Documentos

ISSN 1677-9274
Dezembro, 2010
103

Tutorial do comando *Awk*

Embrapa

ISSN 1677-9274
Dezembro, 2010

Empresa Brasileira de Pesquisa Agropecuária
Embrapa Informática Agropecuária
Ministério da Agricultura, Pecuária e Abastecimento

Documentos 103

Tutorial do comando *Awk*

Fábio Danilo Vieira

Embrapa Informática Agropecuária
Campinas, SP
2010

Embrapa Informática Agropecuária
Av. André Tosello, 209 - Barão Geraldo
Caixa Postal 6041 - 13083-886 - Campinas, SP
Fone: (19) 3211-5700 - Fax: (19) 3211-5754
www.cnptia.embrapa.br
sac@cnptia.embrapa.br

1ª edição on-line 2010

Dados Internacionais de Catalogação na Publicação (CIP)
Embrapa Informática Agropecuária

Vieira, Fábio Danilo.
Tutorial do comando Awk / Fábio Danilo Vieira. - Campinas : Embrapa Informática Agropecuária, 2010..
37 p. : il. – (Documentos / Embrapa Informática Agropecuária ; ISSN 1677-9274, 103).

1. Comando awk . 2. Programação. 3. Scripts. 4. Shell. 5. Linux. I. Título. II. Série.

005.15 CDD (21. ed.)

Autor

Fábio Danilo Vieira
Analista da Embrapa Informática Agropecuária
Av. André Tosello, 209, Barão Geraldo
Caixa Postal 6041 - 13083-970 - Campinas, SP
Telefone: (19) 3211-5798
e-mail: fabiodv@cnptia.embrapa.br

Apresentação

Mesmo que o Windows ainda seja, disparado, o sistema operacional mais utilizado e conhecido pelas pessoas no mundo todo, o Linux vem se difundindo cada vez mais em diversos nichos da população, sejam profissionais de informática ou não.

Uma das grandes vantagens do Linux sobre seu concorrente é a diversidade de comandos que seu terminal *shell* (interface de linha de comando) possui, o que se tornou uma característica forte deste sistema operacional.

O *awk* é um desses comandos que faz o terminal *shell* do Linux ser a marca forte desse sistema. É um comando tão importante e com tantas funcionalidades que muitos o confundem com uma linguagem de programação.

O presente trabalho procura fazer uma pequena introdução sobre as principais possibilidades de utilização do *awk*, desde a construção de simples linhas de comando até a programação de pequenos scripts para execução no shell do Linux.

Kleber Xavier Sampaio de Souza
Chefe Geral
Embrapa Informática Agropecuária

Sumário

Tutorial do comando *Awk*

Fábio Danilo Vieira

O comando *awk*

Pela quantidade de recursos que possui e pela sua total integração ao *shell (*interface de linha de comando*)* do Unix e Linux, muitos consideram o *awk* como uma linguagem de programação. Comando ou linguagem, todos concordam que o *awk* é uma ferramenta excelente para resolver desde os problemas simples até os mais complexos que aparecem no dia-a-dia de usuários e administradores de sistemas operacionais Unix e Linux.

O *awk* possui esse nome em virtude dos sobrenomes dos três programadores que escreveram o comando: Alfred V. Aho, Peter J. Weinberg e Brian W. Kernighan. Eles queriam desenvolver um comando que englobasse as funções da família grep (grep, egrep e fgrep) e do sed, o que resultou no *awk*.

A função do *awk* é analisar um conjunto de linhas de entrada, uma por uma, procurando as que satisfaçam a determinados padrões ou condições especificados pelo usuário.

Veja a estrutura básica a seguir:

```
[ padrão / condição ]          [ (ação) ]
```

Exemplo:

```
awk '$1 == "Jose" { print $2, $3 }' arquivo1
```

O exemplo anterior ilustra bem uma típica instrução *awk*, ou seja, do tipo "**padrão-ação**". A instrução mostrada imprime o segundo e o terceiro campos de cada linha do arquivo **arquivo1** quando o primeiro campo for igual a "Jose". Em resumo, o comando *awk* irá executar a ação especificada (que pode ter diversos passos) para cada padrão ou condição que encontrar na linha que estiver lendo.

Utilizando o *awk*

Há duas formas de se executar o *awk*:

• A partir da linha de comando, como a seguir:

```
      awk '<padrao-acao>' [arquivo_1] [arquivo_2] ...
[arquivo_n]
```

Note que tanto o padrão quanto a ação vêm entre apóstrofos. Isso serve para inibir do *Shell* a interpretação de caracteres especiais, como o cifrão ($), e também para permitir que o comando se prolongue por quantas linhas forem necessárias.

