

Bioinformatics Programming 2013

# Perl – Basics

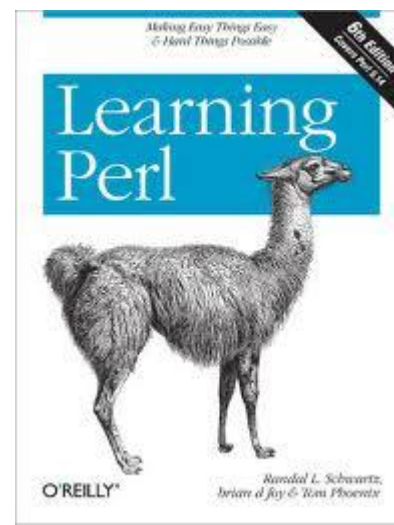
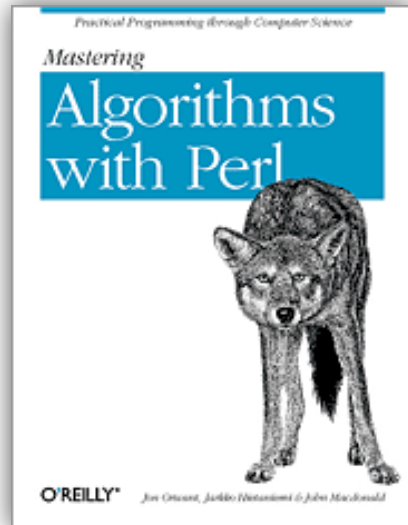
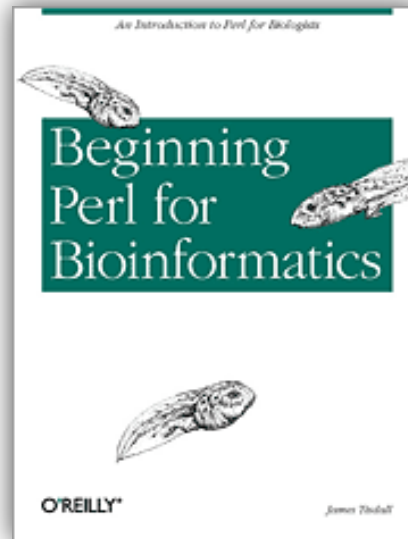
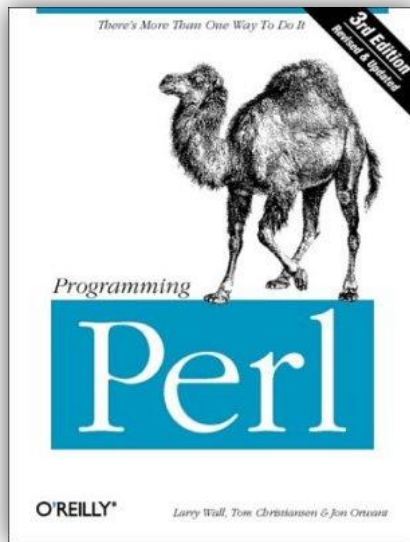
**Chia-Lang Hsu (許家郎)**

Department of Life Science,  
National Taiwan University

# Why is Perl so heavily used in Bioinformatics ?

- Perl is popular in bioinformatics because it is originally a text-processing language.
- **Text is King** - Perl makes it easy to:
  - implement NLP and bio-informatics algorithms,
  - extract textual data,
  - generate textual data.
- **The Language Isn't (Half) Bad** - It also has the benefits of having:
  - a decent expressiveness,
  - a relatively low learning but also a decent performance.
- **Portability and Extensibility Made Easy**
  - is portable across many platforms,
  - comes with a very large library of extensions.

# References



# You will learn from this course ...

- 2013/08/01:
  - Basic Perl data structure & statement
- 2013/08/20:
  - Regular Expression
- 2013/08/27:
  - Perl Packages
  - BioPerl
  - GD

# Install Perl for Windows

DWIM Perl

Windows

Linux

Earlier releases



Legal

<http://dwimperl.com/>

## DWIM Perl for Windows



DWIM Perl for Windows is a Strawberry Perl derivative for Windows. It contains everything you will need for your Perl development.

