Tópicos Especiais em Fundamentos da Computação
Shell Scripting

AWK

Andrei Rimsa Álvares
andrei@cefetmg.br

Sumário

- Introdução
- A linguagem AWK
- Exemplos

INTRODUÇÃO

Introdução

- AWK é uma **linguagem** de processamento e busca de padrões que procura em um ou mais arquivos por registros (normalmente linhas) que casam com um determinado padrão

- AWK processa linhas através de **ações**, como escrever o registro na saída padrão ou incrementar um contador cada vez que encontra um casamento

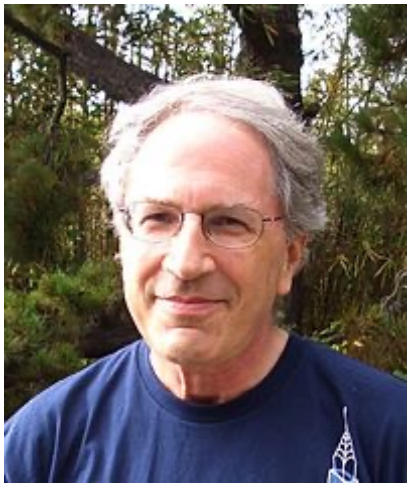
- Diferentemente de linguagens procedurais, AWK é **orientado a dados**: descreve-se dados que se deseja processar e o que deve-se ser feito quando encontrá-lo

- Pode-se usar AWK para gerar relatórios ou filtrar entradas

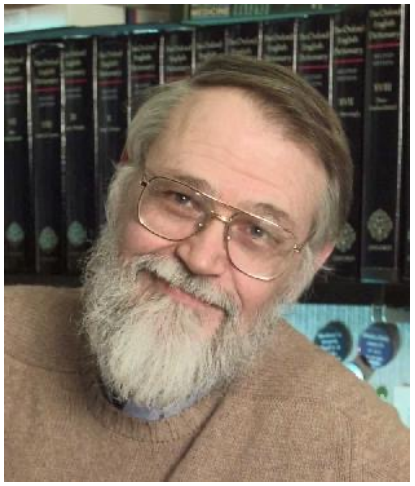
Introdução

- A linguagem AWK foi projetada por três autores

Alfred V. Aho♤



Peter J. Weinberger♡



Brian W. Kernighan♧

  - Eles conceberam a linguagem para ser fácil de usar sacrificando a velocidade de execução na versão original

♤: https://en.wikipedia.org/wiki/Alfred_Aho
♡: http://en.wikipedia.org/wiki/Peter_J._Weinberger
♧: http://en.wikipedia.org/wiki/Brian_Kernighan

Introdução

- AWK se baseia em construções da linguagem C, como
  - Formato flexível
  - Execução condicional
  - Comandos de repetição
  - Variáveis numéricas e strings
  - Expressões regulares
  - Expressões relacionais
  - `printf` de C

A LINGUAGEM AWK

Shell Scripting

Sintaxe

- O comando **gawk** possui a seguinte sintaxe

```
gawk [options] [program] [file-list]
gawk [options] -f [program-file] [file-list]
```

  - A ferramenta recebe sua entrada de uma lista de arquivos (*file-list*) ou da entrada padrão
  - A saída da ferramenta vai para a saída padrão

Será usada a versão GNU da ferramenta awk: gawk

Argumentos

- Na sintaxe anterior, *program* é um programa em gawk que se inclui na linha de comando
  - Isso permite escrever pequenos programas AWK sem ter que criar um arquivo separado para isso
  - Para evitar que a shell interprete caracteres especiais do programa, coloque os comandos AWK entre aspas simples

- *program-file* é o nome do arquivo que guarda um programa escrito em AWK
  - Colocar um programa grande e/ou complexo em um arquivo pode reduzir erros e problemas de redigitação

Argumentos

- *`file-list`* contém os caminhos para arquivos ordinários que AWK processa (arquivos de entrada)

- Se não especificar nenhum arquivo, AWK recebe a entrada da entrada padrão

Opções

- Algumas opções em linha de comando

Opção	Descrição
`-F fs`	Usar **`fs`** como separador de campos (variável FS)
`-f program-file`	Ler o programa em AWK do arquivo **`program-file`** ao invés da linha de comando
`-v var=value`	Atribuir o valor **value** à variável **var** antes de executar o programa (pode-se especificar essa opção várias vezes)

