A Quick Guide To PERL

This is a Quick reference Guide for PERL 5.8.6 programming. Perl definition is given by its creator, Larry Wall: “Perl is a language to get your job done” and he added “There is more than one way to do it”!

This guide is not exhaustive, its purpose is to give a few essential reminder to the Perl syntax, but basic knowledge of Perl programming is required.

To find help about a Perl function or keyword use perldoc:

perldoc -f split
perldoc -q FAQ

For more information about Perl in general see:
http://www.perl.org

References
For more information on Perl syntax you can refer to O'Reilly's book “Programming Perl, 3rd edition”.

Structure of a Perl script
#!/usr/bin/perl
Structure of a Perl script
first line of a Perl script
call subroutine 
statement list
statement list
exit 0;
last line (optional)
A Quick Guide To PERL

which perl gives the path to the Perl executable
(coULD be /usr/local/bin/perl)

Variables Scalars ($)
In Perl the variables are not strictly typed (no integer, char, float, reference, objects etc...) This is a strength and a weakness of Perl.

$var = "any content";
assign a string
print scalar "aa"
my $var;
declare a variable as global lexically
our $var;
declare a variable as global lexically
local $var;
declare a variable as local dynamically

Variables Arrays (or Lists) (@)
Array or lists is an indexed collection of values, the first index starts at position zero.

@var = ("aa","bb","cc");
assign an array of 3 elements
print scalar "aa"
push @var;  add an element to @var (right)
$var = "any content";
assign a string
my $var;
declare a variable as local
exchange values
assign several scalars at once
($a,$b,$c) = 41,42,"Jo"
assign several scalars at once
($lt,$rt) = ($rt,$lt)
swap values
my $var;
declare a variable as local lexically
our $var;
declare a variable as global lexically
local $var;
declare a variable as local dynamically

Variables Hashes (%)
A hash is a structure where a key is associated to a value

%var = {"red"=>x0000FF,
assign values to 3 hash elements
"blue"=>x0FF00,
"green"=>x00FF000};
print scalar "blue"
%var{"yellow"} = xFFFF00;
add a new hash element
Get value x0000FF = 255
@ex = %var;
assign an array of 3 elements
print scalar "aa"
$var = $ex;
convert hash to array
%var = @ex;
convert array to hash
print $var[1];
print scalar "bb"
%var = @ex;
convert array to hash
print scalar "aa"
print $var[0];
print scalar "bb"
print %var;
give the list of keys for the %var
print scalar "aa"
give the list of values for the %var
print scalar "aa"
delete the hash element
delete $var{"yellow"}
delete hash element

Special Variables
Perl has a large collection of special variables. Here is a short extract.

$_
default input
$@ in a subroutine contains the list of arguments
$n process ID
$ record separator (default = \n)
$e eval error or exception
$^O contain arguments of the command-line

$ARGV[0] first argument
$ENV contain environment variables
@INC contain list of directories for modules to import

Control Operators
&& logical AND, OR and NOT
|| string comparison
! numerical comparison
lt ge ne eq cmp
Example:
if ($var == 42) { print "$var is numeric";}
else if ($var eq "XLII") { print "$var is a string";}
else { print "$var is not equal to 42";}

Generally:
if (expr1) {
statement list1
if (expr1) execute
if (expr1) statement list1

elsif (expr2) {
statement list2
execute elsif and else (like if)

else { executes list3
execute else (like if)

statement if (expr)
reverse if, execute statement if expr is true (also with unless, while, until)

unless (expr) {
execute statement unless expr is true, handle elsif and else (like if)
}

Loops
while (expr) {
repeat statement while expr is true

do {
repeat statement a certain number of times

until (expr)
}
end loops (while, for, etc...)

for (init; expr; incr){
repeat statement a certain number of times

statement list
jump to next item in the loop
redo;
restart loop with current item

Example: prints 1 to 10
for($i=1;$i<=10;$i++){
print "$i
";
}
Example: prints each element of array @list
foreach $index (@list){
print $index;
}
Subroutines, example:

```perl
sub add_it {
    local ($a,$b)=@_;  # get arguments
    $var = $a+$b;     # sum the values
    return $var;      # return the result
}
```

$result = &add_it(3,5);  # call subroutine with arguments, $result contains 8.

File Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>open()</td>
<td>open a file Handler</td>
</tr>
<tr>
<td>close()</td>
<td>close a file Handler</td>
</tr>
</tbody>
</table>

Example:

```perl
open (FH, "filename");  # open file filename for reading
while (<FH>) {  
    $text .= $_;  # concatenate $_ in $text
}  
close(FH);  # close filehandle, $text contains the content of file filename
```

System calls

```perl
system("ls -l");  # execute a system command and continue the current Perl script
exec("rm tmp");  # execute a system command and quit the current Perl script
```

Regular Expressions

Please use the QuickGuide to Perl Regular Expressions in the same series.

Perl modules

http://www.cpan.org  # CPAN repository for Perl modules.

use Mymodule;  # preload a module or pragma at compilation time
require Mymodule;  # preload a module at execution time

Perl looks for the real name of the module “Mymodule.pm”

This document was written and designed by Laurent Falquet and Vassilios Ioannidis from the Swiss EMBnet node and being distributed by P&PR Publications Committee of EMBnet.

EMBnet - European Molecular Biology Network - is a bioinformatics support network of bioinformatics support centers situated primarily in Europe. Most countries have a national node which can provide training courses and other forms of help for users of bioinformatics software.

You can find information about your national node from the EMBnet site:

http://www.embnet.org/

A Quick Guide To PERL
First edition © 2005