

Linux 操作系统及应用

附录 A — AWK

李亦农 唐晓晟

hoplee@bupt.edu.cn txs@bupt.edu.cn

Contents

| | |
|----------|---|
| 1 概述 | 1 |
| 2 变量 | 2 |
| 3 模式 | 3 |
| 4 动作 | 4 |
| 5 I/O 语句 | 5 |
| 6 函数 | 5 |
| 7 其他 | 6 |
| 8 限制 | 6 |
| 9 实例 | 7 |

1 概述

- **awk** 是一种编程语言，她是由 AT&T 贝尔实验室的 Alfred Aho, Peter Weinberger 和 Brian W. Kernighan 开发的，Brian W. Kernighan 目前仍在维护及增强 **awk**。
- **awk** 能非常方便地处理与数据加工和信息检索相关的任务。
- **awk** 还可以和 **shell** 程序相互结合，从而增强自己的功能。
- 一个 **awk** 程序是由一系列的“模式—动作”语句构成的：

```
1 pattern {action}  
2 pattern {action}  
3 pattern {action}  
4 .....
```

- **awk** 程序为每个输入行依次地进行每一个模式的匹配寻找，对每一个匹配上的模式执行相应的动作，接着读取下一行并再次开始匹配，直到所有的输入都处理完毕。
- 在一条语句中可以省略模式或者动作，缺省的模式为匹配所有行，缺省的动作作为输出当前行：**print \$0**

- 正是因为模式和动作可有可无，动作才要被花括号括起来以便与模式相区别。
- `awk`的程序可以以两种方法来运行：
 1. 直接命令行方式：


```
awk 'pattern-action statements' input_file_list
```

 其中“模式—动作”语句必须括在单引号中。
 2. `awk`命令程序方式：
 通常将`awk`程序放在一个单独的文本文件中，然后用`-f`选项来引用：


```
awk -f awk_file input_file_list
```
- `awk`从输入中一次读取一行（一条记录），缺省的`RS`为`\n`。
- 然后`awk`将记录分割为一个个的字段，缺省的`FS`为`Blank`。一行中的第一个字段称为`$1`，第二个字段称为`$2`，...，整个记录称为`$0`。
- `awk`提供了`print`和`printf`语句用于显示输出。
 - `print`为无格式输出语句：


```
print expr1,expr2,...,exprN
```

`print`语句显示每个表达式的串值，默认的`ORS`和`OFS`分别为`\n`和`Blank`。
 - `printf`为格式化输出语句：


```
printf format,expr1,expr2,...,exprN
```

`format`含有要显示的信息以及要转换的规格说明，其语法与C语言中的输出函数中的格式说明类似。
 - 常用的转换字符如下表所示：

| | |
|-----------------|--|
| <code>%c</code> | 单个字符 |
| <code>%d</code> | 十进制数 |
| <code>%e</code> | <code>[-]d.ddddddE[+ -]dd</code> |
| <code>%f</code> | <code>[-]d.dddddd</code> |
| <code>%g</code> | <code>e</code> 或 <code>f</code> 中较短的，并去掉无用的0 |
| <code>%o</code> | 无符号八进制数 |
| <code>%s</code> | 串 |
| <code>%x</code> | 无符号十六进制数 |
| <code>%%</code> | 显示一个% |

- 使用`printf`时，不会自动输出`ORS`，必须自己在`format`中使用`\n`来显式地产生。

2 变量

- `awk`的变量分为内部变量和用户定义的变量两种。
- `awk`的内部变量有：

| | |
|-----------------------|-----------|
| <code>ARGC</code> | 命令行参数的个数 |
| <code>ARGCIND</code> | 当前命令行参数下标 |
| <code>ARGV</code> | 命令行参数数组 |
| <code>ENVIRON</code> | 环境变量数组 |
| <code>FILENAME</code> | 当前输入文件名 |

| | |
|------------|------------------------------------|
| FNR | 当前文件中的记录号 |
| FS | 字段分隔符 |
| IGNORECASE | 忽略正则表达式和串的大小写 |
| NF | 当前记录中的字段数 |
| NR | 至今读取的记录数 |
| OFMT | 数的输出格式，缺省为%.6g |
| OFS | 输出字段分隔符 |
| ORS | 输出记录分隔符 |
| RS | 输入记录分隔符 |
| RSTART | 由 <code>match()</code> 匹配的第一个字符的索引 |
| RLENGTH | 由 <code>match()</code> 匹配的串的长度 |
| SUBSEP | 下标分隔符，缺省为\034 |

- 当前记录的字段可以用`$1`, `$2`, ..., `$NF`来表示。
- 用户定义的变量类似于 Shell 中的情形。

3 模式

- 模式是一种表达式。
- **BEGIN**和**END**是两个特殊的模式，**BEGIN**在第一条记录被读取之前匹配，**END**在最后一条记录处理完之后匹配。
- 关系表达式：**awk**有 6 个关系运算符和 2 个正则表达式匹配运算符：

| | |
|----|------|
| < | 小于 |
| <= | 小于等于 |
| == | 等于 |
| != | 不等于 |
| >= | 大于等于 |
| > | 大于 |
| ~ | 匹配 |
| !~ | 不匹配 |

