Découvrez nos magazines :

- GNU/Linux Magazine
  - Dernier Numéro Paru
  - Tous les anciens Numéros
  - Abonnements : GNU/Linux Magazine
  - Toutes les offres d'abonnement avec ce titre
- GNU/Linux Magazine HORS-SÉRIE
  - Dernier Numéro Paru
  - Tous les anciens Numéros
  - Abonnements : GNU/Linux Magazine HS
  - Toutes les offres d'abonnement avec ce titre
- Linux Pratique
  - Dernier Numéro Paru
  - Tous les anciens Numéros
  - Abonnements : Linux Pratique
  - Toutes les offres d'abonnement avec ce titre
- Linux Pratique HORS-SÉRIE
  - Dernier Numéro Paru
  - Tous les anciens Numéros
  - Abonnements : Linux Pratique HS
  - Toutes les offres d'abonnement avec ce titre
- Linux Pratique Essentiel
  - Dernier Numéro Paru
  - Tous les anciens Numéros
  - Abonnements : Linux Pratique Essentiel
  - Toutes les offres d'abonnement avec ce titre

News

Les nouveautés de Perl 5.10 – première partie

27/02/2009

Posté par admin | Signature : Sébastien Aperghis-Tramoni
Tags : GLMF
0 Commentaire | Ajouter un commentaire

~~Retrouvez cet article dans : Linux Magazine 104~~

Après plusieurs années d'un long développement, Perl 5.10 est finalement sorti le mardi 18 décembre, le jour du vingtième anniversaire de la sortie de Perl 1.0. Regardons un peu tout ce qu'apporte cette nouvelle version majeure.

Pourquoi une si longue attente ?

Si les utilisateurs de Perl ont attendu plus longtemps que prévu cette nouvelle version, c'est déjà parce que comme pour la plupart des Logiciels libres, le plan est de publier " quand c'est prêt ".
Bien sûr, des cycles se mettent assez naturellement en place, et, dans le cas de Perl 5, il y avait une nouvelle version à peu près tous les ans ou tous les deux ans : les deux dernières majeures étaient la 5.6.0 en 2000 et la 5.8.0 en 2002. Bien sûr, il y avait des sous-versions, mais, même la dernière, la 5.8.8, date de février 2006.
Une des raisons est que Perl 5.8 est très stable, et permet déjà de compenser ses points faibles et d'ajouter de nouvelles fonctionnalités par l'utilisation de modules du CPAN. Il n'y avait donc pas une forte pression de la part des utilisateurs (et donc des sociétés) pour une nouvelle version majeure.
De plus, le développement de Perl 6 avait un temps jeté le trouble : y aurait-il de nouvelles versions de Perl 5, une fois Perl 6 disponible ?
La question a été tranchée de manière simple. En partant de la constatation que Perl 5 et Perl 6 sont deux projets totalement différents au niveau technique et architectural, il a été décidé que Perl 5 continuerait d'être maintenu et même amélioré tant qu'il y aurait des utilisateurs. Il s'est alors produit un déclic, et le développement est reparti de plus belle, en mettant l'accent sur la transition vers Perl 6 et, pour ce faire, sur l'ajout progressif de fonctionnalités du futur langage. Cela a au final allongé ce cycle de développement, mais le produit à l'arrivée est à la hauteur de l'attente.
Le nombre de nouveautés significatives pour les utilisateurs est d'ailleurs suffisamment conséquent pour que nous ayons préféré répartir leur description sur plus d'un article, afin que ceux-ci restent digestes ;-)

Ce qui a été supprimé

En premier lieu, certaines fonctionnalités anciennes ou expérimentales ont été supprimées, car elles étaient peu fonctionnelles, voire posaient problème. Comme la plupart n'étaient pas vraiment employées, et étaient marquées expérimentales ou obsolètes, cela ne pose pas de véritable problème de compatibilité.
Tout d'abord, les variables spéciales ~~$*~~ et ~~$#~~ ont été supprimées. La première était un vestige de Perl 4 équivalent à l'option ~~/m~~ des expressions régulières, la seconde une tentative d'émulation du

format OFMT de awk. Les pseudo-hashs ne sont plus supportés, ce qui a permis d'accélérer l'accès aux éléments des vrais hashs. Enfin, le compilateur Perl vers bytecode et C, ainsi que la passerelle Java-Perl JPL, ont été supprimés (ils ne fonctionnaient pas, et il existe des alternatives à JPL sur le CPAN, dont Inline::Java).

