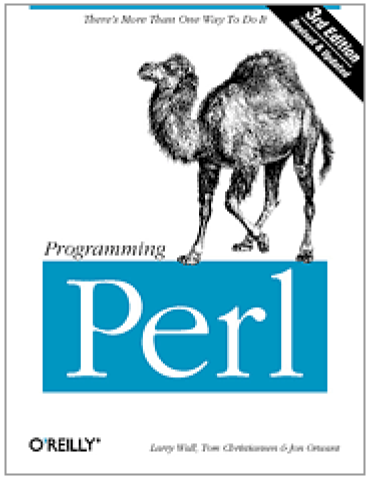
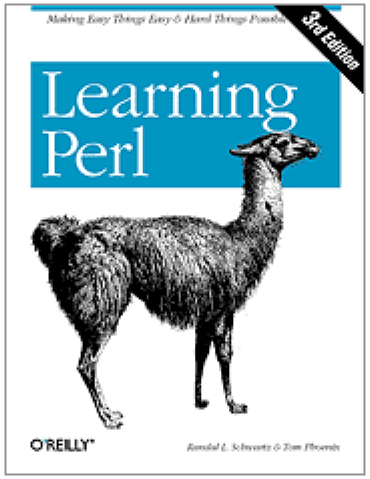
Perl Programming

Slides Contents

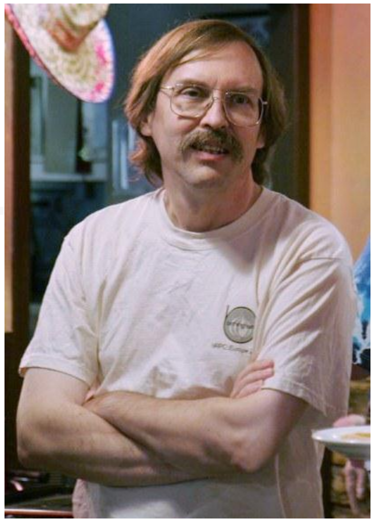



Introduction

There's More Than One Way To Do It

Making Easy Things Easy and Hard Things Possible

Introduction (1)



- ❑ Scripting language
  - Perl、Python、Ruby、PHP(?)、…
- ❑ PERL
  - **P**ractical **E**xtraction and **R**eport **L**anguage
  - Created by **Larry Wall** and first released in 1987
- ❑ Useful in
  - Text manipulation
  - Web development
  - Network programming
  - GUI development
  - System Prototyping
  - anything to replace C, shell, or whatever you like

Introduction (2)

- Built-in in most unix-like operation systems, but…
  - /usr/ports/lang/perl5.{12,14}
- Compiled and interpreted
  - Efficient
- Syntax Sugar
  - die unless $a == $b;
- Modules
  - Object oriented
  - CPAN
- Perl6
  - http://dev.perl.org/perl6/
  - Pugs – http://www.pugscode.org/
  - Parrot – http://www.parrotcode.org/ (/usr/ports/lang/parrot)

Introduction – Hello World (1)

```
#!/usr/bin/perl
use warnings;
use strict;
# My First Perl Program
print “Hello”, “ world!\n”; # Comments
```

- `#!/usr/bin/perl`
  - The location of perl interpreter
- `use warnings;`
  - Enable all warnings
- `use strict;`
  - It is nice to be
- `# My First Perl Program`
  - Comment, to the end of line
- `print “Hello”, “ world!\n”; # Comments`
  - Built-in function for output to STDOUT
- C-like “;” termination

Introduction – Hello World (2)

- name.pl

```
#!/usr/bin/perl
print "What is your name? ";
chomp($name = <STDIN>);
print("Hello, $name!\n");
```

```
$name = <STDIN>;
chomp $name;
```

chomp is not pass by value

Value interpolation into string

- scalar variable = <STDIN>
  - Read *ONE* line from standard input
- chomp
  - Remove trailing "\n" if exists
- Variables are global unless otherwise stated
- Run Perl Program

```
% perl name.pl          (even no +x mode or perl indicator)
% ./name.pl                (Need +x mode and perl indicator)
```

Scalar Data

1 + 1 == 10

Scalar Data (1) – Types

- Use prefix ‘$’ in the variable name of a scalar data
  - $scalar_value
- Numerical literals
  - Perl manipulates numbers as double-decision float point values
  - Float / Integer constants, such as:
    - 12, -8, 1.25, -6.8, 6.23e23, 0377, 0xff, 0b00101100
- Strings
  - Sequence of characters
  - Single-Quoted Strings (No interpolation)
    - ‘$a\n is printed as is’, ‘don\’t’
  - Double-Quoted Strings (With interpolation)
    - “$a will be replaced by its value.\n”
    - Escape characters
      - \a, \b, \n, \r, \t

Scalar Data (2) – Operators

- Operators for Numbers
  - Arithmetic
    - +, -, *, /, %, **
  - Numerical comparison
    - <, <=, ==, >=, >, !=
- Operators for Strings
  - Concatenation "."
    - "Hello" . " " . "world!" ➔ "Hello world!"
  - Repetition "x"
    - "abc" x 4 ➔ "abcabcabcabc"
  - Comparison
    - lt, le, eq, ge, gt, ne
- perlop(1)

Scalar Data (3) – Assignments

- Operators for assignment
  - Ordinary assignment
    - $a = 17
    - $b = “abc”
  - Short-cut assignment operators
    - Number: +=, -=, *=, /=, %=, **=
    - String: .=, x=
      - $str .= “.dat” ➔$str = $str . “.dat”
  - Auto-increment and auto-decrement
    - $a++, ++$a, $a--, --$a

Scalar Data (4) – Conversion

- Implicit conversion depending on the context
  - Number wanted? (3 + “15”)
    - Automatically convert to equivalent numeric value
    - Trailing nonnumeric are ignored
      - “123.45abc” ➔ 123.45
  - String wanted?
    - Automatically convert to equivalent string
    - “x” . (4 * 5) ➔”x20”

Scalar Data (5) – String Related Functions

- Find a sub-string
  - index(original-str, sub-str [,start position])

```
index("a very long string", "long");      # 7
index("a very long string", "lame");      # -1
index("hello world", "o", 5);             # 7
index("hello world", "o", 8);             # -1
```

- Sub-string
  - substr(string, start, length)

```
substr("a very long string", 3, 2);       # "er"
substr("a very long string", -3, 3);      # "ing"
```

- Formatting data
  - sprintf (C-like sprintf)
- perlfunc(1)
  - $ perldoc -f index

List, Array, and Hash

List

- ❑ Ordered scalars
- ❑ List literal
  - Comma-separated values
  - Ex:
    - ➢ (1, 2, 3, 4, 5)
    - ➢ ($a, 8, 9, “hello”)
    - ➢ ($a, $b, $c) = (1, 2, 3)
    - ➢ ($a, $b) = ($b, $a) ➔ swap
- ❑ List constructor
  - Ex:
    - ➢ (1 .. 5) ➔ (1,2,3,4,5)
    - ➢ (“a” .. “z”) (“a” .. “A”) ➔ (a,b,c,d,e,…,z)
    - ➢ (1.3 .. 3.1) (1.1 .. 3.1) ➔ (1,2,3)
    - ➢ (3 .. 1) ➔ ()
    - ➢ (“aa” .. “ag”) ➔ (aa, ab, ac, ad, ae, af, ag)
    - ➢ (“aa” .. “aZ”) (“a” .. “ZZ”) ➔ …
    - ➢ ($a .. $b) ➔ depend on values of $a and $b

