perlmodlib

Table des matières

1 NAME/NOM

perlmodlib - Pour construire de nouveaux modules et trouver les existants

2 DESCRIPTION

3 LA LIBRAIRIE DE MODULES PERL

Un certain nombre de modules sont inclus dans la distribution de Perl. Ils sont décrit plus loin, à la fin du *.pm*. Vous pouvez alors découvrir les fichiers dans le répertoire des librairies qui se terminent par autre chose que *.pl* ou *.ph*. Ce sont d'anciennes librairies fournies pour que les anciens programmes les utilisant continuent de fonctionner. Les fichiers *.pl* ont tous été converti en modules standards, et les fichiers *.ph* fabriqués par **h2ph** sont probablement terminé comme extension des modules fabriqués par **h2xs**. (Certains *.ph* peuvent être déjà disponible par le module POSIX. Le fichier **pl2pm** de la distribution peut vous aider dans votre conversion, mais il s'agit que d'un mécanisme du processus et par conséquent loin d'être une preuve infaillible.

3.1 Pragmatic Modules

Ils fonctionnent comme les pragmas par le fait qu'ils ont tendance à affecter la compilation de votre programme, et ainsi fonctionne habituellement bien que lorsqu'ils sont utilisés avec `use`, ou `no`. La plupart de ceux-ci ont une portée locale, donc un BLOCK interne peut outrepasser n'importe lequel en faisant:

```
no integer;
no strict 'refs';
```

ce qui dure jusqu'à la fin de ce BLOC.

À la différence des pragmas qui effectuent `$^H` variable de conseils, les déclarations `use vars` et `use subs` ne sont pas limitées au bloc. Elles vous permettent de prédéclarer des variables ou des sous-programmes dans un *fichier* plutôt que juste pour un bloc. De telles déclarations sont pertinentes pour le fichier entier dans lequel elles ont été déclarées. Vous ne pouvez pas les annuler avec `no vars` ou `no subs`.

Les pragmas suivants sont définis (et ont leur propre documentation).

use autouse MODULE => qw(sub1 sub2 sub3)
: Reporte `require MODULE` jusqu'à ce que quelqu'un appelle un des sous-programmes indiqués (qui doivent être exportés par MODULE). Ce pragma devrait être utilisé avec prudence, et seulement si nécessaire.

blib
: manipule @INC au moment de la compilation pour utiliser la version déinstallée par MakeMaker's d'une paquetage.

diagnostics
: force le diagnostic explicite des messages d'alerte

integer
: Calcul arithmétique en integer au lieu de double

less
: demande moins de quelque chose au compilateur

lib
: manipule @INC au moment de la compilation

locale
: utilisez ou ignorez l'état actuel de locale pour des opérations internes (voir *perllocale*)

ops
: limitez les opcodes nommés quand vous compilez ou exécutez du code Perl

overload
: surcharge les opérations basic de Perl

re
: modifie le comportement des expression rationnelles

sigtrap
: permet la capture de signaux simples

strict
: restreint les constructions non sûres

subs
: prédéclare les noms de fonctions

vmsish
: adopte certains comportements spécifiques à VMS

vars
: prédéclare les noms de variables globales

3.2 Standard Modules

En standard, on s'attend à ce que des modules empaquetés se comportent tous d'une façon bien définie en ce qui concerne la pollution de namespace parce qu'ils utilisent le module Exporter. Voir leur propre documentation pour des détails.

AnyDBM_File
: fournissez le cadre de travail pour de multiples DBMs

AutoLoader
: charge les fonctions seulement à la demande

AutoSplit
: scinde un paquetage pour le chargement automatique

Benchmark
: benchmark pour tester les temps d'exécution de votre code

CPAN
: interface pour le Comprehensive Perl Archive Network

CPAN::FirstTime
: crée un fichier de configuration CPAN

CPAN::Nox
: exécutez CPAN tout en évitant les extensions compilées

Carp
: prévient les erreurs (de la perspective de l'appelant)

Class::Struct
: déclare des types de données similaires au struct

Config
: accède aux informations de configuration de Perl

Cwd
: donne le nom du répertoire de travail courant

DB_File
: accède à Berkeley DB

Devel::SelfStubber
: génère les stubs pour un module qui se charge lui-même (SelfLoading)

DirHandle
: fournit les méthodes de objets pour les descripteurs de répertoires

DynaLoader
: charge dynamiquement les librairies C dans le code Perl

English
: utilise les noms anglais (ou awk) jolies pour des variables laides de ponctuation

Env
: importe les variables d'environnement

Exporter
: implémente la méthode d'import par défaut des modules

ExtUtils::Embed
: utilitaires pour encapsuler du Perl dans les applications C/C++

