Perl Programming

Reference

- Man Page
  - % man perl
  - % man perlintro (brief introduction and overview)
  - % man perlrun (how to execute perl)
  - % man perldata (data structure)
  - % man perlop (operators and precedence)
  - % man perlsub (subroutines)
  - % man perlfunc (built-in functions)
  - % man perlvar (predefined variables)
  - % man perlsyn (syntax)
  - % man perlre (regular expression)
  - % man perlopentut (File I/O)
  - % man perlform (Format)

Slides Contents

- Introduction
- Scalar data
- Arrays, List and Hash
- Control Structures
- Basic I/O
- Regular Expression
- Subroutine
- File
- Format
- Process management
- System information manipulation
- String manipulation

Introduction

Introduction

- Perl
  - **Practical Extraction and Report Language**
    - Text manipulation
    - Web development
    - Network programming
    - GUI development
    - …
- Easy to use
  - **White space between tokens**
- Compiled and interpreted
  - **Won't get a syntax error once the program is started**

The "Hello, World" (1)

Comment, to the end of line

Perl indicator
Optional arguments
man perlrun

```
#!/usr/bin/perl

# My First Perl Program
print ("Hello, World!\n");
```

Built-in function
man perlfunc

C-like ";" termination

Run Perl Program

% perl hello.pl	(even no +x mode or indicator)
% ./hello.pl	(+x mode and perl indicator)

The "Hello, World" (2)

Parentheses for built-in functions are never required

Grab one line of input

```
#!/usr/bin/perl

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

Scalar variable
man perldata

Script-like
variable embedding

Remove newline

The "Hello, World" (3)

```
#!/usr/bin/perl

print "What is your name?";
$name = <STDIN>;
chomp $name;

if ($name eq "tytsai") {
        print ("Hello, tytsai! NA slides !!\n");
}else{
        print ("Hello, $name!\n");
}
```

if-else
if-elsif-else
man perlsyn

Operator
man perlop

The "Hello, World" (4) - Array

Array @ Initialization with qw operator
man perldata

```
#!/usr/bin/perl

@pre = ("廁所", "教室", "操場");
@post = qw(放屁  大喊我愛你  喔耶);

$i = 0;
$j = 0;

for ($i = 0; $i < @pre; $i++){
      for ($j = 0; $j < @post; $j++){
            print ("I go to $pre[$i] to do $post[$j]!\n");
      }
}
```

Num of elements

Subscript reference with $

The "Hello, World" (5) - Hash

Hash %
Key ⇔ value
Key can be any scalar value
man perldata

```
#!/usr/bin/perl

%toy = qw(
        mom         judy
        dad         chiky
        son         freaky
        dog         miky
        5           ordinary
);

print "enter key: ";
$mykey = <STDIN>;
chomp ($mykey);
print "$toy{$mykey} plays $mykey\n";
```

Subscript reference with $

Specify key with {}

The "Hello, World" (6) - Regular Expression

RE match operator

Regular expression
man perlre

```
#!/usr/bin/perl

$name1 = "tytsai";
$name2 = "TytsaI";
$name3 = "TytsasaI";
$result1 = $name1 =~ /^tytsai/;
$result2 = $name2 =~ /^tytsai/i;

print ("Result1 = $result1, name1 = $name1\n");
print ("Result2 = $result2, name2 = $name2\n");

$result3 = $name1 =~ tr/a-z/A-Z/;
$result4 = $name3 =~ s/sa/SASASA/g;

print ("Result3 = $result3, name1 = $name1\n");
print ("Result4 = $result4, name3 = $name3\n");
```

Translation operator

Substitution operator

The "Hello, World" (7) - Subroutine

```
#!/usr/bin/perl

print ("Hello, world!\n");
print ("Please enter first number: ");
$n1 = <STDIN>; chomp ($n1);
print ("Please enter second number: ");
$n2 = <STDIN>; chomp ($n2);

print add($n1, $n2) ;
print "\n";

sub add {
        my($sub_n1, $sub_n2) = @_;
        return $sub_n1 + $sub_n2;
}
```

Local variable within block

Subroutine definition
man perlsub

Subroutine parameters array
man perlvar

The "Hello, World" (8) - Open file

Open a file and assign a file descriptor

Logical OR operator

Built-in "die" function