Array (1)

- An indexed list, for random access
- Use prefix '@' in the variable name of an array
  - @ary = ("a", "b", "c")
  - @ary = qw(a b c)
  - @ary2 = @ary
  - @ary3 = (4.5, @ary2, 6.7) ➔ (4.5, "a", "b", "c", 6.7)
  - $ary3[-1] ➔ The last element of @ary3
  - $ary3[$#ary3] ➔ $#ary3 is the last index
  - ($d, @ary4) = ($a, $b, $c) ➔ $d = $a, @ary4 = ($b, $c)
  - ($e, @ary5) = @ary4 ➔ $e = $b, @ary5 = ($c)
  - $count = @ary3 ➔ 5, scalar context returns the length of an array
  - ($count) = @ary3 ➔ $count = 4.5, the 1st element of @ary3
  - (@a, @b) = (@b, @a) ➔ @a = (@b, @a), @b = ()

Array (2)

- ❑ Slice of array
  - Still an array, use prefix '@'
  - Ex:
    - ➢ @a[3] = (2)
    - ➢ @a[0,1] = (3, 5)
    - ➢ @a[1,2] = @a[0,1]
- ❑ Beyond the index
  - Access will get "undef"
    - ➢ @ary = (3, 4, 5)
    - ➢ $a = $ary[8]
  - Assignment will extend the array
    - ➢ @ary = (3, 4, 5)
    - ➢ $ary[5] = "hi" ➔ @ary = (3, 4, 5, undef, undef, "hi")

Array (3)

- ❑ Interpolation by inserting whitespace
  - @ary = (“a”, “bb”, “ccc”, 1, 2, 3)
  - $all = “Now for @ary here!”
    - ➢ “Now for a bb ccc 1 2 3 here!”
  - $all = “Now for @ary[2,3] here!”
    - ➢ “Now for ccc 1 here!”
- ❑ Array context for file input
  - @ary = <STDIN>
    - ➢ Read multiple lines from STDIN, each element contains one line, until the end of file.
  - print @ary ➔ Print the whole elements of @ary without whitespace inserted
  - print “@ary” ➔ with whitespace inserted

Array (4)

Initially, @a = (1, 2);

- Array operations
  - Push, pop and shift
    - Use array as a stack
      - push @a, 3, 4, 5 ➔ @a = (1, 2, 3, 4, 5)
      - $top = pop @a ➔ $top = 5, @a = (1, 2, 3, 4)
    - As a queue
      - $a = shift @a ➔ $a = 1, @a = (2, 3, 4)
  - Reverse list
    - Reverse the order of the elements
      - @a = reverse @a ➔ @a = (4, 3, 2)
  - Sort list
    - Sort elements as strings in ascending ASCII order
      - @a = (1, 2, 4, 8, 16, 32, 64)
      - @a = sort @a ➔ (1, 16, 2, 32, 4, 64, 8)
  - Join list
      - @a=(1,2,3); $b = join ":", @a ➔ $b = “1:2:3”

Hash (1)

- Collation of scalar data
  - An array whose elements are in <key, value> orders
  - Key is a string index, value is any scalar data
  - Use prefix “%” in the variable name of a hash
  - Ex:
    - %age = (john => 20, mary => 30,); ➔ same as (“john”, 20, “mary”, 30)
    - $age{john} = 21; ➔ “john” => 21
    - %age = qw(john 20 mary 30)
    - print “$age{john} \n”

Hash (2)

- Hash operations

  %age = (john => 20, mary => 30,);

  - keys
    - Yield a list of all current keys in hash
      - keys %age ➔ (“john”, “mary”)
  - values
    - Yield a list of all current values in hash
      - values %age ➔ (20, 30)
  - each
    - Return key-value pair until all elements have been accessed
      - each(%age) ➔ (“john”, 20)
      - each(%age) ➔ (“mary”, 30)
  - delete
    - Remove hash element
      - delete $age{“john”} ➔ %age = (mary => 30)

More on Variables

More on Variables (1) – undef

- ❑ Scalar data can be set to undef
  - $a = undef
  - $ary[2] = undef
  - $h{“aaa”} = undef
  - undef is convert to 0 in numeric, or empty string “” in string
- ❑ You can do undef on variables
  - undef $a ➔ $a = undef
  - undef @ary ➔ @ary = empty list ()
  - undef %h ➔ %h has no <key, value> pairs
- ❑ Test whether a variable is defined
  - if (defined $var) {…}

More on Variables (2) – use strict

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

- "use strict" contains
  - use strict vars
    - Need variable declaration, prevent from typo

```
use strict;
my ($x);                        # Use 'my' to declaration
use vars qw($y)                 # Use  'use vars' to declaration
```

  - use strict subs
  - use strict refs
    - perlreftut(1)
- "no strict" to close the function
- Use "use warnings" pragma to enable warnings
  - Variables without initialized occur warnings
  - "use diagnostics;" can produce verbose warning diagnostics

Predefined variables

- $_ ➔ default input and pattern-searching space
- $, ➔ output field separator for print
- $/ ➔ input record separator (default: newline)
- $$ ➔ pid
- $< ➔ uid
- $> ➔ euid
- $0 ➔ program name (like $0 in shell-script)
- $! ➔ errno, or the error string corresponding to the errno
- %ENV ➔ Current environment variables (Appendix)
- %SIG ➔ signal handlers for signals (Appendix)
- @ARGV ➔ command line arguments (1st argument in $ARGV[0])
- $ARGV ➔ current filename when reading from <> (Basic I/O)
- @_ ➔ parameter list (subroutines)
- STDIN, STDOUT, STDERR ➔ file handler names

Flow Control

Branches – if / unless

- True and False
  - 0, “0”, “”, or undef are false, others are true
  - “00”, “0.00” are true, but 00, 0.00 are false
- if-elsif-else

```
if( $state == 0 ) {
        statement_1; statement_2; …; statement_n
} elsif( $state == 1 ) {
        statements;
} else {
        statements;
}
```

- unless: short cut for if (! ….)

```
unless( $weather eq “rain” ) {
        go-home;
}
```

```
if( ! $weather eq “rain” ) {
        go-home;
}
```

- print “Good-bye” if $gameOver;
- Keep_shopping() unless $money == 0;

Relational Operators

- if ($a == 1 && $b == 2) {…}
- if ($a == 1 || $b == 2) {…}
- if ($a == 1 && (! $b == 2)){…}
- if (not ($a == 1 and $b == 2) or ($c == 3)) {…}
  - not > and > or
- || > = > or
  - $a = $ARGV[0] || 40; → if $ARGV[0] is false, then $a = 40
  - $a = $ARGV[0] or 40; → $a = $ARGV[0]
- open XX, "file" or die "open file failure!";
  - or can be used for statement short-cut.
- perlop(1)

