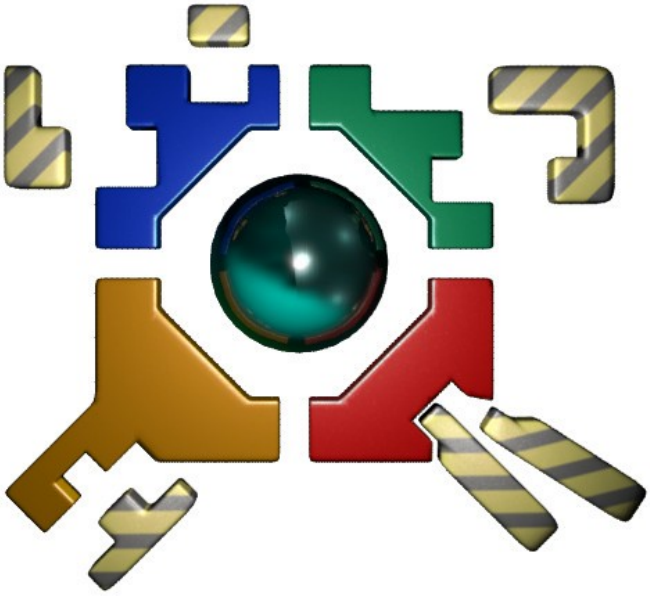


De manière générale comment fonctionne GDeps?

Generate **Dep**endencie**s** (http://gdeps.org) est un projet open source sous licence MIT, écrit en python 3.

Son état d'avancement en est à la version pré-alpha.

Sommaire

GDeps:

Site web: http://gdeps.org/

Auteur: Pierre Pontier

API: http://gdeps.org/doc/index.html

Forum: http://gdeps.org/forum/

Sources: https://sourceforge.net/p/gdeps/mercurial/ci/default/tree/

Le concept:

GDeps permet de compiler vos dépendances ou applications C++ sur les plate-formes Windows à l'aide d'un script spécifiquement écrit pour le projet. Le script est écrit en python 3 et utilise l'API de GDeps.

L'objectif n'est pas de réécrire la solution pour un IDE ou la configuration d'un projet CMake. L'objectif est d'utiliser un script python qui va faire son possible pour générer les binaires d'un projet à l'aide d'une solution tel qu'un «.sln» ou d'un projet tel que CMake.

L'idée est de définir un environnement de compilation unique pour un ensemble de librairies ou d'application écrites en C++.

Au final et en admettant que chaque année vous devez mettre à jour vos dépendances sur un gros projet, vous gagnerez peut-être 1 mois de travail par an. Car toutes vos dépendances se mettront à jour et compileront d'un simple clique.

Pour ce faire chaque projet à compiler est défini par un script qui exécutera le processus suivant.

1. Config: GDeps lie la configuration de l'environnement de travail.
    - Il lit la définition de l'emplacement des applications de contrôle de version. Elles seront utilisées pour télécharger vos projets cvs, svn, git et mercurial.
    - Il lit la définition de l'emplacement de 7zip si besoin. A savoir que l'implémentation de zlib en python pourra être aussi utilisé pour extraire des archives qui seront téléchargées.
    - Il lit la définition de l'emplacement des compilateurs et des IDE avec ou sans options de configuration.
    - Il lit la définition de l'emplacement des générateurs de solutions comme CMake ou comme «boost.build».
2. Update: Si besoin GDeps téléchargera un ou plusieurs dépôts.
    - Et si besoin il téléchargera et extraira une archive.
3. Make: Si besoin GDeps générera les solutions ou comme dans le cas de boost.build et bakefile compiler directement les binaires.
4. Build: Si besoin GDeps compilera les binaires via les IDE tel que CodeBlocks ou Visual C++.

5. Report: GDeps génère un rapport d'erreurs et d'avertissements. Il génère aussi une archive des binaires crées.

Installation:

Installer la librairie GDeps:

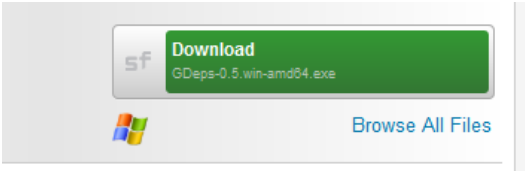
Les scripts d'un projet GDeps utilisent la librairie GDeps. Pour l'installer rendez vous sur https://sourceforge.net/projects/gdeps/

Pour se faire il y a deux méthodes soit vous installer GDeps via l'installateur soit via le code source.

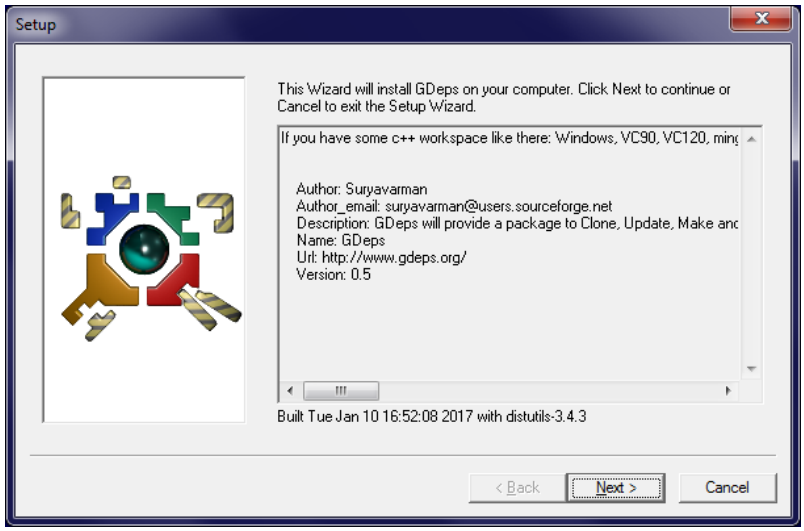
Si vous souhaitez être sûr d'utiliser la dernière version de GDeps reportez vous à la section # Installer GDeps via les sources

