perldebug

Table des matières

1 NAME/NOM

perldebug - Débogage de Perl

2 DESCRIPTION

Tout d'abord, avez-vous essayé d'utiliser l'option **-w** ?

3 Le Débogueur Perl

Si vous invoquez Perl avec l'option **-d**, votre script tournera dans le débogueur de sources Perl. Il fonctionne comme un environnement Perl interactif, demandant des commandes de débogage qui vous laissent examiner le code source, placer des points d'arrêt, obtenir des traces des états passés de la pile, changer les valeurs des variables, etc. C'est si pratique que vous lancez souvent le débogueur tout seul juste pour tester interactivement des constructions en Perl afin de voir ce qu'elles font. Par exemple :

```
$ perl -d -e 42
```

En Perl, le débogueur n'est pas un programme séparé à la façon dont c'est habituellement le cas dans l'environnement compilé typique. À la place, l'option **-d** dit au compilateur d'insérer des informations sur le source dans les arbres d'analyse qu'il va donner à l'interpréteur. Cela signfie que votre code doit d'abord se compiler correctement pour que le débogueur travaille dessus. Puis lorsque l'interpréteur démarre, il précharge une bibliothèque Perl spéciale contenant le débogueur lui-même.

Le programme s'arrêtera *juste avant* la première instruction de son exécution (mais voyez plus bas en ce qui concerne les instructions pendant la compilation) et vous demandera d'entrer une commande de débogage. Contrairement à ce qu'on pourrait attendre, lorsque le débogueur s'arrête et vous montre une ligne de code, il affiche toujours la ligne qu'il est *sur le point* d'exécuter, plutôt que celle qu'il vient juste d'exécuter.

Toute commande non reconnue par le débogueur est directement exécutée `eval`uée) comme du code Perl dans le paquetage courant (Le débogueur utilise le paquetage DB pour gérer les informations sur son propre état).

Tout espace blanc précédant ou suivant un texte entré au prompt du débogueur est d'abord supprimé avant tout autre traitement. Si une commande de débogage coïncide avec une fonction de votre propre programme, faites précéder simplement la fonction par quelque chose qui n'a pas l'air d'une commande de débogage, tel qu'un `;`, ou peut-être un `+`, ou en l'encadrant avec des parenthèses ou des accolades.

3.1 Commandes du Débogueur

Le débogueur comprend les commandes suivantes :

h [commande]

Affiche un message d'aide.

Si vous fournissez une autre commande de débogage comme argument de la commande `h`, elle affichera uniquement la description de cette commande. L'argument spécial `h h` produira un listage d'aide plus compact, conçu pour tenir en un seul écran.

Si la sortie de la commande `h` (ou de toute commande, en fait) est plus longue que votre écran, faites-la précéder d'un symbole de tube pour qu'elle soit affichée page par page, comme dans

```
DB> |h
```

Vous pouvez changer le programme de pagination utilisé via la commande `O pager=...`.

p expr

Identique à `print {$DB::OUT} expr` dans le paquetage courant. Cela signifie en particulier que, puisque c'est simplement la propre fonction `print` de Perl, les structures de données imbriquées et les objets ne sont pas affichés, contrairement à ce qui se passe avec la commande `x`.

Le handle de fichier `DB::OUT` est ouvert vers */dev/tty*, quelle que soit la redirection possible de STDOUT.

x expr

Évalue son expression dans un contexte de liste et affiche le résultat d'une façon joliment formatée. Les structures de données imbriquées sont affichées récursivement, contrairement à ce qui se passe avec la vraie fonction `print`. Voir *Dumpvalue* si vous aimeriez faire cela vous-même.

Le format de sortie est gouverné par de multiples options décrites sous Options Configurables (§3.2).

V [pkg [vars]]

Affiche la totalité (ou une partie) des variables du paquetage (avec par défaut `main`) en utilisant un joli afficheur de données (les hachages montrent leurs couples clé-valeur de façon que vous voyiez qui correspond à qui, les caractères de contrôle sont rendus affichables, etc.). Assurez-vous de ne pas placer là de spécificateur de type (comme `$`), mais juste les noms de symboles, comme ceci :

```
V DB filename line
```

Utilisez `~pattern` et `!pattern` pour avoir des expressions rationnelles positives et négatives.

Ceci est similaire au fait d'appeler la commande `x` pour chaque variable applicable.

X [vars]

Identique à `V currentpackage [vars]`.

