Elle s'opère de cette façon, par un seul participant :

```
# placement dans la branche master
$ git checkout master

# fusion (conflits éventuels à résoudre comme de nouvelles contributions)
$ git merge <branch>

# partage du résultat avec tous les membres du projet
$ git push origin master
```