It contains:

- [Strawberry Perl 5.14.2.1 RC](#) which itself is a standard Perl with several extensions already installed.
- A large part of [Task::Kensho](#), a list of recommended packages.
- Padre, [the Perl IDE 0.94](#).
- `Module::Version 0.12` so you can use `mversion` to check which version of each module you have.
- [Moose 2.0402](#), the post modern Object System.
- [Dancer 1.3092](#) to build a light-weight web application.
- [Plack and plackup 0.9985](#) to serve your web pages.
- [Perl::Critic 1.117](#), to police yourself.
- [Perl::Tidy 20101217](#), to keep your code nice.
- [DateTime 0.72](#) to make it easy to deal with dates and time.
- [SQLite 1.35](#), to hold your data tight.
- [MySQL 4.020](#), [PostgreSQL 2.18.1](#) and [DBD::ODBC 1.31](#) drivers.
- Lots of additional modules... (see details in the [README](#) file.)

Download 

The current release is [Dwimperl-5.14.2.1-v7-32.exe](#), released on 2012.02.12



Want to get notified when a new release is published?

The [Perl Weekly newsletter](#) includes announcements of DWIM Perl and major Perl projects as they are made available by their developers.

New releases will be also announced on our new [Facebook page](#) and on our [Google+ Page](#).

To get started with DWIM Perl, check out the [Perl Tutorial](#) written by [Gabor Szabo](#), the maintainer of DWIM Perl.

# Run Padre and execute my first perl script

The image shows the Padre IDE interface. The main editor window displays a Perl script named `hello_world.pl` with the following code:

```
1 use strict;
2
3 my $b = shift;
4
5 my $a = "Hello, World!!";
6
7 print $a."\n";
8 print $b."\n";
```

A dialog box titled "執行設定" (Execution Settings) is open, with the command line field containing `perl hello_world.pl abc`. The dialog has buttons for "O 確認" (OK) and "C 取消" (Cancel).

A terminal window titled "perl hello\_world.pl abc" shows the output of the script:

```
Hello, World!!
abc
Press any key to continue . . .
```

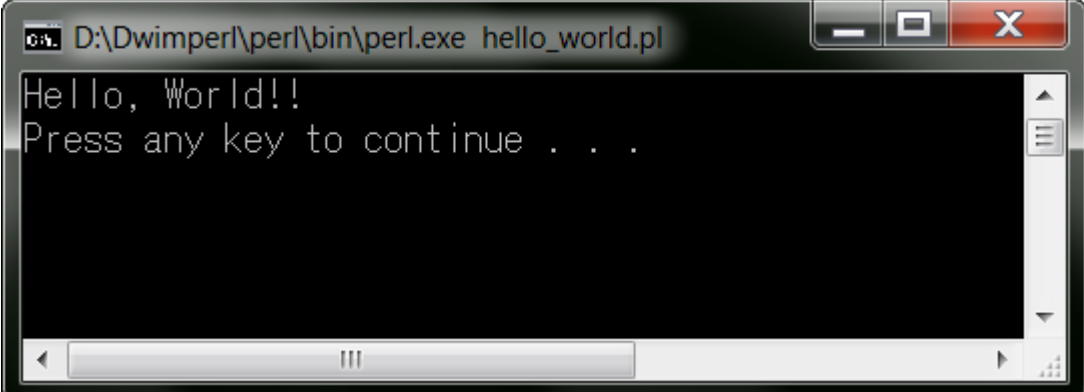
The Padre IDE window has a menu bar with options: F 檔案 (File), E 編輯 (Edit), S 搜尋 (Search), V 檢視 (View), Perl, Refactor, R 執行 (Run), D 除錯 (Debug), T 工具 (Tools), W 視窗 (Windows), H 幫助 (Help). The status bar at the bottom indicates "Perl(&P) WIN 8,14 87% R/".