• A partir de um arquivo de comando, usando a opção -f, como a seguir:

```
      awk -f <arquivo de programa> <lista de arquivos
de entrada>
```

A leitura dos dados é feita linha por linha, que é por padrão uma sequência de caracteres terminada por um *new-line* (ASCII 10). E cada linha (ou registro) lida é dividida em campos, que, por padrão, são uma sequência de caracteres separados por <TAB> (ASCII 9) ou por espaços em branco.

Para continuarmos com nossas próximas explicações, vamos utilizar o arquivo **clientes**, que possui os seguintes registros:

```
$ cat clientes
Claudia Maria     Salvador          555-6666
Eduardo Silva     Campinas          222-3333
Juliano Mendes    Manaus            333-2222
Luiz Carlos       Curitiba          777-9999
Mario Sergio      Florianopolis     888-7777
Maria Julia       Salvador          666-8888
```

É importante notar que cada registro possui os campos: nome da pessoa, cidade e telefone, separados por <TAB>, e, no campo nome, o primeiro nome está separado do segundo por um espaço em branco. Nesse caso, então, temos 4 campos, pois por padrão, o *awk* interpreta **qualquer** espaço em branco como separador de campos. Entretanto, podemos definir o separador que queremos usar. Se definíssemos como **somente** <TAB>, por exemplo, os registros teriam somente 3 campos. Independente disso, o primeiro campo de um registro será chamado de $1, o segundo de $2 e assim por diante. O conjunto de todos os campos, ou seja, o registro inteiro será chamado de $0.

Listagem de dados

Para se fazer uma listagem simples de um arquivo usando o *awk*, pode se usar o seguinte comando::

```
$ awk '{ print }' clientes
Claudia Maria     Salvador          555-6666
Eduardo Silva     Campinas          222-3333
Juliano Mendes    Manaus            333-2222
```

```
Luiz Carlos       Curitiba          777-9999
Mario Sergio      Florianópolis     888-7777
Maria Julia       Salvador          666-8888
```

Observe que a saída é idêntica a produzida pelo comando **cat** anterior. E se desejarmos imprimir apenas o primeiro e o terceiro campos, é só utilizar o $1 e o $3, como no exemplo a seguir:

```
$ awk '{ print $1,  $3 }' clientes
Claudia Salvador
Eduardo Campinas
Juliano Manaus
Luiz Curitiba
Mario Florianópolis
Maria Salvador
```

Note que apenas o primeiro nome, do campo nome, foi exibido, pois o *awk*, por padrão, interpreta tanto espaço em branco como <TAB> como separador de campos. Sendo assim, para o comando o campo cidade está na posição 3. Agora, se desejar que o separador seja o <TAB>, basta digitar o comando abaixo:

```
$ awk 'BEGIN{FS="\t"}{print $1, $3}' clientes
Claudia Maria 555-6666
Eduardo Silva 222-3333
Juliano Mendes 333-2222
Luiz Carlos 777-9999
Mario Sergio 888-7777
Maria Julia 666-8888
```

Pode-se observar que agora o nome é listado por completo, e o campo telefone é considerado o terceiro campo ($3). Mais adiante será explicado o que são as opções BEGIN e FS, mas, o que se pode adiantar é que a

opção FS (*Field Separator*) foi usada para mudar o separador de campos padrão para <TAB> (**t**).

Os padrões usados pelo *awk*

O *awk* possui 3 formas de definir padrões, pelos quais se executam as ações:

- Padrões chamados de expressões relacionais, que servem para selecionar registros que atendam a determinadas condições de **comparação**;
- Padrões chamados de expressões regulares, que servem para selecionar registros que contenham determinadas **cadeias de caracteres**;
- Padrões especiais chamados BEGIN e END, que servem para determinar instruções a serem executadas **ANTES do processamento do primeiro registro e APÓS o último**, respectivamente (utilizado num exemplo anterior).

Trabalhando com expressões relacionais

Os padrões relacionais são usados para estabelecer comparações, sendo que os operadores utilizados são os definidos pela linguagem C (Tabela 1).

Tabela 1. Operadores relacionais utilizados pelo *awk*.

Operadores	Resultados
==	Igual a
>	Maior que
>=	Maior ou Igual
<	Menor que
<=	Menor ou Igual

Também podem ser utilizados os seguintes operadores lógicos (Tabela 2).

Tabela 2. Operadores lógicos utilizados pelo *awk*.

