Perl 命令行常用参数 2017.8.29

执行控制：

-e 指定字符串以作为脚本（多个字符串迭加）执行
-M 导入模块
-I 指定目录以搜索标准位置前的模块

整洁性：

-w 打开警告
-Mstrict 打开严格编译指示（pragma）

数据：

-0 （这是个零）指定输入记录分隔符
-a 将数据分割成名为 **@F** 的数组
-F 指定分割时 -a 使用的模式（请参阅 perldoc -f split）
-i 在适当的位置编辑文件（请参阅 perldoc perlrun 以获取大量详细信息）
-c 进行 perl 的语法检查，但不执行 perl 命令。
-l 使用 -l 有两个效果，第一自动 chomp 输入分隔号，第二把$/值付给$\(这样 print 的时候就会自动在末尾加\n)。
-n 使用 **<>** 将所有 @ARGV 参数当作文件来逐行运行，会将读入的内容隐式的逐一按行来遍历文件，每一行将缺省保存在 **$_**。
-p 和 –n 一样，但是还会打印 **$_** 的内容。

请注意：**-p** 开关（打印）和 **-n** 开关（不打印，内存 cache 中）的使用：
当您想显式打印数据时，使用 –n 开关。
-p 开关隐式地将 print $_ 语句插入到 –n 开关所产生的循环中。
-p 开关更适用于对文件进行的**完全处理**（可结合**-i** 备份文件）。
–n 开关更适用于**选择性文件处理**，这样的处理只需打印**特定数据**。

以下是一些常用的简单实例

```
# perl –pi –e ‘s/aaa/bbb/’ filename
```

修改当前文件中的内容，不生成中间文件，速度很快。记住**-i** 开关，因为它让你原地编辑文件。

```
# perl –ne ‘print if /^aaaaa/’ filename
```

像 **grep** 一样过滤文件中需要的内容。这个地方使用了-n，所以一次是做一行的操作，直到整个文件读完。另外在管道时，-n 也会一样来遍历管道送过来的内容。

```
# perl -n -e ‘print “$. – $_”’ filename
```

这个例子中的，没用 -ne，只是命令写成了-n –e，其实一样，这个例子中是给当前文件中的内容都加一个行号打印出来。

*注：**$.** 表示**当前行号***

perl -pe ‘$_ = “$. $_”’ filename
这个其实和上面一样，分别只是使用了-p 替换了-n，这个有个什么好处啦，别的地方都一样，但 –p 按行来遍历完文件后，会给$_打印出来。

大家还记得 awk 分割域(awk ‘{ i = NF – 1; print $1 + $i }’)啊，是不是很方便，下面我们来看看 perl
perl –lane ‘print $F[0] + $F[-2]’ filename
这个神奇的地方在于-a，使用-a 后，因为-n 分行读进来，然后-a 给数据分割成@F 的数组。

perl -ne ‘print if /^START$/ .. /^END$/’ filename
打印正则中从$start 到$end 的地方

perl -ne ‘print if $. >= 15; exit if $. >= 17;’ filename
有效地打印数字范围中的行

perl -p -i.bak -e ‘s/\bfoo\b/bar/g’ *.c
原地修改：**-i** 开关的神奇之处在于它对 @ARGV 中的每个文件都用该脚本对该文件输出所产生的文件版本进行替代。

perl -ne ‘print scalar reverse $_’ test
给文件中的内容**反向排序**，比如文件中有 fukai，就会变成 iakuf

常用参数说明

第一部分：Safety Net Options 安全网参数

在使用 Perl 尝试一些聪明(或 stupid)的想法时,错误难免会发生。有经验的 Perl 程序员常常使用**三个参数**来提前找到错误所在。

1：**-c**
这个参数编译 Perl 程序但不会真正运行它，由此检查所有语法错误。每次修改 perl 程序之后（我都会立刻使用它来找到任何语法错误。
$ perl –c program.pl

2：**-W**
它会提示你任何潜在的问题。Perl 5.6.0 之后的版本已经用 **use warnings;** 替换了-w。你应该使用 **use warnings;**因为它要比 –w 更灵活。

3：**-T**
它把 perl 放到了 **tain 模式**。在这个模式里，Perl 会质疑任何程序外传来的数据。例如，从 Perl 命令行读取，外部文件里读取或是 CGI 程序里传来的数据。
这些数据在-T 模式里都会被 Tainted 掉。

第二部分：让短小的 Perl 程序运行在 Perl 命令行

1：-e

可以让 Perl 程序在 Perl 命令行上运行

例如，我们可以在 Perl 命令行上运行"HelloWorld"程序而不用把它写入文件再运行。

```
$ perl –e ‘print “Hello World\n”’
```

多个 –e 也可以同时使用，运行顺序根据它出现的位置。

```
$ perl –e ‘print “Hello”; ‘-e’ print “World\n”’
```

像所有的 Perl 程序一样，只有程序的最后一行不需要以 ; 结尾。

2：-M

可以像通常一样引用模块

```
$ perl -MLWP::Simple –e ‘getstore(“http://www.163.com/”, “163.html”)’
# 下载整个网页
```

-M+模块名和 use 模块名一样

第三部分：隐式循环

1：-n

增加了循环的功能，使你可以一行一行来处理文件

```
$ perl –n –e ‘print;’ 1.txt    # $ perl –ne ‘print;’ 1.txt
```