Flow Control – while / until

- while and do-while

```
$a = 10; while ( $a  ) {  print "$a\n"; --$a }
```

```
$a = 10; print "$a\n" and --$a while $a ;
```

```
do {
            statements-of-true-part;
} while (condition);
```

- until and do-until
  - until (…) = while (! …)

```
$a = 10; until ($a == 0) { print "$a\n"; --$a }

do {
            statements-of-false-part;
} until (expression);
```

Flow Control – for / foreach

- for

  @a = (1, 2, 3, 4, 5)

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

- foreach
  - For example:

    %age = (john => 20, mary => 30,);

```
foreach $name (keys %age) {
        print "$name is $age{$name} years old.\n";
}
```

```
foreach (keys %age) {
        print "$_ is $age{$_} years old.\n";
}
```

  - The "foreach" keyword is actually a synonym for the "for" keyword

Flow Control – last, next, redo

- Loop-control
  - last
    - Like C “break”
  - next
    - Like C “continue”
  - redo
    - Jump to the beginning of the current loop block without revaluating the control expression
    - Ex:

```
for($i=0;$i<10;$i++) { # infinite loop
        if($i == 1) {
                redo;
        }
}
```

Flow Control – Labeled Block

- Give name to block to archive “goto” purpose
- Use last, next, redo to goto any labeled block
- Example:

```
LAB1: for($i=1;$i<=3;$i++) {
 LAB2: for($j=1;$j<=3;$j++) {
  LAB3: for($k=1;$k<=3;$k++) {
   print “$i $j $k\n”;
   if(($i==1)&&($j==2)&&($k==3)) {last LAB2;}
   if(($i==2)&&($j==3)&&($k==1)) {next LAB1;}
   if(($i==3)&&($j==1)&&($k==2)) {next LAB2;}
  }
 }
}
```

```
1 1 1
1 1 2
1 1 3
1 2 1
1 2 2
1 2 3
------
2 1 1
2 1 2
2 1 3
2 2 1
2 2 2
2 2 3
2 3 1
------
3 1 1
3 1 2
------
3 2 1
…
```

Subroutine

Subroutine (1)

- Definition

```
sub max {
        my ($a, $b) = @_;
        return $a if $a > $b;
        $b;
}
print &max (20, 8);
```

The value of the last statement will be returned

- Return value
  - Either single scalar value or a list value
- Arguments
  - @_ contains the subroutine's invocation arguments, and is private to the subroutine
  - $_[0], $_[1], …, $[$#_] to access individual arguments
  - Pass by value
  - perlsub(1): Pass by Reference

Subroutine (2)

- Variables in subroutine
  - Private variables
    - Use "my" operator to create a list of private variables
  - Semi-private variables
    - Private, but visible within any subroutines calls in the same block
    - Use "local" to create a list of semi-private variables

```
sub add;
sub rev2 {
        local($::n1, $::n2) = @_;
        my ($n3) = add;
        return ($::n2, $::n1, $n3);
}
sub add {
        return ($::n1 + $::n2);
}
```

Subroutine (3)

- To call subroutines
  - NAME(LIST);
    - & is optional with parentheses.
  - NAME LIST;
    - Parentheses optional if predeclared/imported.
  - &NAME(LIST);
    - Circumvent prototypes.
  - &NAME;
    - Makes current @_ visible to called subroutine.
  - NAME;
    - Like NAME() iff sub foo pre-declared, else “NAME"

Input/Output

Standard I/O

File I/O

Basic I/O (1) – Input

- Using <STDIN>
  - In scalar context, return the next line or undef
  - In list context, return all remaining lines as a list, end by EOF
    - Including array and hash

```
while( $line = <STDIN>) {
        # process $line
}
while(<STDIN>) {
        # process $_
}
```

Basic I/O (2) – Output

- print LIST
  - Take a list of strings and send each string to STDOUT in turn
  - A list of strings are separated by $, ($, = “ ”)
    - Ex:
      - print(“hello”, $abc, “world\n”);
      - print “hello”, $abc, “world\n”;
      - print “hello $abc world\n”;
- printf
  - C-like printf
    - printf “%15s, %5d, %20.2f”, $name, $int, $float;

File (1) – open and close

- Automatically opened file handlers
  - STDIN, STDOUT, STDERR
- Open

```
open FILEHD, "<", "filename";          # open for read
open FILEHD, ">", "filename";          # open for write
open FILEHD, ">>", "filename";         # open for append
```

- Open with status checked

```
open(FILEHD, "<", "filename") || die "error-message: $!";
open FILEHD, "<", "filename" or die "error-message: $!";
```

- Use <FILEHD> to read from file handlers, just like <STDIN>
- Output ex:

```
open FH, ">>", "file";
print FH "abc";              # output "abc" to file handler FH
close FH;                    # close file handler
```

File (2)

- Open with redirection
  - Open with redirection for read

```
open FD, "-|", "who";
```

  - Open with redirection for write
    - After the file handler closed, start the redirection.

```
open FD, "|-", "mail –s \"Test Mail\" liuyh@cs.nctu.edu.tw";
close FD;
```

- Directory
  - chdir function

```
chdir("/home") || die "cannot cd to /home ($!)";
```

  - Globbing (Do the same as csh)

```
@a = </etc/host*>;
@b = glob("/etc/host*");                        # @a = @b
```

File (3) – File and Directory Manipulation

- ❑ unlink(filename-list) ➔ remove files

```
unlink("data1.dat", "hello.pl");
unlink("*.o");
```

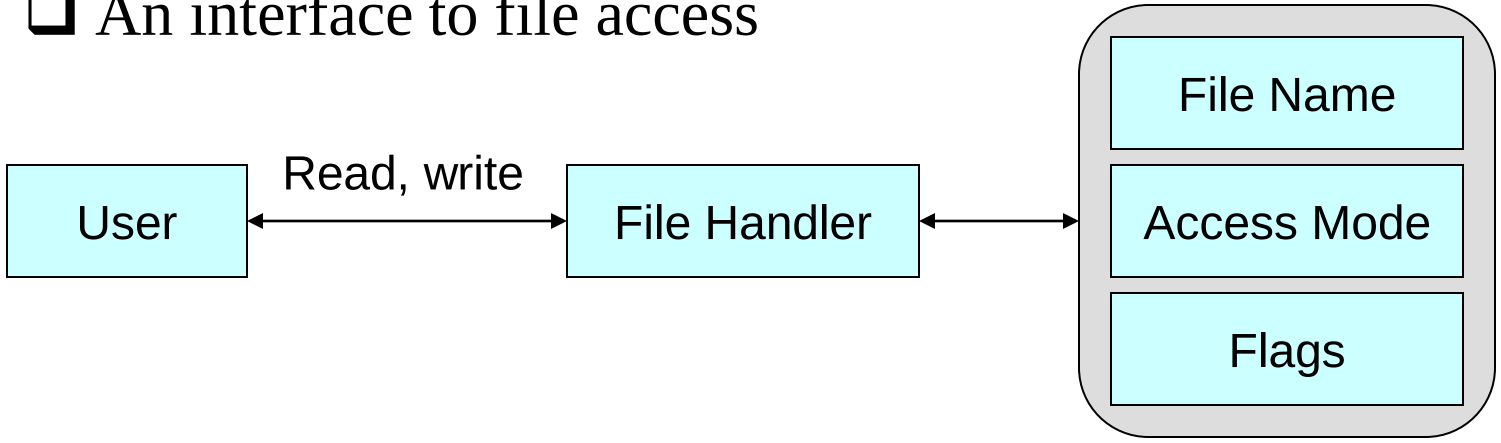
- ❑ rename(old-filename, new-filename) ➔ rename a file
- ❑ Create a link
  - link(origin, link-file) ➔ create a hard link
  - symlink(origin, link-file) ➔ create a symbolic link
- ❑ mkdir(dirname, mode) ➔ create a directory

```
mkdir("test", 0777);
```