Operadores	Resultados
&&	E
||	Ou
!	Não

No exemplo seguinte, vamos mostrar como é possível utilizar os operadores de comparação com cadeias de caracteres, listando os registros do arquivo **clientes** em que o primeiro campo (com o primeiro nome) inicie a partir da letra J:

```
$ awk '$1 > "J" { print }' clientes
Juliano Mendes  Manaus          333-2222
Luiz Carlos     Curitiba        777-9999
Mario Sergio    Florianópolis   888-7777
Maria Julia     Salvador        666-8888
```

É possível também montar expressões compostas, combinando operadores relacionais simples com os operadores lógicos `||` (ou), `&&` (e) e ! (não).

Supondo que se esteja procurando o telefone de uma pessoa cujo segundo nome sabe-se que começa pela letra S ou M, pode-se usar o comando abaixo:

```
$ awk '$2 >= "M" || $2 >= "S" { print }' clientes
Claudia Maria   Salvador        555-6666
Eduardo Silva   Campinas        222-3333
Juliano Mendes  Manaus          333-2222
Mario Sergio    Florianópolis   888-7777
```

Trabalhando com expressões regulares

Para pesquisarmos um arquivo procurando por uma cadeia de caracteres usando expressões regulares, devemos colocar o padrão a ser encontrado entre um par de barras (/).

Por exemplo, se desejarmos listar os registros de clientes em que os nomes comecem pela letra “M”, usaríamos o seguinte comando:

```
$ awk '/M/ { print }' clientes
Claudia Maria    Salvador        555-6666
Juliano Mendes   Manaus          333-2222
Mario Sergio     Florianópolis   888-7777
Maria Julia      Salvador        666-8888
```

Podemos reparar que os dois primeiros registros não satisfazem o que queremos, mas a pesquisa, como foi feita, está correta. Se quisermos que apenas o nome, ou seja, o primeiro campo se inicie pela letra “M”, devemos usar $1, como o exemplo a seguir:

```
$ awk '$1 ~ /M/ { print }' clientes
Mario Sergio     Florianópolis   888-7777
Maria Julia      Salvador        666-8888
```

No mundo das expressões regulares, os símbolos ( ) [ ] \ ^ $. * ? + | são metacaracteres com sentido especial. Os caracteres ^ e $ servem, por exemplo, para pesquisar no ínicio e no fim de uma cadeia de caracteres, respectivamente. Um grupo de caracteres entre colchetes ([ ]) servirá para pesquisar os registros que tenham um desses caracteres. Se quisermos, então, listar os registros em que o nome ou o segundo nome comecem por “C” ou “J”, fazemos o seguinte:

```
$ awk '$1 ~ /^[CJ]/ || $2 ~ /^[CJ]/ { print }' clientes
Claudia Maria    Salvador        555-6666
```

```
Juliano Mendes  Manaus          333-2222
Luiz Carlos     Curitiba        777-9999
Maria Julia     Salvador        666-8888
```

Podemos fazer uma pesquisa por um intervalo de valores, usando a estrutura:

```
padrão1, padrão2
```

como no exemplo a seguir, onde queremos os registros cujos nomes das pessoas contenham da letra “C” até a “J”:

```
$ awk '$1 ~ /C/ , /J/ { print }' clientes
Claudia Maria   Salvador        555-6666
Eduardo Silva   Campinas        222-3333
Juliano Mendes  Manaus          333-2222
```

Além dos metacaracteres citados anteriormente, o *awk* reconhece os seguintes caracteres (Tabela 3).

Tabela 3. Caracteres especiais utilizados pelo *awk*.

Sequência	Significado
\n	*Newline*
\r	*Carriage Return*
\b	*Backspace*
\f	*FormFeed*
\t	<TAB>I
\ddd	Valor octal ddd

Vale lembrar que o caractere “\t” (do <TAB>) já foi utilizado num exemplo anterior.

Utilizando BEGIN e END

Quando precisamos fazer algum processamento antes que o *awk* faça a leitura do primeiro registro, usamos o padrão BEGIN (para fazer um cabeçalho, por exemplo). Se a necessidade é de fazer um processamento depois da leitura do último registro, devemos usar o padrão END (para gerar totais, por exemplo).