Syntaxe

Defined-or

La syntaxe de Perl 5.10 a été enrichie de plusieurs ajouts provenant de Perl 6. L'un des plus attendus est le fameux defined-or qui permet d'écrire correctement et de manière courte l'affectation d'une valeur par défaut :

```
$var = defined $input ? $input : $default;
$option = $default unless defined $option;
```

devient maintenant :

```
$var = $input // $default;
$option //= $default;
```

La double barre oblique // a été retenue comme symbole pour sa ressemblance avec la double barre verticale ||, ce qui permet de se souvenir facilement que cela fait pareil, sauf que cela marche dans tous les cas (puisque || ne sait pas différencier une valeur fausse d'une valeur non définie).

say

Autre ajout venu de Perl 6, la fonction say fonctionne comme print, mais en ajoutant un saut de ligne à la fin. C'est donc similaire à la fonction println() de Pascal et Java, mais en plus court, ce qui est utile pour les unilignes (et les adeptes du golf).

```
say "OH HAI";    # équivalent à :  print "OH HAI\n"
```

Opérations sur les fichiers

On peut noter deux petits ajouts en matière d'opérations sur les fichiers. D'une part, on peut maintenant empiler les tests comme -f,-s, ce qui permet de gagner en concision. Ainsi, pour vérifier qu'un chemin est un fichier accessible en écriture et exécution, on pourra écrire -f -w -x $file là où il fallait écrire -f $file && -w _ && -x _. D'autre part, les fonctions chdir(), chmod() et chown() fonctionnent maintenant sur les descripteurs de fichiers si votre système le permet.

Smart match

Encore une fonctionnalité importée de Perl 6, le smart match ou comparaison avancée. Derrière ce nouvel opérateur booléen ~~, se cache un mécanisme assez puissant, mais pas totalement facile à maîtriser, qui permet de comparer deux éléments de manière totalement générique. On ne peut guère que traduire et commenter la table de correspondance de perlsyn qui détaille son

fonctionnement en fonction du type de chaque opérande. Il faut noter à l'avance que cet opérateur est toujours commutatif (~~$a ~~ $b~~ est toujours équivalent à ~~$b ~~ $a~~) et que la première règle de la table qui s'applique détermine son comportement.

```
Type $a Type $b   Type de correspondance   Code équivalent
======= =======   =====================   ===============
code[1] code[1]   égalité des références   $a == $b
autre   code[1]   scalar sub truth         $b->($a)

hash    hash      clés de hash identiques  [sort keys %$a]~~[sort keys %$b]
hash    tableau   existence d'une tranche  grep {exists $a->{$_}} @$b
hash    regex     grep des clés du hash    grep /$b/, keys %$a
hash    autre     existence d'une entrée   exists $a->{$b}

tableau tableau   tableaux identiques[2]
tableau regex     grep du tableau          grep /$b/, @$a
tableau nombre    nombre dans tableau      grep $_ == $b, @$a
tableau autre     chaîne dans tableau      grep $_ eq $b, @$a

autre   undef     indéfini                 !defined $a
autre   regex     correspondance de motif  $a =~ /$b/
code()  code()    résultats identiques     $a->() eq $b->()
autre   code()    simple closure truth     $b->()  # $a est ignorée
nombre  nombre[3] égalité numérique        $a == $b
autre   chaîne    égalité de chaîne        $a eq $b
autre   nombre    égalité numérique        $a == $b

autre   autre     égalité de chaîne        $a eq $b
```

[1] Ce doit être une référence de code dont le prototype (s'il existe) n'est pas vide. Les fonctions avec un prototype vide sont traitées par les règles ~~code()~~.

[2] C'est-à-dire que chaque élément correspond à l'élément de même index dans l'autre tableau. Si une référence circulaire est détectée, on revient à l'égalité des références.

[3] Un vrai nombre ou une chaîne qui ressemble à un nombre.

Le code équivalent indiqué dans cette table n'est bien sûr là que pour illustrer le véritable code s'arrêtant dès que c'est possible.

