Awk em Exemplos, Parte 3

funções string e ...controles de cheques?

Daniel Robbins
Presidente/CEO, Gentoo Technologies, Inc.
Abril de 2001

Em sua conclusão desta série sobre o awk, Daniel introduz algumas funções string importantes do awk, e então mostra como escrever um programa de balanço de cheques completo do princípio. Junto, você irá aprender como escrever suas próprias funções, e usar os arrays multidimensionais do awk. No fim do artigo, você vai ter mais experiência com o awk, permitindo qeu você crie scripts mais poderosos.

Formatando a saída

Mesmo que a declaração print do awk faça o trabalho a maioria das vezes, às vezes é necessário mais. Para estas ocasiões, o awk oferece duas velhas amigas chamadas printf() e sprintf(). Sim, estas função, como muitas outras partes do awk, são idênticas às suas contrapartes do C. O printf() irá escrever uma string formatada em stdout, enquanto sprintf() retorna uma string formatada que pode ser atribuída a uma variável. Se você não está familiarizado com printf() e sprintf(), um artigo introdutório de C irá introduzir rapidamente estas duas funções de impressão. Você pode ver a página man do printf() escrevendo "man 3 printf" em seu sistema Linux.

Aqui temos um código awk exemplo com o sprintf() e printf(). Como você pode ver, tudo é quase idêntico ao C.

```
x=1
b="foo"
printf("%s got a %d on the last test\n","Jim",83)
myout=sprintf("%s-%d",b,x)
print myout
```

O código irá escrever:

```
Jim got a 83 on the last test
foo-1
```

Funções string

O awk possui funções string em abundância, e isto é bom. No awk, as funções string são realmente necessárias, pois não é possível tratar uma string como um array de caracteres como em outras linguagens, como o C, C++ e Python. Por exemplo, se o código abaixo for executado:

```
mystring="How are you doing today?"
print mystring[3]
```

será gerada uma mensagem de erro como a abaixo:

```
awk: string.gawk:59: fatal: attempt to use scalar as array
```

Apesar de não serem tão convenientes quanto os tipos seqüência do Python, as funções string do awk servem fazem o serviço. Vamos dar uma olhada nelas.

Primeiro, temos a função básica length(), que retorna o comprimento de uma string. Veja como usá-la:

```
print length(mystring)
```

Este código irá imprimir o valor:

```
24
```

OK, vamos adiante. A próxima função de string é chamada index, e irá retornar a posição da ocorrência de uma substring em outra string. Ela irá retornar 0 se a string não for encontrada. Usando mystring, podemos usar a função desta forma:

```
print index(mystring,"you")
```

O awk escreve:

```
9
```

Vamos passar agora para duas funções mais fáceis, tolower() e toupper(). Como você pode estar adivinhando, estas funções irão retornar a string com todos os caracteres convertidos para minúsculas ou maiúsculas, respectivamente. Note que tolower() e toupper() retornam a nova string, e não modificam a original. Este código:

```
print tolower(mystring)
print toupper(mystring)
print mystring
```

Irá produzir esta saída:

```
how are you doing today?
HOW ARE YOU DOING TODAY?
How are you doing today?
```

Até agora, tudo bem, mas exatamente como selecionamos uma substring ou mesmo um único caracter de uma string? É aqui que substr() vem. A chamada a substr() é feita assim:

```
mysub=substr(mystr,startpos,maxlen)
```

mystring deve ter ou uma variável string ou uma string literal da qual você quer extrair uma substring. startpos deve estar configurada para o caracter de início, e maxlen deve conter o comprimento máximo da string que deve ser extraída. Note que eu disse *comprimento máximo*. Se lengh(mystring) é mais curto que startpos+maxlen, o resultado será truncado. substr() não irá modificar a string original, mas irá retornar a substring. Veja um exemplo:

```
print substr(mystring,9,3)
```

O awk irá escrever

```
you
```

Se você programa regularmente em uma linguagem que usa índices de array para acessar partes de uma string (e quem não faz?), faça uma nota mental que o substr() é o substituto awk. Você precisará usá-lo para extrair caracteres e substring, e como o awk é uma linguagem baseada em strings, você irá utilizar isto com freqüência.

