DOCUMENTATION

- PAGURE -

Outil d'aide à l'installation de dépendances

Version 1.0

Fabien Rétif

20/11/2020

Table des matières

1 Contexte 3
2 Compilateurs et librairies de parallélisation 4
3 Pré-requis 5
4 Description de PAGURE 5
5 Utilisation 6
6 Architecture du code 9
    1. Les groupes 10
    2. Les variables PAGURE 11
    3. Les librairies 13
    4. Les constructeurs 15
    5. Les filtres 16
    6. Comportement général 16
7 FAQ 17

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 brique 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 monté 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 librairies 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 unité de calcul vont résoudre en parallèle chacun des sous-domaine. Ce principe permet une réduction majeur du temps de calcul.

En pratique, il existe deux grandes méthodes qui dépendent de l'architecture informatique employée. Sur les architectures dite à mémoire partagé, on utilise des unités de calcul qui partage une même mémoire comme typiquement les cœurs au sein d'un processeur. La librairie OpenMP est la librairie de référence pour la parallélisation sur des architectures à mémoire partagé. 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 librairies 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.

Pour les environnements de type PC, il est recommandé d'avoir les droits administrateurs pour installer les paquets du système d'exploitation comme notamment les compilateurs C/C++ et Fortran.

Pour les environnements de type super-calculateur, il est important de charger les modules appropriés tel que les compilateurs, les librairies MPI et le module use.own avant d'exécuter PAGURE.

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 Utilisation

Pour utiliser PAGURE, il convient d'abord de vérifier les droits d'exécution sur le fichier « pagure.sh ».

ls -l pagure.sh

Pour mettre les droits d'exécution :

chmod +x pagure.sh

Voici la liste des arguments de PAGURE.

Argument	Obligatoire	Type	Description
--prefix	X	Chaîne de caractère	Chemin absolu du répertoire cible pour installer les librairies. Si le répertoire n'existe pas, il sera automatiquement créé. Exemple : «--prefix=/home/user/softs »
--system		Chaîne de caractère	Étiquette du système cible. cluster suse mint centos Par défaut : cluster Exemple : « --system=suse »
--compiler	X	Chaîne de caractère	Étiquette du compilateur à utiliser. La version du compilateur sera automatiquement déduite de la commande de compilation et sera affichée dans les logs de sortie. gnu intel Exemple : «--compiler=gnu »
--mpi		Chaîne de caractère	Étiquette de la librairie MPI à utiliser. Si l'argument n'est as utilisé, PAGURE proposera uniquement les librairies séquentielles. openmpi110 openmpi300 intel2016 intel2017

			intel2018 intel2019 mpich321 mpich332 Exemple : «--mpi=openmpi110 »
--python-version		Chaîne de caractère	Version de l'interpréteur Python. Si la version de interpréteur renseignée n'est pas disponible mais que PAGURE est en mesure de l'installer alors l'installation de l'interpréteur Python dans la version renseignée sera proposé au cours de l'installation des librairies. Exemple : « --python-version=3.7 »
--filter		Chaîne de caractère	Étiquette du filtre à utiliser. Pour connaître la liste des filtres disponibles : ./pagure.sh –list Exemple : « --filter=SWAN »
--module-dir		Chaîne de caractère	Chemin du répertoire cible pour l'installation des modules. Sur les super-calculateurs, il convient d'utiliser le répertoire « privatemodules » de son espace utilisateur. Par défaut : $prefix/Module/local Exemple : « --module-dir=/home/user/privatemodules »
--show-old-version		Booléen	Booléen (0 ou 1) pour afficher les anciennes versions des librairies. Par défaut « 0 »
--force-reinstall		Booléen	Booléen (0 ou 1) pour forcer de réinstaller une librairie. Par défaut « 0 »
--force-download		Booléen	Booléen (0 ou 1) pour forcer le téléchargement des librairies. A utiliser en cas de fichier corrompu lors d'un précédent téléchargement. Par défaut « 0 »
--auto-remove		Booléen	Booléen (0 ou 1) pour supprimer les fichiers téléchargés et les fichiers du code source qui ont servi à la compilation. L'auto-remove peut être mis à 0 en cas de debug. Par défaut « 1 »