On le voit, si certaines règles sont simples à comprendre, d'autres sont déjà plus subtiles. On voit néanmoins que certaines opérations qui avant demandaient pas mal de code vont s'en trouver simplifiées. Ainsi, pour vérifier si deux tableaux sont identiques :

```
say "ces tableaux sont ", \@a ~~ \@b ? "identiques" : "différents";
```

given .. when

Perl 5.10 propose une construction similaire au ~~switch~~ présent dans bien des langages, mais ici bien plus puissante, car, s'appuyant sur la comparaison avancée, elle permet des tests bien plus complexes que ceux possibles avec ~~switch~~. Plus exactement, cette construction se compose ainsi :

- ~~given (EXPR)~~ affecte la valeur de l'expression à ~~$_~~.
- ~~when (EXPR)~~ évalue l'expression comme une condition booléenne. La manière exacte d'évaluer cette condition dépend de ce que contient l'expression, mais, en général, ~~given($value)~~ est équivalent à ~~given($_ ~~ $value)~~. Il y a plusieurs exceptions indiquées dans ~~perlsyn~~, mais on peut résumer en disant que cela fonctionne généralement comme on s'y attend.
- ~~default~~ correspond comme on peut s'y attendre aux autres cas, si aucune condition ~~when~~ n'a

été vérifiée.

Pour simplifier l'utilisation, lorsqu'une condition ~~when~~ se vérifie, on sort du given en même temps que du bloc correspondant. C'est l'inverse du ~~switch~~ en C où il faut explicitement indiquer break en fin de chaque case. On peut toutefois contrôler plus finement le flux d'exécution avec les mots-clés ~~break~~ pour sortir du ~~given~~ et ~~continue~~ pour transférer l'exécution au ~~when~~ suivant. Bien sûr, les conditions ~~when~~ sont examinées dans l'ordre, et c'est la première qui se vérifie qui est exécutée.

Cela peut paraître complexe, mais c'est en fait assez naturel et on peut aussi bien reproduire le ~~switch~~ du C :

```
given ($number) {
        when (42) { say "La Réponse"   }
        when (56) { say “fer standard” }
    }
```

que le ~~case..esac~~ du Bourne shell (sauf qu'on peut utiliser des expressions régulières) :

```
use Regexp::Common qw/net/;

    given ($host) {
        when ("localhost"       ) { say "adresse locale" }
        when (/$RE{net}{IPv4}/  ) { say "adresse IPv4"   }
        when (/$RE{net}{domain}/) { say "FQDN"           }
        default { say “type d’argument inconnu” }
    }
```

Mais l'intérêt réside bien sûr dans l'exploitation du smart match, et du fait qu'on n'est nullement limité dans le type des arguments et des données de comparaison.

```
my @primes = (
        2,  3,  5,  7,  11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
        47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97
    );

    given ($value) {
        when ( undef ) {
            say “valeur non définie"
        }
        when ( 42 ) {
            say "La Réponse"
        }
        when ( $_ >= 0 && $_ < 100 ) {
            when ( @primes ) {
                say “petit nombre premier”
            }
            default {
               say “n’est pas un petit nombre premier”
            }
        }
        when ( /$RE{net}{IPv4}/ ) {
            say "adresse IPv4"
        }
        when ( \&is_leap_year) {
            say “$value est une année bissextile"
        }
        default {
            say "type de valeur non géré"
        }
    }
```

L'exemple est bien sûr un peu artificiel, mais il montre déjà une partie des possibilités de

comparaison de cette nouvelle construction, et illustre le fait qu'on peut même imbriquer les ~~when~~.

Il faut par contre noter que suite à des problèmes, pour partie, techniques, l'opérateur de comparaison avancé et ~~given .. when~~, bien qu'à la base inspirés par Perl 6, sont au final différents de leurs équivalents dans le futur langage, tant sur le plan de la syntaxe que sur le plan des sémantiques. Par exemple, le ~~ de Perl 5.10 est commutatif alors qu'en Perl 6 ce n'est pas le cas, et la gestion des tableaux est assez différente.

Plus ennuyeux, cela ne marche pour le moment qu'avec les données de base. Les objets, la surcharge et en général la magie ne fonctionnent pour le moment pas avec la comparaison avancée (et donc ~~given .. when~~), qui regarde actuellement la structure sous-jacente au lieu d'invoquer les appels corrects. Cela devrait normalement être corrigé dans la version 5.10.1.