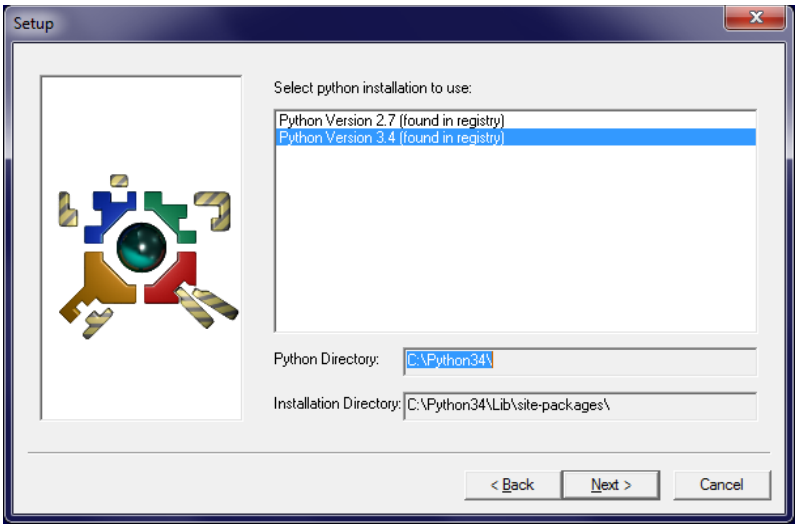
Installer GDeps via l'installateur:

Ça ne sera pas forcément la dernière version mais vous n'aurez pas à cloner le dépôt.



Une fois télécharger suivez les indications d'installation.





Choisissez votre version de Python 3 et cliquez sur suivant et finalisez l'installation.

A partir de cette étape vous avez installez la libraire GDeps au sein de votre version de python 3.

Installer GDeps via les sources:

Il s'agit de la dernière version du code de GDeps.

GDeps utilise mercurial pour gérer les version des fichiers.

Vous devez donc avoir installer mercurial sur votre machine.

https://www.mercurial-scm.org/

Si vous ne souhaitez pas utiliser mercurial via un terminal mais via une GUI je vous conseil TortoiseHg. Pour la suite nous utiliserons TortoiseHg. Vous pouvez le télécharger via le lien suivant.

http://tortoisehg.bitbucket.org/download/

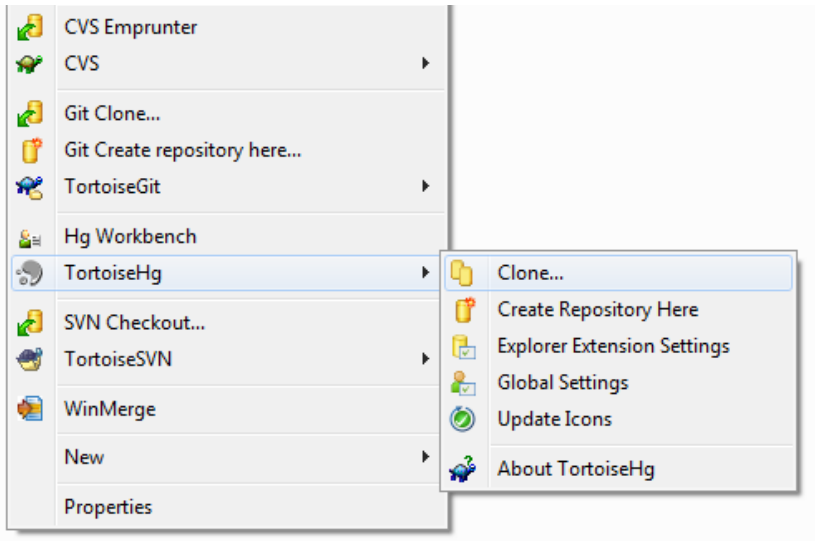
Une fois installé rendez vous sur https://sourceforge.net/p/gdeps dans la section code, vous y verrez la ligne de commande à exécuter pour cloner le dépôt.

```
hg clone http://hg.code.sf.net/p/gdeps/mercurial gdeps-mercurial
```

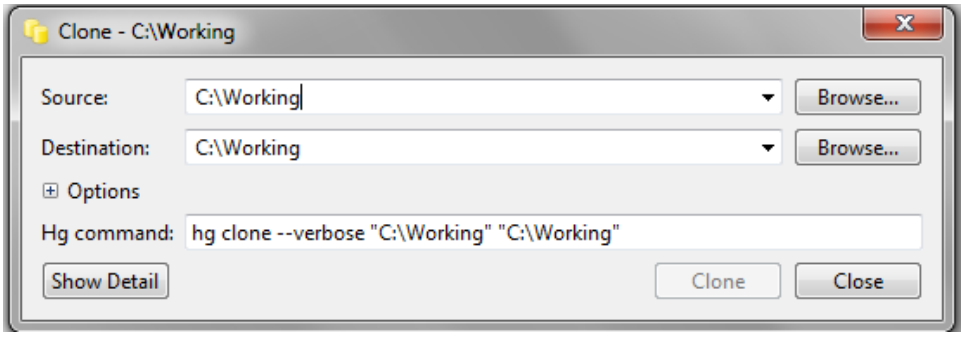
Vous avez par défaut le lien RO (read only) pour la lecture seul. C'est cette adresse que nous allons utiliser.

```
http://hg.code.sf.net/p/gdeps/mercurial
```

Dans le dossier qui contiendra le dossier GDeps. Faites un clic droit et sélectionné TortoiseHg > Clone... comme sur l'image suivante.