Agora vamos passar para algumas funções com mais substância, a primeira é chamada match(). O match() é bastante parecida com o index(), exceto que em vez de procurar por uma substring como o index() faz, ele procura por uma expressão regular. A função match() irá retornar a posição inicial da combinação, ou zero se não houver combinação. Além disso, o match() irá configurar duas variáveis chamadas RSTART e RLENGTH. RSTART contém o valor de retorno (a localização da primeira combinação), e RLENGTH irá conter seu comprimento em caracteres (ou -1 se nenhuma combinação for encontrada). Usando RSTART, RLENGTH, substr(), e um pequeno laço, você pode facilmente fazer iterações sobre todas as combinações encontradas em sua string. Veja um exemplo da chamada ao match():

```
print match(mystring,/you/), RSTART, RLENGTH
```

O awk irá escrever:

```
9 9 3
```

Substituições de string

Vamos, agora, olhar duas funções de substituição de string, sub() e gsub(). Estas são diferentes das funções anteriores por que *modificam a string original*. Veja um modelo que mostra como chamar sub():

```
sub(regexp,replstring,mystring)
```

Quando você chama sub(), ela irá procurar a primeira seqüência de caracteres em mystring que combina com a regexp, e irá substituir aquela seqüência com replstring. sub() e gsub() possuem argumentos idênticos, a única coisa que diferencia elas é que sub() irá substituir a primeira combinação com regexp que encontrar (se houver alguma), e o gsub() executa uma substituição global, trocando todas as combinações de regexp. Veja um exemplo da chamada de su() e gsub():

```
sub(/o/,"0",mystring)
print mystring
mystring="How are you doing today?"
gsub(/o/,"0",mystring)
print mystring
```

Precisamos reconfigurar mystring para seu valor original por que a primeira chamada a sub() modificou diretamente mystring. Quando executado, este código fará com que o awk apresente:

```
H0w are you doing today?
H0w are y0u d0ing t0day?
```

Obviamente expressões regex mais complexas são possíveis. Fica a critério do leitor testar algumas regexp mais complicadas.

Completaremos nossa cobertura das funções de string introduzindo a função split(). O trabalho de split() é "cortar" uma string, e colocar as várias partes em um array indexada por inteiro. Aqui temos um exemplo da chamada a split():

```
numelements=split("Jan,Feb,Mar,Mar,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov,Dec",mymonths,",")
```

Quando chamamos split(), o primeiro argumento contém a string literarl ou variável string a ser cortada. No segundo argumento, você especifica o nome do array que split() irá preencher com as partes que ele cortar. No terceiro elemento, especifique o separador que será usado para cortar as strings. Quando split() retorna, ela retorna o número de elementos string que foram divididos. O split() atribui cada um a um índice de array iniciando em um, de forma que o seguinte código:

```
print mymonth[1],mymonth[numelements]
```

...irá escrever

```
Jan Dec
```

Formas de string especiais

Uma nota rápida -- quando chamar length(), sub(), ou gsub(), você pode omitir o último argumento, e o awk irá aplicar a função a $0 (a linha atual inteira). Para escrever o comprimento de cada linha em um arquivo, use este script awk:

```
{
        print length()
}
```

Diversão financeira

Algumas semanas atrás, eu decidi escrever meu próprio programa de balanço de cheques em awk. Eu decidi que usaria um arquivo texto simples delimitado por tabulações no qual eu informaria meus depósitos e retiradas mais recentes. A idéia era passar estes dados a um script awk que iria acrescentar automaticamente todas as somas e informar o balanço. Aqui está como eu decidi registrar todas minhas transações no meu "ASCII checkbook":

```
23 Aug 2000     food    -       -       Y       Jimmy's Buffet  30.25
```

Cada campo no arquivo é separado por uma ou mais tabulações. Depois da data (campo 1, $1), existem dois campos chamados "categoria de despesa" (expense category) e "categoria de entrada" (income category). Quando estou informando uma despesa conforme a linha acima, eu coloco um apelido de quatro letras no campo exp, e um "-" (entrada em branco) no campo inc. Isto significa que este item particular é um "gasto com alimentação" :) Um depósito irá se parecer com isto:

```
23 Aug 2000     -       inco    -       Y       Boss Man        2001.00
```

