perlfaq3

Table des matières

1 NAME/NOM

perlfaq3 - Outils de programmation

2 DESCRIPTION

Cette section de la FAQ répond à des questions relatives aux outils de programmation et à l'aide de programmation.

2.1 Comment fais-je pour... ?

Avez-vous déjà été voir le CPAN (voir *perlfaq2*) ? Il y a des chances pour que quelqu'un ait déjà écrit un module susceptible de résoudre votre problème. Avez-vous déjà lu les pages man appropriées ? Voilà un bref sommaire :

```
Bases                   perldata, perlvar, perlsyn, perlop, perlsub
Exécution               perlrun, perldebug
Fonctions               perlfunc
Objets                  perlref, perlmod, perlobj, perltie
Structure de données    perlref, perllol, perldsc
Modules                 perlmod, perlmodlib, perlsub
Regexp                  perlre, perlfunc, perlop, perllocale
Évoluer vers perl5      perltrap, perl
Lier au langage C       perlxstut, perlxs, perlcall, perlguts, perlembed
Divers                  http://www.cpan.org/misc/olddoc/FMTEYEWTK.tgz
                        (ce n'est pas une page man, mais très utile)
```

Un sommaire rudimentaire des pages de manuel de Perl existantes se trouve dans *perltoc*.

2.2 Comment utiliser Perl de façon interactive ?

L'approche typique consiste à utiliser le débogueur Perl, décrit dans *perldebug*, avec un programme "vide", comme cela :

```
perl -de 42
```

Maintenant, tapez plutôt un code Perl valide, et il sera immédiatement évalué (exécuté). Vous pouvez également examiner la table des symboles, voir l'évolution de la pile, vérifier les valeurs des variables, fixer des points d'arrêt, et faire d'autres opérations typiquement disponibles dans les débogueur symboliques.

2.3 Existe-t-il un shell Perl ?

Le psh (Perl sh) en est à sa version 1.8. C'est un shell qui combine la nature interactive de shell Unix avec la puissance de Perl. Son but est d'être un shell complet utilisant la syntaxe et les fonctionnalités de Perl pour les structures de contrôles et d'autres choses. Vous pouvez récupérer psh sur http://sourceforge.net/projects/psh/.

Zoidberg est un projet similaire qui fournit un shell développé en perl, configuré en perl et utilisé en perl. Il peut être utilisé comme shell de login et comme shell de développement. Vous pouvez le trouver sur http://zoidberg.sf.net ou sur votre miroir CPAN.

Le module Shell.pm (distribué avec Perl) fait que Perl tente l'exécution des commandes qui ne font pas partie du langage Perl en tant que commandes shell. perlsh de la distribution source est simpliste et inintéressant, mais correspondra peut-être à ce que vous recherchez.

2.4 Comment puis-je connaître les modules installés sur mon système ?

Vous pouvez utiliser le module ExtUtils::Installed pour voir toutes les distributions installées. Il peut prendre un peu de temps pour produire sa magie. Les bibliothèques standards qui viennent avec Perl sont regroupées sous le nom "Perl" (vous pouvez les voir via Module::CoreList).

```
use ExtUtils::Installed;

my $inst    = ExtUtils::Installed->new();
my @modules = $inst->modules();
```

Si vous souhaitez une liste de tous les noms de fichiers des modules Perl, vous pouvez utiliser File::Find::Rule.

```
use File::Find::Rule;

my @files = File::Find::Rule->file()->name( '*.pm' )->in( @INC );
```

Si vous ne disposez pas de ce module, vous pouvez obtenir le même résultat via File::Find qui fait partie de la distribution standard.

```
use File::Find;
my @files;
find(
  sub {
    push @files, $File::Find::name
      if -f $File::Find::name && /\.pm$/
  },

  @INC
 );

print join "\n", @files;
```

Si vous voulez juste vérifier rapidement qu'un module est disponible, vous pouvez chercher sa documentation. Si vous pouvez lire la documentation d'un module, il y a de grandes chances qu'il soit installé. Si vous ne le pouvez pas, c'est peut-être que le module n'en propose pas (dans de très rares cas).

```
prompt% perldoc Nom::Module
```

Vous pouvez aussi essayer d'inclure ce module dans un script en une ligne pour voir si perl le trouve.

```
prompt% perldoc -MNom::Module -e1
```

2.5 Comment déboguer mes programmes Perl ?

Avez-vous essayé `use warnings` ou utilisé `-w` ? Cela permet d'afficher des avertissements (des warnings) pour détecter les pratiques douteuses.

Avez-vous utilisé `use strict` ? Cela vous empêche d'utiliser des références symboliques, vous oblige à prédéclarer les sous-programmes que vous appelez comme simple mot, et (probablement le plus important) vous oblige à déclarer vos variables avec `my`, `our` ou `use vars`.

Avez-vous vérifié les résultats de chacune des commandes système ? Le système d'exploitation (et donc Perl) vous indique si elles ont fonctionné ou pas, et la raison de l'échec éventuel.

```
open(FH, "> /etc/cantwrite")
  or die "Couldn't write to /etc/cantwrite: $!\n";
```

Avez-vous lu *perltrap* ? C'est plein de trucs et astuces pour les programmeurs Perl débutants ou initiés. Il y a même des sections pour ceux d'entre vous qui viennent des langages *awk* et *C*.