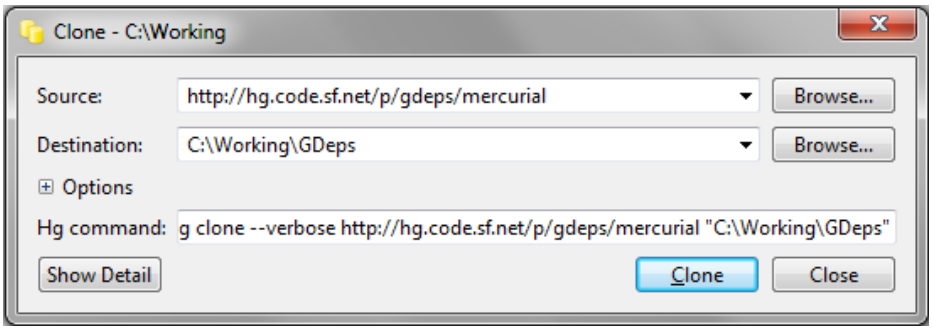
Vous arrivez alors sur cette fenêtre :

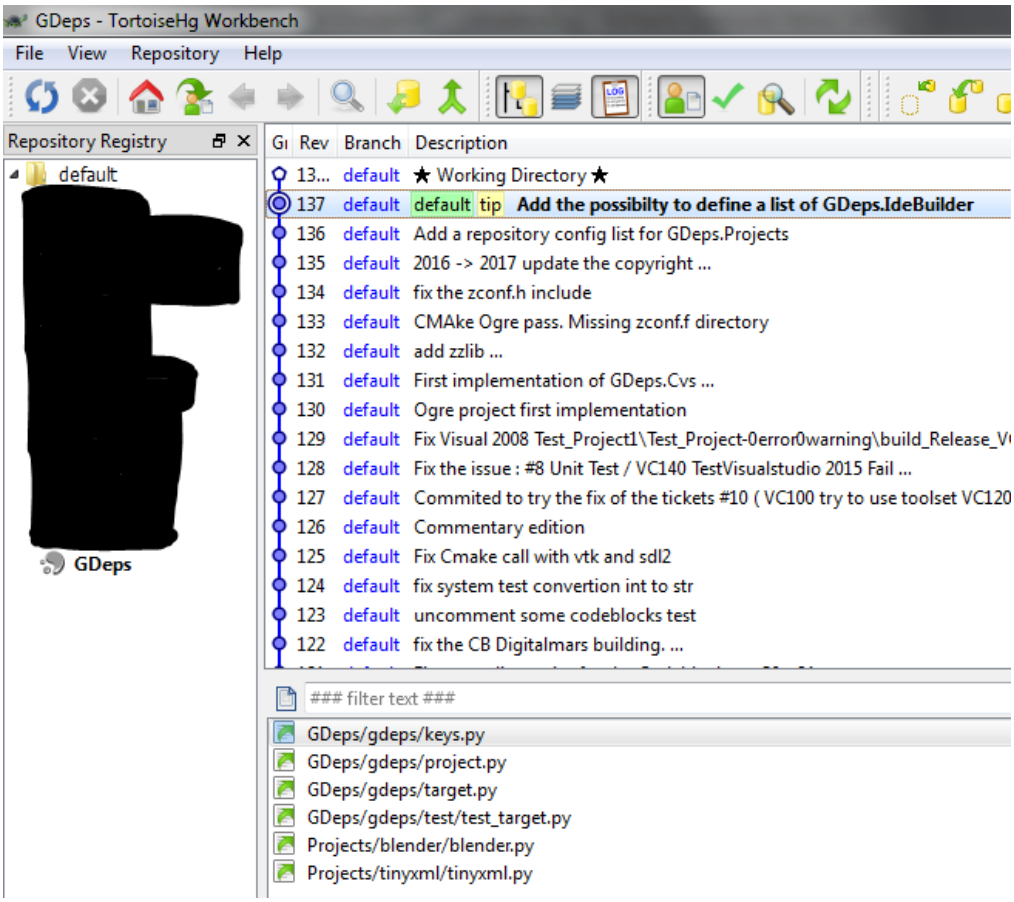


Pour la source utilisez le lien suivant :

http://hg.code.sf.net/p/gdeps/mercurial

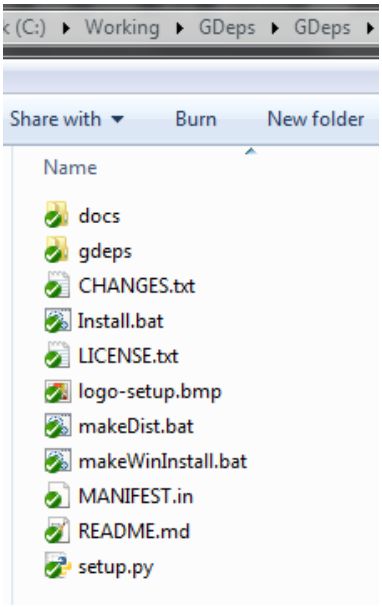
Pour la destination écrivez le chemin du dossier qui sera créé par mercurial pour y déposer les sources.





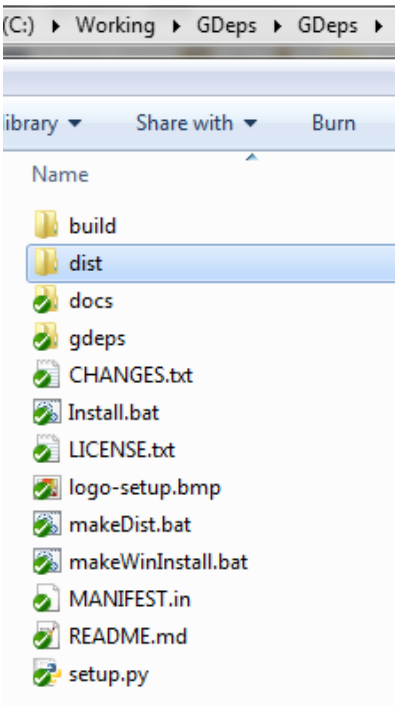
Une fois les sources acheminées, une fenêtre du «Workbench» de ToirtoiseHg s'affiche. Maintenant il ne nous reste que à installer la librairie dans notre distribution de Python 3.

Pour se faire allez dans le dossier nouvellement créé et ouvrez le sous dossier GDeps.