ExtUtils::Install
: installez des fichiers d'ici à là

ExtUtils::Liblist
: détermine les librairies à utiliser et comment les utiliser

ExtUtils::MM_OS2
: méthode pour écraser le comportement d'Unix dans ExtUtils::MakeMaker

ExtUtils::MM_Unix
: méthodes utilisées par ExtUtils::MakeMaker

ExtUtils::MM_VMS
méthode pour écraser le comportement d'Unix dans ExtUtils::MakeMaker

ExtUtils::MakeMaker
crée une extension de Makefile

ExtUtils::Manifest
utilitaires pour écrire et vérifier un fichier MANIFEST

ExtUtils::Mkbootstrap
fabrique un fichier d'amorçage à l'usage de DynaLoader

ExtUtils::Mksymlists
écrivez les fichiers d'options d'éditeur de liens pour les extensions dynamiques

ExtUtils::testlib
ajoute les répertoire blib/* à @INC

Fatal
Transforme les erreurs dans les fonctions internes ou dans les fonctions de Perl fatales

Fcntl
Charge les définitions de C Fcntl.h

File::Basename
sépare un nom de répertoire en parties

File::CheckTree
effectue plusieurs contrôles sur des tests de fichiers dans un arbre

File::Compare
compare des fichiers ou des descripteurs de fichiers

File::Copy
copie des fichiers ou des descripteurs de fichiers

File::Find
traverse un arbre de fichiers

File::Path
crée ou supprime une série de répertoires

File::stat
by-name interface to Perl's builtin stat() functions

FileCache
maintenez plus de fichiers ouverts que les autorisations du système le permettent

FileHandle
fournit les méthodes des objets pour les descripteurs de fichiers

FindBin
localise le répertoire original du script Perl

GDBM_File
accède à la librairie gdbm

Getopt::Long
traitement étendu des options de ligne de commande

Getopt::Std
commutateurs (switches) de processus de caractères simples avec groupe de commutateurs (switch clustering)

I18N::Collate
comparez des données scalaires de 8 bits selon la configuration locale actuelle

IO
charge divers modules d'E/S

IO::File
fournit les méthodes d'objets pour les descripteurs de fichiers

IO::Handle
fournit les méthodes des objets pour les opérations d'E/S

IO::Pipe
fournit les méthodes des objets pour les tubes (pipe)

IO::Seekable
fournit les méthodes pour les objets d'E/S

IO::Select
interface OO pour l'appel système sélectionné

IO::Socket
interface des objets pour les communications par socket

IPC::Open2
ouvre un process pour à la fois lire et écrire

IPC::Open3
ouvre un process pour lire, écrire et capturer les erreurs

Math::BigFloat
module pour les nombres à virgule de longueur arbitraire

Math::BigInt
module pour les entiers de taille arbitraire

Math::Complex
module pour les nombres complexes et les fonctions mathématiques associées

Math::Trig
interface simple pour Math::Complex pour ceux qui ont besoin des fonctions trigonométriques seulement pour les nombres réels

NDBM_File
lie l'accès aux fichier ndbm

Net::Ping
Bonjour, il y a quelqu'un ?

Net::hostent
interface par nom pour les fonctions internes de Perl gethost*()

Net::netent
interface par nom pour les fonctions internes de Perl getnet*()

Net::protoent
interface par nom pour les fonctions internes de Perl getproto*()

Net::servent
interface par nom pour les fonctions internes de Perl getserv*()

Opcode
désactive les opcodes nommés pendant la compilation ou l'exécution de code Perl

Pod::Text
converti des données POD en texte ASCII formaté

POSIX
interface pour le standard IEEE 1003.1

SDBM_File
lie l'accès au fichiers sdbms

Safe
compile et exécute le code dans des compartiments restreints

Search::Dict
`cherche une clef dans le fichier du dictionnaire`

SelectSaver
sauve et restaure le descripteur de fichier sélectionné

SelfLoader
charge les fonctions seulement à la demande

Shell
lance des commandes shell de façon transparente dans Perl

Socket
charge la définition et les manipulateurs de structure de socket.h

Symbol
manipule les symboles Perl et leurs noms

Sys::Hostname
essaye toutes les méthodes conventionnelles pour obtenir un nom de machine

Sys::Syslog
interface pour les appels à la commande Unix syslog(3)

Term::Cap
interface pour termcap

Term::Complete
module de complétion de mots

Term::ReadLine
interface vers des paquetages `readline` variés

Test::Harness
Lance des scripts de tests standards de Perl avec des statistiques

Text::Abbrev
crée une table d'abréviation d'une liste

Text::ParseWords
parse du texte dans un tableau de marques

Text::Soundex
implémentation de l'algorithme Soundex Algorithm comme décrit par Knuth