Por exemplo, para se colocar um cabeçalho e uma mensagem "Fim dos registros" no rodapé na leitura do arquivo clientes, deve-se utilizar o seguinte comando:

```
$ awk 'BEGIN { print "Nome\t\tCidade\t\tTelefone" } {
print } END { print "Fim dos registros" }' clientes
Nome            Cidade          Telefone
Claudia Maria   Salvador        555-6666
Eduardo Silva   Campinas        222-3333
Juliano Mendes  Manaus          333-2222
Luiz Carlos     Curitiba        777-9999
Mario Sergio    Florianópolis   888-7777
Maria Julia     Salvador        666-8888
Fim dos registros
```

As variáveis no *awk*

O *awk* trabalha com dois tipos de variáveis:

- Variáveis definidas pelo próprio programador, ou seja, variáveis de trabalho que serão controladas pelo próprio programa.
- Variáveis Internas, que são variáveis pré-definidas e que são muito utilizadas pelo *awk*. Devem ser sempre usadas em letras maiúsculas, sem o cifrão ($) precedendo-as, sendo que cada uma possui uma função.

Na Tabela 4, observa-se as variáveis internas do *awk*.

Tabela 4. Variáveis internas utilizadas pelo *awk*.

Variável	Significado
ARGG	Número de argumentos (parâmetros) recebidos pelo programa
ARGV	Vetor contendo os parâmetros passados para o programa
FILENAME	Nome do arquivo de entrada atual
FNR	Número do registro no arquivo atual
FS	Separador de campos. Seu padrão é branco e <TAB>
NR	Quantidade de registros do arquivo em processamento
OFMT	Formato de saída para números. Seu padrão é %.6g
OFS	Separador de campos na saída. Seu padrão é branco
ORS	Separador dos registros de saída. Seu padrão é newline
RS	Separador de registros de entrada. Seu padrão é newline

Alterando o exemplo dado anteriormente, iremos usar o padrão END para totalizar os registros processados, utilizando-se variável interna NR:

```
$ awk 'BEGIN { print "Nome\t\tCidade\t\tTelefone" }
{ print } END { print "Total de registros = " NR }'
clientes
Nome            Cidade          Telefone
Claudia Maria   Salvador        555-6666
Eduardo Silva   Campinas        222-3333
Juliano Mendes  Manaus          333-2222
Luiz Carlos     Curitiba        777-9999
Mario Sergio    Florianópolis   888-7777
Maria Julia     Salvador        666-8888
Total de registros = 6
```

Vamos ver um exemplo de como contar o número de registros que possuem um determinado padrão:

```
$ awk '$3 ~ /^S/  { print; soma=soma+1 } END {print
"Encontrados ", soma, " registros"}' clientes
Claudia Maria   Salvador        555-6666
Maria Julia     Salvador        666-8888
Encontrados  2  registros
```

No exemplo anterior, foi contado o número de registros em que o nome da cidade iniciasse por "S", utilizando-se uma variável auxiliar chamada soma, a qual o *awk* inicializa automaticamente.

Para o próximo exemplo (e alguns outros também), estaremos utilizando o arquivo carros, o qual é composto do modelo do carro, da velocidade máxima, do tempo mínimo para ir de 0 a 100 km/h, do consumo mínimo e do preço médio. Vejamos o conteúdo do arquivo:

```
% cat carros
Corsa-3portas     150 15.20 12.20 16068.00
Corsa-4portas     182 11.10 10.00 16928.44
Corsa-Sedan       182 11.10 10.00 17376.49
Corsa-Wagon       183 12.20 12.71 20253.45
Pálio             188  9.50 10.90 19974.15
Pálio-Weekend     185 11.92 10.65 21200.44
Tipo              176 11.70 11.00 18310.70
Gol               175 12.40 11.60 16960.50
Parati            173 12.20 11.31 18809.22
```

Desse jeito, estão muito confusas as informações. Para melhorar esta visualização, foi construído um programa *awk*. É possível desenvolver um arquivo com extensão *awk* e construir os comandos *awk* dentro do mesmo. Para torná-lo executável, basta executar o comando "`chown a+x nome_do_arquivo.awk`". Veja como ficou esse programa:

```
$ cat listacar.awk
awk '
BEGIN { printf "%15s %10s %9s %7s %10s\n",
"Modelo", "Vel.Max.", "0 a 100", "Cons.", "Preco"; }
{printf "%15s %7s %11s %8s %11s\n",  $1, $2, $3, $4,
$5;
VelM = VelM + $2 ; Pr = Pr + $5; }
END { printf "\n%7s\n%8s %5.2f\n%5s %10.2f\n",
"MEDIAS:", "Velocidade", VelM / NR, "Preco ", Pr / NR;
}' carros
```

O programa inicia-se com a criação de um cabeçalho dentro do padrão BEGIN, no qual se constrói a formatação desejada pelo comando printf (que veremos mais adiante). Os valores das velocidades e dos preços foram acumulados nas variáveis VelM e Pr, respectivamente e, logo depois, usando o padrão END, listadas as médias que foram calculadas, dividindo-se os totais das variáveis VelM e Pr pela quantidade de registros (NR).