Fundamentos Básicos

- Um programa em AWK consiste em uma ou mais linhas contendo um **padrão** (`pattern`) e/ou **ação** (`action`) no seguinte formato

```
pattern { action }
```

  - O **padrão** seleciona linhas da entrada, enquanto AWK executa a **ação** em todas as linhas selecionadas pelo padrão
    - Se vários padrões selecionarem uma mesma linha, AWK executa as ações associadas à cada padrão na ordem que aparecem no programa
    - É possível que AWK envie uma única linha da entrada mais de uma vez para a saída padrão
  - Se uma linha de programa não tiver um **padrão**, AWK seleciona todas as linhas da entrada
  - Se uma linha de programa não tiver uma **ação**, AWK copia a linha selecionada para a saída padrão

Padrões

- Um padrão é uma expressão regular (entre barras)
  - O operador ~ testa se um determinado campo ou variável casa com uma expressão regular, enquanto o operador !~ se não casa
  - Pode-se fazer comparações numéricas e textuais usando os seguintes operadores relacionais

Operador	Significado
<	Menor que
<=	Menor ou igual a
==	Igual a

Operador	Significado
!=	Não igual a
>=	Maior que ou igual a
>	Maior que

  - Pode-se combinar qualquer padrão usando operadores lógicos: || (OR) ou && (AND)

Padrões Especiais

- Dois padrões únicos
  - **`BEGIN`**: executa as ações **antes** do processamento da entrada
  - **`END`**: executa as ações **após** o processamento da entrada

- O operador **`,`** (vírgula) é o operador de faixa
  - Se separar dois padrões com uma vírgula em um única linha de programa AWK, seleciona-se um grupo de linhas começando com a primeira linha que casa o primeiro padrão enquanto a última linha é a próxima linha subsequente casada pelo segundo padrão
    - Após casar o segundo padrão, AWK continua o processamento buscando o primeiro padrão novamente
  - Se não casar o segundo padrão, AWK seleciona todas as linhas até o final da entrada

Ações

- A ação de um comando AWK causa a execução dessa ação quando o padrão é casado
  - Se não especificar uma ação, AWK executa a ação padrão que é o comando **`print`** (copia o registro da entrada para a saída padrão)
  - Quando o comando **`print`** possui argumentos (variáveis ou constantes strings), AWK apenas os mostra
  - Pode-se usar a saída do comando **`print`** para enviar para um arquivo (>), anexar (>>) ou para outro programa (|)
  - Itens separados com vírgulas em um comando **`print`** são separados com o separador de campos de saída (OFS); sem vírgulas os itens são concatenados
- Pode-se incluir várias ações em uma linha separando-as por ponto e vírgula

Comentários

- AWK usa o símbolo # como comentários
  - Desconsidera qualquer coisa em um linha de programa seguida desse símbolo
  - Pode-se usá-lo para documentar o código

Variáveis

- Embora não se precise declarar variáveis antes de usá-las, pode-se atribuir valores iniciais para elas
  - Variáveis inteiras não atribuídas previamente são iniciadas com **`0`**
  - Strings não atribuídas previamente são iniciadas com **`null`**

Variáveis

- Além de suportar variáveis de usuário, AWK mantém variáveis de programa que podem ser usadas tanto nos **padrões** quanto nas **ações**

Variável	Significado
`$0`	O registro atual
`$1-$n`	Campos no registro atual
`FILENAME`	Nome do arquivo de entrada atual
`FS`	Separador de campo de entrada (padrão: espaço ou tab)
`RS`	Separador de registro de entrada (padrão: nova linha)
`NF`	Número de campos no registro atual
`NR`	Número de registro do registro atual
`OFS`	Separador de campos de saída (padrão: espaço)
`ORS`	Separador de registro de saída (padrão: nova linha)