T

Produit une trace de la pile. Voir plus bas pour des détails sur sa sortie.

s [expr]

Pas à pas. Poursuit l'exécution jusqu'au début de l'instruction suivante, en descendant dans les appels de sous-programmes. Si une expression comprenant des appels de fonction est fournie, elle sera elle aussi suivie pas à pas.

n [expr]

Suivant. Exécute les appels de sous-programme, jusqu'à atteindre le début de la prochaine instruction. Si une expression comprenant des appels de fonction est fournie, ces fonctions seront exécutées avec des arrêts avant chaque instruction.

r

Continue jusqu'au retour du sous-programme courant. Affiche la valeur de retour si l'option `PrintRet` est mise (valeur par défaut).

<CR>

Répète la dernière commande `n` ou `s`.

c [line|sub]

Continue, en insérant facultativement un point d'arrêt valable une fois seulement à la ligne ou au sous-programme spécifié.

l

Liste la prochaine fenêtre de lignes.

l min+incr

Liste `incr+1` lignes en commençant à `min`.

l min-max

Liste les lignes de `min` à `max`. `l -` est synonyme de `-`.

l line

Liste une seule ligne.

l subname

Liste la première fenêtre de lignes en provenance d'un sous-programme. *subname* peut être une variable contenant une référence de code.

-

Liste la précédente fenêtre de lignes.

w [line]

Liste une fenêtre (quelques lignes) autour de la ligne courante.

.

Retourne le pointeur de débogage interne sur la dernière ligne exécutée, et affiche cette ligne.

f filename

Passe à la visualisation d'un fichier différent ou d'une autre instruction `eval`. Si *filename* n'est pas un chemin complet tel que trouvé dans les valeurs de %INC, il est considéré être une expression rationnelle.

/motif/

Recherche un motif (une expression rationnelle de Perl) vers l'avant ; le / final est optionnel.

?motif?

Recherche un motif vers l'arrière ; le ? final est optionnel.

L

Liste tous les points d'arrêts et toutes les actions.

S [[!]regex]

Liste les noms de sous-programmes [sauf] ceux correspondant à l'expression rationnelle.

t

Bascule le mode de traçage (voir aussi l'`O`option `AutoTrace`).

t expr

Trace l'exécution de expr. Voir Exemples de Listages des Frames in *perldebguts* pour des exemples.

b [ligne] [condition]

Place un point d'arrêt avant la ligne donnée. Si ligne est omise, place un point d'arrêt sur la ligne qui est sur le point d'être exécutée. Si une condition est spécifiée, elle est évaluée chaque fois que l'instruction est atteinte : un point d'arrêt est réalisé seulement si la condition est vraie. Les points d'arrêt ne peuvent être placés que sur les lignes qui commencent une instruction exécutable. Les conditions n'utilisent pas **if** :

```
b 237 $x > 30
b 237 ++$count237 < 11
b 33 /pattern/i
```

b subname [condition]

Place un point d'arrêt avant la première ligne du sous-programme nommé. *subname* peut être une variable contenant une référence de code (dans ce cas *condition* n'est pas supporté).

b postpone subname [condition]

Place un point d'arrêt à la première ligne du sous-programme après sa compilation.

b load filename

Place un point d'arrêt avant la première ligne exécutée du *fichier*, qui doit être un chemin complet trouvé parmi les valeurs %INC.

b compile subname

Place un point d'arrêt à la première instruction exécutée après que le sous-programme spécifié ait été compilé.

d [ligne]

Supprime un point d'arrêt sur la *ligne* spécifiée. Si *ligne* est omis, supprime le point d'arrêt sur la ligne sur le point d'être exécutée.

D

Supprime tous les points d'arrêt définis.

a [ligne] commande

Fixe une action devant être effectuée avant que la ligne ne soit exécutée. Si *ligne* est omis, place une action sur la ligne sur le point d'être exécutée. La séquence d'opérations réalisées par le débogueur est :

```
1. vérifie la présence d'un point d'arrêt sur cette ligne
2. affiche la ligne si nécessaire (trace)
3. effectue toutes les actions associées à cette ligne
4. interroge l'utilisateur en cas de point d'arrêt ou de pas à pas
5. évalue la ligne
```

Par exemple, ceci affichera $foo chaque fois que la ligne 53 sera passée :

```
a 53 print "DB FOUND $foo\n"
```

a [ligne]

Supprime une action de la ligne spécifiée. Si *ligne* est omis, supprime l'action de la ligne sur le point d'être exécutée.

A

Supprime toutes les actions définies.

W expression

Ajoute une expression de surveillance (« watch » ? NDT) globale. Nous espérons que vous savez ce que c'est, car elles sont supposées être évidentes. **AVERTISSEMENT** : il est bien trop facile de détruire vos actions de surveillance en omettant accidentellement l'*expression*.

W

Supprime toutes les expressions de surveillance.

O booloption ...

Fixe chaque option booléenne listée à la valeur `1`.

O anyoption? ...

Affiche la valeur d'une ou plusieurs options.

