perlport

Table des matières

1 NAME/NOM

perlport - Écrire du code Perl portable

2 DESCRIPTION

Perl fonctionne sur une grande variété de système d'exploitation. Alors que la plupart d'entre eux ont beaucoup en commun, il ont aussi des fonctionnalités qui leur sont très spécifiques et uniques.

Ce document tente de vous aider à trouver ce qui constitue un code Perl portable, de manière que si vous avez décidé d'écrire du code portable, vous sachiez où sont les limites et ainsi ne pas les franchir.

C'est un compromis entre tirer pleinement avantage d'**un** type de machine particulier, et tirer avantage d'une **quantité** d'entre eux. Bien sûr, plus votre quantité est large (et ainsi plus variée), plus le nombre de dénominateurs commun chute, vous laissant un choix plus restreint de solution pour accomplir une tâche précise. Il est donc important, avant d'attaquer un problème, de considérer l'enveloppe des compris que vous êtes prêt a accepter. Précisément, s'il y est important pour vous que la tâche nécessite une portabilité totale, ou s'il est seulement important qu'elle soit réalisée. C'est le choix le plus difficile à faire. Le reste est facile, parce que Perl fournis un très grand nombre de possibilité quelque soit l'approche que vous avez de votre problème.

D'un autre sens, écrire du code portable va inévitablement réduire les choix disponibles. C'est une discipline a acquérir.

Faites attention aux points importants:

Tous les programmes Perl n'ont pas besoin d'être portable

Il n'y a aucune raison de ne pas utiliser Perl comme un langage pour enchaîner l'exécution d'outil Unix, ou pour prototyper une application Macintosh, ou pour gérer la base de registre de Windows. Si la portabilité d'un programme n'a pas de sens, inutile de s'en préoccuper.

L'immense majorité de Perl est portable

N'allez pas penser qu'il est difficile d'écrire du code Perl portable. Ce n'est pas le cas. Perl essaye de réduire de lui-même les différences entre les différentes plateformes, et de fournir tout les moyens nécessaires pour utiliser ces fonctionnalités. Dès lors presque tous les scripts Perl fonctionne sans aucune modification. Mais il *existe* quelques spécificités dans l'écriture de code portable, et ce document est entièrement consacré à ces spécificités.

Voici la règle générale: Lorsque vous entamez une tache qui est communément réalisées sur des plateformes différente, pensez à écrire du code portable. De cette manière, vous ne vous limitez pas trop dans le nombre de solutions possible, et dans le même temps, vous ne limitez pas l'utilisateur à un trop petit choix de plateforme. D'un autre côté, lorsque vous devez utilisez des particularités d'une plateforme spécifique, comme c'est souvent le cas pour la programmation système (Quelque soit le type de plateforme Unix, Windows, Mac OS, VMS, etc.), écrivez du code spécifique.

Lorsqu'un script doit tourner que sur deux ou trois système d'exploitation, vous n'avez qu'à considérer les différences entre ces systèmes particuliers. L'important est de décidez sur quoi le script doit fonctionner et d'être clair dans votre décision.

Le document est séparé en trois sections principales: les problèmes généraux de la portabilité (PROBLÈMES (§3), les spécificités par plateforme (PLATEFORMES (§5), et les différences de comportement des fonctions intégrées de perl suivant les plateformes (IMPLÉMENTATION DES FONCTIONS (§6).

Ces informations ne doivent pas être considérés comme exhaustives; elles peuvent décrire des comportements transitoires de certain portage, elles sont presque toutes en évolutions permanente. Elle doivent donc être considérés comme étant en cours de modification perpétuelles. (<IMG SRC="yellow_sign.gif" ALT="Under Construction">).

3 PROBLÈMES

3.1 Les retours chariots

Dans la plupart des systèmes d'exploitations, les lignes des fichiers sont séparées par des retours chariots. Mais la composition de ces retour chariots diffèrent d'OS en OS. Unix utilise traditionnellement `\012`, un type d'entrée/sortie de Windows utilise `\015\012`, et Mac OS utilise `\015`.

Perl utilise `\n` pour représenter le retour chariot "logique", où ce qui est logique peut dépendre de la plateforme utilisée. Avec MacPerl, `\n` correspond toujours à `\015`. Pour les Perl DOS, il correspond habituellement à `\012`, mais lors d'un accès à un fichier en mode "texte", STDIO le converti vers (ou à partir de) `\015\012`.

A cause de la conversion "texte", Perl sous DOS ont des limitations à l'utilisation de `seek` et `tell` lorsqu'un fichier est utilisé en mode "texte". Spécifiquement, si vous n'utilisez que des emplacements obtenus avec `tell` (et rien d'autre) pour `seek`, vous pouvez l'utiliser sans problèmes en mode "texte". En général utiliser `seek` ou `tell` ou quelques opérations portant sur un nombre d'octet plutôt qu'un nombre de caractères, sans faire attention à la taille du `\n`, peut ne pas être portable. Si vous utilisez le mode `binmode`, ces limitations n'existent pas.

Une erreur de conception commune dans la programmation des sockets est de penser que `\n` eq `\012` partout. Lors de l'utilisation de protocoles tel que les protocoles Internet `\012` et `\015` doivent être spécifié, et les valeurs de `\n` et `\r` ne sont fiables.

```
print SOCKET "Hi there, client!\r\n";      # MAUVAIS
print SOCKET "Hi there, client!\015\012"; # BON
```

[NOTE: ceci ne concerne pas nécessairement les communications qui sont filtrées par un autre programme ou module avant d'être envoyées sur la socket; le serveur web EBCDIC le plus populaire, par exemple, accepte \r\n, et les convertis ainsi que tous les caractères d'un flux texte, de l'EBCDIC à l'ASCII.]

Quoi qu'il en soit, utiliser \015\012 (ou \cM\cJ, ou \x0D\x0A) peu vite devenir fatigant et laid, et peut même induire en erreur ceux qui maintiennent le code. Le module Socket fournit la Bonne Chose à faire pour ceux qui le veulent.

```
use Socket qw(:DEFAULT :crlf);
print SOCKET "Hi there, client!$CRLF"      # BON
```

En lisant *à partir* d'une socket, rappelez-vous que le séparateur par défaut ($/) est \n, mais un code comme celui-ci devrait reconnaître $/ comme \012 ou \015\012:

```
while (<SOCKET>) {
    # ...
}
```

Meilleur:

```
use Socket qw(:DEFAULT :crlf);
local($/) = LF;      # pas nécessaire si $/ est déjà \012

while (<SOCKET>) {
    s/$CR?$LF/\n/;   # il n'est pas sur que la socket utilise LF ou CRLF, OK
#   s/\015?\012/\n/; # La même chose
}
```

Cet exemple est le meilleur que le précédent même sous Unix, car maintenant les \015 et (\cM sont supprimés (et c'est tant mieux).

3.2 Stockage et taille des nombres

Différent CPUs stocke les entiers et les flottants dans des ordres différents(en anglais *endianness*) et des tailles différentes (32-bit et 64-bit sont les plus courant). Ceci affecte vos programmes s'ils essayent de transférer des nombres en format binaire d'une architecture CPU vers une autre via certain canaux: que ce soit en 'live' via une connexion réseau ou en les stockant dans un fichier.