Variables d'état

Pour répondre à certains besoins, une nouvelle classe de variables a été introduite : les variables d'état. Déclarées avec le mot-clé ~~state~~, elles sont comme les variables ~~my~~ seulement visibles dans leur portée lexicale, mais contrairement à ces dernières, leur valeur est persistante et se conserve d'un appel sur l'autre. Elles sont donc comparables aux variables ~~static~~ en C. Petit exemple trivial qui affiche les cinq premières puissances de 2 :

```
sub next_power {
        state $x = 1;   # n'est affecté qu'au premier appel
        return $x <<= 1
    }
    say next_power() for 1..5
```

Avant, on aurait utilisé une fermeture pour ce genre de code. Cela marchait bien (et cela marche toujours), mais ~~state~~ permet d'éviter certains coûts en mémoire et en complexité associés aux fermetures. Mais, cette fonctionnalité n'est pour le moment disponible que pour les variables scalaires. Le support pour ~~state @array~~ a été testé, mais n'était pas suffisamment finalisé pour 5.10.0. Il pourrait par contre apparaître en 5.10.1 ou 5.12.

Activation de ces nouveautés

Comme certaines de ces nouvelles fonctionnalités pourraient casser du code existant, elles sont désactivées par défaut, et peuvent être activées à la demande avec la pragma lexicale ~~feature~~. Les fonctionnalités ainsi contrôlables sont ~~say~~, ~~state~~ et ~~switch~~. Pour simplifier, on peut aussi utiliser le raccourci use feature ":5.10" qui indique de toujours autoriser tous les nouveaux ajouts de Perl 5.10. Et comme cela ne fonctionne qu'à partir de cette version du langage, ~~use 5.010~~ a le même effet.

Pour les unilignes, l'option ~~-E~~ est équivalente à ~~-e~~, mais active aussi les ajouts de Perl 5.10 :

```
perl -E 'say "OH HAI WORLD"'
```

$_ lexical

La variable ~~$_~~ peut maintenant être lexicalisée avec my, afin d'être sûr de limiter des modifications à cette variable au seul bloc lexical de travail.

Prototype _

Le prototype _ a été ajouté. Il est équivalent au prototype $, mais utilisera $_ si aucune variable n'est passée en argument. Pour cette raison, il ne peut être utilisé qu'en fin de prototype ou avant un point-virgule.

Blocs UNITCHECK

Les adeptes de vérifications poussées ou de code automodifiant disposent maintenant des blocs UNITCHECK, similaires aux blocs CHECK (ils sont exécutés en fin de la phase de compilation de l'unité qui les définit), mais qui sont exécutés durant les évaluations dynamiques, alors que les blocs CHECK ne l'étaient que durant la phase de compilation initiale.

Pragmas lexicales

Il est maintenant possible d'écrire facilement des pragmas lexicales, c'est-à-dire qui se comportent comme strict et warnings et limitent leurs effets aux blocs lexicaux d'où elles sont invoquées. Cela consiste principalement à utiliser le hash %^H pour stocker son état, et à le récupérer grâce à caller().

Rafaël Garcia-Suarez a écrit la pragma encoding::source à la fois à titre d'exemple, et aussi pour remplacer la pragma encoding jugée insatisfaisante. Cela permet de mélanger au sein d'un même fichier du texte écrit dans différents encodages :

```
use encoding::source 'utf8';
    # le code écrit ici est en UTF-8

    {
        use encoding::source 'latin-1';
        # mais celui dans ce bloc est en ISO-Latin-1
    }

    # et là on revient en UTF-8
```

À utiliser avec parcimonie, pour éviter de s'arracher les cheveux !

Voir aussi perlpragma et la documentation de caller() pour plus de détails concernant l'implémentation des pragmas lexicales.

Objets

Le modèle objet de Perl n'a pas été fondamentalement changé, et reste toujours aussi simple (voire simpliste), mais a été amélioré afin d'offrir un meilleur support pour les nouveaux modèles qui se sont développés ces deux dernières années. En effet, les systèmes d'objets inversés (inside-out objects), les rôles et autres méta-modèles se sont appuyés sur le modèle objet existant, au détriment de la performance. Perl 5.10 a intégré certains des mécanismes sous-jacents afin de simplifier le code de ces nouveaux modèles.

