Perl

Général

Aide & documentation

```
perldoc –f <fonction>   # Affiche la doc de la fonction
man perlfunc       # Liste des fonctions disponibles
man CGI            # Liste les fonctions du module CGI
man perl      # Liste les aides disponibles
```

Pragma

Il s'agit de directives pour le compilateur que l'on active par le mot clef **use** et que l'on désactive par le mot clef **no**.

```
use strict ;  # Impose la déclaration des variables.
        # De plus rend la portée de la variable locale au bloc où elle est déclarée (entre { et }).
no strict ;   # Déclaration facultative.
use integer ; # Division entière : 10/3 = 3
no integer ;  # Division normale : 10/3 = 3.3333
```

Gestion d'erreur

```
open (…) or die ("….n"); # en cas d’erreur, affiche le message sur STDERR et sort.
open (…) or die ("$ ! "); # en cas d’erreur affiche le n°. En l’absence de \n affiche aussi le numéro de la ligne courante.
warn("…");         # affiche le message sans terminer l’exécution.
```

Options

-e	permet d'exécuter du code perl directement en ligne de commande
-w	active les messages de warning
-d	exécution en mode debug
-n	boucle while sur la commande
-p	comme -n mais affiche les lignes en plus (print)
-l	active le process de fin de ligne

Utilisation en ligne de commande

Pour balayer l'entrée standard utiliser les options -lne :

```
cat <fichier>|perl –lne ‘if (/^expression régulière$/) { print "$1 $2 …" }’
```

Variables

Déclaration

my

```
my $var;    # déclaration pour un bloc de code
our $var;   # déclaration globale
local $var; # déclaration pour un bloc de code ainsi que les sous routines appelées à partir de ce bloc de code
state $var; # la variable n'est initialisé qu'une seule fois
```

local

```
our $global_var = 42;  # Variable globale

sub exemple_local {
    local $global_var = 10;             # Modifie temporairement $global_var
    print "Dans local: $global_var\n";  # Affiche 10
    sous_fonction();                    # Appel d'une autre fonction
}

sub sous_fonction {
    print "Dans sous_fonction: $global_var\n";  # Affiche 10
}

exemple_local();
print "Après local: $global_var\n";  # Affiche 42
```

state

```
use feature 'state';

sub compteur {
    state $count = 0;  # $count est initialisé une seule fois
    $count++;
    return $count;
```

```
}

print compteur();  # Affiche 1
print compteur();  # Affiche 2
print compteur();  # Affiche 3
```

```
defined  # pour savoir si une variable est initialisée.
undef    # pour désinitialiser une variable.
```

boolean

Il n'existe pas de type booléen en perl. Une variable dont la valeur est 0 est considéré comme false dans le cadre d'un test alors que toute autre valeur, caractère ou valeur non nulle, sera considérée comme vrai.

Scalaire : $calaire

Base

10	décimal
010	octal
0x10	héxadécimal

Exemples

```
$var="30 km" # $dist=4*$var ;    120 (prend le début numérique)
$var="20"    # $d=2x$var ;       2020 (duplication)
$var="20"    # $d=2x$var-12 ;    2008
```

Tableau : @rray

Un tableau est un ensemble ordonné de valeurs. Les indices du tableau vont de 0 à $#tab (indice du dernier élément).

```
@var=(4,"truc",$var,10..20) # ( ) contexte de liste, tableau de 14 valeurs
```

Affectation scalaire/tableau

```
$v1=(4, "truc",$var,10..20);        # 20 (on récupère la dernière valeur de
la liste)
($v1)=(4, "truc",$var,10..20);        # 4 (on récupère la première valeur de
la liste)
($v1,$v2)=(4,"truc",$var,10..20);   # 4, "truc" (on récupère les 2 premières
valeurs de la liste).
$nb=@var;               # 14 (on récupère le cardinal de la liste.
```

```
($var,@tab)=@tab         # récupère la première valeur et le reste de la
liste.
@tab=(@tab1,@tab2) ;            # concatène les 2 tableaux.
(@tab,$nb)=(@tab1,@tab2) ;      # @tab prend toute la concaténation, $nb
reste vide.
@liste=qw/a b c d/;          # qw=quoted word, équivalent à
@liste=("a","b","c","d");
```

Accès direct aux éléments

```
$tab[0]         # premier élément
$tab[$#tab]     # dernier élément. Equivalent à $tab[-1] (on lit le tableau
à l'envers).
@var=@tab[10..15]   # on prend les élément de 10 à 15
@var=@tab[10,15,20]   # on prend les éléments 10,15 et 20.
(fonction())[…]       # pour récupérer des éléments d'une liste rendue par
une fonction.
```

parcourir tableau

```
for my $element (@tableau) { print "$element\n" }
for (my $i=0; $i <= $#tableau ; $i++) { print "$tableau[$i]\n"}
```

Fonctions sur les tableaux

```
shift   $var=shift(@tab);        # extraction du 1er élément et décalage de
la liste
unshift   unshift(@tab,…);        # on insère au début de la liste les
éléments indiqués.
pop   pop(@tab);            # extraction du dernier élément de la liste.
push push(@tab,…);            # on insère les éléments à la fin de la liste.
reverse   @tab=reverse(@tab);        # inverse les éléments du tableau.
sort @tab=sort(@tab);        # tri ASCII croissant.
    @tab=sort{$a cmp $b}@tab    # tri ASCII croissant
    @tab=sort{$b cmp $a}@tab    # tri ASCII décroissant
    @tab=sort{$a <=> $b}@tab    # tri numérique croissant
    @tab=sort{$b <=> $a}@tab    # tri numérique décroissant
split   @tab=split(/:/,$ligne);        # récupère la liste des éléments
séparés par :
        @tab=split(/;/,$ligne,-1);      # retourne un tableau avec tous les
éléments même si les derniers sont vides
        @tab=split(/,/,$ligne,2);       # retourne un tableau avec maximum 2
éléments. Le dernier aura peut-être des chaînes contenant des virgules
join $var=join('-',(1,2,3,4));    # concatène en une seule chaîne 1-2-3-4
splice splice(@tab,$deb,$n);        # enlève n éléments de tab en commençant
à deb retourne les éléments supprimés
grep @tab=grep(/regexp/,@tab);    # recherche l'expression régulière dans le
```

```
tableau, retourne liste correspondances
if      ("@tab" =~ /$val/)              # teste l'appartenance de la valeur
au tableau
if      (@tab)                          # teste si le tableau est vide
```