Un conflit d'ordre de stockage met le désordre dans les nombres: Si un hôte little-endian (Intel, Alpha) stocke 0x12345678 (305419896 en décimal), un hôte big-endian (Motorola, MIPS, Sparc, PA) le lira comme étant 0x78563412 (2018915346 en décimal). Pour éviter ce problème lors de communication réseau (socket) utilisez les formats "n" et "N" des fonctions pack() et unpack(), l'ordre "réseau", car il sont garantie d'être portable.

Des tailles différentes provoquera des troncature même entre des plateforme de même ordre: la plateforme de taille plus courte perdra la partie haute du nombre. Il n'y a pas de bonne solution pour ce problème sauf éviter de transférer ou stocker des nombres en format binaire.

Vous pouvez circonvenir ces deux problèmes de deux façons: Soit en transférant et stockant les nombres en format texte, et non pas en binaire, ou en utilisant des modules tel que Data::Dumper (inclus dans la distribution standard depuis Perl 5.005) et Storable.

3.3 Fichiers

La plupart des plateformes de nos jours stockent les fichiers d'une manière hiérarchique. Alors, il est raisonnable de penser que n'importe quelle plateforme supporte une notion de "chemin" pour identifier de manière unique un fichier sur le système. La seule chose qui change est l'écriture de ce chemin.

Quoique similaire, les spécifications de chemin diffèrent entre Unix, Windows, Mac OS, OS/2, VMS, RISC OS et probablement les autres. Unix, par exemple, est l'un des rares OS a avoir une idée d'un répertoire racine unique.

VMS, Windows, et OS/2 peuvent fonctionner comme Unix en utilisant / comme séparateur de chemin, ou de leur propre façon (tel qu'avoir plusieurs répertoires racines et des fichiers périphériques non-rattachés à une racine tel que NIL: et LPT:).

<Mac OS> utilise : comme séparateur à la place de /.

Le perl RISC OSpeut émuler les noms de fichiers Unix avec / comme séparateur, ou utiliser le séparateur natif . pour les chemin et : pour le système de complétion de signal et les noms de disque.

Comme pour le problème des retours chariot, il existe des modules qui peuvent aider. Le module File::Spec fournit les méthodes permettant de faire la Bonne Chose quelque soit la plateforme sur laquelle le programme fonctionne.

```
use File::Spec;
chdir(File::Spec->updir());        # Remonter au répertoire parent
$file = File::Spec->catfile(
    File::Spec->curdir(), 'temp', 'file.txt'
);
# sur Unix et Win32, './temp/file.txt'
# sur Mac OS, ':temp:file.txt'
```

File::Spec est disponible dans la distribution standard, depuis la version 5.004_05.

En général, un programme en production ne doit pas contenir de chemin codé en dur; les demander à l'utilisateur ou les obtenir d'un fichier de configuration est meilleur, en gardant à l'esprit que la syntaxe des chemins varie d'une machine à l'autre.

Ceci est particulièrement visible dans des scripts tels que Makefiles et suites de tests qui supposent souvent que / est le séparateur pour les sous-répertoire.

Toujours dans la distribution standard File::Basename peut aussi être utile en décomposant un chemin en morceau (nom du fichier, chemin complet et suffixe du fichier).

Même sur une plateforme unique (si l'on peut considérer Unix comme étant une plateforme unique), rappelez-vous de ne pas compter sur l'existence ou le contenu de fichier spécifique au système, tels que */etc/passwd*, */etc/sendmail.conf*, ou */etc/resolv.conf*. Par exemple */etc/passwd* peut exister mais ne pas contenir les mots de passe encryptés parce que le système utilise une forme de sécurité plus performante – ou il peut ne pas contenir tous les comptes si le système utilise NIS. Si le code doit reposer sur de tels fichiers, incluez une description des fichiers et de leur format dans la documentation, et faites en sorte que l'utilisateur puisse facilement modifier l'emplacement par défaut du fichier.

N'ayez pas deux fichiers portant le même nom dans des casses différentes, comme *test.pl* et <Test.pl>, car de nombreuses plateformes sont insensible à la casse pour les noms de fichier. De plus, essayez de ne pas avoir de caractères non alphanumérique (excepté par .) dans les noms de fichier, et conservez la convention 8.3, pour une portabilité maximale.

De même, si vous utilisez AutoSplit, essayez de conserver ces convention pour la fonction split ; ou, au minimum, faites en sorte que le fichier résultat est un nom unique(indépendamment de la casse) pour les 8 premiers caractères.

Ne présumez pas que < ne sera jamais le premier caractère d'un nom de fichier. Utilisez toujours > explicitement pour ouvrir un fichier en lecture.

```
open(FILE, "<$existing_file") or die $!;
```

3.4 Interaction avec le système

Toutes les plateformes ne fournissent pas forcément la notion de ligne de commande. Ce sont généralement des plateforme qui repose sur une interface graphique (GUI) pour les interactions avec l'utilisateur. Un programme nécessitant une ligne de commande n'est donc pas forcément portable partout. Mais c'est probablement à l'utilisateur de s'arranger avec ça.

Certaines plateformes ne peuvent détruire ou renommer des fichiers ouverts par le système. Pensez donc à toujours fermer (close) le fichiers lorsque vous en avez terminé avec eux. N'effacez pas (unlink) ou ne renommez pas (rename) un fichier ouvert. N'utilisez pas tie ou open sur un fichier déjà ouvert ou connecté; utilisez d'abord untie ou close.

N'ouvrez jamais un même fichier plus d'une fois en même temps en écriture, car certains systèmes d'exploitation posent des verrous sur de tels fichiers.

Ne comptez pas sur la présence d'une variable d'environnement spécifique dans %ENV. Ne vous attendez pas à ce que les entrées de %ENV soit sensible à la casse, où même ne respecte correctement celle-ci.

Ne comptez pas sur les signaux.

Ne comptez pas sur la complétion des noms de fichier. Utilisez opendir, readdir, et closedir à la place.

Ne comptez pas sur une copie privée des variables d'environnement par programme, ni même sur un répertoire courant différent par programmes.

3.5 Communication inter-processus (IPC)

En général, n'accédez jamais directement au système dans un code qui se doit d'être portable. Ça signifie pas de `system`, `exec`, `fork`, `pipe`, `` ` ``, `qx//`, `open` avec un |, ni aucune des autres choses qui font qu'être un hacker perl sous Unix vaux le coup.

Les commandes qui lance des processus externes sont en général supportées par la plupart des plateformes (même si beaucoup d'entre elle ne supporte pas d'exécution simultanée (fork en anglais)), mais le problèmes avec elles c'est ce que l'on exécute grâce à elles. Les outils ont très souvent des noms différents suivant les plateformes, ou ne sont pas au même endroit, et en général affiche leur résultat différemment suivant la plateforme. Vous pouvez donc rarement comptez sur eux pour produire un résultat consistent.