Text::Tabs
agrandit ou diminue des tableaux avec la fonction Unix expand(1) et unexpand(1)

Text::Wrap
formate les lignes pour former des paragraphes simples

Tie::Hash
définitions de base des classes pour les tableaux associatifs liés (tied hashes)

Tie::RefHash
définitions de base des classes pour définitions de base des classes pour les tableaux associatifs liés (tied hashes) avec comme références les clés avec des références comme clés

Tie::Scalar
définitions de base des classes pour scalaires liés (tied)

Tie::SubstrHash
tableau associatif avec taille-de-tableau-fixe, longueur-de-clé-fixe

Time::Local
calcul efficace de l'heure locale et GMT

Time::gmtime
interface par nom pour les fonctions internes de Perl gmtime()

Time::localtime
interface par nom pour les fonctions internes de Perl localtime()

Time::tm
objet interne utilisé par Time::gmtime et Time::localtime

UNIVERSAL
classe de base pour TOUTES les classes (références bénites (blessed))

User::grent
interface par nom pour les fonctions internes de Perl getgr*()

User::pwent
interface par nom pour les fonctions internes de Perl getpw*()

Pour trouver *tous* les modules installés sur votre système, incluant ceux sans documentation ou en dehors de la release standard, faites ceci:

```
% find `perl -e 'print "@INC"'` -name '*.pm' -print
```

Ils doivent avoir leur propre documentation installée et accessible via votre commande système man(1). Si cela échoue, essayer le programme *perldoc*.

3.3 Extension de Modules

Les extensions de modules sont écrits en C (ou un mixte de Perl et de C) et peuvent être liées (linked) statiquement ou en général sont chargées dynamiquement dans Perl si et quand vous en avez besoin. Les extensions de modules supportées comprennent les Socket, Fcntl, et les modules POSIX.

La plupart des extensions C de modules populaires n'arrivent pas tout prêt (ou du moins, pas complètement) due à leur taille, leur volatibilitée, ou simplement par manque de temps de tests adéquats et de configuration autour des multitudes de plates-formes où Perl est beta-testé. Vous êtes encouragé à les regarder dans archie(1L), la FAQ Perl ou Meta-FAQ, les pages WWW, et même avec leur auteurs avant de poster des questions pour leurs conditions et dispositions actuelles.

4 CPAN

CPAN signifie le Comprehensive Perl Archive Network. Il s'agit d'une réplique globale de tous les matériaux Perl connus, incluant des centaines de modules non chargés. Voici les catégories principales de ces modules:

- Les Extensions de langage et la Documentation des outils
- Support au Développement
- Interface pour le Système d'exploitation
- Réseau, contrôle de modems and Processus d'intercommunication
- Types de données et utilitaires de type de données
- Interfaces base de données
- User Interfaces
- Interfaces pour / Emulations d'autres langages de programmation
- Nom de fichiers, Système de fichiers et verrous de fichiers (voir aussi Descripteur de fichiers)
- Traitements de chaînes de caractères, traitements de textes de langage, analyse, et recherche
- Option, argument, paramètre, et traitement de fichier de configuration
- Internationalisation et Locale
- Authentification, Sécurité, and Encryption
- World Wide Web, HTML, HTTP, CGI, MIME
- Serveur and utilitaires de Démons
- Archivage et Compression
- Images, Manipulation de Pixmap et Bitmap, Dessins et Graphiques
- Mail et News Usenet
- Utilitaires de Contrôle de Flux (callbacks et exceptions etc)
- Utilitaires pour les descripteurs de fichier ou pour les chaînes d'entrée/sortie
- Modules variés

Les sites officiels CPAN en date de cette écriture sont les suivants. Vous devriez essayer de choisir un près de chez vous:

- Afrique

Afrique du Sud	ftp://ftp.is.co.za/programming/perl/CPAN/

- Asie

Hong Kong	ftp://ftp.hkstar.com/pub/CPAN/
Japon	ftp://ftp.jaist.ac.jp/pub/lang/perl/CPAN/
	ftp://ftp.lab.kdd.co.jp/lang/perl/CPAN/
Corée du Sud	ftp://ftp.nuri.net/pub/CPAN/
Taiwan	ftp://dongpo.math.ncu.edu.tw/perl/CPAN/
	ftp://ftp.wownet.net/pub2/PERL/

- Australie

Australie	ftp://ftp.netinfo.com.au/pub/perl/CPAN/
Nouvelle Zélande	ftp://ftp.tekotago.ac.nz/pub/perl/CPAN/

