awk的模式和动作

**【语法结构】**

awk -F 'Pattern1{Action1};Pattern2{Action2}' input-file或

awk -F '{Action}' input-file

1. -F 为字段分隔符。若不指定，默认会使用空格作为分隔符。
2. /Pattern/和{Action}需要用单引号引起来，防止shell作解释。
3. /Pattern/是可选的。如果不指定，awk 将处理输入文件中的所有记录。如果指定一个模式，awk 则只处理匹配指定的模式的记录。
4. {Action} 为 awk 命令，可以是单个命令，也可以多个命令。整个 Action(包括里面的所有命令)都必须放在{ 和 }之间。
5. Input-file 要处理的目标文件
6. Action必须被｛｝包裹，没有被｛｝包裹的就是Pattern

**【Pattern】**

awk 可接受许多不同型态的 Pattern. 一般常使用 "关系表达式"(Relational expression) 来当成 Pattern。awk 通过判断 Pattern 的值来决定是否执行其后所对应的Actions。以下是几种常见的Pattern：

**BEGIN**

BEGIN 为 awk 的保留字, 是一种特殊的 Pattern.

BEGIN 成立(值为true)的时候是: "awk 程序一开始执行, 尚未读取任何数据之前." 所以在 BEGIN { Actions } 语法中, 其 Actions 部份仅于程序一开始执行时被执行一次. 当 awk 从数据文件读入数据行后, BEGIN 便不再成立, 故不论有多少数据行, 该 Actions 部份仅被执行一次.一般常把 "与数据文件内容无关" 与 "只需执行ㄧ次" 的部分置于该Actions(以 BEGIN 为 Pattern)中.

例如:

BEGIN {

FS = "[ \t:]" # 于程序一开始时, 改变awk切割字段的方式

RS = "" # 于程序一开始时, 改变awk分隔数据行的方式

count = 100 # 设定变量 count 的起始值

print " This is a title line " # 印出一行 title

}

....... # 其它 Pattern { Actions } .....

有些awk程序甚至"不需要读入任何数据行". 遇到这情况可把整个程序置于以 BEGIN 为 Pattern的 Actions 中.

例如 :

BEGIN { print " Hello ! the Word ! " }

注意 :执行该类仅含 BEGIN { Actions } 的程序时, awk 并不会开启任何数据文件进行处理.

**END**

END 为 awk 的保留字, 是另一种特殊的 Pattern.

END 成立(其值为true)的时机与 BEGIN 恰好相反, 为:"awk 处理完所有数据, 即将离开程序时"平常读入数据行时, END并不成立, 故其对应的 Actions 并不被执行; 唯有当awk读完所有数据时, 该 Actions 才会被执行

注意 : 不管数据行有多少, 该 Actions 仅被执行一次.

**关系表达式**

使用像 " A 关系运算符 B" 的表达式当成 Pattern.

当 A 与 B 存在所指定的关系(Relation)时, 该 Pattern 就算成立(true).

例如 :

length($0) <= 80 { print $0 }

上式中 length($0)<= 80 是一个 Pattern, 当 $0(数据行)之长度小于等于80时该 Pattern 之值为true, 将执行其后的 Action (打印该数据行).

awk 中提供下列 关系运算符(Relation Operator)

运算符 含意

> 大于

< 小于

>= 大于或等于

<= 小于或等于

== 等于

!= 不等于

~ match

!~ not match

上列关系运算符除~(match)与!~(not match)外与 C 语言中之含意一致.

~(match) 与!~(match) 在 awk 之含意简述如下 :

若 A 为一字符串, B 为一正则表达式.

A ~B 判断 字符串A 中是否 包含 能匹配(match)B式样的子字符串.

A !~B 判断 字符串A 中是否 未包含 能匹配(match)B式样的子字符串.

例如 :

$0 ~ /program[0-9]+\.c/ { print $0 }

$0 ~ /program[0-9]+\.c/ 整个是一个 Pattern, 用来判断$0(数据行)中是否含有可 match /program[0-9]+\.c/ 的子字符串, 若$0 中含有该类字符串, 则执行 print (打印该行数据).