UNIVERSAL::DOES

La méthode globale ~~DOES()~~ a été ajoutée afin de pallier les problèmes de sémantique et d'utilisation de ~~UNIVERSAL::isa()~~, et de fournir un support plus général. Plus précisément, ~~isa()~~ a une sémantique de vérification d'héritage de classes, alors que ~~DOES()~~ (prévue pour être facilement surchargée) se contente de vérifier si un objet ou une classe fait quelque chose, ce qui est plus adapté pour établir des relations entre classes et rôles. On peut donc s'en servir aussi bien en remplacement de ~~isa()~~ pour vérifier qu'un objet hérite d'une classe donnée

```
if ( $object->DOES($parent_class) ) {
        ...
    }
```

que pour vérifier qu'une classe implémente bien un rôle donné :

```
if ( $class->DOES("Logger") ) {
        ...
    }
```

Hash::Util::FieldHash

Ce module a été écrit et intégré à Perl 5.10 afin d'offrir un meilleur support des objets inversés. En soi, ce module n'offre que peu d'intérêt hormis pour ceux qui veulent écrire un système objet avancé. En effet, les problèmes typiquement rencontrés sont la gestion des références aux véritables attributs, la libération de mémoire à la destruction de l'objet (comme les attributs ne sont plus dans l'objet, ils ne disparaissent plus avec lui) et le support des threads. ~~Hash::Util::FieldHash~~ fournit les fonctions permettant de répondre à ces problèmes.

À noter que le module ~~Hash::Util::FieldHash::Compat~~, disponible sur le CPAN, permet d'émuler cette nouvelle API sur les précédentes version de Perl.

Algorithme de résolution C3

D'abord, un peu de terminologie. C3 est un algorithme de résolution des méthodes ou MRO (method resolution order), c'est-à-dire un algorithme permettant de déterminer un ordre pour traverser la hiérarchie des classes afin de déterminer la bonne méthode à exécuter.

L'exemple classique est une hiérarchie de classe en losange, où la classe ~~D~~ hérite des classes ~~B~~ et ~~C~~ qui, elles-mêmes, héritent de la classe ~~A~~.

L'algorithme classique de résolution des méthodes en Perl effectue une recherche par profondeur (depth first search ou DFS), et renvoie donc les classes dans l'ordre (~~D, B, A, C~~). Le problème est que ~~A~~ apparaît avant ~~C~~ alors que cette dernière est une sous-classe de ~~A~~. C3 en revanche fournit un ordre plus correct (~~D, B, C, A~~).

L'algorithme C3 est une linéarisation conçue pour éviter qu'une classe n'apparaisse avant une de

ses sous-classes en préservant à tout moment une triple cohérence sur l'ordre de précédence locale, le graphe de précédence étendu et la monotonie (d'où le nom de C3). Originellement implémenté en Dylan, il a été adopté comme MRO par défaut dans Python depuis la version 2.3, et était disponible en Perl 5.8 avec le module ~~Class::C3~~. Directement intégré dans Perl 5.10, il est plus rapide et plus facilement contrôlable par la pragma ~~mro~~ qui permet de sélectionner la méthode de recherche :

```
use mro "dfs";  # utilise le MRO DFS pour cette classe
    use mro "c3";   # utilise le MRO C3 pour cette classe
```

À noter que pour raison de compatibilité arrière, DFS reste le MRO activé par défaut.

Optimisations

De manière moins visible pour l'utilisateur, Perl 5.10 a été l'objet d'un énorme chantier de nettoyage interne du code source, dont certaines parties ont maintenant 20 ans d'âge. En particulier, Nicholas Clark, pumpking de 5.8, a effectué un gros travail pour factoriser plusieurs structures internes (typeglobs, GV, CV, formats), permettant de gagner quelques octets par ci ou par là, démultipliés par leur fréquence d'utilisation. En conséquence, un même programme Perl doit consommer moins de mémoire avec Perl 5.10 qu'avec 5.8.

Durant ce travail de factorisation des structures, Nicholas en a profité pour modifier la manière dont sont gérées les constantes et elles sont maintenant plus légères (près de 400 octets gagnés) et un peu plus rapides. Le module ~~constant.pm~~ a de plus été modifié par votre serviteur et publié sur le CPAN pour qu'il soit fonctionnel sur Perl 5.5, ce qui permet de profiter de certaines fonctionnalités qui n'étaient disponibles que depuis 5.8.

Certains cas spéciaux de la fonction ~~sort()~~ ont été optimisés, par exemple ~~@a = sort @a~~ qui évite maintenant de créer un tableau intermédiaire. De même, ~~reverse sort ...~~ effectue maintenant le tri directement en ordre inverse. D'autres cas ont été pris en compte afin de prendre moins de mémoire et de les rendre un peu plus rapides. Il est par ailleurs possible d'utiliser une fonction récursive comme tri personnalisé pour ~~sort()~~.