- 在比较表达式中，若两个操作数都是数值，则进行数值比较，否则进行串比较。
- 类型强制转换：`num`, `"", string+0`
- 正则表达式：**awk**把在“~”和“!~”右边的任一串或变量都解释为一个正则表达式。
- 当用引号括起来的字面字符串用作一个正则表达式时，如果字符串中含有元字符，需要再加一层反斜线，以保护正则表达式中的元字符。
- **awk**里引入了一个新的概念：**字符类**。这个概念来自于 **POSIX** 标准。
- 字符类是一种特殊的表达式，用于描述具有某种特定属性的字符集合。她具体能表示什么样的字符集合是和应用程序的地区特性有关的。
- 字符类只能在正则表达式中的一对方括号中出现，用于表示特定的字符集合。
- 常用的字符类有：

| | |
|-------------------------|------------------|
| <code>[:alnum:]</code> | 字母或数字字符 |
| <code>[:alpha:]</code> | 字母字符 |
| <code>[:blank:]</code> | 空格或制表符 |
| <code>[:cntrl:]</code> | 控制字符 |
| <code>[:digit:]</code> | 数字字符 |
| <code>[:graph:]</code> | 既能看见又能打印的字符 |
| <code>[:lower:]</code> | 小写字母字符 |
| <code>[:print:]</code> | 可打印字符（非控制字符） |
| <code>[:punct:]</code> | 标点符号字符 |
| <code>[:space:]</code> | 空白字符，包括空格、制表、换页等 |
| <code>[:upper:]</code> | 大写字母 |
| <code>[:xdigit:]</code> | 十六进制数字字符 |

- 模式组合：用圆括号和逻辑运算符`||`、`&&`和`!`可以把简单的模式组合成复合模式。优先级从左到右依次增高。
- 模式范围由逗号分隔的两个模式组成：
`pattern1,pattern2 {action}`
表示对于在`pattern1`和`pattern2`出现之间的每一条记录都要执行动作。包括`pattern1`和`pattern2`。

4 动作

- 动作决定对模式选中的记录进行什么操作。
- `awk`的运算包括算数运算和串运算。
- `awk`可以使用传统的算数表达式来计算数值，算数运算在内部以浮点形式完成。常用的算数运算符有：
`+, -, *, /, ++, --, %, ^, +=, -=, *=, /=, %=, ^=`和`=`
其含义和C语言里一样。
- `awk`还提供了一些内部算数函数：`atan1(y,x)`, `cos(x)`, `exp(x)`, `int(x)`, `log(x)`, `rand()`, `sin(x)`, `sqrt(x)`和`srand(x)`。其中`x,y`是任意表达式。`rand()`返回(0,1)范围内的随机数，`srand(x)`用于设置`rand()`的种子。
- `awk`提供的串运算符只有一个：`space`，进行串的串接。
- `awk`提供的串函数有：
其中，`r`代表一个正则表达式，`s`和`t`代表串表达式，`n`和`p`代表整数，`a`代表数组。

| | |
|---------------------------------|--|
| <code>gsub(r,s)</code> | 将当前记录中的 <code>r</code> 替换为 <code>s</code> ，全局，返回替换数 |
| <code>gsub(r,s,t)</code> | 在串 <code>t</code> 中全局用 <code>s</code> 替换 <code>r</code> ，返回替换数 |
| <code>index(s,t)</code> | 返回 <code>s</code> 中串 <code>t</code> 的位置，不出现时为0 |
| <code>length(s)</code> | 返回串 <code>s</code> 的长度 |
| <code>match(s,r)</code> | 返回 <code>r</code> 在 <code>s</code> 中出现的位置，不出现时为0 |
| <code>split(s,a)</code> | 利用FS把 <code>s</code> 分裂成数组 <code>a</code> ，返回字段数 |
| <code>split(s,a,r)</code> | 利用 <code>r</code> 把 <code>s</code> 分裂成数组 <code>a</code> ，返回字段数 |
| <code>sprintf(fmt,exprs)</code> | 根据格式串 <code>fmt</code> ，返回经过格式编排的 <code>expr_list</code> |
| <code>sub(r,s)</code> | 在当前记录中把第一个 <code>r</code> 替换成 <code>s</code> 之后的部分，返回替换的个数 |

| | |
|----------------------------|--------------------|
| <code>sub(r,s,t)</code> | 在t中把第一个r替换成s之后的部分 |
| <code>substr(s,p)</code> | 返回从位置p开始的s之后的部分 |
| <code>substr(s,p,n)</code> | 返回从位置p开始，长度为n的s的子串 |
| <code>tolower(s)</code> | 将串s中的大写字母改为小写 |
| <code>toupper(s)</code> | 将串s中的小写字母改为大写 |