Ao se executar o programa, a saída será a seguinte:

```
% ./listacar.awk
Modelo          Vel.Max.   0 a 100   Cons.      Preco
Corsa-3portas      150       15.20    12.20    16068.00
Corsa-4portas      182       11.10    10.00    16928.44
Corsa-Sedan        182       11.10    10.00    17376.49
Corsa-Wagon        183       12.20    12.71    20253.45
Palio              188        9.50    10.90    19974.15
Pálio-Weekend      185       11.92    10.65    21200.44
Tipo               176       11.70    11.00    18310.70
Gol                175       12.40    11.60    16960.50
Parati             173       12.20    11.31    18809.22

MEDIAS:
Velocidade 177.11
Preco    18431.27
```

Operadores aritméticos do *awk*

O processo de realização de contas em *awk* é muito similar ao da linguagem C. A seguir, na Tabela 5, é mostrada os operadores aritméticos do *awk*.

Tabela 5. Operadores aritméticos utilizados pelo *awk*.

Operadores	Resultados
+	Soma
-	Diferença
*	Multiplicação
/	Divisão
%	Resto de divisão
^ou **	Exponenciação
=	Igualdade

No *awk* também são aceitos operadores abreviados, como na linguagem C. `Exemplos: ++, --, +=, -=, *=, /=, ^=.`

Funções matemáticas

O *awk* possui diversas funções matemáticas internas para manipulação de dados numéricos, as quais estão na Tabela 6.

Tabela 6. Funções matemáticas utilizados pelo *awk*.

Função	Valor retornado
sqrt (x)	Raiz quadrada de x
sin (x)	Seno de x
cos (x)	Cosseno de x
log (x)	Logaritmo natural de x
int (x)	Parte inteira de x
exp (x)	Função exponencial de x
rand (x)	Gera um número randômico entre 0 e 1

Abaixo um exemplo com a função `sqrt` (raiz quadrada):

```
$ awk 'BEGIN { print "Raiz quadrada de 4 eh "sqrt(4) }'
```

Raiz quadrada de 4 eh 2

Veja outro exemplo, só que agora com a função int buscando a parte inteira da raiz quadrada de 10, que devolve um valor decimal:

```
$ awk 'BEGIN { print "Parte inteira da raiz quadrada de
10 eh "int(sqrt(10)) }'
```

Parte inteira da raiz quadrada de 10 eh 3

Trabalhando com cadeias de caracteres

Todas as cadeias de caracteres (ou *strings*) no *awk* são tratadas entre aspas (“ ”).

Essas cadeias de caracteres são formadas concatenando-se variáveis, constantes, campos, elementos de vetores (*arrays*), funções e outros programas.

Observe o exemplo a seguir:

```
$ awk '{ print NR ":" $0}' carros
1:Corsa-3portas 150 15.20 12.20 16068.00
2:Corsa-4portas 182 11.10 10.00 16928.44
3:Corsa-Sedan 182 11.10 10.00 17376.49
4:Corsa-Wagon 183 12.20 12.71 20253.45
5:Pálio 188  9.50 10.90 19974.15
6:Pálio-Weekend 185 11.92 10.65 21200.44
7:Tipo 176 11.70 11.00 18310.70
8:Gol 175 12.40 11.60 16960.50
9:Parati 173 12.20 11.31 18809.22
```

Como pode ser observado, cada registro do arquivo **carros** foi precedido pelo seu número sequencial no arquivo, seguido por dois-pontos (:), sem espaços em branco. Assim sendo, as três cadeias foram concatenadas e a cadeia resultante foi impressa, sem o uso de nenhum operador de concatenação, ou seja, para concatenar cadeias de caracteres basta justapor as mesmas.

O *awk* também possui várias funções internas para tratamento de cadeias de caracteres, que podem ser vistas na tabela seguinte, onde `c1 e c2` são cadeias de caracteres, `exp` é uma expressão regular (que pode ser uma cadeia ou algo com `/exp/`) e p, n são inteiros (Tabela 7).

Tabela 7. Funções manipulação de *strings* utilizados pelo *awk*.