Pattern 中被用来比对的字符串为$0 时(如本例), 可仅以正则表达式部分表示整个Pattern.故本例的 Pattern 部分$0 ~/program[0-9]+\.c/ 可仅用/program[0-9]+\.c/表之

**常量**

常量有两种类型,字符串常量和数值常量.字符串常量必须用引号括起来,数值常量不需要,如数值0和字符串"0"是不一样的.

数值常量中,0代表假,非0的数值常量代表真

字符串常量中,空字符串为假,非空字符串为真

例如

1{print}

上式利用数值常量1作为一个逻辑值.

该式中数值常量为1非0为真

**空Pattern**

如果awk的语句中省略了pattern，即pattern为空，则代表永远为真，并执行后面紧跟的action.如果action省略则默认action为print

例如

{name=1}5

上式还原完整的格式为 空pattern {name=1}5 {print}.

**正则表达式**

直接使用正则表达式当成 Pattern; 此为 $0 ~ 正则表达式 的简写.

该 Pattern 用以判断 $0(数据行) 中是否含有匹配该正则表达式的子字符串; 若含有则成立(true) 执行其对应的 Actions.

例如 :

/^[0-9]\*$/ { print "This line is a integer !" }

与 $0 ~/^[0-9]\*$/ { print "This line is a integer !" } 相同

**混合 Pattern**

之前所介绍的各种 Patterns, 其计算后结果为一逻辑值(True or False).awk 中逻辑值彼此间可通过&&(and), ||(or),  !(not) 结合成一个新的逻辑值.故不同 Patterns 彼此可通过上述结合符号来结合成一个新的 Pattern. 如此可进行复杂的条件判断.

例 如 :

FNR >= 23 && FNR <= 28 { print "     " $0 }

上式利用&& (and) 将两个 Pattern 求值的结果合并成一个逻辑值.

该式将数据文件中 第23行 到 28行 向右移5格(先输出5个空白字符)后输出.

/2/0{print}

上式利用正则表达式和数值常量0结合的结果合并为一个逻辑值.

该式中非空字符串永远为真，不管正则结果是真还是假，结果永远为真，打印文件所有行，等同于1{print}简写为{print}

**Pattern1 , Pattern2**

遇到这种 Pattern, awk 会设定一个区间的开关，即switch(或flag).

当awk读入的数据行使得 Pattern1 成立时, awk 会打开(turn on)这 switch.

当awk读入的数据行使得 Pattern2 成立时, awk 会关上(turn off)这个 switch.

该 Pattern 成立的条件是 :

当这个 switch 被打开(turn on)时 (包括 Pattern1, 或 Pattern2 为真的情况)

例 如 :

FNR >= 23 && FNR <= 28 { print "     " $0 }

可改写为

FNR == 23 , FNR == 28 { print "     " $0 }

说 明 :

当 FNR >= 23 时, awk 就 turn on 这个 switch; 因为随着数据行的读入, awk不停的累加 FNR. 当 FNR = 28 时, Pattern2 (FNR == 28) 便成立, 这时 awk 会关闭 switch.

当 switch 打开的期间, awk 会执行  print "     " $0

例如:

'/2/,/2/

其实等价于

/2/

说明:

当当前读入的行即$0中匹配到2这个字符，awk就turn on这个switch;从当前读入的行开始继续向下读，直到读入的行中匹配到2这个字符，awk就turn off这个switch.

例如:

'/1/,/2/

说明:

当当前读入的行匹配到1这个字符,awk就turn on这个switch,从当前读入的行开始继续向下读,直到读入的行匹配到2这个字符,awk就turn off这个switch.

**【Actions】**

Actions 是由下列指令(statement)所组成 :

* 表达式 ( function calls, assignments..)
* print 表达式列表
* printf( 格式化字符串, 表达式列表)
* if( 表达式 ) 语句 [else 语句]
* while( 表达式 ) 语句
* do 语句 while( 表达式)
* for( 表达式; 表达式; 表达式) 语句
* for( variable in array) 语句
* delete
* break
* continue
* next
* exit [表达式]
* 语句

awk 中大部分指令与 C 语言中的用法一致, 此处仅介绍较为常用或容易混淆的指令的用法.