O option=value ...

Fixe la valeur d'une ou plusieurs options. Si la valeur contient des espaces, elle doit être mise entre guillemets. Par exemple, vous pouvez définir `O +pager="less -MQeicsNfr"` pour appeler **less** avec ces options spécifiques. Vous pouvez utiliser des apostrophes ou des guillemets, mais si vous le faites, vous devez protéger toutes les occurrences d'apostrophes ou de guillemets (respectivement) dans la valeur, ainsi que les échappements les précédant immédiatement mais ne devant pas les protéger. En d'autres termes, vous suivez les règles de guillemettage indépendamment du symbole ; e.g. : `O option='this isn\'t bad'` or `O option="She said, \"Isn't it?\""`.

Pour des raisons historiques, le `=value` est optionnel, mais a pour valeur par défaut 1 uniquement lorsque cette valeur est sûre – c'est-à-dire principalement pour les options booléennes. Il vaut toujours mieux affecter une valeur spécifique en utilisant `=`. L'`option` peut être abrégé, mais ne devrait probablement pas l'être par souci de clarté. Plusieurs options peuvent être définies ensemble. Voir Options Configurables (§3.2) pour en trouver une liste.

< ?

Affiche toutes les commandes Perl constituant les actions précédant le prompt.

< [command]

Définit une action (une commande Perl) devant se produire avant chaque prompt du débogueur. Une commande sur plusieurs lignes peut être entrée en protégeant les fins de ligne par une barre oblique inverse. **AVERTISSEMENT** Si `command` est absent, toutes les actions sont effacées !

<< command

Ajoute une action (une commande Perl) devant se produire avant chaque prompt du débogueur. Une commande sur plusieurs lignes peut être entrée en protégeant les fins de ligne par une barre oblique inverse.

> ?

Affiche les commandes Perl constituant les actions suivant le prompt.

> command

Définit une action (une commande Perl) devant se produire après le prompt lorsque vous venez d'entrer une commande provoquant le retour à l'exécution du script. Une commande sur plusieurs lignes peut être entrée en protégeant les fins de ligne par des barres obliques inverses (parions que vous ne pouviez pas le deviner). **AVERTISSEMENT** Si `command` est absent, toutes les actions sont effacées !

>> command

Définit une action (une commande Perl) devant se produire après le prompt lorsque vous venez d'entrer une commande provoquant le retour à l'exécution du script. Une commande sur plusieurs lignes peut être entrée en protégeant les fins de ligne par des barres obliques inverses.

{ ?

Affiche les commandes du débogueur précédant le prompt.

{ [command]

Définit une action (une commande du débogueur) devant se produire avant chaque prompt du débogueur. Une commande sur plusieurs lignes peut être entrée de la façon habituelle. **AVERTISSEMENT** Si `command` est absent, toutes les actions sont effacées !

Puisque cette commande est en quelque sorte nouvelle, un avertissement est généré s'il apparaît que vous avez accidentellement entré un bloc. Si c'est ce que vous vouliez faire, écrivez-le sous la forme `;{ ... }` ou même `do { ... }`.

{{ command

Ajoute une action (commande du débogueur) devant se produire avant chaque prompt du débogueur. Une commande sur plusieurs lignes peut être entrée, si vous ne savez pas comment : voyez ci-dessus.

! number

Relance une commande précédente (la commande précédente, par défaut).

! -number

Relance la nième commande précédente

! pattern

Relance la dernière commande ayant commencé par le motif. Voir aussi `O recallCommand`.

!! cmd

Lance cmd dans un sous-processus (lisant sur DB::IN, écrivant sur DB::OUT). Voir aussi `O shellBang`. Notez que le shell courant de l'utilisateur (en fait, sa variable `$ENV{SHELL}`) sera utilisé, ce qui peut interférer avec une interprétation correcte du statut ou du signal de sortie et les informations de coredump.

H -number

Affiche les n dernières commandes. Seules les commandes de plus de un caractère sont affichées. Si *number* est omis, les affiche toutes.

q or ^D

Quitte ("quit" ne marche pas ici, à moins que vous en ayez créé un alias). C'est la seule manière supportée pour quitter le débogueur, même si le fait d'entrer `exit` deux fois peut fonctionner.

Placez l'option `inhibit_exit` à 0 si vous voulez être en mesure de sauter jusqu'à la fin du script. Vous pouvez aussi avoir besoin de placer $finished à 0 si vous voulez avancer pas à pas dans la destruction globale.

R

Redémarre le débogueur en `exec()`utant une nouvelle session. Nous essayons de maintenir votre historique en cours de route, mais certains réglages internes et les options de la ligne de commande peuvent être perdus.

