

# DOCUMENTATION

- PAGURE -

Outil d'aide à l'installation de dépendances



Version 1.0



## Table des matières

1 Contexte.....	3
2 Compilateurs et librairies de parallélisation.....	4
3 Pré-requis.....	5
4 Description de PAGURE.....	5
5 Architecture du code.....	6
1. Les groupes.....	7
2. Les variables PAGURE.....	8
3. Les librairies.....	9
4. Les constructeurs.....	12
5. Comportement.....	12
6 Utilisation.....	13
7 FAQ.....	13

# 1 Contexte

L'installation de modèles numériques dans un environnement informatique se relève souvent laborieux à cause de l'évolution permanente des codes informatiques qui brise la compatibilité entre les composants logiciels d'un système informatique. Un modèle numérique comme tous les logiciels informatiques se base sur un ensemble de briques logiciels tel que des compilateurs, des bibliothèques et des logiciels tiers. Régulièrement, ces briques logiciels sont mises à jour par leurs développeurs afin de corriger des bogues ou d'ajouter de nouvelles fonctionnalités. Les logiciels évoluent alors d'une version 1.0 à une version 1.1 par exemple pour des modifications mineures ou à une version 2.0 lorsque les modifications sont multiples ou affectent une grande partie de l'application. La montée d'une version 1.x à 2.x indique généralement une rupture de compatibilité entre les versions 1.x et 2.x. C'est à dire que nous ne pouvons plus utiliser le logiciel en version 2.x de la même manière que la version 1.x., il faut adapter notre utilisation. L'installation d'un modèle numérique nécessite donc une surveillance des versions de chaque dépendance du modèle.

De nos jours, nos systèmes d'exploitation disposent d'outil de gestion de mise à jour afin de minimiser les erreurs liées aux dépendances. Sous Linux, par exemple, il existe des gestionnaires paquets (apt, zypper, yast,...) qui s'occupent de récupérer les bonnes versions des logiciels afin de maintenir une cohérence et donc une stabilité du système. Cependant, les logiciels disponibles dans les gestionnaires de paquets sont des versions emballées pour une utilisation standard alors que nos modèles ont des exigences particulières en terme de performance, de parallélisation, de stockage, d'échange de flux de données, etc. Ainsi, il est obligatoire d'installer manuellement certaines dépendances afin de les configurer selon les exigences du modèle.

C'est ainsi que PAGURE a vu le jour afin d'automatiser l'installation de la chaîne des dépendances des modèles numériques sur des environnements de type PC ou super-calculateur.

## 2 Compilateurs et bibliothèques de parallélisation

A l'heure actuelle, il existe deux grands fournisseurs de compilateurs qui sont le projet GNU et la société Intel. Le projet GNU propose un ensemble de compilateurs appelé GNU Compiler Collection, abrégé en GCC, pour divers langages de programmation comme C, C++, Objective-C, Java, Ada et Fortran. Ces compilateurs sont libres de droit et fréquemment mis à jour par la communauté. La version courante est 7.5 et la dernière version à ce jour est la version 10. La plupart des logiciels ne supportent pas encore la version 10, il est donc fortement recommandé d'utiliser la version 7.5 des compilateurs GNU/GCC.

De l'autre côté, la société Intel développe ses propres compilateurs pour les langages C, C++ et Fortran qui sont optimisés pour les processeurs de la marque. Les processeurs Intel sont très répandus dans le domaine du calcul scientifique puisqu'il équipe la majorité des super-calculateurs dans le monde. Les compilateurs Intel sont payant à l'année et sont mis à jour régulièrement par la société. La version courante est 19.x.

Il est très important de faire la différence entre les compilateurs GNU et les compilateurs Intel puisque sauf indication du développeur, il n'y a aucune garantie qu'un code qui compile avec GNU/GCC compilera avec les compilateurs Intel et inversement. A cause du coût des compilateurs Intel, la plupart des logiciels sont testés avec les compilateurs GNU/GCC.

Un autre point important est celui de la parallélisation. Aujourd'hui, la plupart des modèles numériques repose sur le principe de la parallélisation où le domaine de calcul est divisé en sous-domaine et plusieurs unités de calcul vont résoudre en parallèle chacun des sous-domaine. Ce principe permet une réduction majeure du temps de calcul.