```
#!/usr/bin/perl

openfile();

sub openfile {
    open (FD1, "data.txt") || die "can't open file: $!";

    while( defined ($line = <FD1>) ) {
        print ("$line");
    }
}
```

Read one line via file handler
Use defined() to test whether **undef**

Predefined variable
System error message

The "Hello, World" (9) - Open command

```
#!/usr/bin/perl

$subject = "Alert Mail from hello9.pl";
$address = "tytsai\@csie.nctu.edu.tw";

mailsub($subject, $address);

sub mailsub {
    my ($sub, $add) = @_;

    open MAILFD, "| mail -s \"$sub\" $add";
    print MAILFD "Nothing more than a word\n";
    close MAILFD;
}
```

Print to different file descriptor

Open a command via pipe symbol

The "Hello, World" (10) - format

```
#!/usr/bin/perl

open (FD1, "data2.txt") || die "can't open file: $!";

while (defined($line = <FD1>)){
    ($name, $age, $school) = split(" ", $line);
    write;
}

close (FD1) || die "can't close file: $!";

format STDOUT =
@<<<<<<<<<<< @<<<<< @<<<<<<<
$name, $age, $school
.

format STDOUT_TOP =

  Name           Age      School
=========== ===== =======
.
```

Field definition line
man perlform

Field value line

End of format definition

Top-of-page format definition

Scalar Data

Scalar data

- Number
  - **Perl manipulates number as double-decision floating values**
  - **Float / Integer constants, such as:**
    - 1.25, -6.8, 6.23e23, 12, -8, 0377, 0xff
- String
  - **Sequence of characters**
  - **Single-Quoted Strings**
    - '$a is still $a', 'don\'t', 'hello\n'
  - **Double-Quoted Strings (variable with interpolation)**
    - "$a will be replaced\n"
    - Escape characters
      - \n, \t, \r, \f, \b, \a

Scalar Operators

- Operators for Numbers
  - **Arithmetic**
    - +, –, *, / , %, **, ++, --
  - **Logical comparison**
    - <, <=, ==, >=, >, !=
- Operator for Strings
  - **Concatenation “.”**
    - “hello” . “ “ . “world”
  - **Repetition “x”**
    - “abc” x 4 ➔ abcabcabcabc
  - **Comparison**
    - lt, le, eq, ge, gt, ne

Scalar conversion

- Number or String ?
  - **Numeric operator**
    - Automatically convert to equivalent numeric value
    - Trailing nonnumeric are ignored
    - Ex:
      - “ 123.45abc” will be 123.45
  - **String operator**
    - Automatically convert to equivalent string
    - Ex:
      - “x” . (4*5) will be “x20”

Scalar Variable

- Hold single scalar value
  - **Ordinary Assignment**
    - $a = 17
    - $b = $a + 3
  - **Binary assignment operators**
    - $a += 5 is the same as $a = $a + 5
    - -=, *=, /=, %= , **=, .=
      - > $str = $str . ".dat"
  - **Autoincrement and autodecrement**
    - ++$a, $a++

Array and List Data

List

- List
  - **An ordered scalar data**
  - **List literal representation**
    - Comma-separated values
    - Ex:
      - > (1,2,3)
      - > (“abc”, 4.8)
      - > ($a, 8, 9, “hello”)
  - **List constructor operator**
    - Ex:
      - > (1 .. 5) ➔ same as (1, 2, 3, 4, 5)
      - > (1.2 .. 4.2) ➔ same as (1.2, 2.2, 3.2, 4.2)
      - > (2 .. 5, 10, 12) ➔ same as (2, 3, 4, 5, 10, 12)
      - > (1.3 .. 3.1) ➔ same as (1.3, 2,3)
      - > ($a .. $b) ➔ depend on values of $a and $b

Array (1)

- Array
  - **A variable that holds list**
    - @ary = (“a”, “b”, “c”);
    - @ary = qw(a b c);
    - @ary2 = @ary
    - @ary3 = (4.5, @ary2, 6.7) # (4.5, “a”, “b”, “c”, 6.7)
    - $count = @ary3; # 5, length of @ary3
    - ($a, $b, $c) = (1, 2, 3)
    - ($a, $b) = ($b, $a) # swap
    - ($d, @ary4) = ($a, $b, $c) # $d = $a, @ary4 = ($b, $c)
    - ($e, @ary5) = @ary4 # $e = $ary4[0], others to @ary5
    - ($first) = @ary3; # $first = $ary3[0]
    - print $ary3[-1] # print 6.7
    - print $ary3[$#ary3] # print 6.7, $#ary3 is the last index

Array (2)

- Access a list of elements
  - **Slice of array (use @ prefix, not $)**
    - @a[0,1] = @[1, 0]
    - @a[0,1,2] = @[1,1,1]
    - @a[1,2] = (9, 10)
- Beyond the index
  - **Access will get "undef"**
    - @ary = (3, 4, 5)
    - $a = $ary[8];
  - **Assign will extend the array**
    - @ary= (3, 4, 5)
    - $ary[5] = "hi" # (1, 2, 3, undef, undef, "hi")

Array (3)

- Related functions
  - **push and pop**
    - Use array as a stack
    - Ex:

```
> push(@ary, $new);              # @ary = ($new, @ary)
> push(@ary, $new, 2, $two);     # multiple push
> $top = pop(@ary);
```

  - **reverse**
    - Reverse the order of the elements
    - Ex:

```
> @a = reverse(@a);
> @a = reverse(@b);
```

  - **sort**
    - Sort elements as strings in ascending ASCII order
    - Ex:

```
> @a = (1, 2, 4, 8, 16, 32, 64)
> @a = sort(@a);                 # gets 1, 16, 2, 32, 4, 64, 8
```

  - **chomp**
    - Do chomp to every elements of array
    - Ex:

```
> chomp(@ary);
```

Array (4)

- <STDIN> to array
  - **Return all remaining lines up to EOF**
  - **Ex:**
    - @a = <STDIN>; # press Ctrl + D
- Interpolation of array
  - **Elements are interpolated in sequence with “ ”**
  - **Ex:**
    - @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!”

Hash (1)

- Collection of scalar data
  - **<key, value> pairs**
  - **Key is the string index, value is any scalar data**
  - **Defined by "%" symbol, accessed by $ with {}**
  - **Ex:**
    - $h{"aaa"} = "bbb" # <"aaa", "bbb">
    - $h{234.5} = 456.7 # <"234.5", 456.7>
    - print $h{"aaa"}
- Hash assignment
  - **@a = %h # array a is ("aaa", "bbb", "234.5", "456.7"**
  - **%h2 = @a # h2 is like h**
  - **%h3 = %h # h3 is like h**
  - **%h4 = ("aaa", "bbb", "234.5", "456.7");**
  - **%h5 = reverse %h2 # construct hash with key and value swapped**

Hash (2)

- Related functions
  - **keys**
    - Yield a list of all the current keys in hash
    - Ex:

```
> @list = keys(%h);        # @list = (“aaa”, “234.5”)
```

  - **values**
    - Yield a list of all the current values in hash
    - Ex:

```
> @vals = values(%h);     # @vals = (“bbb”, 456.7);
```

  - **each**
    - Return key-value pair until all elements have been accessed
  - **delete**
    - Remove hash elements

Hash (3)

- **Ex:**

```
$h{"tytsai"}= "Tsung-Yi Tsai";
$h{"csie"}="Best department of computer Science";

while (($k, $v) = each (%h)) {
        print "$k is the key of $v\n";
}

delete $h{"tytsai"};
```

Control Structure

if and unless (1)

```
if (expression) {
    statements-of-if-parts;
}else{
    statements-of-else-part;
}


if (expression) {
    statements-of-if-parts;
}elsif(expression2){
    statements-of-elsif-parts;
}else{
    statements-of-else-part;
}
```

Ex:

```
print  "how old are your?";
$age = <STDIN>;

if ($age < 18) {
    print "Young lady!!\n";
}else{
    print "Such a nice day\n";
}
```

if and unless (2)

```
if (expression) {
    statements-of-if-parts;
}

unless (expression) {
    statements-of-else-parts;
}
```

Ex:

```
print  "how old are your?";
$age = <STDIN>;

if ($age < 18) {
    print "Young lady!!\n";
}

unless ($age < 18) {
    print "Such a nice day\n";
}
```

Truth is based on string value in scalar context:
"0" , "" or undef are false, others are true
0, "0", "" , undef are false
1, "1", "00", "0.000" are true

while and until

```
# while true, do body
while (expression) {
    statements-of-while-body;
}

# while not true, do body
until (expression) {
    statements-of-until-body;
}
```

Ex:

```
print  "how old are your?";
$n = <STDIN>;

while ($n > 0){
    print "At one time, I were $n years old.\n";
    $n--;
}

until ($n > 18){
    print "I am $n++, I want to be man in future.\n";
}
```

do while and do until

```
do {
    statements-of-do-body;
}while expression;

do {
    statements-of-do-body;
}until expression;
```

Ex:

```
$a = 10;
do {
    print "now is $a\n";
    $a--;
}while $a > 0;

$a = 0;
do{
    print "now is $a\n";
    $a++;
}until $a > 10;
```

for and foreach

```
for (init; test; update) {
    statements-of-for-body;
}

foreach $i (@some_list) {
    statements-of-foreach;
}
```

Ex:

```
for ($i = 1; $i <= 10; $i++){
    print "$i ";
}

@a = (1, 2, 3, 4, 5);
foreach $b (reverse @a){
    print $b;
}
```

last and next statement

- last
  - **Like C “break;”**
- next
  - **Like C “continue”;**
- redo
  - **Jump to the beginning of the current block without revaluating the control expression**

```
$n = 6;
while($n > 0){
    print "first, $n\n";
    $n--;
    if($n == 3){
        print "second, $n\n";
        redo;
    }
    print "third, $n\n";
}
```

```
first, 6
third, 5
first, 5
third, 4
first, 4
second, 3
first, 3
third, 2
first, 2
third, 1
first, 1
third, 0
```

Labeled Block (1)

- Labeled block
  - **Give name to block to achieve “goto” purpose**
  - **Use “last”, “next”, “redo” to goto any labeled block**
    - last: immediately exist the loop in question
    - next: skip the rest of the current iteration of loop
    - redo: restart the loop without evaluating

Labeled Block (2)

```
LAB1: for ($i = 1; $i <= 3; $i++){
   LAB2: for($j = 1; $j <= 3; $j++){
      LAB3: for($k = 1; $k <= 3; $k++){
         print "i = $i, j = $j, k = $k\n";
         if(($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;}
      }
   }
}
```

Result:

```
i = 1, j = 1, k = 1
i = 1, j = 1, k = 2
i = 1, j = 1, k = 3
i = 1, j = 2, k = 1
i = 1, j = 2, k = 2
i = 1, j = 2, k = 3
i = 2, j = 1, k = 1
i = 2, j = 1, k = 2
i = 2, j = 1, k = 3
i = 2, j = 2, k = 1
i = 2, j = 2, k = 2
i = 2, j = 2, k = 3
i = 2, j = 3, k = 1
i = 3, j = 1, k = 1
i = 3, j = 1, k = 2
i = 3, j = 2, k = 1
i = 3, j = 2, k = 2
i = 3, j = 2, k = 3
i = 3, j = 3, k = 1
i = 3, j = 3, k = 2
i = 3, j = 3, k = 3
```

Basic I/O

Input (1)

- Using STDIN
  - **In scalar context, return the next line or undef**
  - **In list context, return all remaining lines as a list**

- Using diamond operator "<>"
  - **Like STDIN, but diamond operator gets data from the files specified on the command line**
    - Command line arguments will go to @ARGV and diamond operator looks @ARGV

Input (2)

```
Ex:

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

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

@ARGV = (“aaa.txt”, “bbb.txt”, “ccc.txt”);
while (<>){          # this loop will gets lines from these three files
    # process $_
}
```

Output

- Using print
  - **Take a list of strings and send each string to stdout in turn**
    - Ex:
      print (“hello”, $abc, “ world\n”);

- Using printf
  - **C-like printf**
    - Ex:
      printf(“%15s, %5d, %20.2f\n”, $s, $n, $r);

Predefined variables

- man perlvar
  - $_ # default input and pattern-searching space
  - $, # output field separator for print
  - $/ # input record separator (newline)
  - $$ # pid
  - $<, $> # uid and euid
  - $0 # program name
  - %ENV # Current environment variables
  - %SIG # signal handlers for various signals
  - @ARGV # command line arguments
  - @_ # parameter list
  - $ARGV # current filename when reading from <>
  - STDIN, STDOUT, STDERR

Regular Expression

Regular Expression

- RE
  - **A pattern to be matched against a string**
    - Sometimes you just want to know the result
    - Sometimes you want to find and replace it

Ex:

```
# match the pattern "^tytsai" against $_
while (<>) {
    if ( /^tytsai/ ){
        print $_;
    }
}
```

Regular Expression Pattern - Single-Character Pattern

> Match single character

- /a/ , /./ , /[abc]/, /[abc\]]/ , /[0-9]/, /[a-zA-Z0-9]/, /[^0-9]/

- **Predefined Character Class Abbreviations**
  - digit
    - > \d means [0-9] # digit
    - > \D means [^0-9] # non-digit
  - word
    - > \w means [a-zA-Z0-9_] # word char
    - > \W means [^a-zA-Z0-9_] # non word
  - space
    - > \s means [\r\t\n\f] # space char
    - > \S means [^ \r\t\n\f] # non-space

Programming Perl, P.161 or man perlre

Regular Expression Pattern - Grouping Patterns (1)

> Match more than one character

- **Sequence**
  - Match a sequence of characters
  - Ex: /abc/ # match an a followed by b , by c
- **Multipliers**
  - * # >= 0, {0,}
  - + # >= 1, {1,}
  - ? # 0 or 1, {0,1}
  - {a,b} # a ~ b, inclusive
  - {a, } # >= 5
  - {a} # = 5

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

Regular Expression Pattern - Grouping Patterns (2)

- **Parentheses as memory**
  - Still match the pattern, but remember the matched string for future reference
  - Use \ and *number* to reference the memorized part
  - Ex:

```
> /a(.*)b\1c/        # match aTYbTYc or abc, not aEbEEc
```

  - Use (?:..) instead (..) to not memorize
- **Alternation**
  - Match exactly one of the alternatives
  - Use | to specify alternatives
  - Ex:

```
> /red|blue|green/
```

Interpolation in RE

> Variable interpolation

```
$sentence = "Every good bird does fly";
$what = "bird";
$what2 = "[bw]ird";
if ($sentence =~ /$what/) { print "I saw $what \n";}
if ($sentence =~ /$what2/) { print "I saw $what \n";}
```

- Use \U quoting escape to deal with non-aphanumeric char

```
$sentence = "Every good bird does fly";
$what2 = "[bw]ird";
if ($sentence =~ /\Q$what2\E/) { print "I saw $what \n";}
```

Special variables in RE

- $1, $2, $3 …
  - **Set to the same value as \1, \2, \3 … when memorizing**
  - **Ex:**

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

- $`, $&, $'
  - **Store before-matched, matched, after-matched strings**
  - **Ex:**

```
$_ = "this is a sample string";
/sa.*le/;               # now, $` = "this is a ",
                        #      $& = "sample"
                        #      $' = " string"
```

Operators before // - Substitution

- Substitution
  - **s/pattern/replacement/**
  - **Ex:**

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

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

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

Operators before // - Translation

- Translation
  - **tr/search-list/replacement-list/**
  - **Ex:**

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

$word = "bookkeeper";
$word =~ tr/a-zA-Z//s;          # squash duplicate, $word = "bokeper"
$me = "TThiSS iSS a TTTest";
$me =~ tr/TS/#!/s;              # $me = "#hi! i! a #est"

$he = ""abc@\$\%";
$he =~ tr/@$%//d;               # delete found but not given a replacement
                                # now, $he = "abc"
$it ="0123456789";
$it =~ tr/0-9/987654/d;         # now, $it = "987654"
```

Related functions

- split
  - **You can specify the delimit as regular expression**
  - **Unmatched string will form a list**
  - **Ex:**

```
$message = sshd:*:22:22:Secure Shell Daemon:/var/empty:/usr/sbin/nologin
@fields = split(":", $message);
```