Tableau à 2 dimensions

```
$Array[0][0] = "0 0";
$Array[0][1] = "0 1";
$Array[0][2] = "0 2";
$Array[0][3] = "0 3";
$Array[1][0] = "1 0";
$Array[1][1] = "1 1";
$Array[1][2] = "1 2";

print "$#Array\n";                # dernier élément pour la première
dimension = 1 donc 2 éléments
print scalar @{$Array[0]}."\n";   # nombre d'éléments pour la deuxième
dimension = 4

push(@{$Array[3]},"0");           # Ajout d'un élément avec push
```

Table de hachage

Affectation

```
%table=('monday','lundi', ….'sunday','dimanche') ;
%table=(monday => 'lundi', …, sunday => 'dimanche');
```

Accès

```
$table{$cle}=$valeur;
```

Fonctions

```
keys(%tab)        # liste des clés
values(%tab)        # liste des valeurs
each(%tab)        # parcourt la liste et donne clé-valeur
delete $table{$cle};    # supprime la clé et sa valeur associée.
exists $table{$cle}  # test l'existence de la clé dans la table (même s'il
n'y a pas de valeur associée).
```

Parcourir une table

```
@cle=keys %table;                    # retourne la liste des clefs
@valeur=values %table;                 # retourne la liste des valeurs.
for $cle (keys (%table)) { traitement }           # retourne les clefs dans un ordre aléatoire
for $cle (sort keys (%table)) { traitement }       # retourne les clefs trié en ordre alphabétique
for $valeur (values (%table)) { traitement }        # retourne les valeurs dans un ordre aléatoire
@cle=sort {$table{$a} <=> $table{$b}} keys(%table);  # retourne la liste des clefs dans ordre des valeurs associées
```

Nombre d'éléments dans une table

On obtient le résultat en comptant les clefs :

```
$nombre = keys (%table);
```

hash de hash

```
my %hash;

$hash{"cle1"}{"cle2"}="toto";

for my $cle1 (sort keys %hash) {
    print "$cle1 : \n";
    for my $cle2 (sort keys %{$hash{$cle1}}) {
        print "\t$cle2 : $hash{$cle1}{$cle2}\n";
    }
}
```

Variables prédéfinies

$PROGRAM_NAME	$0	nom du script en cours d'exécution tel que passé dans la ligne de commande
@ARGV		Contient les arguments de la ligne de commande
$PID	$$	n° de processus exécutant le script
$OSNAME	$^O	système d'exploitation
$BASETIM	$^T	Heure à laquelle le script a démarré en seconde depuis le 01/01/1970
$EXECUTABLE_NAME	$^X	nom (avec chemin) de l'interpréteur perl
$UID	$<	uid réel du processus
$EUID	$>	uid effectif du processus
$GID	$(	gid réel du processus (liste des groupes séparés par des espaces)
$EGID	$)	gid effectif du processus
$RS	$/	séparateur d'enregistrement en lecture (\n par défaut)

$OFS	$,	séparateur de champs pour l'opérateur de sortie
$ORS	$\	séparateur d'enregistrement pour l'opérateur de sortie
$NR	$.	numéro de ligne courante du dernier fichier lu
	$[	Index du premier élément d'un tableau et du premier caractère dans une sous-chaîne de caractères. Par défaut 0 mais peut être positionné à 1
$PERL_VERSION	$]	Version de perl et niveau de patch
$BASETIME	$^T	Heure à laquelle le script a démarré (en secondes depuis le 01/01/1970)
$ARGV		Contient le nom du fichier courant quand on lit depuis <>
@ARGV		Contient les arguments de la ligne de commande du script
@INC		Contient la liste des répertoires où chercher pour les directives **require** et **use**
%INC		Contient une entrée pour chacun des fichiers inclus par **require**. La clef est le nom du fichier et la valeur contient la localisation
%ENV, $ENV{expr}		contient les variables d'environnement. Modifiable pour les processus fils.
%SIG, $SIG{INT}		contient les handlers de signaux

https://perl.mines-albi.fr/DocFr/perlvar.html

Parcourir les variables d'environnement

```
while (($cle,$valeur) = each %ENV) {
    print "$cle=$valeur\n";           # Affiche les variables
d'environnement.
}
```

Tester si on est root

```
if ($<) {
   print "on n'est pas root";
} else {
   print "on est root\n";
}
```

lire un fichier d'un bloc

```
{
  local $/;   # on défini la variable $/ à undef
  my $contenu = <FICHIER>;  # la variable contient tout le contenu du
fichier
}
# $/ revient à la valeur par défaut \n
```

Opérateurs

Mathématiques

Multiplication	*
Division	/
Addition	+
Soustraction	-
Modulo (reste)	%
Incrémentation (pré & post)	++
décrémentation (pré & post)	- -

Logiques booléens

Négation logique	!	utilisation possible de « not » depuis perl5
OU	||	évaluation booléenne complète
	or	évaluation booléenne incomplète (plus rapide sans exécuter la partie droite si la gauche est déjà vraie)
	xor	ou exclusif
ET	&&	utilisation possible de « and » depuis perl5

Logiques bit à bit

Négation bit à bit	~	Egalement appelé complément à 1
ET bit à bit	&	Retourne 1 si les deux bits de même poids sont à 1
OU bit à bit	|	Retourne 1 si l'un ou l'autre des deux bits de même poids est à 1 (ou les deux)
OU exclusif bit à bit	^	Retourne 1 si l'un des deux bits de même poids est à 1 (mais pas les deux)
Rotation à gauche	<<	Décale les bits vers la gauche (multiplie par 2 à chaque décalage)
Rotation à droite avec conservation du signe	>>	Décale les bits vers la droite (divise par 2 à chaque décalage)
Rotation à gauche avec remplissage de 0	>>>	Décale les bits vers la droite (divise par 2 à chaque décalage)

Comparaison numérique

Inférieur	<
Supérieur	>
Inférieur ou égal	<=
Supérieur ou égal	>=
Egalité	==
Différent	!=

Attention = correspond à l’affectation.

