

Suivez le guide !

Ce manuel d'utilisation permet de s'orienter au sein de ce projet.
Il contient l'ensemble des remarques et consignes importantes à garder à l'esprit et à respecter.

Que puis-je faire à l'aide de ce projet ?

C'est un outil permettant de faciliter la création d'une base de données géo-référencées.

Il assiste certaines tâches. Il peut notamment se charger :

- de récupérer les données que l'utilisateur souhaite importer
- d'effectuer certains pré-traitements sur ces données
- de les importer dans une base "prête à requêter"

De plus, le projet contient des scripts d'installation permettant d'effectuer les actions préliminaires de adéquate de façon automatisée.
(pour systèmes *Linux* uniquement)

Avant de commencer

Arborescence du projet

Par souci de clarté / lisibilité, certains fichiers et dossiers ont été omis.
les éléments utiles --> **ont une description**

|—— README.pdf --> **fichier "lisez-moi"**
|—— documents

```

| ..... |—— in
| ..... |—— out
| ..... |—— reflexions
| ..... | ..... |—— choix
| ..... | ..... |—— procedures
| ..... | ..... |—— creation_fichier_import
| ..... | ..... | ..... |—— example.xlsx --> fichier excel type
| ..... | ..... |—— miniconda
| ..... | ..... | ..... |—— desinstallation.pdf
| ..... | ..... |—— postgresql
| ..... | ..... | ..... |—— desinstallation.pdf
| ..... | ..... |—— utilisation
| ..... | ..... |—— manuel_utilisateur.pdf --> ce manuel
| ..... |—— rendus
|
|—— src --> racine du code source
..... |—— python
..... | ..... |—— config.py
..... | ..... |—— main.py --> outil principal
..... | ..... |—— packages
..... |—— res
..... | ..... |—— gd.yml --> env. conda à importer
..... |—— scripts
..... |—— create_venv.sh --> regénère le venv en .yml
..... |—— excel2conf.py --> excel vers json de config
..... |—— prepare_database.sh --> installe / configure postgresql
..... |—— prepare_environment.sh --> prépare l'env. d'exécution

```

Quelques indications et consignes

Ne pas altérer de quelque manière que ce soit (modification, suppression,

déplacement, création, etc.) la structure du projet dans le répertoire `src` . En effet ce dossier contient du code source et sa structure ainsi que ses fichiers et leurs contenus sont nécessaires au fonctionnement du programme. Ne pas y toucher, donc. (*sauf indication contraire*)

Prérequis

Pour une installation à partir de zéro

(À n'effectuer que la toute première fois.)

- un système d'exploitation *Linux*
- un système propre, c'est-à-dire à jour et fonctionnel (*comme pour tout logiciel*)
- des droits administrateur, "**root**", afin de pouvoir effectuer des `sudo` lors de la phase d'installation
- une connexion internet

Pour l'exécution de l'outil seulement

- n'importe quel système d'exploitation (seul *Ubuntu 16.04 LTS* a été testé)
- avoir effectué les installations de la phase préparatoire (base de données **postgresql** + **environnement d'exécution**)
- éventuellement une connexion internet s'il faut récupérer des données distantes

Phase préparatoire

Préparation de la base

Si une ou plusieurs base(s) `postgresql` sont d'ores-et-déjà présente(s),

ignorer cette étape en passant directement à la suivante. S'adapter en conséquence.

Il ne doit y avoir aucune trace résiduelle d'une quelconque installation antérieure ou actuelle de postgresql/postgis (aller consulter `documents/out/reflexions/procedures/postgresql/desinstallation.pdf` et suivre les consignes pour le vérifier !).

Une fois ces vérifications effectuées, naviguer dans le projet pour se rendre dans le dossier qui regroupe l'ensemble des scripts (`src/scripts`).

Éventuellement, modifier le nom de la base ainsi que le mot de passe pour l'utilisateur `postgres` comme désiré.

Les deux variables concernées se situent en haut du script.

Pour finir, lancer le script en suivant les indications qui apparaissent.

```
sudo ./prepare_database.sh
```

Préparation de l'environnement d'exécution

La commande `conda` (*anaconda* ou *miniconda*) ne doit pas être installée. Taper `conda --version` pour le vérifier.

