[GNU Awk 4.0: Teaching an Old Bird Some New Tricks](http://blog.chinaunix.net/uid-14293861-id-2977155.html)

简单翻译:  
  
gawk 4.0中加入的新特性:  
  
1.现在,gawk提供了引用外部脚本文件的机制.@include "文件名"这样的语句可以让gawk把该文件包含进来.和命令行的-f 参数类似,gawk会在当前目录寻找该文件.嵌套引用也是支持的,而且gawk不会引用相同的脚本两次.同时,伴随gawk很多年的igawk脚本将会被废弃  
  
2.两个新的模式BEGINFILE和ENDFILE被添加进来,BEGINFILE模式匹配的行为在每个文件的第一个记录被读取之前运行,如果一个文件不存在或者不能被打开,gawk会因致命错误退出程序,如果在BEGINFILE中添加检测语句,发现文件打不开时,执行nextfile进入到下一个文件,则程序会继续运行.ENDFILE行为可以在每个文件读取完之后做一些善后工作.  
  
3.你可以通过一个字符串变量间接的调用一个函数,例如  
  
function foo(a,b){ .... }  
function bar(a,b){ .... }  
BEGIN{  
fun="foo";@fun(1, 2) #间接调用foo(1,2)  
fun="bar";@fun(1, 2) #间接调用bar(1,2)  
}  
  
4.gawk已经支持真正的多位数组a[x][y],而不是以前通过不可见字符连接多个索引来模拟多维数组.并且和C以及其他编译型语言不同,awk的多维数组不要求是矩形的.  
  
5.默认支持了switch/case语句.gawk很久以前就有了switch语句,不过除非在编译的时候指定选项,否则是不可用的.  
  
6.通过FPAT匹配字段内容确定字段而不仅仅是通过FS匹配字段间隔来确定字段.前者类似于match函数匹配$0得到的数组,后则类似于split函数分割$0得到数组  
  
7.gawk网络编程中支持ipv6  
  
8.对应每个长选项,都一个短选项与之对应.  
  
9.默认激活范围表达式:{1,9},由于为了兼容其他awk版本.gawk 4.0以前只有在指定--posix参数或者--re-interval参数时,范围表达式才可以使用  
  
  
接下来将要进行的工作:  
  
几个重要的特性将会加入到gawk中.  
  
1.XMLGawk项目是基于gawk3.1.6产生的gawk分支版本.它提供更好的方式去加载awk动态扩展库以及更好的运行他们,这些都将会分别加入到gawk以及发行版中.  
  
2.尽管gawk在很多年前就有了加载动态扩展库的能力,但是该API一直没有趋于稳定并且不易使用,我已经重新设计了一套API使得在awk程序中可以调用c语言写成的函数.下一步我将会发布这个API  
  
3.目前gawk发行版中包含三个独立的可执行文件,普通的gawk,分析awk程序的pgawk,以及调试awk程序的dgawk.新的gawk核心使得将这三个程序合成为一个并且能通过参数分别运行他们成为可能.这将会简化编译步骤并且简化安装过程.  
  
4.更新文档中过时的内容以及程序例子.

大家可以到GNU的ftp上下载下来爽一爽， <ftp://ftp.gnu.org/gnu/gawk>，粗略的看了下介绍，新版本的gawk功能更强大了！！！  
下面是4.0.0版本gawk的一些新的features（测试了下部分功能）：  
<http://lists.gnu.org/archive/html/info-gnu/2011-06/msg00013.html>

Copyright (C) 2010, 2011 Free Software Foundation, Inc.  
  
   Copying and distribution of this file, with or without modification,  
   are permitted in any medium without royalty provided the copyright  
   notice and this notice are preserved.  
  