En pratique, il existe deux grandes méthodes qui dépendent de l'architecture informatique employée. Sur les architectures dites à mémoire partagée, on utilise des unités de calcul qui partagent une même mémoire comme typiquement les cœurs au sein d'un processeur. La bibliothèque OpenMP est la bibliothèque de référence pour la parallélisation sur des architectures à mémoire partagée. L'avantage est la rapidité d'exécution puisqu'il n'y a aucun échange de données à faire entre les unités de calcul mais l'inconvénient est le nombre d'unités de calcul disponibles en parallèle. A l'heure actuelle, on peut avoir 24 cœurs au sein d'un processeur dernière génération. Sur les architectures à mémoire distribuée, on utilise des ordinateurs distants avec un échange de données entre les unités de calcul. L'avantage est de pouvoir utiliser un nombre illimité d'unités de calcul en parallèle mais l'inconvénient est le coût des échanges de données entre les unités. La norme MPI est la norme de référence en parallélisation sur les architectures à mémoire distribuée. Cette norme est implémentée par différents projets comme OpenMPI, MPICH ou encore IntelMPI.

Toutes ces bibliothèques de parallélisation sont à compiler avec des compilateurs comme GNU/GCC ou Intel (sauf IntelMPI qui ne fonctionne qu'avec les compilateurs Intel). On peut

donc très bien imaginer avoir deux versions de la librairies OpenMPI, une compilée avec les compilateurs GNU/GCC et l'autre avec les compilateurs Intel.

**En matière de compilation et de parallélisation, le critère a toujours respecté est d'utiliser les mêmes fournisseurs et les mêmes versions des compilateurs et des librairies de parallélisation pour tous les logiciels qui en dépende. Une erreur de fournisseurs ou de versions peut entraîner des comportements anormaux des logiciels dépendants qui sont alors très dure à diagnostiquer.**

### 3 Pré-requis

PAGURE fonctionne sous Linux et macOS. Les pré-requis sont d'avoir installer les compilateurs C/C++ et gfortran dans la version que l'on souhaite utiliser. Sur les super-calculateurs, il suffit généralement de charger le module approprié.

### 4 Description de PAGURE

PAGURE est un code libre de droit développé et maintenu par Fabien Rétif qui permet l'installation de logiciel sur des environnements de type PC ou super-calculateur soit à partir des paquets du système d'exploitation soit en compilant les fichiers sources du logiciel. Il prend la forme d'un script Bash à exécuter sur la machine avec des arguments de configuration. Il se veut simple à utiliser tant que l'on connaît la chaîne des dépendances à installer et surtout facile à déployer dans des environnements à base Linux.

La collection des logiciels installés sur la machine est gérée par l'outil module (<https://modules.readthedocs.io/en/latest/>) qui sera automatiquement installé à la première exécution s'il n'est pas déjà installé. L'outil module permet de mettre à jour un ensemble de variables systèmes (PATH, LD\_LIBRARY\_PATH,...) en fonction des librairies que l'on souhaite utiliser. Il est donc possible d'avoir plusieurs versions du même logiciel sans générer de conflit.

PAGURE est capable de gérer les différents fournisseurs et versions des compilateurs et des librairies de parallélisation. Les fournisseurs et les versions sont à renseigner à l'exécution du script par des arguments.



## 5 Architecture du code

PAGURE est programmé en Bash. Il comprend un script principal nommé « `pagure.sh` » et un ensemble de sous-scripts situé dans le dossier « `include` » (Figure n°1). Chaque sous-script correspond à un groupe qui rassemble la définition des librairies d'un même thème.

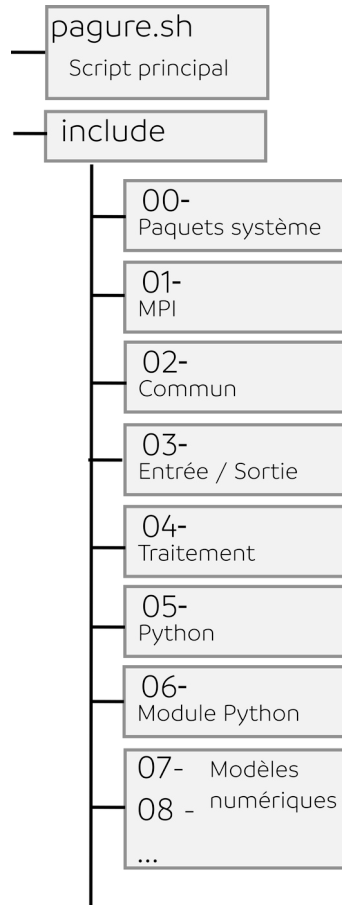


Figure n° 1 : Arborescence des fichiers.

## 1. Les groupes

Un groupe correspond à un ensemble de librairies du même thème. Par exemple, le groupe MPI contient toutes les librairies relatives à la parallélisation en MPI tel que les librairies openMPI ou MPICH. Les groupes sont identifiés par un numéro allant de 0 à 200 (voir variable *maxGroup* dans le fichier « *pagure.sh* »).

### Groupe 00

Le groupe 00 réfère à l'installation des paquets du système d'exploitation nécessaire à l'installation des librairies spécifiées dans les autres groupes. Chaque système d'exploitation ayant son propre gestionnaire de paquet et ses propres noms de paquet, il existe autant de fichiers que de systèmes d'exploitation supportés par PAGURE.

