目 录

1 gawk 程序的介绍 3
1.1 gawk 程序的基本格式 . . . 3
1.2 gawk 程序的数据字段变量 . . . 3
1.3 在程序脚本中使用多个命令 . . . 4
1.4 在处理数据之前运行脚本 . . . 4
1.5 在处理数据之后运行脚本 . . . 4

2 使用变量 5
2.1 内置变量 . . . 5
2.1.1 字段和记录分隔变量 . . . 5
2.1.2 数据变量 . . . 5
2.2 用户定义的变量 . . . 6

3 使用数组 7
3.1 定义数组变量 . . . 7
3.2 在数组变量中遍历 . . . 7
3.3 删除数组变量 . . . 7

4 使用模式 8
4.1 正则表达式 . . . 8
4.2 数学表达式 . . . 8

5 结构化命令 9
5.1 if 语句 . . . 9
5.2 while 语句 . . . 9
5.3 do-while 语句 . . . 10
5.4 for 语句 . . . 10

6 格式化打印 10

7 内置函数 11
7.1 数学函数 . . . 11

7.2 字符串函数 11
7.3 时间函数 12

8 用户定义的函数 12
8.1 定义函数 12
8.2 创建函数库 13

1 gawk 程序的介绍

1.1 gawk 程序的基本格式

gawk 编辑器是 linux 环境下常用的命令行编辑器，和 sed 编辑器一样是流编辑器。gawk 程序的基本格式如下所示：

```
gawk [options] <program> <file>
```

gawk 程序可用的选项如下所示：

-F fs	指定描述一行中数据字段分隔符
-f file	指定读取程序的文件名
-v var=value	定义 gawk 程序中使用的变量和默认值
-mf N	指定数据文件中要处理的字段的最大数目
-mr N	指定数据文件中的最大记录大小
-W keyword	指定 gawk 的兼容模式或警告级别

gawk 程序脚本由左大括号和右大括号定义，因为 gawk 命令行假定脚本是单文本字符串，所以必须将脚本包括在单引号内。例子如下：

```
gawk '{print "Hello John!"}' file
```

1.2 gawk 程序的数据字段变量

gawk 可以将每行中的每个数据元素分配给数据字段变量，其中 $ 表示整行文本，$1 表示文本行中的第一个数据字段，$2 表示文本行中的第二个数据字段，$n 表示文本行中的第 n 个数据字段。

gawk 读取一行文本时，使用定义的字段分隔符描述各数据字段。gawk 默认的字段分隔符为任意空白字符。

使用数据字段变量的例子如下：

```
gawk '{print $1}' data
```

可以使用“-F fs”参数项来制定字段分隔符，如下所示：

```
gawk -F : '{print $1}' /etc/passwd
```

1.3 在程序脚本中使用多个命令

在程序脚本中可以使用多条命令，这些命令用分号“;”隔开，使用例子如下所示：

```
1  echo "My name is Rich" | gawk '{$4="Dave"; print $0}'
```

1.4 在处理数据之前运行脚本

通过 BEGIN 关键字，可以让 gawk 在读取数据之前运行脚本，使用例子如下所示：

```
1  gawk 'BEGIN {print "Hello World!"}' file
```

如果既想要在读取数据之前运行脚本，又想要在读取数据时候运行脚本处理数据，可以使用两个大括号，使用例子如下所示：

```
1  gawk 'BEGIN {print "Hello World!"} {print $0}' file
```

1.5 在处理数据之后运行脚本

通过 END 关键字，可以让 gawk 在读取数据和处理数据结束之后运行脚本，使用例子如下所示：

```
1  gawk 'BEGIN {print "Hello World!"} {print $0} END{print "byebye"}'
```

可以将这些技术写成一个脚本文件，例子如下：

```
1  BEGIN
2  {
3      print "The latest list of users and shells"
4      print " Userid       Shell"
5      print " ------       -------"
6  }
7
8  {
9      print $1 "       " $7
10 }
11
12 END
13 {
14     print "This concludes the listing"
15 }
```

2 使用变量

2.1 内置变量

2.1.1 字段和记录分隔变量

gawk 的字段和记录分隔变量如下：

FILEDWIDTHS	每个数据字段的宽度
FS	输入字段分隔符号
RS	输入记录分隔符号
OFS	输出字段分隔符号
ORS	输出记录分隔符号

OFS 的使用例子如下所示：

```
gawk 'BEGIN{FS=","; OFS="-"} {print $1,,$2,$3}' data1
```

FILEWIDTHS 的使用例子如下所示：

```
# 此方法不支持长度为变量的数据字段
gawk 'BEGIN{FILEDWIDTHS="3 5 2 5"}{print $1,$2,$3,$4}' data1
```

默认情况下 RS 的值为换行符，所以 gawk 一次读入一行。如果将 RS 设为空值，并将数据以空行分隔，那么 gawk 就可以一次读入一段文本。例子如下所示：

```
gawk 'BEGIN{FS="\n"; RS=""}{print $1,$4}' data
```

