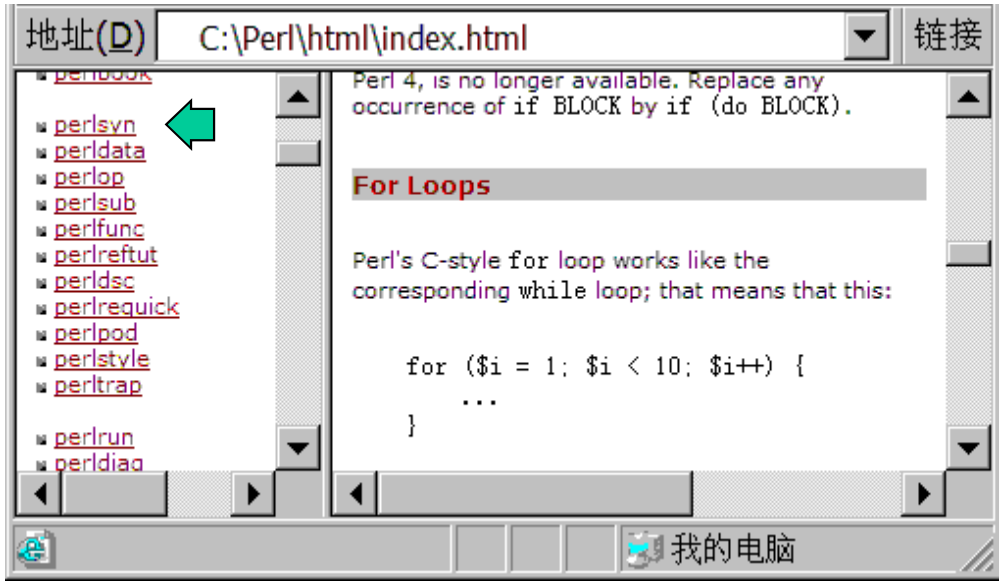
Perl 入门和提高　Lesson 2

周晓方

How to get help? 获得帮助

- All the viewgraphs and books
- On-line documents (for *ActivePerl* only)
  - C:/perl/html/index.htm
  - Click perlfunc, perlfaq, perlxs, ……
- perldoc command
  - perldoc -f print
  - perldoc -m benchmark
  - perldoc perldoc
  - perldoc perlop查看perl关于运算符的文档
  - Sometimes use with "pod2text", "pod2html"



—— perldoc perlfunc

Context上下文: scalar, list

```
@junk = (1, "I", undef, 66..69, "foo");
print @junk; #array context      ===> 1I66676869foo
print @junk . "\n"; #scalar context=> 8
$size = @junk; #scalar context   ===> $size is 8
$size = scalar @junk;            ===> $size is 8
              #scalar()  force a scalar context
```

- **scalar @array** returns number of elements
- **scalar (LIST)** returns its final element
- These are scalar context

```
print @junk . "\n";          print @junk + 0;
$size = @junk;
```

- These are list context

```
print @junk, "\n";           ($head) = @junk;
```

- Function behaviors differently to context

```
$line = <FILE>;        @lines = <FILE>;
```

Operators, See also "perlop"

- Binary op: + - * **(exponentiation) / %(Modulus)
- Unary op: +var -var ++var --var var++ var--
- Logical op: && || // ! **and** or not xor
- Bitwise op: & | ^ >> << ~
- Numeric relation: == != < <= > >= **<=>**
- String relation: eq ne lt le gt ge **cmp**
- Ternary op: a ? b : c (可以是左值)
- Range op: (1..5, "A".."D")
- String op: . **x** (小写字母x)
- Assign: = += -= *= /= %= .= **= x= <<= >>= &= &&= |= ||= ^=
- Other: ( ) [ ] { } \(referenct) ->(infix dereference)

数据弱类型
➔
运算符强类型

Operators (cont.)不考

- **||**测验左式Truth，**//**测验左式Definedness
- 标量环境的**..**(以及**...**)表示“翻转”，常用于迭代过滤，详见perlop关于flip-flop一段

```
open F, "README.txt"; # 以Perl的README.txt为例
while (<F>) {         # 用循环语句迭代if
        print if /you need/ .. /and then/; # .. flip-flop
//      print if 10..20;      # 特指读到第几行
}
```