# 1<sup>st</sup> Perl script

```
use strict;
```

```
print "Hello, World!!\n";
```

Save file as \*.pl and Press F5 to execute this script



A screenshot of a Windows command prompt window. The title bar shows the command: `D:\Dwimper\perl\bin\perl.exe hello_world.pl`. The window content displays the output of the script: `Hello, World!!` followed by `Press any key to continue . . .`. The window has standard Windows window controls (minimize, maximize, close) in the top right corner and a scrollbar on the right side.

# You are familiar with Python, right?

---

其實Perl跟Python長得有點像，  
但還是有點不一樣



# Data type – string and numeric

## Perl

- `my $a = 1;`
- `my $b = 'abc';`
- `my $c = "ABC";`
- `my $d = $b . $c;`  
>> abcABC
- `my $e = $b x 3;`  
>> abcabcabc

## Python

- `a = 1`
- `b = 'abc'`
- `c = "ABC"`
- `d = b + c`
- `e = b * 3`

- ① A variable name starts with “\$”
- ② Use “my” to declare variables to be local to the enclosing block .
- ③ Each command line ends with a **semicolon** (;)

# What is the difference between “ and “” ?

- Try to execute the following scripts:

```
use strict;

my $a = “John”;

print '$a is a good man.\n';
print "$a is a good mam.\n";
print $a . “ is a good man.\n”;
print "\$a is a good man.\n";
```

# Data structure – list or array (1)

## Perl

- `my @A1;`
- `my @A2 = ('a', 'b', 'c');`
- `my @A3 = (1, 3, 5);`
- `my $v = $A3[1];`
- `$A3[2] = 0;`  
`>> (1, 3, 0)`
- `my @A4 = (@A2, @A3);`  
`>> (1, 3, 0, 'a', 'b', 'c')`
- `my @a = (1, 3, 5, 7, 9);`
- `my @b = @a[1..3];`

## Python

- `A1 = []`
- `A2 = ['a', 'b', 'c']`
- `A3 = [1, 3, 5]`
- `v = A3[1]`
- `A3[2] = 0`
- `A4 = A2 + A3`
- `a = [1, 3, 5, 7, 9]`
- `b = a[1:4]`

- ① A array/list name starts with “@”.
- ② Use `print join("\s", @array)` to list the array elements.

# Data structure – list or array (2)

## Perl

- `my @a = (1, 7, 5, 3, 9)`
- `push(@a, 0);`  
`>> (1,7,5,3,9,0)`
- `scalar(@a);`  
`>> 6`
- `shift(@a);`  
`>> (7,5,3,9,0)`
- `pop(@a);`  
`>> (7,5,3,9)`
- `@a = sort{$a<=>$b}(@a);`  
`>> (3,5,7,9)`
- `@a = sort{$b<=>$a}(@a);`  
`>> (9,7,3,5)`

## Python

- `a = [1, 7, 5, 3, 9]`
- `a.append(0)`
- `len(a)`
- `a.pop(0)`
- `a.pop()`
- `a.sort()`
- `a.sort().reverse()`

# Data structure – list or array (3)

## Perl

- `my @a = (1, 7, 5, 3, 9);`
- `@a = reverse(@a);`  
`>> (9,3,5,7,1)`
- `splice(@a, 2, 0);`  
`>> (9,3,7,1)`
- `splice(@a, 1, 2);`  
`>> (9,1)`
- `$#a`  
`>> 1 # the last index`

## Python

- `a = [1, 7, 5, 3, 9]`
- `a.reverse()`
- `del a[2]`
- `a[1:3] = []` or `del a[1:3]`