Comparaison alphabétique

Inférieur	lt
Supérieur	gt
Inférieur ou égal	le
Supérieur ou égal	ge
Egalité	eq
Différent	ne

Divers

Répétition	x	print '-' x 80 ; affiche une ligne de 80 '–'
Concaténation	.	Concatène 2 chaînes

- Opérateurs
- Opérateurs et priorités

Structures complexes

Références

opérateur \

Les références peuvent être créées en utilisant l'opérateur « \ » sur une variable, une fonction ou une valeur

```
$refscalaire = \$scalaire;
$reftableau  = \@tableau;
$rehashage   = \%hashage;
$reffonction = \&fonction;  # référence sur une fonction
```

opérateur [ ]

Il est possible de créer une référence sur un tableau anonyme

```
$reftableau = [1,2,['a','b','c']];  # le 3° élément du tableau est lui-même une référence sur un tableau anonyme de 3 éléments
```

opérateur =>

Il est possible de créer une référence sur une table de hashage anonyme

```
$refhash = { 'monday' => 'lundi', 'tuesday' => 'mardi' };
```

opérateur *

Opérateur de référencement historique du C.

Déréférencement

Opérateur { }

```
print ${$refscalaire};                # peut s'écrire $$refscalaire
print @{$reftableau};                 # peut s'écrire @$reftableau
for my $key (%($refhashage) { };      # peut s'écrire %$refhashage
&{$refonction}();
```

opérateur ->

```
print $reftableau->[0];  # affiche le premier élément du tableau
print $refhashage->{cle} # affiche la valeur associé à cle dans la table de
hashage
```

Références symboliques

Lorsqu'une valeur utilisée comme référence est déjà définie mais n'est pas une référence dure, la valeur du scalaire est considérée comme le nom d'une variable.

```
$name        = "variable";
$$name       = 1;   # affecte $variable
${$name}     = 2;   # affecte $variable
${$name x 2} = 3;   # affecte $variablevariable
$name->[0]   = 4;   # affecte $variable[0]
@$name       = ();  # efface @variable
&$name();           # Appelle &$variable()
```

Fonctions

Fonctions mathématiques

```
abs(x)     # valeur absolue
cos(x)     # cosinus
exp(x)     # exponentielle
int(x)     # partie entière
log(x)     # logarithme
sin(x)     # sinus
```

```
sqrt(x)       # racine carrée
rand    # entier pseudo aléatoire
srand      # nouvelle séquence de nb aléatoires
atan2(x,y) # arc tangente de y/x
```

Arrondi à la nième décimale

```
my $arrondi=int($valeur * 10 ** $n + 0.5) / 10 ** $n;
```

Fonctions manipulation de chaînes de caractères

```
chop(chaine);               # suppression du dernier caractère
chomp(chaine);              # suppression du retour chariot
$ligne=~s/\r\n/\n/;         # pour faire un dos2unix quand le chomp ne
fonctionne pas
index(chaine,sschaine);     # indice de la 1ière occurrence de chaine dans
sschaine
rindex(chaine,sschaine);    # indice de la dernière occurrence de chaine
dans sschaine
index(chaine,sschaine,pos); # indice de la 1ière occurrence de chaine dans
sschaine à partir de la position pos
length(chaine);             # taille de la chaîne
lc(chaine);                 # retourne la chaîne en minuscule (perl 5)
lcfirst(chaine);            # retourne la chaîne avec le 1er caractère en
minuscule
split(sep,chaine);      # sépare la chaîne en éléments en fonction du
séparateur
substr(chaine,début,lng);   # extraction d'une sous chaîne
uc(chaine);                 # retourne la chaîne en majuscules (perl5)
ucfirst(chaine);            # met le 1er caractère de la chaîne en majuscule
chr(code);                  # retourne le caractère qui correspond au code
ASCII
ord('x');                   # retourne le code ASCII d'un caractère
$toto=~s/toto/titi/g;       # substitution de toto par titi dans la chaîne
de caractères (autant de fois que nécessaire)
$toto=quotemeta(chaine);    # ajoute des \ à tous les caractères autre que
[A-Za-z_0-9]
$toto=sprintf("%04d",$i);   # retourne le numérique formaté sur 4 digit
(0003 pour 3)
```

printf

```
printf "fmt1 fmt2 ... fmtx",val1,val2,...,valx
```

Même convention que la fonction printf du langage C.

Format	Conversion
%%	Caractère %
%c	Un caractère dont on fournit le code
%s	Une chaîne
%d	Un entier signé, en décimal
%u	Un entier non signé, en décimal
%o	Un entier non signé, en octal
%x	Un entier non signé, en hexadécimal
%e	Un nombre en virgule flottante, en notation scientifique
%f	Un nombre en virgule flottante, avec un nombre de décimales fixe
%g	Un nombre en virgule flottante, %e ou %f (au mieux)
%X	Comme %x mais avec des lettres majuscules
%E	Comme %e, mais en utilisant un E majuscule (si nécessaire)
%G	Comme %g, mais en utilisant un E majuscule (si nécessaire)
%p	Un pointeur (affiche la valeur perl de l'adresse en hexadécimal
espace	Précède les nombres positifs par un espace
+	Précède les nombres positifs par un signe plus
-	Justifie le champ à gauche
0	Utilise des 0 à la place des espaces pour justifier à droite
#	Précède le nombre non nul en octal par “0”
:	Précède le nombre non nul en hexadécimal par “0x”
nombre	Taille minimum du champ
.nombre	“précision” : nombre de décimales pour un nombre en virgule flottante
l	Interprète un entier comme le type C “long” ou “unsigned long”
h	Interprète un entier comme le type C “short” ou “unsigned short”