Un exemple courant de code Perl est d'ouvrir un tube vers sendmail:

```
open(MAIL, '|/usr/lib/sendmail -t') or die $!;
```

Ceci est correct lorsque l'on sait que sendmail sera disponible. Mais ceci n'est pas valable sur beaucoup de systèmes non-Unix, et même sous certains Unix qui n'aurait pas sendmail installé. Si vous recherchez une solution portable, jetez un oeil aux modules `Mail::Send` et `Mail::Mailer` de la distribution `MailTools`. `Mail::Mailer` fournit plusieurs méthodes pour poster du courrier, incluant mail, sendmail et une connexion directe SMTP (via `Net::SMTP`) si votre système ne comporte pas de transporteur de courrier (MTA en anglais).

La règle à retenir pour du code portable est : Faites-le entièrement en Perl portable, ou utilisez un module (qui peut être implémenter en utilisant du code plateforme-dépendant, mais qui présente une interface commune).

Les IPC Systeme V (`msg*(), sem*(), shm*()`) ne sont pas toujours disponibles, même pas sur toutes les plateformes Unix.

3.6 Sous-programme externe (XS)

Le code XS peut, en général, être écrit de manière à fonctionner sur n'importe quelle plateforme; mais alors il ne doit pas utiliser des librairies, des fichier d'inclusion (header), etc., dépendants, ou non disponible sur toutes les plateformes, le code XS peut lui-même être spécifique à une plateforme, au même titre que du code Perl. Si les librairies et les headers sont portables, il est alors raisonnable de s'assurer que le code XS est portable aussi.

Plusieurs problèmes peuvent surgir lors de l'écriture de code XS portable: la disponibilité d'un compilateur C sur le système de l'utilisateur final. Le C porte en lui ses propres problèmes de portabilité et écrire du code XS vous expose à certains d'entre eux. Écrire en perl pure est comparativement un moyen plus simple d'assurer la portabilité.

3.7 Modules Standard

En général, les modules standard fonctionnent sur toutes les plateformes. L'exception notable est `CPAN.pm` (qui utilise actuellement des programmes externes qui peuvent ne pas être disponible, les modules spécifiques à une plateforme (tels que `ExtUtils::MM_VMS`), et les modules DBM.

Il n'y a aucun module DBM qui soit disponible sur toutes les plateformes. `SDBM_File` et les autres sont en général disponible sur tous les Unix et les portages DOS, mais pas avec MacPerl, ou seul `NBDM_File` et `DB_File` sont disponibles.

La bonne nouvelle est qu'au moins un module DBM doit être disponible, et que le module `AnyDBM_File` utilisera le module DBM qu'il pourra trouver. Bien sûr, le code doit être plus strict, retombant sur un dénominateur commun (par ex., ne pas excéder 1K pour chaque enregistrement).

3.8 Date et Heure

Les notions de l'heure du jour et de la date calendaire du système sont gérés de manière très différente. Ne pensez pas que la zone horaire soit stocké dans `$ENV{TZ}`, et même si c'est le cas, ne pensez pas que vous pouvez contrôler la zone horaire grâce à cette variable.

Ne pensez pas que le temps commence le 1er Janvier 1970 à 00:00:00, car ceci est spécifique à un système d'exploitation. Le mieux est de stocker les dates dans un format non ambigu. La norme ISO 8601 défini le format de la date: AAAA-MM-JJ. Une représentation texte (comme `1 Jan 1970`) peut facilement être converti en une valeur spécifique à l'OS à l'aide d'un module comme `Date::Parse`. Un tableau de valeur, tel que celui retourné par `localtime`, peut facilement être converti en une représentation spécifique à l'OS en utilisant `Time::Local`.

3.9 Jeux de caractère et encodage des caractères.

Présumez que très peu de chose sur les jeux de caractères. Ne présumez rien du tout sur les valeurs numériques (`ord()`, `chr()`) des caractères. Ne présumez pas que les caractères alphabétiques sont encodé de manière continue (au sens numérique du terme). Ne présumez rien sur l'ordre des caractères. Les minuscules peuvent être avant ou après les majuscules, elles peuvent être enlacé de telle sorte que le a et le A soit avant le b, les caractères accentués peuvent même être placée de telle sorte que ä soit avant le b.

3.10 Internationalisation

Si vous vous reposez sur POSIX (une supposition plutôt large, en pratique çà signifie UNIX) vous pouvez lire plus à propos du système de locale POSIX dans *perllocale*. Le système de locale tente de rendre les choses un petit peu plus portable ou au moins plus pratique et facile d'accès pour des utilisateur non anglophone. Le système concerne les jeux de caractères et leur encodage, les formats des dates et des temps, entre autres choses.

3.11 Ressources Système

Si votre code est destiné à des systèmes ayant peu ou pas de système de mémoire virtuel, vous devrez alors être *particulièrement* attentif à éviter des constructions telles que:

```
# NOTE: Ceci n'est plus "mauvais" en perl5.005
for (0..10000000) {}                        # mauvais
for (my $x = 0; $x <= 10000000; ++$x) {}    # bon

@lines = <VERY_LARGE_FILE>;                 # mauvais

while (<FILE>) {$file .= $_}                # parfois mauvais
$file = join('', <FILE>);                   # meilleur
```

Les deux dernières apparaissent moins intuitives pour la plupart des gens. La première méthode agrandi répétivement une chaîne, tandis que la seconde alloue un large bloc de mémoire en une seule fois. Sur certain système, cette dernière méthode est plus efficace que l'autre.

3.12 Sécurité

La plupart des plateformes multi-utilisateurs fournissent des niveaux de sécurité de base reposant sur le système de fichier. Certaine ne le font pas (Malheureusement). Dès lors les notions d'identifiant utilisateur, de répertoire de "base", et même l'état d'être connecté, ne sont pas reconnu par un grand nombre de plateforme. Si vous écrivez des programmes dépendant le la sécurité, il est très utilise de connaître le type de systèmes qui sera utilisé et d'écrire du code spécifiquement pour cette plateforme (ou ce type de plateforme).

3.13 Style

S'il est nécessaire d'avoir du code spécifique à un plateforme, essayez de regrouper tout ce qui est spécifique en un seul endroit, rendant le portage plus facile. Utilisez le module `Config` et la variable spéciale `$^O` pour différencier les plateformes, comme d'écrit dans PLATEFORMES (§5).

4 Testeurs du CPAN

Les modules disponible sur le CPAN sont testés par un groupe de volontaires sur des plateformes différentes. Ces testeurs sont prévenus par mail de chaque nouveau téléchargement, et répondent sur la liste par PASS, FAIL, NA (Non applicable à cette plateforme ou UNKNOWN (inconnu), accompagné d'éventuel commentaires.

Le but de ce test est double: premièrement, aider les développeurs à supprimer des problèmes de leur code qui n'était pas apparu par manque de test sur d'autres plateformes; deuxièmement fournir aux utilisateurs des informations sur le fonctionnement ou non d'un module donné sur une plateforme donnée.

Mailing list: cpan-testers@perl.org

Résultats des tests: `http://www.connect.net/gbarr/cpan-test/`

5 PLATEFORMES

Depuis la version 5.002, Perl fournit une variable `$^O` qui indique le système d'exploitation sur lequel il a été compilé. Elle fut implémentée pour aider à accélérer du code qui autrement aurait du utiliser `use Config;` et la valeur de `$Config{'osname'}`. Bien sûr, pour obtenir des informations détaillées sur le système, regarder `%Config` est très recommandé.