Les fichiers du groupe 00 contiennent des instructions du type « *sudo apt-get install* ».

Note : Dans le cas d'un système d'exploitation de type « cluster », ce groupe ne sera pas exécuté puisqu'il nécessite des droits administrateurs.

### Groupe 01

Le groupe 01 réfère à l'installation de librairies de parallélisation comme OpenMPI ou MPICH. Le fichier du groupe 01 contient toutes les définitions des librairies MPI et de leurs versions.

### Groupe 02

Le groupe 02 réfère à l'installation des librairies communes à l'ensemble des librairies des autres groupes. On y retrouvera la bibliothèque LAPACK ou encore la bibliothèque de compression ZLIB.

### Groupe 03

Le groupe 03 réfère à l'installation des librairies d'entrée/sortie ou de support à des formats de fichier. On y retrouvera la bibliothèque NetCDF ou encore la bibliothèque de support du format de fichier GRID.

### Groupe 04

Le groupe 04 réfère à l'installation des librairies de traitements. On y retrouvera les bibliothèques GDAL, GMT, CDO...

## Groupe 05

Le groupe 05 réfère à l'installation de l'interpréteur Python. On y retrouve toutes les versions de Python installables directement depuis PAGURE.

## Groupe 06

Le groupe 06 réfère à l'installation des librairies Python tel que numpy, scipy...

Attention, certaines librairies Python reposent sur des librairies C/C++ qui auront du être installées préalablement dans les groupes 02,03,04 ou 05. Bien souvent, les dépendances au sein de PAGURE sont décrites par un message d'information au moment de l'installation de la librairie.

## Groupe 07, 08 et plus

Les groupes 07, 08 et plus réfèrent à l'installation des modèles numériques ou des applications du dernier niveau de dépendance.

## 2.Les variables PAGURE

PAGURE contient un ensemble de variables qui sont initialisées au démarrage. Ces variables sont accessibles dans tous les sous-scripts et permettent de configurer dynamiquement les paramètres des librairies. Le tableau n°1 présente les différentes variables à disposition.

Nom de la variable	Description
basedir	Chemin absolu du répertoire d'exécution du script.
prefix	Chemin absolu du répertoire cible de l'installation.
systemOS	Étiquette du système d'exploitation cible. Liste des valeurs possibles : cluster suse mint centos  Par défaut, « cluster ».
compilo	Étiquette du compilateur à utiliser.



	Exemple : « gcc7.5 »
mpilib	Étiquette de la librairie MPI à utiliser. Si l'exécution ne mentionne pas l'utilisation de librairie MPI, l'étiquette vaut « none ».  Exemple : « openmpi110 »
mpi_dep	Module de la librairie MPI correspondant à la valeur de l'étiquette « mpilib ».  Exemple : « openmpi/\$compilo/1.10.7 »
pythonVersion	Version de l'interpréteur Python à utiliser.  Exemple : « 3.7 »
pythonInterpreter	Commande Python de l'interpréteur à utiliser. Il correspond à : python\${pythonVersion}  Exemple : python3.7
moduleDir	Répertoire absolu pour l'installation des fichiers contenant tous les modules de l'outil Module.  Par défaut « \$prefix/Modules/local »
showOldVersion	Booléen (0 ou 1) pour afficher les anciennes versions des librairies.  Par défaut « 0 »
group	Indice du groupe courant.
index	Indice de la bibliothèque courante.

Tableau n° 1 : Variables PAGURE.

### 3. Les librairies

Les librairies sont identifiées par le numéro du groupe auquel elles appartiennent et un numéro propre à chaque occurrence de librairie. Par exemple la librairie OpenMPI en version 1.10.7 sera identifiée par « 11 », le premier chiffre correspond au groupe 01 et le deuxième correspond à l'identifiant de l'occurrence. La librairie OpenMPI en version 3.0.0 sera identifiée par « 12 », le premier chiffre correspond au groupe 01 et le deuxième correspond à l'identifiant de l'occurrence.

Les librairies sont définies par plusieurs paramètres présentés dans le tableau n°2. Les valeurs peuvent parfois contenir des variables PAGURE afin de proposer une gestion dynamique des valeurs de ces paramètres.