Fonctions exécution

```
exec(« cmd »)   # exécute la commande, pas de valeur de retour et pas de retour au programme perl (sauf erreur)
system(« cmd ») # sortie console, attend la fin d’exécution. Valeur retournée code de retour de la commande.
my $rc=qx(cmd)  # idem permet de récupérer la sortie de la commande. Pas de sortie console.
                # Code de retour dans variable $?
                # Peut s’écrire qx(cmd), qx/cmd/ ou qx`cmd`
`cmd`           # idem qx, écriture simplifiée.
```

> Avec la commande exec on quitte perl pour ne plus y revenir. La seule raison de placer du code Perl après un exec() est d'expliquer que la commande n'a pu être trouvée dans le PATH

Fonctions Date/heure

```
time      # retourne le nombre de secondes depuis le 01/01/1970
mktime    # convertit une date en nombre de secondes depuis le 01/01/1970
(use POSIX 'mktime';)
gmtime    # retourne heure Greenwitch
localtime # retourne heure locale
strftime  # formatage date/heure      (use POSIX 'strftime';)
```

Convertir un nombre de secondes depuis le 01/01/1970

```
my ($sec, $min, $hour, $day, $month, $year, $wday)=localtime(time());
```

Le mois retourné est une valeur entre 0 et 11, il faut donc ajouter 1.

L'année retournée est par rapport à l'année 1900, il faut donc ajouter 1900.

Afficher en clair le résultat

```
strftime("%d%m%Y",0,0,0,$day,$month,$year);
```

Convertir en nombre de secondes depuis le 01/01/1970

```
use POSIX 'mktime';

mktime(sec,min,heure,jour,mois-1,année-1900)
```

Fonctions diverses

```
getlogin # donne le nom du login
exit     # interruption du script avec code retour
sleep(x) # suspend l'exécution pendant x secondes
die      # interruption avec message
alarm(x) # Met en place un SIGALARM à délivrer au processus après x
secondes. Une seule horloge à la fois. alarm(0) pour annuler.
caller   # retourne le contexte de l'appel de la routine en cours (voir
exemple)
crypt    # crypte une chaîne de caractères
```

caller

```
sub debug {
    my ($package, $filename, $line, $subroutine) = caller();
    print "Appel depuis sous routine $subroutine du package $package du
fichier $filename ligne $line\n";
}
```

```
sub fonction {
    debug();
}

fonction();
```

Structures de contrôle

Boucles

```
for (my $i=$n; $i > 0 ; $i--) { ... }  # Boucle de $n à 1
for $var (liste) { ... }               # Parcourt la liste
foreach $var (liste) { ... }           # Cette forme préserve la variable
$var si utilisée avant
for (@tab) { ... }                     # Équivalent à for $_ (@tab) { ... }
while (...) { ... }
until (...) { ... }
do { ... } while (...)
```

Contrôle des itérations

```
last   # équivalent du break, sort de la boucle
next   # équivalent du continue, passe à l'itération suivante
redo   # rejoue l'itération
```

Tests

Stucture classique

```
if (...) {
} elsif (...) {
} else {
}
```

notation inversée (test à la fin)

```
<instruction> if (...);        # Attention une seule instruction pas de bloc
de commande !
<instruction> unless ( ...);   # Test inversé équivalent à if (! ... )
```

Utiliser == sinon un seul signe = équivaut à une affectation.

Procédure / Fonctions

```
sub toto {
  les paramètres se retrouvent dans @_ : $_[0], $_[1], ...
  return $retour ou bien return @retour pour retourner des paramètres
}
```

```
print @_."\n";          # affiche le nombre d'arguments car le . s'applique
à des scalaires.
print scalar(@_)."\n";  # idem
print "@_\n";           # affiche la liste des éléments
```

Passage de variables par adresse

```
sub toto {
  my $ref_tab1=$_[0] ; my $ref_tab2=$_[1] ;
  my @tab1=@$ref_tab1 ; my %tab2=%$ref_tab2 ;
}

toto(\@tab1,\%tab2);
```

Dans cet exemple on a créé une copie locale dont les modifications ne sont pas visibles à l'extérieure de la procédure.

Avec modification visible à l'extérieure de la procédure

```
sub Ma_Fonction {
    my ($hash1, $hash2, $hash3) = @_;

    print "Cle 1 de hash toto = $hash1->{1}\n";
    print "Cle 1 de hash pouet = $hash2->{3}\n";
    print "Cle 1 de hash truc = $hash3->{5}\n";

    $hash1->{1} = 'POUETTTT';
}

Ma_Fonction(\%toto, \%pouet, \%truc);
```

```
sub toto {
  my $ref_tab1=$_[0];
```

```
  $ref_tab1->[$#{ref_tab1}]="modification du dernier élément du tableau";
}

toto(\@tab1);
```

Retourner une table hash

```
sub toto {
    my %resultat;
    ...
    return (\%resultat);
}

my $ref=&toto;
my %resultat=%$ref;
```

Gestion des erreurs

try / catch / finally

```
use experimental 'try';

try {
    appel_fonction();
}
catch ($e) {
    warn "Erreur appel fonction : $e";
}
finally {
    print "Fin\n";
}
```

autodie

autodie

```
eval {
    use autodie;

    open(my $fh, '<', $some_file);
    my @records = <$fh>;
    ...
    close($fh);
};

if ($@ and $@->isa('autodie::exception')) {
```

```
    if ($@->matches('open')) { print "Error from open\n";   }
    if ($@->matches(':io' )) { print "Non-open, IO error."; }
} elsif ($@) {
    # A non-autodie exception.
}
```

E/S

E/S standard

Handle de fichier

```
STDIN    # entrée standard (clavier)
STDOUT   # sortie standard (écran)
STDERR   # sortie erreurs
```

Exemples

```
$var = <STDIN>;            # affecte à partir de l'entrée standard (clavier ou redirection)
@tab = <STDIN>;            # affecte à partir de l'entrée standard plusieurs lignes dans une liste.
print STDERR "erreurn"    # affiche sur la sortie d'erreur (comme warn ou die)

# saisie d'un mot de passe sans affichage à l'écran
do {
    system("stty -echo");
    printf "Saisie mot de passe : "
    $pw=<STDIN>; chomp $pw;
    system("stty echo");
    printf "\n";
} while ($pw eq "");
```