5.1 Unix

Perl fonctionne sur une grande majorité d'Unix et d'Unix-like (par exemple regardez la plupart des fichiers du répertoire *hints/* du code source). Sur la plupart de ces système, la valeur de `$^O` (comme `$Config{'osname'}`) est déterminée en retirant la ponctuation et mettant en minuscule le premier champs de la chaîne retournée en tapant `uname -a` (ou une commande similaire) au prompt. Voici par exemple pour les Unix les plus populaires:

```
uname         $^O         $Config{'archname'}
--------------------------------------------
AIX           aix         aix
FreeBSD       freebsd     freebsd-i386
Linux         linux       i386-linux
HP-UX         hpux        PA-RISC1.1
IRIX          irix        irix
OSF1          dec_osf     alpha-dec_osf
SunOS         solaris     sun4-solaris
SunOS         solaris     i86pc-solaris
SunOS4        sunos       sun4-sunos
```

Notez que parce que `$Config{'archname'}` peut dépendre du type d'architecture matérielle elle peut varier beaucoup, bien plus que `$^O`.

5.2 DOS dérivés

Perl a été porté sur les PC tournant sous des systèmes tels que PC-DOS, MS-DOS, OS/2, et la plupart des plateformes Windows que vous pourriez imaginer (à l'exception de Windows CE, si vous le comptiez comme tel). Les utilisateurs familier du style de shell *COMMAND.COM* et/ou *CMD.EXE* doivent être conscient que chacune de ces spécifications de fichier possèdent de subtiles différences:

```
$filespec0 = "c:/foo/bar/file.txt";
$filespec1 = "c:\\foo\\bar\\file.txt";
$filespec2 = 'c:\foo\bar\file.txt';
$filespec3 = 'c:\\foo\\bar\\file.txt';
```

Les appels systèmes peuvent accepter indifféremment / ou \ comme séparateur de chemin. Mais attention, beaucoup d'utilitaire en ligne de commande du type DOS traite / comme un préfixe d'option, ils peuvent donc perturbés par des noms de fichiers contenant /. A part lors des appels de programmes externes, / fonctionne parfaitement, et probablement mieux, car il est plus consistent avec l'usage populaire, et évite d'avoir à se souvenir ce qui doit être protégé par / et ce qui ne doit pas l'être.

Le système de fichier FAT de DOS ne peut utiliser des noms de fichiers "8.3". Sous les système de fichiers "insensibles à la casse, mais préservant la casse" HPFS (OS/2) et NTFS (NT) vous devez faire attention à la casse renvoyée par des fonctions comme `readdir` ou utilisée par des fonctions telles que `open` ou `opendir`.

DOS traite aussi quelques noms de fichiers comme étant spéciaux, tels que AUX, PRN, NUL, CON, COM1, LPT1, LPT2 etc. Malheureusement ces noms de fichiers ne fonctionneront pas même si vous y préfixer un chemin absolu. Il vaut donc mieux éviter de tels noms, si vous voulez que votre code soit portable sous DOS et ses dérivés.

Les utilisateurs de ces systèmes d'exploitations peuvent aussi vouloir utiliser des scripts tels que *pl2bat.bat* ou *pl2cmd* pour envelopper leurs scripts.

Les retours chariots (`\n`) sont convertis en `\015\012` par STDIO lors de l'écriture et la lecture des fichiers. `binmode(FILEHANDLE)` conservera `\n` comme `\n` pour ce handle. Des lors que c'est une opération sans effet sur les autres systèmes, `binmode` devrait être utilisé pour les programmes cross-plateforme qui travaillent sur des données binaires.

Voici les valeurs des variables `$^O` et `$Config{'archname'}` pour les différentes versions de DOS perl:

```
OS            $^O        $Config{'archname'}
--------------------------------------------
MS-DOS        dos
PC-DOS        dos
OS/2          os2
Windows 95    MSWin32    MSWin32-x86
Windows NT    MSWin32    MSWin32-x86
Windows NT    MSWin32    MSWin32-alpha
Windows NT    MSWin32    MSWin32-ppc
```

Consultez aussi:

L'environnent djgpp pour DOS, `http://www.delorie.com/djgpp/`
L'environnent EMX pour DOS, OS/2, etc. `emx@iaehv.nl`, `http://www.juge.com/bbs/Hobb.19.html`
Instruction pour la compilation sur Win32, *perlwin32*.
The ActiveState Pages, `http://www.activestate.com/`

5.3 Mac OS

N'importe quel module utilisant la compilation XS, sont hors de portée de la plupart des gens, car MacPerl est construit en utilisant un compilateur payant (et pas qu'un peu !). Quelques modules XS pouvant fonctionner avec MacPerl sont fournis en package binaire sur le CPAN. Consultez *MacPerl: Power and Ease* et Testeurs du CPAN (§4) pour plus de détails.
Les répertoires sont spécifiés de la manière suivante:

```
volume:répertoire:fichier          chemin absolu
volume:répertoire:                 chemin absolu
:répertoire:fichier                chemin relatif
:répertoire:                       chemin relatif
:fichier                           chemin relatif
fichier                            chemin relatif
```

Les fichiers d'un répertoire sont stockés dans l'ordre alphabétique. Les noms de fichiers sont limités à 31 caractères différent de `:`, qui est réservé comme séparateur de chemin.
Utilisez `FSpSetFLock` et `FSpRstFLock` du module `Mac::Files` à la place de `flock`.
Les application MacPerl ne peuvent être lancées en ligne de commande; pour les programme s'attendant à recevoir des paramètres dans `@ARGV` peuvent utiliser le code suivant qui ouvre une boite de dialogue demandant les arguments.

```
if (!@ARGV) {
    @ARGV = split /\s+/, MacPerl::Ask('Arguments?');
}
```

Si un script MacPerl est lancé grâce à du Drag'n'drop, `@ARGV` contiendra les noms de fichiers lâchés sur le script.
Les utilisateurs de Mac peuvent utiliser les programmes sous une sorte de ligne de commande grâce à MPW (Macintosh Programmer's Workshop, un environnement de développement libre d'Apple). MacPerl à d'abord été introduit comme un outil MPW, et MPW peut être utilisé comme un shell:

```
perl myscript.plx des arguments
```

ToolServer est une autre application d'Apple qui fournit un accès aux outil MPW et aux application MacPerl, permettant aux programmes MacPerl d'utiliser `system`, `` ` ``, et `open` avec un tube.
"Mac OS" est le nom propre du système d'exploitation, mais la valeur de `$^O` est "MacOS". Pour déterminer l'architecture, la version, ou soit la version de l'application ou de l'outil MPW, regardez:

```
$is_app    = $MacPerl::Version =~ /App/;
$is_tool   = $MacPerl::Version =~ /MPW/;
($version) = $MacPerl::Version =~ /^(\S+)/;
$is_ppc    = $MacPerl::Architecture eq 'MacPPC';
$is_68k    = $MacPerl::Architecture eq 'Mac68K';
```

Mac OS X, est basé sur OpenStep de NeXT, sera capable d'exécuter MacPerl en natif (dans la boite bleue, et même dans la boite jaune, une fois que certains changement des appels toolbox seront faits).
Consultez aussi:

The MacPerl Pages, `http://www.ptf.com/macperl/`.
The MacPerl mailing list, `mac-perl-request@iis.ee.ethz.ch`.

