



Search

Browse by Category

- 中国开源项目
- Backup

Communication

Database

EMAIL

Embedded Linux

Enterprise/Business

Games

Graphics Engine

Interface

Kernel

Linux distribution

Linux Tools

Multimedia

Network

中国开源项目 :: Software Development :: Programming Languages :: Perl :: perl中Grep函数的使用

## perl中Grep函数的使用

关于grep函数

(如果你是个Perl的新手,你可以先跳过下面的两段,直接到 Grep vs.loops 样例这一部分,放心,在后面你还会遇到它)

grep BLOCK LIST  
grep EXPR, LIST

grep 函数会根据 LIST 中的元素对 BLOCK 或 EXPR 做出评估,而且会把局部变量 \$\_ 设置为当前所用的 LIST 中的元素。BLOCK 块是一个或多个由花括号分隔开的Perl 语句。而 List 则是一串被排序的值。EXPR 是一个或多个变量,操作符,字符,函数,子程序调用的综合体。Grep 会返回一组经 BLOCK 或 EXPR 块的估值后是真的元素。如果 BLOCK 块由多个语句组成,那么 Grep 以 BLOCK 中的最后一条语句的估计值为准。LIST 可以是一个列表也可以是一个数组。在标量上下文中,grep 返回的是可以被 BLOCK 或 EXPR 估为真的元素个数。

请避免在 BLOCK 或 EXPR 块中修改 \$\_ ,因为这会相应的修改 LIST 中的元素。同时还要避免把 grep 返回的列表做为左值使用,因为这也会修改 LIST 中的元素。(所谓左值变量就是一个在赋值表达式左边的变量)。

Article	273
Created	11-3-2007
Author	opensourceproject.org.cn
Rating	★★★★★

Management
☒ Office/Text Editors
☒ Open Source Sponsor
☒ Science
☒ Security
☒ Software Development
☒ Ajax
☒ Programming Languages
☒ C/C++
☒ Eclipse
☒ Java
☒ Pascal
☑ Perl
☒ PHP
☒ Python
☒ Ruby
☒ SHELL
☒ Windows API
☒ System administration

一些 Perl hackers 可能会利用这个所谓的"特性",但是我建议你不要使用这种混乱的编程风格.

## grep 与 循环

这个例子打印出 myfile 这个文件中含有 terriosm 和 nuclear 的行(大小写不敏感).

```
open FILE "<myfile" or die "Can't open myfile: $!";
print grep /terrorism|nuclear/i, <FILE>;
```

对于文件很大的情况,这段代码耗费很多内存。因为 grep 把它的第二个参数作为一个列表上下文看待,所以 < > 操作符返回的是整个的文件。更有效的代码应该这样写:

```
while ($line = <FILE>) {
    if ($line =~ /terrorism|nuclear/i) { print $line }
}
```

通过上面可以看到,使用循环可以完成所有 grep 可以完成的工作。那为什么我们还要使用 grep 呢?一个直观的答案是 grep 的风格更像 Perl,而 loops (循环)则是 C 的风格。一个更好的答案是,首先, grep 很直观的告诉读者正在进行的操作是从一串值中选出想要的。其次, grep 比循环简洁。(用软件工程的说法就是 grep 比循环更具有内聚力)。基本上,如果你对 Perl 不是很熟悉,随便你使用循环。否则,你应该多使用像 grep 这样的强大工具.

## 计算数组中匹配给定模式的元素个数

在一个标量上下文中,grep 返回的是匹配的元素个数.

```
$num_apple = grep /^apple$/i, @fruits;
```

^ 和 \$ 匹配符的联合使用指定了只匹配那些以 apple 开头且同时以 apple 结尾的元素。这里 grep 匹配 apple 但是 pineapple 就不匹配。

## 输出列表中的不同元素

```
@unique = grep { ++$count{$_} < 2 }
              qw(a b a c d d e f g f h h);
print "@unique\n";
```

输出结果: a b c d e f g h

`$count{$_}` 是 Perl 散列中的一个元素,是一个键值对 ( Perl中的散列和计算机科学中的哈希表有关系,但不完全相同) 这里 `count` 散列的键就是输入列表中的各个值,而各键对应的值就是该键是否使 `BLOCK` 估值为真的次数。当一个值第一次出现的时候 `BLOCK` 的值被估为真 (因为小于2), 当该值再次出现的时候就会被估计为假 (因为等于或大于2)。

### 取出列表中出现两次的值

```
@crops = qw(wheat corn barley rice corn soybean hay
             alfalfa rice hay beets corn hay);
@duplicates = grep { $count{$_} == 2 }
               grep { ++$count{$_} > 1 } @crops;
print "@duplicates\n";
```

