Guide du débutant Github / Jupyter :

Installer Karstnet et travailler avec (développer)

P. Collon – Aout 2020

Avant-propos

Ce guide a été rédigé pour faciliter la prise en main de codes Python mis à disposition via des plateformes de gestion de versions et de travail collaboratif comme Github. Destiné surtout aux étudiants de master qui auraient à travailler dans ce genre d'environnement, il peut être utile à tout collègue "nouveau" dans cet univers. Tout ceci est illustré ici avec le code *Karstnet* développé en collaboration avec Philippe Renard de l'Université de Neuchâtel. Néanmoins, plusieurs chose peuvent s'appliquer à d'autres codes Python proposés sur Github.

Ce guide est rédigé en français pour faciliter la compréhension par mes élèves des termes techniques nouveaux. Une version anglaise devrait suivre.

Si vous avez des remarques, suggestions de corrections et/ou mise à jour, n'hésitez pas à me contacter : pauline.collon@univ-lorraine.fr

Contenu

1. Pré-requis

Anaconda et Python

Installer Anaconda (python 3), de manière à disposer d'un environnement de développement intégré permettant d'utiliser les notebooks Jupyter : https://www.anaconda.com/products/individual

Github desktop (pour utiliser le versionnage)

Il vous faudra d'abord vous créer un compte sut github.com . Ensuite, je vous conseille d'installer Github desktop : https://desktop.github.com/

Modules extérieurs nécessaires à karstnet

Karstnet utilise actuellement les bibliothèques suivantes :

- Networkx : https://networkx.github.io/documentation/stable/index.html pour la création et gestion des graphes
- Mplstereonet : pour les stéréo https://github.com/joferkington/mplstereonet

Pour ces deux modules, l'installation est très simple grâce au protocole pip :

- Ouvrez une console : Démarrer / Anaconda / Anaconda prompt
- Installez les 2 bibliothèques en écrivant simplement dans la console :

```
pip install networkx
pip install --upgrade mplstereonet
```

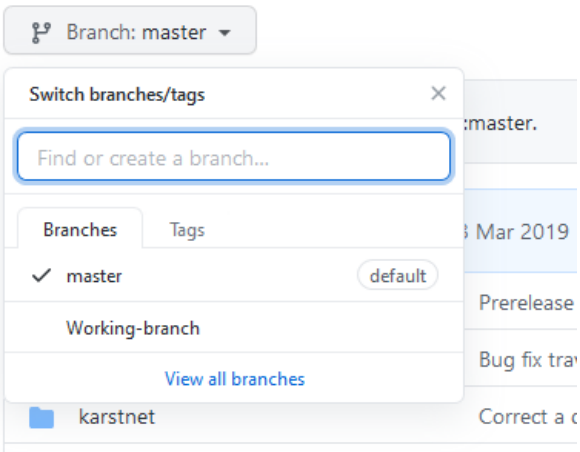
A noter que l'option upgrade garanti l'une des dernières versions de ces librairies, indispensable pour avoir les options offertes par certaines corrections de bugs

2. Création d'une branche & Récupération de karstnet

Lorsqu'on veut travailler de manière collaborative, il est fondamental de ne jamais travailler directement sur la branche principale mais de se créer une branche correspondant à la fonctionnalité sur laquelle on souhaite travailler. Une fois le desktop installé, allez sur la page du projet :

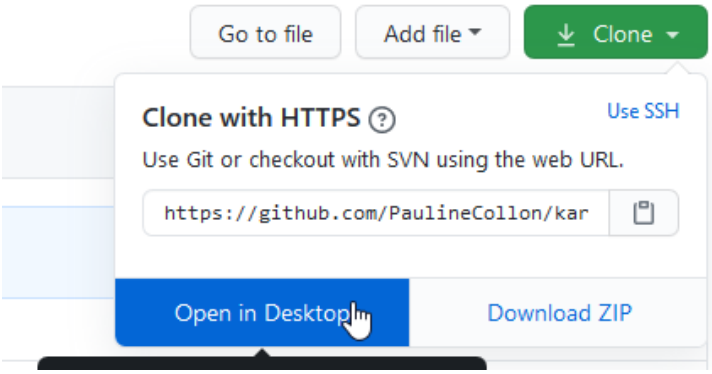
- si vous etes un étudiant ENSG : https://github.com/PaulineCollon/karstnet
- si vous êtes un collaborateur académique : https://github.com/UniNE-CHYN/karstnet