- ❑ rmdir(dirname) ➔ remove a directory
- ❑ chmod(mode, filename) ➔ change file modes
- ❑ chown(UID, GID, filename) ➔ change ownership

File Handler

- An interface to file access



- File handler in C

```
FILE *fp = fopen("count.dat","r");
fgets(numb, 8, fp);
```

- File handler in Perl

```
open FH, “<“, “count.dat” or die “open file failure: $!”;
$numb = <FH>;
```

File Handler: <>

- while (<>) { print }

- Using diamond operator <>
  - Get data from files specified on the command line
    - $ARGV records the current filename
    - @ARGV shifts left to remove the current filename
  - Otherwise read from STDIN

- while (<FH>) { print }
  - Read from file handler FH

Regular Expression

String pattern matching & substitution

Regular Expression

- String pattern
  - What is the common characteristic of the following set of strings?
  - {goodboy, goodgirl, badboy, goodbadgirl, goodbadbadboy, …}
  - Basic regex: R_1 = "good", R_2 = "bad" , R_3 = "boy" , R_4 = "girl"
- If R_x and R_y are regular expressions, so are the following
  - (R_x or R_y)
    - R_5= (R_1 or R_2) gives {good, bad}
    - R_6= (R_3 or R_4) gives {boy, girl}
  - (R_x . R_y)➔ R_7 =(R_5 . R_6) gives {goodboy, goodgirl, badboy, badgirl}
  - (R_x*) : repeat R_x as many times as you want, including 0 times
    - R_8 =R_5* gives {good, bad, goodgood, goodbad, badgood, badbad, …}
- Our final pattern is: ("good" or "bad")* . ("boy" or "girl")
- Regular expressions can be recognized very efficiently

Regular Expression in Perl (1)

- if ($string =~ /(good|bad)*(boy|girl)/) {…}
  - Return true if any substring of $string matches
  - /^hello$/ will match the entire string
  - if (/xxxxx/) {…} matches $_
- Match single character
  - /a/, /./, /[abc]/, /[0-9]/, /[a-zA-Z0-9]/, /[^0-9]/, /[abc\]]/
  - Predefined character class abbreviations
    - digit
      - \d ➔ [0-9]   \D ➔ [^0-9]
    - word
      - \w ➔ [a-zA-Z0-9_]   \W ➔ [^a-zA-Z0-9_]
    - whitespace
      - \s ➔ [\r\t\n\f]   \S ➔ [^ \r\t\n\f]

Regular Expression in Perl (2)

- Match more than one character
  - Multipliers

{m,n}	➔ m ~ n times, inclusive
*	➔ {0,}
?	➔ {0,1}
+	➔ {1,}
{m,}	➔ >=m times.
{m}	➔ =m times.

```
/fo+ba?r/        # f, one or more o, b, optional a, r
/a.{5}b/         # a, any five non-newline char, b
```

Regular Expression in Perl (3)

- Grouping sub-regex by (…)
  - Besides matching, also remember the matched string for future reference
  - \g1 refer to 1st grouping, \g2 for 2nd, …
  - Ex:
    - /a(.*)b\g1c/ # match aXYbXYc or abc, but not aXbc
- $1, $2, $3, …
  - The same value as \g1, \g2, \g3, but can be used outside /xxx/
  - Ex:

```
$_ = "this is a test";
/(\w+)\W+(\w+)/;     # match first two words,
                     # $1 = "this", $2 = "is"
print "$1, $2\n";
```

Regular Expression in Perl (4)

- $\`, $&, $’
  - Store before-matched, matched, after-matched strings
  - Ex:

```
$_ = “this is a sample string”;
/sa.*le/;                            # $` = “this is a ”
                                     # $& = “sample”
                                     # $’ = “ string”
```

Regular Expression in Perl (5) – Substitution

- Sed-like
  - s/pattern/replacement/
- Ex:

```
$_ = "foot fool buffoon";
s/foo/bar/g;                 # $_ = "bart barl bufbarn"

$sc = "this is a test";
$sc =~ s/(\w+) \g1/<$1>/g;
                       # $sc = "th<is> a test"

$war3 = "WAR War war";
$war3 =~ s/war/peace/gi;
                       # $war3 = "peace peace peace"
```

Regular Expression in Perl (6) – Translation

- tr/search-list/replacement-list/
  - A little bit like tr command
- Ex:

```
$t = "This is a secret";
$t =~ tr/A-Za-z/N-ZA-Mn-za-m/;
                              # rotate right 13 encrypt

$r = "bookkeeper";
$r =~ tr/a-zA-Z//s;           # squash duplicate [a-zA-Z]
$a = "TTestt thiiis ccasse";
$a =~ tr/Ttic/0123/s;         # $e = "0es1 1h2s 3asse"

$n = "0123456789";
$n =~ tr/0-9/987654/d;
          # delete found but not given a replacement
          # $n = "987654"
$n =~ tr/4-9/0-2/; # $n = "222210"
```

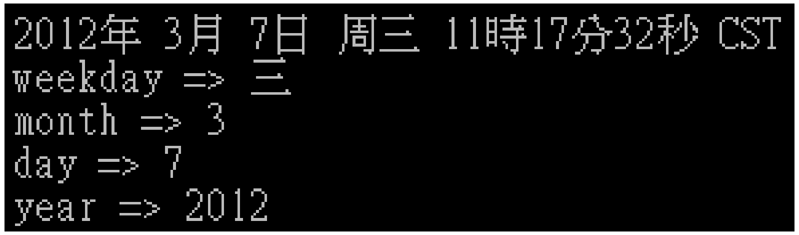
Regular Expression in Perl (7)

- Related functions
  - split(separator, string)
    - You can specify the delimit as regular expression
    - Unmatched string will form a list
    - Ex:

```
$s = "sshd:*:22:22:ssh:/var/empty:/sbin/nologin"
@fields = split(/:/, $s);
```

Regular Expression Example – Date Extraction

- #!/usr/bin/perl
- $date = `date`;
- print "$date"; # 2012年 3月 7日 周三 11時17分03秒 CST
- $date =~ /^(\d+)\D+(\d+)\D+(\d+)\S+\s+周(\S+)/;
- %hash = (
  year => $1, month => $2, day => $3, weekday => $4,
-);
- for (keys %hash) {
-     print "$_ => $hash{$_}\n";
- }



```
2012年 3月 7日 周三 11時17分32秒 CST
weekday => 三
month => 3
day => 7
year => 2012
```

