Introdução

O awk é um utilitário poderoso aplicável a qualquer situação onde seja necessário extrair conteúdo a partir de arquivos texto com formatação coerente. O nome awk não tem tradução, e consiste nas iniciais dos seus três autores, um dos quais é também co-autor de outra linguagem famosa: o C. O awk está disponível desde 1977, mas as versões disponíveis atualmente são mais recentes, datando da segunda metade da década de 80.

Um programa em awk tem a seguinte forma geral:

```
padrão_1 { ação_1 }
padrão_2 { ação_2 }
padrão_n { ação_n }
```

Você pode gravar os programas em awk na forma de arquivos texto (em geral com a extensão ".awk", mas isto não é obrigatório), e executá-los invocando o interpretador awk.

Para entender o mecanismo do awk, nada melhor que um exemplo. Considere um relatório de presenças de uma turma de universidade, contendo o nome de cada aluno, o código de sua turma e o número de faltas no semestre corrente, gravado com o nome de chamada.txt. Seu formato (pouco usual, mas ideal para o processamento pelo awk) é como segue:

```
Augusto Cesar Campos;3001;6
Ronaldo Hornburg;3001;2
Flavio Luis Costa;3002;3
Gabriela Silva;3002;8
Arnaldo Cesar Coelho;3003;15
```

Note que temos um registro por linha, e todos os campos são separados por um símbolo fixo e que não ocorre no conteúdo - o ponto-e-vírgula. Estas são condições ideais para o processamento em awk, e muitas vezes você passará muito mais tempo tentando garantir que seus arquivos de entrada alcancem a forma ideal, do que realmente processando-os para obter o resultado esperado.

Vamos a um exemplo simples: a partir do arquivo chamada.txt, gerar uma listagem simples dos alunos que tiverem ultrapassado o limite de faltas aceitável, que é de 12 faltas.

Crie o seguinte arquivo com o nome de faltosos.awk:

```
$3 > 12 {
          OFS=";"          # define qual deverá ser o separador de campos na saída
          print $1,$2,$3   # imprime os 3 campos
        }
```

e execute-o assim:

```
awk -F ";" -f faltosos.awk chamada.txt
```

Note que há um parâmetro extra, o -F, que serve para definir qual o caracter separador de campos no arquivo de entrada. Se você não informa o -F, o awk sempre assume que os brancos (um ou mais espaços ou tabs) serão os separadores.

Como esperado, o programa acima produzirá a seguinte saída:

```
Arnaldo Cesar Coelho;3003;14
```

Vamos agora dissecá-lo. Note que o programa não tem nenhuma espécie de cabeçalho, iniciando diretamente com uma estrutura típica de awk - uma condição, seguida por uma ação colocada entre chaves.

A condição oferecida é $3 > 12 - ou seja, a ação correspondente será efetuada apenas para as linhas do arquivo de entrada onde o valor do terceiro campo for maior que 12. Neste caso, a ação é bastante simples: é definido o separador dos campos do registro de saída como sendo o ponto-e-vírgula na variável interna OFS (algo que você não precisa fazer, mas a tradição dos filtros de Unix manda sempre que possível devolver a saída no mesmo formato da entrada), e em seguida imprime na saída padrão (em geral a tela) os três campos do arquivo de entrada.

O awk irá ler o arquivo de entrada, linha por linha, e para cada linha da entrada irá testar todas as condições (nao apenas a primeira encontrada!), executando as ações pertinentes. Você pode ter um número indefinido de conjuntos de condições e ações, todas elas com o mesmo formato acima.

Processando cabeçalhos e rodapés

Muitas vezes você irá precisar mudar o formato do arquivo, acrescentar cabeçalhos e rodapés (ao arquivo de saída, e não a cada página individual), calculando totais para o conjunto das linhas e outras operações comuns. O awk tem suporte a tudo isto, como veremos a seguir.

Experimente o seguinte programa resumo_chamada1.awk:

```
$3 > 12 {
        reprovados++
       }
{
  alunos++
  total_faltas = total_faltas + $3
}
END {
  if (reprovados>0) {
    print "Neste semestre,",reprovados,"alunos rodaram por faltas."
  }
  print "Foi registrado um total de",total_faltas,"faltas,"
  print "perfazendo uma média de",total_faltas/alunos," por aluno."
}
```

Execute-o com o comando

```
awk -F ";" -f resumo_chamada1.awk chamada.txt
```

e você deverá ver a seguinte saída:

```
  Neste semestre, 1 alunos rodaram por faltas.
  Foi registrado um total de 33 faltas,
  perfazendo uma média de 5.66667  por aluno.
```

Creio que a estrutura do programa deve parecer óbvia para qualquer pessoa com experiência em linguagens de terceira geração, mas alguns comentários podem ser úteis.