Les réglages suivants sont actuellement préservés : l'historique, les points d'arrêt, les actions, les options du débogueur, et les options de ligne de commande de Perl **-w**, **-I**, and **-e**.

|dbcmd

Exécute la commande du débogueur en redirigeant DB::OUT dans votre pager courant.

||dbcmd

De même que |`dbcmd` mais un `select` temporaire de DB::OUT est réalisé.

= [alias value]

Définit un alias de commande, comme

```
= quit q
```

ou liste les alias définis.

command

Exécute une commande en tant qu'instruction Perl. Un point-virgule final y sera ajouté. Si l'instruction Perl peut être confondue avec une commande du débogueur Perl, faites-la aussi précéder d'un point-virgule.

m expr

Liste quelles méthodes peuvent être appelées sur le résultat de l'expression évaluée. Cette évaluation peut être une référence à un objet consacré, ou à un nom de paquetage.

man [manpage]

Malgré son nom, cette commande appelle le visualisateur de documentation par défaut de votre système, pointant sur la page en argument, ou sur sa page par défaut si *manpage* est omis. Si ce visualisateur est **man**, les informations courantes de `Config` sont utilisées pour invoquer **man** via le MANPATH correct ou l'option **-M** *manpath*. Les recherches ratées de la forme `XXX` correspondant à des pages de manuel connues de la forme *perlXXX* seront réessayées. Ceci vous permet de taper `man debug` ou `man op` depuis le débogueur.

Sur les systèmes traditionnellement privés d'une commande **man** utilisable, le débogueur invoque **perldoc**. Cette détermination est occasionnellement incorrecte à cause de vendeurs récalcitrants ou, de façon plus fort à propos, du fait d'utilisateurs entreprenants. Si vous tombez dans une de ces catégories, fixez simplement manuellement la variable $DB::doccmd pour qu'elle pointe vers le visualisateur permettant de lire la documentation Perl sur votre système. Ceci peut être placé dans un fichier rc, ou défini via une affectation directe. Nous attendons toujours un exemple fonctionnel de quelque chose ressemblant à :

```
$DB::doccmd = 'netscape -remote http://something.here/';
```

3.2 Options Configurables

Le débogueur a de nombreuses options définissables via la commande `O`, soit de façon interactive, soit via l'environnement ou un fichier rc.

`recallCommand`, `ShellBang`

Les caractères utilisés pour rappeler une commande ou générer un nouveau shell. Par défaut, ces deux variables sont fixées à `!`, ce qui est malheureux.

`pager`

Programme à utiliser pour la sortie des commandes redirigées par un tube vers un paginateur (pager ? NDT) (celles qui commencent par un caractère |). Par défaut, `$ENV{PAGER}` sera utilisé. Puisque le debogueur utilise les caractéristiques de votre terminal courant pour les caractères gras et le soulignement, si le pager choisi ne transmet pas sans changements les séquences d'échappement, la sortie de certaines commandes du débogueur ne seront plus lisibles après qu'elles aient traversé le pager.

`tkRunning`

Exécute Tk pour le prompt (avec ReadLine).

`signalLevel`, `warnLevel`, `dieLevel`

Niveau de verbosité. Par défaut, le débogueur laisse tranquille vos exceptions et vos avertissements, parce que les altérer peut empêcher le bon fonctionnement de certains programmes. Il essayera d'afficher un message lorsque des signaux INT, BUS ou SEGV arriveront sans être traités (mais voyez la mention des signaux dans BUGS (§7) ci-dessous).

Pour désactiver ce mode sûr par défaut, placez ces valeurs à quelque chose supérieur à 0. À un niveau de 1, vous obtenez une trace pour tous les types d'avertissements (c'est souvent gênant) et toutes les exceptions (c'est souvent pratique). Malheureusement, le débogueur ne peut pas discerner les exceptions fatales et non-fatales. Si `dieLevel` vaut 1, alors vos exceptions non-fatales sont aussi tracées et altérées sans cérémonie si elle proviennent de chaînes `eval`uées ou de tout type d'`eval` à l'intérieur des modules que vous essayez de charger. Si `dieLevel` est à 2, le débogueur ne se soucie pas de leur provenance : il usurpe vos handlers d'exceptions et affiche une trace, puis modifie toutes les exceptions avec ses propres embellissements. Ceci peut être utile pour certaines traces particulières, mais tend à désespérément détruire tout programme qui prend au sérieux sa gestion des exceptions.

AutoTrace
Mode de trace (similaire à la commande `t`, mais pouvant être mise dans `PERLDB_OPTS`).