Cliquez sur la flèche pour dérouler le menu « Branch » :

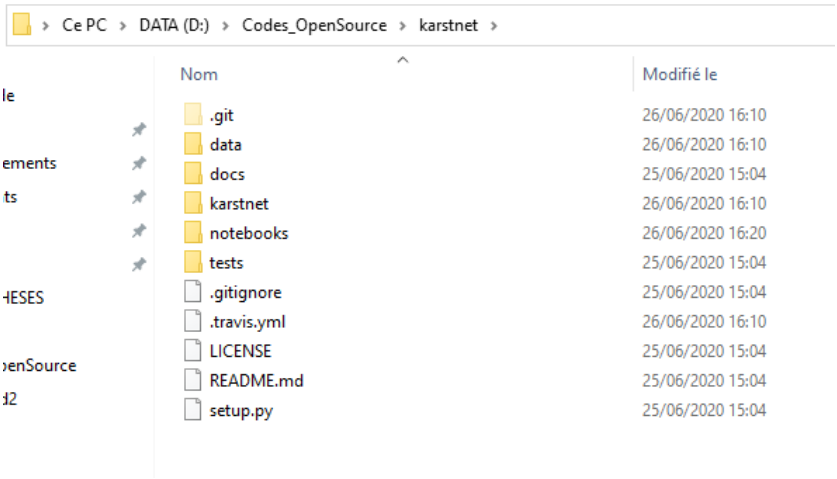


Dans la fenetre de dialogue, créez votre branche : « VotreNom_project »

Une fois la branche créée, vous pouvez cloner le répertoire en allant à droite sur

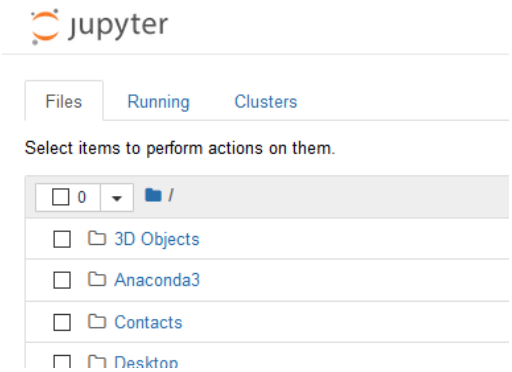


Ceci ouvre la branche dans le GithubDesktop. Lancez le clonage en choisissant judicieusement le dossier où vous souhaitez copier le code. Vous pouvez vérifier dans l'explorateur windows que le clonage a bien fonctionné :



3. Optionnel : Paramétrer Jupyter pour pouvoir lire des notebooks qui ne sont pas sur le disque C

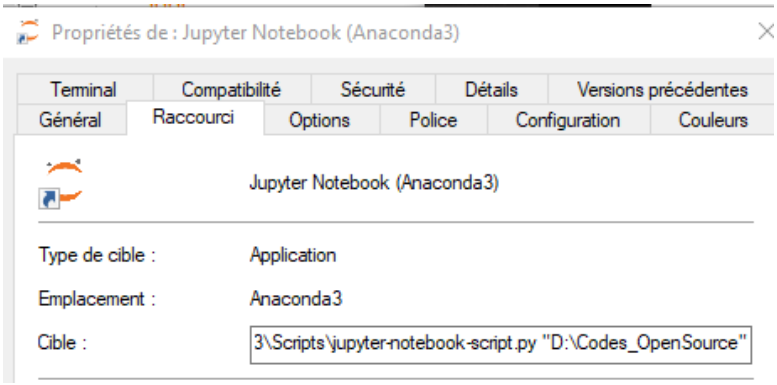
Par défaut, lire des notebooks Jupyter se fait en lançant l'application Jupyter (Démarrer / Anaconda / Jupyter notebooks). Cela ouvre une fenêtre dans votre navigateur web qui est paramétrée pour ouvrir le disque C:/ . De ce fait, il faut alors stocker tous ces notebooks (et donc ses codes) qqpart sur ce disque.