Neste caso, eu coloquei um "-" (branco) na categoria exp, e coloquei "inco" na categoria inc. "inco" é meu apelido para entradas genéricas (tipo contracheque). O uso de apelidos para as categorias me permite gerar um detalhamento de gastos e pagamentos por categoria. Sobre o resto dos registros, todos os outros campos são bem auto-explicativos. O campo cleared? ("Y" ou "N") registra se a transação já foi feita na minha conta, depois disso temos uma descrição da transação, e uma quantia positiva de dólares.

O algoritmo usado para calcular o balanço não é muito difícil. O awk simplesmente precisa ler cada linha, uma por uma. Se uma categoria de despesa é listada mas não há uma categoria de entrada (é "-"), então temos um débito. Se uma categoria de entrada é listada, mas nenhuma categoria de despesa ("-"), então a quantia de dólares é um crédito. E, se tanto a categoria de entrada e despesa são listadas, então a quantia é uma "transferência de categoria", ou seja, a quantia de dólares será subtraída da categoria de despesa e acrescentada à categoria de entrada. Novamente, todas estas categorias são virtuais, mas são bastante úteis para controlar entradas e despesas, bem como o orçamento.

O código

É hora de dar uma olhada no código. Começamos com a primeira linha, o bloco BEGIN e uma definição de função:

balance, parte 1

```
#!/usr/bin/env awk -f
BEGIN {
        FS="\t+"
        months="Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec"
}

function monthdigit(mymonth) {
        return (index(months,mymonth)+3)/4
}
```

Acrescentando a primeira linha "#!..." a qualquer script awk irá permitir que o mesmo seja executado diretamente do shell, desde que se faça um "chmod +x myscript" primeiro. As linhas restantes definem nosso bloco BEGIN, que é executadoantes que o awk comece a processar nosso arquivo de cheques. Configuramos FS (o separador de campos) para ser "\t+", o que diz ao awk que os campos serão separados por uma ou mais tabulações. Além disto, definimos uma string chamada months que é usada por nossa função monthdigit(), que aparece em seguida.

As últimas três linhas mostram como definir sua própria função awk. O formato é simples -- escreva "function", e o nome da função, e então os parâmetros, separados por vírgulas, entre parêntesis. Depois disto, um bloco de código "{}" contém o código que você quer que a função execute. Todas funções podem acessar variáveis globais (como nossa variável months). Além disto, o awk tem a declaração "return", que permite que a função retorne um valor, e opera de forma similar ao "return" do C, Python, e outras linguagens. Esta função particular converte um nome de mês em string de três caracteres para o seu equivalente numérico. Por exemplo, este código:

```
print monthdigit("Mar")
```

...irá escrever isto:

```
3
```

Agora é hora de avançar para outras funções.

Funções financeiras

Existem três funções que executam o controle financeiro para nós. Nosso bloco de código principal, que iremos ver logo, processa cada linha do talonário em seqüência, chamando cada uma destas funções de forma que a transação apropriada é registrada em um array awk. Existem três tipos básicos de transação, crédito (doincome), débito (doexpense), e transferência (dotransfer). Você notará que todas as três funções aceitam um argumento, chamado mybalance. mybalance é um "placeholder" para um array de duas dimensões, que iremos passar às funções como um argumetno. Até agora, não havíamos tratado de arrays multi-dimensionais, entretanto, conforme pode ser visto abaixo, a sintaxe é bastante simples. Basta separar cada dimensão com uma vírgula, e está tudo certo.