Avez-vous essayé le débogueur Perl, décrit dans *perldebug* ? Vous pouvez exécuter votre programme et voir ce qu'il fait, pas à pas et ainsi comprendre pourquoi ce qu'il fait n'est pas conforme à ce qu'il devrait faire.

2.6 Comment mesurer les performances de mes programmes Perl ?

Vous devriez utiliser le module Devel::DProf dans les distributions standard récentes (ou de CPAN), ainsi que Benchmark.pm de la distribution standard. Benchmark vous permet de mesurer le temps d'éxécution de portions spécifiques de votre code alors que Devel::DProf vous donne des détails sur les endroits où le code consomme du temps.

Voici un exemple d'utilisation de Benchmark :

```
use Benchmark;

@junk = `cat /etc/motd`;
$count = 10_000;

timethese($count, {
          'map' => sub { my @a = @junk;
                         map { s/a/b/ } @a;
                         return @a },
          'for' => sub { my @a = @junk;
                         local $_;
                         for (@a) { s/a/b/ };
                         return @a },
         });
```

Voici l'affichage généré (sur une machine particulière–vos résultats dépendront de votre matériel, du système d'exploitation, et de la charge de travail de votre machine) :

```
Benchmark: timing 10000 iterations of for, map...
       for:  4 secs ( 3.97 usr  0.01 sys =  3.98 cpu)
       map:  6 secs ( 4.97 usr  0.00 sys =  4.97 cpu)
```

Soyez conscient qu'un bon benchmark est très difficile à écrire. Il ne teste que les données que vous lui passez, et ne prouve vraiment que peu de choses sur les différences de complexités entre divers algorithmes.

2.7 Comment faire une liste croisée des appels de mon programme Perl ?

Le module B::Xref peut être utilisé pour générer une liste croisée des appels pour les programmes Perl.

```
perl -MO=Xref[,OPTIONS] scriptname.plx
```

2.8 Existe-t-il un outil de mise en page de code Perl ?

Perltidy est un script Perl qui indente et reformate les scripts Perl pour les rendre plus lisibles en essayant de respecter les règles de *perlstyle*. Si vous écrivez des scripts Perl ou passez beaucoup de temps à en lire, vous l'apprécierez sûrement. Il est disponible sur http://perltidy.sourceforge.net.

Bien sûr, si vous respectez les recommandations de *perlstyle*, vous ne devriez pas avoir besoin de reformater. L'habitude de formater votre code au fur et à mesure que vous l'écrivez vous évitera bien des erreurs. Votre éditeur devrait vous aider à le faire. Dans emacs, le perl-mode et son successeur, le cperl-mode, peuvent vous être d'une grande aide pour quasiment tout le code, et même d'autres éditeurs moins programmables peuvent vous fournir une assistance significative. Tom Christiansen et de nombreux utilisateurs de vi ne jure que par les réglages suivants sous vi et ses clones :

```
set ai sw=4
map! ^O {^M}^[O^T
```

Placez cela dans votre fichier *.exrc* (en remplaçant les accents circonflexes par des caractères de contrôle) et c'est bon. En mode insertion, ^T est pour l'indentation, ^D pour la suppression de l'indentation, et ^O pour l'indentation d'un bloc. Un exemple plus complet, avec des commentaires, peut être trouvé sur http://www.cpan.org/authors/id/TOMC/scripts/toms.exrc.gz.

Le programme a2ps sur http://www-inf.enst.fr/%7Edemaille/a2ps/black+white.ps.gz fait beaucoup pour imprimer des sorties de documents joliment imprimées. Voyez aussi enscript sur http://people.ssh.fi/mtr/genscript/.

2.9 Existe-t-il un ctags pour Perl ?

(contribution de brain d foy)

Exuberent ctags reconnaît Perl : http://ctags.sourceforge.net/.

Vous pouvez aussi essayer pltags : http://www.mscha.com/pltags.zip.

2.10 Existe-t-il un environnement de développement intégré (IDE) ou un éditeur Perl sous Windows ?

Les programmes Perl sont juste du texte donc n'importe quel éditeur peut convenir.

Si vous êtes sur Unix, vous disposez déjà d'un IDE : Unix lui-même. La philosophie Unix se traduit par un ensemble de petits outils qui réalisent chacun une seule tâche mais bien. C'est comme une boite à outils.

Si vous souhaitez absolument un IDE, consultez la liste suivante (par ordre alphabétique et non par ordre de préférence) :

Eclipse

http://e-p-i-c.sf.net/

Le projet d'intégration de Perl dans Eclipse propose l'edition et le débogage avec Eclipse.

Enginsite

http://www.enginsite.com/

Perl Editor par EngInSite est un environnement de développement complètement intégré (IDE) pour créer, tester et déboguer des scripts Perl. Il tourne sur Windows 9x/NT/2000/XP ou plus.

Komodo

http://www.ActiveState.com/Products/Komodo/

L'IDE d'ActiveState multi-plateformes (en octobre 2004, cela concernait Windows, Linux et Solaris), multi-langage IDE connaît Perl et inclut un débogueur d'expressions rationnelles et du débogage distant.

Open Perl IDE

http://open-perl-ide.sourceforge.net/

Open Perl IDE est un environnement de développement intégré permettant l'écriture et le débogage de scripts Perl avec la distribution ActivePerl d'ActiveState sous Windows 95/98/NT/2000.