Função	Valor retornado	Retorno
index(c1, c2)	Retorna a posição da cadeia c1 na cadeia c2	Zero se não presente
length(c1)	Retorna o tamanho da cadeia c1	--
match(c1, exp)	Retorna a posição em c1 onde exp ocorre	Zero se não presente
split(c1, v [,c2])	Divide c1 para o vetor v em cada c2 encontrado	Número de campos
sprintf(fmt, lista)	Retorna lista formatada de acordo com fmt	---
sub(exp, c1 [,c2])	A primeira exp encontrada na cadeia c2 é substituída por c1	Número de substituições
gsub(exp, c1 [,c2])	Toda exp encontrada na cadeia c2 é substituída por c1	Número de substituições
substr(c1, p, n)	Retorna a subcadeia que começa na posição p de c1 com n caracteres	--

A seguir, um exemplo usando index, que devolve a posição da primeira ocorrência de “E”:

```
$ awk 'BEGIN { print index("SISTEMA", "E") }'
5
```

Ao se utilizar o padrão BEGIN, não é necessário especificar nenhum nome de arquivo.

Vamos ver um exemplo usando a função length, que retorna o tamanho da string:

```
$ awk 'BEGIN { print length("SISTEMA") }'
7
```

Observe um exemplo usando a função match, que procura pela ocorrência de um padrão dentro de uma cadeia de caracteres:

```
$ awk 'BEGIN { print match("Joao Jose", /Jo+s/) }'
6
```

Foi retornado o valor 6, que corresponde ao nome “Jose”, pois foi procurado por uma sequência “Jo” seguida ou não de outro caractere “o” mais o caractere “s”.

A seguir, um exemplo usando a função `split`, que divide uma cadeia de caracteres num vetor de cadeias dependendo do separador usado em seu terceiro argumento (caso este seja passado, senão utiliza-se o valor da variável interna FS):

```
$ awk 'BEGIN { print split("ponto-e-virgula", v, "-")
}'
3
```

O retorno da função mostra que a cadeia foi dividida em 3 partes para o vetor v, que se listássemos, exibiria:

```
v[1] = "ponto"
v[2] = "e"
v[3] = "virgula"
```

O comando `if`

O comando `if` no *awk*, como na linguagem C, possui a seguinte sintaxe:

```
if (expressao)
   {
    comando 1
    comando 2
    . . .
    comando n
   }
else
   {
    comando 1
    comando 2
    . . .
    comando n
```

```
    }
```

Dentro de expressão pode ser incluídos operadores lógicos (`||`, `&&`, `!`), operadores relacionais (`<`, `<=`, `>`, `>=`, `==` e `!=`) e operadores de pesquisa em expressões regulares (`~` e `!~`).

Observe um exemplo do comando `if` utilizando o arquivo **carros** que, ao seu final, exibirá o carro mais veloz e o mais econômico:

```
$ cat velecon.awk
awk 'BEGIN { MinCons=99999 }
{
       if ( $2 > MaxVel )
       {
         CarVel=$1
         MaxVel=$2
       }
       if ( $4 < MinCons )
       {
         CarEcon=$1
         MinCons=$4
       }
    }
    END {
       print "O", CarVel, "eh o mais rápido, pois de-
senvolve ", \
            MaxVel, "km e\nO", CarEcon, "eh o mais eco-
nômico, pois faz", MinCons, "km/l\n"
}' carros
```

Executando este arquivo *awk*, a saída será:

```
$ ./velecon.awk
O Pálio eh o mais rápido, pois desenvolve 188 km e
O Corsa-4portas eh o mais econômico, pois faz 10.00
km/l
```

O comando `while`

A sintaxe do comando `while` é o seguinte:

```
while (expressao)
        {
          comando 1
          comando 2
          . . .
          comando n
        }
```

Os comandos dentro de `while` são executados até que o valor de expressão seja falso `(0 – zero)`.

Vejamos um exemplo de `while` para listar cada campo do arquivo **carros** em uma linha:

```
$ cat camposcar.awk
awk '{
    i = 1
    while ( i <= NF )
    {
        print $i
        i ++
    }
}' carros
```

O script inicializará a variável `i` com 1 e o comando `while`, enquanto o valor de i for menor ou igual ao número de campos da linha atual, irá imprimir (`print`) cada campo em uma linha diferente, acrescentando 1 a i após a impressão (`print`). A saída gerada por esse `script` *awk* será a seguinte, com cada campo lido em uma linha diferente:

```
$ ./camposcar.awk
Corsa-3portas
150
15.20
12.20
16068.00
Corsa-4portas
182
11.10
10.00
16928.44
Corsa-Sedan
182
11.10
10.00
17376.49
Corsa-Wagon
183
12.20
12.71
20253.45
Pálio
188
9.50
10.90
19974.15
Pálio-Weekend
185
11.92
10.65
21200.44
Tipo
```

```
176
11.70
11.00
18310.70
Gol
175
12.40
11.60
16960.50
Parati
173
12.20
11.31
18809.22
```

O comando `for`

A sintaxe do comando `for` é a seguinte:

```
for (expressao1; expressao2; expressao3)
     {
      comando 1
      comando 2
      ...
      comando n
     }
```

onde em expressao1 se atribui, geralmente, o valor inicial de uma variável, que será incrementada pelo valor de expressão3, sendo que as instruções dentro das chaves ({}) serão executadas enquanto expressao2 for verdadeira.