Iremos armazenar as informações em "mybalance" da seguinte forma: a primeira dimensão do array vai de 0 a 12, e especifica o mês, ou zero para o ano inteiro. Nossa segunda dimensão é uma categoria de três letras, como "foo" ou "inco", e é a categoria que estamos tratando. Assim, para encontrar o balanço total para a categoria food, basta olhar em mybalance[0,"food"]. Para encontrar a entrada de junho, basta olhar em mybalance[6,"inco"].

balance, parte 2

```
function doincome(mybalance) {
        mybalance[curmonth, $3] += amount
        mybalance[0,$3] += amount
}

function doexpense(mybalance) {
        mybalance[curmonth,$2] -= amount
        mybalance[0,$2] -= amount
}

function dotransfer(mybalance) {
        mybalance[0,$2] -= amount
        mybalance[curmonth,$2] -= amount
        mybalance[0,$3] += amount
        mybalance[curmonth,$3] += amount
}
```

Quando doincome() ou qualquer uma das outras funções é chamada, a transação é registrada em dois lugares -- mybalance[0,category] e mybalance[curmonth,category], o balanço da categoria para o ano inteiro e o balanço da categoria para o mês atual, respectivamente. Isto nos permite gerar facilmente um relatório anual ou detalhado por mês de entrada/despesas.

Se você olhar estas funções, irá notar que o array referenciado por mybalance é passado por referência. Além disto, também nos referimos a várias variáveis globais: curmonth, que tem o valor numérico do mês do registro atual, $2 (a categoria de despesa), $3 (a categoria de entradas), e amount ($7, a quantia de dólares). Quando doincome() e as outras funções são chamadas, todas estas variáveis devem ter sido configuradas corretamente para o registro atual (linha), o registro que está sendo processado.

O bloco principal

Aqui está o bloco principal de código que contém o código que trata cada linha de dados de entrada. Lembre-se, como estamos configurando o FS corretamente, podemos nos referir ao primeiro campo como $1, o segundo campo como $2, etc. Quando doincome() e outros são chamados, as funções podem acessar os valores atuais de curmonth, $2, $3 e amount que estão definidos fora das funções. Dê uma olhada no código em acompanhe a explicação que vem depois.

balance, parte 3

```
{
        curmonth=monthdigit(substr($1,4,3))
        amount=$7

        #record all the categories encountered
        if ( $2 != "-" )
                globcat[$2]="yes"
        if ( $3 != "-" )
                globcat[$3]="yes"

        #tally up the transaction properly
        if ( $2 == "-" ) {
                if ( $3 == "-" ) {
                        print "Error: inc and exp fields are both blank!"
                        exit 1
                } else {
                        #this is income
                        doincome(balance)
                        if ( $5 == "Y" )
                                doincome(balance2)
                }
        } else if ( $3 == "-" {
                #this is an expense
                doexpense(balance)
                if ( $5 == "Y" )
                        doexpense(balance2)
        } else {
                #this is a transfer
                dotransfer(balance)
                if ( $5 == "Y" )
                        dotransfer(balance2)
        }
}
```

No bloco principal, as duas primeiras linhas configuram curmonth para um valor inteiro entre 1 e 12, e configuram o valor de amount para o valor do campo 7 (para tornar o código fácil de entender). A seguir, temos cinco linhas interessantes, em que escrevemos os valores em um array chamado globcat. globcat, ou array de categorias globais, é usado para gravar todas as categorias encontradas no arquivo -- "inco", "misc", "food", "util", etc. Por exemplo, se $2 == "inco", fazemos globcat["inco"] ser "yes". Mais tarde, podemos iteragir pela lista de categorias com um simples laço "for (x in globcat)".

Nas próximas vinte linhas, analisamos o campo $2 e $3, e registramos a transação apropriadamente. Se $2=="-" e $3!="-", temos uma entrada, e então chamamos doincome(). Se a situação é inversa, chamamos doexpense(). Se tanto $2 quano $3 contém categorias, chamamos dotransfer(). Cada vez, passamos o array "balance" para estas funções de forma que os dados apropriados são gravados no mesmo.