OptiPerl

http://www.optiperl.com/

OptiPerl est un IDE Windows permettant la simulation de l'environnement CGI et incluant un débogueur et un éditeur avec décoration syntaxique.

PerlBuilder

http://www.solutionsoft.com/perl.htm

PerlBuidler est un environnement de développement intégré pour Windows qui permet les développement Perl.

visiPerl+

http://helpconsulting.net/visiperl/

De Help Consulting, pour Windows.

Visual Perl

http://www.activestate.com/Products/Visual_Perl/

Visual Perl est le pug-in Visual Studio.NET d'ActiveState.

Zeus

http://www.zeusedit.com/lookmain.html

Zeus pour Window est un autre éditeur/IDE multi-langages pour Win32 qui vient avec le support Perl.

Pour les éditeurs : si vous êtes sous Unix vous disposez certainement de vi ou d'un de ses clones et peut-être aussi d'emacs. Vous n'avez donc rien à télécharger. Dans emacs, le mode cperl-mode (M-x cperl-mode) vous fournira un mode d'édition Perl parmi les meilleurs de tous les éditeurs.

Si vous utilisez Windows, vous pouvez utiliser n'importe quel éditeur qui permet d'éditer du texte brut, tel que NotePad, WordPad ou le Bloc-Notes. Les traitemens de texte comme Microsoft Word ou WordPerfect ne fonctionnent pas bien car ils insèrent plein d'informations cachées (ceci étant, la plupart proposent aussi l'enregistrement au format "Texte seul"). Vous pouvez aussi télécharger des éditeurs de textes conçus spécialement pour la programmation comme Textpad (http://www.textpad.com) et UltraEdit (http://www.ultraedit.com) entre autres.

Si vous utilisez MacOS, les mêmes conseils s'appliquent. MacPerl (pour l'envrionnement Classic) vient avec un éditeur simple. Les éditeurs externes les plus populaires sont BBEdit (http://www.bbedit.com/) or Alpha (http://www.his.com/~jguyer/Alpha/Alpha8.html). Les utilisateurs de MacOS X peuvent aussi utiliser les éditeurs Unix. Neil Bowers (l'homme derrière Geekcruises) propose une liste des éditeurs connaissant Perl sur Mac (http://www.neilbowers.org/macperleditors.html).

GNU Emacs

http://www.gnu.org/software/emacs/windows/ntemacs.html

MicroEMACS

http://www.microemacs.de/

XEmacs

http://www.xemacs.org/Download/index.html

Jed

http://space.mit.edu/~davis/jed/

ou les clones de vi tels :

Elvis

ftp://ftp.cs.pdx.edu/pub/elvis/ http://www.fh-wedel.de/elvis/

Vile

http://dickey.his.com/vile/vile.html

Vim

http://www.vim.org/

Pour les amoureux de vi en général, sous Windows ou autres :

http://www.thomer.com/thomer/vi/vi.html

nvi (http://www.bostic.com/vi/, disponible sur CPAN dans src/misc) est encore un autre clone de vi, malheureusement sans version Windows, qui peut être intéressant sur les plateformes Unix car, d'une part, ce n'est pas à proprement parler un clone de vi mais bien le vrai vi ou plutôt son évolution et, d'autre part, parce qu'il peut utiliser Perl comme langage de programmation interne. nvi n'est pas le seul dans ce cas : vim et vile offrent cette possibilité de Perl intégré.

Les programmes suivants sont des éditeurs Win32 multi-langages qui supportent Perl :

Codewright

http://www.borland.com/codewright/

MultiEdit

http://www.MultiEdit.com/

SlickEdit

http://www.slickedit.com/

Il existe aussi un petit éditeur de texte gadget écrit en Perl et qui est distribué dans le module Tk sur CPAN. ptkdb (http://world.std.com/~aep/ptkdb/) est débogueur Perl/Tk qui agit un peu comme une sorte d'environnement de développement. Perl Composer (http://perlcomposer.sourceforge.net/) est un IDE pour la création d'IHM (Interface Homme Machine) en Perl/Tk.

En plus d'un édtieur/IDE, vous aurez sans doute besoin d'un shell plus puissant en environnement Win32. Voici quelques possibilités :

Bash

fourni comme paquetage Cygwin (http://sources.redhat.com/cygwin/).

Ksh

depuis la boite à outils MKS (http://www.mks.com/), ou le Bourne shell de l'environnement U/WIN (http://www.research.att.com/sw/tools/uwin/)

Tcsh

ftp://ftp.astron.com/pub/tcsh/, voir aussi http://www.primate.wisc.edu/software/csh-tcsh-book/

Zsh

ftp://ftp.blarg.net/users/amol/zsh/ , voir aussi http://www.zsh.org/

MKS et U/WIN sont commerciaux (U/WIN est gratuit dans un cadre éducation ou recherche) alors que Cygwin repose sur la licence Publique GNU (mais ça ne compte pas pour une utilisation aevc Perl). Cygwin, MKS et U/WIN proposent chacun, en plus de leurs shells respectifs, un ensemble complet d'outils standard Unix.

Si vous transférez des fichiers textes entre Unix et Windows en utilisant FTP, assurez-vous de les faire en mode ASCII afin que les caractères de fin de ligne soient correctement convertis.