5.4 VMS

L'utilisation de Perl sous VMS est expliquée dans le *vms/perlvms.pod* de la distribution perl. Notez que perl sur VMS peut accepter indifféremment les spécifications de fichiers de style VMS et Unix:

```
$ perl -ne "print if /perl_setup/i" SYS$LOGIN:LOGIN.COM
$ perl -ne "print if /perl_setup/i" /sys$login/login.com
```

mais pas un mélange des deux:

```
$ perl -ne "print if /perl_setup/i" sys$login:/login.com
Can't open sys$login:/login.com: file specification syntax error
```

Utiliser Perl à partir du shell Digital Command Language (DCL) nécessite souvent de d'autres marques de quotation que sous un shell Unix. Par exemple:

```
$ perl -e "print ""Hello, world.\n"""
Hello, world.
```

Il y a plusieurs moyen d'inclure vos scripts perl dans un fichier DCL .COM si vous le souhaitez. Par exemple:

```
$ write sys$output "Hello from DCL!"
$ if p1 .eqs. ""
$ then perl -x 'f$environment("PROCEDURE")
$ else perl -x - 'p1 'p2 'p3 'p4 'p5 'p6 'p7 'p8
$ deck/dollars="__END__"
#!/usr/bin/perl

print "Hello from Perl!\n";

__END__
$ endif
```

Tenez compte du `$ ASSIGN/nolog/user SYS$COMMAND: SYS$INPUT` si votre script perl inclus essaye de faire des choses telle que `$read = <STDIN>;`.

Les noms de fichiers sont dans le format "nom.extension;version". La taille maximale est de 39 caractères pour le nom de fichier et aussi 39 pour l'extension. La version est un nombre compris entre 1 et 32767. Les caractères valides sont `/[A-Z0-9$_-]/`. Le système de fichier RMS de VMS est insensible à la casse et ne la préserve pas. `readdir` renvoie un nom de fichier en minuscule, mais l'ouverture reste insensible à la case. Les fichiers sans extensions sont terminés par un point, donc faire un `readdir` avec un fichier nommé *A.;5* retournera *a.* (ce fichier peut être ouvert par `open(FH, 'A')`).

RMS a une limitation à huit niveau de profondeur pour les répertoires à partir de n'importe quel racine logique (autorisant un maximum de 16 niveau) avant VMS 7.2. Dès lors `PERL_ROOT:[LIB.2.3.4.5.6.7.8]` est un répertoire valide mais `PERL_ROOT:[LIB.2.3.4.5.6.7.8.9]` non. Les auteurs de *Makefile.PL* peuvent l'avoir pris en compte, mais il peuvent au moins référencer le dernier par `/PERL_ROOT/lib/2/3/4/5/6/7/8/`.

Le module `VMS::Filespec`, qui est installe lors de la compilation sur VMS, est un module en Perl pure qui peut être installé facilement sur des plateformes non-VMS et peut être utile pour les conversion à partir et vers des formats natifs RMS.

La valeur de `\n` dépend du type de fichier qui est ouvert. Ça peut être `\015`, `\012`, `\015\012`, ou rien du tout. Lire d'un fichier traduits les retour chariots en `\012`, à moins que `binmode` soit exécuté sur ce handle de fichier, comme pour les perls DOS.

La pile TCP/IP est optionnel sur VMS, les routines socket ne sont donc pas forcément installés. Les sockets UDP peuvent ne pas être supportés.

La valeur de `$^O` sur OpenVMS est "VMS". Pour déterminer l'architecture sur laquelle le programme est exécuté sans utiliser `%Config` vous pouvez examiner le contenu du tableau `@INC`:

```
if (grep(/VMS_AXP/, @INC)) {
    print "I'm on Alpha!\n";
} elsif (grep(/VMS_VAX/, @INC)) {
    print "I'm on VAX!\n";
} else {
    print "I'm not so sure about where $^O is...\n";
}
```

Consultez aussi:

perlvms.pod

vmsperl list, `vmsperl-request@newman.upenn.edu`

Précisez `SUBSCRIBE VMSPERL` dans le corps du message.

vmsperl on the web, `http://www.sidhe.org/vmsperl/index.html`

5.5 Platformes EBCDIC

Les versions récentes de Perl ont été portées sur des plateformes telles que OS/400 sur les minicomputers AS/400 ainsi que OS/390 pour Mainframes IBM. De tels ordinateurs utilise le jeu de caractère EBCDIC en interne (habituellement le jeu de caractère ID 00819 sur OS/400 et IBM-1047 sur OS/390). Notez que sur les mainframe perl fonctionne actuellement sous le "Unix system services for OS/390" (connu précédemment sous le nom OpenEdition).

La version actuelle, la R2.5 de USS pour OS/390, de ce sous-système Unix ne supporte pas l'astuce du `#!` pour l'invocation des scripts. Mais, sur le OS/390 les script peuvent utiliser un en-tête similaire à ceci:

```
: # use perl
    eval 'exec /usr/local/bin/perl -S $0 ${1+"$@"}'
        if 0;
#!/usr/local/bin/perl     # juste un commentaire

print "Hello from perl!\n";
```

Sur ces plateformes, gardez à l'esprit que le jeu de caractère EBCDIC peut avoir un effet sur certaines fonctions perl (telles que `chr`, `pack`, `print`, `printf`, `ord`, `sort`, `sprintf`, `unpack`), aussi bien que sur les modification binaire à l'aide de constante ASCII en utilisant des opérateurs comme `^`, `&` et `|`, sans oublier de mentionner l'interface avec des ordinateurs ASCII en utilisant des sockets. (cf Les retours chariots (§3.1)).

Heureusement, la plupart des serveurs web pour mainframe convertissent correctement les `\n` de l'instruction suivante en leurs équivalent ASCII (notez que `\r` est le même sous Unix et OS/390):

```
print "Content-type: text/html\r\n\r\n";
```

La valeur de `$^O` sur OS/390 est "os390".

Voici des moyens simple de déterminé si le programme tourne sur une plateforme EBCDIC :

```
if ("\t" eq "\05")   { print "EBCDIC may be spoken here!\n"; }

if (ord('A') == 193) { print "EBCDIC may be spoken here!\n"; }

if (chr(169) eq 'z') { print "EBCDIC may be spoken here!\n"; }
```

Notez que vous ne devez pas vous reposer sur les caractères de ponctuation car leurs codes EBCDIC peut changer de page de code en page de code (et une fois que votre module ou script est connu comme fonctionnant avec EBCDIC, Il y aura toujours quelqu'un qui voudra qu'il fonctionne avec tous les jeux de caractères EBCDIC).

Consultez aussi:

perl-mvs list

La liste perl-mvs@perl.org est pour les discussions sur les problèmes de portage ainsi que les problèmes généraux de l'utilisation des Perls. EBCDIC. Envoyer un message contenant "subscribe perl-mvs" à majordomo@perl.org.