- join
  - **Take a glue and list to form a string**
  - **Ex:**

```
$original = join(":", @fields);
```

Subroutines

Subroutine

- Definition
  - With “sub” keyword
  - Subroutine definition is global
- Return value
  - Either single scalar data or a list

Ex:

```
$a = 5;
$b = 10;

$c = ADD($a, $b);
@d = LIST_TWO($a, $b);

sub ADD{
      my($n1, $n2) = @_;
      return $n1 + $n2;
}

sub LIST_TWO{
      my($n1, $n2) = @_;
      return ($n1, $n2);
}
```

Arguments

- @_
  - **Contain the subroutine invocation arguments**
  - **@_ is private to the subroutine**
    - Nested subroutine invocation gets its own @_
  - **$_[0], $_[1], …, $_[$#_] to access individual arguments**

Variables in subroutine

- Private variables
  - Use “my” operator to create a list of private variables
- Semiprivate
  - Private, but visible within any subroutines calls in the same block
  - Use “local” to create a list of semi-private variables

```
$value = “orignial”

tellme( );  spoof( );  tellme( );
# original  temporary  original

sub spoof{
   local ($value) = "temporary";
   tellme();
}

sub tellme {  print "$value";}
```

```
$value = “orignial”

tellme( );  spoof( );  tellme( );
# original  original  original

sub spoof{
   my ($value) = "temporary";
   tellme();
}

sub tellme {  print "$value";}
```

File

Open and close (1)

- 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";**
- Close
  - **close(*FILEHD*)**

Open and close (2)

- Open with redirection
  - **Ex:**

```
#!/usr/bin/perl

open (FD, "ypcat passwd | grep /tytsai |");
while(<FD>){
        chomp;
        print "$_\n";
}

open (FD2, "|/usr/bin/mail -s \"Mail from perl\" tytsai\@csie.nctu.edu.tw");

print FD2 "this is test\n";
```

File test

```
$name = "index.html";
if (-e $name) {
      print "file: $name exists\n";
}
```

File test	Meaning
-r	File is readable by effective uid/gid
-w	File is writable by effective uid/gid.
-x	File is executable by effective uid/gid.
-o	File is owned by effective uid.
-R	File is readable by real uid/gid.
-W	File is writable by real uid/gid.
-X	File is executable by real uid/gid.
-O	File is owned by real uid.
-e	File exists.
-z	File has zero size (is empty).
-s	File has nonzero size (returns size in bytes).
-f	File is a plain file.
-d	File is a directory.
-l	File is a symbolic link.
-p	File is a named pipe (FIFO), or Filehandle is a pipe.
-S	File is a socket.
-b	File is a block special file.
-c	File is a character special file.
-t	Filehandle is opened to a tty
-u	File has setuid bit set.
-g	File has setgid bit set.
-k	File has sticky bit set.
-T	File is an ASCII text file (heuristic guess).
-B	File is a "binary" file (opposite of -T).
-M	Script start time minus file modification time, in days.
-A	Same for access time.
-C	Same for inode change time (Unix, may differ for other platforms)

Directory

- Use “chdir” function
  - **Change current directory**
  - **Return successful or not**
  - **Ex:**

    chdir(“/etc”) || die “cannot cd to /etc ($i)”;
- Globbing
  - **Expansion of path that contains * into list**
  - **Globbing can be done through**
    - <path>
    - glob function

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

# /etc/host.conf /etc/hosts /etc/hosts.allow /etc/hosts.equiv /etc/hosts.lpd0
```

File and Directory Manipulation

- Removing file
  - **unlink(filename-list);**
  - **Ex:**

