Awk em Exemplos, Parte 1

Uma introdução à grande linguagem com um nome estranho

Daniel Robbins
Presidente/CEO, Gentoo Technologies, Inc.
Dezembro de 2000

O awk é uma linguagem legal com um nome bastante estranho. Em seu primeiro artigo de uma série de três, Daniel Robbins irá rapidamente acelerar suas habilidades de programação awk. Conforme a série avança, mais tópicos serão cobertos, culminando com uma aplicação demo avançada real.

Em defesa do awk

Nesta série de artigos, irei tornar você um programador awk proficiente. Eu admito, o awk não tem um nome bonito, e a versão GNU do awk, chamada gawk, tem um nome bastante estranho. Aqueles que não são familiares com a linguagem podem ouvir "awk" e pensar em uma mistura de código tão retrógrado e antiquado que é capaz de levar o guru UNIX mais conhecedor à loucura (fazendo com que o mesmo fique gritando "kill -9!" enquanto corre em direção à máquina de café).

Certo, o awk não tem um grande nome. Mas é uma grande linguagem. O awk foi feito para tratar processamento de texto e geração de relatórios, e também muitas funcionalidades bem projetadas que permitem usar o mesmo para programação séria. E, diferente de algumas linguagens, a sintaxe do awk é familiar, e copia algumas das melhores partes de linguagens como o C, python, e o bash (apesar de, tecnicamente, o awk ter sido criado antes tando do python quanto do bash). O awk é uma das linguagens que, uma vez aprendidas, torna-se parte chave do seu arsenal de codificação estratégica.

O primeiro awk

Vamos adiante e começar a brincar com o awk para ver como ele funciona. Na linha de comando, entre com o seguinte comando:

```
$ awk '{ print }' /etc/passwd
```

Você deve ver o conteúdo do seu arquivo /etc/passwd aparecer na tela. Agora, uma explicação do que o awk fez. Quando chamamos o awk, especificamos o arquivo /etc/passwd como arquivo de entrada. Quando executamos o

awk, ele executou o comando print para cada linha no /etc/passwd, em ordem. Toda a saída é enviada a stdout, e conseguimos um resultado idêntico ao de usar o comando cat sobre /etc/passwd.

Agora, para uma explicação do bloco de código `{ print }`. No awk, as chaves são usadas para agrupar blocos de códigos, semelhante ao que acontece no C. Dentro de nosso bloco de código, temos um único comando print. No awk quando um comando print aparece sozinho, o conteúdo completo da linha é impresso.

Aqui temos um outro exemplo do awk que faz exatamente a mesma coisa:

```
$ awk '{ print $0 }' /etc/passwd
```

No awk, a variável `$0` representa a linha corrente completa, assim, `print` e `print $0` fazem exatamente a mesma coisa.

Se você quiser, pode criar um programa que irá apresentar dados sem nenhuma relação aos dados de entrada. Aqui há um exemplo:

```
$ awk '{ print "" }' /etc/passwd
```

Onde você passar a string "" para o comando print, é impressa uma linha em branco. Se você testar este script, irá ver que o awk imprime uma linha em branco para cada linha em seu arquivo /etc/passwd. Novamente, isto é por que o awk executa seu script para cada linha no arquivo de entrada. Aqui há outro exemplo:

```
$ awk '{ print "hiya" }' /etc/passwd
```

Este script irá encher a sua tela de hiya's. :)

Múltiplos campos

O awk é realmente bom em tratar texto que foi dividido em múltiplos campos lógico, e permite que você referencie cada campo individual sem esforços adicionais, a partir do seu script awk. O seguinte script irá imprimir uma lista de todas as contas de usuários no seu sistema:

```
$ awk -F":" '{ print $1 }' /etc/passwd
```

No exemplo acima, quando chamamos o awk, usamos a opção -F para especificar que ":" é um separador de campos. Quando o awk processa o comando `print $1`, ele irá imprimir o primeiro campo que aparece em cada linha do arquivo de entrada. Segue outro exemplo:

```
$ awk -F":" '{ print $1 $3 }' /etc/passwd
```

Aqui está um trecho da saída deste script:

```
halt7
operator11
root0
shutdown6
sync5
bin1
...etc.
```