Funções

- Algumas funções internas para manipulação de números e strings

Função	Significado
`length(`*`str`*`)`	Retorna o número de caracteres da string *str*; se não usar parâmetros retorna o número de caracteres do registro atual
`int(`*`num`*`)`	Retorna a parte inteira de *num*
`index(`*`str1`*`, `*`str2`*`)`	Retorna o índice da string *str1* na string *str2*; retorna 0 se não existir em *str2*
`split(`*`str`*`, `*`arr`*`, `*`del`*`)`	Coloca pedaços da string *str* delimitados por *del*, no arranjo *arr*[1]..*arr*[n]; retorna a quantidade de elementos
`printf(`*`fmt`*`, `*`args`*`)`	Formata os argumentos args de acordo com o formatador fmt (imita a função de mesmo nome em C)
`substr(`*`str`*`, `*`pos`*`, `*`len`*`)`	Retorna a substring de *str* que começa na posição *pos* com tamanho *len*
`tolower(`*`str`*`)`	Retorna uma cópia de str com todas as letras minúsculas
`toupper(`*`str`*`)`	Retorna uma cópia de str com todas as letras maiúsculas

Operadores Aritméticos

Operador	Significado
**	Eleva a expressão precedente ao operador à potência da expressão seguinte
*	Multiplica a expressão precedente ao operador pela expressão seguinte
/	Divide a expressão precedente ao operador pela expressão seguinte
%	Obtém o resto da divisão da expressão precedente ao operador pela seguinte
+	Adiciona a expressão precedente ao operador pela expressão seguinte
-	Subtrai a expressão precedente ao operador pela expressão seguinte
=	Atribuir o valor da expressão seguinte a variável que precede o operador
++	Incrementa a variável precedente ao operador
--	Decrementa a variável precedente ao operador
+=	Adiciona e atribui a expressão seguinte a variável que precede o operador
-=	Subtrai e atribui a expressão seguinte a variável que precede o operador
*=	Multiplica e atribui a expressão seguinte a variável que precede o operador
/=	Divide e atribui a expressão seguinte a variável que precede o operador
%=	Obtém o resto da divisão e atribui a expressão seguinte a variável que precede o operador

Arranjos Associativos

- Os arranjos associativos de AWK são uma de suas mais poderosas funcionalidades (em perl são chamados de **`hash`**)
  - Os arranjos são indexados por strings, mas pode-se usar strings numéricas como índices

- A sintaxe para atribuir um valor a um elemento num arranjo associativo

```
array[string] = value
```

  - tal que **`array`** é o nome do arranjo, **`string`** é o índice do elemento do arranjo e **`value`** o valor que está associando ao elemento

Arranjos Associativos

- Usando a seguinte sintaxe, pode-se navegar em um arranjo associativo com o comando for

```
for (elem in array) action
```

  - tal que **`elem`** é uma variável que recebe o valor de cada elemento do arranjo a medida que o comando for navega na coleção, **`array`** é o nome do arranjo e **`action`** é a ação que AWK irá executar para cada elemento do arranjo

printf

- AWK suporta o comando **`printf`** no lugar de **`print`** para controlar o formato da saída gerada (similar a encontrada em C)

- O comando **`printf`** possui a seguinte sintaxe

```
printf "control-string", arg1, arg2, ..., argn
```

  – tal que **`control-string`** determina como **`printf`** irá formatar os argumentos (**`arg1`** a **`argn`**); os argumentos podem ser variáveis ou outras expressões

Dica: pode-se usar no control-string \n para nova linha e \t para tabulação

printf

- O argumento control-string possui especificações de conversões, em que elas possuem a seguinte sintaxe

```
%[-][x[.y]]conv
```

  – tal que – faz com que **printf** alinhe a esquerda o argumento, **x** é a largura mínima do campo, **.y** é o número de casas a direita de um número decimal; **conv** indica o tipo de conversão numérica e pode ser qualquer letra da tabela abaixo

conv	Tipo de conversão
d	Decimal
e	Notação exponencial
f	Número ponto-flutuante
g	Usar **f** ou **e**, qual for menor

conv	Tipo de conversão
o	Octal sem sinal
s	String de caracteres
x	Hexadecimal sem sinal

Estruturas de Controle

- AWK suporta as seguintes estruturas de controle
  - Fluxo condicional: **`if...else`**
  - Fluxo de repetição: **`while`** e **`for`**
    - Com os comandos adicionais **`break`** e **`continue`** para alterar o fluxo de execução

Dica: não precisa usar chaves em volta de comandos quando especificar um único comando simples

if...else

- A estrutura de controle condicional **`if...else`** testa o estado obtido pela condição *`condition`* e transfere o controle baseado nisso