Em primeiro lugar, note que nenhuma variável é declarada. Os tipos e tamanhos das variáveis são definidos pelo contexto, e você pode perfeitamente somar valores a uma variável que tem conteúdo string, ou concatenar uma string a uma variável que tem conteúdo numérico. Uma consulta à documentação explicará como o awk trata este tipo de operação incomum.

Note também que não é necessário definir nada em especial para manter múltiplas condições em um mesmo programa. O interpretador awk irá testar todas elas.

Propositalmente usei duas condições especiais no mesmo programa. Note que o segundo bloco de ações não está associado a condição nenhuma - isto significa que ele deverá ser executado para todas as linhas.

O terceiro bloco está associado a outra condição especial - a condição END (sempre em maiúsculas), que é executada após a leitura do último registro de entrada. Uma condição análoga é a BEGIN, executada antes do interpretador realizar a abertura do primeiro arquivo de entrada.

Note que usei dois estilos de atribuições de expressões a variáveis: o estilo direto, como em total_faltas = total_faltas + $3, e o estilo da linguagem C, como em alunos++. O awk suporta os dois formatos, e eu poderia ter escrito de outras maneiras, como em alunos = alunos + 1 e alunos += 1. Note que o sinal de = sempre significa atribuição - se você precisar fazer uma comparação, será preciso escrever um duplo sinal de igual ("=="). Veja:

Certo: if (a == 0) { print "zero!" }

Errado: if (a = 0) { print "zero!" }

Analisando a saída do programa, veja que a formatação dos valores numéricos está bastante feia - nem sempre você irá querer valores com 5 casas decimais. Para evitar isto, você pode usar o printf (típico da linguagem C e do mundo Unix) com sua sintaxe padrão, inclusa na documentação do awk. Se escrevêssemos a última linha do programa assim:

```
printf ("perfazendo uma média de %5.2f por aluno.\n",
        total_faltas/alunos);
```

Teríamos o seguinte resultado:

```
Neste semestre, 1 alunos rodaram por faltas.
Foi registrado um total de 34 faltas,
perfazendo uma média de  5.67 por aluno.
```

Tabela do significado das letras:

%c = um caracter
%d, %i = um inteiro decimal
%e, %E = um número em notação científica
%f = um número em ponto flutuante
%G, %g = um número em notação científica ou ponto flutuante - o que for menor
%o = um inteiro octal sem sinal
%s = uma string
%x, %X = um número hexadecimal

Por exemplo, para escrever um número real em notação de ponto flutuante, ocupando 6 casas, sendo 2 após o ponto, use %6.2f.

As expressões

um dos pontos que diferenciam as linguagens de programação entre si é a notação utilizada para descrever suas expressões. Já vimos alguns exemplos, e vamos agora tentar listar todas as possibilidades.

Em primeiro lugar, as definições de tipos. O awk irá reconhecer como número qualquer valor que seja um inteiro, um valor decimal ou um número em notação científica. Exemplos: 102, 102.0, 1.02e+2, 1020e-1

Uma string em geral será definida como uma sequência de caracteres incluída entre aspas duplas. Exemplo: "eleitorado da flórida".

Variáveis não precisam de declaração - elas recebem seu valor logo no seu primeiro uso. Ao contrário de outras linguagens como o perl, as variáveis não precisam de nenhum prefixo especial (como

$variavel). O prefixo $ é utilizado apenas para referenciar os campos do registro de entrada, onde $1 é o primeiro campo, $2 é o segundo campo, e assim por diante - $0 é o registro inteiro. A variável NF sempre conterá o número de campos do registro corrente, e NR indica o número do registro corrente.