# Data structure – hash or dictionary

## Perl



- `my %a;`
- `my %a = ('spam' => 1, 'eggs' => 3);`
- `$a{'ice'} = 6;`
- `my @k = keys(%a);`  
`>> ('spam', 'eggs', 'ice')`
- `my @v = values(%a);`  
`>> (1, 3, 6)`
- `scalar(keys(%a))`  
`>> 3`
- `delete($a{'eggs'});`  
`>> ('spam' => 1, 'ice' => 6)`

## Python

- `a = {}`
- `a = {'spam': 1, 'eggs': 3}`
- `a['ice'] = 6`
- `k = a.keys()`
- `v = a.values()`
- `len(a)`
- `del a['eggs']`

# Print array and hash data

## Array

- `my @a = (1, 7, 5, 3, 9);`
- `foreach my $e (@a)`  
`{`  
 `print $e.“\n”;`  
`}`
- Or
- `print join(“\s”, @a).“\n”;`  
`print join(“\|”, @a).“\n”;` → `1|7|5|3|9`  
`print join(“\t”, @a).“\n”;`  
`print join(undef, @a).“\n”;` → `17539`

## Hash

- `my %a = ('spam' => 1,`  
 `'eggs' => 3,`  
 `'ice' => 6);`
- `foreach my $key (keys %a)`  
`{`  
 `my $value = $a{$key};`  
 `print “$key\t$value\n”;`  
`}`

- ① “\n”: 換行符號
- ② “\s”: 空白(space)
- ③ “\t”: Horizontal tab
- ④ `join`(DELIMITER, ARRAY) # convert array to string

# Blocks

## Perl

- **Block 1 {**  
    **Block 2 {**  
        **Block 3 {**  
  
        } End Block 3  
    } End Block 2  
} # End Block 1
- Indent is not required.

## Python

- **Block 1 :**  
    **Block 2 :**  
        **Block 3 :**  
  
        End Block 3  
    End Block 2  
End Block 1
- Indent is required.



# Statements

## if ... elsif ... else...

- `my $a = 10;`
- `if ($a > 10)`
  - `{`
    - `print "Block 1\n";`
  - `}`
  - `elsif ($a == 10)`
    - `{`
      - `print "Block 2\n";`
    - `}`
  - `else`
    - `{`
      - `print "Block 3\n";`
    - `}`

```
>> Block 2
```

## unless ... elsif ... else

- `my $a = 10;`
- `unless ($a > 10)`
  - `{`
    - `print "Block 1\n";`
  - `}`
  - `elsif ($a == 10)`
    - `{`
      - `print "Block 2\n";`
    - `}`
  - `else`
    - `{`
      - `print "Block 3\n";`
    - `}`

```
>> Block 1
```

# Loops – for, foreach, until

## for

- `my @a = (2, 4, 6, 8, 10);`
- `for my $i (0 .. $#a)`  
  {  
    `print "$a[$i]\n";`  
  }
- `for (my $i=0; $i<=#a; $i++)`  
  {  
    `print "$a[$i]\n";`  
  }

## foreach

- `foreach my $i (@a)`  
  {  
    `print "$i\n";`  
  }

- ① `$#array` returns the last index of a array
- ② `$a++` → `$a+=1`

# Loops – while & until

## while

- my \$a = 10;
- **while** (\$a > 0)
  - {
  - print "Bingo!\n";
  - \$a --;
  - }

## until

- **until** (\$a == 0)
  - {
  - print "Bingo!\n";
  - \$a --;
  - }

# File access

## Read File

- `my $filename = "input.txt";`
- `open(IN, "$filename") or die ("Error!\n");`
- Method 1:  
`my @input = <IN>;`
- Method 2:  
`while (my $line = <IN>)`  
`{`  
`do something ...`  
`}`
- `close(IN);`

## Write File

- `my $filename = "output.txt";`
- `open(OUT, ">$filename");`
- `print OUT "write something\n";`
- `close(OUT);`