在 `grep` 的第一个列表元素被传给 `BLOCK` 或 `EXPR` 块前,第二个参数被当作列表上下文看待。这意味着,第二个 `grep` 将在左边的 `grep` 开始对 `BLOCK` 进行估值之前完全读入 `count` 散列。

### 列出当前目录中的文本文件

```
@files = grep { -f and -T } glob '*.*';
print "@files\n";
```

`glob` 函数是独立于操作系统的,它像 Unix 的 `shell` 一样对文件的扩展名进行估计。单个的 `*` 表示匹配所以当前目录下不以 `.` 开头的文件, `.*` 表示匹配当前目录下以 `.` 开头的文件。如果一个文件是文本文件 `-f` 和 `-T` 文件测试符则返回真。使用 `-f and -T` 进行测试要比单用 `-T` 进行测试有效,因为如果一个文件没有通过 `-f` 测试,那么 `-T` 测试就不会进行,而 `-f` 测试比 `-T` 耗时更少。

### 从数组中选出元素并消除重复

```
@array = qw(To be or not to be that is the question);
```

```
print "@array\n";
@found_words =
    grep { $_ =~ /b|o/i and ++$counts{$_} < 2; } @array;
print "@found_words\n";
```

输出结果:

To be or not to be that is the question

To be or not to questio

逻辑表达式 `$_ =~ /b|o/i` 匹配包含有 `b` 或 `o` 的元素(区别大小写)。把匹配操作放在计数工作前要比把计数工作放在前面有效些。比如,如果左边的表达式测试失败,那么右边的表达式就不会被计算。

## 选出二维坐标数组中横坐标大于纵坐标的元素

```
# An array of references to anonymous arrays
@data_points = ( [ 5, 12 ], [ 20, -3 ],
                 [ 2, 2 ], [ 13, 20 ] );
@y_gt_x = grep { $_->[0] < $_->[1] } @data_points;
foreach $xy (@y_gt_x) { print "$xy->[0], $xy->[1]\n" }
```

输出结果:

5, 12

13, 20

## 在数据库中查找餐馆

这个例子实现数据库的方法不适合在实际中使用的,但是它说明了使用 `grep` 函数的时候,只要你的内存够用, `BLOCK` 块的复杂度基本没有限制。

```
# @database is array of references to anonymous hashes
@database = (
    { name      => "Wild Ginger",
      city      => "Seattle",
      cuisine   => "Asian Thai Chinese Korean Japanese",
      expense   => 4,
      music     => "\0",
      meals     => "lunch dinner",
      view      => "\0",
      smoking   => "\0",
```

```

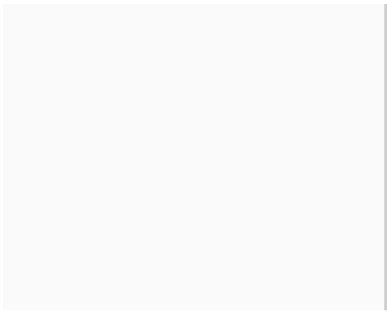
        parking    => "validated",
        rating     => 4,
        payment    => "MC VISA AMEX",
    },
# { ... }, etc.
);




sub findRestaurants {
    my ($database, $query) = @_;
    return grep {
        $query->{city} ?
            lc($query->{city}) eq lc($_->{city}) : 1
        and $query->{cuisine} ?
            $_->{cuisine} =~ /$query->{cuisine}/i : 1
        and $query->{min_expense} ?
            $_->{expense} >= $query->{min_expense} : 1
        and $query->{max_expense} ?
            $_->{expense} <= $query->{max_expense} : 1
        and $query->{music} ? $_->{music} : 1
        and $query->{music_type} ?
            $_->{music} =~ /$query->{music_type}/i : 1
        and $query->{meals} ?
            $_->{meals} =~ /$query->{meals}/i : 1
        and $query->{view} ? $_->{view} : 1
        and $query->{smoking} ? $_->{smoking} : 1
        and $query->{parking} ? $_->{parking} : 1
        and $query->{min_rating} ?
            $_->{rating} >= $query->{min_rating} : 1
        and $query->{max_rating} ?
            $_->{rating} <= $query->{max_rating} : 1
        and $query->{payment} ?
            $_->{payment} =~ /$query->{payment}/i : 1
    } @$database;
}

%query = ( city => 'Seattle', cuisine => 'Asian|Thai' );
@restaurants = findRestaurants(\@database, \%query);
print "$restaurants[0]->{name}\n";

```

输出结果: Wild Ginger



 Email  Print  Add Comment

How helpful was this article to you?  Rate

Copyright @ 2007 - 2009 OpenSourceProject.org.cn. All rights reserved. 版权所有[OpenSourceProject.org.cn](http://OpenSourceProject.org.cn)