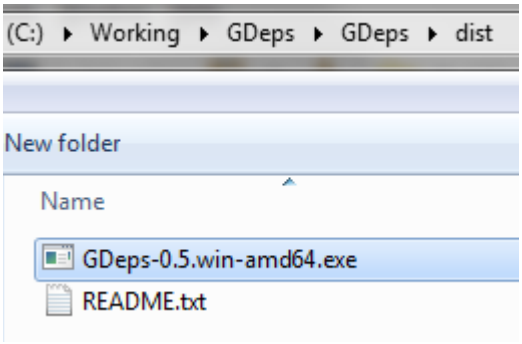


Vous aurez juste à exécuter «Install.bat» pour installer GDeps dans votre distribution de python 3.

Si vous souhaitez installer GDeps dans une distribution particulière de python3, exécutez «makeWinInstall.bat» cela vous générera un dossier «build» et «dist».



Ouvrez le dossier «dist» et exécutez l'installateur.



Ensuite référez vous aux instructions de la section # Installer GDeps via l'installateur:

Comment utiliser GDeps :

API :

GDeps est une API. La documentation est générée par Sphinx. Elle est publiée à l'adresse suivante :

http://gdeps.org/doc/index.html

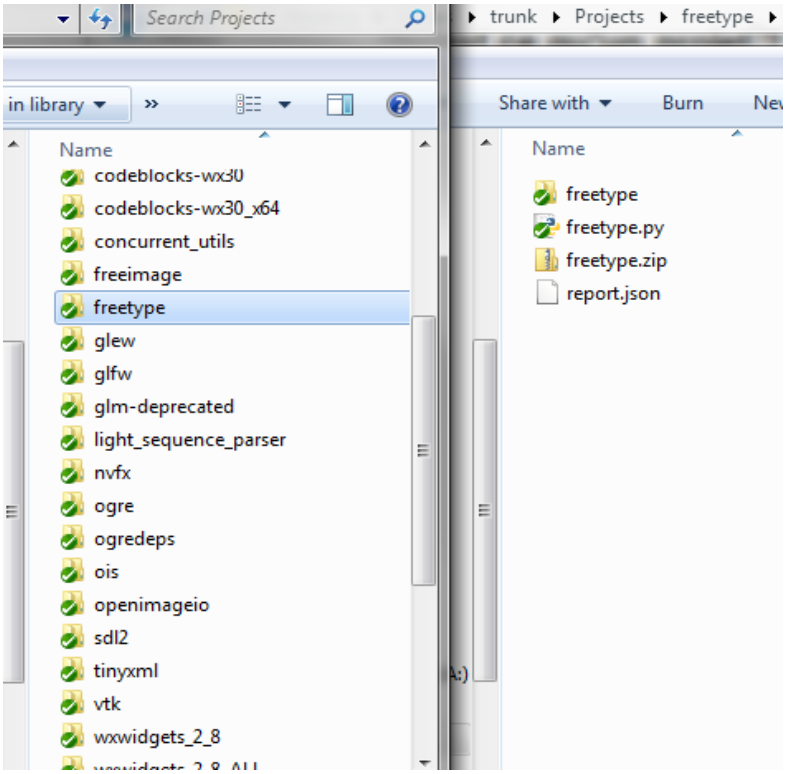
Si vous souhaitez écrire une question, suggestion etc. Vous pouvez vous inscrire sur le forum de gdeps.org.

http://gdeps.org/forum/

Qu'est ce qu'un projet ?

On utilise le terme «Projet» pour définir le script python utilisant GDeps pour télécharger et construire, une librairie ou un exécutable. Chaque projet porte le nom de la librairie ou de l'application à construire.

Ci-dessous un exemple avec la librairie freetype:



L'exécution du script a généré un sous-dossier «freetype», une archive contenant les binaires et un rapport des erreurs et des avertissements au format «json».

Certains des projets ont leur rapport et leur binaire publiés à l'adresse suivante :

http://gdeps.org/reports/

Voici le rapport de «freetype»:

http://gdeps.org/reports/report.php?folder=freetype

et voici les binaires :

http://gdeps.org/reports/freetype/freetype.zip

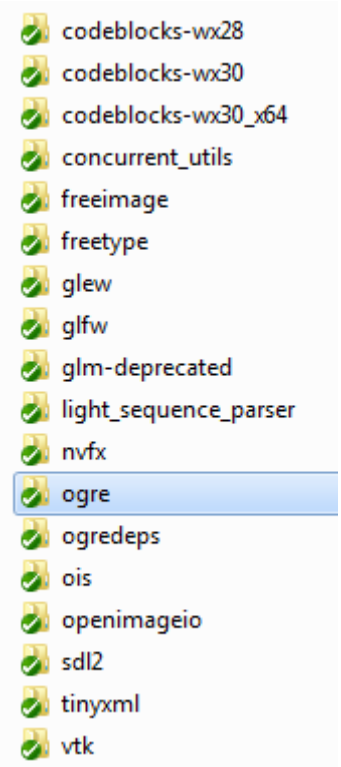
Préparation de l'architecture des dossiers :

Avant de télécharger il va vous falloir penser à où voudriez-vous le dossier contenant toutes vos dépendances.

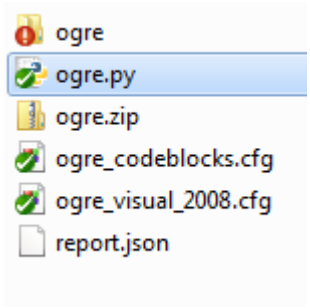
Les projets GDeps par défaut sont tous dans un répertoire commun qui se nomme «Projects». Chaque dossier de projet porte le même nom que le script qu'il contient. Les script utilisent des dépendances relatives aux chemin du fichier python.