As operações aritméticas disponíveis são as seguintes, já em ordem de precedência:

-x - negação
x^y - potenciação
x*y - multiplicação
x/y - divisão
x%y - resto da divisão
x+y - soma
x-y - subtração

A única operação com strings é a concatenação, e ela é feita sem necessidade de sinais. Exemplos:

```
detalhe = "O resultado é: " result
print "Atenção: " detalhe
frase = palavra1 " " palavra2 palavra3
```

Há toda uma lista de variáveis internas úteis, e você pode encontrá-las na documentação. Alguns destaques:

OFMT - formato (estilo do printf) usado na conversão de números para visualização.
FS - separador de campos da entrada
OFS - separador dos campos de saída
ORS - separador de registros de saída
ARGC, ARGV - descritores de parâmetros de chamada
ENVIRON - vetor contendo todas as variáveis do ambiente
ERRNO - número do erro ocorrido na última chamada ao sistema
NF - número de campos na linha corrente
NR - número do registro corrente

As funções internas

O awk não inova nas funções disponíveis para o programador. Vejamos uma lista das mais importantes; deixamos à documentação a tarefa de explicar os detalhes. Lembre-se de que você também pode definir suas próprias funções, o que contribui bastante para a criação de programas modularizados.

Funções numéricas:

int(x) - "trunca" o valor de x, desprezando a parte decimal.
sqrt(x) - raiz quadrada positiva
exp(x) - exponencial de x, ou seja, e elevado a x
log(x) - logaritmo natural de x
sin(x) - seno de x - em radianos
cos(x) - cosseno de x - em radianos
atan2(y,x) - arco tangente de y/x em radianos
rand() - número real pseudo-aleatório entre 0 e 1
srand(x) - inicializa o gerador de números aleatórios

Funções de strings

index(string, substring) - posição da primeira ocorrência da substring dentro da string.
length(string) - comprimento da string
match(string,regexp) - verifica se a expressão regular regexp ocorre dentro da string
split(string, array, [separador]) - separa os componentes da string em elementos do array, usando opcionalmente o separador.
sprintf(formato, expressões) - formata as expressões de acordo com as definições do formato, de maneira análoga ao comando printf.
sub(regexp,valor,string) - procura uma ocorrência da expressão regexp em string, e substitui por

valor.
gsub(regexp,valor,string) - similar a sub, mas substitui todas as ocorrências encontradas, e não apenas a primeira.
gensub(regexp,replacement,tipo,string) - substituição genérica
substr(string,inicio,tamanho) - retorna uma substring de string, começando da posição inicio e contendo tamanho caracteres.
tolower(string) - converte para minúsculas
toupper(string) - converte para maiúsculas

Entrada e saída

close(nome) - fecha um arquivo ou pipe - desnecessária em scripts simples
fflush(nome) - esvazia o buffer
system(comando) - executa o comando através da shell

Tempo

systime() - retorna o número de segundos decorrido desde 1/1/1970
strftime(formato,tempo) - retorna uma data formatada de acordo com as definições do comando date do Unix.

Controle de fluxo

Já vimos exemplos da forma mais simples de seleção, o if. O awk suporta o if com a seguinte sintaxe:

```
if (x%2 == 0) {
  print "x é par"
} else {
  print "x é ímpar"
}
```

A condição sempre deve estar entre parênteses!

O conceito de laço também está implementado, com teste no começo ou no final do bloco. Exemplo:

```
# teste no inicio
while (i <= 3) {
  print $i
  i++
}

# teste no final
do {
  i++
  print $i
} while (i <= 3)
```

A contagem através do for funciona com a sintaxe da linguagem C: for (inicialização; condição; incremento). Exemplo:

```
for (i=1; i <= 3; i++) {
  print $i
}
```

O comando break pode ser usado para sair de um loop a qualquer momento. O comando continue pode ser usado para forçar o próximo ciclo de um loop, pulando os próximos passos ainda restantes.

Outros comandos ainda têm influência sobre o fluxo da execução de programas. O comando next força a leitura do próximo registro de entrada, sem verificar as demais condições do programa. O nextfile fecha o arquivo de entrada corrente, e passa a processar o próximo, se houver. O comando exit encerra a execução do programa como um todo.

Vetores (arrays)

A linguagem awk permite o uso de vetores (unidimensionais por definição), com sintaxe parecida com a da linguagem pascal, colocando o índice entre colchetes ao lado do nome do vetor. As regras para atribuição de valores a vetores são as mesmas do que as das variáveis comuns:

```
vetor[15] = "fosfosol";
```

Vamos a mais um exemplo baseado no nosso arquivo de chamadas - chame-o de por_turma.awk:

```
{
 if (!($2 in faltas)) {
   conta_turmas++;
   turmas[conta_turmas]=$2
 }
 faltas[$2] += $3
 alunos[$2]++;
}

END {
 for (i=1;i<=conta_turmas;i++)  {
   print "Média de faltas da turma",turmas[i],
        ":",faltas[turmas[i]]/alunos[turmas[i]]
 }
}
```

Execute-o com:

```
awk -F ";" -f por_turma.awk chamada.txt
```

para ver o seguinte resultado:

```
Média de faltas da turma 1 : 4
Média de faltas da turma 2 : 5.5
Média de faltas da turma 3 : 15
```

Note que mantemos três vetores separados - o primeiro, chamado turmas, é indexado de forma sequencial (índice crescente a partir de 1). Os outros dois, chamados de faltas e alunos, são indexados pelo código da turma.