- **...**;(作为语句,v5.12+)表示“yada-yada”
- **|.** **&.** **^.**字符串的“按位或、与、异或”
  - use feature 'bitwise' 或 use v5.28
  - 相应地，也有 |.= &.= ^.=
- **~~** 规则很复杂的smart-match

Precedence

Level	Operator	Description	Associativity
22	(), [], {}	Function Calls, Parentheses, Array subscripts	Left to right
21	->	Infix dereference Operator	Left to right
20	++, --	Auto increment, Auto decrement	None
19	**	Exponentiation	Right to left
18	!, ~, +,+, -,\	Logical not, bitwise not, unary plus, unary minus, reference	Right to left
17	=~, !~	Match, Not match	Left to right
16	*, /, % x	Multiply, Divide, Modulus, Repetition	Left to right
15	+, -,.	Add, Subtract, String concatenation	Left to right
14	<<,>>	Bitwise left shift, Bitwise right shift	Left to right
13	-f -e 等	File test Operators	None
12	< <= > >= ge 等	Relational Operators	None
11	== != eq 等	Equality Operators	None
10	&	Bitwise and	Left to right
9	|, ^	Bitwise or, Bitwise xor	Left to right

Precedence (cont.)

8	&&	Logical and	Left to right
7	||	Logical or	Left to right
6	..	Range Operator	None
5	?:	Ternary or conditional Operator	Right to left
4	= += -= 等	Assignment Operators	Right to left
3	,	Comma Operator	Left to right
2	not	Low precedence logical Operators	Left to right
1	and	Low precedence logical Operators	Left to right
0	or, xor	Low precedence logical Operators	Left to right

- 逐渐习惯Perl的写法
  - *if (($a > 4) && ($b > 6)) {...;}* — if ($a > 4 and $b > 6) {…;}
  - *if (not open F, "a.txt") { die; }* — die if not open $, "a.txt";
open F, "a.txt" or die;
  - *if ($a < 0) { $a = -$a;}* — $a >= 0 or $a = -$a;

Precedence (cont.)

- 逻辑运算和逻辑位操作的优先级比关系运算的优先级低
  - 判断$a最低位是否是0可以用($a & 0x1) == 0
    - $a & 0x0001 == 0 相当于 $a & (0x0001 == 0)
  - 计算$a或$b，结果存入$c
    - $c = $a || $b; # correct
    - $c = $a or $b; # wrong
- 在易混淆之处，宜加上括号，也有助于阅读

关联数组 Hash --%

- Index (or key) is an scalar (usually a string, not only numbers).
- Order-less.
- Relational database

```
%assoArr = ("Jack", "Dec 2", "Joe", "June 2", "Jane", "Feb 13");
%assoArr = ("Jack"=>"Dec 2", "Joe"=>"June 2", "Jane"=>"Feb 13");
$assoArr{"Marry"} = "Oct 10";
$assoArr{"Jennifer"} = "Mar 20";
print "Joe's birthday is: " . $assoArr{Joe} . "\n";
=>Joe's birthday is: June 2
%assoArr = (); #empty hash.
```

多数情况下，key串的引号可省略

```
%hex2bin = (0 => "0000", 1 => "0001", ... , F=>"1111");
$hex = "A19e";
$bin = join("_",map($hex2bin{$_},
                    split(//,uc  $hex)));
=>1010_0001_1001_1110
另一种办法：（记住perl的口号 There's More Than One Way To Do It）
print sprintf "%b", hex $hex;
```

Functions working with HASH, 1

- Nevery call **`print %hash;`** since hash is orderless.
- @key = sort **keys** %hash
  - foreach $key (sort keys %hash) {print $hash{$key};}
- @key = keys %hash; @value = **values** %hash;
- ($key, $val) = **each**(%hash) # less memory use

each( ) returns once a pair, returns undef at last. 历遍哈希表

```
print "$key=$value\n"
   while (($key,$value) = each %ENV);
   # never modify %hash inside the while each loop!!!
   TZ=BJT-8
   PROCESSOR_REVISION=0806
   NUMBER_OF_PROCESSORS=1
   USERPROFILE=C:\WINNT\Profiles\xfzhou
   TMPDIR=c:/temp
   COMPUTERNAME=N30
   . . .
Q: how to access environment variables in C programs?
```