Fichiers textes

```
open(my $IN,"/tmp/fichier");              # Ouverture d'un fichier pour lecture
open(my $OUT,"| commande");               # Redirection dans une commande
open(my $OUT,">/tmp/fichier");            # Ecrase le fichier
open(my $OUT,">>/tmp/fichier");           # Ajoute au fichier
open(my $IN,"ls –l /etc |");              # On pipe la commande pour récupérer le résultat dans notre entrée fichier.
print $OUT "…\n";                         # Ecriture simple dans le fichier
```

```
close(my $OUT);                              # Ferme le fichier.
while (defined($ligne=<$IN>)) {   }          # Pour être sûr, au cas où une
ligne contiendrait undef
foreach my $ligne (reverse(<$IN>)) { }       # Pour parcourir le fichier à
l'envers
```

lecture d'un fichier ligne par ligne

```
open(my $IN,"< $fichier") || die "Impossible d'ouvrir le fichier $fichier :
$!";
while (defined($ligne=<$IN>)) {
    chomp $ligne;
}
close $IN;
```

lecture d'un fichier en entier dans un tableau

```
open(my $IN,"< $fichier") || die "Impossible d'ouvrir le fichier $fichier :
$!";
my @lignes=<$IN>;
close $IN;
```

écriture dans un fichier ligne par ligne

```
open(my $OUT,">$fichier") || die "Impossible d'écrire dans le fichier
$fichier : $!";
for my $ligne (@lignes) {
    print $OUT "$ligne\n";
}
close $OUT;
```

Passage d'un filehandle dans une variable

```
sub maproc {
    my $filehandle=$_[0];
    my $nom_fichier=$_[1];

    open($filehandle,">$nom_fichier");
}

maproc(*OUT,"nom_du_fichier");
```

Encodage des caractères

- En cas de problème d'encodage, avec l'UTF8 notamment, il est possible de le préciser :

```
open(my $OUT,">:utf8","/tmp/fichier");
```

- Ajout un BOM (Byte Order Mark) en début de fichier

```
open(my $OUT,">:utf8","/tmp/fichier");
print $OUT "\x{FEFF}";  # BOM pour UTF-8
```

- Pour vérifier l'encodage d'un fichier :

```
file -i /tmp/fichier.txt
fichier.txt: text/plain; charset=utf-8
```

- Possibilité d'utiliser la commande iconv pour convertir des fichiers.

chomp

```
chomp $ligne;   # vire le \n de la fin
chomp @tab;     # vire le \n sur chaque élément
```

Modifier un fichier texte

A l'aide de l'unité Tie::File on peut manipuler le fichier au travers d'un tableau qui contient toutes les lignes du fichier.

```
use Tie::File;

tie(my @fichier,'Tie::File',"/chemin/vers/fichier") || die("Impossible
d'ouvrir le fichier : $!");

for (@fichier) {
  s/<expression régulière>/chaine de remplacement/g;        # remplace
l'expression régulière dans tout le fichier
}

untie @fichier;
```

Toutes les fonctions habituelles sur les tableaux sont utilisables (push, pop, unshift, shift, splice) sauf qu'elles s'appliquent directement au fichier.

Source

Fonctions sur les fichiers

```
@liste_fichiers = glob("/tmp/*.txt");     # renvoi la liste des fichiers
.txt dans le répertoire /tmp
symlink($fichier,$lien);                  # création lien sur fichier
```

```
(attention le nom du nouveau lien doit être complet)
readlink($toto);                          # retourne le fichier pointé par
un lien
unlink(@liste_de_fichier);                # efface la liste de fichiers,
retourne le nombre de fichier(s) effacé(s).
rename($file1,$file2);                    # renomme file1 en file2 (retourne
0 en cas d'échec et 1 en cas de succès)
move($file1, $file2);                     # déplace fichier file1 en file2
(use File::Copy)
copy($file1, $file2);                     # copy file1 en file 2 (use
File::Copy)
cp($file1, $file2);                       # copy file1 en file 2 en
conservant les droits du fichier (use File::Copy)
$rep=getcwd();                            # récupère répertoire courant (use
Cwd)
$rep=abs_path($chemin);                   # retourne le chemin absolue (use
Cwd 'abs_path')
$rep=File::Spec->abs2rel($chemin, $base); # retourne le chemin relatif (use
File::Spec)
chdir($chemin);                           # changer le répertoire courant
dirname($fichier);                        # retourne le chemin du fichier
(use File::Basename)
basename($fichier,@suffixlist);           # retourne le nom du fichier (use
File::Basename)
($nom,$path,ext)=fileparse($fic,@suffix); # retourne le nom, le chemin et
l'extension du fichier (use File::Basename)
mkpath                                    # création arborescence (use
File::Path)
rmtree                                    # suppression arborescence (use
File::Path)
utime($a_time,$mtime,$fichier);           # modifie la date d'accès et de
modification d'un fichier
```

La fonction rename ne fonctionne pas d'un disque à un autre, pour cela utiliser de préférence la fonction move.

La fonction copy ne conserve pas les droits d'accès au fichier. Pour cela utiliser de préférence la fonction cp.

stat / lstat

```
my @file_status=stat("/tmp/fichier.txt");
```

Retourne un tableau avec les informations du fichier :

```
 0 dev      device number of filesystem
```

```
 1 ino      inode number
 2 mode     file mode  (type and permissions)
 3 nlink    number of (hard) links to the file
 4 uid      numeric user ID of file's owner
 5 gid      numeric group ID of file's owner
 6 rdev     the device identifier (special files only)
 7 size     total size of file, in bytes
 8 atime    last access time in seconds since the epoch
 9 mtime    last modify time in seconds since the epoch
10 ctime    inode change time in seconds since the epoch (*)
11 blksize  preferred I/O size in bytes for interacting with the file (may
vary from file to file)
12 blocks   actual number of system-specific blocks allocated on disk
(often, but not always, 512 bytes each)
```

La fonction lstat fait la même chose que stat mais fournit les données du lien symbolique au lieu du fichier pointé par le lien.