Cette structure ne convient pas forcément à toutes les organisations de travail. Ainsi, personnellement je stocke toutes mes données et mes projets sur un deuxième disque, le disque « D » (ou « E » selon l'ordi mais l'appellation ne change pas grand-chose). Dans ce cas, une manipulation simple permet d'ouvrir Jupyter non pas « sur » l'emplacement par défaut, mais sur un autre disque. Pour cela (windows10):

- Cliquez sur démarrer / Anaconda et cliquez en maintenant appuyé pour copier l'icône de lancement de Jupyter sur le bureau.
- Une fois que le raccourci Jupyter est sur le bureau faites un clic-droit dessus / Propriétés

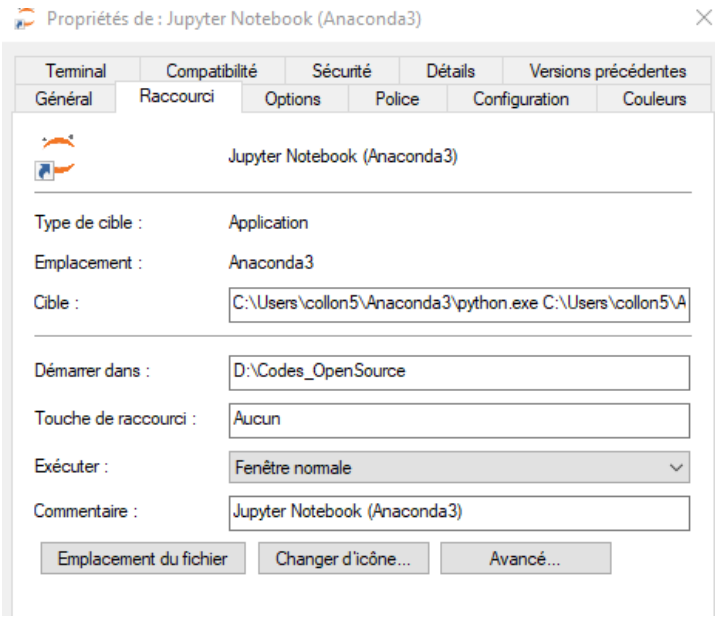
- Choisir l'onglet « Raccourci » et dans la fenêtre « cible : » , changer le chemin à la fin de la commande



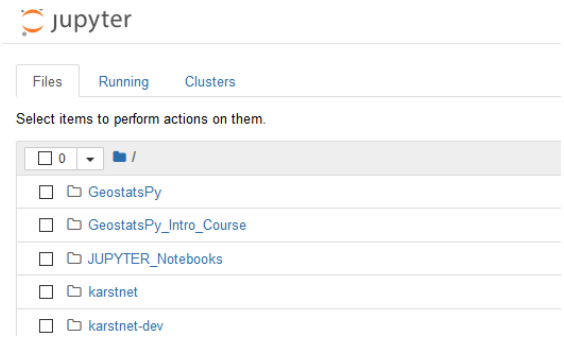
- Dans mon cas la commande est modifiée de la sorte :

C:\Users\collon5\Anaconda3\python.exe C:\Users\collon5\Anaconda3\cwp.py C:\Users\collon5\Anaconda3 C:\Users\collon5\Anaconda3\python.exe C:\Users\collon5\Anaconda3\Scripts\jupyter-notebook-script.py "D:\Codes_OpenSource"

- Et j'ai également modifié l'option « Démarrer dans : » (mais je ne suis pas sûre que ce soit indispensable) :



Si on fait ça, ensuite pour lancer Jupyter sur le bon répertoire il faut simplement double-cliquer sur le raccourci bureau. On voit qu'on lance bien Jupyter au bon endroit :



4. Travailler avec Karstnet et des notebooks

Quelques rappels de python sur la programmation modulaire

En python, vous pouvez comme dans d'autres langages (et c'est même conseillé), séparer votre code dans plusieurs fichiers afin de favoriser la modularité : fonctions et classes peuvent ainsi être appelées par différents programmes.