LineInfo
Fichier ou tube dans lequel écrire les infos sur les numéros de ligne. Si c'est un tube (disons, `|visual_perl_db`), alors un court message est utilisé. C'est le mécanisme mis en oeuvre pour interagir avec un éditeur esclave ou un débogueur visuel, comme les hooks spéciaux de `vi` ou d'`emacs`, ou le débogueur graphique `ddd`.

inhibit_exit
si à 0, permet *le passage direct* à la fin du script.

PrintRet
Affiche la valeur de retour après la commande `r` s'il est mis (par défaut).

ornaments
Affecte l'apparence de l'écran de la ligne de commande (voir *Term::ReadLine*). Il n'y a actuellement aucun moyen de les désactiver, ce qui peut rendre certaines sorties illisibles sur certains affichages, ou avec certains pagers. C'est considéré comme un bug.

frame
Affecte l'affichage des messages à l'entrée et à la sortie des sous-programmes. Si `frame & 2` est faux, les messages sont affichés uniquement lors de l'entrée (L'affichage à la sortie peut être utile si les messages sont entrecroisés).
Si `frame & 4` est faux, les arguments des fonctions sont affichés en plus du contexte et des infos sur l'appelant. Si `frame & 8` est faux, les `FETCH` surchargés `chaînifiés` (stringify) et `liés` (tie, NDT) sont autorisés sur les arguments affichés. Si `frame & 16` est faux, la valeur de retour du sous-programme est affichée.
La longueur à partir de laquelle la liste d'arguments est tronquée est régie par l'option suivante :

maxTraceLen
La longueur à laquelle la liste d'arguments est tronquée lorsque le bit 4 de l'option `frame` est mis.

Les options suivantes affectent ce qui se produit avec les commandes `V`, `X`, et `x` :

arrayDepth, hashDepth
Affiche seulement les N premiers éléments (" pour tous).

compactDump, veryCompact
Change le style du vidage des tableaux et des hachages. Si `compactDump` est utilisé, les tableaux courts peuvent être affichés sur une seule ligne.

globPrint
Définit si l'on doit afficher le contenu des globalisations.

DumpDBFiles
Vidage des tableaux contenant des fichiers débogués.

DumpPackages
Vidage des tables de symboles des paquetages.

DumpReused
Vidage des contenus des adresses "réutilisées".

quote, HighBit, undefPrint
Change le style du vidage des chaînes. La valeur par défaut de `quote` est `auto` ; on peut permettre le vidage soit entre guillemets, soit entre apostrophes en la fixant à `"` ou `'` respectivement. Par défaut, les caractères dont le bit de poids fort est mis sont affichés *tels quels*.

UsageOnly
Vidage rudimentaire de l'usage de la mémoire par paquetage. Calcule la taille totale des chaînes trouvées dans les variables du paquetage. Ceci n'inclut pas les lexicaux dans la portée d'un fichier de module, ou perdus dans des fermetures.

Pendant le démarrage, les options sont initialisées à partir de `$ENV{PERLDB_OPTS}`. Vous pouvez y placer les options d'initialisation `TTY`, `noTTY`, `ReadLine`, et `NonStop`.

Si votre fichier rc contient :

```
parse_options("NonStop=1 LineInfo=db.out AutoTrace");
```

alors votre script s'exécutera sans intervention humaine, plaçant les informations de trace dans le fichier *db.out* (Si vous l'interrompez, vous avez intérêt à réinitialiser `LineInfo` sur */dev/tty* si vous voulez voir quelque chose).

TTY

Le TTY à utiliser pour les I/O de débogage.

noTTY

Si elle est mise, le débogueur entre en mode `NonStop`, sans se connecter à un TTY. En cas d'interruption (ou si le contrôle passe au débogueur via un réglage explicite de $DB::signal ou de $DB::single par le script Perl), il se connecte au TTY spécifié par l'option `TTY` au démarrage, ou à un TTY trouvé lors de l'exécution en utilisant le module `Term::Rendezvous` de votre choix.

Ce module doit implémenter une méthode appelée `new` qui retourne un objet contenant deux méthodes : `IN` et `OUT`. Celles-ci devraient retourner deux handles de fichiers à utiliser pour déboguer les entrées et sorties, respectivement. La méthode `new` devrait inspecter un argument contenant la valeur de `$ENV{PERLDB_NOTTY}` au démarrage, ou de `"/tmp/perldbtty$$"` autrement. Ce fichier n'est pas inspecté du point de vue de la correction de son appartenance, des risques de sécurité sont donc théoriquement possibles.

ReadLine

Si elle est fausse, le support de readline dans le débogueur est désactivé de façon à pouvoir déboguer les application l'utilisant.

NonStop

Si elle est mise, le débogueur entre en mode non interactif jusqu'à ce qu'il soit interrompu manuellement, ou par programme en fixant $DB::signal ou $DB::single.