Commandes système

```perl
chmod(0700,$FILE);
# toujours commencer par 0 en l'absence de sticky bit
my $uid = getpwnam "utilisateur";
# récupération user id
my $gid = getgrnam "groupe";
# récupération group id
my $name = getpwuid($num);
# récupération username à partir du uid
($name, $passwd, $uid, $gid, $quota, $comment, $gcos, $dir, $shell) =
getpwuid($uid);  # récupération des infos du fichier /etc/passwd
($group, $passwd, $gid, $members) = getgrgid($num);
# à partir du gid récupération groupe, et des membres du groupe
chown($uid,$gid,$fichier);
# modification propriétaire fichier
chroot("chemin"); chdir("/");
# chroot (suivre par un chdir pour s'assurer que l'on n'est pas en dehors de
l'arborescence)
umask
# retourne la valeur courante de umask
umask 0666
# retourne la valeur de umask et la position à la valeur 0666
```

Test des fichiers et répertoires

-r	le fichier est en lecture
-w	le fichier est en écriture
-x	le fichier est exécutable

-e	le fichier existe
-Z	le fichier a une taille nulle
-s	le fichier n'a pas une taille nulle (retourne sa taille)
-f	le fichier est un fichier normal
-d	le fichier est un répertoire
-l	le fichier est un lien symbolique
-S	le fichier est une socket
-b	le fichier est un fichier de blocs spéciaux
-u	le fichier a le bit setuid
-g	le fichier a le bit setgid
-k	le fichier a le sticky bit
-T	le fichier est un fichier texte
-B	le fichier est un fichier binaire
-M	Age du fichier en jours quand le script a été lancé
-A	idem pour le dernier accès au fichier
-C	idem pour le dernier changement sur le fichier

Manipulation de répertoire

```
mkdir filename,mode  # crée le répertoire avec les droits spécifiés. Retourne vrai en cas de succès.
rmdir filename       # efface le répertoire si vide. Retourne vrai en cas de succès. En l'absence de filename, utilise $_.
```

Liste des fichiers dans un répertoire

```
opendir(DIR, « répertoire ») or die(« Erreur ouverture répertoire ») ;
@liste_fichier=readdir(DIR) ;
closedir(DIR) ;
```

ou plus simplement avec la fonction glob :

```
@liste_fichier=glob("/tmp/*.txt");
```

Parallelisme

fork

```
use strict;
use warnings;

my $pid = fork();
```

```
if (defined $pid) {
    if ($pid == 0) {
        # Processus enfant
        print "Je suis le processus enfant, mon PID est $$\n";
        sleep(2); # on attend un peu
        print "Processus enfant terminé\n";
    } else {
        # Processus parent
        print "Je suis le processus parent, mon PID est $$, et le PID de mon
enfant est $pid\n";
        waitpid($pid, 0); # Attendre la fin du processus enfant
        print "Processus enfant terminé, continuons\n";
    }
} else {
    die "Impossible de créer un processus enfant : $!";
}
```

multithreads

Nécessite l'installation du module CPAN threads

```
cpan install threads
```

```
use strict;
use warnings;
use threads;
use threads::shared;

# Variable partagée
my $shared_var :shared = 0;

# Fonction à exécuter dans un thread
sub worker {
    my $thread_id = threads->tid();
    {
        lock $shared_var;
        $shared_var++;
        print "Thread $thread_id a incrémenté la variable partagée à
$shared_var\n";
    }
    sleep(1); # Simuler un travail
}

# Créer plusieurs threads
my @threads;
for my $i (1..3) {
    push @threads, threads->create(\&worker);
}
```

```
# Attendre la fin de tous les threads
foreach my $thr (@threads) {
    $thr->join();
}

print "Valeur finale de la variable partagée : $shared_var\n";
```

Astuces

Session interractive ou non

```
sub Is_interactive {
        return -t STDIN && -t STDOUT;
}
```

Dos2Unix

```
perl -i -pe 's/\r\n/\n/g' fichier        Transforme le fichier du format dos au format unix
perl -i.bak -pe 's/\r\n/\n/g' fichier    Transforme le fichier du format dos au format unix et le sauvegarde avant en .bak
```

Suppression d'une ligne particulière dans des fichiers

```
perl -i -lne 'if (/^(19\/08\/2019.+)$/) { } else {print "$_"}' *.csv
Supprime les lignes concernées dans les fichiers *.csv
perl -i.bak -lne 'if (/^(19\/08\/2019.+)$/) { } else {print "$_"}' *.csv
Renomme les fichiers origines en .bak avant transformation
```

Compter un caractère sur chaque ligne d'un fichier

Compte le nombre de ; par ligne d'un fichier

```
perl -lne 'my $count=tr/;/;/; print "ligne $. : $count";' fichier
```

CPAN modules

Des modules supplémentaires sont disponibles sur le site https://www.cpan.org/.

Interface CPAN

Rechercher un module

- Lancer la commande :

```
sudo perl -MCPAN -e shell
```

- Recherche des modules parlant de JSON :

```
cpan[1]> i /json/
```

Installation via cpanm

Installation cpanminus sur Ubuntu

```
sudo apt install cpanminus
```

Installation cpanminus autrement

```
sudo cpan  App::cpanminus
```

Voir dépannage en cas d'échec.

Installer un module

```
sudo cpanm <Nom_du_Module>
sudo cpanm JSON
```

Ne pas oublier le sudo

Désinstaller un module

```
sudo cpanm --uninstall <Nom_du_Module>
```

Installation manuelle

Télécharger le module

```
wget module.tar.gz
tar -xzf module.tar.gz
cd module
```

Compilation/Installation

```
perl Makefile.PL
make
make test
make install
```

JSON

Sauvegarde hash dans un fichier JSON

Ecriture

```
#!/usr/bin/perl
use strict;
use JSON;

my %hash=(
    'toto' => 'toto',
    'titi' => 'titi'
);
# création objet JSON
my $json = JSON->new->utf8->pretty(1);
# conversion hash en JSON
my $json_text = $json->encode(\%hash);
# sauvegarde dans un fichier
open($OUT,">fichier.json") || erreur("Impossible d'écrire dans le fichier
fichier.json : $!");
print $OUT "$json_text";
close $OUT;
```