Sur MacOS, l'application MacPerl propose un éditeur de texte simple limité à 32k qui peut se comporter comme un IDE rudimentaire. À l'opposé, l'outil MPW Perl peut utiliser le shell MPW lui-même comme un éditeur (sans la limite des 32k).

Affrus
: est un véritable environnement de développement en Perl avec un débogueur complet (http://www.latenightsw.com).

Alpha
: est un éditeur écrit et extensible en Tcl qui supporte de nombreux langages de programmation ou à base de balises tels que Perl et HTML (http://www.his.com/~jguyer/Alpha/Alpha8.html).

BBEdit et BBEdit Lite
: sont des édtiteurs de texte pour Mac OS qui ont un mode reconnaissant le Perl (http://web.barebones.com/).

Pepper et Pe sont des édtiteurs de texte gérant les langages de programmation et développés respectivement pour Mac OS X et BeOS (http://www.hekkelman.com/).

2.11 Où puis-je trouver des macros pour Perl sous vi ?

Pour une version complète du fichier de configuration de Tom Christiansen pour vi, voyez http://www.cpan.org/authors/Tom_Christiansen/scripts/toms.exrc.gz, il s'agit du fichier standard pour les émulateurs vi. Cela fonctionne mieux avec nvi, la dernière version de vi de Berkeley, qui peut éventuellement être compilée avec un interpréteur Perl inclut – voir http://www.cpan.org/src/misc/.

2.12 Où puis-je trouver le mode perl pour emacs ?

Depuis la version 19 patchlevel 22 de Emacs, perl-mode.el et l'aide pour le débogueur Perl sont inclus. Vous devriez l'avoir dans la distribution standard version 19 d'Emacs.

Dans le répertoire du code source de Perl, vous trouverez un répertoire nommé "emacs", qui contient un mode d'édition perl qui colore les instructions Perl, fournit une aide contextuelle, et autres choses sympathiques.

Notez que le mode perl de Emacs changera les lignes du genre `"main'foo"` (apostrophe), et modifiera les tabulations et surlignages. Vous utilisez probablement de toute façon `"main::foo"` dans votre nouveau code Perl, donc ceci n'est pas bien grave.

2.13 Comment utiliser des 'curses' avec Perl ?

Le module Curses du CPAN fournit une interface objet chargeable dynamiquement pour la librairie "curses". Une petite démo se trouve dans le répertoire http://www.cpan.org/authors/Tom_Christiansen/scripts/rep.gz ; Ce programme répète régulièrement une commande et rafraîchit l'écran comme il faut, rendant ainsi **rep ps axu** similaire à **top**.

2.14 Comment puis-je utiliser X ou Tk avec Perl ?

Les modules Tk forment une interface pour les outils Tk, entièrement orientée objet et basée sur Perl qui vous évite d'utiliser Tcl pour avoir accès à Tk. Sx est une interface pour l'ensemble Athena Widget. Les deux sont disponibles sur CPAN. Voyez le répertoire http://www.cpan.org/modules/by-category/08_User_Interfaces/.

D'inestimables aides pour la programmation en Perl/Tk sont la FAQ Perl/Tk sur http://phaseit.net/claird/comp.lang.perl.tk/ptkFAQ.html, le guide de référence Perl/Tk disponible sur http://www.cpan.org/authors/Stephen_O_Lidie/ et les pages de manuel en ligne sur http://www-users.cs.umn.edu/%7Eamundson/perl/perltk/toc.html.

2.15 Comment rendre mes programmes Perl plus rapides ?

La meilleure façon pour y parvenir est d'utiliser un meilleur algorithme. Cela peut souvent faire une énorme différence. Le livre de Jon Bentley *Programming Pearls* (sans faute de frappe !) donne aussi de bonnes astuces d'optimisation. Quelques conseils pour l'amélioration de la vitesse : utilisez benchmark pour être certain que vous optimisez la bonne partie de votre code, cherchez de meilleurs algorithmes plutôt que des micro-optimisations pour votre code actuel, et si rien ne s'arrange, considérez qu'il vous faut juste acheter un matériel plus rapide. Si vous ne l'avez pas déjà fait, vous deviez probablement lire la réponse à la question posée plus haut "Comment mesurer les performances de mes programmes Perl ?".

Une approche différente est d'utiliser Autoload pour le code peu utilisé. Voyez les modules AutoSplit et AutoLoader dans la distribution standard pour ce faire. Vous pouvez aussi localiser le goulot d'étranglement et penser à écrire cette partie en C, de la même façon que nous avons l'habitude d'écrire les parties lentes de C en assembleur. Au lieu de réécrire en C, vous pouvez aussi utiliser des modules ayant leurs sections critiques déjà écrites en C (par exemple, le module PDL du CPAN).

Si vous liez votre exécutable perl à une librairie partagée *libc.so*, vous pouvez souvent gagner entre 10 % et 25 % en performance en le liant plutôt avec une librairie statique libc.a. Cela fera un exécutable plus important, mais vos programmes (et programmeurs) Perl vous en remercieront. Voyez le fichier *INSTALL* dans le code source de la distribution pour plus d'informations.

Le programme undump était une ancienne tentative visant à améliorer la vitesse des programmes Perl en les sauvegardant sous forme déjà compilée sur le disque. Ce n'est plus une option viable, puisque cela ne fonctionnait que sur peu d'architectures, et que ce n'était pas une bonne solution de toute façon.