- Europe

Autriche	ftp://ftp.tuwien.ac.at/pub/languages/perl/CPAN/
Belgique	ftp://ftp.kulnet.kuleuven.ac.be/pub/mirror/CPAN/
Rép. Tchèque	ftp://sunsite.mff.cuni.cz/Languages/Perl/CPAN/
Danemark	ftp://sunsite.auc.dk/pub/languages/perl/CPAN/
Finlande	ftp://ftp.funet.fi/pub/languages/perl/CPAN/
France	ftp://ftp.ibp.fr/pub/perl/CPAN/
	ftp://ftp.pasteur.fr/pub/computing/unix/perl/CPAN/
Allemagne	ftp://ftp.gmd.de/packages/CPAN/
	ftp://ftp.leo.org/pub/comp/programming/languages/perl/CPAN/

```
                    ftp://ftp.mpi-sb.mpg.de/pub/perl/CPAN/
                    ftp://ftp.rz.ruhr-uni-bochum.de/pub/CPAN/
                    ftp://ftp.uni-erlangen.de/pub/source/Perl/CPAN/
                    ftp://ftp.uni-hamburg.de/pub/soft/lang/perl/CPAN/
    Grèce           ftp://ftp.ntua.gr/pub/lang/perl/
    Hongrie         ftp://ftp.kfki.hu/pub/packages/perl/CPAN/
    Italie          ftp://cis.utovrm.it/CPAN/
    the Netherlands ftp://ftp.cs.ruu.nl/pub/PERL/CPAN/
                    ftp://ftp.EU.net/packages/cpan/
    Norvège         ftp://ftp.uit.no/pub/languages/perl/cpan/
    Pologne         ftp://ftp.pk.edu.pl/pub/lang/perl/CPAN/
                    ftp://sunsite.icm.edu.pl/pub/CPAN/
    Portugal        ftp://ftp.ci.uminho.pt/pub/lang/perl/
                    ftp://ftp.telepac.pt/pub/CPAN/
    Russie          ftp://ftp.sai.msu.su/pub/lang/perl/CPAN/
    Slovénie        ftp://ftp.arnes.si/software/perl/CPAN/
    Espagne         ftp://ftp.etse.urv.es/pub/mirror/perl/
                    ftp://ftp.rediris.es/mirror/CPAN/
    Suède           ftp://ftp.sunet.se/pub/lang/perl/CPAN/
    RU              ftp://ftp.demon.co.uk/pub/mirrors/perl/CPAN/
                    ftp://sunsite.doc.ic.ac.uk/packages/CPAN/
                    ftp://unix.hensa.ac.uk/mirrors/perl-CPAN/
```

– Amérique du Nord

```
    Ontario          ftp://ftp.utilis.com/public/CPAN/
                     ftp://enterprise.ic.gc.ca/pub/perl/CPAN/
    Manitoba         ftp://theory.uwinnipeg.ca/pub/CPAN/
    Californie       ftp://ftp.digital.com/pub/plan/perl/CPAN/
                     ftp://ftp.cdrom.com/pub/perl/CPAN/
    Colorado         ftp://ftp.cs.colorado.edu/pub/perl/CPAN/
    Floride          ftp://ftp.cis.ufl.edu/pub/perl/CPAN/
    Illinois         ftp://uiarchive.uiuc.edu/pub/lang/perl/CPAN/
    Massachusetts    ftp://ftp.iguide.com/pub/mirrors/packages/perl/CPAN/
    New York         ftp://ftp.rge.com/pub/languages/perl/
    Caroline du Nord ftp://ftp.duke.edu/pub/perl/
    Oklahoma         ftp://ftp.ou.edu/mirrors/CPAN/
    Oregon           http://www.perl.org/CPAN/
                     ftp://ftp.orst.edu/pub/packages/CPAN/
    Pennsylvanie     ftp://ftp.epix.net/pub/languages/perl/
    Texas            ftp://ftp.sedl.org/pub/mirrors/CPAN/
                     ftp://ftp.metronet.com/pub/perl/
```

– Amérique du Sud

```
    Chili            ftp://sunsite.dcc.uchile.cl/pub/Lang/perl/CPAN/
```

Pour une liste à jour des sites CPAN, voir *http://www.perl.com/perl/CPAN* ou *ftp://ftp.perl.com/perl/*.

5 Modules: Création, Utilisation, et Abus

(la section suivante est empruntée directement des fichiers des modules de Tim Buncees, disponible depuis votre site CPAN plus proche.)

Le Perl implémente une classe en utilisant un module, mais la présence d'un module n'implique pas la présence d'une classe. Un module est juste un espace de nom (namespace). Une classe est un module qui fournit les sous-programmes qui peuvent être utilisés comme méthodes. Une méthode est juste un sous-programme qui prévoit que son premier argument est le nom d'un module (pour des méthodes "statiques"), ou une référence à quelque chose (pour des méthodes "virtuelles").