Sort

Sort

- # perldoc –f sort
- Without any modification, sort is based on ASCII code
- Sort by number, you can do the following

```
@list = (1, 2, 4, 8, 16, 32);
@sorted = sort {$a <=> $b} @list;
```

- You can sort by specifying your own method, defined as subroutine, use $a, $b, and return negative, 0, and positive

```
sub by_number {
   if($a < $b) {
      return 1;            # means changing to $b, $a
   } elsif($a == $b) {
      return 0;            # means the same
   } else {
      return -1;           # means remaining $a, $b
   }
}
```

CPAN

CPAN (1)

- Comprehensive Perl Archive Network
  - http://www.cpan.org/
  - http://search.cpan.org/
- 常用的五十個CPAN模組
  - http://perl.hcchien.org/app_b.html
- /usr/ports
  - p5-*
    - s/::/-/
  - Use "psearch" to find them out
- Contributing to CPAN
  - http://www.cpan.org/misc/cpan-faq.html#How_contribute_scripts

CPAN (2)

- Install CPAN
  - Search the name of perl modules in CPAN

Gisle Aas > libwww-perl-5.805 > LWP::Simple

  - The LWP::Simple is in the libwww module
  - Use psearch “p5-<name>” to find the perl module in freebsd ports tree
  - Install it
- Use CPAN
  - manual pages installed, you can use such as perldoc LWP::Simple
  - When you search the module name, the results are the same as the manual page

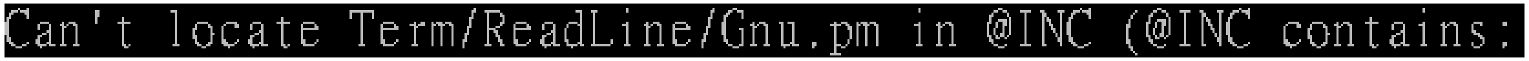
CPAN (3)

- A simple HTTP Proxy (with EVIL power!)

```
#!/usr/bin/perl

use HTTP::Proxy;
use HTTP::Recorder;

my $proxy = HTTP::Proxy->new();

# create a new HTTP::Recorder object
my $agent = new HTTP::Recorder;

# set the log file (optional)
$agent->file("/tmp/myfile");

# set HTTP::Recorder as the agent for the proxy
$proxy->agent($agent);

# start proxy
$proxy->start();
```

CPAN (4)

- If you see similar errors on execution of perl…

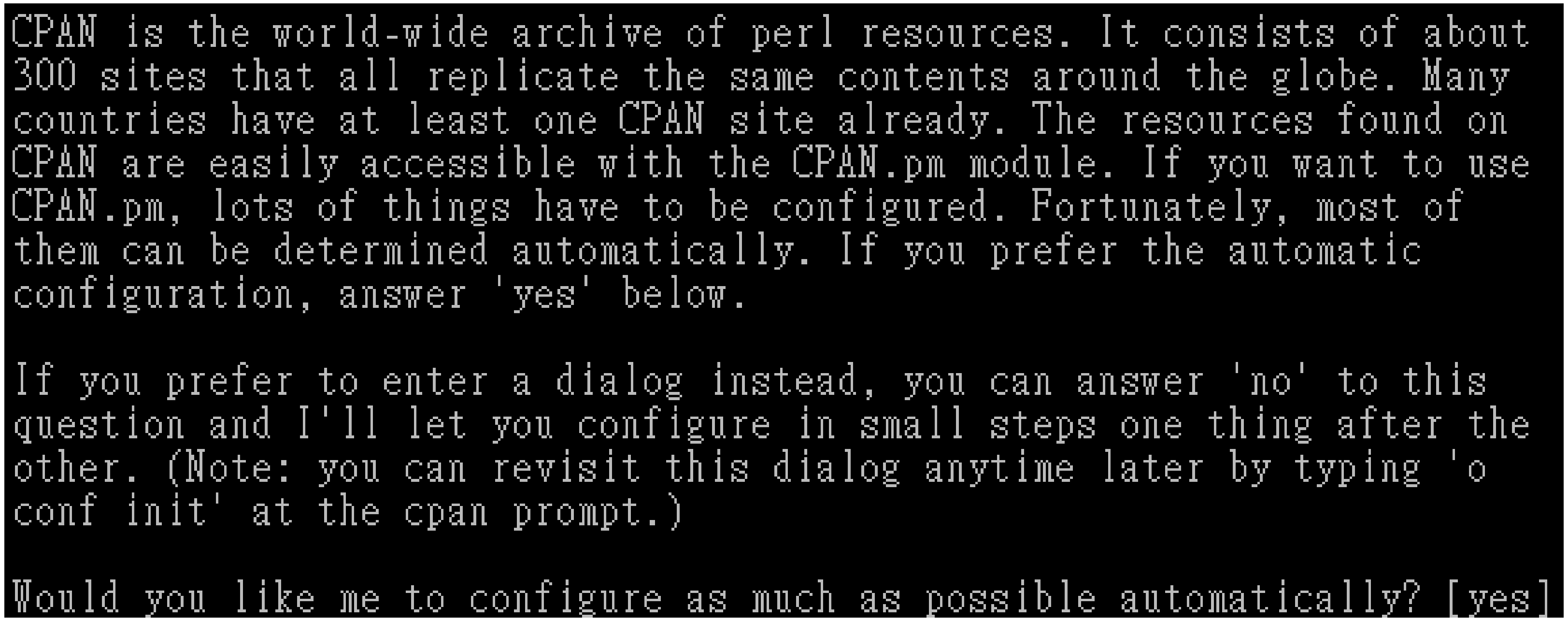


```
Can't locate Term/ReadLine/Gnu.pm in @INC (@INC contains:
```

  - Your @INC is incorrect, or
  - You need to install the required modules

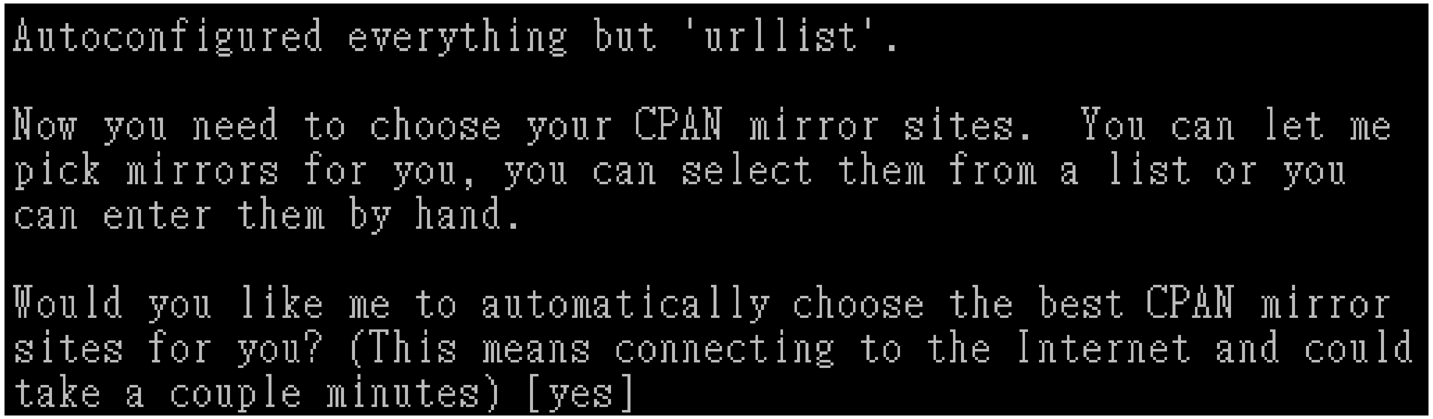
- We can install modules from command line tool
  - cpan
- Useful tool to install modules in CYGWIN