Como pode ser visto, o awk imprime o primeiro e o terceiro campos do arquivo /etc/passwd, que são o username e uid, respectivamente. Agora, mesmo que o script funcione, ele não é perfeito -- não há nenhum espaço entre os dois campos de saída! Se você está acostumado a programar no bash ou no python, você esperaria que o comando `print $1 $3` colocasse um espaço entre os dois campos. Entretanto, quando duas strings aparecem uma ao lado da outra em um programa awk, o awk concatena as mesmas sem adicionar um espaço intermediário. O seguinte comando irá inserir um espaço entre os dois campos:

```
$ awk -F":" '{ print $1 " " $3 }' /etc/passwd
```

Quando você chamar o print desta forma, ele irá concatenar `$1`, `" "`, e `$3`, criando uma saída legível. Obviamente, podemos também inserir algum texto descritivo se necessário:

```
$ awk -F":" '{ print "username: " $1 "\t\tuid:" $3 }' /etc/passwd
```

Este script irá gerar a seguinte saída:

```
username: halt          uid:7
username: operator      uid:11
username: root          uid:0
username: shutdown      uid:6
username: sync          uid:5
username: bin           uid:1
...etc.
```

Scripts externos

Passar seus scrips para o awk como um argumento de linha de comando pode ser útil para alguns scripts "one-liners", mas quando precisamos de programas complexos, com múltiplas linhas, você definitivamente vai quere compor seus scrips em um arquivo externo. O awk pode ser informado para fazer o source deste script pela opção -f:

```
$ awk -f myscript.awk myfile.in
```

Ao colocar seus scrips em seus próprios arquivost texto você pode aproveitar algumas funcionalidades extras do awk. Por exemplo, este script multi-linhas faz a mesma coisa que um de nossos one-liners anterior, imprimindo o primeiro campo de cada linha no /etc/passwd:

```
BEGIN {
        FS=":"
}

{ print $1 }
```

a diferença entre estes dois métodos tem a ver em como configuramos o separador de campos. Neste script, o separador de campos é espscificado dentro do próprio código (configurando a variável FS), enquanto nossos exemplos prévios configuravam o FS passando a opção -F":" para o awk na linha de comando. Geralmente é melhor configurar o separador de campo dentro do próprio script, simples por que isto significa que você tem um argumento de linha de comando a menos para lembrar de escrever. Iremos cobrir a variável FS com mais detalhes mais adiante, neste artigo.

Os blocos BEGIN e END

Normalmente, o awk executa cada bloco do código do seu script para cada linha de entrada. Entretanto, exsitem várias situações de programação em que você pode querer executar a inicialização do código *antes* que o awk comece o processamento do texto do arquivo de entrada. Para estas situações, o awk permite que você defina um bloco BEGIN. Usamos um bloco BEGIN no exemplo anterior. Como o bloco BEGIN é executando antes que o awk comece a processar o arquivo de entrada, é um excelente lugar para inicializar a variável FS (field separator - separador de campo), escrever um cabeçalho, ou inicializar outras variáveis globais às quais você irá se referir mais tarde no programa.

O awk também fornece outro bloco especial, chamado bloco END. O awk executa este bloco depois que todas as linhas no arquivo de entrada tenham sido processadas. Tipicamente, o bloco END é usado para executar cálculos finais ou imprimir resumos que devem aparecer no fim dos dados da saída.

Expressões regulares e blocos

O awk permite que se use expressões regulares para executar seletivamente um bloco individual de código, dependendo se a expressão regular combina ou não com a linha atual. Segue um script de exemplo que imprime

somente as linhas que contém a seqüência de caracteres `foo`:

```
/foo/ { print }
```

Obviamente, você pode usar expressões regulares mais complicadas. Veja um exemplo que somente imprime linhas que contenham números de ponto flutuante:

```
/[0-9]+\.[0-9]*/ { print }
```

Expressões e blocos

Existem muitas outras formas de executar seletivamente um bloco de código. Podemos colocar qualquer tipo de expressão booleana antes de um bloco de código para controlar quando um bloco em particular será executado. O awk irá executar um bloco de código somente se a expressão booleana que o precede resultar em verdadeiro. O exemplo seguinte irá imprimir o terceiro campo de todas as linhas que tem um primeiro campo igual a `fred`. Se o primeiro campo da linha atual não for igual a `fred`, o awk irá continuar processando o arquivo e não irá executar a declaração `print` para a linha corrente:

```
$1 == "fred" { print $3 }
```