2.16 Comment faire pour que mes programmes Perl occupent moins de mémoire ?

Quand Perl a des échanges internes trop longs, il préfère imputer cela à un problème de mémoire. Les scalaires en Perl utilisent plus de mémoire que les chaînes de caractères en C, les tableaux en utilisent plus aussi, et les tables de hachage moins. Bien qu'il y ait encore beaucoup à faire, les versions récentes répondent à ces problèmes. Par exemple, dans la version 5.004, les clés dupliquées des tables de hachage sont partagées par toutes les tables de hachage les utilisant, ne nécessitant pas de ré-allocation.

Dans certains cas, l'usage de substr() ou vec() pour simuler des tableaux peut être très bénéfique. Par exemple, un tableau d'un millier de booléens prendra au moins 20 000 octets, mais il peut être remplacé par un vecteur de 125 octets – réalisant ainsi une économie considérable de mémoire. Le module standard Tie::SubstrHash peut aussi aider pour certains types de structures de données. Si vous travaillez avec des structures de données spécifiques (matrices, par exemple), les modules qui les implémentent en C peuvent utiliser moins de mémoire que leurs équivalents en Perl.

Autre chose, essayez de savoir si Perl a été compilé avec le malloc système ou le malloc de Perl. Quel qu'il soit, essayez d'utiliser l'autre, et voyez si cela fait une différence. Les informations sur malloc se trouvent dans le fichier *INSTALL* du source de la distribution. Vous pouvez savoir quel malloc vous utilisez en tapant `perl -V:usemymalloc`.

Évidemment, le moeyn le plus sûr de réduire l'occupation mémoire est d'en utiliser le moins possible dès le départ. De bonnes pratiques de programmation peuvent aider :

– Ne faites pas de slurp !
  Ne lisez pas un fichier complet en mémoire si vous pouvez le traiter ligne par ligne. Plus concrètement, utilisez une boucle comme :

```
#
# Bonne idée
#
while (<FILE>) {
   # ...
}
```

  plutôt que :

```
#
# Mauvaise idée
#
@data = <FILE>;
foreach (@data) {
    # ...
}
```

  Tant que les fichiers traités sont petits, cela n'a pas beaucoup d'importance mais s'ils deviennent gros alors cela fera une énorme différence.
– Utilisez map et grep intelligement
  Souvenez-vous que map et grep utilise un argument sous forme de LISTE, donc faire :

```
@wanted = grep {/pattern/} <FILE>;
```

chargera le fichier complet en mémoire. Pour de gros fichiers, il vaut mieux utiliser une boucle :

```
while (<FILE>) {
        push(@wanted, $_) if /pattern/;
}
```

– Évitez les guillemets et les stringifications inutiles
Ne placez de grande chaînes de caractères entre guillemets si ce n'est pas absolument nécessaire. Le code suivant :

```
my $copie = "$grande_chaine";
```

réalise 2 copies de $grande_chaine (une pour $copie et une autre pour les guillemets). Alors que :

```
my $copie = $grande_chaine;
```

ne fait qu'une seule copie.
C'est la même chose pour la stringification (la transformation en chaîne) de gros tableaux :

```
{
        local $, = "\n";
        print @gros_tableau;
}
```

est plus efficace d'un point de vue occupation mémoire que :

```
print join "\n", @gros_tableau;
```

ou que :

```
{
        local $" = "\n";
        print "@gros_tableau";
}
```

– Utilisez les références
Passez vos tableaux et tables de hachages par référence plutôt que par valeur. Déjà, c'est le seul moyen de passer plusieurs listes ou tables de hachage (ou les deux) en un seul appel ou return. Cela évite aussi une recopie de tout leur contenu. Par contre, il faut le faire avec précaution car tout changement sera alors propagé aux données originales. Si vous avez réellement besoin de modifier une copie, vous devrez sacrifier la mémoire nécessaire à cette copie.
– Liées les grosses variables avec le disque
Pour de très "grosses" structures de données (celles dont la taille dépasse la taille mémoire disponible), pensez à utiliser l'un des modules DB qui stocke l'information sur disque plutôt qu'en mémoire. Cela a évidemment un impact sur le temps d'accès mais ce sera très probablement plus efficace que d'utiliser de façon massive le swap mémoire sur le disque.

2.17 Est-ce sûr de retourner un pointeur sur une donnée locale ?

Oui. Le ramasse-miettes (le gestionnaire de mémoire) de Perl fait attention à cela.

```
sub makeone {
    my @a = ( 1 .. 10 );
    return \@a;
}

for ( 1 .. 10 ) {
    push @many, makeone();
}

print $many[4][5], "\n";

print "@many\n";
```

2.18 Comment puis-je libérer un tableau ou une table de hachage pour réduire mon programme ?

(contribution de Michael Carman)

Vous ne pouvez pas. La mémoire allouée aux variables lexicales (c'est à dire les variables my()) ne peut pas être réutilisée même si on sort de sa portée (NdT: mais ce n'est heureusement pas le cas des données référencées par ces variables). Elle reste réservée au cas où la variable reviendrait à nouveau à portée. Le mémoire allouée aux variables globales peut être réutilisée (par votre programme) en leur appliquant undef() ou delete().