AS/400 Perl information `http://as400.rochester.ibm.com/`

5.6 Acorn RISC OS

Comme les Acorns utilise l'ASCII avec les retour chariots (\n) des fichiers texte égal à \012 comme sous Unix et que l'émulation des noms de fichier Unix est validée par défaut, il y a de grande chance pour que la plupart des scripts simples fonctionne sans modification. Le système de fichier natif est modulaire, et ces modules peuvent être insensible ou sensible à la casse, et conservent en général la casse des noms de fichier. Certains de ces modules ont des limites sur la taille des noms de fichier et ont alors tendance à tronquer les noms des fichiers et des répertoires pour qu'ils ne dépassent pas cette limite - les scripts doivent être savoir que le module par défaut est limité à **10** caractères et à un maximum de 77 éléments dans un répertoire, mais que la plupart n'imposent pas de telles limitations.

Les noms de fichier natifs sont de la forme

```
SystèmeDeFichier#ChampSpécial::NomDisque.$.Répertoire.Répertoire.Fichier
```

où

```
ChampSpécial n'est en général pas présent mais peut contenir . et $ .
SystèmeDeFichier =~ m|[A-Za-z0-9_]|
NomDisque   =~ m|[A-Za-z0-9_/]|
$ representes le répertoire racine
. est le séparateur de chemin
@ est le répertoire courant (par Système de fichier mais global à la machine)
^ est le répertoire parent
Répertoire and Fichier =~ m|[^\0- "\.\$\%\&:\@\\^\|\177]+|
```

La conversion par défaut des noms de fichier est à peu près `tr|/.|./|;`

Notez que `"ADFS::HardDisc.$.File" ne 'ADFS::HardDisc.$.File'` et que la deuxième étape de l'interprétation de `$` dans les expression régulière risque d'échouer sur le `$.` si le script ne fait pas attention.

Les chemins logiques spécifiés par des variables d'environnement contenant des listes de recherche séparées par des virgules sont aussi autorisées, de même que `System:Modules` est un nom de fichier valide, et que les système préfixera `Modules` par toutes les section de `System$Path` jusqu'à ce qu'un nom ainsi conçu pointe sur un objet sur disque. Écrire un nouveau fichier `System:Modules` ne sera autorisé que si `System$Path` un seul chemin. Le système de fichier converti aussi les variables systèmes en nom de fichier si elles sont entourées de signe <>, donc `<System$Dir>.Modules` recherche le fichier `$ENV{'System$Dir'} . 'Modules'`. L'explication évidente est que **les noms de fichiers absolus peuvent commencer par** <> et doivent être protégé lorsque l'on utilise `open` pour la lecture de donnée.

Comme `.` est le séparateur de chemin et que les noms de fichiers ne peuvent pas être présumé être unique après le dixième caractère, Acorn a implémenté le compilateur C de manière à retirer les extensions `.c` `.h` `.s` et `.o` des noms de fichiers spécifiés dans le code source et de stocker les fichiers respectifs dans des sous-répertoire au nom correspondant à l'extension. En voici quelques exemples:

```
foo.h           h.foo
C:foo.h         C:h.foo         (Variable de chemin logique)
sys/os.h        sys.h.os        (Le compilateur C baragouine de l'Unix)
10charname.c    c.10charname
10charname.o    o.10charname
11charname_.c   c.11charname    (présume que le système de fichier tronque à 10)
```

La librairie d'émulation Unix convertit les noms de fichier en natif en présument que ce type de conversion est nécessaire, et permet à l'utilisateur de définir sa liste d'extension pour lesquelles une telle conversion est nécessaire. Ça peut sembler transparent, mais considérez que `foo/bar/baz.h` et `foo/bar/h/baz` se traduisent tous les deux par `foo.bar.h.baz`, et que `readdir` et `glob` ne peuvent et n'essayeront pas d'émuler la conversion inverse. Les autres `.` dans les noms de fichiers seront convertis en `/`.

Comme précisé précédemment l'environnement accessible par `%ENV` est global, et la convention est qu'une variable spécifique à un programme soit de la forme `Programme$Nom`. Chaque module de systèmes de fichiers maintiennent un répertoire courant et celui du module courant est le répertoire courant **global**. En conséquence, les scripts sociaux ne change pas le répertoire courant, mais repose sur des noms de fichiers complets, et les scripts (et les Makefiles) ne peuvent supposer que le faite de créer un processus fils ne va pas modifier le répertoire courant de son père (et de tous les autres processus en général).

Les handle de fichier natifs du système d'exploitation sont globaux et sont alloués en descendant depuis 255, 0 étant réservé à la librairie d'émulation Unix. Par conséquence, ne vous reposez pas sur le passage de `STDIN`, `STDOUT`, ou `STDERR` à vos enfants.

Le désir des utilisateur d'exprimer des noms de fichier de la forme `<Foo$Dir>.Bar` sur la ligne de commande sans être protégés pose un problème: La sortie résultant de la commande " doit jouer aux devinettes. Il présume que la chaîne `<[^<>]+\$[^<>]>` est une référence vers une variable d'environnement, alors que tout le reste impliquant < ou > est une redirection, ce qui est 99% du temps exact. Bien sûr un problème récurrent est que les scripts ne peuvent compter sur le fait qu'un quelconque outil Unix soit disponible, et que s'il l'est, il utilise les mêmes arguments que la version Unix.

Les extensions et XS sont, en théorie, compilable par tous en utilisant des logiciels libres. En pratique, beaucoup ne le font pas, les utilisateurs des plateformes Acorn étant habitués aux distributions binaires. MakeMaker fonctionne, mais aucun make n'arrivera à traiter un Makefile créé par MakeMaker; et même si ceci est un jour corrigé, le manque d'un shell de type Unix pourra poser des problèmes avec les règles du makefile, particulièrement les lignes de la forme `cd sdbm && make all`, et tout ce qui utilise les protections.

"RISC OS" est le nom propre du système d'exploitation, mais la valeur de `$^O` est "riscos" (parce que nous n'ayons pas crier).

Consultez aussi:

perl list

5.7 Autres perls

Perl a été porté sur un très grand nombre de plateforme ne correspondant à aucune des catégorie ci-dessus. Certaines, telles que AmigaOS, BeOS, QNX, et Plan 9, ont été intégré dans le code source Perl standard. Vous pourriez avoir à consulter le répertoire *ports/* du CPAN pour des information, et même pour des binaires pour: aos, atari, lynxos, riscos, Tandem Guardian, vos, *etc.* (oui, nous savons que certain de ces OS peuvent correspondre à la catégorie Unix, mais il n'en sont pas de complétement standard.)

Consultez aussi:

Atari, Guido Flohr's page `http://stud.uni-sb.de/~gufl0000/`

HP 300 MPE/iX `http://www.cccd.edu/~markb/perlix.html`

Novell Netware

Un PERL.NLM libre basé sur perl5 est disponible pour Novell Netware à partir de `http://www.novell.com/`

6 IMPLÉMENTATION DES FONCTIONS

La liste ci-dessous contient les fonctions qui ne sont pas implémentées ou différemment sur des plateformes diverses. Chaque description est suivie de la liste des plateforme auxquelles elle s'applique.

La liste peut parfaitement être incomplète voir même fausse. En cas de doute, consulter le README spécifique à la plateforme de la distribution des sources perl, et d'autre document d'un portage donné.

Faites attention, que même les Unix connaissent des différences.