Un module est un fichier qui (par convention) fournit une classe du même nom (sans le .pm), plus une méthode d'importation dans cette classe qui peut s'appeler pour chercher les symboles exportés. Ce module peut appliquer certaines de ses méthodes en chargeant les objets dynamiques en C ou en C++, mais cela devrait être totalement transparent à l'utilisateur du module. De même, le module pourrait installer une fonction AUTOLOAD dans des définitions de sous-programme à la demande, mais c'est également transparent. Seulement un fichier *.pm* est nécessaire pour exister. Voir *perlsub*, *perltoot*, et *AutoLoader* pour des détails au sujet du mécanisme de AUTOLOAD.

5.1 Directives pour la création de modules

Des modules similaires existent-ils déjà sous une certaine forme?

Si oui essayez, s'il vous plaît de réutiliser les modules existants en entier ou en héritant des dispositifs utiles dans une nouvelle classe. Si ce n'est pratique, voyez avec les auteurs de ce module pour travailler à étendre ou à mettre en valeur les fonctionnalité des modules existants. Un exemple parfait est la pléthore de modules dans perl4 pour traiter des options de ligne de commande.

Si vous écrivez un module pour étendre un ensemble de modules déjà existant, coordonnez-vous s'il vous plaît avec l'auteur du module. Cela aide si vous suivez la même convention de nom et d'interaction de module que l'auteur initial.

Essayez de concevoir le nouveau module pour être facile étendre et réutiliser.

Utilisez les références sacrifiées (blessed). Utilisez deux arguments pour sacrifier le nom de classe donné comme premier paramètre du constructeur, ex. :

```
sub new {
      my $class = shift;
      return bless {}, $class;
}
```

ou même ceci si vous voudriez qu'il soit utilisé comme méthode statique ou virtuelle :

```
sub new {
      my $self  = shift;
      my $class = ref($self) || $self;
      return bless {}, $class;
}
```

Un passage de tableau comme références permet ainsi plus de paramètres pouvant être ajoutés plus tard (et également plus rapide). Convertissez les fonctions en méthodes le cas échéant. Coupez les grandes méthodes en les plus petites plus flexibles. Héritez des méthodes d'autres modules si approprié.

Evitez les essais nommés de classe comme: `die "Invalid" unless ref $ref eq 'FOO'`. D'une façon générale vous pouvez effacer la partie "`eq 'FOO'`" sans que cela pose problème. Laissez les objets s'occuper d'eux! D'une façon générale, évitez les noms codés en dur de classe aussi loin que possible.

Evitez `$r->Class::func()` en utilisant `@ISA=qw(... Class ...)` et `$r->func()` fonctionnera (voir *perlbot* pour plus de détails).

Utilisez autosplit pour les fonctions peu utilisées ou nouvellement ajoutées pour que cela ne soit pas un fardeau pour les programmes qui ne les utilisent pas. Ajoutez les fonctions de test au module après le __END__ en utilisant AutoSplit ou en disant:

```
eval join('',<main::DATA>) || die $@ unless caller();
```

Votre module passe-t-il le test 'de la sous classe vide? Si vous dites " `@SUBCLASS::ISA = qw(YOURCLASS);` " vos applications devraient pouvoir utiliser la SOUS-CLASSE exactement de la même façon que YOURCLASS. Par exemple, est-ce que votre application fonctionne toujours si vous changez: `$obj = new VOTRECLASSE;` en: `$obj = new SOUS-CLASSE;`

Evitez de maintenir n'importe quelle information d'état dans vos modules. Cela le rend difficile d'utilisation pour de multiple autres modules. Gardez à l'esprit l'information d'état de subsistance dans les objets.

Essayez toujours d'utiliser **-w**. Essayez d'utiliser `use strict;` (ou `use strict qw(...);`). Souvenez-vous que vous pouvez ajouter `no strict qw(...);` aux blocs individuels code qui nécessite moins de contraintes. Utilisez toujours **-w**. Utilisez toujours **-w**! Suivez les directives de perlstyle(1).

Quelques directives simples de modèle

Le manuel de perlstyle fourni avec Perl a beaucoup de points utiles.

La façon de coder est une question de goût personnel. Beaucoup de gens font évoluer leur style sur plusieurs années pendant qu'elles apprennent ce qui les aide à écrire et mettre à jour un bon code. Voici un ensemble de suggestions assorties qui semblent être largement répandues par les réalisateurs expérimentés:

Employez les underscore pour séparer des mots. Il est généralement plus facile de lire $un_nom_de_variable que $UnNomDeVariable, particulièrement pour les personnes de langue maternelle autre que l'anglais. C'est une règle simple qui fonctionne également avec NOM_DE_VARIABLE.

