Awk em Exemplos, Parte 2

Registros, laços e arrays

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

Nesta seqüência de seu artigo anterior, introdução ao awk, Daniel Robbins continua a explorar o awk, uma grande linguagem com um estranho nome. Daniel irá mostrar como tratar registros multi-linhas, usar construções em laço, e criar e usar arrays no awk. No fim deste artigo, você estará vem versado em uma ampla gama de funcionalidades do awk, e estará pronto para escrever seus próprios scripts poderosos awk.

Registros Multi-linhas

O awk é uma excelente ferrametna para ler e processar dados estruturados, como o arquivo de sistema /etc/passwd. O arquivo /etc/passwd é o banco de dados de usuários do UNIX, e é um arquivo delimitado por dois-pontos, contendo muita informação importante, incluindo todas as contas de usuário existentes e IDs de usuário, entre outras coisas. Em meu artigo anterior, eu mostrei como o awk pode facilmente tratar este arquivo. Tudo que precisamos fazer foi configurar a variável FS (separador de campos) para ":".

Se a variável FS for configurada corretamente, o awk pode ser configurado para tratar quase qualquer tipo de dado estruturado, desde que exista um registro por linha. Entretanto, só configurar FS não serve para nada se queremos tratar um registro que existe em múltiplas linhas. Nestas situações, precisamos também modificar a variável RS, ou separador de registro. A variável RS informa ao awk quando o registro atual terminou e um novo registro começa.

Por exemplo, vamos ver como tratar a tarefa de processar uma lista de participantes do Programa Federal de Proteção às Testemunhas:

```
Jimmy the Weasel
100 Pleasant Drive
San Francisco, CA 12345

Big Tony
200 Incognito Ave.
Suburbia, WA 67890
```

Idealmente, queremos que o awk reconheça cada endereço de três linhas como sendo um registro individual, em vez de três registros separados. Tornaria nosso código muito mais simples se o awk reconhecer a primeira linha como primeiro campo ($1), o endereço como o segundo campo ($2), e a cidade, estado e CEP como o campo $3. O código abaixo irá fazer exatamente o que queremos:

```
BEGIN {
        FS="\n"
```

```
        RS=""
}
```

No código acima, configurando FS para "\n" diz ao awk que cada campo aparece em sua própria linha. Configurando o RS para "", estamos informando ao awk que cada registro de endereço está separado por uma linha em branco. Uma vez que o awk sabe como a entrada está formatada, ele pode fazer todo o tratamento para nós, e o resto do script é simples. Vamos dar uma olhada em um script completo que irá tratar esta lista de endereços e imprimir cada registro em uma única linha, separando cada campo com uma vírgula.

address.awk

```
BEGIN {
        FS="\n"
        RS=""
}

{
        print $1 ", " $2 ", " $3
}
```

Se este script for salvo como address.awk, e a lista de endereços for armazenada em um arquivo chamado address.txt, você pode executar este script escrevendo "awk -f address.awk address.txt". Este código produz a seguinte saída:

```
Jimmy the Weasel, 100 Pleasant Drive, San Francisco, CA 12345
Big Tony, 200 Incognito Ave., Suburbia, Wa 67890
```

OFS e ORS

Na declaração print do address.awk, você pode ver que o awk concatena (une) as strings que são colocadas próximas em uma linha. Usamos esta funcionalidade para inserir uma vírgula e um espaço (", ") entre os três campos de endereço que aparecem na linha. Apesar deste método funcionar, ele tem um aspecto horrível. Ao invés de inserir ", " literais entre os campos, podemos fazer que o awk faça isto configurando uma variável especial do awk chamada OFS. Dê uma olhada no seguinte trecho de código:

```
print "Hello", "there", "Jim!"
```

As vírgulas nesta linha não são parte da string literal. Em vez disto, elas informam ao awk que "Hello", "there", e "Jim!" são campos separados, e que a variável OFS deve ser impressa entre cada string. Por padrão, o awk produz a seguinte saída:

```
Hello there Jim!
```

Isto nos mostra que, por padrão, o OFS está configurado como " ", um espaço simples. Entretanto, podemos facilmente redefinir o OFS de forma que o awk insira nosso separador de campos favorito. Aqui está uma versão revisada do nosso programa original address.awk que usa o OFS para colocar aquelas strings ", " intermediárias:

Uma versão revisada o address.awk

```
BEGIN {
        FS="\n"
        RS=""
        OFS=", "
}

{
        print $1, $2, $3
}
```

O awk também tem uma variável especial chamada ORS, ou "separador de registros na saída". Configurando o ORS, que é, por padrão, uma nova linha ("\n"), podemos controlar os caracteres que são automaticamente

impressos no fim de uma declaração print. O valor padrão do ORS faz com que o awk imprime cada nova declaração print em uma nova linha. Se quisermos que a saída tenha espaçamento duplo, podemos configurar o ORS para ser "\n\n". Ou, se queremos que os registros sejam separados por um espaço simples (e não uma nova linha), basta configurar o ORS para " ".