Ce qui visuellement donne ceci :

Contenu du dossier commun «Projects» :



Contenu du dossier «ogre» :

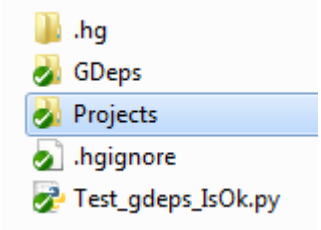


Contenu du script «ogre.py» avec la définition des chemins mise en évidence. Il faut remarquer que les chemins des dépendances sont définis relativement au script de celui-ci.

```
#!python3

# Copyright 2007-2017 Gemr. All Rights Reserved.
# Licensed to MIT see LICENSE.txt

import os
import shutil
import gdeps

__author__ = 'Suryavarman (http://sourceforge.net/u/suryavarman/profile/)'

script_dir = os.path.dirname(os.path.abspath(__file__))
parent_dir = os.path.normpath(script_dir + r"\\..\\")
config_file_path = parent_dir + r"\\Config.cfg"
# config_file_path = script_dir + r"\\ogre_codeblocks.cfg"
# config_file_path = script_dir + r"\\ogre_visual_2008.cfg"
config_file = gdeps.ConfigFile(config_file_path)
folder_dir = script_dir + r"\\ogre"

params = {}

params["config_filepath"] = config_file_path
params["folderdir"] = folder_dir
params["repo_alias"] = "hg"
params["repo_url"] = "http://bitbucket.org/sinbad/ogre/"  # http://www.ogre3d.o
params["repo_cloneargs"] = "-u v1-9"
params["repo_updateargs"] = "v1-9"
params["maker_alias"] = "cmake"
params["maker_solutionname"] = "OGRE"

targets = []

# Dependencies directories:
boost_dir = parent_dir + r"\\boost_1-55\\boost_1-55"
ogre_deps_path = os.path.normpath(parent_dir + r"\\ogredeps\\ogredeps")
freeimage_deps_path = os.path.normpath(parent_dir + r"\\freeimage\\freeimage")
freetype_deps_path = os.path.normpath(parent_dir + r"\\freetype\\freetype")
zlib_deps_path = os.path.normpath(parent_dir + r"\\zlib\\zlib")
zziplib_deps_path = os.path.normpath(parent_dir + r"\\zziplib\\zziplib")
ois_deps_path = os.path.normpath(parent_dir + r"\\ois\\ois")
tinyxml_deps_path = os.path.normpath(parent_dir + r"\\tinyxml\\tinyxml")
directory_deps = os.path.normpath(folder_dir + r'\\Dependencies')

"""
```

Téléchargement d'un projet :

Si vous avez cloner le dépôt vous trouverez la liste des projets dans le dossier «Projects».

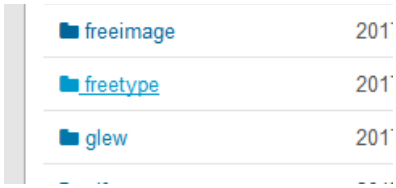


L'autre manière est de télécharger seulement les projets que l'on souhaite.

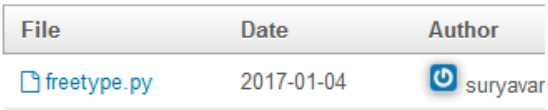
La liste des projets existants est disponible via le lien suivant :

https://sourceforge.net/p/gdeps/mercurial/ci/default/tree/Projects/

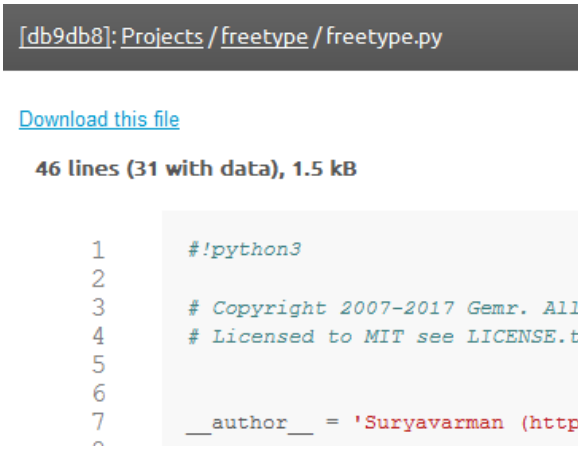
Pour télécharger l'une de ces projets cliquer sur le dossier de l'un d'eux comme sur l'exemple ci-dessous :



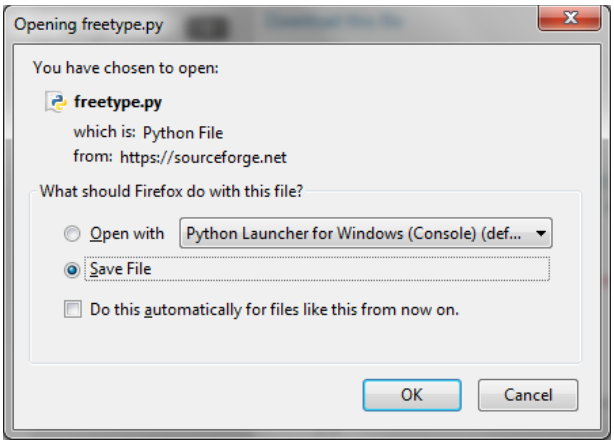
Cliquer sur le script python portant le nom du projet :

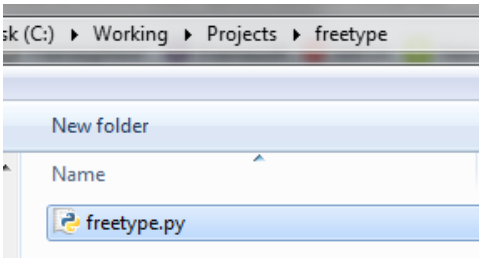


Cliquer sur le lien «Download this file» :



Sauvegarder le script dans un dossier portant le même nom.