Les opérations internes de gestion d'Unicode sont plus rapides. En particulier, le cache est maintenant plus efficace et plus souvent utilisé.
La création de références à des tableaux et listes vides est un cas spécial, maintenant plus rapide.

Sécurité

La sécurité n'est pas en reste avec quelques nouveautés destinées à nous protéger des problèmes actuels. Ainsi, la variable ~~$AUTOLOAD~~ supporte maintenant correctement le traçage des valeurs quand le taint mode est activé. De même, les fonctions ~~printf()~~ et ~~sprintf()~~ ont été modifiées pour rejeter tout argument reconnu comme non fiable.
En interne, Perl utilise maintenant les fameuses fonctions ~~strlcat()~~ et ~~strlcpy()~~ introduites par les développeurs d'OpenBSD, et toute l'API interne et externe a été revue afin de spécifier comme constantes toutes les chaînes qui pouvaient l'être. L'intérêt est de fournir plus de sémantique au compilateur afin qu'il puisse effectuer davantage de vérifications, ce qui permet de se prémunir de

certains problèmes. Cela facilite aussi la compilation sur les compilateurs les plus avancés, qui sont souvent plus tatillons.

Modules

Un grand nombre de modules font leur apparition dans la distribution standard (la majeure partie existent sur le CPAN depuis plusieurs années). Petite revue des plus intéressants.
Module::CoreList et sa commande corelist permettent de savoir dans quelle version de Perl un module a été introduit.
Archive::Tar permet, comme son nom l'indique, de créer et manipuler des archives au format Unix tar(1) en pur Perl, ce qui est utile sur les systèmes qui ne disposent pas de cette commande. Avec Compress::Zlib, qui est un module permettant de manipuler des flux gzip, on peut créer et extraire des distributions au format habituel.
Module::Pluggable est un module permettant de gérer facilement un système de plugins.
Pod::Simple est le nouveau moteur d'analyse du Pod, le format de documentation de Perl. Plusieurs des modules et outils pour transformer et afficher le Pod ont été modifiés pour l'utiliser à la place de leurs propres analyseurs.
encoding::warnings est une pragma lexicale qui produit des avertissements chaque fois qu'une chaîne de caractères ASCII contenant des caractères étendus est implicitement convertie en UTF-8.
La commande xsubpp, qui transforme les fichiers XS en C, a été factorisée en deux modules, ExtUtils::ParseXS et ExtUtils::CBuilder.
Module::Build est un nouvel environnement de construction des distributions CPAN, destiné à remplacer ExtUtils::MakeMaker qui dépend de l'utilitaire make(1). Étant en pur Perl et orienté objet, il a été conçu pour pallier un des gros problèmes de ExtUtils::MakeMaker, à savoir la difficulté pour le personnaliser.
CPANPLUS est une API permettant de manipuler les distributions et miroirs CPAN. Sur ce point, il constitue avec sa commande cpanp un remplaçant plus modulaire au traditionnel shell CPAN.pm, qui a lui-même subi de nombreuses améliorations. Mais, CPANPLUS permet d'ajouter plus facilement de nouvelles fonctionnalités comme l'interrogation des tickets ouverts liés à un module ou la transformation des distributions CPAN en paquets RPM ou Debian avec la commande cpan2dist.
Inversement, de nombreux modules qui n'étaient avant disponibles que dans la distribution standard sont maintenant aussi sur le CPAN : XSLoader, Sys::Syslog, threads, threads:shared, Exporter, File::Path, constant. Et bien sûr, tous les modules déjà en double vie comme CPAN.pm, Tie::HiRes, Math::BigInt, Math::Complex, Test::More, ExtUtils::MakeMaker, etc. ont été mis à jour avec leur plus récente version.

Autres améliorations

Dans les autres changements notables, on peut noter que Perl 5.10 intègre la base de données des caractères Unicode 5.0.0, et propose des messages de diagnostic plus clairs. En particulier, Perl essayera d'indiquer le nom de la variable dans le message "Use of uninitialized value" (" essayera ", car ce n'est pas toujours possible).