O primeiro if (if (!($2 in faltas))) verifica se o segundo campo da linha corrente ainda não consta como índice do vetor faltas. Caso ainda não conste, o valor de conta_turmas é incrementado, e um novo item é acrescentado ao vetor turmas. Note o uso de um ponto de exclamação para negar (not lógico) a condição do if!

Logo em seguida, o total de faltas do registro corrente é somado ao registro da turma correspondente (caso ele ainda não exista, é criado automaticamente), e o contador de alunos da turma é incrementado. Tudo isto em uma condição vazia, executada para todas as linhas de entrada.

Em seguida, em uma condição END (executada apenas ao fim do arquivo) de entrada, fazemos uma repetição for simples, iterando todos os itens do vetor turmas e utilizando-os como índice dos vetores de faltas e alunos para calcular as médias.

Usando Outros programas

O awk não tem a riqueza de linguagens similares como o Perl, capaz de interfacear diretamente com as chamadas do sistema operacional. Desta forma, muito se faz através da interação com outros utilitários, como o sort e o grep (embora em geral ele não seja necessário). Você pode chamar estes utilitários diretamente a partir do seu programa awk, ou pode montar encadeamentos através das

pipes da shell. Vamos a um exemplo prático, lembrando que esta não é a melhor maneira de fazer isto - é apenas para fins didáticos!

Vamos construir um pequeno script capaz de mostrar os nomes e tamanhos dos 10 maiores arquivos de um diretório, e ao final exibir a soma total dos seus tamanhos.

Embora o ls tenha seu próprio mecanismo de ordenação, vamos usar o comando sort para ter maior efeito didático. Em primeiro lugar, edite o script awk capaz de realizar a parte "quente" da nossa tarefa, e salve-o como 10maiores.awk

```
FNR <= 10 {
  total=total+$5
  print FNR,$9,$5
}
END {
  print total
}
```

Note que a primeira condição verifica se o número da linha corrente é menor ou igual a 10!

Agora execute, na shell, a seguinte linha contendo uma cadeia de comandos:

```
  ls -l | sort -nrk5 | awk -f 10maiores.awk
```

Esta linha tira proveito do mecanismo conhecido como pipe (simbolizado pela barra vertical), onde a saída de um comando é utilizada como entrada pelo próximo comando. Note que o comando sort ordena uma lista qualquer, e os parâmetros utilizados (nrk5) significam: "ordenar em ordem numérica [n] invertida [r] pelo quinto campo [k5]". Você deverá ver uma saída como a que segue:

```
1 bigfile.gzgz 2167512
2 rejeitados.zip 377106
3 kmago-0.4.tar.gz 270939
4 timofometro_1.1.tar.gz 204782
5 acomjufa.zip 145368
6 ATT00190.jpg 52861
7 edy3.jpg 45230
8 pppcosts-0.66.src.tar.gz 40031
9 pppcosts-0.66.ELF.tar.gz 23274
10 procmail-sanitizer.tar.gz 22987
3350090
```

Printf

You use *printf* for formatted printing. printf prints everything on one line if you don't put newline charater into format string, constants to be printed or into variables to be printed. Format string contains text and conversion specifications. Conversion specifications consist of percent character "%", zero or more flags, field width, precision, conversion specifier and conversion qualifier. Percent character and conversion specifier must be there, other are optional. Conversion specifications correspond one by one to the variables or constants to be printed. Most common *conversion specifiers* are:

%d signed integer
%e signed fractional with exponent
%f signed fractional without exponent
%s sequence of characters in corresponding variable or constant
%% percent character

Escape sequences are used to format printf output. Most common *escape sequences* are:

\f form feed, new page
\n new line (\012 or \015)
\r carriage return, overprint
\t horizontal tab
\v vertical tab
\' single quote
\" double quote
\\ backslash
\0 null (character value 000)
\a alert, bell
\b backspace
\040 space
\ddd octal notation
\xddd hexadecimal notation

Some printf examples:

```
printf("%d %e %f", A, B, C)
printf("%s %d %% %s", "It is", proof, " alcohol")
printf("First line, number %d \n and second line, number %d", n1, n2)
```