Si `conda` est d'ores-et-déjà installé :

- obtenir la liste des environnements présents avec `conda info --envs`
- aller regarder le nom dans `src/res` du fichier en `.yaml` , c'est le nom de l'environnement que l'on souhaite importer
- vérifier qu'un environnement de la liste ne porte pas déjà ce nom, sinon il va y avoir un conflit
- si c'est le cas, il faut changer le nom de l'environnement importé en changeant le nom du fichier `.yaml` , ainsi que les occurrences de son

nom au sein du fichier lui-même (faire un rechercher/remplacer)

- importer cet environnement avec la commande `conda env create -f le_nouveau_nom.yml`
- l'activer en tapant `. activate le_nouveau_nom`

Sinon, il faut seulement lancer le script approprié :

```
./prepare_environment.sh
```

Phase d'exécution

Création d'une configuration d'import

Il y a deux manières de fabriquer un fichier de configuration d'import.

La première méthode, assez terre-à-terre, est de l'écrire à la main. Pour cela, faire l'étape 1 du lancement ci-dessous, puis modifier le fichier `/srv/geodata/configuration/data.conf.json`. Ce fichier, copié à partir de `src/res/data.conf.json`, contient une node **documentation**. Lire son contenu avant de commencer la rédaction du fichier de configuration.

La seconde méthode permet de renseigner les imports d'une manière plus commode : renseigner les imports dans un fichier excel. Suivre la procédure suivante :

- prendre pour point de départ le fichier type `documents/out/reflexions/procedures/creation_fichier_import/example.xlsx`
- le modifier en calquant sa syntaxe et son formatage
attention à la respecter attentivement
écueils les plus courants :
 - cellules vides pas réellement vides " "

- saut de ligne au sein d'une cellule
- séparateur en trop (, ou ;)
- lancer le script de conversion comme suit

```
src/scripts/excel2conf.py /path/absolu/excel.xlsx
```

qui va générer un fichier en `.data.conf.json` dans le même répertoire que le fichier excel source
- l'ouvrir pour vérifier et corriger
en effet certaines fautes sont difficilement visibles dans le fichier excel
exemple : un saut de ligne provenant d'un copier/coller
une fois transcrit en JSON, les erreurs se remarquent plus facilement

Lancement

1. première exécution "à vide" afin de générer l'arborescence de travail (par défaut dans `/srv`)
2. fabriquer un fichier de configuration : voir "Préparation au lancement"
3. le déplacer dans `/srv/geodata/configuration/` et le nommer `data.conf.json` Éventuellement conserver le `data.conf.json` existant : le déplacer et/ou le renommer.
4. exécution de l'outil
 - aller dans `src/python`
 - si la commande `which python` n'indique pas `nom_user/miniconda/envs/nom_environnement/bin/python` alors activer l'environnement virtuel conda

```
. activate nom_environnement
```
 - lancer le programme `./main.py`

Pour importer à nouveau des données, deux choix :

- réitérer les étapes **1**, **2** et **3** ci-dessus
- réitérer l'étape **1** puis ajouter dans le `data.conf.json` déjà en place les imports du fichier fabriqué

Vérifications

Beaucoup d'erreurs de causes multiples sont susceptibles de survenir pendant l'exécution. Il est donc souvent nécessaire d'effectuer de multiples tentatives pour une même donnée à importer. En effet, à chaque essai et jusqu'à réussite, il faut modifier la configuration associée à cette donnée. Pour cela se référer aux messages de journalisation affichés dans le terminal.

La liste des données ayant été traitées complètement, c'est-à-dire importées en base avec succès, se trouve dans

`/srv/geodata/configuration` . Cette liste n'a pas vocation à être seulement lue. En effet, elle peut être modifiée, par exemple pour réitérer un import déjà effectué. Le programme se base sur ce fichier pour déterminer si une donnée a été importée précédemment avec succès. Ainsi il suffit de supprimer la ligne correspondant au chemin de la donnée à importer à nouveau.