Voici un exemple de l'utilisation de la variable `$ENV{PERLDB_OPTS}` :

```
$ PERLDB_OPTS="NonStop frame=2" perl -d myprogram
```

Ceci exécutera le script **myprogram** sans intervention humaine, affichant l'arbre des appels avec leurs points d'entrée et de sortie. Notez que `NonStop=1 frame=2` est équivalent à `N f=2`, et qu'à l'origine, les options ne pouvait être abrégées que par leur première lettre (modulo les options `Dump*`). Il est néanmoins recommandé que vous les énonciez toujours complètement dans un souci de lisibilité et de compatibilité future.

D'autres exemples incluent

```
$ PERLDB_OPTS="NonStop frame=2" perl -d myprogram
```

qui exécute le script de façon non interactive, affichant les infos à chaque entrée dans un sous-programme et pour chaque ligne exécutée du fichier appelé *listing* (Si vous l'interrompez, vous feriez mieux de réinitialiser `LineInfo` vers quelque chose d'« interactif » !).

D'autres exemples incluent (en utilisant la syntaxe standard du shell pour montrer les valeurs des variables d'environnement) :

```
$ ( PERLDB_OPTS="NonStop frame=1 AutoTrace LineInfo=tperl.out"
    perl -d myprogram )
```

qui peut être utile pour déboguer un programme qui utilise lui-même `Term::ReadLine`. N'oubliez pas de détacher le shell du TTY dans la fenêtre qui correspond à */dev/ttyXX*, en entrant une commande comme, disons

```
$ sleep 1000000
```

Voir Éléments Internes du Débogueur in *perldebguts* pour plus de détails.

3.3 entrées/sorties du débogueur

Prompt

Le prompt du débogueur ressemble à

```
DB<8>
```

ou même

```
DB<<17>>
```

où ce nombre est le numéro de la commande, que vous utiliseriez pour y accéder avec le mécanisme intégré d'historique à la mode de **csh**. Par exemple, `!17` répéterait la commande numéro 17. La profondeur des signes inférieur à et supérieur à indique la profondeur d'imbrication du débogueur. Vous pouvez obtenir plus d'un ensemble de signes d'inégalité, par exemple, si vous êtes déjà sur un point d'arrêt et que vous affichez le résultat d'un appel de fonction qui lui-même contient un point d'arrêt, ou si vous sautez dans une expression via la commande `s/n/t expression`.

Commandes multilignes

Si vous désirez entrer une commande multiligne, telle qu'une définition de sous-programme contenant plusieurs instructions ou bien un format, protégez par une barre oblique inverse les fins de lignes qui termineraient normalement la commande de débogage. En voici un exemple :

```
DB<1> for (1..4) {          \
cont:     print "ok\n";    \
cont: }
ok
ok
ok
ok
```

Notez que cette affaire de protection d'une fin de ligne est spécifique aux commandes interactives tapées dans le débogueur.

Trace de la pile

Voici un exemple de ce dont pourrait avoir l'air une trace de la pile via la commande `T` :

```
$ = main::infested called from file 'Ambulation.pm' line 10
@ = Ambulation::legs(1, 2, 3, 4) called from file 'camel_flea' line 7
$ = main::pests('bactrian', 4) called from file 'camel_flea' line 4
```

Le caractère à gauche ci-dessus indique le contexte dans lequel la fonction a été appelée, avec `$` et `@` désignant les contextes scalaires et de liste respectivement, et `.` un contexte vide (qui est en fait une sorte de contexte scalaire). L'affichage ci-dessus signifie que vous étiez dans la fonction `main::infested` lorsque vous avez effectué le vidage de la pile, et qu'elle a été appelée dans un contexte scalaire à la ligne 10 du fichier *Ambulation.pm*, mais sans aucun argument, ce qui indique qu'elle a été appelée en tant que `&infested`. La ligne suivante de la pile montre que la fonction `Ambulation::legs` a été appelée dans un contexte de liste depuis le fichier *camel_flea*, avec quatre arguments. La dernière ligne montre que `main::pests` a été appelée dans un contexte scalaire, elle aussi depuis *camel_flea*, mais à la ligne 4.

Si vous exécutez la commande `T` depuis l'intérieur d'une instruction `use` active, la trace contiendra à la fois une ligne pour `require` et une ligne pour `eval`.

Format de Listage des Lignes

Ceci montre les types de listage que la commande `l` peut produire :

```
  DB<<13>> l
101:                @i{@i} = ();
102:b               @isa{@i,$pack} = ()
103                     if(exists $i{$prevpack} || exists $isa{$pack});
104             }
105
106             next
107==>              if(exists $isa{$pack});
108
109:a           if ($extra-- > 0) {
110:                %isa = ($pack,1);
```

Les lignes sur lesquelles on peut placer un point d'arrêt sont indiquées avec `:`. Les lignes ayant un point d'arrêt sont indiquées par `b` et celles ayant une actions par `a`. La ligne sur le point d'être exécutée est indiquée par `=>`.