Les noms de Package/Module sont une exception à cette règle. Le Perl réserve officieusement des noms minuscules de module pour des modules de 'pragma' comme les nombre entier et strict. D'autres modules normalement commencent par une majuscule et utilisent ensuite les cas mélangés sans des souligné (besoin d'être court et portable).

Vous pouvez trouvez pratique d'utiliser la case des lettres pour indiquer la portée ou la nature d'une variable. Par exemple:

```
  $TOUT_EN_MAJUSCULES : seulement les constantes (prenez garde aux
désaccords avec les variables de Perl)
  $Seulement_Quelques_Majuscules  portée le temps d'un paquetage,
variables globales/statiques
  $aucune_majuscules   portée d'une variable dans une fonction avec
my() ou local()
```

Les noms de fonction et de méthode semblent mieux fonctionner quand tout est en minuscule. ex., `$obj->as_string()`.

Vous pouvez employer un underscore devant le nom des variables pour indiquer qu'une variable ou une fonction ne devrait pas être utilisée en dehors du module qui l'a définie.

Choisir quoi exporter.

N'exportez pas les noms de méthode!

N'exportez pas toute autre chose par défaut sans bonne raison!

Les exportations polluent le namespace de l'utilisateur du module. Si vous devez exporter quelque chose utiliser @EXPORT_OK de préférence à @EXPORT et éviter des noms communs ou courts pour réduire le risque de désaccords sur les noms.

D'une façon générale quelque chose non exporté est encore accessible de l'extérieur du module en utilisant la syntaxe de ModuleName::item_name (ou `$blessed_ref->method`). Par convention vous pouvez employer un underscore précédent le nom de variable pour indiquer officieusement qu'il s'agit de variables 'internes' et pas pour l'usage public.

(il est possible d'obtenir des fonctions privées en disant: `my $subref = sub { ... }; &$subref;`. Mais il n'y a aucune façon de les appeler directement comme méthode, parce qu'une méthode doit avoir un nom dans la table de symbole.)

En règle générale, si le module essaye d'être orienté objet alors n'exportez rien. S'il c'est juste une collection de fonctions alors @EXPORT_OK quelque chose mais l'utilisation de @EXPORT est à faire avec prudence.

Choisir un nom pour le module.

Ce nom devrait être descriptif, précis, et aussi complet que possible. Evitez n'importe quel risque d'ambiguïté. Essayez toujours d'utiliser deux ou plus de mots. D'une façon générale le nom devrait refléter ce qui est spécial au sujet de ce que le module fait plutôt que de la façon dont il le fait. Veuillez employer les noms emboîtés de module pour grouper officieusement ou pour classer un module par catégorie. Il devrait y a une très bonne raison pour un module de ne pas avoir un nom emboîté. Les noms de module devraient commencer par une majuscule.

Ayant 57 modules tous appelé Sort ne rendra pas la vie facile pour n'importe qui (avoir cependant 23 appelés Sort::Quick est seulement marginalement meilleur :-) . Imaginez quelqu'un essayant d'installer votre module à côté de beaucoup d'autres. Si vous avez un doute demandez des suggestions dans comp.lang.perl.misc.

Si vous développez une suite de modules/classes liés, habituellement on utilise les classes emboîtées avec un préfixe commun car ceci évitera des désaccords de namespace. Par exemple: Xyz::Control, Xyz::View, Xyz::Model etc... Utilisez les modules dans cette liste comme guide nommant.

Si vous ajoutez un nouveau module à un ensemble, suivez les normes de l'auteur initial pour nommer les modules et l'interface des méthodes dans ces modules.

Pour être portable, chaque composant d'un nom de module devrait être limité à 11 caractères. S'il pourrait être employé sur MS-DOS alors vous devez vous assurer que chacun fait moins de 8 caractères. Les modules emboîtés facilitent ceci.

Est-ce que vous avez bien fait ?

Comment savez-vous que vous avez pris les bonnes décisions? Avez-vous sélectionné une conception d'interface qui posera des problèmes plus tard? Avez-vous sélectionné le nom le plus approprié? Avez-vous des questions?

La meilleure façon de savoir est de prendre beaucoup de suggestions utiles, et de demander à quelqu'un qui sait. Comp.lang.perl.misc est lu par toutes les personnes qui développent des modules et c'est le meilleur endroit pour demander.

Tout que vous devez faire est de poster un court sommaire du module, de son but et de ses interfaces. Quelques lignes sur chacune des méthodes principales est probablement suffisant. (si vous signalez le module entier il pourrait être ignoré par les personnes occupées - généralement les mêmes personnes dont vous aimeriez avoir l'avis !)

Ne vous inquiétez dans votre post si vous ne pouvez pas dire quand le module sera prêt - juste dites-le dans le message. Il pourrait être intéressant d'inviter d'autres pour vous aider, ils peuvent le terminer pour vous!