Sur la plupart des systèmes d'exploitation, la mémoire allouée à un programme ne peut pas être retournée au système. C'est pourquoi les programmes ayant un temps d'exécution très long se ré-exécutent eux-même. Quelques systèmes d'exploitation (notamment, les systèmes utilisant mmap(2)) peuvent parfois récupérer certains espaces mémoires qui ne sont plus utilisés, mais, pour que cela fonctionne avec Perl, il faut que perl soit configuré et compilé pour utiliser le malloc du système et non celui de perl.

En général, vous n'avez pas à (et vous ne devriez pas) vous préoccuper de l'allocation et de la désallocation de mémoire avec Perl.

Voir aussi "Comment faire pour que mes programmes Perl occupent moins de mémoire ?".

2.19 Comment rendre mes scripts CGI plus efficaces ?

Après les mesures classiques décrites pour rendre vos programmes Perl plus rapides ou courts, il y a d'autres possibilités pour les programmes CGI. Il peut être exécuté plusieurs fois par seconde. Étant donné qu'à chaque fois, il doit être recompilé, et demande un mégaoctet ou plus de mémoire allouée, cela peut couler le système en performances. Compiler en C **ne vous avancera pas davantage** car le démarrage du processus est le point le plus lent.

Il y a 2 façons classiques pour éviter cette surcharge. Une solution consiste à exécuter le serveur HTTP Apache (disponible à http://www.apache.org/) avec le module mod_perl ou mod_fastcgi.

Avec mod_perl et le module Apache::Registry (distribué avec mod_perl), httpd fonctionne avec un interpréteur Perl inclus qui précompile votre script puis l'exécute dans le même espace mémoire sans fork. L'extension Apache donne aussi à Perl l'accès à la librairie API du serveur, ce qui fait qu'un module écrit en Perl peut faire quasiment tout ce qu'un module en C peut. Pour en savoir plus sur mod_perl, voyez http://perl.apache.org/.

Avec le module FCGI (du CPAN) et le module mod_fastcgi (disponible sur http://www.fastcgi.com/) chacun de vos programmes devient un processus CGI démon permanent.

Chacune de ces 2 solutions peut avoir de grandes conséquences sur votre système et sur votre façon d'écrire vos programmes CGI, donc utilisez-les avec précaution.

Voir http://www.cpan.org/modules/by-category/15_World_Wide_Web_HTML_HTTP_CGI/.

2.20 Comment dissimuler le code source de mon programme Perl ?

Effacez-le. :-) Plus sérieusement, il y a bon nombre de solutions (pour la plupart peu satisfaisante) avec différents degrés de "sécurité".

Tout d'abord, en aucun cas, vous ne pouvez ôter le droit de lecture du script, car le code source doit pouvoir être lu pour pouvoir être compilé et interprété. (Ce qui ne signifie pas que le code source d'un script CGI est accessible en lecture pour les visiteurs du site web – il l'est uniquement pour ceux qui ont accès au système de fichiers). Donc vous devez laisser ces permissions au niveau socialement sympathique de 0755.

Certains voient cela comme un problème de sécurité. Si votre programme fait des actions mettant en cause la sécurité du système, et repose sur la confiance aveugle que vous avez dans le fait que les visiteurs ne savent pas comment exploiter ces trous de sécurités, alors, votre programme n'est pas sécurisé. Il est souvent possible de détecter ce genre de trous et de les exploiter sans jamais voir le code source. La sécurité qui repose sur l'obscurité, autrement dit qui repose sur la non visibilité des bugs (au lieu de les résoudre) est une sécurité très faible.

Vous pouvez essayer d'utiliser le cryptage du code via des filtres de source (depuis la version 5.8 les modules Filter::Simple et Filter::Util::Call sont inclus dans la distribution standard), mais n'importe quel programmeur sérieux pourra les décrypter. Vous pouvez essayer d'utiliser le compilateur et interpréteur en binaire décrit ci-dessous, mais les plus curieux pourront encore le décompiler. Vous pouvez essayer le compilateur en code natif décrit ci-dessous, mais des crackers peuvent encore le désassembler. Cela pose différents degrés de difficultés aux personnes qui en veulent à votre code, mais ça ne les empêchera pas de le retrouver (c'est vrai pour n'importe quel langage, pas juste pour Perl).

Il est très facile de récupérer les sources d'un programme Perl. Il vous suffit de fournir ce programme à l'interpréteur perl et d'utiliser les modules de la hiérarchie B::*. Le module B::Deparse, par exemple, fait échouer la plupart des tentatives de dissumulation de code. Encore une fois, cela n'a rien de spécifique à Perl.

Si cela vous ennuie que des personnes puissent profiter de votre code, alors la première chose à faire est d'indiquer une licence restrictive au début de votre code, ce qui vous donne une sécurité légale. Mettez une licence à votre programme et saupoudrez-le de menaces diverses telle que "Ceci est un programme privé non distribué, de la société XYZ. Votre accès à ce code source ne vous donne pas le droit de l'utiliser bla bla bla." Nous ne sommes pas juristes, bien sûr, donc vous devriez en consulter un pour vous assurer que votre texte de licence tient devant un tribunal.