# Example 1: counting DNA nucleotides

## Input:

A DNA fasta file. For example,

```
>this is a fasta header
```

```
ATCGGCGGAGAAaacccttgggccccgggggTGGCG
```

```
aacccttggcgATGGTGACGATAGCAGATaagatag
```

```
TGGAGTaaGGTAT
```

## Output:

For example,

A	22
---	----

T	14
---	----

C	17
---	----

G	32
---	----

# Script for Example 1

```
1 use strict;
2
3 my $file = "example.fasta";
4
5 open IN, "$file";
6 my @seq = <IN>;
7 my $header = shift @seq; # remove the first element (fasta header) of the array
8
9 my %counter;
10 foreach my $seq (@seq)
11 {
12     chomp($seq); # remove any newline character from the end of a string
13     $seq = uc($seq); # convert string to uppercased version
14                 # similar function "lc" => convert string to lowercased version
15     my @dna = split(undef, $seq); # convert a string into a array
16     foreach my $dna (@dna)
17     {
18         $counter{$dna} ++;
19     }
20 }
21
22 while (my ($key, $value) = each %counter)
23 {
24     print "$key\t$value\n";
25 }
26
```

# Functions for processing string

- `my $s = "ATCGggta";`
- `print uc($s)`  
`>> ATCGGGTA`
- `print lc($s)`  
`>> atcgggta`
- `print length($s);`  
`>> 8`
- `print ucfirst("hello");`  
`>> Hello`
- `print lcfirst("HELLO");`  
`>> hELLO`
- `my $a = "abcdefgfijklmnop";`
- `my $b = substr($a, 3, 4);`
- `my $c = index($a, $b);`  
`>> 3`
- `my $url = "www.ym.edu.tw";`
- `my @url = split(".", $url);`  
`>> ('www', 'ym', 'edu', 'tw')`
- `my $line = join("_", @url);`  
`>> www_ym_edu_tw`

- ① `substr(STR, OFFSET, LENGTH);`
- ② `Index(STR, SUBSTR, [POSITION]);`

# Example 2: counting common words

## Input:

Two gene lists:

```
"TP53 NRG1 ATP5B ATF2 BIRC6 LARP7 EIF4H2 SFRS2"
```

```
"FBXL18 TLE3 DICS1 LARP7 ATAD2 NRG1 ATP2A TP53 PANK2"
```

## Output:

Common genes include: TP53, LARP7, NRG1



# Script for Example 2

```
1 use strict;
2
3 my $list1 = "TP53 NRG1 ATP5B ATF2 BIRC6 LARP7 EIF4H2 SFRS2";
4 my $list2 = "FBXL18 TLE3 DICS1 LARP7 ATAD2 NRG1 ATP2A TP53 PANK2";
5
6 my @list1 = split(" ", $list1);
7 my @list2 = split(" ", $list2);
8
9 my %list1;
10 my %commons;
11
12 foreach my $gene (@list1)
13 {
14     $list1{$gene} = 0;
15 }
16
17 foreach my $gene (@list2)
18 {
19     if (exists $list1{$gene}) ## used to check whether an key in hash exists.
20     {
21         $commons{$gene} = 0;
22     }
23 }
24
25 print "Common genes include: ".join("\, ", keys(%commons))."\n";
26
```

# Subroutine

## Ex. 1

- `sub max`  
  {  
    `my ($a, $b) = @_;`  
    `if ($a > $b)`  
    {  
      `return($a);`  
    }  
    `else`  
    {  
      `return($b);`  
    }  
  }

- `my ($a, $b) = (10, 20);`
- `my $max = max($a, $b);`

## Ex. 2

- `sub max`  
  {  
    `my $a = shift; #等於shift @_`  
    `my $b = shift;`  
    `if ($a > $b)`  
    {  
      `return($a);`  
    }  
    `else`  
    {  
      `return($b);`  
    }  
  }

# Reference in Perl

A reference is a scalar value that points to a memory location that holds some type of data.