Changes from 3.1.8 to 4.0.0  
---------------------------  
  
1. The special files /dev/pid, /dev/ppid, /dev/pgrpid and /dev/user are  
   now completely gone. Use PROCINFO instead.  
  
2. The POSIX 2008 behavior for `sub' and `gsub' are now the default.  
   THIS CHANGES BEHAVIOR!!!!

1. echo '11122211' |awk '{sub(/1{3}/,"")}1'
2. 22211

复制代码

3. The \s and \S escape sequences are now recognized in regular expressions.

1. echo '111 222  11' |awk '{gsub(/\s/,"")}1'
2. 11122211

复制代码

4. The split() function accepts an optional fourth argument which is an array  
   to hold the values of the separators.

1. echo '111-222|33' |awk '{split($0,a,/[-|]/,seps);print "a[1] = "a[1] RS "a[2] = "a[2] RS "a[3] = "a[3] RS "spes[1] = "seps[1] RS "speS[2] = "seps[2]}'
2. a[1] = 111
3. a[2] = 222
4. a[3] = 33
5. spes[1] = -
6. speS[2] = |

复制代码

5. New -b / --characters-as-bytes option that means "hands off my data"; gawk  
   won't try to treat input as a multibyte string.  
  
6. New --sandbox option; see the doc.

1. --sandbox
2. Disable the system() function, input redirections with getline, output redirections with print and printf, and dynamic extensions. This is particularly useful when you want to run awk scripts from questionable sources and need to make sure the scripts can't access your system (other than the specified input data file).

复制代码

7. Indirect function calls are now available.

1. --With indirect function calls, you tell gawk to use the value of a variable as the name of the function to call.

复制代码

8. Interval expressions are now part of default regular expressions for  
   GNU Awk syntax.  
  
9. --gen-po is now correctly named --gen-pot.  
  
10. switch / case is now enabled by default. There's no longer a need  
    for a configure-time option.

1. --Control flow in the switch statement works as it does in C.
2. seq 10 |awk '{switch ($0%2){
3. case "0":
4. print "even number: "$0;break
5. default:
6. print "odd number: "$0
7. }
8. }'
9. odd number: 1
10. even number: 2
11. odd number: 3
12. even number: 4
13. odd number: 5
14. even number: 6
15. odd number: 7
16. even number: 8
17. odd number: 9
18. even number: 10

复制代码

11. Gawk now supports BEGINFILE and ENDFILE. See the doc for details.  
  
--The body of the BEGINFILE rules is executed just before gawk reads the first record from a file. FILENAME is set to the name of the current file, and FNR is set to zero.  
--The ENDFILE rule is called when gawk has finished processing the last record in an input file. For the last input file, it will be called before any END rules. (这两个功能真的很酷,尤其是在处理多个文件时，如下面：)

1. head f1 f2
2. ==> f1 <==
3. aaa
4. bbb
5. ccc
6. ==> f2 <==
7. aaa
8. bbb
9. ccc
10. awk 'BEGIN{print"BEGIN: ---"}BEGINFILE{print "\nBEGINFILE: +++"}{print}ENDFILE{print"ENDFILE: +++\n"}END{print"END: ---"}' f1 f2
11. BEGIN: ---
12. BEGINFILE: +++
13. aaa
14. bbb
15. ccc
16. ENDFILE: +++
17. BEGINFILE: +++
18. aaa
19. bbb
20. ccc
21. ENDFILE: +++
22. END: ---

复制代码

12. Directories named on the command line now produce a warning, not  
    a fatal error, unless --posix or --traditional.  
  
13. The new FPAT variable allows you to specify a regexp that matches  
    the fields, instead of matching the field separator. The new patsplit()  
    function gives the same capability for splitting.  
  
--The value of FPAT should be a string that provides a regular expression. This regular expression describes the contents of each field.

1. echo '111-222|33' |awk -vFS="[-|]" '{print "$1 = "$1 RS "$2 = "$2 RS "$3 = "$3}'
2. $1 = 111
3. $2 = 222
4. $3 = 33
5. #如果用FPAT呢？
6. echo '111-222|33' |awk -vFPAT="[^-|]+" '{print "$1 = "$1 RS "$2 = "$2 RS "$3 = "$3}'
7. $1 = 111
8. $2 = 222
9. $3 = 33

复制代码

14. All long options now have short options, for use in `#!' scripts.  
  
15. Support for IPv6 added via /inet6/... special file. /inet4/... forces  
    IPv4 and /inet chooses the system default (probably IPv4).  
  