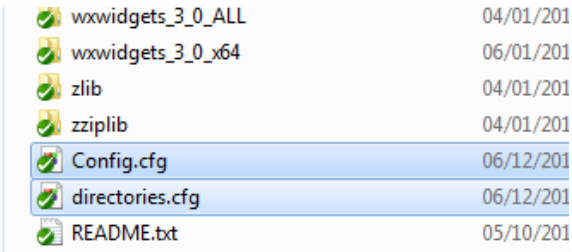
Configuration de l'environnement de travail :

Emplacements des fichiers de configuration :

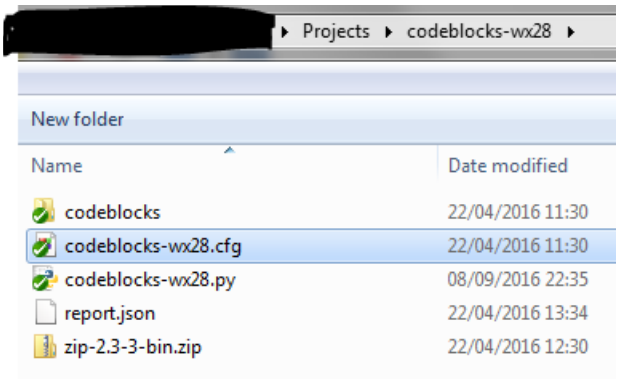
Par défauts il y a deux fichiers de configuration:

1. directories.cfg
2. Config.cfg

Ils sont tout les deux situés au même niveau que les dossiers des projets.



Pour certains il est nécessaire ou plus intéressant d'avoir un fichier de configurations spécifique. Celui là se situ alors à l'intérieur d'un dossier de projet au même niveau que le script python.



Références:

API: http://gdeps.org/doc/gdeps.html#module-gdeps.config

Code: gdeps.Config

Description des chemins :

«directories.cfg» est le fichiers par défaut qui regroupe tout les chemins nécessaire à la définition de l'environnement de travail.

Voici celui correspondant à mon environnement de travail pour le développement de GDeps.

https://sourceforge.net/p/gdeps/mercurial/ci/default/tree/Projects/directories.cfg

```
[cb_32]
dir = C:\CodeBlocks
appdatacbdir = C:\\Users\\Gandi\\AppData\\Roaming\\codeblocks

[vc2008_32]
dir = C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\vcpackages

[vc2010_32]
dir = C:\Windows\Microsoft.NET\Framework\v4.0.30319

[vc2012_32]
dir = C:\Windows\Microsoft.NET\Framework\v4.0.30319
```

Il s'agit d'associé un chemin à un nom.

Pour se faire on définit un alias/ une balise. Par exemple pour la version 32bits de «Codeblocks», j'ai utilisé l'alias [cb_32]. Le choix du nom n'a pas d'importance, hormis qu'il doit d'écrire un tant soit peu ce que représente le chemin.

Chaque balise contiendra une association clefs et de valeurs.

La clef «dir» est utilisée pour définir le chemin de l'application.

Certains objets de l'api nécessitent de connaître des chemins suplémentaires. Par exemple Code-blocks nécessite de savoir si il va utilisé le fichier de configuration par défaut situé dans AppData ou un autre qu'on va lui spécifier.

La clef «appdatacbdir» est définit par la variable gdeps.Codeblocks.ms_KeyAppdataCBDir

https://sourceforge.net/p/gdeps/mercurial/ci/default/tree/GDeps/gdeps/codeblocks.py#l77

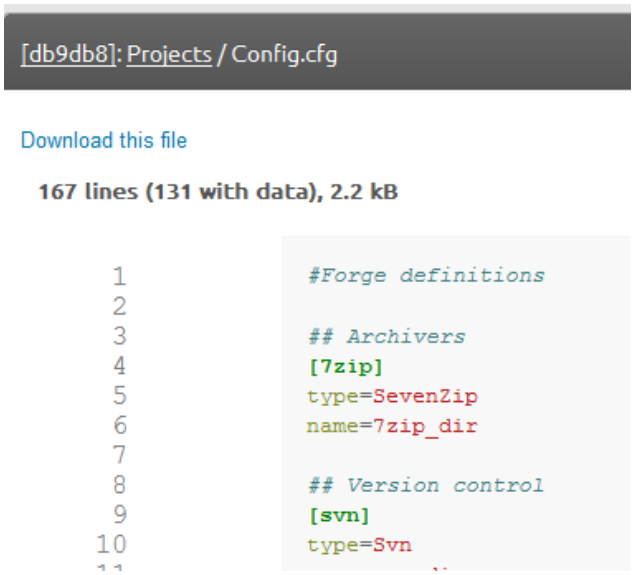
Attention aux «backslash», parfois ces deux «\\» parfois c'est un «\» . A terme ça sera toujours deux.

Configuration globale :

Par défaut tout les projets utilisent la même configuration «Config.cfg». Celle-ci se situe au même niveau que «directories.cfg».

Vous pouvez retrouvez la configuration que j'utilise pour tester mes projets, via le lien suivant :

https://sourceforge.net/p/gdeps/mercurial/ci/default/tree/Projects/Config.cfg



Le but de ce fichier est d'associer un Objet de GDeps aux alias définis dans «directories.cfg».

Le nom des balises est important TODO

Cette configuration est réparties en plusieurs sections. Les applications pour gérer les fichiers compressés, les applications pour gérer les dépôts, les applications pour construire les projets, les compilateurs, les ides et les autres fichiers de configuration à inclure.

Archivers:

Est la section pour définir les applications qui pourront être utilisées pour extraire les archives.

Actuellement seul 7Zip est défini.

```
## Archivers
[7zip]
type=SevenZip
name=7zip_dir
```

Références:

API: http://gdeps.org/doc/gdeps.html#gdeps.sevenzip.SevenZip

Code: gdeps.SevenZip

Revision control:

Est la section pour définir Svn, Git, Mercurial et CVS.

```
## Version control
[svn]
type=Svn
name=svn_dir

[git]
type=Git
name=git_dir

[hg]
type=Mercurial
name=hg_dir

[cvs]
type=Cvs
name=cvs_dir
```

Références:

API:

http://gdeps.org/doc/gdeps.html#gdeps.hg.Mercurial

http://gdeps.org/doc/gdeps.html#gdeps.git.Git

http://gdeps.org/doc/gdeps.html#gdeps.svn.Svn

http://gdeps.org/doc/gdeps.html#gdeps.cvs.Cvs

Code:

gdeps.Mercurial

gdeps.Git

gdeps.Svn

gdeps.Cvs

Makers:

C'est la section des applications qui permettent de générer des projets pour des ide ou qui s'occupent de compiler directement en fonction de la définition d'un workspace.

```
## Maker
[cmake]
type=CMake
name=cmake_dir
```

Pour la liste des projets actuelle seul CMake a été défini. Je n'ai pas eu à définir le maker boost.build.