Le support pour les systèmes d'exploitation anciens ou exotiques a été amélioré, pour s'accorder avec leurs spécificités. Par exemple l'EBDIC, toujours utilisé sur les mainframes IBM (OS/390, z/OS, etc.). Perl s'intègre aussi mieux aux nouveaux environnements OpenVMS et sait tenir compte des nouvelles fonctionnalités de la CRTL et d'ODS-5 (le système de fichiers d'OpenVMS).
Windows n'est pas en reste avec entre autres le support des noms Unicode quand c'est possible (sur les partitions NTFS) et une variable ~~${^WIN32_SLOPPY_STAT}~~ qui permet d'accélérer notablement la fonction ~~stat()~~. C'est très utile pour Cygwin où les accès au système de fichiers sont encore plus lents que les accès natifs Win32.

Conclusion

Malgré ce long inventaire à la Prévert, nous n'avons abordé qu'une partie des changements de Perl 5.10, ceux considérés comme les plus importants ou intéressants. Le lecteur curieux est invité à lire ~~perl5100delta~~ pour une description plus exhaustive et détaillée. Le prochain article sera consacré au gros morceau de Perl 5.10 : les évolutions des expressions régulières.

~~Retrouvez cet article dans : Linux Magazine 104~~

Laissez une réponse

Vous devez avoir ouvert une session pour écrire un commentaire.

Identifiez-vous
Inscription
S'abonner à UNIX Garden
Catégories
Administration réseau
Administration système
Agenda-Interview
Audio-vidéo
Bureautique
Comprendre
Distribution
Embarqué
Environnement de bureau
Graphisme
Jeux
Matériel
News
Programmation
Réfléchir
Sécurité
Utilitaires
Web

- **Pages**
  - A propos
  - Nuage de Tags
  - Contribuer

Archives

- avril 2011
- mars 2011
- février 2011
- janvier 2011
- décembre 2010
- novembre 2010
- octobre 2010
- septembre 2010
- août 2010
- juillet 2010
- juin 2010
- mai 2010
- avril 2010
- mars 2010
- février 2010
- janvier 2010
- décembre 2009
- novembre 2009
- octobre 2009
- septembre 2009
- août 2009
- juillet 2009
- juin 2009
- mai 2009
- avril 2009
- mars 2009
- février 2009
- janvier 2009
- décembre 2008
- novembre 2008
- octobre 2008
- septembre 2008
- août 2008
- juillet 2008
- juin 2008
- mai 2008
- avril 2008
- mars 2008
- février 2008
- janvier 2008
- décembre 2007
- novembre 2007
- février 2007

GNU/Linux Magazine
GNU/Linux Magazine HS
Linux Pratique
Linux Pratique HS
Linux Pratique Essentiel
Linux Pratique Essentiel HS
Misc

Misc HS

Articles secondaires

- 15/3/2009
  Smart Middle Click 0.5.1 : ouvrez les liens JavaScript dans des onglets

  Tout d'abord, un petit raccourci utile : quand vous voulez ouvrir un lien dans un onglet, plutôt que d'utiliser le menu contextuel, cliquez simplement dessus avec le bouton du milieu. Hop, c'est ouvert ! C'est simple et diablement efficace, parfois un peu trop.....

  Voir l'article...
  30/10/2008
  Google Gears : les services de Google offline

  Lancé à l'occasion du Google Developer Day 2007 (le 31 mai dernier), Google Gears est une extension open source pour Firefox et Internet Explorer permettant de continuer à accéder à des services et applications Google, même si l'on est déconnecté....

  Voir l'article...
  7/8/2008
  Trois questions à...

  Alexis Nikichine, développeur chez IDM, la société qui a conçu l'interface et le moteur de recherche de l'EHM....

  Voir l'article...
  11/7/2008
  Protéger une page avec un mot de passe

  En général, le problème n'est pas de protéger une page, mais de protéger le répertoire qui la contient. Avec Apache, vous pouvez mettre un fichier ~~.htaccess~~ dans le répertoire à protéger....

  Voir l'article...
  6/7/2008
  hypermail : Conversion mbox vers HTML

  Comment conserver tous vos échanges de mails, ou du moins, tous vos mails reçus depuis des années ? mbox, maildir, texte... les formats ne manquent pas. ...

  Voir l'article...
  6/7/2008
  iozone3 : Benchmark de disque

  En fonction de l'utilisation de votre système, et dans bien des cas, les performances des disques et des systèmes de fichiers sont très importantes....

  Voir l'article...