Listage de frame (par quoi traduire ? NDT)

Lorsque l'option `frame` est utilisée, le débogueur affiche les entrées dans les sous-programmes (et optionnellement les sorties) dans des styles différents. Voir *perldebguts* pour des exemples incroyablement longs de tout ceci.

3.4 Débogage des instructions lors de la compilation

Si vous avez des instructions exécutables lors de la compilation (comme du code contenu dans un bloc BEGIN ou CHECK ou une instruction `use`), elles ne seront `pas` stoppées par le débogueur, bien que les `requires` et les blocs INIT le soient, et les instructions de compilation peuvent être tracées avec l'option `AutoTrace` mise dans `PERLDB_OPTS`. Depuis votre propre code Perl, toutefois, vous pouvez transférer de nouveau le contrôle au débogueur en utilisant l'instruction suivante, qui est inoffensive si le débogueur n'est pas actif :

```
$DB::single = 1;
```

Si vous fixez `$DB::single` à 2, cela équivaut à avoir juste tapé la commande `n`, tandis qu'une valeur de 1 représente la commande `s`. La variable `$DB::trace` devrait être mise à 1 pour simuler le fait d'avoir tapée la commande `t`.

Une autre façon de déboguer le code exécutable lors de la compilation est de démarrer le débogueur, de placer un point d'arrêt sur le *load* d'un module :

```
   DB<7> b load f:/perllib/lib/Carp.pm
Will stop on load of 'f:/perllib/lib/Carp.pm'.
```

puis de redémarrer le débogueur en utilisant la commande `R` (si possible). On peut utiliser `b compile subname` pour obtenir la même chose.

3.5 Personnalisation du Débogueur

Le débogueur contient probablement suffisamment de hooks de configuration pour que vous n'ayez jamais à le modifier vous-même. Vous pouvez changer le comportement du débogueur depuis le débogueur lui-même, en utilisant sa commande `O`, depuis la ligne de commande via la variable d'environnement `PERLDB_OPTS`, et par des fichiers de personnalisation.

Vous pouvez réaliser une certaine personnalisation en installant un fichier *.perldb* contenant du code d'initialisation. Par exemple, vous pourriez créer des alias ainsi (le dernier en est un que tout le monde s'attend à y voir) :

```
$DB::alias{'len'}  = 's/^len(.*)/p length($1)/';
$DB::alias{'stop'} = 's/^stop (at|in)/b/';
$DB::alias{'ps'}   = 's/^ps\b/p scalar /';
$DB::alias{'quit'} = 's/^quit(\s*)/exit/';
```

Vous pouvez changer les options de *.perldb* en utilisant des appels comme celui-ci :

```
parse_options("NonStop=1 LineInfo=db.out AutoTrace=1 frame=2");
```

Le code est exécuté dans le paquetage `DB`. Notez que *.perldb* est traité avant `PERLDB_OPTS`. Si *.perldb* définit le sous-programme `afterinit`, cette fonction est appelée après que l'initialisation du débogueur soit terminée. *.perldb* peut être contenu dans le répertoire courant, ou dans le répertoire home. Puisque ce fichier est utilisé par Perl et peut contenir des commandes arbitraires, pour des raisons de sécurité, il doit être la propriété du superutilisateur ou de l'utilisateur courant, et modifiable par personne d'autre que son propriétaire.

Si vous voulez modifier le débogueur, copiez et renommez *perl5db.pl* dans la bibliothèque Perl et bidouillez-le à coeur joie. Vous voudrez ensuite fixer votre variable d'environnement `PERL5DB` pour qu'elle dise quelque chose comme ceci :

```
BEGIN { require "myperl5db.pl" }
```

En dernier recours, vous pouvez aussi utiliser `PERL5DB` pour personnaliser le débogueur en modifiant directement des variables internes ou en appelant des fonctions du débogueur.

Notez que toute variable ou fonction n'étant pas documentée ici (ou dans *perldebguts*) est considérée réservée à un usage interne uniquement, et est sujette en tant que telle à des modifications sans préavis.

3.6 Support de Readline

Tel que Perl est distribué, le seul historique de ligne de commande fourni est simpliste, vérifiant la présence de points d'exclamation en début de ligne. Toutefois, si vous installez les modules Term::ReadKey et Term::ReadLine du CPAN, vous aurez des possibilités d'édition comparables à celle que fournit le programme *readline*(3) du projet GNU. Recherchez-les dans le répertoire *modules/by-module/Term* du CPAN. Ceux-ci ne supportent toutefois pas l'édition normale de ligne de commande de **vi**.

