Awk

Característica	Como fazer
Busca	Operador ~, Função `match`
Substituição	Funções `sub`, `gensub`
Divisão	Função `split`
ER crua	/entre barras/
Ignore M/m	Variável `IGNORECASE`
Global	Função `gsub`, modificador `g`

Awk é uma linguagem antiga (1977) que combina processamento de texto com estruturas de uma linguagem genérica, possuindo condicionais, operações aritméticas e afins.

Além do Awk clássico do Unix, também existe o GNU Awk, conhecido como gawk, que traz algumas funcionalidades novas e um melhor suporte às expressões regulares. O gawk é amplamente utilizado em sistemas Linux.

Grande parte dos exemplos deste tópico funcionará em ambas versões. Caso contrário, será feita uma menção sobre quais são as características exclusivas do GNU Awk.

As expressões regulares são parte integrante da linguagem. Basta colocar uma expressão entre barras que o Awk saberá que aquilo não é uma string, e sim uma expressão regular com metacaracteres. Se sua expressão possuir uma barra /, lembre-se de escapá-la \/ para evitar problemas.

O operador til ~ é usado para fazer comparações de strings com expressões regulares. Para uma comparação inversa, em que o teste retornará sucesso se a expressão não casar com a string, use o operador `!~`.

```
if ("Awk" ~ /^A/) {
   print "Casou"
}
if ("Awk" !~ /^X/) {
   print "Não casou"
}
```

Para fazer testes ignorando a diferença entre maiúsculas e minúsculas, antes mude para 1 o valor da variável `IGNORECASE`, que funciona como uma chave que liga e desliga esse comportamento. Mude o valor da variável para zero quando quiser retornar à comparação normal.

```
IGNORECASE = 1
if ("Awk" ~ /^a/) {
   print "Casou"
}
```

O Awk do Unix não reconhece essa variável, então a única maneira de simular esse comportamento é usando a criatividade: converta a string para minúsculas ao fazer o teste e use somente minúsculas em sua expressão.

```
if (tolower("Awk") ~ /^a/) {
   print "Casou"
}
```

Outra maneira de fazer testes é utilizar uma expressão regular para casar linhas específicas de um arquivo. Quando o Awk está processando um arquivo de texto, ele o lê linha a linha. A linha da vez é guardada na variável especial `$0`.

Ao colocar uma expressão regular diretamente como um teste, ela será testada na linha atual (como se fosse `$0 ~ /er/`) e os comandos do bloco seguinte só serão executados nas linhas que casarem com o padrão.

```
/[0-9]/ {
   print "Linha com números:", $0
}
```

Dessa maneira, fica fácil testar expressões regulares na linha de comando, seja com o conteúdo de um arquivo, seja com um texto vindo da entrada padrão (STDIN).

```
prompt$ echo Awk | awk '/^A/ { print "Casou" }'
Casou
```

Outra maneira de casar um texto é utilizar a função `match`. A vantagem é que a cada vez que é usada, a função define as variáveis `RSTART` e `RLENGTH`, que guardam o ponto de início (índice) e o tamanho do trecho casado pela expressão.

No GNU Awk, é possível informar um array como terceiro parâmetro para a função `match`. Nesse caso, a primeira posição (índice zero) desse array será preenchida com o trecho casado pela expressão, e as posições seguintes guardarão o conteúdo de cada grupo.

```
match("Awk", /(.)(.)(.)/, resultado)
print resultado[0]   # Awk
print resultado[1]   # A
print resultado[2]   # w
print resultado[3]   # k
print RSTART         # 1
print RLENGTH        # 3
```

A substituição é feita tradicionalmente pela função `sub`, que troca apenas a primeira ocorrência do padrão. Sua irmã `gsub` encarrega-se de fazer a substituição de todas as ocorrências (global). O detalhe é que o texto alterado é gravado na própria variável que continha o texto original, em vez de ser retornado pela função. Com isso, não é possível usar strings diretamente, sendo sempre necessário o uso de uma variável.

```
texto = "Awk";
sub(/[A-Za-z]/, ".", texto)
print texto                  # .wk

texto = "Awk";
gsub(/[A-Za-z]/, ".", texto)
print texto                  # ...
```