Si vous souhaitez utiliser une fonction nommée *fc1* que vous avez écrite dans un fichier *mes_fonctions*, il suffit, dans votre notebook (ou votre autre fichier python) d'écrire au début :

```
from mes_fonctions import fc1
```

Si vous souhaitez utiliser toutes les fonctions du fichier *mes_fonctions*, il faut écrire :

```
from mes_fonctions import *
```

Dans ce cas on utilise la fonction fc1 directement :

```
a = fc1()
```

Pour éviter les confusions si un nom est commun à plusieurs fonctions, on préfère souvent utiliser une autre façon d'importer des éléments qui force le renseignement du domaine auquel appartient la fonction :

```
import mes_fonctions
```

Dans ce cas, pour utiliser la fonction fc1, il faudra préciser qu'elle appartient au **module** (= fichier python) *mes_fonctions* ainsi :

```
a = mes_fonctions.fc1()
```

Comme c'est un peu long, on utilise souvent les alias :

```
import mes_fonctions as fc
```

fc est un « alias » choisi par l'utilisateur. Dans ce cas, pour utiliser la fonction fc1, on écrit :

```
a = fc.fc1()
```

Tout ceci ne fonctionne que si tous les fichiers sont situés au même endroit, cad dans le même répertoire. Or, notamment lorsqu'on utilise des notebooks, on peut vouloir utiliser des éléments qui sont rangés « ailleurs ».

Si le notebook est à la racine et que les codes (e.g. *mes_fonctions.py*) sont rangés dans un sous-dossier *package* alors on modifie simplement l'import ainsi :

```
from package.mes_fonctions import *
```

ou

```
import package.mes_fonctions as fc
```

Si maintenant les fichiers de code sont à un autre endroit il convient alors d'utiliser la fonction *chdir* de la bibliothèque *os* pour indiquer le chemin :

```
import os
os.chdir("D:\Codes_OpenSource\package")
import mes_fonctions as fc
```

➔ **Tout ceci s'applique donc à Karstnet et vous pourrez à tout moment tester et utiliser les fonctions de Karstnet pour peu que vous ayez correctement importé le module karstnet au début de votre notebook (ou code python).**

CONSEILLÉ : Utiliser Karstnet grâce à « l'installeur » pip

Option 1 : si vous voulez utiliser Karstnet mais sans en modifier le code

Karstnet peut également être installé grâce à pip. Dans ce cas, ce sera la version principale du projet github karstnet qui sera installée.

- Ouvrez une console : Démarrer / Anaconda / Anaconda prompt
- Déplacez-vous dans le répertoire contenant le projet (par exemple dans mon cas, où un changement de disque est nécessaire cd/d permet le changement de disque, puis on indique le répertoire où aller) :

```
cd/d D:\Codes_OpenSource\karstnet
```

- Installez Karstnet depuis cet emplacement local:

```
pip install .
```

A noter que vous pouvez combiner déplacement et installation en une seule étape :

```
pip install path\to\SomeProject
```

donc dans mon cas:

```
pip install D:\Codes_OpenSource\karstnet
```

Si vous souhaitez désinstaller Karstnet tapez :

```
pip uninstall .
```

Si cette option est pratique pour utiliser Karstnet, il faut bien noter que même si vous en modifiez le code, celui lu par Jupyter ne tiendra pas compte des modifications mais continuera à faire appel à la version originale…

Option 2 : si vous voulez utiliser des versions modifiables de Karstnet

Pour que les modifications locales du code récupéré depuis Github soit effectives dans les notebooks Jupyter, il convient d'installer Karstnet avec l'option –e, qui siginifie qu'on installe le odule en mode « éditable »

```
pip install -e path\to\SomeProject
```