Exemple d'utilisation

Usage de PAGURE

./pagure.sh

PAGURE 1.0 - compile your dependencies like a pilot -
PAGURE comes with ABSOLUTELY NO WARRANTY
Author: Fabien RETIF
Usage :
  pagure.sh --list   To list all filters available

  pagure.sh [--prefix=PREFIX] [--system=CLUSTER|SUSE|MINT|CENTOS] [--compiler=GNU|INTEL] [--mpi=openmpi110|openmpi300|intel2016|intel2017|intel2018|intel2019|mpich321|mpich332] [--python-version=X.X] [--filter=NAME_OF_FILTER] [--module-dir=MODULE_DIR] [--show-old-version=0|1] [--force-reinstall=0|1] [--force-download=0|1] [--auto-remove=0|1]

Liste les filtres disponibles

./pagure.sh --list

SWAN
TELEMAC
WW3
DELFT3D
SYMPHONIE

Installation de librairies sur un système d'exploitation OpenSUSE avec GNU/GCC, OpenMPI v1.10.7 et Python 3.7

./pagure.sh --prefix=/home/user/softs --system=suse --compiler=GNU –mpi=openmpi110 --python-version=3.7

Installation de librairies sur un super-calculateur avec Intel, IntelMPI 2017 et Python 3.7

./pagure.sh --prefix=/home/user/softs --system=cluster --compiler=INTEL --mpi=intel2017 –python-version=3.7 --module-dir=/home/user/privatemodules

6 Architecture du code

PAGURE est programmé en Bash. Il comprend un script principal nommé « pagure.sh » et un ensemble de fichier situé dans le dossier « include » (Figure n°1). Les fichiers contenus dans le dossier « include » sont chargés par le fichier « pagure.sh » au cours de son exécution.

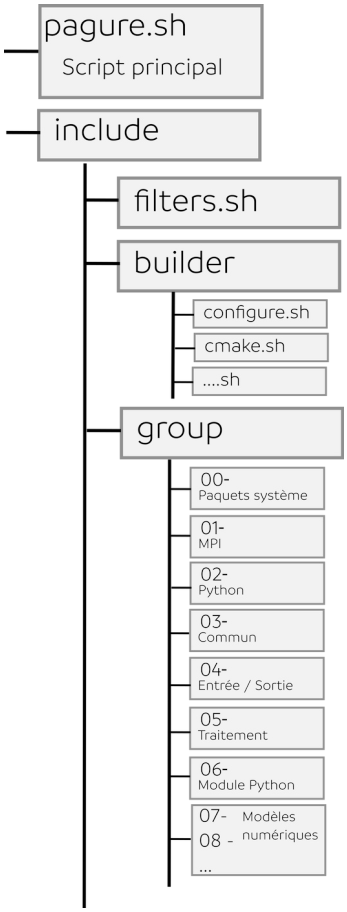

Figure n° 1 : Arborescence des fichiers.

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 identifié 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 fichier que de système d'exploitation supporté 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 de l'interpréteur Python. On y retrouve toutes les versions de Python installables directement par PAGURE.

Groupe 03

Le groupe 03 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 04

Le groupe 04 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 05

Le groupe 05 réfère à l'installation des librairies de traitements. On y retrouvera les bibliothèque GDAL, GMT, CDO...

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 »
forceDownload	Booléen (0 ou 1) pour forcer le téléchargement des librairies. Utiliser en cas de fichier corrompu lors d'un précédent téléchargement. Par défaut « 0 »
forceReinstall	Booléen (0 ou 1) pour forcer de réinstaller une librairie. Par défaut « 0 »
autoRemove	Booléen (0 ou 1) pour supprimer les fichiers ayant servis à la compilation. Par défaut « 1 »
group	Indice du groupe courant.
index	Indice de la bibliothèque courante.

Tableau n° 1 : Variables PAGURE.

3.Les librairies

Les librairies sont identifiés 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 définit 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 télécharger. Si le lien vaut « localfile », alors il sera demandé à l'utilisateur de fournir le chemin absolue 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 (*double quote)* et tous les dollars de variable par \ Exemple : \"\${MPI_LINK_FLAGS}\"
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 $patch_02 Note : il est important d'échapper tous les guillemets (*double quote)* et tous les dollars de variable par \ Exemple : \"\${MPI_LINK_FLAGS}\"
builder	X	Chaîne de caractère	Étiquette du constructeur à utiliser. Les constructeurs disponibles sont : configure cmake python ou tout autre constructeurs personnalisés. Exemple : « configure »
dependencies		Chaîne de caractère	Nom des modules séparé par un espace blanc dont la librairie dépend. Exemple : « proj/gcc75/6.3.1 gdal/gcc75/3.0.4 » Note : Le nom des modules sont généralement écrit dynamiquement à l'aide des variables PAGURE. Exemple : « proj/$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 caractère	Contenu du fichier module. Note : il est important d'échapper tous les guillemets (*double quote)* et tous les dollars de variable par \ Exemple : \"\${MPI_LINK_FLAGS}\"

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 un nouveau fichier dans le dossier « builder » portant le même nom que l'étiquette.