cpan tool – (1)



```
CPAN is the world-wide archive of perl resources. It consists of about
300 sites that all replicate the same contents around the globe. Many
countries have at least one CPAN site already. The resources found on
CPAN are easily accessible with the CPAN.pm module. If you want to use
CPAN.pm, lots of things have to be configured. Fortunately, most of
them can be determined automatically. If you prefer the automatic
configuration, answer 'yes' below.

If you prefer to enter a dialog instead, you can answer 'no' to this
question and I'll let you configure in small steps one thing after the
other. (Note: you can revisit this dialog anytime later by typing 'o
conf init' at the cpan prompt.)

Would you like me to configure as much as possible automatically? [yes]
```

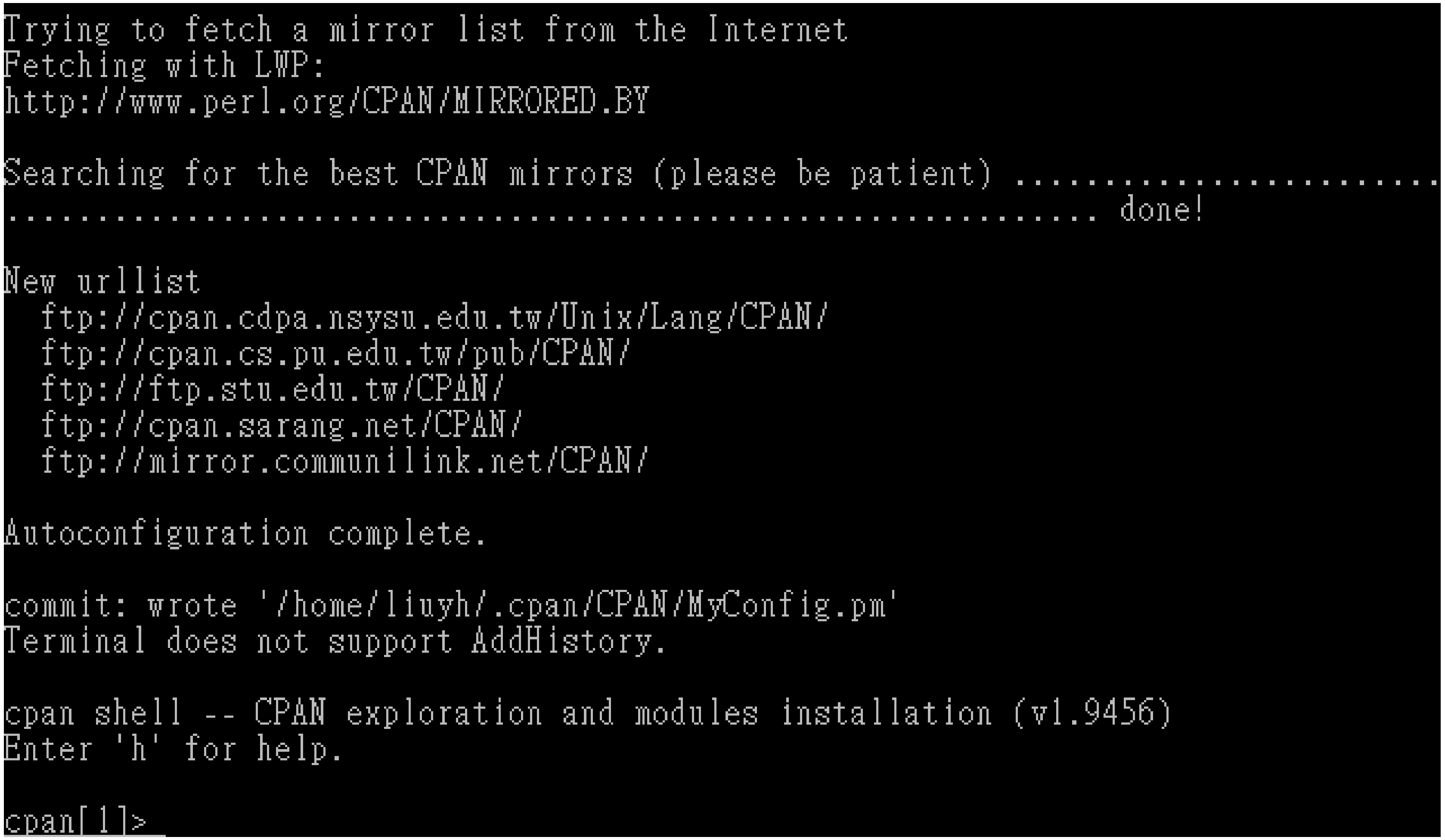
<Enter>



```
Autoconfigured everything but 'urllist'.

Now you need to choose your CPAN mirror sites.  You can let me
pick mirrors for you, you can select them from a list or you
can enter them by hand.

Would you like me to automatically choose the best CPAN mirror
sites for you? (This means connecting to the Internet and could
take a couple minutes) [yes]
```

<Enter>

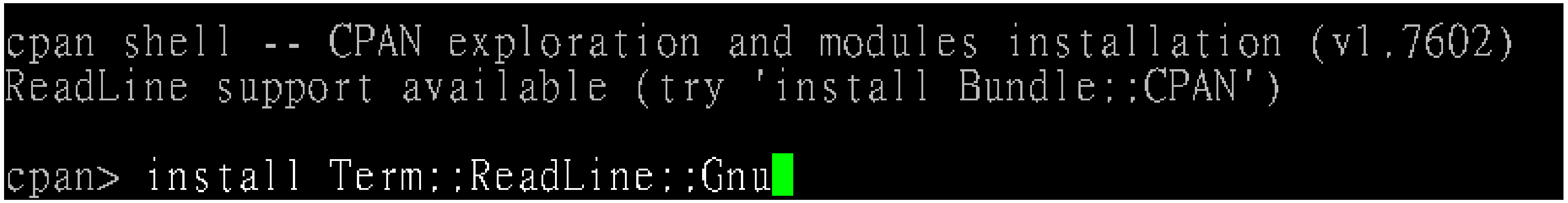
cpan tool – (2)



```
Trying to fetch a mirror list from the Internet
Fetching with LWP:
http://www.perl.org/CPAN/MIRRORED.BY

Searching for the best CPAN mirrors (please be patient) ........................
............................................................ done!

New urllist
  ftp://cpan.cdpa.nsysu.edu.tw/Unix/Lang/CPAN/
  ftp://cpan.cs.pu.edu.tw/pub/CPAN/
  ftp://ftp.stu.edu.tw/CPAN/
  ftp://cpan.sarang.net/CPAN/
  ftp://mirror.communilink.net/CPAN/

Autoconfiguration complete.

commit: wrote '/home/liuyh/.cpan/CPAN/MyConfig.pm'
Terminal does not support AddHistory.

cpan shell -- CPAN exploration and modules installation (v1.9456)
Enter 'h' for help.

cpan[1]>
```

cpan tool – (3)