README et autres fichiers additionnels.

Il est bien connu que les développeurs de logiciels documentent habituellement entièrement le logiciel qu'ils écrivent. Si cependant le monde est dans le besoin pressant de votre logiciel et qu'il n'y a pas assez de temps pour écrire toute la documentation s'il vous plaît au moins fournissez un fichier README qui contient:

– Une description du module/paquetage/extension etc.
– Une note sur le copyright - voir plus loin.
– Prérequis - ce dont vous pouvez avoir besoin.
– Comment le construire - les changements éventuels dans Makefile.PL etc.
– Comment l'installer.
– Les changements récents de cette version, spécialement les incompatibilités.
– Changements / améliorations que vous prévoyez de faire dans le futur.

Si le fichier README semble devenir trop large, séparer le en plusieurs sections dans des fichiers séparés: INSTALL, Copying,(A copier) ToDo(A faire) etc.

Ajouter une note sur le copyright.

Comment vous décidez du type de licence est une décision personnelle. Le mécanisme général est d'insérer votre Copyright et de faire une déclaration aux autres qu'ils peuvent copier/utiliser/modifier votre travail.

Perl, par exemple, est fournie avec deux types de licence: GNU GPL et The Artistic Licence (voir les fichiers README, Copying, et Artistic). Larry a de bonnes raisons pour ne pas utiliser que GNU GPL.

Ma recommandation personnelle, en dehors du respect pour Larry, est que Perl, et la communauté Perl au sens large est simplement défini comme tel:

```
Copyright (c) 1995 Your Name. All rights reserved.
This program is free software; you can redistribute it and/or
modify it under the same terms as Perl itself.
```

Ce texte devrait au moins apparaître dans le fichier README. Vous pouvez également souhaiter l'inclure dans un fichier Copying et dans vos fichiers sources. Rappelez-vous d'inclure les autres mots en plus de copyright.

Donnez au module un nombre de version/issue/release.

Pour être entièrement compatible avec les modules Exporter et MakeMaker vous devriez enregistrer le numéro de version de votre module dans une variable non-my appelée $VERSION. Ceci devrait être un nombre à virgule flottante avec au moins deux chiffres après la décimale (c.-à-d., centième, par exemple, `$VERSION = " 0,01 "`). n'utilisez pas une version du modèle "1.3.2". Voir Exporter.pm dans Perl5.001m ou plus pour des détails.

Il peut être pratique pour ajouter une fonction ou méthode de rechercher le nombre. Utilisez le nombre dans les annonces et les noms de fichier d'archives quand vous faites une version d'un module (ModuleName-1.02.tar.Z). Voir perldoc ExtUtils::MakeMaker.pm pour des détails.

Comment construire et distribuer un module.

C'est une bonne idée de poster une annonce de la disponibilité de votre module (ou du module lui-même si il est petit) dans le groupe de discussion comp.lang.perl.announce. Ceci assurera au moins une très large distribution en dehors de la distribution Perl.

Si possible vous pouvez placer le module dans un des archives importantes ftp et inclure les détails de son emplacement dans votre annonce.

Quelques notes au sujet des archives ftp: Veuillez utiliser un nom de fichier descriptif qui inclut le numéro de version. La plupart des répertoires entrants ne seront pas lisibles/listables, c.-à-d., vous ne pourrez pas voir votre fichier après l'avoir téléchargé. Rappelez-vous d'envoyer votre message d'avis par mail aussitôt que possible après avoir téléchargé votre module, autrement votre fichier peut obtenir effacé automatiquement. Accordez du temps pour que le fichier soit traité et/ou contrôlez que le fichier a été traité avant d'annoncer son emplacement.

FTP Archives for Perl Modules:

Suivre les instructions et les liens sur

```
http://franz.ww.tu-berlin.de/modulelist
```

ou posez le dans un de ces sites:

```
ftp://franz.ww.tu-berlin.de/incoming
ftp://ftp.cis.ufl.edu/incoming
```

et prévenez *<upload@franz.ww.tu-berlin.de>*.

En utilisant l'interface WWW vous pouvez demander au serveur de téléchargement de refléter vos modules de votre ftp ou de votre site Web dans votre propre répertoire sur CPAN!

Rappelez-vous s'il vous plaît de m'envoyer une entrée mise à jour pour la liste de modules!

Faites attention quand vous faites une nouvelle version d'un module.

Tâchez toujours de rester compatible avec les versions précédentes. Autrement essayez d'ajouter un mécanisme pour retourner à l'ancien comportement si les gens comptent là-dessus. Documentez les changements incompatibles.