Veja como o exemplo anterior foi alterado para usar o comando `for` ao

invés do `while`:

```
$ cat camposcar2.awk
awk '{
    for ( i = 1; i <= NF; i++ )
    {
        print $i
    }
}' carros
```

Nesse exemplo, o comando for inicializou a variável i com 1, incrementando-a em 1 até que seu valor fosse menor ou igual ao número de campos do registro (NF). Ou seja, fez o mesmo que o comando `while` no exemplo anterior a este, sendo que a saída é exatamente igual também.

Um resumo dos comandos `break, continue, next e exit`

A seguir é dado um resumo dos comandos utilizados dentro dos comandos de laço explicados anteriormente, `while e for`:

- `break` – quando esse comando é encontrado, a execução do programa desvia para a primeira instrução após o `loop;`
- `continue` – assim que é encontrado, o fluxo do programa é desviado para a primeira instrução do `loop`;
- `next` – provoca o desvio do fluxo de comandos e voltará a pesquisar padrões a partir do comando da primeira dupla “padrão-ação”;
- `exit` – o comando `exit` faz o programa se comportar como se a leitura do arquivo tivesse acabado. Nada mais é lido e, se existir o padrão END, ele será executado.

Trabalhando com vetores

Os vetores e seus elementos são tratados da mesma forma que as variáveis, não necessitando também que sejam declarados, ou seja, passarão a existir assim que lhes for atribuído um valor.

A seguir, um exemplo de utilização de vetor para listar o arquivo **clientes** de trás para a frente:

```
$ cat tac.awk
awk ‘
{
    Registro [NR] = $0;
}
END {
    for ( i = NR; i >= 1; i-- )
    {
        print Registro[i]
    }
}’ clientes
```

Nesse exemplo, foi guardado o conteúdo de cada linha do arquivo **clientes**, representada por $0, dentro do vetor **Registro**. Usando o comando `for`, o programa faz um `loop` para listar cada linha do vetor, partindo da última linha (NR) até a primeira. Executando este `script`, o resultado é o seguinte:

```
$ ./tac.awk
Maria Julia     Salvador        666-8888
Mario Sergio    Florianópolis   888-7777
Luiz Carlos     Curitiba        777-9999
Juliano Mendes  Manaus          333-2222
Eduardo Silva   Campinas        222-3333
```

```
Claudia Maria   Salvador        555-6666
```

O *awk* permite também que indexemos os vetores com valores não numéricos. No exemplo a seguir, foi criado um vetor que armazena o campo velocidade do arquivo **carros**, sendo que o indexador é o campo modelo (não numérico):

```
$ cat indexauto.awk
awk '
{
    Velocidades [$1] = $2;
}
END {
    for ( Modelo in Velocidades )
    {
        print Modelo, "\t", Velocidades[Modelo]
    }
}' carros
```

Após a execução desse programa, a saída foi a seguinte:

```
$ ./indexauto.awk
Pálio-Weekend     185
Pálio             188
Tipo              176
Corsa-4portas     182
Gol               175
Corsa-Wagon       183
Parati            173
Corsa-Sedan       182
Corsa-3portas     150
```

A saída com *print*

O comando print é a instrução mais usada para saída de dados no *awk*. Ele é usado para saída de dados simples, sem muita formatação.

Seu formato básico é:

```
print expressao_1, expressao_2, ... expressao_n
```

Ele imprime o valor de cada expressão separado pelo OFS colocando um ORS ao seu final

cat indexauto.awk

```
awk '
{
    Velocidades [$1] = $2
}
END {
    for ( Modelo in Velocidades )
    {
        print Modelo, "\t", Velocidades[Modelo] |
"sort"
    }
}' carros
```

o resultado será outro:

$ ./indexauto.awk

```
Corsa-3portas     150
Corsa-4portas     182
Corsa-Sedan       182
Corsa-Wagon       183
Gol               175
Pálio             188
```

```
Pálio-Weekend      185
Parati             173
Tipo               176
```

A saída formatada com *printf*

O comando printf possui a seguinte sintaxe:

```
printf formato, expressao_1, expressao_2, ...
expressao_n
```

Onde formato é uma cadeia que contém tanto as informações a serem impressas quanto as definições dos formatos das cadeias resultantes de `expressao_1, expressao_2, ... expressao_n`. Os formatos sempre começam por um % e terminam por uma letra, conforme mostrada na Tabela 8.