2.1.2 数据变量

gawk 的数据变量如下所示：

ARGC	出现的命令行参数的个数
ARGIND	当前正在处理的文件在 ARGV 中的索引
ARGV	命令行参数数组
CONVFMT	数字的转换格式
ENVIRON	当前 shell 环境变量及其值的关联数组
ERRNO	当读取或关闭输入文件时发生错误时的系统错误

FILENAME	用于输入到 gawk 程序的数据文件的文件名
FNR	数据文件的当前记录号
IRNORECASE	如果设置为非 0，则忽略大小写
NF	数据文件中数据字段的个数
NR	已处理的输入记录的个数
OFMT	显示数字的输出格式
RLENGTH	匹配函数中匹配上的子字符串的长度
RSTART	匹配函数中匹配上的子字符串的开始索引

2.2 用户定义的变量

可以在脚本中定义变量，如下例所示：

```
gawk '
BEGIN{
testing="This is a test"
print testing
testing=45
print testing
}'
```

也可以在命令行中赋值变量，如下例所示：

```
# script1的内容
BEGIN{FS=","}
{print $n}
# 命令行的内容
gawk -f script1 n=2 data1
```

需要注意的是，在设置变量时，值不能在代码的 BEGIN 部分使用。如果要在代码的 BEGIN 部分之前设置变量，需要带上“-v”参数项，如下所示：

```
# script2的内容
BEGIN{print "The starting value is", n; FS=","}
{print $n}
# 命令行的内容
gawk -v n=3 -f script2 data1
```

3 使用数组

3.1 定义数组变量

数组变量的赋值格式如下：

```
var[index] = element
```

gawk 程序的数组变量和 python 中的字典类似，如下例所示：

```
gawk 'BEGIN{
capital["Illinois"] = "Springfield"
print capital["Illinois"]
}'
```

3.2 在数组变量中遍历

如果要在 gawk 中遍历数组，可以使用 for 语句，格式如下：

```
for(var in array)
{
    statements
}
```

for 语句在各语句中循环，每次向变量 var 分配 array 关联数组中的下一个索引值，如下例所示：

```
gawk 'BEGIN{
var["a"] = 1
var["g"] = 2
var["m"] = 3
var["u"] = 4
for (test in var)
{
    print "Index:", test, " — Value:", var[test]
}
}'
```

需要注意的是，索引值不是以一定的顺序返回的。

3.3 删除数组变量

从关联数组删除数组索引需要一个特殊的命令，如下所示：

```
delete array[index]
```

具体操作如下例所示：

```
gawk 'BEGIN{
var["a"] = 1
var["g"] = 2
for(test in var)
{
    print "Index:",test," - Value:",var[test]
}
delete var["g"]
print "------"
for(test in var)
{
    print "Index:",test," - Value:",var[test]
}
}'
```

4 使用模式

4.1 正则表达式

在使用正则表达式时，正则表达式必须出现在程序脚本的左括号之前，如下例所示：

```
gawk 'BEGIN{FS=","} /11/{print $1}' data1
```

如果想让正则表达式匹配特定的数据字段，需要使用匹配操作符。匹配操作符需要制定数据字段变量和正则表达式，格式如下所示：

```
$n ~ /^data/
```

使用例子如下所示：

```
gawk 'BEGIN{FS=","} $2 ~ /^data2/{print $0}' data1
```

还可以通过! 符号来否定正则表达式的匹配，从而将命令作用于没有匹配到的数据，如下例所示：

```
gawk 'BEGIN{FS=","} $2 ! ~ /^data2/{print $1}' data1
```

4.2 数学表达式

gawk 程序中的数学表达式如下所示：

x==y	值 x 等于 y
x<=y	值 x 小等于 y
x<y	值 x 小于 y

x>=y	值 x 大等于 y
x>y	值 x 大于 y

具体操作如下例所示：

```
gawk -F : '$1 == "root"{print $2}' /etc/passwd
```

5 结构化命令

5.1 if 语句

gawk 的 if 语句格式如下：

```
if(condition)
{
    statements
}
```

如果只有一条命令，可以不要大括号，如下所示：

```
# 格式1
if(condition)
    statement1
# 格式2
if(condition) statement1
```

gawk 的 if 语句还支持 else 分句，格式如下：

```
if(condition)
{
    statements
}
else
{
    other statements
}
```

还可以在单个行上使用 else 分句，格式如下所示：

```
if(condition) statement1; else statement2
```

5.2 while 语句

while 语句的格式如下所示：

```
while(condition)
{
    statements
}
```

5.3 do-while 语句

do-while 语句的格式如下所示：

```
do
{
    statements
}while(condition)
```

5.4 for 语句

gawk 支持 C 语言形式的 for 循环，格式如下：

```
for(variable assignment; condition; iteration process)
{
    statements
}
```

6 格式化打印

gawk 的格式化打印命令是 printf 命令，与 C 语言的用法一样，格式如下：

```
print "format string", var1, var2
```