5.2 Directives pour convertir des bibliothèques Perl 4 en modules

Il n'y a pas de prérequis pour convertir quel module que ce soit.

Si il n'est pas rompu, ne le fixez pas! Les bibliothèques Perl 4 devraient continuer à fonctionner sans problèmes. Vous pouvez avoir à faire quelques changements mineurs (comme changer les variables @ qui ne sont pas des tableaux en chaînes doublement cotées) mais il n'y a aucun besoin de convertir un fichier pl en module juste pour cela.

Prendre en considération les implications.

Toutes les applications de Perl qui se servent du script devront être changées (légèrement) si le script est converti en module. Cela vaut la peine si vous projetiez de faire d'autres changements en même temps.

Tirez le meilleur de l'occasion.

Si vous allez convertir un script en module vous pouvez profiter de l'occasion pour remodeler l'interface. Les 'directives pour la création de module' inclues plus haut plusieurs issues que vous devriez considérer.

L'utilitaire pl2pm est votre point de départ.

Cet utilitaire lira les fichiers *.pl (donnés comme paramètres) et écrira les fichiers *.pm correspondants. L'utilitaire pl2pm fait ce qui suit :

- Ajoute les lignes standards de prologue de module
- Convertissez les spécificateurs de package ' en ::
- Convertissez les die(...) en croak(...)
- Quelques autres changement mineurs

Le processus mécanique de pl2pm n'est pas une preuve garantie. Le code converti aura besoin de contrôles soigneux, particulièrement pour tous les rapports de module. N'effacez pas le fichier initial pl jusqu'à ce que le nouveau pm ne fonctionne!

5.3 Directives pour réutiliser le code d'application

- Des applications complètes sont rarement complètement sous la forme de bibliothèques/modules Perl.
- Beaucoup d'applications contiennent du code Perl qui pourrait être réutilisé.
- Aidez à sauver le monde! Partagez votre code sous une forme qui est facile à réutiliser.
- Débloquer le code réutilisable dans un ou plusieurs modules séparés.
- Saisissez l'occasion de reconsidérer et remodeler les interfaces.
- Dans certains cas 'l'application' peut alors être réduite à un petit fragment de code construits sur les modules réutilisables. Dans ce cas l'application pourrait être appelé comme :

```
% perl -e 'use Module::Name; method(@ARGV)' ...
```

ou

```
% perl -mModule::Name ...    (perl5.002 ou plus récent)
```

6 NOTE

Le Perl n'impose pas de parties privées et publiques de ses modules comme vous avez pu voir dans d'autres langages comme C++, ADA, ou Modula-17. Le Perl n'a pas d'infatuation (satisfaction excessive et ridicule que l'on a de soi N.D.T) avec l'intimité imposée. Il préférerait que vous êtes restiez hors de sa salle à manger parce que vous n'avez pas été invités, pas parce qu'il a un fusil de chasse.

Le module et son utilisateur ont un contrat, dont une partie est un droit commun, et une partie qui est "écrite". Une partie du contrat de droit commun est qu'un module ne pollue aucun namespace qu'on ne lui ai pas demandé. Le contrat écrit pour le module (cad la documentation) peut prendre d'autres dispositions. Mais vous savez quand vous utilisez <use RedefineTheWorld>, vous redéfinissez le monde et voulez bien en accepter les conséquences.

7 TRADUCTION

7.1 Version

Cette traduction française correspond à la version anglaise distribuée avec perl 5.005_02. Pour en savoir plus concernant ces traductions, consultez http://perl.enstimac.fr/.

7.2 Traducteur

Yves Maniette <yves@giga.sct.ub.es>

7.3 Relecture

Personne pour l'instant.

8 À propos de ce document

Ce document est la traduction française du document orginal distribué avec perl. Vous pouvez retrouver l'ensemble de la documentation française Perl (éventuellement mise à jour) en consultant l'URL <http://perl.enstimac.fr/>.

Ce document PDF a été produit Paul Gaborit. Si vous utilisez la version PDF de cette documentation (ou une version papier issue de la version PDF) pour tout autre usage qu'un usage personnel, je vous serai reconnaissant de m'en informer par un petit message <mailto:Paul.Gaborit@enstimac.fr>.

Si vous avez des remarques concernant ce document, en premier lieu, contactez la traducteur (vous devriez trouver son adresse électronique dans la rubrique TRADUCTION) et expliquez-lui gentiment vos remarques ou critiques. Il devrait normalement vous répondre et prendre en compte votre avis. En l'absence de réponse, vous pouvez éventuellement me contacter.

Vous pouvez aussi participer à l'effort de traduction de la documentation Perl. Toutes les bonnes volontés sont les bienvenues. Vous devriez trouver tous les renseignements nécessaires en consultant l'URL ci-dessus.