Pour beaucoup de fonctions, vous pouvez aussi interroger `%Config`, exporté par défaut de `Config.pm`. Par exemple, pour vérifier que la plateforme connaît l'appel système `lstat`, consultez `$Config{'d_lstat'}`. Lisez *Config.pm* pour une description complète des variables disponibles.

6.1 Liste Alphabétique des Fonctions Perl

-X FILEHANDLE

-X EXPR

-X

`-r`, `-w`, et `-x` ont un sens très limités; les applications et les répertoire sont exécutable, et il n'existe pas uid/gid (Mac OS)

`-r`, `-w`, `-x`, et `-o` indique si un fichier est oui ou non accessible qui peut ne pas refléter les protections de fichiers basées sur UIC. (VMS)

`-s` retourne la taille des données et non pas la taille totale des données et des ressources. (Mac OS).

`-s` par nom sur un fichier ouvert retournera la place réservée sur disque plutôt que la taille actuelle. `-s` sur un handle retournera lui la taille courante (RISC OS)

`-R`, `-W`, `-X`, `-O` ne peuvent être différencié de `-r`, `-w`, `-x`, `-o`. (Mac OS, Win32, VMS, RISC OS)

`-b`, `-c`, `-k`, `-g`, `-p`, `-u`, `-A` ne sont pas implémentés. (Mac OS)

`-g`, `-k`, `-l`, `-p`, `-u`, `-A` n'ont pas beaucoup de sens . (Win32, VMS, RISC OS)

`-d` est vrai si on lui passe une spécification de périphérique sans un répertoire explicite. (VMS)

`-T` et `-B` sont implémentés, mais peuvent mal classer les fichier texte Mac contenant des caractères étrangers; ça arrive sur toutes les plateforme, mais affecte plus souvent Mac OS. (Mac OS)

`-x` (or `-X`) détermine si le fichier possède une extension exécutable `-S` n'a pas de sens. (Win32)

`-x` (or `-X`) détermine si un fichier est du type exécutable. (RISC OS)

binmode FILEHANDLE

Sans intérêt. (Mac OS, RISC OS)

Rouvre le fichier et réinitialise les pointeurs; si la fonction échoue, le handle de fichier peut être fermé, ou le pointeur peut être à une position différente. (VMS)

La valeur retournée par `tell` peut être différente après l'appel, et le handle peut être vidé. (Win32)

chmod LIST

Son sens est limité. Désactiver/activer la permission en écriture est remplacé par la désactivation/activation d'un verrou sur le fichier. (Mac OS)

Ne peut modifier que les permission lecture/écriture pour le "propriétaire", les droits du "groupe" et des "autres" n'ont pas de signification. (Win32)

Ne peut modifier que les permission lecture/écriture pour le "propriétaire", et les "autres". (RISC OS)

chown LIST

Non implémenté. (Mac OS, Win32, Plan9, RISC OS)

Ne fait rien, mais ne renvoie pas d'erreur. (Win32)

chroot FILENAME

chroot

Non implémenté. (Mac OS, Win32, VMS, Plan9, RISC OS)

crypt PLAINTEXT,SALT

Peut ne pas être disponible si la librairie ou le source n'était pas fourni lors de la compilation de perl. (Win32)

dbmclose HASH

Non implémenté. (VMS, Plan9)

dbmopen HASH,DBNAME,MODE

Non implémenté. (VMS, Plan9)

dump LABEL

Pas utile. (Mac OS, RISC OS)

Non implémenté. (Win32)

Invoque le deboggeur VMS . (VMS)

exec LIST

Non implémenté. (Mac OS)

fcntl FILEHANDLE,FUNCTION,SCALAR

Non implémenté. (Win32, VMS)

flock FILEHANDLE,OPERATION

Non implémenté (Mac OS, VMS, RISC OS).

Disponible uniquement sous Windows NT (pas sous Windows 95). (Win32)

fork

Non implémenté. (Mac OS, Win32, AmigaOS, RISC OS)

getlogin

Non implémenté. (Mac OS, RISC OS)

getpgrp PID

Non implémenté. (Mac OS, Win32, VMS, RISC OS)

getppid

Non implémenté. (Mac OS, Win32, VMS, RISC OS)

getpriority WHICH,WHO

Non implémenté. (Mac OS, Win32, VMS, RISC OS)

getpwnam NAME

Non implémenté. (Mac OS, Win32)

Pas utile. (RISC OS)

getgrnam NAME
Non implémenté. (Mac OS, Win32, VMS, RISC OS)

getnetbyname NAME
Non implémenté. (Mac OS, Win32, Plan9)

getpwuid UID
Non implémenté. (Mac OS, Win32)
Pas utile. (RISC OS)

getgrgid GID
Non implémenté. (Mac OS, Win32, VMS, RISC OS)

getnetbyaddr ADDR,ADDRTYPE
Non implémenté. (Mac OS, Win32, Plan9)

getprotobynumber NUMBER
Non implémenté. (Mac OS)

getservbyport PORT,PROTO
Non implémenté. (Mac OS)

getpwent
Non implémenté. (Mac OS, Win32)

getgrent
Non implémenté. (Mac OS, Win32, VMS)

gethostent
Non implémenté. (Mac OS, Win32)

getnetent
Non implémenté. (Mac OS, Win32, Plan9)

getprotoent
Non implémenté. (Mac OS, Win32, Plan9)

getservent
Non implémenté. (Win32, Plan9)

setpwent
Non implémenté. (Mac OS, Win32, RISC OS)

setgrent
Non implémenté. (Mac OS, Win32, VMS, RISC OS)

sethostent STAYOPEN
Non implémenté. (Mac OS, Win32, Plan9, RISC OS)

setnetent STAYOPEN
Non implémenté. (Mac OS, Win32, Plan9, RISC OS)

setprotoent STAYOPEN
Non implémenté. (Mac OS, Win32, Plan9, RISC OS)

setservent STAYOPEN
Non implémenté. (Plan9, Win32, RISC OS)

endpwent
Non implémenté. (Mac OS, Win32)

endgrent
Non implémenté. (Mac OS, Win32, VMS, RISC OS)

endhostent
Non implémenté. (Mac OS, Win32)

endnetent
Non implémenté. (Mac OS, Win32, Plan9)

endprotoent
Non implémenté. (Mac OS, Win32, Plan9)

endservent
Non implémenté. (Plan9, Win32)

getsockopt SOCKET,LEVEL,OPTNAME

Non implémenté. (Mac OS, Plan9)

glob EXPR

glob

La complétion est interne, mais seuls le méta-caractères `*` et `?` sont supportés. (Mac OS)

Fonctionnalité dépendant d'un programme externe perlglob.exe ou perlglob.bat peut être surpassé par quelque chose comme File::DosGlob, ce qui est recommandé. (Win32)

La complétion est interne, mais seuls le méta-caractères `*` et `?` sont supportés. Elle dépends d'appels système, qui peuvent retourner les noms de fichier dans n'importe quel ordre. Comme la plupart des systèmes de fichier sont insensible à la casse, même les noms "triés" ne le seront pas dans un ordre dépendant de la casse. (RISC OS)

ioctl FILEHANDLE,FUNCTION,SCALAR

Non implémenté. (VMS)