Lecture

```
#!/usr/bin/perl
use strict;
use JSON;
```

```
# Lire le fichier JSON
open($IN,"<fichier.json") || erreur("Impossible d'ouvrir le fichier
fichier.json : $!");
my $json_text = do { local $/; <$IN> };
close $IN;

# Convertir le JSON en hash Perl
my $hash_ref = JSON->new->utf8->decode($json_text);
print "$hash_ref->{toto}\n";
print "$hash_ref->{titi}\n";
```

Concaténation de 2 fichiers

```
#!/usr/bin/perl
use strict;
use warnings;
use JSON;
use Data::Dumper;

# Initialisation du parser JSON
my $json = JSON->new->utf8;

# Noms des fichiers
my $file1 = 'file1.json';
my $file2 = 'file2.json';

sub readJSONfromfile {
        my $file=$_[0];

        local $/;
        open($IN,"<$file") || die "Impossible d'ouvrir le fichier $file :
$!";
        my $content = <$IN>;
        close $IN;
        return $json->decode($content);
}

# Fonction récursive pour fusionner deux hashes
sub merge_hashes {
    my ($hash1, $hash2) = @_;
    my %result = %{$hash1};  # Copie du premier hash

    for my $key (keys %{$hash2}) {
        if (exists $result{$key}) {
            if (ref($result{$key}) eq 'HASH' && ref($hash2->{$key}) eq
'HASH') {  # Fusion récursive si les deux valeurs sont des hashes
                $result{$key} = merge_hashes($result{$key}, $hash2->{$key});
            } elsif (ref($result{$key}) eq 'ARRAY' && ref($hash2->{$key}) eq
'ARRAY') { # Fusion des tableaux
                push @{$result{$key}}, @{$hash2->{$key}};
```

```
            } else { # Pour les autres cas, on garde la valeur du second
fichier
                $result{$key} = $hash2->{$key};
            }
        } else { # Si la clé n'existe pas dans le premier hash, on l'ajoute
            $result{$key} = $hash2->{$key};
        }
    }
    return \%result;
}

# Lecture et décodage du premier fichier JSON
my $data1=&readJSONfromfile($file1);
print "Données du fichier 1:\n";
print Dumper($data1);

# Lecture et décodage du second fichier JSON
my $data2=&readJSONfromfile($file2);
print "\nDonnées du fichier 2:\n";
print Dumper($data2);

# Fusion des données selon leur type
my $merged_data;

if (ref($data1) eq 'ARRAY' && ref($data2) eq 'ARRAY') { # Fusion de tableaux
    $merged_data = [@$data1, @$data2];
} elsif (ref($data1) eq 'HASH' && ref($data2) eq 'HASH') { # Fusion d'objets
de manière récursive
    $merged_data = merge_hashes($data1, $data2);
} else {
    die "Les fichiers doivent contenir soit deux objets, soit deux tableaux
JSON";
}

print "\nDonnées fusionnées:\n";
print Dumper($merged_data);

# Encodage du résultat en JSON avec indentation
my $result = $json->pretty->encode($merged_data);

# Écriture dans un nouveau fichier
my $output_file = 'merged.json';
open(my $out, '>', $output_file) or die "Impossible de créer $output_file:
$!";
print $out $result;
close($out);

print "\nFusion terminée. Résultat écrit dans $output_file\n";
```

Validation

Validation par rapport à un schéma

Utilisation du module JSON::Validator

```
use JSON;
use JSON::Validator;

        if (-s "$FILE_CONF") {
                open($IN,"$FILE_CONF")||erreur("Impossible d'ouvrir le
fichier $FILE_CONF : $!");
                my $json_text = do { local $/; <$IN> };
                close $IN;
                $cfg = JSON->new->utf8->decode($json_text);

        } else {
                erreur("Fichier de configuration introuvable :
$CYAN$FILE_CONF");
        }
        my $schema = {
                type => 'object',
                required => ['ports', 'services', 'serveurs'],
                properties => {
                        url => { type => 'string', format => 'uri' },
                        ports => {
                                type => 'object'
                        },
                        services => {
                                type => 'array',
                                items => { type => 'string' }
                        },
                        serveurs => {
                                type => 'array',
                                items => {
                                        type => 'object',
                                        required => ['nom' ],
                                        properties => {
                                                nom => { type => 'string' },
                                                IP => { type => 'string',
format => 'ipv4' },
                                                tcp => {
                                                        type => 'array',
                                                        items => {
                                                                type =>
'integer',
                                                                minimum =>
1,
                                                                maximum =>
```

```
65535
                                                        }
                                                },
                                                udp => {
                                                        type => 'array',
                                                        items => {
                                                                type =>
'integer',
                                                                minimum =>
1,
                                                                maximum =>
65535
                                                        }
                                                },
                                                url => {
                                                        type => 'array',
                                                        items => { type =>
'string', format => 'uri' }
                                                }
                                        }
                                }
                        }
                }
        };
        my $validator = JSON::Validator->new;
        $validator->schema($schema);
        my @errors = $validator->validate($cfg);
        if (@errors) {
                for my $err (@errors) {
                        print "$ROUGE\t- $err\n";
                }
                erreur("Erreur fichier de configuration $FILE_CONF");
        }
```

Formats disponibles

Validation en ligne

Validation JSON en ligne

---

module CGI

Il est possible de gérer des requêtes http au travers du module CGI.