Você também deve ter percebido várias linhas com "if ($5 == "Y"), registre a mesma transação em *balance2*". O que exatamente estamos fazendo aí? Lembre-se que $5 contém um "Y" ou um "N", e registra se a transação foi efetivada em nossa conta. Como nós registramos a transação em balance2 somente se a transação foi efetivada, balance2 contém o balanço real da conta, enquanto "balance" irá conter todas as transações, efetivadas ou não. Você pode usar balance2 para verificar sua entrada de dados (já que deve ser idêntica ao saldo de sua conta bancária de acordo com seu banco), e usar "balance" para certificar-se que você não irá ultrapassar o saldo de sua conta (uma vez que irá levar em conta quaisquer cheques que tenham sido passados e que não tenham sido descontados).

Gerando o relatório

Depois que o bloco principal tenha processado cada registro de entrada, temos um registro compreensivo dos débitos e créditos divididos por categorias e por mês. Agora, tudo que precisamos é definir um bloco END que gere um relatório, um bloco modesto neste caso:

balance, parte 4

```
END {
        bal=0
        bal2=0
        for (x in globcat) {
                bal=bal+balance[0,x]
                bal2=bal2+balance2[0,x]
        }
        printf("Your available funds: %10.2f\n", bal)
        printf("Your account balance: %10.2f\n", bal2)
}
```

Este trecho escreve um resumo parecido com o seguinte:

```
Your available funds:    1174.22
Your account balance:    2399.33
```

Em nosso bloco END, usamos a construção "for (x in globcat)" para iteragir através de todas as categorias, fazendo um balanço mestre baseado em todas as transações registradas. Na verdade fazemos dois balanços, um para fundos disponíveis, e outro para o balanço da conta. Para executar o programa e processar seus próprios dados financeiros que você tenha inserido em um arquivo chamado "mychecbook.txt", coloque todo o código acima em um arquivo texto chamado "balance", faça "chmod +x balance", e então escreva "./balance mycheckbook.txt". O script de balanço irá então somar todas as transações e escrever um balanço resumido de duas linhas para você.

Atualizações

Eu uso uma versão mais avançada deste programa para administrar minhas próprias finanças pessoais. Minha versão (que eu não posso incluir aqui devido a limitações de espaço) escreve um resumo mensal de entradas e despesas, incluindo totais anuais, resumo de entradas e várias outras coisas. Melhor ainda, ele escreve os relatórios em formato HTML, de forma que eu posso olhar o mesmo no meu browser web :) Se você acha que este programa é útil, eu encorajo você a acrescentar estas funcionalidades ao seu script. Você não precisa configurar o mesmo para *registrar* nenhuma informação adicional, todas as informações que você precisa já estão em balance e balance2. Basta atualizar o bloco END, e você está com tudo!

Espero que você tenha gostado desta série. Para maiores informações sobre o awk, veja a lista de recursos abaixo.

Recursos

- Leia o Awk em exemplos, Parte 1 e Awk em exemplos, Parte 2.
- Se você é do tipo que prefere um livro, o sed & awk, 2nd edition é uma escolha excelente.
- Certifique-se de checar o FAQ do comp.lang.awk. Ele também muitos links adicionais sobre o awk.
- O awk tutorial, de Patric Hartigan, vem com muitos scripts awk úteis.

- O Thompson's TAWK Compiler, compila scripts awk em executáveis binários bem rápidos. Existem versões disponíveis para Windows, OS/2, DOS e UNIX.
- O GNU Awk User's Guide está disponível como referência online.

Sobre o autor

Residindo em Albuquerque, New Mexico, Daniel Robbins é o Presidente/CEO da Gentoo Technologies, Inc., o criador do **Gentoo Linux**, um Linux avançado para o PC, e o sistema **Portage**, a próxima geração de sistema de ports para o Linux. Ele também tem servido como autor para os livros da Macmillan *Caldera OpenLinux Unleashed*, *SuSE Linux Unleashed*, e *Samba Unleashed*. Daniel está envolvido com computadores de alguma forma desde o segundo grau, quando foi exposto pela primeira vez para a linguagem de programação Logo, bem como a uma dose perigosa de Pac Man. Isto provavelmente explica por que ele tem trabalhado como Lead Graphic Artist na **SONY Electronic Publishing/Psygnosis**. Daniel gosta de gastar seu tempo com sua esposa, Mary, e sua nova filhinha, Hadassah. Você pode entrar em contato com Daniel no email drobbins@gentoo.org.