16. Added a warning for /[:space:]/ that should be /[[:space:]]/.  
  
17. Merged with John Haque's byte code internals. Adds dgawk debugger and  
    possibly improved performance.  
  
18. `break' and `continue' are no longer valid outside a loop, even with  
    --traditional.  
  
19. POSIX character classes work with --traditional (BWK awk supports them).  
  
20. Nuked redundant --compat, --copyleft, and --usage long options.  
  
21. Arrays of arrays added. See the doc. (这个更强！)

1. awk 'BEGIN{arr["a"]["b"]=1;arr["a"]["c"]=2;
2. for( i in arr)
3. for( j in arr[i])
4. print i,j,arr[i][j]
5. }'
6. a b 1
7. a c 2

复制代码

22. Per the GNU Coding Standards, dynamic extensions must now define  
    a global symbol indicating that they are GPL-compatible. See  
    the documentation and example extensions.  
    THIS CHANGES BEHAVIOR!!!!  
  
23. In POSIX mode, string comparisons use strcoll/wcscoll.  
    THIS CHANGES BEHAVIOR!!!!  
  
24. The option for raw sockets was removed, since it was never implemented.  
  
25. If not in POSIX mode, gawk turns ranges of the form [d-h] into  
    [defgh] before compiling a regexp.  Maybe this will stop all the  
    questions about [a-z] matching uppercase letters.  
    THIS CHANGES BEHAVIOR!!!!  
  
26. PROCINFO["strftime"] now holds the default format for strftime().  
  
27. Updated to latest infrastructure: Autoconf 2.68, Automake 1.11.1,  
    Gettext 0.18.1, Bison 2.5.  
  
28. Many code cleanups. Removed code for many old, unsupported systems:  
        - Atari  
        - Amiga  
        - BeOS  
        - Cray  
        - MIPS RiscOS  
        - MS-DOS with Microsoft Compiler  
        - MS-Windows with Microsoft Compiler  
        - NeXT  
        - SunOS 3.x, Sun 386 (Road Runner)  
        - Tandem (non-POSIX)  
        - Prestandard VAX C compiler for VAX/VMS  
        - Probably others that I've forgotten  
  
29. If PROCINFO["sorted\_in"] exists, for(iggy in foo) loops sort the  
    indices before looping over them.  The value of this element  
    provides control over how the indices are sorted before the loop  
    traversal starts. See the manual.  
  
30. A new isarray() function exists to distinguish if an item is an array  
    or not, to make it possible to traverse multidimensional arrays.  
  
31. asort() and asorti() take a third argument specifying how to sort.  
    See the doc.

[awk的数组之升级版](http://bbs.chinaunix.net/thread-4251429-1-1.html)

这里主要讲是gawk4.0+版本中关于数组的2种新用法，所以还没有升级版本的筒子们赶快动起来吧。   
  
1. 预定义遍历的数组  
  
一般情况下，用for item in array的方法输出数组的值，其顺序是没有定义的，也就是”乱序的“。但是很多时候，我们希望  
数组的值按照一定的要求输出，比如按照数值大小升序或降序的方式等等。此时，一般的做法，是通过asort或asorti来间接实现。  
不过，现在好了，gawk4.0+版本提供了非常方便的对数组输出的控制模式。   
这里涉及到一个gawk的一个内置数组PROCINFO，大家可以运行这个查看它的详细信息：

awk 'BEGIN{for(i in PROCINFO){if(isarray(PROCINFO[i])){for( j in PROCINFO[i])print i,j,PROCINFO[i][j]}else{print i,PROCINFO[i]}}}'

复制代码

其中控制数组遍历模式的是"sorted\_in"，如下面的列表:

|  |  |
| --- | --- |
| ROCINFO ["sorted\_in"] | Description |
| @unsorted | Array indexes are processed in arbitrary order (default awk behavior). |
| @ind\_str\_asc | The array is sorted with indexes compared as strings in ascending order. |
| @ind\_num\_asc | The array is sorted with indexes compared as numbers in ascending order. Non-numeric indexes are treated as zero. |
| @val\_type\_asc | The array is sorted based on values as per its type in ascending order. All numbers come before the strings. The sub-arrays come after the strings. |
| @val\_str\_asc | The array is sorted based on values of elements, treating the values as strings, in ascending order. |
| @val\_num\_asc | The array is sorted based on values of elements, treating values as numbers, in ascending order. |
| @ind\_str\_desc | The array is sorted based on index, treated as strings, in descending order. |
| @ind\_num\_desc | The array is sorted based on index, treated as numbers, in descending order. |
| @val\_type\_desc | The array is sorted based on the value of the element as per its type in descending order. Subarrays come first, then the strings and lastly, the numbers. |
| @val\_str\_desc | The array is sorted based on element values, treated as strings, in descending order. |
| @val\_num\_desc | The array is sorted based on values, treated as numbers, in descending order. |

一言不合举栗子：

# 默认方式，即无序

awk '

BEGIN {PROCINFO ["sorted\_in"] = "@unsorted"

    fruit ["apple"] = 4

    fruit ["mango"] = 12

    fruit ["guava"] = 8

    fruit ["banana"] = 16

    for (j in fruit)

        printf ("%s: %d numbers\n", j, fruit [j])

} '

guava: 8 numbers

mango: 12 numbers

apple: 4 numbers

banana: 16 numbers

复制代码

# 按照value的大小升序

awk '

BEGIN {PROCINFO ["sorted\_in"] = "@val\_num\_asc"

    fruit ["apple"] = 4

    fruit ["mango"] = 12

    fruit ["guava"] = 8

    fruit ["banana"] = 16

    for (j in fruit)

        printf ("%s: %d numbers\n", j, fruit [j])

} '

apple: 4 numbers

guava: 8 numbers

mango: 12 numbers

banana: 16 numbers

复制代码

# 按照index字母顺序降序

awk '

BEGIN {PROCINFO ["sorted\_in"] = "@ind\_str\_desc"

    fruit ["apple"] = 4

    fruit ["mango"] = 12

    fruit ["guava"] = 8

    fruit ["banana"] = 16

    for (j in fruit)

        printf ("%s: %d numbers\n", j, fruit [j])

} '

mango: 12 numbers

guava: 8 numbers

banana: 16 numbers

apple: 4 numbers

复制代码

俗话说，”栗子不过三“，就举到这里先。  
是不是觉得asort/asorti在这个sorted\_in”控制阀“面前弱爆了？！   
友情提示： 因为PROCINFO ["sorted\_in"]是全局性的变量，一旦设定之后，会改变整个awk的数组遍历方式，所以如果你希望在小范围内使用，可以按照下面的方式来做。

…

if ("sorted\_in" in PROCINFO) {

    save\_sorted = PROCINFO["sorted\_in"]

    PROCINFO["sorted\_in"] = "@val\_str\_desc" # or whatever

}

…

if (save\_sorted)

    PROCINFO["sorted\_in"] = save\_sorted

复制代码

事实上，除了awk内置的遍历函数，sorted\_in也可以被赋予自定义的函数。  
自定义的函数有个通用的代码框架如下：

function comp\_func(i1, v1, i2, v2) # 至少包含4个参数

{

    compare elements 1 and 2 in some fashion

    return < 0; 0; or > 0

}

复制代码

栗子如下：

awk '

BEGIN{

    arr[1] = 10

    arr[2] = 2

    arr[3] = 100

        arr["one"] = 10

        arr["two"] = 1

        arr["three"] = 100

    PROCINFO["sorted\_in"] = "cmp\_num\_val\_desc"

    print "#exactly the same as @val\_num\_desc"

    for(i in  arr)

        print "arr["i"] = " arr[i]

    print "如果排序规则改为：1. index：字母在前，数字之后    2. index一致时， value降序"

    PROCINFO["sorted\_in"] = "cmp\_smart\_desc"

    print "#sort in a smarter way"

        for(i in  arr)

        print "arr["i"] = " arr[i]

}

function cmp\_num\_val\_desc(i1, v1, i2, v2)

{

    # numerical value comparison, descending order,

    return (v2 - v1)

}

function cmp\_smart\_desc(i1, v1, i2, v2,   n1, n2)