2.21 Comment compiler mon programme Perl en code binaire ou C ?

(contribution de brian d foy)

De manière générale, ce n'est pas possible. Il existe dans certains cas des solutions. Les gens demandent souvent cela parce qu'ils veulent distribuer leur travail sans donner le code source. Vous ne verrez sans doute aucune amélioration des performances puisque la plupart des solutions ne font qu'intégrer un interpréteur Perl au produit final (mais voyez tout de même Comment rendre mes programmes Perl plus rapides ?).

Le Perl Archive Tooolkit (http://par.perl.org/index.cgi) est à Perl ce que JAR est à Java. Il est disponible sur CPAN (http://search.cpan.org/dist/PAR/).

La hiérarchie B::*, parfois appelée "le compilateur Perl", est en fait un moyen pour les programmes Perl d'accéder à leurs propres entrailles plutôt qu'un moyen de créer des versions pré-compilées. Mais il est vrai que le module B::Bytecode peut transformer votre script en un code binaire que vous pouvez ensuite recharger via le module ByteLoader pour l'exécuter comme un script Perl normal.

Il existe quelques produits commerciaux qui font le travail pour vous si vous achetez la licence correspondante.

Le Perl Dev Kit (http://www.activestate.com/Products/Perl_Dev_Kit/) d'ActiveState peut "transformer votre programme Perl en un fichier directement exécutable sur HP-UX, Linux, Solaris et Windows."

Perl2Exe (http://www.indigostar.com/perl2exe.htm) est un programme en ligne de commande qui converit les scripts perl en exécutables. Il existe aussi bien pour Windows que pour les plateformes unix.

2.22 Comment compiler Perl pour en faire du Java ?

Vous pouvez intégrer Java et Perl avec le Perl Resource Kit d'O'Reilly Media. Voir http://www.oreilly.com/catalog/prkunix/.

Perl 5.6 vient avec Java Perl Lingo (ou JPL). JPL, qui est encore en développement, permet d'appeler du code Perl depuis Java. Lire le fichier jpl/README dans l'arborescence des sources Perl.

2.23 Comment faire fonctionner `#!perl` sur [MS-DOS,NT,...] ?

Pour OS/2 utilisez juste

```
extproc perl -S -your_switches
```

comme première ligne dans le fichier `*.cmd` (`-S` est dû à un bug dans la gestion de "extproc" par cmd.exe). Pour DOS, il faudrait d'abord que quelqu'un créer un fichier batch similaire puis le codifie dans `ALTERNATIVE_SHEBANG` (voir le fichier *INSTALL* dans la distribution pour plus d'informations).

L'installation sur Win95/98/NT, avec le portage Perl de ActiveState, va modifier la table de registre pour associer l'extension `.pl` avec l'interpréteur perl. Si vous utilisez un autre portage, peut-être même en compilant votre propre Perl Win95/98/NT à l'aide d'une version Windows de gcc (e.g. avec cygwin ou mingw32), alors vous devrez modifier la base de registres vous-même. En plus d'associer `.pl` avec l'interpréteur, les gens sous NT peuvent utiliser `SET PATHEXT=%PATHEXT%;.PL` pour qu'ils puissent exécuter le programme `install-linux.pl` en tapant simplement `install-linux`.

Sous "Classic" ou MacOS, les programmes Perl auront les attributs Créateur et Type appropriés, un double-clic dessus appellera donc l'application Perl. Sous Mac OS X, les scripts utilisant une ligne `#!...` peuvent être transformés en vraies applications via l'utilitaire DropScript de Wil Sanchez (http://www.wsanchez.net/software/).

IMPORTANT! : Quoi que vous fassiez, SURTOUT ne vous contentez pas seulement d'installer votre interpréteur dans votre répertoire cgi-bin, pour faire fonctionner votre serveur web avec vos programmes. C'est un ÉNORME risque de sécurité. Prenez le temps de bien vérifier que tout fonctionne correctement.

2.24 Puis-je écrire des programmes Perl pratiques sur la ligne de commandes ?

Oui. Lisez *perlrun* pour plus d'informations. Voici quelques exemples. (on considère ici un shell Unix avec les règles standard d'apostrophes.)

```
# Additionner le premier et lz dernier champs
perl -lane 'print $F[0] + $F[-1]' *

# Identifier des fichiers-textes
perl -le 'for(@ARGV) {print if -f && -T _}' *
```

```
# enlever la plupart des commentaires d'un programme C
perl -0777 -pe 's{/\*.*?\*/}{}gs' foo.c

# Rajeunir un fichier d'un mois
perl -e '$X=24*60*60; utime(time(),time() + 30 * $X,@ARGV)' *

# Trouver le premier uid non utilisé
perl -le '$i++ while getpwuid($i); print $i'

# Afficher des chemins raisonnables vers des répertoires man
echo $PATH | perl -nl -072 -e '
    s![^/+]*$!man!&&-d&&!$s{$_}++&&push@m,$_;END{print"@m"}'
```

OK, le dernier n'est pas très simple. :-)

2.25 Pourquoi les commandes Perl à une ligne ne fonctionnent-elles pas sur mon DOS/Mac/VMS ?

Le problème est généralement que les interpréteurs de commandes sur ces systèmes ont des points de vue différents sur les apostrophes, guillemets, etc, par rapport aux shell Unix sous lesquels a été créée cette possibilité de commande à une ligne. Sur certains systèmes, vous devrez changer les apostrophes en guillemets, ce que vous ne devez *PAS* faire sur Unix ou sur des systèmes Plan9. Vous devrez aussi probablement changer un simple % en %%.