Si vous avez un projet qui n'intègre pas boost.build vous pourrez trouver un exemple de configuration ici :

https://sourceforge.net/p/gdeps/mercurial/ci/default/tree/GDeps/gdeps/test/test_boost.cfg

```
## Makers
[boost]
type=boost2
name=boost_build_dir
```

Et ici vous retrouverez l'alias pour le chemin d'accès :

https://sourceforge.net/p/gdeps/mercurial/ci/default/tree/GDeps/gdeps/test/directories.cfg#l97

```
[boost_build_dir]
#exepath=E:\\Working\\boost-build\\build
exepath=E:\\Working\\boost-build\\build-develop
```

On remarquera que le maker bakefile ne nécessite pas d'application pour générer les binaires.

Références:

API:

http://gdeps.org/doc/gdeps.html#module-gdeps.cmake

http://gdeps.org/doc/gdeps.html#module-gdeps.boost2

http://gdeps.org/doc/gdeps.html#module-gdeps.bakefile.Bakefile029

Code:

gdeps.CMake

gdeps.boost2

gdeps.Bakefile029

Compilers:

Emplacement pour définir les compilateurs qui seront utiliser par les «makers» et les «ide».

```
## Compilers
[mingw32]
type=Mingw_32
name=mingw_32

[mingw64]
type=Mingw_64
name=mingw_64

[compiler_0]
type=Mingw_32
name=mingw_32

[compiler_1]
type=VC120_32
name=vc120_32

[compiler_2]
type=VC90_32
name=vc90_32

[compiler_3]
type=VC110_32
name=vc110_32

[compiler_4]
type=VC140_32
name=vc140_32

[compiler_5]
type=Mingw_64
name=mingw_64

[compiler_6]
type=VC120_64
name=vc120_64
```

Les «makers» objet dérivant de gdeps.Make peuvent avoir besoin d'une liste de gdeps.Compiler pour générer les binaires des projets. La liste qu'ils vont utiliser sera celle définie par les balises [compiler_ 0], [compiler_1], etc.

Références:

API:

http://gdeps.org/doc/gdeps.html?highlight=compiler#module-gdeps.compiler

http://gdeps.org/doc/gdeps.html?highlight=compiler#module-gdeps.clang

http://gdeps.org/doc/gdeps.html?highlight=compiler#module-gdeps.mingw

http://gdeps.org/doc/gdeps.html?highlight=compiler#module-gdeps.vc

http://gdeps.org/doc/gdeps.html?highlight=compiler#module-gdeps.dm

Code:

gdeps.Compiler

gdeps.Clang_32

gdeps.Clang_64

gdeps.Mingw_32

gdeps.Mingw_64

gdeps.VC90_32

gdeps.VC100_32

gdeps.VC100_64

gdeps.VC110_32

gdeps.VC110_64

gdeps.VC120_32

gdeps.VC120_64

gdeps.VC140_32

gdeps.VC140_64

gdeps.Dm_32

Ides:

Emplacement pour définir la liste des ide que vous souhaitez utiliser .

Pour définir

Les «makers» objet dérivant de gdeps.Make peuvent avoir besoin d'une liste de gdeps.Ide pour compiler les projets. La liste qu'ils vont utiliser sera celle définie par les balises [ide_ 0], [ide_1], etc.

```
[ide_0]
type=CodeBlocks
name=cb_32
compiler_alias_0=compiler_0
compiler_alias_1=compiler_5

[ide_1]
type=Visual2008
name=vc2008_32
compiler_alias_0=compiler_2

[ide_2]
type=Visual2013
name=vc2013_32
compiler_alias_0=compiler_1
compiler_alias_1=compiler_6

[ide_3]
type=Visual2012
name=vc2012_32
compiler_alias_0=compiler_3
compiler_alias_1=compiler_7

[ide_4]
type=Visual2015
name=vc2015_32
compiler_alias_0=compiler_4
compiler_alias_1=compiler_8

[ide_5]
type=Visual2010
name=vc2010_32
compiler_alias_0=compiler_9
compiler_alias_1=compiler_10
```

Références:

API:

http://gdeps.org/doc/gdeps.html?highlight=compiler#module-gdeps.ide

http://gdeps.org/doc/gdeps.html?highlight=compiler#module-gdeps.codeblocks

http://gdeps.org/doc/gdeps.html?highlight=compiler#module-gdeps.visual

Code:

gdeps.Ide

gdeps.Codeblocks

gdeps.Visual2008

gdeps.Visual2010

gdeps.Visual2012

gdeps.Visual2013

gdeps.Visual2015

Includes:

C'est ici que l'on indique le chemin du fichier «directories.cfg»

Configuration locale:

Pour plusieurs raisons on peut souhaiter utiliser un fichier de configuration spécifique pour un projet. Dans ce cas le fichier de configuration sera par défaut à côté du script python du projet.

Voici un exemple avec le projet wxWidgets 2.8:

File	Date
Config_Two_Compilers.cfg	2016-10-05
wxwidgets_2_8.py	2017-01-04

wxWidget ne peut généré que deux sorties une pour Mingw et une pour Visual C++.

Au-delà les binaires seront écrasés car il n'y a pas de sorties distinctes pour chacune des versions des compilateurs.

Il faut donc spécifier au «Maker» de wxWidget seulement deux compilateurs.

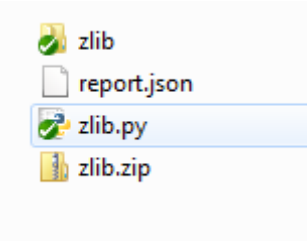
```
#Forge definitions

## Version control
[svn]
type=Svn
name=svn_dir

# Compilers
[compiler_0]
type=Mingw_32
name=mingw510_32

[compiler_1]
type=VC120_32
name=vc120_32

[includes]
include_directories = ./../directories.cfg
```

Exécution d'un projet:

Rien de plus simple vous pouvez double cliquer sur le script du projet.

