将一个列表当成一个标量，中括号[]能够实现这个转换：

实例：

$wife{“Jacob”} = [“Leah”, “Rachel”, “Bilhah”, “Zilpah”];

这个语句创建了一个未命名的数组，并将这个数组的引用放入散列元素$wife{“Jacob”}中。因此我们有了一个命令的散列，其中包含一个未命名的数组，这就是Perl处理多维数组和嵌套数据类型的方法。同普通数组和散列的赋值方法一样，可以单独对其进行赋值：

实例：

$wife{“Jacob”}[0] = “Leah”;

$wife{“Jacob”}[1] = “Rachel”;

$wife{“Jacob”}[2] = “Bilhah”;

$wife{“Jacob”}[3] = “Zilpah”;

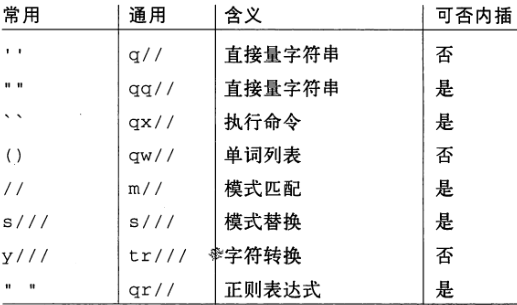
**chop函数：**

chop函数不加区别的除去字符串的最后一个字符，并将结果返回。

chomp函数仅删除结束标记，同时返回被删除的字符数。

**比较操作符：**





**“此处”文档**

指定用以结束引起材料的字符串跟在一个<<后面，所有当前行到结束行(不包括)之间的行都是字符串的内容。结束字符串可以是标识符(单词)或者某些引起的文本。如果结束符被引起，引起的类型决定文本的变换，就像普通的引起一样。没有引起的标识符当作用双引号引起对待。反斜杠标识符当作用单引号引起。在<<和未引起的标识符之间不能有空白，不过如果一个带引号的字符串做标识符，则可以有空白(如果插入空白，它会被当作一个空标识符，这样做是允许的，但它会和第一个空白行匹配)。结束字符串必须在终止行独立出现，不带引号以及两边没有多余的空白。

实例：

print <<EOF;

The price is $Price. # $Price的值可以被解析。

EOF

print <<”EOF”;

The price is $Price.

EOF

print <<’EOF’;

The price is $Price; #$Price的值不会被解析。

EOF

print << x 10;

The Camels are coming! Hurrah! Hurrah!

#上面一行是一定要加上的，不加报错。可什么也没有，但一定要有这一行。

print <<” ” x 10;

The Camels are coming! Huarrah! Hurrah!

#上面一行开头有一个空格，且这一行和这个空格是一定要有的。不然报错。

print <<`EOF`; #执行命令

echo hi there

echo lo there

EOF

print <<”dromedary”, <<”camelid”;#可以堆叠

I said bactrian.

dromedary

She said llamd.

camelid

funkshun(<<"THIS",23,<<"THAT");

Here's a line

or two.

THIS

And here's another.

THAT

($quote = <<'QUOTE') =~ s/^\s+//gm;

The Road goes ever on and on,

down form the door where it began.

QUOTE

print $quote;

@sauces = <<End\_Lines =~ m/(\S.\*\S)/g;

normal tomato

spicy tomato

green chile

pesto

white wine

End\_Lines

print @sauces;

**V-字符串直接量**

一个以v开头，后面跟着一个或多个局点分隔的整数直接量，会被当作一个字符串直接量。

实例：

$crlf = v13.10; # ASCII 码回车，换行。

v-字符串在表示IP地址和版本号时很有用。尤其是在字符可以拥有大于255的值的今天，v-字符串提供了一个可以表示任意大小的版本，并用简单字符串比较就可以得到正确结果的方法。

**钻石操作符**

当且仅当行输入操作符是一个while循环的**唯一一个**条件时，其值会自动赋予特殊变量$\_。

实例：

while (<STDIN>) { print; } #此时输入在$\_中暂存。

while (<FH1> && <FH2>) { … } #此时两个输入都将被丢弃。

for (; <STDIN> ;) { print ; } #此时输入在$\_中暂存。

**箭头操作符**

二元箭头操作符 -> 是一个中缀析值操作符，如果右边是一个[…]数组脚标、一个{…}散列脚标，或者一个(…)子例程参数列表，那么左边必须是一个对应的数组、散列或者子例程的引用(硬引用或者符号引用都可)。

实例：

$aref->[42]; #一个数组的析值

$href->{"corned beff"}; #一个散列析值

$sref->(1,2,3); #一个子例程析值 子例程也就是函数吧。这个应该是传参数

或者是某种类型的方法调用，右边必须是一个方法名(或者包含该方法名的简单标量变量)，而且左边必须得出一个对象名或者一个类的名字。

实例：

$yogi = Bear->new("Yogi"); #一个类方法调用

$yogi->swipe($pinic); #一个对象方法调用

**!~操作符**

二元!~操作符类似=~操作符，其返回值是=~对应的返回值的逻辑非。

实例：下面表达式功能相同

$string !~ /pattern/;

not $string =~ /pattern/;

**x操作符**

标量环境中：

print '-' x 80; #打印一行划线

print "\t" x ($tab/8), ' ' x ($tab%8); #跳过

在列表环境中，如果左操作符是在小括号中的列表，x的作用是一个列表复制器，而不是字符串复制器。

实例：

@ones = (1) x 80; #一个有80个1的列表

@ones = (5) x @ones; #把所有元素设置为5

x操作符用于初始化数组和散列片段

实例：

@keys = qw(perls before swine);

@hash{@keys} = (" ") x @keys;

**范围操作符**

范围操作符：..根据环境的不同实际上是两种不同的操作符。

在标量环境里，..返回一个布尔值，只要它的左操作数为假就一直为假。

创建私有变量用my，进行有选择的访问全局变量用our，以及给全局变量提供临时值用local。

**local：**

在全局变量上使用local操作符时，每次执行local时都给该变量一个临时值，但是这并不影响该变量的全局可视性，当程序抵达动态作用域的末尾时，临时值被抛弃并恢复原来的值。但它仍然是一个全局变量，只是在执行那个块的时候碰巧保存了一个临时值而已。如果你在该全局变量包含临时值时调用其它函数，并且该函数访问了这个全局变量，那么它看到的将是临时变量，而不是初始值。

**our：**