格式化说明符的格式如下：

```
%[modifier]control-letter
```

modifier 是可选的格式化功能，如下所示：

width	指定输出字段最小宽度的数字值，格式为“n”
prec	指定浮点数中小数点右侧位数的数值，格式为“.n”
-	将数据左对齐

control-letter 是格式化说明符中使用的控制字，如下所示：

c	将数据显示为 ASCII 字符
d	显示整数值
i	显示整数值
e	用科学计数法显示数字
f	显示浮点数值
g	以科学计数法和浮点数的较短者显示数字
o	显示八进制数值
s	显示文本字符串
x	显示十六进制数值，对 a 到 f 使用小写字母
X	显示十六进制数值，对 A 到 F 使用大写字母

7 内置函数

7.1 数学函数

gawk 内置的数学函数如下图所示：

函数	描述	函数	描述
atan2(x, y)	x/y 的反正切，x 和 y 以弧度表示	rand()	大于 0 小于 1 的随机浮点值
cos(x)	x 的余弦，x 以弧度表示	sin(x)	x 的正弦，x 以弧度表示
exp(x)	x 的指数	sqrt(x)	x 的平方根
int(x)	x 的整数部分，截止到 0	srand(x)	用指定值计算的随机数
log(x)	x 的自然对数		

7.2 字符串函数

gawk 内置的字符串函数如下图所示：

函数	描述
sprintf(*format*,*variables*)	返回一个字符串，该字符串与使用指定的 *format* 和 *variables* 的 printf 的输出类似
sub(*r*, *s* [,*t*])	搜索变量 *$0* 或目标字符串 *t*，以匹配正则表达式 *r*。如果找到，则将第一个找到的目标替换为字符串 *s*
substr(*s*, *i* [,*n*])	返回子字符串 *s* 的从索引 *i* 开始的第 *n* 个字符。如果未指定 n，则使用其余的 *s*
tolower(*s*)	将 *s* 中的所有字符转换为小写
toupper(*s*)	将 *s* 中的所有字符转换为大写

函数	描述
asort(*s* [,*d*])	基于数据元素值对数组 *s* 进行排序。索引值被替换为表示新排序顺序的一系列数字。另外，可以指定将新排序后的数组存放在 d 数组中
asorti(*s* [,*d*])	基于索引值对数组 *s* 进行排序。产生的数组将索引值包含为数据元素值，并带有表示排序顺序的一系列数字。另外，可以指定将新排序后的数组存放在 *d* 数组中
gensub(*r*, *s*, *h* [, *t*])	搜索变量 *$0*，或目标字符串 t(如果指定)以匹配正则表达式 *r*。如果 *h* 是以 *g* 或 *G* 开头的字符串，则以 *s* 替换匹配的文本。如果 *h* 是一个数字，则表示用 *r* 进行替换的次数
gsub(*r*, *s* [,*t*])	搜索变量 *$0*，或目标字符串 *t*(如果指定)以匹配正则表达式 *r*。如果找到，则全局替换字符串 *s*
index(*s*, *t*)	返回字符串 *s* 中字符串 *t* 的索引，如果找不到，则返回 *0*
length([*s*])	返回字符串 *s* 的长度，如果未指定，则返回 *$0* 的长度
match(*s*, *r* [,*a*])	在具有正则表达式 *r* 的地方返回字符串 *s* 的索引。如果指定数组 *a*，则包含 *s* 与正则表达式匹配的部分
split(*s*, *a* [,*r*])	使用 *FS* 字符或正则表达式(如果已指定)将 *s* 分为数组 *a*。返回字段的个数

7.3 时间函数

gawk 内置的时间函数如下图所示：

函数	描述
mktime(*datespec*)	将以 YYYY MM DD HH MM SS 格式显示的日期[DST]，转换为时间戳值
strftime(format[, *timestamp*])	将日时间戳的当前时间，或时间戳（如果提供），使用 date() shell 函数格式转换为格式化的日和日期
systime()	返回当前日时间的时间戳

8 用户定义的函数

8.1 定义函数

使用 function 关键字定义自己的函数，格式如下：

```
function name([variable])
{
    statements
}
```

gawk 的函数与 C 语言的函数类似，可以直接传递参数，还可以用 return 语句返回值。具体操作的例子如下所示：

```
gawk '
function myprint()
{
    print "%-16s - %s\n", $1, $4
}

BEGIN
{
    FS="\n"
    RS=""
```

```
    }

    {
        myprint()
    }' data1
```

8.2 创建函数库

gawk 可以创建函数库，然后用“-f”参数项引用函数库文件。具体操作如下例所示：

```
    # 函数库文件名是funclib
    function myprint()
    {
        print "%-16s - %s\n", $1, $4
    }

    function myrand(limit)
    {
        return int(limit * rand())
    }

    function printthird()
    {
        print $3
    }

    # gawk脚本文件民是script4
    BEGIN
    {
        FS="\n"
        RS=""
    }

    {
        myprint()
    }
```

然后在命令行上用“-f”参数项制定库文件和程序文件，如下所示：

```
    gawk -f funclib -f script4 data1
```

需要注意的是，使用“-f”命令行参数时不能使用内嵌的 gawk 脚本，但是可以使用多个“-f”参数。