Donc dans mon cas (exemple) :

```
pip install -e D:\Codes_OpenSource\karstnet
```

Ou si vous êtes déjà dans le bon répertoire:

```
pip install -e .
```

Normalement vous obtenez ceci :

```
(base) D:\>pip install -e D:\Codes_OpenSource\karstnet
Obtaining file:///D:/Codes_OpenSource/karstnet
Installing collected packages: karstnet
  Running setup.py develop for karstnet
Successfully installed karstnet
```

Karsnet est maintenant installé. Vous pouvez lancer Jupyter et ouvrir les notebooks de démonstration, et notamment le « simpleGraph_example » : tout devrait fonctionner sans problème.

Développer de nouvelles fonctionnalités pour/avec Karstnet

Qqs rappels de base pour commencer (cf. notebooks exemples)

Si vous n'avez jamais utilisé Jupyter et ses Notebooks, c'est l'occasion ou jamais d'apprendre. Voici un lien vers un site de l'université Paris Diderot qui explique rapidement et efficacement le principe :

https://python.sdv.univ-paris-diderot.fr/18_jupyter/

et sinon il y a toujours OpenClassRoom :

https://openclassrooms.com/fr/courses/4452741-decouvrez-les-librairies-python-pour-la-data-science/5574801-faites-vos-premiers-pas-dans-un-notebook-jupyter

Pour que les fonctionnalités de Karsnet soit accessibles dans le notebook, commencez par importer le module :

```
import karsnet as kn
```

Toutes les fonctions et classes définies dans karsnet peuvent alors etre appelée en faisant précéder le nom de la fonction ou de la classe par l'alias choisi (ici kn), par exemple pour la fonction « test_kn() » qui n'a d'autre intérêt que de vérifier que l'import est correct :

```
kn.test_kn()
```

A noter : pour qu'un changement du code de karsnet soit « mis à jour » dans les notebooks, il faudra systématiquement relancer le « noyau » Jupyter :

Créer une nouvelle fonction qui utiliserait la structure « graphe » d'un réseau karstique :

Faites-vous un nouveau notebook et importez-y Karsnet comme présenté ci-dessus. Vous pouvez alors développer et tester un code python facilement dans ce Notebook.

Lorsque votre code est opérationnel, il vaut mieux alors copier ce code non plus dans un Notebook, mais dans un **nouveau module (un fichier .py)** qui sera donc accessible plus facilement par la suite.

Si vous **rajoutez un module à Karstnet, celui-ci devra être appelé dans le fichier __init__.py**:

```
from karstnet.nouveau_module import *
```

Et pour utiliser ses fonctions et/ou classe depuis un autre module de Karstnet, pensez à l'appeler ainsi :

```
#----Internal module dependancies
from karstnet.nouveau_module import *
```

Formater son code

Pour garantir un code propre et lisible, certaines vérifications automatiques ont été mises en place sur Karstnet, et notamment un test automatique de bon formatage du code. Sans les bons outils, cela peut vite devenir un « cauchemar » mais avec ces outils, non seulement cela devient une promenade de santé mais en plus votre code est propre et lisible sans trop d'effort ☺

La fusion de modifications faites sur une branche avec la branche principale lance en effet les vérifications indexées dans le fichier *.travis.yml*. Parmi celles-ci, la ligne

```
pycodestyle karstnet tests
```

lance donc un test de conformité aux normes PEP8 du code Python. Pour éviter de mauvaises surprises, il est fortement conseillé de faire tous ces tests AVANT en local (éventuellement après un commit pour être sûr de conserver la version avant modifications). Voici la maneouvre à suivre :

1) Si ce n'est déjà fait, installer pycodestyle et un outil d'auto formatage. Pour ma part j'ai essayé Autopep8 avec succès, je vous le recommande donc. :

```
pip install --upgrade pycodestyle
pip install --upgrade autopep8
```

A noter que pycodestyle est souvent déjà installé avec Anaconda, mais au moins vous etes sûrs d'avoir ainsi la dernière version…

2) Lancer un test sur son code :

```
pycodestyle --first D:\chemin\vers\le\fichier\py\a\tester.py
```