*千万避免**each**迭代时修改**hash***

环境变量在hash表%ENV中

Functions working with HASH, 2

- Remove one hash from hash table
  - 删除单个元素: **delete** $hash{$key};# remove whole pair
  - This is wrong: *undef $hash {key};* #key still exists
  - 删除hash片断: delete @hash{key1, key2, key3…};
- Delete a whole hash table
  - %hash = ();
  - **undef** %hash;
- Check existence of a hash
  - if (**exists** $hash{$key}) {….}
  - This is wrong: if (*$hash{$key}*) {….} 值可能是0或undef
  - This is also wrong: if (*defined $hash{$key}*) …
- **delete**和**exists**函数也可以用于普通数组 *很少用*
  - 和splice( )不同，delete( )后，下标不变，exists( )为假
  - 只有delete了数组的最后一个元素，数组才会缩小。

Reference to hash table

- Reference: **** Dereference: **%**

```
$Rhash = \%hash;
```

%{$Rhash} is a hash

- Refer to anonymous hash

```
$Rh = { 'dog'=>'bark', 'cat'=>'mew', };
print "Sound of dog is ${$Rh}{dog} \n";
```

- Short hands 省略{}、写成->的样子
  - ${$a} ≡ $$a, @{$a} ≡ @$a, %{$a} ≡ %$a, etc
  - Ref to list: $a = **[**`1, 2, 3, "AAB"`**]**

```
${$a}[0]=50; $$a[0]=50; $a->[0]=50; # same
```

  - Ref to hash:$h= **{**cat=>"rat", dog=>"meat"**}**

```
${$h}{rat}="rice"; $$h{rat}="meat";
print $h->{rat};
```

  - 注意优先级，perl的$$a[j]和C语言的*x[j]不同
    $$a[j]和${$a[j]}不同

标量+数组+散列+引用 ➔ 复杂与多变的Perl数据结构

- Perl数据结构可以方便地增加新的内容
- 成绩单的例子
  - 成绩单：
    姓名➔Perl成绩　　或　　学号➔Perl成绩
    (既便是{学号➔Perl}，也要用hash表，而不是array)
  - 增加MCU成绩：
    学号➔[Perl成绩，MCU成绩]
  - 改成hash的hash：
    学号➔{Perl➔Perl成绩，MCU➔MCU成绩}
  - MCU区分理论和实验成绩：
    学号➔{Perl➔xxx, MCU➔{quiz➔xxx, lab➔xxx}}

回家作业

- 按照文件大小列出当前目录中的文件。
  附件: 学号-02.pl　　提示：
  - Perl中列当前目录中的文件到数组：
    `@list = grep {-f $_} <*>;`
  - Perl中获得文件大小：`$s = (stat($fn))[7];`
  - 用散列存放文件名，`%f = (name => size,);`
  - 获得对散列Values排序的Key：
    `@k = sort {$f{$a} <=> $f{$b}} keys %f;`
  - 打印格式示例：

```
20      ntuser.ini
323     transcript
17689   help.txt
20480   WebpageIcons.db
```

这两处没有逗号

正则表达式

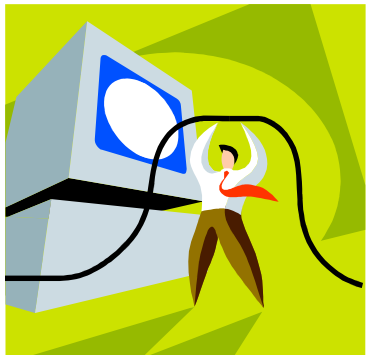
- 计算理论中的正则表达式
- UNIX中的正则表达式
- Perl的文本匹配和替换

Try some Regular Expressions

```
$a = 'dog catches cats.';
print $a =~ /cat/ ? 'Y ' : 'N ';  # Match
print $a =~ /^cat/ ? 'Y ' : 'N ';
$a =~ s/cat|dog/pet/g;            # Substitute
print "\n$a\n";
$a =~ tr/a-z/za-y/;               # Translate
print "$a\n";
```