5.Les filtres

Les filtres permettent de filtrer les librairies installables par PAGURE en fonction des besoins. Ils représentent en quelques sortes une chaîne de dépendances en pré-sélectionnant des librairies et en spécifiant leurs ordres d'installation. Les filtres sont composés d'une étiquette et de l'ensemble des identifiants des librairies qui le composent.

Exemple : filters["SYMPHONIE"]="11,26,31,33,34,35"

Le filtre SYMPHONIE permet d'installer uniquement les librairies 11,26,31,33,34,35 en commençant par la librairie 11 puis 26 puis 31…

Note : Le groupe de librairie n°1 concerne les librairies MPI. Dans l'exemple de SYMPHONIE, il est choisit d'installer la librairie 11 qui correspond à la librairie OpenMPI en version 1.10.7. Si vous ne spécifiez pas la même version de librairie dans l'argument –mpi de PAGURE, vous obtiendrez un message d'erreur vous invitant à utiliser la même version de librairie MPI que celle utilisé dans le filtre.

6.Comportement général

Le comportement de PAGURE consiste à vérifier tous les arguments spécifiés à l'exécution puis à appeler les groupes séquentiellement en fonction de leur identifiant dans l'ordre croissant et de proposer l'installation des librairies du groupe en fonction de leur identifiant dans l'ordre croissant.

Pour chaque librairie, l'utilisateur peut choisir d'installer la librairie en tapant « yes » ou de l'ignorer pour passer à la suivante en tapant « no ». L'installation d'un librairie consiste à compiler la librairie grâce aux constructeurs (variable $builder), à l'installer dans un répertoire d'installation (variable $dirinstall) puis de créer le module qui permettra de l'utiliser (variable $dirmodule et $modulefile). Le script se termine une fois que le dernier groupe a fini de traiter la dernière librairie. Pendant l'exécution du script, il n'y a pas de retour en arrière possible, les librairies et les groupes défilent séquentiellement.

Dans le cas d'utilisation d'un filtre, le principe reste identique sauf les librairies défilent en fonction de l'ordre spécifié par le filtre.

Un fichier de log nommé « pagure.log » permet de conserver le déroulement de l'installation. Attention, ce fichier de log est systématiquement écraser à chaque exécution.

7 FAQ

- Je veux mettre à jour une librairie ?

Pour mettre à jour une librairie, il suffit d'éditer la description de la librairie dans le fichier du groupe dans laquelle se trouve la librairie. Si vous souhaitez conserver la version actuelle de la librairie, vous pouvez dupliquer la description de la librairie et attribuer un nouvel identifiant à cette nouvelle version. Par exemple : je veux mettre une autre version de Ruby. Il faut alors éditer le fichier « include/group/03-common.sh » puis trouver la partie correspond à Ruby :

```
#ruby 2.7.2
index=6
name["$group$index"]=ruby
version["$group$index"]=2.7.2
details["$group$index"]=""
url["$group$index"]=https://cache.ruby-lang.org/pub/ruby/2.7/ruby-2.7.2.tar.gz
filename["$group$index"]=ruby-2.7.2.tar.gz
dirname["$group$index"]=ruby-2.7.2
builder["$group$index"]="configure"
#dependencies["$group$index"]=""
dirinstall["$group$index"]="${name["$group$index"]}/$compilo/${version["$group$index"]}"
args["$group$index"]=""
dirmodule["$group$index"]="${name["$group$index"]}/$compilo"
modulefile["$group$index"]="#%Module1.0
proc ModulesHelp { } {
global dotversion

puts stderr \"\tRuby ${version["$group$index"]}\"
}

module-whatis \"Ruby ${version["$group$index"]}\"
prepend-path PATH $prefix/${dirinstall["$group$index"]}/bin
prepend-path LD_LIBRARY_PATH $prefix/${dirinstall["$group$index"]}/lib
prepend-path LIBRARY_PATH $prefix/${dirinstall["$group$index"]}/lib
prepend-path C_INCLUDE_PATH  $prefix/${dirinstall["$group$index"]}/include
prepend-path INCLUDE $prefix/${dirinstall["$group$index"]}/include
prepend-path CPATH $prefix/${dirinstall["$group$index"]}/include
prepend-path MANPATH $prefix/${dirinstall["$group$index"]}/share/man
"
```

