1 NAME

perlstyle – Perl 风格指南

2 DESCRIPTION

每一个程序员都应该注意自己的程序风格。当然，程序员可以选择自己喜爱的 风格，然而，有一些通用的规则可以使你的程序更加易读、易懂、易于维护。

The most important thing is to run your programs under the −w flag at all times. You may turn it off explicitly for particular portions of code via the `no warnings` pragma or the `$^W` variable if you must. You should also always run under `use strict` or know the reason why not. The `use sigtrap` and even `use diagnostics` pragmas may also prove useful.

任何时候都应该用 −w 选项来运行你的程序，这一点非常重要。必要的时候 你可以在程序中通过 `no warnings` 编译指示或者 `$^W` 变量来临时关闭 它。另外，建议你在程序开始处加入 `use strict`。最后，`use sigtrap` 和 `use diagnostics` 编译指示也提供了一些有用的特性，你可以查看它们 的手册页去做进一步的了解。

Regarding aesthetics of code lay out, about the only thing Larry cares strongly about is that the closing curly bracket of a multi-line BLOCK should line up with the keyword that started the construct. Beyond that, he has other preferences that aren't so strong:

在代码排版的美观方面，Larry Wall唯一强烈建议的是，多行代码块之后的右花 括号应该和起始这个块的关键字垂直对齐。另外，他还希望你做到下面这些（尽 管态度不是很强烈）：

- 4 列缩进
- 如果可能的话，左花括号和关键字要写在同一行，不然就另起一行并与关键字对齐。
- 如果一个块有多行的话，左花括号前面应该留有一个空格。
- 只有一行内容的块应该连同花括号一起写在同一行内。
- 分号前不要加空格。
- 在较短的单行块中，应该省略分号。
- 对于大多数运算符来说，应该在两边各加一个空格。
- 较复杂些的下标（数组下标或者 hash 下标）两边应该加上空格。
- 功能相对独立的两个程序段落之间应该留有空行。
- Uncuddled elses. else应该另起一行。
- 函数调用时，函数名称和左圆括号之间不应留任何空白。
- 每个逗号之后都应该加一个空格。
- 长行应该在优先级较低的运算符右面断行。
- Space after last parenthesis matching on current line. 本行最后一个空格后加空格。
- 所有的语句都应该按照层次垂直对齐。
- 在不引起歧义的前提下忽略多余的标点符号。

Larry 制订这些规则时自然有他的原因，不过他也并不要求所有的人都和他完全 一样。（你可以因地制宜地制订出自己的风格）

下面还有一些更多的风格问题我们一起来探讨一下：

- Just because you CAN do something a particular way doesn't mean that you SHOULD do it that way. Perl is designed to give you several ways to do anything, so consider picking the most readable one. For instance

  你“能”做这件事并不表示你就“该”这么做。Perl 对任何事情往往都提供多种方法，最好选择最易读的那种。比如：

  ```
  open(FOO,$foo) || die "Can't open $foo: $!";
  ```

  is better than

  就比这个要好：

  ```
  die "Can't open $foo: $!" unless open(FOO,$foo);
  ```

  because the second way hides the main point of the statement in a modifier. On the other hand

  因为后一种方法把主要的部分隐藏在了后面。而这个：

  ```
  print "Starting analysis\n" if $verbose;
  ```

  is better than

  则比这个好：

  ```
  $verbose && print "Starting analysis\n";
  ```

  because the main point isn't whether the user typed -v or not.

  因为主要的问题并不在于用户是否输入了-v。

  Similarly, just because an operator lets you assume default arguments doesn't mean that you have to make use of the defaults. The defaults are there for lazy systems programmers writing one-shot programs. If you want your program to be readable, consider supplying the argument.

  类似的，一些操作符有默认值并不表示你一定要采用它。默认设置是为了给那些懒鬼系统程序员用来写一次性程序用的。如果你希望你的程序更加易读，最好提供明确的参数。

  Along the same lines, just because you CAN omit parentheses in many places doesn't mean that you ought to:

  同样道理，有些地方是可以省略括号，但你没必要非省略它们不可：

  ```
  return print reverse sort num values %array;
  return print(reverse(sort num (values(%array))));
  ```

  When in doubt, parenthesize. At the very least it will let some poor schmuck bounce on the % key in vi.

  如果可能存在歧义，就用括号。

  Even if you aren't in doubt, consider the mental welfare of the person who has to maintain the code after you, and who will probably put parentheses in the wrong place.

  即使如果此处没有歧义，也考虑一下在你之后维护这段代码的人的精神健康，他甚至可能会把括行放错地方。

- Don't go through silly contortions to exit a loop at the top or the bottom, when Perl provides the `last` operator so you can exit in the middle. Just "outdent" it a little to make it more visible:

  不要刻意要在循环的顶部或者底部退出，perl提供`last`操作符让你可以在中间退出。可以让它突出来一些以便更显眼。

```
LINE:
    for (;;) {
        statements;
      last LINE if $foo;
        next LINE if /^#/;
        statements;
    }
```

- Don't be afraid to use loop labels--they're there to enhance readability as well as to allow multilevel loop breaks. See the previous example.

  使用循环标记也没关系，因为它们就是用来增加可读性以及设计多层循环用的。看前面这个例子。

- Avoid using grep() (or map()) or `backticks` in a void context, that is, when you just throw away their return values. Those functions all have return values, so use them. Otherwise use a foreach() loop or the system() function instead.