Tabela 8. Opções do comando *printf*.

Letra	A expressão será impressa como:
c	Simples caractere
d	Número no sistema decimal
e	Notação científica exponencial
f	Número com ponto decimal
g	O menor entre os formatos %e e %f com supressão dos zeros não significativos
o	Número no sistema octal
s	Cadeia de caracteres
x	Número no sistema hexadecimal
%	Imprime um %

Na Tabela 9, pode-se ver exemplos de uso do *printf:*

Tabela 9. Exemplos de uso do *printf* no comando *awk*.

Entrada	Saída	
printf “%d”, 99/2	49	
printf “%e”, 99/2	4.950000e+01	Notação científica
printf “%f”, 99/2	49.500000	
printf “%6.2f”, 99/2	49.50	6 inteiros e 2 decimais sem zero à esquerda
printf “%g”, 99/2	49.50	
printf “%o”, 9	11	9 convertido para octal dá 11
printf “%x”, 20	14	20 convertido para hexadecimal dá 14
printf “|%s|”, “Flamary”	|Flamary|	Formato de edição de cadeias
printf “|%15s|”, “Flamary”	| Flamary|	Ídem alinhando à direita, totalizando 15 posições
printf “|%-15s|”, “Flamary”	|Flamary |	15 posições alinhadas à esquerda
printf “|%.3smengo|”, “Flamary”	|Flamengo|	Só as 3 primeiras posições + cadeia mengo
printf “|%-10.2s|”, “Eugenio”	|Eu |	10 posições alinhadas à esquerda, pegando as 2 primeiras letras

Recebendo parâmetros

O *awk* permite também que passemos parâmetros para programas escritos com ele. Esses parâmetros serão recebidos por uma variável inteira e por um vetor. Essas variáveis são as seguintes:

- ARGC – Essa variável contém a quantidade de parâmetros recebidos;
- ARGV – Vetor que contém os valores recebidos. Seus elementos variam desde ARGV[0], que contém o nome da rotina, até ARGV[ARGC – 1].

Vejamos um exemplo com o uso dessas variáveis:

```
$ cat param.awk
awk '
BEGIN {
      for (i = 0; i < ARGC; i++)
```

```
    {
        print ARGV[i], "\n"
    }
}' $*
```

Trabalhando com o *Shell* do Linux

Em todos os exemplos mostrados até aqui, com exceção do anterior, o *awk* não interagiu com o *shell* (interface de linha de comando) do Linux. Nesses exemplos, para evitar que o *shell* interpretasse algum caractere das instruções do *awk* (já que alguns deles são significativos para o *shell*), sempre eram colocados apóstrofos em volta dessas instruções.

Vamos ver um exemplo em que isso pode ser percebido, pois o $1 será interpretado pelo *shell*, e não pelo *awk*:

```
$ cat procura.awk
awk '/'$1'/' clientes
```

Observe que o que estão entre apóstrofos são os caractere barra (/), e não $1. Portanto, $1 está sob interpretação do *shell*, e não do *awk*. Se fosse interpretado pelo *awk*, $1 seria o primeiro campo do arquivo, como sabemos, mas para o *shell*, $1 será interpretado como sendo o primeiro parâmetro passado para o programa `procura.awk`. Veja um exemplo da execução dele:

```
$ ./procura.awk Mario
Mario Sergio    Florianópolis   888-7777
```

O programa procurou pelo parâmetro passado “Mario” dentro do arquivo **clientes**, encontrando apenas um registro.

Em resumo, a possibilidade de programas e linhas de comandos que se pode desenvolver que lidam com arquivos textos utilizando *awk* é infinita, bastando apenas o programador ter uma necessidade em que este co-

mando possa atuar. Obviamente, haverá casos em que o *awk* não será a solução ideal, mas saber desenvolver com esta ferramenta poderá ajudar bastante enquanto essa solução ideal não seja encontrada.

Literatura recomendada

AURELIO.NET [Site do autor]. Disponível em: <http://aurelio.net/*shell*>. Acesso em: 13 dez. 2010.

NEVES, J. C. **Programação *Shell* Linux**. Rio de Janeiro: Brasport, 2005. p. 261-295.

Informática Agropecuária

Ministério da
Agricultura, Pecuária
e Abastecimento

BRASIL
UM PAÍS DE TODOS
GOVERNO FEDERAL

CGPE 9059