- A sintaxe da estrutura é dada a seguir, em que a parte **`else`** é opcional

```
if (condition)
    {commands}
[else {commands}]
```

- Exemplo

if1

```
BEGIN {
    nam="sam"
    if (nam == "max")
        print "nam is max"
    else
        print "nam is not max, it is", nam
}
```

```
$ gawk -f if1
nam is not max, it is sam
```

while

- A estrutura de controle de repetição **while** itera e executa os comando enquanto a condição *condition* for verdadeira

- A sintaxe do comando **while** é dada a seguir

```
while (condition)
    {commands}
```

- Exemplo

while1

```
BEGIN {
    n=1
    while (n <= 5)
        {
        print "2^" n, 2**n
        n++
        }
}
```

```
$ gawk -f while1
2^1 2
2^2 4
2^3 8
2^4 16
2^5 32
```

for

- A estrutura de controle de repetição **for** começa executando o comando *init*, então itera sobre os comandos enquanto a condição *condition* for verdadeira e após cada iteração executa o comando *increment*

- A sintaxe do comando **for** é dada a seguir

```
for (init; condition; increment)
    {commands}
```

- Exemplo

for1

```
BEGIN {
    for (n=1; n <= 5; n++)
        print "2^" n, 2**n
}
```

```
$ gawk -f for1
2^1 2
2^2 4
2^3 8
2^4 16
2^5 32
```

EXEMPLOS

Space
Scrn
Scroll Lock
Pause
rs112054560
examples
Insert
Home
MediaFocus

Arquivo de Entrada

- Considere o seguinte arquivo com informações sobre carros, em que as colunas contêm na ordem: fabricante, modelo, ano de fabricação, distância percorrida (em mil milhas) e preço separados por tabulação

```
$ cat > cars << EOF
> plym    fury      1970   73     2500
> chevy   malibu    1999   60     3000
> ford    mustang   1965   45     10000
> volvo   s80       1998   102    9850
> ford    thundbd   2003   15     10500
> chevy   malibu    2000   50     3500
> bmw     325i      1985   115    450
> honda   accord    2001   30     6000
> ford    taurus    2004   10     17000
> toyota  rav4      2002   180    750
> chevy   impala    1985   85     1550
> ford    explor    2003   25     9500
> EOF
```

Sem Padrão

- O programa a seguir possui apenas a ação **`print`**
  - Como não tem padrão, AWK seleciona todas as linhas
  - Quando usado sem argumentos, **`print`** mostra todas as linhas por completo

```
$ gawk '{ print }' cars
plym    fury    1970    73      2500
chevy   malibu  1999    60      3000
ford    mustang 1965    45      10000
volvo   s80     1998    102     9850
ford    thundbd 2003    15      10500
chevy   malibu  2000    50      3500
bmw     325i    1985    115     450
honda   accord  2001    30      6000
ford    taurus  2004    10      17000
toyota  rav4    2002    180     750
chevy   impala  1985    85      1550
ford    explor  2003    25      9500
```

Sem Ação

- O programa a seguir possui um padrão, mas nenhuma ação explícita (as barras indicam que o padrão é uma expressão regular)
  - AWK seleciona apenas as linhas que contém a string chevy
  - Quando não se especifica uma ação, usa-se por padrão **`print`**

```
$ gawk '/chevy/' cars
chevy   malibu  1999   60  3000
chevy   malibu  2000   50  3500
chevy   impala  1985   85  1550
```

Campos

- Esse exemplo seleciona todas as linhas (não possui padrão), e para cada uma delas exibe o terceiro campo ($3) + um espaço (OFS indicado pela vírgula) + o primeiro campo ($1)

```
$ gawk '{print $3, $1}' cars
1970 plym
1999 chevy
1965 ford
1998 volvo
2003 ford
...
```

As chaves são usadas para diferenciar o padrão das ações

- Outro exemplo usando um padrão com a mesma ação

```
$ gawk '/chevy/ {print $3, $1}' cars
1999 chevy
2000 chevy
1985 chevy
```

Operador de Casamento

- Esse padrão usa o operador de casamento (~) para selecionar todas as linhas que contém a letra h no primeiro campo

```
$ gawk '$1 ~ /h/' cars
chevy   malibu  1999   60  3000
chevy   malibu  2000   50  3500
honda   accord  2001   30  6000
chevy   impala  1985   85  1550
```