Results:

```
Y N
pet petches pets.
ods odsbgdr odsr.
```



RE Rule 1:通配符,元字符

- Literal: `m/a/`, `m/\$/`, `m/AbCd/`
- ^ caret: Begin of a str/line:
  `m/^#/` -- a line begin with pond sign
- $ dollar: End of a str/line:
  `s/$/##/m` -- append "##" to each line
- | (verital bar) alternation:
  `s/dog|cat/pet/g` -- all dog or cat to pet
- . dot: any char except new line (except ///s):
  `m/./` non-empty string
- Char-set with meta-char "-" and "^",
  e.g. `[A-Z] [a-z0-9] [+\-] [^0-9]`
- \ escape char :
  e.g. match for "/usr/": `$path =~ /\/usr\//`

More on RE Rule 1

- Quantifiers: * (any times, even 0), + (1 or more), ? (none or once)
  例如 `$line =~ s/ +/ /g;` 合并相邻的空格
- Quantifiers: {n} *n* times, {n,} *n* or more, {n1, n2} *n*1 to *n*2 times
- greedy (+, *) or lazy (+?, *?) 贪婪和懒惰，实例：

```
$a = "zzzzzzzzzzzzzzzzzfaaaadccccdtttdeee"
$a =~ /f.*d/  #matches faaaadccccdtttd
$a =~ /f.*?d/ #matches faaaad
```

- ( ): group and pattern memory
- Words `\w [a-zA-Z0-9_],` `\W [^a-zA-Z0-9_]`
- Spaces `\s [\t \n],` `\S [^\t \n]`
- Digits `\d [0-9],` `\D [^0-9]`
- \b, word boundary (zero width) 表示位置，0宽度
  \B, non-word boundary 表示位置，\b的补

More Samples

/0|[1-9][0-9]*/	匹配自然数
/\$[a-z][a-z0-9_]*/i	匹配普通scalar名称
/\bcat\b/	匹配cat,但不匹配catch
/^$/	匹配空行
s/#.*//	删除注释部分

- 带修饰g和e的RE,带反向引用的RE

s/\d+/0/g	全部数字串变'0'
s/(\d+)/ $1 /g	全部数字前后加空格
s/(\d+)/$1+1/ge	全部数字串加1 (e表达式)

- 留心元字符、$和@, RE是带插值功能的

/my\@school\.com/	匹配my@school.com
m'my@school.com'	同上(无插值有meta)
m!my\@school\.com!	同上(先插值后meta)

Regular expression--Rule 2

- 看文档: ***perlre*** ***perlretut*** ***perlop***
- Match: //, m// $catstring =~ /cat/; # interpolation & meta
- Substitute: s/// $a =~ s/([0-9]+)/$1+1/ge; #interpolation & meta in match, interpolation (but no meta) in replacement
- Translate: tr/// $a =~ tr/A-Z/a-z/; # no interpolation!
  - $result !~ tr/0-9/a-j/; # translate 0-9 to a-j, and return false if happened.
- Delimiters
  - m.cat., m/\/usr\//, m{/usr/}, m!/usr/!
  - m/$file/ (interplation and meta), **m'$file' (neither interplation nor meta)**
  - s{bbb}{12345}; s/bbb/12345/
- (Default var is $_) Binding operator: =~ or !~, !~ for not match
- Only works on scalar. ***WRONG***: *@arrayName =~ m/pattern/*;
- Only matches the left most. Only mathes once (by default)

RE Rule 3: 处理插值和通配符

- Interpolation 1st, then processing meta chars
  - A very strong but not easy to debug feature.
  - *$a="cat|dog";* $b =~ /$a/; # match cat or dog
  - @a=("cat", "dog"); $" = "|"; $b=~ /@a/;
  - $a="/usr/"; $b =~ /$a/; #得到//usr// **WRONG**
  - Solution:
    1. Escape $a: $a= '\/usr\/'; # not "\/usr\/" !!
    2. Change delimiters: $b=~ m[$a];
    3. Calling quotemeta( ): $a = quotemeta("/usr/");
- Exception:
  - tr/// is always built in compile time.