Notez que l'option [--first] utilisée ici permet de limiter la liste des « malfaçons » à la première de chaque groupe (puisqu'une erreur sur une ligne se répercute sur la suite)

3) Corriger son code : lancer un outil d'auto formatage (ici autopep 8 avec un niveau d'agressivité de 2) :

```
autopep8 --in-place --aggressive -aggressive
D:\chemin\vers\le\fichier\py\a\tester.py
```

D'expérience, l'outil fonctionne vraiment bien, la seule « erreur » qu'il ne corrige pas est la « ligne trop longue ». Je ne peux que vous conseiller de le faire à la main, voir en cours d'écriture de code.

4) Vérifier le résultat en relançant pycodestyle :

```
pycodestyle --first D:\chemin\vers\le\fichier\py\a\tester.py
```

Tant que des erreurs sont listées, il faut les corriger (cf. lignes trop longues), relancer autopep8 puis pycodestyle. Si la console ne vous affiche aucun message, c'est que tout est bien ☺ ! Bravo, vous êtes prêt à faire une demande de « pull request ».

Gérer les versions : Github

Si vous développez votre code/Notebook, il est important d'utiliser le gestionnaire de version (au moins à peu près) correctement.

Beaucoup de tutoriels expliquent les principes de Github:

https://git-scm.com/book/fr/v2
https://openclassrooms.com/fr/courses/2342361-gerez-votre-code-avec-git-et-github
https://sodocumentation.net/fr/github/topic/1214/demarrer-avec-github

Si vous avez installé Github desktop, vous n'entrerez pas vos demandes en lignes de commandes. En plus de l'évidente doc officielle de Github desktop que je vous recommande pour une utilisation approfondie (premier lien), je vous rajoute un tuto d'intro en français :

https://docs.github.com/en/desktop
https://docs.github.com/en/desktop/contributing-and-collaborating-using-github-desktop/making-changes-in-

a-branch
https://o7planning.org/fr/10283/utiliser-github-avec-github-desktop

Ce que vous devez retenir :

- Toujours **travailler sur une branche**, jamais sur le dépôt principal
- *Commit* = enregistrer les modifications dans le répertoire local, régulièrement (au moins à chaque fin de journée !). Un commit peut aussi être annulé (https://docs.github.com/en/desktop/contributing-and-collaborating-using-github-desktop/reverting-a-commit)
- *Push* = pousser vos changements sur le dépôt Github, là encore à chaque incrément significatif. Si vous travaillez sur votre branche, poussez vos modifications les rend visibles par tous, mais aussi assure une sauvegarde extérieure à vos travaux (sur le net) alors que sinon tout reste sur votre ordinateur, et s'il plante, vous perdez tout… Il ne faut donc pas appréhender cette action. Si vous faites une erreur, il est toujours possible de revenir en arrière… À noter :
    - pousser n'est permis sur github que pour des incréments limités en taille (donc si vous respectez les règles ci-dessus, aucun souci)
    - avant de pousser, Github vous demandera de vérifier si des mises à jour n'ont pas été faites entre temps sur votre branche (*fetch origin*) afin de s'assurer que la fusion est possible. Si cette action est suivie d'une proposition de *pull* (= tirer = récupérer les modifications faites entre-temps sur la branche) vous devez accepter. Soit les modifications faites ne portent pas sur des parties communes, dans ce cas il n'y aura pas de problème, soit elles portent sur des parties communes. Il faudra alors d'abord résoudre les conflits avant de pouvoir continuer (https://docs.github.com/en/github/collaborating-with-issues-and-pull-requests/resolving-a-merge-conflict-on-github)
- *Pull request* = demande de fusion /validation de votre travail sur la branche avec une autre branche (généralement la principale). Ceci ne doit se faire QUE lorsque le travail est abouti, propre et vérifié. Si vous êtes dans le cadre d'un projet étudiant, vous pouvez laisser votre encadrant s'en charger.

Voilà, je pense avoir fait le tour des éléments nécessaires pour bien démarrer. Bon code à tous !