Paramètres	Obligatoire	Type	Description
index	X	Entier	Identifiant de la librairie qui définit l'ordre des dépendances au sein du groupe. On placera en indice n°1 la librairie qu'il faut installer en premier.  Exemple : « 4 »
name	X	Chaîne de caractère	Nom de la librairie.  Exemple : « metis »
version	X	Chaîne de caractère	Version de la librairie.  Exemple : « 4.2.3 »
details		Chaîne de caractère	Message d'information pour l'utilisateur.  Exemple : « Requis par la librairie NetCDF »
url	X	Chaîne de caractère	Lien HTTP ou étiquette « localfile » pour télécharger le code source. Si le lien commence par http:// alors le fichier distant sera téléchargé. Si le lien vaut « localfile », alors il sera demandé à l'utilisateur de fournir le chemin absolu vers le fichier attendu par le paramètre « filename ».
filename	X	Chaîne de caractère	Nom du fichier de l'archive contenant le code source. Les formats acceptés sont : *.tar.gz *.tgz *.zip  Exemple : « metis-4.2.3.tar.gz »
dirname	X	Chaîne de caractère	Nom du répertoire résultant de l'extraction de l'archive.  Exemple : « metis-4.2.3 »
patch_01		Chaîne de caractère	Chemin relatif à la variable \$dirname du fichier à corriger.
patch_01_file		Chaîne de caractère	Contenu du correctif à appliquer au fichier \$patch_01  Note : il est important d'échapper tous les guillemets ( <i>double quote</i> ) et tous les dollars de

			variable par \
			Exemple : <code>\\${MPI_LINK_FLAGS}</code>
patch_02		Chaîne de caractère	Chemin relatif à la variable \$dirname du fichier à corriger.
patch_02_file		Chaîne de caractère	Contenu du correctif à appliquer au fichier \$patch_02  Note : il est important d'échapper tous les guillemets ( <i>double quote</i> ) et tous les dollars de variable par \
			Exemple : <code>\\${MPI_LINK_FLAGS}</code>
builder	X	Chaîne de caractère	Étiquette du constructeur à utiliser. Les constructeurs disponibles sont : configure cmake python ou tout autre constructeurs personnalisé.  Exemple : « configure »
dependencies		Chaîne de caractère	Nom des modules séparé par un espace blanc dont la librairie dépend. Exemple : « proj/gcc75/6.3.1 gdal/gcc75/3.0.4 »  Note : Le nom des modules sont généralement écrit dynamiquement à l'aide des variables PAGURE.  Exemple : « proj/\$compilo/6.3.1 gdal/\$compilo/3.0.4 »
dirinstall	X	Chaîne de caractère	Chemin relatif à la variable \$prefix dans lequel sera installé la librairie.  Note : Le chemin est généralement écrit dynamiquement à l'aide des autres variables ou des variables PAGURE.  Exemple : « \${name["\$group\$index"]}/openmpi110/\$compilo/\${version["\$group\$index"]} »
args		Chaîne de caractère	Arguments de compilation.  Exemple : « --with-proj=\$prefix/proj/\$compilo/6.3.1 »
dirmodule		Chaîne de caractère	Chemin relatif à la variable PAGURE « moduleDir » dans lequel sera installé le module.
modulefile		Chaîne de	Contenu du fichier module.

		caractère	Note : il est important d'échapper tous les guillemets ( <i>double quote</i> ) et tous les dollars de variable par \  Exemple : <code>\\"\${MPI_LINK_FLAGS}\"</code>
--	--	-----------	--

Tableau n° 2 : Variables pour définir les librairies.

## 4. Les constructeurs

Pour compiler une librairie il existe différents d'outils qui gère les étapes de compilation.

La méthode traditionnelle passe par l'appel à un outil de configuration (configure) puis à la commande de compilation « make » puis à la commande d'installation « make install ». L'outil Cmake propose une nouvelle manière de réaliser la compilation en passant par un fichier de configuration et l'appel à la commande cmake. Pour certaines librairies, il arrive de devoir saisir un ensemble de commande personnalisé pour effectuer la compilation.

Les constructeurs PAGURE permettent de choisir la méthode de compilation à utiliser pour la librairie. On retrouve dans PAGURE trois constructeurs génériques qui sont :

configure

cmake

python

Lorsqu'il s'agit de passer par une méthode personnalisé, il convient de déclarer une nouvelle étiquette et d'implémenter le nouveau comportement dans le fichier « pagure.sh » comme c'est le cas pour « metis » par exemple.

## 5. Comportement

Le comportement de PAGURE consiste à appeler les groupes séquentiellement en fonction de leur identifiant dans l'ordre croissant et d'appeler les librairies du groupe en fonction de leur identifiant dans l'ordre croissant. Ainsi une librairie avec l'indice n°1 sera installée avant la librairie avec l'indice n°2 pour respecter ainsi l'ordre de dépendances.

Pour chaque librairie, l'utilisateur peut choisir de l'installer ou de l'ignorer pour passer à la librairie suivante en répondant « Yes » ou « No ». Le script se termine une fois que le dernier groupe aura fini de traiter sa dernière librairie. Pendant l'exécution du script, il n'y a pas de retour en arrière possible, les librairies et les groupes défilent séquentiellement en fonction de l'ordre des dépendances préalablement établi par leur indice.

6 Utilisation

7 FAQ