O awk oferece uma seleção completa de operadores de comparação, incluindo os usuis "==", "<", ">", "<=", ">=", e "!=". Além disto, o awk ainda oferece os operadores "~" e "!~", que significam "combina" e "não combina". Eles são usados especificando uma variável no lado esquerdo do operador, e uma expressão regular no lado direito. Veja um exmeplo que irá imprimir somente o terceiro campo da linha se o quinto campo da mesma linha contém a seqüência de caracteres `root`:

```
$5 ~ /root/ { print $3 }
```

Declarações condicionais

O awk também oferece uma declaração `if` legal, parecida com a do C. Se você quisesse, poderia reescrever o script anterior usando uma declaração `if`:

```
{
        if ( $5 ~ /root/ ) {
                print $3
        }
}
```

Ambos os scripts funcionam de forma idêntica. No primeiro exemplo, a expressão booleana é colocada fora do bloco, enquanto no segundo exemplo, o bloco é executado para cada linha de entrada, e seletivamente executamos o comando print usando uma declaração `if`. Os dois métodos estão disponíveis, e você pode escolher o que melhor se adapta às outras partes do seu script.

Aqui há um exemplo um pouco mais complicado de uma declaração `if` do awk. Como você pode ver, mesmo com testes condicionais aninhados e complexos, as declarações `if` são quase idênticas às contrapartes no C:

```
{
        if ( $1 == "foo" ) {
                if ( $2 == "foo" ) {
                        print "uno"
                } else {
                        print "one"
                }
        } else if ( $1 == "bar" ) {
                print "two"
        } else {
                print "three"
        }
}
```

Usando declarações `if`, podemos transformar este código:

```
! /matchme/ { print $1 $2 $3 }
```

neste:

```
{
        if ( $0 ! ~ /matchme/ ) {
                print $1 $2 $3
        }
}
```

Ambos scripts irão escrever somente as linhas que *não* contém a seqüência de caracters `matchme`. Novamente, você pode escolher o método que funciona melhor no seu código. Os dois fazem a mesma coisa.

O awk também permtie o uso dos operadoers booleanos "||" (para o "ou lógico"), e "&&" (para o "e lógico"), para permitir a criação de expressões booleanas mais complexas:

```
( $1 == "foo" ) && ( $2 == "bar" ) { print }
```

Este exemplo irá imprimir somente as linhas onde o campo um é igual a `foo` *e* o campo dois é igual a `bar`.

Variáveis numéricas

Até agora, somente imprimimos strings, a linha inteira, ou campos específicos. Entretanto, o awk também nos permite executar cálculos tanto de inteiros quanto de ponto flutuante. Usando expressões matemáticas, é muito fácil escrever um script que conte o número de linhas em branco de um arquivo. Aqui tem um:

```
BEGIN   { x=0 }
/^$/    { x=x+1 }
END     { print "I found " x " blank lines. :)" }
```

No bloco BEGIN, nós inicializamos nossa variável inteira para zero. A seguir, cada vez que o awk encontra uma linha em branco, ele irá executar a declaração `x=x+1`, incrementando o x. Após todas as linhas terem sido processadas, o bloco END será executado, e o awk irá escrever um resumo final, especificando o número de linhas em branco que ele encontrou.

Variáveis String

Uma das coisas legais sobre as variáveis do awk é que elas são simples e são *stringy*. Eu considero as variáveis awk como "stringy" por que todas são representadas internamente como strings. Ao mesmo tempo, as variáveis são "simples" por que você pode executar operações matemáticas em uma variável, desde que ela contenha uma string de um número válido, o awk irá tomar as providências para a conversão de string para número. Para ver o que quero dizer, dê uma olhada no seguinte exemplo:

```
x="1.01"
# We just set x to contain the *string* "1.01"
x=x+1
# We just added one to a *string*
print x
# Incidentally, these are comments :)
```

A saída o awk será:

```
2.01
```

Interessante! Apesar de termos atribuído o valor string `1.01` para a variável `x`, ainda assim conseguimos acrescentar um a ela. Não conseguiríamos fazer isto no bash ou no python. A princípio, o bash não suporta aritmética de ponto flutuante. E, enquanto o bash possue variáveis "stringy", elas não são "simples": para executar qualquer operação matemática, o bash exige que o cálculo esteja entre os horríveis `$( )`. Se estivermos usando python, teríamos que

converter explicitamente nossa string `1.01` para um valor de ponto flutuante antes de executar qualquer cálculo sobre ele. Mesmo que isto não seja difícil, é um passo adicional. Com o awk, isto é automático, e é o que torna nosso código bonito e claro. Se quizermos elevar ao quadrado e acrescentar um ao primeiro campo de cada linha de entrada, poderíamos usar o seguinte script:

```
{ print ($1^2)+1 }
```

Se você fizer algumas experiências, irá descobrir que se uma variável em particular não contém um número válido, o awk irá tratar a mesma como sendo zero quando estiver calculando nossa expressão matemática.