- 流控：awk提供和C语言相似的流控语句：

```

1  if (condition) statement [ else statement ]
2  while (condition) statement
3  do statement while (condition)
4  for (expr1; expr2; expr3) statement
5  for (var in array) statement
6  break
7  continue
8  delete array[index]
9  delete array
10 exit [ expression ]
11 { statements }
```

5 I/O 语句

- awk的输入、输出语句如下所示：

| | |
|----------------------------------|---|
| <code>close(file)</code> | 关闭文件（或管道） |
| <code>getline</code> | 从当前输入、文件或管道中读取下一个输入记录，并进行通常的字段分裂处理，同时设置NR、FNR和FNR |
| <code>getline < file</code> | 从文件中getline |
| <code>getline var</code> | 读取下一条记录并将其赋给var，同时设置NR和FNR |
| <code>getline var<file</code> | 从文件中读取下一条记录并将其赋给var，同时设置NR和FNR |
| <code>next</code> | 立即从第一个模式开始处理下一条记录 |
| <code>nextfile</code> | 停止处理当前输入文件，立即处理下一输入文件。FILENAME和ARGIND被更新，FNR被置为1 |
| <code>system(cmd-line)</code> | 执行Shell命令cmd-line并返回命令的退出状态 |
| <code>fflush([file])</code> | 刷新输出文件的缓冲区 |

- 其他的I/O重定向也可在awk中使用。对于print和printf语句，>>file会将输出追加到file文件中，|会将输出送到管道中。同样的command | getline将会使得getline函数从管道中读取数据。
- 当遇到文件结束时，getline将返回0，任何错误将导致getline返回-1。

6 函数

- 用户可以自己定义函数，语法为：

```

1 function function_name(arg_list){
2     statements
3 }

```

- 数组参数可通过引用传递，标量参数将用值传递。
- 在函数内部的形式参数是局部变量，其他变量都是全局变量。
- `return`语句可有可无。
- 函数在调用时在函数名与实参表的左括号之间不得留有空格。因为空格是字符串串接运算符。

7 其他

- `awk`提供一维数组。数组和数组元素无须声明，通过使用它来表明它的存在。
- 数组下标可以是一个数或串：`arr[x]`和`arr["x"]`
- 可以使用`for`语句对数组所有定义的下标进行循环：`for (i in arr) statement`，此时下标是随机选取的。
- 可以用`delete`语句删除数组元素：
`delete array_name[subscripts]`
- 时间函数：

| | |
|--|---|
| <code>sysftime()</code> | 返回从 <i>Epoch</i> 到现在的秒数 |
| <code>strftime([format[, timestamp]])</code> | 返回用 <code>format</code> 格式化后的 <code>timestamp</code> 指定的时间（缺省为当前时间） |

- 任何以`#`开头的行都是注释。
- 几个语句可以用分号分隔以便出现在同一行上。
- 续行符为`\`。

8 限制

- 100 个字段
- 每个输入记录 2500 个字符
- 每个输出记录 2500 个字符
- 每个单个字段 1024 个字符
- 每个`printf`串 1024 个字符
- 括起来的串 400 个字符
- 字符类 400 个字符
- 15 个打开的文件

- 1 个管道
- 数值的大小由本地机器限制

9 实例

1. 此程序会显示所有输入行之中字段的最大个数。

```
1 awk '{if (NF > max) max = NF}
2 END {print max}'
```

2. 此程序会显示出超过 80 个字符的每一行。此处只有模式被列出，动作是采用缺省值显示整个记录。

```
1 awk 'length($0) > 80'
```

3. 显示拥有至少一个字段的的所有行。这是一个简单的方法，将一个文件里的所有空白行删除。

```
1 awk 'NF > 0'
```

4. 此程序会显示出范围是 0 到 100 之间的 7 个随机数。

```
1 awk 'BEGIN {for (i = 1; i <= 7; i++)
2 print int(101 * rand())}'
```

5. 此程序会显示出所有指定的文件的总字节数。

```
1 ls -l files | awk '{x += $5};\
2 END {print "total bytes: " x}'
```

6. 此程序会将指定文件里最长一行的长度显示出来。`expand`会将TAB改成SPACE，所以是用实际的右边界来做长度的比较。

```
1 expand file | awk '{if (x < length()) x = length()}'
2 END {print "maximum line length is " x}'
```

7. 将所有用户的登录名称，依照字母的顺序显示出来。

```
1 awk 'BEGIN {FS = ":"}
2 {print $1 | "sort"}' /etc/passwd
```

8. 此程序会将一个文件的总行数显示出来。

```
1 awk '{nlines++}
2 END {print nlines}'
```

9. 此程序也会将一个文件的总行数显示出来，但是计算行数的工作由`awk`来做。

```
1 awk 'END {print NR}'
```

10. 此程序显示出文件的内容时，会在每行的最前面显示出行号，它的功能与`cat -n`和`nl`命令类似。

```
1 awk '{print NR, $0}'
```

The End of Appendix A.