## Array

- `my @a = (1, 2, 3);`
- `my $a = [1, 2, 3];`  
# array ref
- `$a[0] >> 1`
- `$a->[0] >> 1`
- `my @b = @$a 或 @{$a};`
- `my $b = \@a;`

## Hash

- `my %a = ('a' => 1,  
          'b' => 2);`
- `my $a = {'a' => 1,  
          'b' => 2}; # ref`
- `$a{'a'} >> 1`
- `$a->{'a'} >> 1`
- `my %b = %a 或 %{$a};`
- `my $b = \%a;`

# Rewrite script for Example 2

```
1 use strict;
2
3 my $list1 = "TP53 NRG1 ATP5B ATF2 BIRC6 LARP7 EIF4H2 SFRS2";
4 my $list2 = "FBXL18 TLE3 DICS1 LARP7 ATAD2 NRG1 ATP2A TP53 PANK2";
5
6 my @list1 = split(" ", $list1);
7 my @list2 = split(" ", $list2);
8
9 my $list1 = {};
10 my $commons = {};
11
12 foreach my $gene (@list1)
13 {
14     $list1->{$gene} = 0;
15 }
16
17 foreach my $gene (@list2)
18 {
19     if (exists $list1->{$gene}) ## used to check whether a key in hash exists.
20     {
21         $commons->{$gene} = 0;
22     }
23 }
24
25 print "Common genes include: ".join("\, ", keys(%{$commons}))."\n";
26
```

# High dimensional array and has

## Array in Array

- `my $a = [];`
- `$a->[0] = ["baseball", "basketball"];`
- `$a->[1] = ["volleyball", "football"];`
- `$a->[2] = ["soccer", "swimming"];`
  
- `print $a->[1]->[1], "\n";`

## Hash in Hash

- ```
my $h = {  
  'sales' =>  
    {  
      'Brown' => 'Manager',  
      'Smith' => 'Salesman',  
      'Albert' => 'Salesman',  
    },  
  'marketing' =>  
    {  
      'Penfold' => 'Designer',  
      'Jurgenes' => 'Manager'  
    }  
};
```
  
- `print $h->{'sales'}->{'Albert'}, "\n";`

# Example 3: query data

## Input:

Search A gene's other information from a text file.

“database.txt” is a tab-delimited file.

| 1 | Symbol | GeneID | MIM    | Location    | Full name                                     |
|---|--------|--------|--------|-------------|-----------------------------------------------|
| 2 | TP53   | 7157   | 191170 | 17p13.1     | tumor protein p53                             |
| 3 | MDM2   | 4193   | 164785 | 12q14.3-q15 | MDM2 oncogene, E3 ubiquitin protein ligase    |
| 4 | AKT1   | 207    | 164730 | 14q32.32    | v-akt murine thymoma viral oncogene homolog 1 |
| 5 | CDKN2A | 1029   | 600160 | 9p21        | cyclin-dependent kinase inhibitor 2A          |
| 6 | PML    | 5371   | 102578 | 15q22       | promyelocytic leukemia                        |
| 7 | RB1    | 5925   | 614041 | 13q14.2     | retinoblastoma 1                              |
| 8 | RPL11  | 6135   | 604175 | 1p36.1-p35  | ribosomal protein L11                         |

## Output:

Get the “Location” of “AKT1” >> 14q32.32

# Script for Example 3

```
1 use strict;
2
3 my $db_file = "database.txt";
4
5 my $db = {};
6
7 open IN, $db_file;
8 my @data = <IN>;
9 chomp @data;
10 my $head = shift @data;
11 my @head = split("\t", $head);
12
13 ### Database construction
14 foreach my $line (@data)
15 {
16     my @list = split("\t", $line);
17     for my $i (1 .. $#head)
18     {
19         $db->{$list[0]}->{$head[$i]} = $list[$i];
20     }
21 }
22
23 ### Query
24 my $query = "AKT1";
25 my $col = "Location";
26
27 print $db->{$query}->{$col}, "\n";
```