Multi-linhas para tabulado

Digamos que tenhamos escrito um script que converte nossa lista de endereços para uma lista com um registro por linha, delimitado por tabulações, para importar para uma planilha. Após usar uma versão levemente modificada do address.awk, torna-se claro que nosso programa funciona somente para endereços de três linhas. Se o awk encontrar o seguinte endereço, a quarta linha será descartada e não será impressa:

```
Cousin Vinnie
Vinnie's Auto Shop
300 City Alley
Sosueme, OR 76543
```

Para tratar situações como esta, seria interessante se noso código pegar o número de registros por campo em conta, imprimindo cada um em ordem. Por enquanto, o código somente imprime os três primeiros campos do endereço. Aqui temos um código que faz o que queremos:

Uma versão do address.awk que funciona com endereços com qualquer número de campos

```
BEGIN {
        FS="\n"
        RS=""
        ORS=""
}

{
        x=1
        while ( x < NF ) {
                print $x "\t"
                x++
        }
        print $NF "\n"
}
```

Primeiro, configuramos o separador de campos FS para "\n" e o separador de registros RS para "", de forma que o awk trata os endereços multi-linhas corretamente, como antes. A seguir, configuramos o separador de registros de saída, ORS, para "", o que vai fazer com que a declaração print *não* coloque um "nova linha" ao fim de cada chamada. Isto significa que se queremos que algum texto inicie em uma nova linha, temos de escrever explicitamente um `print "\n"`.

No código principal, criamos uma variável chamada x que guarda o número do registro atual que estamos processando. Inicialmente este número é 1. A seguir, usamos um laço while (uma estrutura de loop do awk que é idêntica à que é encontrada na linguagem C) para iterar por todos os registros, menos o último, imprimindo o registro e um caracter de tabulação. Finalmente, imprimimos o último registro e uma nova-linha literal. Novamente, uma vez que o ORS está configurado para "", o print não irá acrescentar nova-linhas para nós. A saída do programa é parecida com o que segue, que é exatamente o que queríamos:

Nossa saída pretendida. Não é bonita, mas é delimitada por tabulações para facilitar importação para uma planilha

```
Jimmy the Weasel        100 Pleasant Drive      San Francisco, CA 12345
Big Tony        200 Incognito Ave.      Suburbia, WA 67890
Cousin Vinnie   Vinnie's Auto Shop      300 City Alley  Sosueme, OR 76543
```

Estruturas de laço

Nós já vimos a estrutura de laço while do awk, que é idêntica à sua contraparte no C. O awk também tem uma estrutura de laço "do..while" que testa a condição no fim do bloco de código, em vez de no início, como um laço while padrão. É similar aos laços "repeat..until" que podem ser encontrados em outras linguagens. Veja um exemplo:

Exemplo do...while

```
{
        count=1
        do {
                print "I get printed at least once no matter what"
        } while ( count != 1 )
}
```

Como a condição é testada após o bloco de código, um laço "do...while", diferente de um laço while normal, sempre irá executar o bloco de código pelo menos uma vez. Por outro lado, um laço while nunca será executado se sua condição é falsa quando o laço é encontrado.

Laços for

O awk permite que se crie laços for, que, como os laços while, é idêntico à sua contraparte C:

```
for ( initial assignment; comparison; increment ) {
        code block
}
```

Aqui temos um exemplo:

```
for ( x = 1; x <= 4; x++ ) {
        print "iteration", x
}
```

Este trecho irá escrever:

```
iteration 1
iteration 2
iteration 3
iteration 4
```

Break e Continue

Novamente, exatamente como no C, o awk tem as declarações break e continue. Estas declarações permitem um melhor controle sobre as várias estruturas de laço do awk. Segue um trecho de código que precisa desesperadamente uma declaração break:

Um laço while infinito

```
while (1) {
        print "forever and ever..."
}
```

Como o 1 é sempre verdadeiro, este laço será executado para sempre. Aqui há um laço que é executado somente dez vezes:

Um exemplo da declaração break

```
x=1
while(1) {
        print "iteration",x
```

```
        if ( x == 10 {
                break
        }
        x++
}
```

Aqui, a declaração break é usada para parar o laço mais interno. O break faz com que o laço seja terminado imediatamente e a execução do programa continue na linha após o código do laço.

A declaração continue complementa o break, e funciona assim:

```
x=1
while (1) {
        if ( x == 4 ) {
                x++
                continue
        }
        print "iteration", x
        if ( x > 20 ) {
                break
        }
        x++
}
```

Este código irá imprimir "iteration 1" até "iteration 21", exceto por "iteration 4". Se a iteração é 4, x é incrementado e a declaração continue é chamada, o que faz com que o awk imediatamente inicie o próximo laço sem executar o resto do bloco de código. A declaração continue funciona para todos os tipos de laços iterativos, da mesma forma que o break. Quando usado no corpo de um laço for, o continue faz com que a variável de controle de laço seja automaticamente incrementado. Aqui há um laço for equivalente:

```
for ( x=1; x<=21; x++ ) {
        if ( x == 4 ) {
                continue
        }
        print "iteration", x
}
```

Não foi necessário incrementar o x antes de chamar o continue, como no nosso laço while, já que o laço for incrementa o x automaticamente.