Une complétion rudimentaire de la ligne de commande est aussi disponible. Malheureusement, les noms des variables lexicales ne sont pas disponibles dans la complétion.

3.7 Support d'un Éditeur pour le Débogage

Si la version d'**emacs** de la FSF est installée sur votre système, il peut interagir avec le débogueur Perl pour fournir un environnement de développement intégré cousin de ses interactions avec les débogueurs C.

Perl est aussi fourni avec un fichier de démarrage pour faire agir **emacs** comme un éditeur dirigé par la syntaxe comprenant (en partie) la syntaxe de Perl. Regardez dans le répertoire *emacs* de la distribution des sources de Perl.

Une configuration similaire de Tom Christiansen pour l'interaction avec toute version de **vi** et du X window system est aussi disponible. Celle-ci fonctionne de façon similaire au support multifenêtré intégré que fournit **emacs**, où c'est le débogueur qui dirige l'interface. Toutefois, à l'heure où ces lignes sont rédigées, la localisation finale de cet outil dans la distribution de Perl est encore incertaine.

Les utilisateurs de **vi** devraient aussi s'intéresser à **vim** et **gvim**, les versions pour rongeurs et huisseries, pour colorier les mots-clés de Perl.

Notez que seul perl peut vraiment analyser du Perl, ce qui fait que tous ces outils d'aide à l'ingénierie logicielle manquent un peu leur but, en particulier si vous ne programmez pas en Perl comme le ferait un programmeur C.

3.8 Le Profileur Perl

Si vous désirez fournir un débogueur alternatif à Perl, invoquez juste votre script avec un deux points et un argument de paquetage donné au drapeau **-d**. Un des débogueurs alternatifs les plus populaires pour Perl est le profileur Perl. Devel::Prof est maintenant inclus dans la distribution standard de Perl. Pour profiler votre programme Perl contenu dans le fichier *mycode.pl*, tapez juste :

```
$ perl -d:DProf mycode.pl
```

Lorsque le script se terminera, le profileur écrira les informations de profil dans un fichier appelé *tmon.out*. Un outil tel que **dprofpp**, lui aussi fourni dans la distribution standard de Perl, peut être utilisé pour interpréter les informations de ce profil.

4 Débogage des expressions rationnelles

`use re 'debug'` vous permet de voir les détails sanglants sur la façon dont le moteur d'expressions rationnelles de Perl fonctionne. De façon à comprendre ces sorties typiquement volumineuses, on doit non seulement avoir une certaine idée de la façon dont les recherches d'expressions rationnelles fonctionnent en général, mais aussi connaître la façon dont Perl compile en interne ses expressions rationnelles pour en faire des automates. Ces sujets sont explorés en détails dans +Débogage des expressions rationnelles in *perldebguts*.

5 Débogage de l'usage de la mémoire

Perl contient un support interne du reporting de son propre usage de la mémoire, mais ceci est un concept plutôt avancé qui requiert une compréhension de la façon dont l'allocation mémoire fonctionne. Voir Débogage de l'utilisation de la mémoire par Perl in *perldebguts* pour les détails.

6 VOIR AUSSI

Vous avez essayé l'option **-w**, n'est-ce pas ?

Voir aussi *perldebguts*, *re*, *DB*, *Devel::Dprof*, *dprofpp*, *Dumpvalue* et *perlrun*.

7 BUGS

Vous ne pouvez pas obtenir d'informations concernant les frames de la pile ou de toute façon les fonctions de débogage qui n'ont pas été compilées par Perl, comme celles des extensions C ou C++.

Si vous modifiez vos arguments @_ dans un sous-programme (comme avec `shift` ou `pop`), la trace de la pile ne vous en montrera pas les valeurs originales.

Le débogueur ne fonctionne actuellement pas en conjonction avec l'option de ligne de commande **-W**, puisqu'elle n'est elle-même pas exempte d'avertissements.

Si vous êtes dans un appel système lent (comme `wait`, `accept`, ou `read` depuis le clavier ou une socket) et que vous n'avez pas mis en place votre propre handler `$SIG{INT}`, alors vous ne pourrez pas revenir au débogueur par CTRL-C, parce que le propre handler `$SIG{INT}` du débogueur ne sait pas qu'il doit lever une exception pour faire un longjmp(3) hors d'un appel système lent.

8 TRADUCTION

8.1 Version

Cette traduction française correspond à la version anglaise distribuée avec perl 5.6.0. Pour en savoir plus concernant ces traductions, consultez http://perl.enstimac.fr/.

8.2 Traducteur

Roland Trique <*roland.trique@uhb.fr*>

8.3 Relecture

Gérard Delafond

9 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.