❑ Install desired modules



```
cpan shell -- CPAN exploration and modules installation (v1.7602)
ReadLine support available (try 'install Bundle::CPAN')

cpan> install Term::ReadLine::Gnu
```

- "q" to exit cpan shell

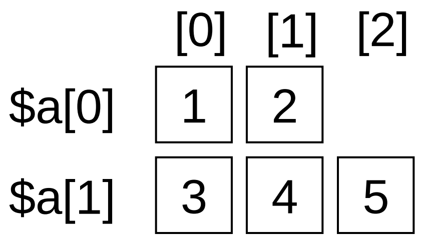
Complex Data Structure

Reference

- Create reference: store address of a variable
  - $scalarref = \$foo;
  - $arrayref = \@ARGV;
  - $hashref   = \%ENV;
  - $coderef   = \&subroutine;
- Use reference
  - $bar = $$scalarref;
  - push(@$arrayref, $filename);
  - $$arrayref[0] = "January";
  - $$hashref{"KEY"} = "VALUE";
  - &$coderef(1,2,3);

Multi-Dimension Array

- Anonymous array
  - $arrayref = [1, 2, 3];
  - $foo = $$arrayref[1];
- 2D array
  - @a = ([1, 2], [3, 4, 5]);
  - $a[0][1] == 2
  - What if @a = ((1, 2), (3, 4, 5));



- $arrayref = [1, 2, ['a', 'b', 'c']];
  - $$arrayref[2][1] == ‘b’
  - Another way to use reference by -> operator
  - $arrayref -> [2] ->[1] == ‘b’
  - $arrayref -> [2][1] == ‘b’ ➔ the 1st -> cannot be ignored

Anonymous Hash

- $hashref = { john => 20, mary => 22};
  - $$hashref{john} == 20
  - $hashref->{john} == 20
- %student = (

```
        age => {john => 20, mary => 22},
        ident => {john => 0, mary => 1},
        NA_score => [ 99, 98 ],
);
```

- $student{age}{john} == 20
- $student{ident}{mary} == 1
- $student{NA_score}[1] == 98

Anonymous Subroutine

- $coderef = sub { print "Boink $_[0]!\n" };

- &$coderef (“XD”);
- $coderef->(“XD”);

Package – A Different Namespace

```
❑ package main;          #  the default namespace
  $life = 3;
  package Mouse;         # switch to our package
  $life = 1;
  package main;          # switch back
  print "$life\n";       # shows 3
```

Perl Object Usage

❑ We have two files in the same directory

- main.pl ➔ The main script, will be run as ./main.pl
- Mouse.pm ➔ Definition of Mouse object

❑ In main.pl,

```
#!/usr/bin/perl -w
use Mouse;
$mouse = new Mouse( “Mickey” );
$mouse -> speak;
print “Age: ”, $mouse->{age}, "\n";
```

use Mouse; ← Tell perl to load the object definition in Mouse.pm

$mouse = new Mouse(“Mickey”);

1. Create new object instance and store the reference to this object in $mouse
2. Pass “Mickey” to the constructor “new”

$mouse -> speak;

Call method and pass $mouse as the 1st argument

Perl Object Definition: Mouse.pm

```
package Mouse;        ← Class name used in creating object
sub new {             ← Constructor
        my $class = shift;
        my $self = { name => $_[0], age => 10, };   ← Data structure for this object
        bless $self, $class;
}
```

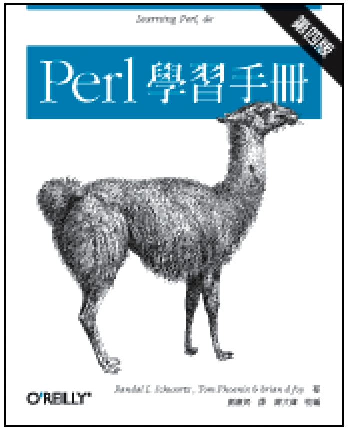
1. Associate the reference $self to the class Mouse, so we can call methods of Mouse on this reference, eg. $self->speak
2. Return the blessed reference $self

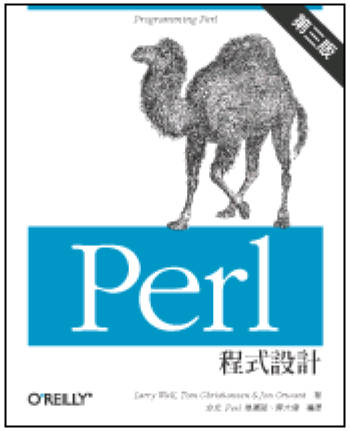
```
sub speak {
        my $self = shift;     ← Retrieve its object data
        print "My name is ", $self->{name}, "\n";
}
1;     ← Perl module must return true at the end of script
```

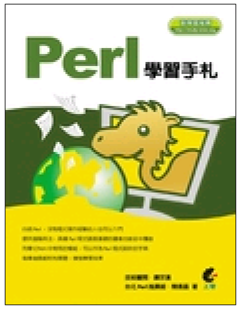
Reference Reading

Reference (1) – Document

- ❑ Book
  - Learning Perl
  - Programming Perl
  - Perl 學習手札
- ❑ Manual pages
- ❑ perldoc
  - perldoc –f PerlFunc
  - perldoc –q FAQKeywords
  - perldoc IO::Select
- ❑ Webs
  - http://perl.hcchien.org/TOC.html
  - http://linux.tnc.edu.tw/techdoc/perl_intro/
  - http://www.unix.org.ua/orelly/perl/sysadmin/







Reference (2) – manual pages

- Man Page

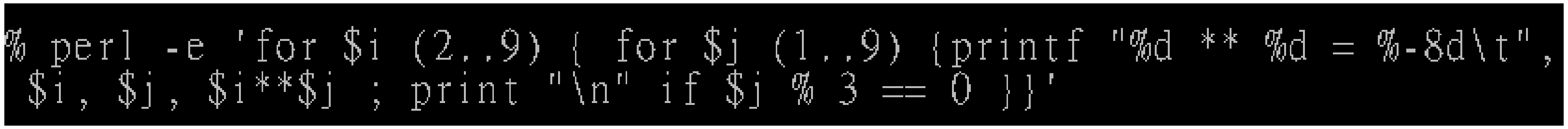
man perl	
man perlintro	→ Perl introduction for beginners
man perlrun	→ Perl execution and options
man perldata	→ Perl data structures
man perlop	→ Perl operators and precedence
man perlsub	→ Perl subroutines
man perlfunc	→ Perl built-in functions
man perlvar	→ Perl predefined variables
man perlsyn	→ Perl syntax
man perlre	→ Perl regular expressions, the rest of the story
man perlopentut	→ Perl open() tutorial
man perlform	→ Perl formats