这与下面的程序一样。

```
LINE: while (<>) { print; }    # <>打开 Perl 命令行里的文件，一行行的读取。
```

每一行缺省保存在$_。

```
$ perl –n –e ‘print “$.-$_”’ filename
```

上面的这一行可以写成

```
LINE: while (<>) { print “$.-$_”; }          # 输出当前行数$.和当前行$_
```

2：-p，和-n 一样，但是还会打印$_的内容

如果想在**循环的前后**做些处理,可以使用 **BEGIN** 或 **END** block。

下面的这一行计算文件里的字数：

```
$ perl –ne ‘END { print $t; } @w = /(\w+)/g; $t += @w’ filename
```

每一行所有匹配的字放入数组 @w，然后把 @w 的元素数目递加到**$t**（标量上下文）。**END** block 里的 print 最后输出文件总字数。

还有两个参数可以让这个程序变得更简单。

3：-a

打开自动分离(split)模式。**空格**是缺省的分离号。输入根据分离号被分离然后放入缺省数组 **@F**

使用 -a，上面的命令可以写成这样：

```
$ perl –ane ‘END { print $x; } $x += @F’ filename
```

4：**-F**

把缺省的分离号改为你想要的（类似 **awk -F**）。

例如把分离号定为**非字符**，上面的命令可以改为：

```
$ perl –F ‘\W’ –ane ‘END { print $x; } $x += @F’ filename
```

下面通过**/etc/passwd** 文件来介绍一个复杂的例子。**/etc/passwd** 是文本文件，每一行是一个用户记录，由冒号:分离，第 7 行是用户的登录 shell 路径。我们可以得出每一个不同 shell 路径被多少个用户使用：

```
$ perl –F ‘:’ –ane ‘$s{$F[6]}++; ‘\>; -e’ END { print “$_:$s{$_}” for key
s%s }’ /etc/passwd
```

虽然现在不是一行,但是你可以看出使用参数可以解决什么问题.

第四部分：Record Separators 数据分隔符

$/：输入分隔符

$：输出分隔符

$/ 用来分隔从文件句柄里读入的数据，缺省$/分隔号是**\n**，这样每次从文件句柄里就会一行行的读取。

$\ 缺省是**空字符**，用来自动加到要 print 的数据尾端，这就是为什么很多时候 print 都要在末尾加上\n。

$/ 和 $\ 可与-n，-p 一起使用。在 Perl 命令行上相对应为**-0**(零)和**-l**(这是 L)。

-0：后面可以跟一个 **16 进制**或 **8 进制**数值,这个值用来赋给$/。

-00：打开段落模式

-0777:打开 **slurp 模式**(即可以一次把整个文件读入),这与把$/设为空字符和 undef 一样效果。

单独使用-l 有两个效果：

第一：自动 chomp 输入分隔号

第二：把**$/**值赋给**$**（这样 print 的时候就会自动在末尾加\n）

-l 参数，用来给每一个输出加\n

```
$ perl –le ‘print “Hello World”’
```

第五部分：原位编辑

1：使用已有的参数我们可以写出很有效的 Perl 命令行程序。常见的 Unix I/O 重定向：

```
$ perl –pe ‘somecode’ input.txt > output.txt
```

这个程序从 input.txt 读取数据,然后做一些处理再输出到 output.txt。你当然也可以把输出重定向到同一个文件里。

上面的程序可以通过-i 参数做的更简单些。

2：**-i**

把源文件更名，然后从这个更名的源文件里读取。最后把处理后的数据写入源文件。
如果 -i 后跟有其他字符串，这个字符串与源文件名合成后来生成一个新的文件名。
此文件会被用来储存原始文件，以免被 -i 参数**覆盖**。

这个例子把所有 php 字符替换为 perl:

```
$ perl –i –pe ‘s/\bPHP\b/Perl/g’ filename
```

程序读取文件的每一行，然后替换字符，处理后的数据重新写入(即覆盖)源文件。

如果不想覆盖源文件,可以使用：

```
$ perl –I .bak –pe ‘s/\bPHP\b/Perl/g’ file.txt
```

这里处理过的数据写入 file.txt, file.txt.bak 是源文件的备份。

Perl 经典的例子

问题：
aaa@domain.com 2
aaa@domain.com 111
bbb@home.com 2222
bbb@home.com 1

类似这种输出，我想把他们变换成下面形式：
aaa@domain.com 113
bbb@home.com 2223

就是将相同邮箱名称后面的数字相加。各位大侠能否给些思路如何用 perl 来实现。

答案：

```
$ perl –anle \
  ‘$cnt{$F[0]} += $F[1]; END { print “$_\t$cnt{$_}” for keys %cnt }’ \
  urfile
```

每次读取 urfile 的一行，由于使用了-a，打开自动分离(split)模式。
空格是缺省的分离号，输入根据分离号被分离然后放入缺省数组 @F 中。
以文件的第一行为例子，$F[0] 就是 aaa@domain.com，$F[1] 就是 2
$cnt{$F[0]} += $F[1] 就是一个哈希，以$F[0]为 key，$F[1]为 value,把相同 key 的数值都叠加起来。然后把文件的每一行都这样处理一次。

END {} 就是在循环完之后再处理，里面的意思就是打印这个%cnt 哈希。

下面的是上面行命令的文本形式：

```
#!/usr/bin/perl

use warnings;
use strict;

my %hash;

while (<>) {
    chomp;
    my @array = split;
    $hash{$array[0]} += $array[1];
}

END {
    foreach (keys %hash) {
        print “$_\t$hash{$_}\n”;
    }
}
```