```
unlink("data1.txt", "hello.pl");
unlink <*.o>;
```

- Renaming a file
  - **rename(file, new-name);**
- Create link
  - **link (original, link-file)** **# ln origninal link-file**
  - **symlink(original, link-file)** **# ln –s original link-file**
- Making and removing directory
  - **mkdir(directory-name, mode)** **# mkdir("test", 0777)**
  - **rmdir(directory-name)**
- Modify permission
  - **chmod(mode, file)** **# chmod(0666, "hello.pl")**
- Change ownership
  - **chown(UID, GID, file)** **# chown(1234, 35, "hello.pl")**

Format

Format

- Format
  - **Report writing template**
  - **Define**
    - Constant part (headers, labels, fixed text)
    - Variable part (reporting data)
  - **Using format**
    - Defining a format
    - Invoking the format

Define a format

> Use "format" keyword

- **syntax**

  format *name* =
  fieldline
  value1, value2, value3, …
  fieldline
  value4, value5, …
  .

- **fieldline**
  - can be either fixed text or "fieldholders" for variable
    - > White space is important in fieldline
    - > White space is ignored in value line
  - If there is any fieldholders in fieldline, there must be a series of scalar variable in the following line

```
format ADDRESS =
==============================
| @<<<<<<<<<<<<<<<<<<<<<<<<<<<|
$name
| @<<<<<<<<<<<<<<<<<<<<<<<<<<<|
$address
| @<<<<<<<<<<<<<, @<<<  @<<<<<<|
$city,  $state,  $zip
==============================
```

Invoking a format

- Through “write” function
  - **write function will write stuff into current file handler using “current” format**
  - **Default current format is the same name with file handler**

```
#!/usr/bin/perl

open (FD1, "data2.txt") || die "can't open file: $!";

while (defined($line = <FD1>)){
   ($name, $age, $school) = split(" ", $line);
   write;
}

close (FD1) || die "can't close file: $!";

format STDOUT =
@<<<<<<<<<<< @<<<<< @<<<<<<<
$name, $age, $school
.
```

Fieldholders

- @<<<
  - **It means “5 character, left justified”**

- Text fields
  - **Use @ to mean text fields**
  - **Use <, >, | to mean left, right and center -justified**
- Numeric Fields
  - **Use @ to mean numeric fields, but use “#” to represent digit**
  - **Ex:**
    - Assets: @######.##
- Multiline Fields
  - **Use @* to place multiple lines in single fieldholders**

The Top-of-page format

- Let report to fit page-size printing device
  - **Perl will call top-of-page format if**
    - In the very beginning of write
    - When the output cannot fit in current page
  - **Default page length**
    - 60 lines
    - Set $= to 30 can change page length to 30 lines
  - **Default top-of-page format name**
    - *filehandlername*_TOP
  - **Variables used in top-of-page format**
    - $%
      - Will be replaced with current page number

Changing Defaults

- Change default file handler
  - **Use select function**
  - **print without file handler will write stuff to default handler**
  - **Ex:**

```
print “hello world\n”;          # print STDOUT “hello world\n”;
$oldFD = select (LOGFILE);
print “Error happened\n”;  # print LOGFILE “Error happened\n”;
select ($oldFD);                # restore to saved file handler
```

- Change default format name
  - **Set $~ variable**
  - **Ex:**

```
$~ = ADDRESS
write;                          # it will use the ADDRESS format
                                # other than STDOUT
```

Process Management

Using system() function

- system function
  - **system( ) will fork a /bin/sh shell to execute the command specified in arguments**
  - **STDIN, STDOUT and STDERR are inherited from the Perl process**
  - **Ex:**

```
system("date");
system("(date;who) > $gohere");
```

Using Backquote

> ` `

- Execute the command and replace itself with execution result
- Ex:

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

```
tytsai@tybsd:~/Perl> who
tytsai          ttyv0    Mar 28 14:05
tytsai          ttyp0    Mar 30 08:27 (ccamd)
tytsai          ttyp1    Mar 28 14:12 (ccamd:S.0)
tytsai          ttyp2    Mar 28 14:12 (ccamd:S.1)
tytsai          ttyp3    Mar 28 14:12 (ccamd:S.2)
tytsai@tybsd:~/Perl> perl process.pl
tytsai on ttyv0 at Mar 28 14:05
tytsai on ttyp0 at Mar 30 08:27 (ccamd)
tytsai on ttyp1 at Mar 28 14:12 (ccamd:S.0)
tytsai on ttyp2 at Mar 28 14:12 (ccamd:S.1)
tytsai on ttyp3 at Mar 28 14:12 (ccamd:S.2)
```

Using Process as Filehandler

- We can either
  - **Open and capture the output from process or**
  - **Open and provide input to process**
- Ex:

```
open(WHOFD, "who |");
open(MAILFD, "| mail tytsai\@csie.nctu.edu.tw")

open(MULTI, "who | grep :S.*|");
while(<MULTI>){
    print $_;
}
```

Using fork() function

- Just as fork(2) do
  - **Create a clone of the current perl process**
  - **Use return PID to distinguish parent and child**
    - Zero for child and nonzero for parent

```
if (!defined($child_pid = fork())){
    # fork failed
    die "cannot fork: $!";
}elsif ($child_pid){
    exec("date");
    die “can’t not exec data: $!";
}else{
    waitpid($child_pid,0);
    print("child has finished\n");
}
```

Sending and Receiving Signals (1)

- Catch the signal in your program
  - **Using %SIG predefined hash**
  - **Using signal name in 'man signal' without prefix "SIG" as the key**
    - Ex:
      > $SIG{'INT'}, $SIG{'TERM'}
  - **Set the hash value to your subroutine to catch the signal**
    - Use "DEFAULT" to restore default action
    - Use "IGNORE" to ignore this signal (no action)
- Sending the signal
  - **Use kill( ) function**
    - kill(*signal*, *pid-list*)
      > kill(2, 234, 235); or kill('INT', 234, 235);

Sending and Receiving Signals (2)

> Ex:

```
#!/usr/bin/perl

$SIG{'TERM'} = 'my_TERM_catcher';


print "before sending signal..\n";

kill(15, $PID);

print "after sending signal..\n";

sub my_TERM_catcher{
   print "I catch you!! Do cleanup works\n";
}
```

System Information Manipulation

User information (1)

- Using getpwuid() or getpwnam()
  - **Pass uid to getpwuid( ) and login-name to getpwnam( )**
  - **Both return the list:**

($name, $passwd, $uid, $gid, $pw_change, $pw_class, $gcos, $dir, $shell, $pw_expire)

```
@a = getpwnam("tytsai");
@b = getpwuid(1001);

print "@a\n";
print "@b\n";

# tytsai * 1001 1001 0  Tsung-Yi Tsai /home/tytsai /bin/tcsh 0
# tytsai * 1001 1001 0  Tsung-Yi Tsai /home/tytsai /bin/tcsh 0
```

User information (2)

- Sequential access to passwd
  - **Use setpwent( ), getpwent( ) and endpwent( )**
- Sequential access to group
  - **Use setgrent( ), getgrent( ) and endgrent( )**

```
setpwent();
while(@list = getpwent()){
    print ("@list\n");
}
endpwent();

setgrent();
while(@list = getgrent()){
    print ("@list\n");
}
endgrent();
```

String Manipulation

Related functions

- Find a substring
  - **index(original-str, sub-str)**

```
$where1 = index("a very long string", "long");      # 7
$where2 = index("a very long string", "lame");      # -1
$where3 = index("hello world", "o", 5);             # 7
$where4 = index("hello world", "o", 8);             # -1
```

- Sub-string
  - **substring(string, start, length);**

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

- Formatting data
  - **sprintf(format, argument-list);**

```
$result = sprintf("%05d", $y);
```

Sort

- Sort
  - Without any modification, sorting is based on ASCII code
  - You can sort by specifying your “comparison method”
  - Ex:

```
@somelist =
      (1,2,4,8,16,32,64,128,256);

@a = sort @somelist;
@b = sort by_number @somelist;

print "a = @a\n";
print "b = @b\n";

sub by_number{
   if($a < $b){
      return -1;
   }elsif ($a == $b){
      return 0;
   }elsif ($a > b){
      return 1;
   }
}
```

Built-in functions

Built-in functions

- For Scalars
  - **chomp, chop, index, length, sprintf, substr, …**
- Numeric
  - **abs, exp, log, hex, int, oct, rand, sin, cos, sqrt, …**
- For @ or %
  - **push/pop, shift, sort, keys, values, delete**
- I/O
  - **open, close, read, write, print/printf, …**
- Time-related
  - **gmtime, localtime, time, times**
- Network
  - **bind, socket, accept, connect, listen, getsockopt/setsockopt, …**
- User and group info
  - **Getpwent/setpwent, getpwuid, getpwnam, getgrent/setgrent, …**