Reference (3) – perldoc

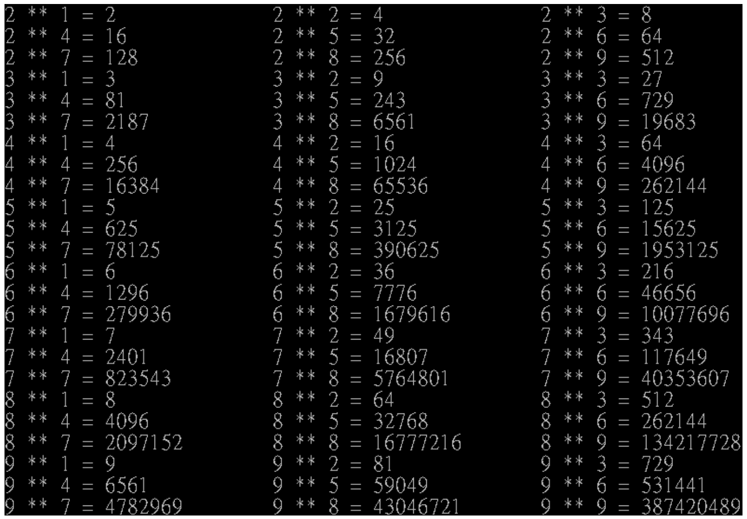
- ❑ Autrijus 大師的話，說 perldoc 要照下列順序讀
  - intro, toc, reftut, dsc, lol, requick, retut, boot, toot, tooc, boot, style, trap, debtut, faq[1-9]?, syn, data, op, sub, func, opentut, packtut, pod, podspec, run, diag, lexwarn, debug, var, re, ref, form, obj, tie, dbmfilter, ipc, fork, number, thrtut, othrtut, port, locale, uniintro, unicode, ebcdic, sec, mod, modlib, modstyle, modinstall, newmod, util, compile, filter, embed, debguts, xstut, xs, clib, guts, call, api, intern, iol, apio, hack.
  - 這樣讀完, 瞭解的會比 Programming Perl 要來得深入的多

Appendix

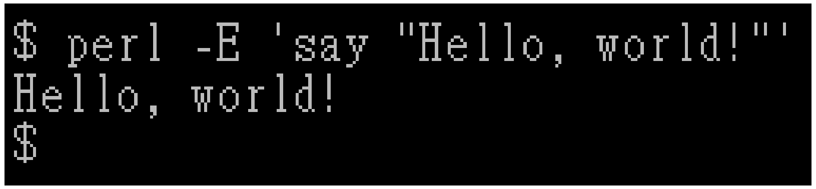
Appendix 1 – One-Line Perl Execution

- 

```
% perl -e 'for $i (2..9) { for $j (1..9) {printf "%d ** %d = %-8d\t",
 $i, $j, $i**$j ; print "\n" if $j % 3 == 0 }}'
```



```
2 ** 1 = 2              2 ** 2 = 4              2 ** 3 = 8
2 ** 4 = 16             2 ** 5 = 32             2 ** 6 = 64
2 ** 7 = 128            2 ** 8 = 256            2 ** 9 = 512
3 ** 1 = 3              3 ** 2 = 9              3 ** 3 = 27
3 ** 4 = 81             3 ** 5 = 243            3 ** 6 = 729
3 ** 7 = 2187           3 ** 8 = 6561           3 ** 9 = 19683
4 ** 1 = 4              4 ** 2 = 16             4 ** 3 = 64
4 ** 4 = 256            4 ** 5 = 1024           4 ** 6 = 4096
4 ** 7 = 16384          4 ** 8 = 65536          4 ** 9 = 262144
5 ** 1 = 5              5 ** 2 = 25             5 ** 3 = 125
5 ** 4 = 625            5 ** 5 = 3125           5 ** 6 = 15625
5 ** 7 = 78125          5 ** 8 = 390625         5 ** 9 = 1953125
6 ** 1 = 6              6 ** 2 = 36             6 ** 3 = 216
6 ** 4 = 1296           6 ** 5 = 7776           6 ** 6 = 46656
6 ** 7 = 279936         6 ** 8 = 1679616        6 ** 9 = 10077696
7 ** 1 = 7              7 ** 2 = 49             7 ** 3 = 343
7 ** 4 = 2401           7 ** 5 = 16807          7 ** 6 = 117649
7 ** 7 = 823543         7 ** 8 = 5764801        7 ** 9 = 40353607
8 ** 1 = 8              8 ** 2 = 64             8 ** 3 = 512
8 ** 4 = 4096           8 ** 5 = 32768          8 ** 6 = 262144
8 ** 7 = 2097152        8 ** 8 = 16777216       8 ** 9 = 134217728
9 ** 1 = 9              9 ** 2 = 81             9 ** 3 = 729
9 ** 4 = 6561           9 ** 5 = 59049          9 ** 6 = 531441
9 ** 7 = 4782969        9 ** 8 = 43046721       9 ** 9 = 387420489
```

- 

```
$ perl -E 'say "Hello, world!"'
Hello, world!
$
```

Appendix 2 – Process

- ❑ system() function
  - system() will fork a /bin/sh shell to execute the command specified in the argument
  - STDIN, STDOUT, and STDERR are inherited from the perl process

```
system(“date”);
system(“date ; who > $savehere”);
```

- ❑ Backquote

```
foreach $_ (`who`) {
    ($who, $where, $when) = /^(\S+)\s+(\S+)\s+(.*)$/;
    print “$who on $where at $when”;
}
```

- ❑ fork()、exec() function

Appendix 3 – Signals

- Catch the signal in your program
  - Using %SIG predefined hash
  - Using signal name in signal(3) without prefix “SIG” as the key
    - Ex: $SIG{‘INT’}, $SIG{‘TERM’}
  - Set the value to “DEFAULT”, “IGNORE”, or your subroutine name

```
$SIG{‘TERM’} = ‘my_TERM_catcher’;
sub my_TERM_catcher {
        print “I catch you!\n”;
}
```

- Sending the signal
  - kill(signal, pid-list)

```
kill(1, 234, 235);        # or kill(‘HUP’, 234, 235);
```

Appendix 4 – Built-in functions

- perlfunc(1)
  - Scalars
    - chomp, chop, index, length, sprintf, substr, …
  - Numeric
    - abs, exp, log, hex, int, oct, rand, sin, cos, sqrt, …
  - For @
    - pop, push, shift, splice, unshift
  - For %
    - delete, each, exists, keys, values
  - I/O
    - open, close, read, write, print, printf, select, …
  - Time-related
    - gmtime, localtime, time, times
  - Network
    - bind, socket, accept, connect, listen, getsockopt, setsockopt, …
  - User and group information
    - getpwent, setpwent, getpwuid, getpwnam, getgrent, setgrent, …

Appendix 5 – Feature

- Starting from Perl 5.10, you can say
  - use feature ":5.10";
    - say
    - switch
      - given / when
    - state
    - unicode_strings
      - starting with 5.11.3.
      - fully implemented 5.13.8.

```
use feature ":5.10";
given($foo) {
    when (undef) {
        say '$foo is undefined';
    }
    when ("foo") {
        say '$foo is the string "foo"';
    }
    when ([1,3,5,7,9]) {
        say '$foo is an odd digit';
        continue; # Fall through
    }
    when ($_ < 100) {
        say '$foo is numerically less than 100';
    }
    default {
        die q(I don't know what to do with $foo);
    }
}
```