{

     # numbers after string value comparison, descending order

     n1 = i1 + 0

     n2 = i2 + 0

     if (n1 != i1)

         return (n2 != i2) ? (v2 - v1) : -1

     else if (n2 != i2)

         return 1

     return v2 - v1

}

'

#exactly the same as @val\_num\_desc

arr[three] = 100

arr[3] = 100

arr[one] = 10

arr[1] = 10

arr[2] = 2

arr[two] = 1

如果排序规则改为：1. index：字母在前，数字之后    2. index一致时， value降序

#sort in a smarter way

arr[three] = 100

arr[one] = 10

arr[two] = 1

arr[3] = 100

arr[1] = 10

arr[2] = 2

复制代码

2. 数组的数组 （Arrays of Arrays)  
  
有了它，awk就可以真正创建多维数组，而不像以前版本那样用一维数组来模拟多维。  
如果有童鞋对perl的hash熟悉的话，那么它可以理解为hash of hash。  
  
下面先看“数组的数组”活生生的样子吧

a[1][1]=1

a[1][2]=2

a[1][3]=3

复制代码

是不是很眼熟，在某种/些语言里有相同的写法。  
没错，这就是一个典型的二维数组，第一维的index为[1]，第二维为[1][2][3]。  
事实上，为了保持每一维度在index使用的灵活性，对于下面的写法也是继续支持的：

a[1][1,"a"]=1

a[1][2,"a"]=2

a[1][3,"a"]=3

复制代码

并且，每一维数组的value可以是一个scalar，也可以是一个subarray

a[1][1,"a"]=1

a[2]=2

a[3][3][4]=3

复制代码

好了，说了这么多，如何打印Arrays of Arrays呢？其实很简单～

for (i in array)

    for (j in array[i])

        print array[i][j]

复制代码

当你不知道某个维度的value是scalar，还是subarray，那么可以加个判断。  
如何判断呢？也很简单，因为新版gawk已经帮你写好函数，就等你用了，它就是isarray。  
官方文档还配备了一个残暴的walk\_array, 简直是无所不至。

function walk\_array(arr, name,      i)

{

    for (i in arr) {

        if (isarray(arr[i]))

            walk\_array(arr[i], (name "[" i "]"))

        else

            printf("%s[%s] = %s\n", name, i, arr[i])

    }

}

复制代码

好久没有写文档了，一口气写了这么多，感觉身体快被掏空 不多说了，再打一套以上两个新功能的“组合拳”就结贴了！   
模拟sort排序

cat file

abc 123 100

abc 456 100

abc 456 10

def 123 10

def 123 100

abc 123 1

xzy 789 0

# sort 排序: 第一列按照字母升序，第二列数字升序，第三列数字降序

sort -k1,1 -k2,2n -k3,3nr file

abc 123 100

abc 123 1

abc 456 100

abc 456 10

def 123 100

def 123 10

xzy 789 0

# awk 3.0+ 排序

awk '

{

    a[$1" "$2" "$3];

        b[$1]=$1;

        c[$2];

        d[$3]

}

END{

        for(i=1;i<=asort(b);i++)

                for(j=1;j<=asorti(c,e);j++)

                        for(k=asorti(d,f);k>=1;k--)

                                if(b[i]" "e[j]" "f[k] in a)

                                        print b[i],e[j],f[k]

}

' file

abc 123 100

abc 123 1

abc 456 100

abc 456 10

def 123 100

def 123 10

xzy 789 0

# gawk 4.0+ 排序

awk '

{

    arr[$1][$2][$3]

}

END{

    PROCINFO["sorted\_in"] = "@ind\_str\_asc"

    for(i in arr){

        PROCINFO["sorted\_in"] = "@ind\_num\_asc"

        for(j in arr[i]){

            PROCINFO["sorted\_in"] = "@ind\_num\_desc"

            for(k in arr[i][j])

                print i,j,k

        }

    }

}

' file

abc 123 100

abc 123 1

abc 456 100

abc 456 10

def 123 100

def 123 10

xzy 789 0

复制代码

两种awk的写法相比，gawk的是不是更加清晰，明了呢   
  
  
艾玛呀，终于写完了，希望能给大家一些启示和帮助，抛砖引玉，如有错误的地方，请不吝指正！