  尽量避免在空环境下使用grep() (或者map())和``，因为使用它们是为了得到它们的返回值。而如果你不用它们的返回值，那不如就用 foreach 循环或者 system() 函数好了。

- For portability, when using features that may not be implemented on every machine, test the construct in an eval to see if it fails. If you know what version or patchlevel a particular feature was implemented, you can test `$]` (`$PERL_VERSION` in `English`) to see if it will be there. The `Config` module will also let you interrogate values determined by the Configure program when Perl was installed.

  出于可移植性的考虑，当使用老版本没有的新特性的时候，用 eval 测试一下是否会失败。如果你知道是在哪个版本才加进来的，你可以用`$]` (在c<English>中是`$PERL_VERSION`) 测试看看。`Config` 模块也可以让你得到 Perl 安装时 Config 模块的各种参数。

- Choose mnemonic identifiers. If you can't remember what mnemonic means, you've got a problem.

  选择容易记忆的变量名。如果你自己都记不住你的变量名，肯定就会有问题。

- While short identifiers like $gotit are probably ok, use underscores to separate words. It is generally easier to read $var_names_like_this than $VarNamesLikeThis, especially for non-native speakers of English. It's also a simple rule that works consistently with VAR_NAMES_LIKE_THIS.

  虽然短名字像 $gotit 这样的也还凑合，但是最好还是用下划线分隔一下单词。$var_name 总比 $VarNamesLikeThis 要易读，特别是非英语国家的人。即使当变量名变成 VAR_NAME_LIKE_THIS 的时候这个规则仍然适用。

  Package names are sometimes an exception to this rule. Perl informally reserves lowercase module names for "pragma" modules like `integer` and `strict`. Other modules should begin with a capital letter and use mixed case, but probably without underscores due to limitations in primitive file systems' representations of module names as files that must fit into a few sparse bytes.

  模块名有时则会例外。Perl 一般将小写字母的名字留给 pragma 模块，像`integer`和`strict`。其他模块则应该用大写字母开头，同时使用混合大小写的风格，而且没有下划线，因为某些原始的系统不能支持很长的文件名。

- You may find it helpful to use letter case to indicate the scope or nature of a variable. For example:

  有时有用字符的大小写来表示变量的性质和作用域也是有用的，比如：

  ```
  $ALL_CAPS_HERE   constants only (beware clashes with perl vars!)
  $Some_Caps_Here  package-wide global/static
  $no_caps_here    function scope my() or local() variables

  $ALL_CAPS_HERE   常量（小心于perl内置变量冲突）
  $Some_Caps_Here  包内静态量/常量
  $no_capts_here   函数作用域的 my() 或者local() 变量
  ```

  Function and method names seem to work best as all lowercase. E.g., $obj->as_string().

  You can use a leading underscore to indicate that a variable or function should not be used outside the package that defined it.

  函数名和方法最好使用全小写，比如：$obj->as_string()。

- If you have a really hairy regular expression, use the /x modifier and put in some whitespace to make it look a little less like line noise. Don't use slash as a delimiter when your regexp has slashes or backslashes.

  如果你有一个相当繁琐的正则表达式，用/x修饰符以及一些空格让它们更容易理解一些。当你的正则表达式中有斜线或反斜线时，不要用斜线作为分隔符。

- Use the new "and" and "or" operators to avoid having to parenthesize list operators so much, and to reduce the incidence of punctuation operators like && and ||. Call your subroutines as if they were functions or list operators to avoid excessive ampersands and parentheses.

  用新的 and 和 or 操作符来避免使用过多的括号以及&&和||。像用列操作符一样调用你的函数来避免过多的括号。

- Use here documents instead of repeated print() statements.

  使用本地文档代替反复的print()语句。

- Line up corresponding things vertically, especially if it'd be too long to fit on one line anyway.

  垂直对齐相应的部分，尤其是当一行写不下的时候。

  ```
  $IDX = $ST_MTIME;
  $IDX = $ST_ATIME       if $opt_u;
  $IDX = $ST_CTIME       if $opt_c;
  $IDX = $ST_SIZE        if $opt_s;

  mkdir $tmpdir, 0700 or die "can't mkdir $tmpdir: $!";
  chdir($tmpdir)      or die "can't chdir $tmpdir: $!";
  mkdir 'tmp',   0777 or die "can't mkdir $tmpdir/tmp: $!";
  ```

- Always check the return codes of system calls. Good error messages should go to STDERR, include which program caused the problem, what the failed system call and arguments were, and (VERY IMPORTANT) should contain the standard system error message for what went wrong. Here's a simple but sufficient example:

  始终检查系统调用的返回值。有用的出错信息应该输出到STDERR，包括哪个程序出了错，哪个调用有问题，参数是什么，而且（非常重要！）要包括标准的系统错误信息。一下是一个简单不过完整的例子：

```
opendir(D, $dir)     or die "can't opendir $dir: $!";
```

- Line up your transliterations when it makes sense:

  ```
  tr [abc]
     [xyz];
  ```

  如果可以的话，字符映射应该对齐。

- Think about reusability. Why waste brainpower on a one—shot when you might want to do something like it again? Consider generalizing your code. Consider writing a module or object class. Consider making your code run cleanly with `use strict` and `use warnings` (or −w) in effect. Consider giving away your code. Consider changing your whole world view. Consider... oh, never mind.

  考虑一下可重用性，当你可能会再用到它的时候，为什么浪费精力写一次性的程序呢？扩展你的程序，并考虑一下写一个模块或者类。看看能否在`use strict`和`use warnings`（或B −w<>）下运行你的程序。试试看发布你的程序，试试看改变你的世界观。试试看。。。oh，还是算了。

- Be consistent.

  坚持不懈。

- Be nice.

  做个好人。

3 TRANSLATORS

周杰 "klaus" <zhoujie@fudan.edu.cn> "flw" <flw@cpan.org>
中国perl协会： www.perlchina.org