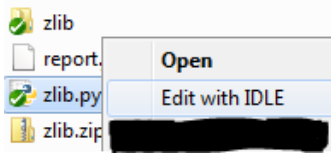
1. Un rapport Json sera généré.

2. Une archive avec les binaires et les sources sera générée.

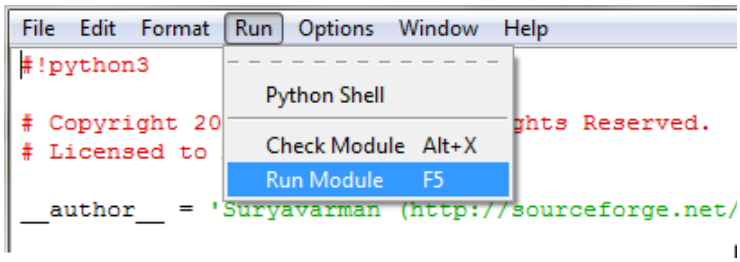


Visionner le rapport via IDLE:

Vous pouvez le visionner via le terminal de IDLE pour se faire ouvrez le script via IDLE.



Pressez F5 ou via le menu Run > Run Module



Attendez la fin de l'exécution pour avoir le rapport ou les erreurs qui ont arrêtées l'exécution du script.

```
Total count issues :
    - errors : 24
    - warnings : 234
======================================================================
Count errors reports : 91
----------------------------------------------------------------------
Error Report : CMake_Compiler_Report_Visual2010_VC100_64_AdressModel.x64_Release zlib
----------------------------------------------------------------------
Error Report : Ide_Report_CodeBlocks_Mingw_32_AdressModel.x86_Debug zlib
----------------------------------------------------------------------
Error Report : Compiler_Report_Visual2010_VC100_64_AdressModel.x64_Debug zlib
----------------------------------------------------------------------
Error Report : CMake_Visual Studio 10_Debug
----------------------------------------------------------------------
Error Report : CMake_Visual Studio 10_Release
----------------------------------------------------------------------
Error Report : Compiler_Report_Visual2015_VC140_32_AdressModel.x86_Debug zlib
----------------------------------------------------------------------
Error Report : Ide_Report_Visual2013_VC120_32_AdressModel.x86_Debug zlib
----------------------------------------------------------------------
Error Report : Ide_Report_Visual2013_VC120_64_AdressModel.x64_Release zlib
----------------------------------------------------------------------
Error Report : Compiler_Report_Visual2012_VC110_32_AdressModel.x86_Release zlib
----------------------------------------------------------------------
Error Report : Ide_Report_Visual2012_VC110_32_AdressModel.x86_Debug zlib
----------------------------------------------------------------------
Error Report : Compiler_Report_Visual2010_VC100_32_AdressModel.x86_Release zlib
----------------------------------------------------------------------
Error Report : CMake_Compiler_Report_Visual2013_VC120_64_AdressModel.x64_Release zlib
----------------------------------------------------------------------
Error Report : CMake_Compiler_Report_Visual2015_VC140_64_AdressModel.x64_Release zlib
----------------------------------------------------------------------
Error Report : CMake_Compiler_Report_Visual2012_VC110_32_AdressModel.x86_Debug zlib
----------------------------------------------------------------------
Error Report : Compiler_Report_Visual2013_VC120_64_AdressModel.x64_Debug zlib
----------------------------------------------------------------------
Error Report : Ide_Report_Visual2008_VC90_32_AdressModel.x86_Release zlib
echo Project : error PRJ0019: A tool returned an error code from "Checking Build System"
echo Project : error PRJ0019: A tool returned an error code from "Building Custom Rule E:/Working/GDeps/trunk/Projects/zlib/zlib/CMakeLists.txt"
echo Project : error PRJ0019: A tool returned an error code from "Building Custom Rule E:/Working/GDeps/trunk/Projects/zlib/zlib/CMakeLists.txt"
echo Project : error PRJ0019: A tool returned an error code from "Building Custom Rule E:/Working/GDeps/trunk/Projects/zlib/zlib/CMakeLists.txt"
echo Project : error PRJ0019: A tool returned an error code from "Building Custom Rule E:/Working/GDeps/trunk/Projects/zlib/zlib/CMakeLists.txt"
echo Project : error PRJ0019: A tool returned an error code from "Building Custom Rule E:/Working/GDeps/trunk/Projects/zlib/zlib/CMakeLists.txt"
----------------------------------------------------------------------
Error Report : CMake_Visual Studio 10 Win64_Debug
----------------------------------------------------------------------
Error Report : Compiler_Report_Visual2013_VC120_32_AdressModel.x86_Debug zlib
----------------------------------------------------------------------
Error Report : Ide_Report_Visual2015_VC140_32_AdressModel.x86_Debug zlib
----------------------------------------------------------------------
Error Report : Ide_Report_Visual2013_VC120_32_AdressModel.x86_Release zlib
```

Le rapport ci-dessus indique qu’il y a 24 erreurs et 234 avertissements.

Toutes les erreurs ne sont pas forcément bloquantes mais de manière générale cela indique que la compilation a échouée pour une ou plusieurs cibles.

Ici on peut voir que le rapport «Ide_Report_Visual2008_VC90_32_AdressModel.x86_Release zlib»

possède 5 fois la même erreur. L’erreur c’est produite lorsque l’IDE visual 2008 a essayé de compiler zlib avec le compilateur VC90 en 32bits et en Release.

Pour vérifier si l’erreur est importante on peux commencé par vérifier si les binaires ont bien été générés.

Dans l’exemple suivant zlib utilise Cmake. GDeps peu alors créer une arborescence de dossier semblable au nom de la cible.

Rendez-vous dans le dossier Visual2008 > VC90_32 > realease_dynamic.
Les binaires ont été générés dans le sous-dossier «Release». La présence des binaires générés à la même date que le lancement du script python laisse supposer que l’erreur n’est pas si grave pour

ceux qui souhaites juste les binaires.