Arrays

Você vai ficar feliz de saber que o awk possui arrays. Entretanto, no awk, é costume iniciar índices de arrays em 1, em vez de 0:

```
myarray[1]="jim"
myarray[2]=456
```

Quando o awk encontra a primeira atribuição, `myarray` é criado e o elemento `myarray[1]` passa a ser "jim". Depois que a segunda atribuição é processada, o array possui dois elementos.

Iterações sobre arrays

Uma vez definido, o awk possui um mecanismo bastante útil para fazer iterações sobre os elementos de um array, como segue abaixo:

```
for ( x in myarray ) {
        print myarray[x]
}
```

Este código irá escrever todos os elementos do array `myarray`. Quando você usa esta forma especial "in" do laço for, o awk irá atribuir todos os índices existentes de `myarray` a `x` (a variável de controle de laço) a cada vez,

executando o bloco de código do laço após cada atribuição. Apesar de ser uma funcionalidade awk bastante útil, ela tem um problema -- quando o awk passa pelos índices do array, ele não segue nenhuma ordem em particular. Isto significa que não há jeito de saber se a saída do código acima será:

```
jim
456
```

ou

```
456
jim
```

Parafraseando Forrest Gump, fazer iterações sobre o conteúdo de um array é como uma caixa de chocolates -- você nunca sabe o que vai obter. Isto tem a ver com o jeito de string dos arrays do awk, que iremos ver agora.

Índices strings de arrays

Em meu artigo anterior, eu mostrei que o awk armazena valores numéricos em um formato string. Apesar do awk executar as conversões necessárias para isto funcionar, isto abre a porta para certos códigos estranhos:

```
a="1"
b="2"
c=a+b+3
```

Depois que o código é executado, o `c` é igual a `6`. Como o awk é "stringy", somar as strings "1" e "2" é funcionalmente igual a somar os números 1 e 2. Em ambos os casos, o awk irá fazer a soma. A natureza "stringy" do awk é bastante intrigante -- você pode se perguntar o que acontece se usarmos índices string para arrays. Por exemplo, veja o seguinte código:

```
myarr["1"]="Mr. Whipple"
print myarr["1"]
```

Como você deve esperar, este código irá escrever "Mr. Whipple". Mas e se tirarmos as aspas que estão em torno do segundo índice "1"?

```
myarr["1"]="Mr. Whipple"
print myarr[1]
```

Adivinhar o resultado deste trecho de código é um pouco mais difícil. O awk irá considerar `myarr["1"]` e `myarr[1]` como sendo dois elementos seprados do array, ou eles se referem ao mesmo elemento? A resposta é que eles se referem ao mesmo elemento, e o awk irá escrever "Mr. Whipple", como no primeiro trecho de código. Apesar disto parecer estranho, por trás das cenas o awk tem usado índices strings para seus arrays todo este tempo"

Depois de aprender este estranho fato, alguns podem se sentir tentados a usar um código esquisito como este:

```
myarr["name"]="Mr Whipple"
print myarr["name"]
```

Não só este código não resulta em erro, mas é funcionalmente idêntico aos nossos exemplos anteriores, e irá escrever "Mr. Whipple" da mesma forma! Como você pode ver, o awk não nos limita a usar índices com inteiros puros, podemos usar índices string se quisermos, sem criar nenhum problema. Onde quer que usemos índices de array não-inteiros, como `myarr["name"]`, estamos usando *arrays associativos*. Tecnicamente, o awk não está fazendo nada diferente nos bastidores quando usamos um índice string (uma vez que mesmo que você usar um índice "inteiro", o awk ainda irá tratar ele como string). Entretanto, você ainda assim deve chamá-los de *arrays associativos* -- soa mais interesssante e irá impressionar seu chefe. O índice stringy será nosso pequeno segredo. :)

Ferramentas para arrays

Quando se trata de arrays, o awk nos dá bastante flexibilidade. Podemos usar índices string, e não somos obrigados a usar seqüências numéricas contínuas de índices (por exemplo, podemos definir `myarr[1]` e `myarr[1000]`, e deixar todos os outros elementos indefinidos). Apesar de isto poder ser bastante útil, em algumas circunstâncias pode causar confusões. Felizmente, o awk oferece um par de funcionalidades para ajudar a tornar os arrays mais gerenciáveis.

Primeiro, podemos excluir elementos de arrays. Se você quer excluir o elemento `1` do seu array `fooarray`, escreva:

```
delete fooarray[1]
```

E, se você quer ver se um determinado elemento do array existe, pode usar o operador booleano especial "in" conforme segue:

```
if ( 1 in fooarray ) {
        print "Ayep!  It's there."
} else {
        print "Nope!  Can't find it."
}
```

Na próxima vez

Nós já cobrimos bastante chão neste artigo. Na próxima vez, eu irei arredondar seu conhecimento awk mostrando como usar as funções matemáticas e de string do awk e como criar suas próprias funções. Também irei percorrer os passos da criação de um programa de balanço de cheques. Até lá, eu encorajo você a escrever alguns programas awk seus, e checar os seguintes recursos.

Recursos

- Leia o Awk em exemplos, Parte 1
- 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.