Muitos operadores

Outra coisa legal no awk é seu complemento completo de operadores matemáticos. Além dos operadores padrão de adição, subtração, multiplicação e divisão, o awk permite que se use o operador de exponenciação "^", previamente demonstrado, o operador de módulo (resto de divisão) "%", e um punhado de outros operadoes de atribuição copiados do C.

Estes incluem o pré e pós incremento/decremento (`t++`, `--foo`), atribuição com soma/subração/multiplicação/divisão (`a+=3`, `b*=2`, `c/=2.2`, `d-=6.2`). Mas isto não é tudo -- ainda temos o operação atribuição conbinado ao módulo e exponenciação também (`a^=2`, `b%=4`).

Separadores de campos

O awk também possui seu próprio complemento de variáveis especiais. Algumas delas permitem que se "afine" como o awk funciona, enquanto outras podem ser lidas para dar informações úteis sobre a entrada. Já tocamos uma destas variáveis especiais, FS. Conforme mencionado anteriormente, esta variável permite que se configure a seqüência de caracteres que o awk espera encontrar entre os campos. Quando estávamos usando /etc/passwk como entrada, configuramos o FS para ":". O FS permite este truque, mas também permite mais flexibilidade.

O valor de FS não está limitado a um único caracter, ele também pode ser configurado para ser uma expressão regular, especificando um padrão de caracteres de qualquer tamanho. Se você está processadno campos separados por uma ou mais tabulações, você vai querer configurar o FS assim:

```
FS="\t+"
```

Acima, nós usamos o caracter de expressão regular especial "+", que significa "um ou mais do caracter anterior".

Se seus campos são separados por espaços em brancos (um ou mais espaços ou tabulações), você pode querer configurar o FS para a seguinte expressão regular:

```
FS="[[:space:]+]"
```

Esta atribuição irá fazer o serviço, mas é desnecessária. Por quê? Por que, por padrão, o FS é configurado para um caracter de espaço, que o awk interpreta como "um ou mais espaços ou tabulações'. Neste exemplo em particular, a configuração padrão do FS é exatamente o que você quer!

Expressões regulares complexas não são problema. Mesmo se seus registros são separados pela palavra "foo", seguida de três dígitos, a seguinte expressão regular irá permitir que seus dados sejam tratados apropriadamente:

```
FS="foo[0-9][0-9][0-9]"
```

Número de campos

As próximas duas variáveis que iremos tratar normalmente não são escritas, mas lidas e usadas para obter informações úteis sobre a entrada. A primeira é a variável NF, também chamada de variável "número de campos". O awk irá configurar automaticamente esta variável com o número de campos no registro atual. Você pode usar a variável NF para apresentar somente certas linhas de entrada:

```
NF == 3 { print "this particular record has three fields: " $0 }
```

Obviamente, você também pode usar a variável NF em declarações condicionais, como segue:

```
{
        if ( NF > 2 ) {
                print $1 " " $2 ":" $3
        }
}
```

Número de registro

O número de registro (NR) é outra variável útil. Ela sempre irá conter o número do registro atual (o awk conta o primeiro registro como registro 1). Até agora, nós tratamos com arquivos de entrada que continham um registro por linha. Para estas situações, o NR também irá informar o número da linha atual. Entretanto, quando iniciarmos a processar registros multi-linhas mais adiante nesta série, este não será mais o caso, portanto seja cuidadoso! O NR pode ser usado como a variável NR para imprimir somente certas linhas da entrada:

```
(NR < 10 ) || (NR > 100) { print "We are on record number 1-9 or 101+" }
```

Outro exemplo:

```
{
        # skip header
        if ( NR > 10 ) {
                print "ok, now for the real information!"
        }
}
```

O awk fornece variáveis adicionais que podem ser usadas para vários objetivos. Iremos cobrir mais destas variáveis nos próximos artigos.

Estamos terminando nossa exploração inicial do awk. Conforme a série prosseguir, irei demonstrar mais funcionalidades avançadas do awk, e iremos terminar a série com uma aplicação do mundo real. Enquanto isto, se você estiver com vontade de aprender mais, verifique os recursos listados abaixo.

Recursos

- 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.