Par exemple :

```
# Unix
perl -e 'print "Hello world\n"'

# DOS, etc.
perl -e "print \"Hello world\n\""

# Mac
print "Hello world\n"
 (then Run "Myscript" or Shift-Command-R)

# MPW
perl -e 'print "Hello world\n"'

# VMS
perl -e "print ""Hello world\n"""
```

Le problème est que rien de tout cela n'est garanti : cela dépend de l'interpréteur de commande. Sous Unix, les deux premiers exemples marchent presque toujours. Sous DOS il est bien possible qu'aucun d'entre eux ne fonctionne. Si 4DOS est l'interpréteur de commandes, vous aurez probablement plus de chances avec ceci :

```
perl -e "print <Ctrl-x>"Hello world\n<Ctrl-x>""
```

Sous Mac, cela dépend de l'environnement que vous utilisez. Le shell MacPerl ou MPW, ressemble plutôt aux shells Unix car il accepte pas mal de variantes dans les apostrophes, guillemets, etc, excepté qu'il utilise librement les caractères de contrôle Mac non ASCII comme des caractères normaux.

L'usage de qq(), q() et qx(), à la place de "guillemets", d'"apostrophes' et d''accents graves' peut rendre les programmes sur une ligne plus faciles à écrire.

Il n'y a pas de solution globale à tout cela. Il y a un manque.

[Kenneth Albanowski a contribué à certaines de ces réponses.]

2.26 Où puis-je en apprendre plus sur la programmation CGI et Web en Perl ?

Pour les modules, prenez les modules CGI ou LWP au CPAN. Pour les bouquins, voyez les deux tout spécialement dédiés au développement pour le web, dans la question sur les livres. Pour des problèmes ou questions du style "Pourquoi ai-je une erreur 500" ou "Pourquoi cela ne fonctionne-t-il pas bien par le navigateur alors que tout marche depuis la ligne de commande shell", lisez <perlfaq9> ou la MetaFAQ CGIs :

```
http://www.perl.org/CGI_MetaFAQ.html
```

2.27 Où puis-je apprendre la programmation orientée objet en Perl ?

Un bon point de départ est *perltoot* puis ensuite vous pouvez utiliser *perlobj*, *perlboot*, *perltooc* et *perlbot* comme références.

Deux bons livres sur la programmation orientée objet en Perl sont "Object-Oriented Perl" de Damian Conway chez Manning et ""Learning Perl ! References, Objects, & Modules" par Randal Schwartz et Tom Phoenix chez O'Reilly Media.

2.28 Où puis-je en apprendre plus sur l'utilisation liée de Perl et de C ?

Si vous voulez appeler du C à partir du Perl, commencez avec *perlxstut*, puis *perlxs*, *xsubpp*, et *perlguts*. Si vous voulez appeler du Perl à partir du C, alors lisez *perlembed*, *perlcall*, et *perlguts*. N'oubliez pas que vous pouvez apprendre beaucoup en regardant comment des auteurs de modules d'extension ont écrit leur code et résolu leurs problèmes.

Vous n'avez peut-être pas besoin de toute la puissance de XS. Le module Inline::C vous permet de placer du code C directement dans votre script Perl. Il gère lui-même toute la magie nécessaire au bon fonctionnement. Vous devez encore connaître un minimum de choses sur l'API Perl mais vous n'avez plus à gérer toute la complexité des fichiers XS.

2.29 J'ai lu perlembed, perlguts, etc., mais je ne peux inclure du perl dans mon programme C, qu'est ce qui ne va pas ?

Téléchargez le kit ExtUtils::Embed depuis CPAN et exécutez 'make test'. Si le test est bon, lisez les pods encore et encore et encore. Si le test échoue, voyez *perlbug* et envoyez un rapport de bug avec les sorties écran de `make test TEST_VERBOSE=1` ainsi que de `perl -V`.

2.30 Quand j'ai tenté d'exécuter mes scripts, j'ai eu ce message. Qu'est ce que cela signifie ?

Une liste complète des messages d'erreur et des avertissements de Perl accompagnés d'un texte explicatif se trouve dans *perldiag*. Vous pouvez aussi utiliser le programme splain (distribué avec perl) pour expliquer les messages d'erreur :

```
perl program 2>diag.out
splain [-v] [-p] diag.out
```

ou modifiez votre programme pour qu'il vous explique les messages :

```
use diagnostics;
```

ou

```
use diagnostics -verbose;
```

2.31 Qu'est-ce que MakeMaker ?

Ce module (qui fait partie de la distribution standard de Perl) est fait pour écrire un Makefile pour un module d'extension à partir d'un Makefile.PL. Pour plus d'informations, voyez *ExtUtils::MakeMaker*.

3 AUTEUR ET COPYRIGHT

4 TRADUCTION

4.1 Version

Cette traduction française correspond à la version anglaise distribuée avec perl 5.8.8. Pour en savoir plus concernant ces traductions, consultez http://perl.enstimac.fr/.

4.2 Traducteur

Sébastien Joncheray <info@raynette.com>. Mise à jour : Paul Gaborit <paul.gaborit at enstimac.fr>.

4.3 Relecture

Pascal Ethvignot <pascal@encelade.frmug.org>, Roland Trique <roland.trique@uhb.fr>, Gérard Delafond.

5 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.