Se a variável com o texto não for informada à função, é utilizada a variável especial `$0`, que contém a linha atual do arquivo processado (ou entrada padrão). Assim o uso dessas funções torna-se mais ágil.

```
prompt$ echo Awk | awk 'sub(/[A-Za-z]/, ".")'
.wk

prompt$ echo Awk | awk 'gsub(/[A-Za-z]/, ".")'
...

prompt$ echo Awk | awk 'sub(/.*/, "&&&")'
AwkAwkAwk
```

Note que ao usar o caractere `&` no segundo argumento da função, ele é expandido para o trecho casado pela expressão. Mas este é o único caractere considerado especial no texto substituto, pois as funções `sub` e `gsub` não têm suporte aos retrovisores.

O GNU Awk criou uma função nova chamada `gensub`, que, além do suporte aos retrovisores, também traz outras novidades que visam a contornar as limitações das outras funções. Uma de suas vantagens é que o texto alterado é retornado pela função em vez de ser gravado em uma variável. Há também um terceiro argumento numérico que indica qual das ocorrências deve ser substituída. Se esse argumento for a letra `g`, todas as ocorrências serão substituídas.

```
print gensub(/\w/, ".", "1", "Awk")   # .wk
print gensub(/\w/, ".", "2", "Awk")   # A.k
print gensub(/\w/, ".", "3", "Awk")   # Aw.
print gensub(/\w/, ".", "g", "Awk")   # ...
```

Os retrovisores são indicados por `\1`, `\2` e amigos. Mas como são parte de uma string (entre aspas), devem ser escapados para funcionar: `\\1`, `\\2`, ... O retrovisor `\0` guarda todo o trecho casado pela expressão.

```
print gensub(/(.)(.)(.)/, "\\3\\2\\1", "g", "Awk")   # kwA
print gensub(/.*/       , "--\\0--"  , "g", "Awk")   # --Awk--
```

A mesma dica de uso da variável `IGNORECASE` também vale para a `gensub` ignorar a diferença entre maiúsculas e minúsculas. Ligue e desligue essa variável com 1 e 0 conforme precisar dela.

```
IGNORECASE = 0
print gensub(/[a-z]/, ".", "g", "Awk")   # A..

IGNORECASE = 1
print gensub(/[a-z]/, ".", "g", "Awk")   # ...
```

A acentuação não é problema, desde que seu sistema esteja configurado corretamente para o português. Confira o valor das variáveis de ambiente `$LANG` e `$LC_ALL`. Use as classes POSIX para casar os caracteres acentuados. No gawk, você também pode usar o barra-letra `\w`.

```
print gensub(/[a-z]/      , ".", "g", "adábuká")   # ..á...á
print gensub(/[[:alpha:]]/, ".", "g", "adábuká")   # .......
print gensub(/\w/         , ".", "g", "adábuká")   # .......
```

Use a função `split` para fazer a divisão de uma string. Seu segundo argumento é o array onde a string dividida será guardada e o terceiro é a expressão regular. Note que o índice inicial é 1 e não 0.

```
split("A w k", resultado, /[ \t]+/)
print resultado[1]   # A
print resultado[2]   # w
print resultado[3]   # k
```

Usuários avançados de Awk gostarão de saber que as variáveis especiais `RS` (separador de registros) e `FS` (separador de campos) também podem ser definidas com uma expressão regular. Apenas se lembre de que como sua definição é feita por meio de uma string, é preciso escapar as contrabarras `\`.

```
prompt$ echo "A -- w -- k" | awk -F '[- ]+' '{ print $2 }'
w
```

Por fim, se você estiver utilizando o GNU Awk e quiser testar a compatibilidade de seus scripts com o Awk original do Unix ou outras versões, veja as opções `--posix`, `--traditional` e `--re-interval`.