- Activer l'exécution CGI au niveau apache :

```
sudo a2enmod cgid
```

- Préciser la localisation des scripts CGI dans la conf du site :

```
        ScriptAlias "/cgi-bin/" "/var/www/html/..../cgi-bin/"
```

- Copier le script dans le répertoire sans oublier de le rendre exécutable

Récupération de paramètres d'une requête http

```perl
#!/usr/bin/perl

use strict;
use warnings;
use CGI;

# Créer un nouvel objet CGI
my $cgi = CGI->new;

# Récupérer les paramètres de la requête POST
my $param1 = $cgi->param('param1');
my $param2 = $cgi->param('param2');

# Imprimer l'en-tête HTTP
print $cgi->header('text/html');

# Afficher les valeurs reçues
print "<html><body>";
print "<h1>Paramètres reçus</h1>";
print "<p>param1: $param1</p>";
print "<p>param2: $param2</p>";
print "</body></html>";
```

Récupération d'un fichier JSON au travers d'une requête http

En affichant les données

```perl
#!/usr/bin/perl

use strict;
use warnings;
use CGI;
use JSON;

# Créer un nouvel objet CGI
my $cgi = CGI->new;

# Récupérer les données de la requête POST
```

```
my $json_text = $cgi->param('POSTDATA');

# Vérifier si des données ont été reçues
if (defined $json_text) {
    # Convertir le JSON en structure de données Perl
    my $data;
    eval {
        $data = decode_json($json_text);
    };
    if ($@) {
        print $cgi->header('application/json', '400 Bad Request');
        print encode_json({ error => "Invalid JSON format" });
        exit;
    }

    # Imprimer l'en-tête HTTP
    print $cgi->header('application/json');

    # Faire quelque chose avec les données reçues (par exemple, les
afficher)
    print encode_json({ received => $data });
} else {
    print $cgi->header('application/json', '400 Bad Request');
    print encode_json({ error => "No data received" });
}
```

En enregistrant les données dans un fichier

```
#!/usr/bin/perl

use strict;
use warnings;
use CGI;
use JSON;

# pretty pour identation
# canonical pour trier en fonction des clefs
my $JSON = JSON->new->utf8->pretty(1)->canonical(1);

my $filename = "data.json";

# Créer un nouvel objet CGI
my $cgi = CGI->new;

# Récupérer les données de la requête POST
my $json_text = $cgi->param('POSTDATA');

# Vérifier si des données ont été reçues
if (defined $json_text) {
        # Convertir le JSON en structure de données Perl
```

```
        my $data;
        eval {
                $data = decode_json($json_text);
        };
        if ($@) {
                print $cgi->header('text/html', '400 Bad Request');
                print "<html><body>";
                print "<p>KO</p>";
                print "</body></html>";
        } else {
                if (open($OUT,">$filename")) {
                        my $json=$JSON->encode($data);
                        print $OUT "$json\n";
                        close $OUT;
                        print $cgi->header('text/html');
                        print "<html><body>";
                        print "<p>OK</p>";
                        print "</body></html>";
                } else {
                        print $cgi->header('text/html', '500 Internal Server
Error');
                        print "<html><body>";
                        print "<p>Unable to write file</p>";
                        print "</body></html>";
                }
        }
} else {
        print $cgi->header('text/html', '400 Bad Request');
        print "<html><body>";
        print "<p>No data received</p>";
        print "</body></html>";
}
```

Compilation en un exécutable

> Fonctionne aussi bien pour obtenir un exécutable sous Linux ou sous Windows après installation de Strawberry Perl par exemple.

Installation modules CPAN

```
cpanm PAR
cpanM PAR::Packer
```

Utilisation

```
pp -o fichier.exe fichier.pl
```

Fichier excel

Librairie Excel::Writer::XLSX

Installation

```
sudo apt install libspreadsheet-writeexcel-perl libexcel-writer-xlsx-perl
```

Utilisation

```
#!/usr/bin/perl
use strict;
use warnings;
use Excel::Writer::XLSX;

# Création du fichier Excel
my $workbook = Excel::Writer::XLSX->new('nouveau_document.xlsx');
my $worksheet = $workbook->add_worksheet('Ma Feuille');

# Création d'un format pour les en-têtes
my $header_format = $workbook->add_format();
$header_format->set_bold();

# Ajout des en-têtes
$worksheet->write(0, 0, 'Nom', $header_format);
$worksheet->write(0, 1, 'Prénom', $header_format);
$worksheet->write(0, 2, 'Âge', $header_format);

# Ajout des données
$worksheet->write(1, 0, 'Dupont');
$worksheet->write(1, 1, 'Jean');
$worksheet->write(1, 2, 42);

$worksheet->write(2, 0, 'Martin');
$worksheet->write(2, 1, 'Sophie');
$worksheet->write(2, 2, 38);

# Fermeture du fichier
$workbook->close();
```

```
print "Le fichier nouveau_document.xlsx a été créé avec succès.\n";
```

Programmation objet

https://djibril.developpez.com/tutoriels/perl/poo/

Caractères UTF-8 sous Windows

Ajouter les lignes suivantes en début de script :

```
use utf8;
use open qw(:std :utf8);
binmode(STDOUT, ":utf8");
```

Terminal Windows

Unitairement

- Avant l'exécution taper

```
chcp 65001
```

PowerShell

unitairement

- Avant l'exécution taper

```
$OutputEncoding = [System.Text.Encoding]::UTF8
[Console]::OutputEncoding = [System.Text.Encoding]::UTF8
```

- Vérification

```
[Console]::OutputEncoding
```

De façon persistante

- exécuter

```
if (!(Test-Path $PROFILE)) {
    New-Item -Type File -Path $PROFILE -Force
```

```
}
echo $PROFILE
```

- puis éditer le fichier Microsoft.PowerShell_profile.ps1 pour y ajouter

```
# Configuration pour UTF-8
$OutputEncoding = [System.Text.Encoding]::UTF8
[Console]::OutputEncoding = [System.Text.Encoding]::UTF8
```

Dépannage

Debug

```
perl –d fichier.pl # lancement en mode debug
# !/usr/bin/perl –w   # le –w permet d'afficher les warning.
```

Can't locate common.pm in @INC

Par défaut le répertoire courant n'est pas dans la variable @INC, il faut donc éventuellement l'ajouter avant l'appel du package à inclure :

```
#!/usr/bin/perl
use warnings;
use strict;
use Cwd;
use File::Basename;

my $BINDIR;
my $PRGNAME;
my $EXT;

BEGIN {
        ($PRGNAME,$BINDIR,$EXT)=fileparse($0,qr/\.[^.]*/);
        unshift(@INC,$BINDIR);
}

use common;
```

Impossible d'installer cpanminus

Lors de l'installation obtention de l'erreur suivante :

```
Couldn't untar local-lib-2.000024.tar: 'Cannot allocate memory'
```

Tenter l'installation de la façon suivante :

```
sudo curl -L http://cpanmin.us | perl - --sudo App::cpanminus
```