Disponible seulement pour les handles de socket, et ne fait que ce que l'appel ioctlsocket() de l'API Winsock fait. (Win32)

Disponible seulement pour les handles de socket. (RISC OS)

kill LIST

Non implémenté, même pas utile pour la vérification de sécurité.(Mac OS, RISC OS)

Disponible uniquement pour les handles de processus retourné par `system(1, ...)`. (Win32)

link OLDFILE,NEWFILE

Non implémenté. (Mac OS, Win32, VMS, RISC OS)

lstat FILEHANDLE

lstat EXPR

lstat

Non implémenté. (VMS, RISC OS)

La valeur de retour peut être farfelu. (Win32)

msgctl ID,CMD,ARG

msgget KEY,FLAGS

msgsnd ID,MSG,FLAGS

msgrcv ID,VAR,SIZE,TYPE,FLAGS

Non implémenté. (Mac OS, Win32, VMS, Plan9, RISC OS)

open FILEHANDLE,EXPR

open FILEHANDLE

Les variantes | ne sont supportés que si ToolServer est installé. (Mac OS)

L'ouverture vers |- et -| ne sont pas supportés. (Mac OS, Win32, RISC OS)

pipe READHANDLE,WRITEHANDLE

Non implémenté. (Mac OS)

readlink EXPR

readlink

Non implémenté. (Win32, VMS, RISC OS)

select RBITS,WBITS,EBITS,TIMEOUT

Implémentés seulement sur les sockets. (Win32)

Fiables que sur les sockets. (RISC OS)

semctl ID,SEMNUM,CMD,ARG

semget KEY,NSEMS,FLAGS

semop KEY,OPSTRING

Non implémenté. (Mac OS, Win32, VMS, RISC OS)

setpgrp PID,PGRP

Non implémenté. (Mac OS, Win32, VMS, RISC OS)

setpriority WHICH,WHO,PRIORITY

Non implémenté. (Mac OS, Win32, VMS, RISC OS)

setsockopt SOCKET,LEVEL,OPTNAME,OPTVAL

Non implémenté. (Mac OS, Plan9)

shmctl ID,CMD,ARG

shmget KEY,SIZE,FLAGS

shmread ID,VAR,POS,SIZE

shmwrite ID,STRING,POS,SIZE

Non implémenté. (Mac OS, Win32, VMS, RISC OS)

socketpair SOCKET1,SOCKET2,DOMAIN,TYPE,PROTOCOL

Non implémenté. (Mac OS, Win32, VMS, RISC OS)

stat FILEHANDLE

stat EXPR

stat

mtime et atime représente la même chose, et ctime est la date de création au lieu de la date du dernier changement de l'inode. (Mac OS)

device et inode n'ont aucun sens. (Win32)

device et inode ne sont pas forcément fiables. (VMS)

mtime, atime et ctime correspondent à la date de dernière modification. device et inode ne sont pas forcément fiables. (RISC OS)

symlink OLDFILE,NEWFILE

Non implémenté. (Win32, VMS, RISC OS)

syscall LIST

Non implémenté. (Mac OS, Win32, VMS, RISC OS)

sysopen FILEHANDLE,FILENAME,MODE,PERMS

Les valeurs traditionnelles des modes "0", "1", et "2" sont implémentés par des valeurs numériques différentes sur quelques systèmes. Les drapeaux exportés par `Fcntl` (O_RDONLY, O_WRONLY, O_RDWR) devraient fonctionner sur tout les systèmes. (Mac OS, OS/390)

system LIST

Implémenté que si ToolServer est installé. (Mac OS)

Comme optimisation, peut ne pas faire appel au shell de commande spécifié par `$ENV{PERL5SHELL}`. `system(1, @args)` crée un processus externe et renvoie immédiatement le handle de processus, sans attendre qu'il se termine. La valeur de retour peut donc être utilisée par `wait` ou `waitpid`. (Win32)

Il n'y a pas de shell pour traiter les méta-caractères, et le standard natif est de passer des ligne de commandes terminées par "\n" "\r" ou "\0" au processus créé. Les redirections tels que `> foo` sont réalisées par la librairie d'exécution du processus créé. `system` *list* fait appel à la fonction `exec` de la librairie d'émulation Unix, qui tente de fournir une émulation de stdin, stdout, stderr en forçant le père, présumant que le programme fils utilise une version compatible de la librairie d'émulation. *scalar* fait appel à la ligne de commande native et une telle émulation n'est pas utilisé. Ce fonctionnement **va** varier. (RISC OS)

times

Seule la première entrée est différente de zéro. (Mac OS)

Les temps "cumulés" sont faux. Sur tout les systèmes autres que Windows NT, le temps "système" est faux, et le temps "utilisateur" est actuellement la date retourné par la fonction clock() de la librairie C. (Win32)

Pas utile. (RISC OS)

truncate FILEHANDLE,LENGTH

truncate EXPR,LENGTH

Non implémenté. (VMS)

umask EXPR

umask

Retourne undef lorsqu'elle n'est pas disponible depuis la version 5.005.

utime LIST

Seule la date de modification est mise à jour. (Mac OS, VMS, RISC OS)

Peut ne pas fonctionné comme attendu. Le comportement dépend l'implémentation de la fonction utime() de la librairie C, et du type de système de fichier utilisé. Le système FAT typiquement ne supporte pas le champs "date d'accès", et peut limiter les marques de temps à une granularité de deux secondes. (Win32)

wait

waitpid PID,FLAGS

Non implémenté. (Mac OS)

Disponible uniquement pour les handles de processus retourné par `system(1, ...)`. (Win32)

Pas utile. (RISC OS)

7 AUTEURS / CONTRIBUTEURS

Abigail <abigail@fnx.com>, Charles Bailey <bailey@genetics.upenn.edu>, Graham Barr <gbarr@pobox.com>, Tom Christiansen <tchrist@perl.com>, Nicholas Clark <Nicholas.Clark@liverpool.ac.uk>, Andy Dougherty <doughera@lafcol.lafayette.edu>, Dominic Dunlop <domo@vo.lu>, M.J.T. Guy <mjtg@cus.cam.ac.uk>, Luther Huffman <lutherh@stratcom.com>, Nick Ing-Simmons <nick@ni-s.u-net.com>, Andreas J. König <koenig@kulturbox.de>, Andrew M. Langmead <aml@world.std.com>, Paul Moore <Paul.Moore@uk.origin-it.com>, Chris Nandor <pudge@pobox.com>, Matthias Neeracher <neeri@iis.ee.ethz.ch>, Gary Ng <71564.1743@CompuServe.COM>, Tom Phoenix <rootbeer@teleport.com>, Peter Prymmer <pvhp@forte.com>, Hugo van der Sanden <hv@crypt0.demon.co.uk>, Gurusamy Sarathy <gsar@umich.edu>, Paul J. Schinder <schinder@pobox.com>, Dan Sugalski <sugalskd@ous.edu>, Nathan Torkington <gnat@frii.com>.

Ce document est maintenu par Chris Nandor.

8 VERSION

Version 1.34, last modified 07 August 1998.

9 TRADUCTION

9.1 Version

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

9.2 Traducteur

Marc Carmier <carmier@immortels.frmug.org>.

9.3 Relecture

Personne pour l'instant.

10 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.