On peut alors dupliquer cette partie et la modifier comme l'exemple ci-dessous :

Les changements sont surlignés en vert.

```
#ruby 2.6.6
index=7
name["$group$index"]=ruby
version["$group$index"]=2.6.6
details["$group$index"]=""
url["$group$index"]=https://cache.ruby-lang.org/pub/ruby/2.6/ruby-2.6.6.tar.gz
filename["$group$index"]=ruby-2.6.6.tar.gz
dirname["$group$index"]=ruby-2.6.6
builder["$group$index"]="configure"
#dependencies["$group$index"]=""
dirinstall["$group$index"]="${name["$group$index"]}/$compilo/${version["$group$index"]}"
args["$group$index"]=""
dirmodule["$group$index"]="${name["$group$index"]}/$compilo"
modulefile["$group$index"]="#%Module1.0
proc ModulesHelp { } {
global dotversion

puts stderr \"\tRuby ${version["$group$index"]}\"
}

module-whatis \"Ruby ${version["$group$index"]}\"
prepend-path PATH $prefix/${dirinstall["$group$index"]}/bin
prepend-path LD_LIBRARY_PATH $prefix/${dirinstall["$group$index"]}/lib
prepend-path LIBRARY_PATH $prefix/${dirinstall["$group$index"]}/lib
prepend-path C_INCLUDE_PATH  $prefix/${dirinstall["$group$index"]}/include
prepend-path INCLUDE $prefix/${dirinstall["$group$index"]}/include
prepend-path CPATH $prefix/${dirinstall["$group$index"]}/include
prepend-path MANPATH $prefix/${dirinstall["$group$index"]}/share/man
"
```

Note : Si le processus de compilation n'a pas évolué entre la version 2.7.2 et la version 2.6.6, il n'y a pas d'autre modifications à faire. En revanche si le processus a évolué et que la compilation ne fonctionne pas, il faut modifier les arguments de compilation ($args) ou le constructeur ($builder) jusqu'à trouver la bonne configuration. Tout ceci est à adapter au cas par cas. Il arrive parfois de devoir installer une dépendance supplémentaire. Dans ce cas, reportez-vous à la question « Je veux ajouter une nouvelle librairie » dans la FAQ.

- Je veux ajouter une nouvelle librairie ?

Pour ajouter une nouvelle librairie, il faut éditer le fichier du groupe dans lequel on souhaite ajouter la nouvelle librairie. L'idéal est de dupliquer une librairie existante qui se rapproche le plus de notre nouvelle librairie. Les points de ressemblances commencent par le constructeur ($builder). Il suffit ensuite d'attribuer un nouvel identifiant ($index) puis il faut éditer le nom, la version, l'url de téléchargement,...

Il arrive parfois de devoir spécifier des dépendances vers d'autre librairie. Dans ce cas, il suffit d'ajouter le module à charger dans la variables « dependencies["$group$index"] ». Pour plus d'informations, reportez-vous à la section 6.3.

Note : Le principe est de commencer par un constructeur standard (configure, cmake, python) puis d'opérer les changements nécessaire pour arriver à la compilation.

- Je veux créer un nouveau constructeur ?

Pour créer un nouveau constructeur, il suffit de créer un nouveau fichier dans le dossier « include/builder ». Ce fichier doit porter le même nom que l'étiquette du constructeur et doit porter l'extension .sh. Par exemple pour le constructeur suivant :

```
builder["$group$index"]="mon_constructeur"
```

PAGURE s'attend à trouver un fichier « mon_constructeur.sh » dans le dossier include/builder.

- Je veux ajouter un nouveau filtre ?

Pour ajouter un nouveau filtre, il suffit d'éditer le fichier « include/filters.sh » et de rajouter un nouveau filtre contenant les identifiants des libraires comme ceci :

```
Filters["MON_FILTRE"]="13,34,36,41,43,44,45,1101"
```

Pour plus d'informations, reportez-vous à la section 6.5.