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 FAQkeyword
```

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`	first line of a Perl script*
`...`	
`statement list`	
`...`	
`exit 0;`	last line (optional)
*`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.

`$var = "any content";`	assign a string
`$value = 42;`	assign a number
`($a,$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 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 $var[0];`	print scalar "aa"
`print $var[1];`	print scalar "bb"
`push(@var, $new);`	add an element to @var (right)
`$getr=pop(@var);`	remove last element of @var (right)
`unshift(@var, $new);`	add an element to @var (left)
`$getl=shift(@var);`	remove first element of @var (left)
`@rvar=reverse(@var);`	return the reverse order of the elements of @var
`@svar=sort(@var);`	return the sorted elements of @var (string sort)
`split(/PATTERN/, $var);`	change a string to a list of elements split by a 'PATTERN'
`join("MOTIF", @var);`	join elements of @var with a 'MOTIF' to form a single string
`$size = @var;`	$size contains the number of elements of the array @var

Variables Hashes (%)

A hash is a structure where a key is associated to a value

`%var = ("red"=>x0000FF, "blue"=>xFF0000, "green"=>x00FF00);`	assign values to 3 hash elements
`print $var{"red"};`	contain value x0000FF = 255
`$var{"yellow"}=xFFFF00;`	add a new hash element
`@ex = %var;`	convert hash to array
`%var = @ex;`	convert array to hash
`print keys(%var);`	give the list of keys for the %var
`print values(%var);`	give the list of values for the %var
`print each(%var);`	same as values
`delete $var{"yellow"}`	delete the 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
`$$`	process ID
`$/`	record separator (default = \n)
`$@`	eval error or exception
`@ARGV`	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
`< > <= >= != == <=>`	numerical comparison
`lt gt le ge ne eq cmp`	string comparison

Example:

```
if ($var == 42) { print "$var is numeric";}
elsif ($var eq "XLII") { print "$var is a string";}
else {print "$var is not equal to 42";}
```

Generally:

```
if (expr1) {                  if expr1 is true execute list1
     statement list1
}
elsif (expr2) {               else if expr2 is true execute list2
     statement list2          (can have many elseif)
}
else {                        else executes list3
     statement list3
}

statement if (expr)           reverse if, execute statement if
                              expr is true (also with unless,
                              while, until)
unless(expr) {                execute statement unless expr is
     statement list           true, handle elsif and else (like if)
}
```

Loops

```
while(expr) {                 repeat statement while expr is true
     statement list
}
do {                          repeat statement until expr is true
     statement list
} until(expr)
for(init; expr; incr){        repeat statement a certain number
     statement list           of times
}
last;                         end loops (while, for, etc...)
next;                         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\n";
}
```

Example: prints each element of array @list

```
foreach $index (@list){
     print $index;
}
```

Subroutines, example:

`sub add_it {`	create a subroutine
`      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

`open` *`HANDLE, filename`*	open a file Handler
`close` *`HANDLE`*	close a file Handler

Example:

`open (FH, “filename”);`	open file filename for reading
`while (<FH>) {`	read each record (line) and store in $_
`      $text .= $_;`	concatenate $_ in $text
`}`	
`close(FH);`	close filehandle, $text contains the content of file filename
`open(FH, “>filename”);`	open filename for output in write
`open(FH, “>>filename”);`	open filename for output in concatenate

Example:

`open(FH, “ls -l |”);`	pipe allow to grab command-line output
`while (<FH>) {`	read and store the output of “ls -l”
`      $filelist .= $_;`	
`}`	

Special Handlers

`<STDIN>`	read from standard input (usually keyboard)
`<STDOUT>`	write to standard output (usually screen)
`<STDERR>`	write to standard error (usually screen)

File Tests

```
if (-e $filename) { open(READ, $filename); }
```

Some possible tests:

`-r`	readable
`-w`	writable
`-x`	executable
`-o`	belong to user
`-e`	exist
`-z`	zero size (file exist)
`-s`	nonzero size
`-f`	file
`-d`	directory
`-l`	symlink
`-T`	text file
`-A`	accessed in days
`@var=stat($filename);`	get full info on files

String Functions

`$var=”my”x4;`	$var contains “mymymymy”
`$new=$var.$var;`	concatenate 2 strings
`$var.=$new;`	assign & concatenate, same as `$var=$var.$new;`
`chop($var);`	delete last char of $var
`chomp($var);`	delete \n if last char of $var
`$c=substr($var,3,5);`	get 5 characters of string $var starting from position 3.
`print “Hello world\n”;`	print a string
`printf(“%10s %4d %5.2f\n”, $s,$i,$r);`	similar as “C/C++” print formatting

System calls

`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”

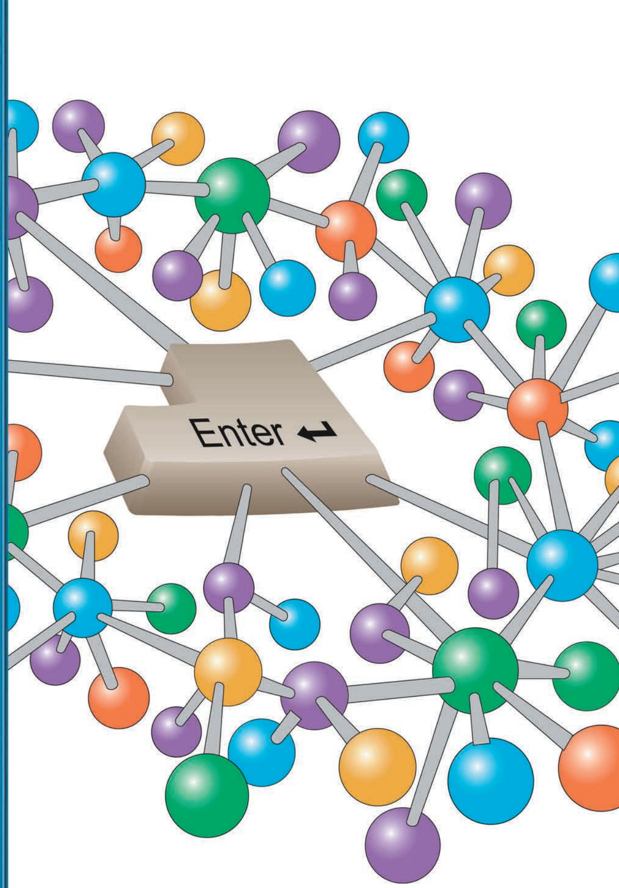
This document was written and designed by Laurent Falquet and Vassilios Ioannidis from the Swiss EMBnet node and being distributed by P&PR Publications Committee of EMBnet.

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



A Quick Guide PERL