- Um exemplo mais complexo que seleciona apenas as linhas cujo o segundo campo começa com t ou m, imprime a terceira coluna, segunda coluna e a quinta coluna prefixada com $

```
$ gawk '$2 ~ /^[tm]/ {print $3, $2, "$" $5}' cars
1999 malibu    $3000
1965 mustang   $10000
2003 thundbd   $10500
2000 malibu    $3500
2004 taurus    $17000
```

Repare que não foi adicionado um espaço entre $ e o preço porque não usou vírgula para separar os campos

Sinal de Dolar ($)

- Esse exemplo mostra três formas que o sinal de dolar pode assumir
  - Primeiro: antes de um número indica um campo
  - Segundo: dentro de uma expressão regular casa o fim do padrão
  - Terceiro: entre aspas indica o caractere $

```
$ gawk '$3 ~ /5$/ {print $3, $1, "$" $5}' cars
1965 ford  $10000
1985 bmw   $450
1985 chevy $1550
```

- Outros exemplos usando operadores relacionais == e <= que fazem comparações numéricas; como não tem ação imprime a toda a linha

```
$ gawk '$3 == 1985' cars
bmw     325i    1985    115 450
chevy   impala  1985    85  1550
```

```
$ gawk '$5 <= 3000' cars
plym    fury    1970    73  2500
chevy   malibu  1999    60  3000
bmw     325i    1985    115 450
toyota  rav4    2002    180 750
chevy   impala  1985    85  1550
```

Comparação Textual

- Quando se usa aspas duplas, AWK faz comparações textuais usando a tabelas ASCII (ou outra local) comparando sequências

```
$ gawk '"2000" <= $5 && $5 < "9000"' cars
plym    fury    1970    73  2500
chevy   malibu  1999    60  3000
chevy   malibu  2000    50  3500
bmw     325i    1985    115 450
honda   accord  2001    30  6000
toyota  rav4    2002    180 750
```

  – Esse exemplo mostra que, usando aspas duplas, as strings 450 e 750 do quinto campo estão entre 2000 e 9000 (provavelmente o resultado inesperado)

O Operador de Faixa (,)

- Após o operador de faixa encontrar o primeiro grupo de linhas ele começa o processamento novamente, procurando uma linha que casa com o padrão antes da vírgula

plym	fury	1970	73	2500
chevy	malibu	1999	60	3000
ford	mustang	1965	45	10000
volvo	s80	1998	102	9850
ford	thundbd	2003	15	10500
chevy	malibu	2000	50	3500
bmw	325i	1985	115	450
honda	accord	2001	30	6000
ford	taurus	2004	10	17000
toyota	rav4	2002	180	750
chevy	impala	1985	85	1550
ford	explor	2003	25	9500

cars

```
$ gawk '/chevy/ , /ford/' cars
chevy   malibu  1999   60  3000
ford    mustang 1965   45  10000
chevy   malibu  2000   50  3500
bmw     325i    1985   115 450
honda   accord  2001   30  6000
ford    taurus  2004   10  17000
chevy   impala  1985   85  1550
ford    explor  2003   25  9500
```

BEGIN

- O seguinte exemplo usa duas ações
  - Uma ação com o padrão BEGIN para imprimir um cabeçalho
  - Uma ação sem padrão que imprime todas as linhas

pr_header

```
BEGIN {
    print "Make Model Year Miles Price"
    print "----------------------------------------"
}
{print}
```

```
$ gawk -f pr_header cars
Make Model Year Miles Price
----------------------------------------
plym    fury        1970    73      2500
chevy   malibu      1999    60      3000
ford    mustang     1965    45      10000
volvo   s80         1998    102     9850
ford    thundbd     2003    15      10500
chevy   malibu      2000    50      3500
bmw     325i        1985    115     450
honda   accord      2001    30      6000
ford    taurus      2004    10      17000
toyota  rav4        2002    180     750
chevy   impala      1985    85      1550
ford    explor      2003    25      9500
```

Função `length`

- Quando a função length é chamada sem argumentos, retorna o número de caracteres na linha atual, incluindo separadores de campo; a variável $0 sempre guarda o conteúdo da linha atual

```
$ gawk '{print length, $0}' cars | sort -n
21 bmw     325i    1985   115    450
22 plym    fury    1970   73     2500
23 volvo   s80     1998   102    9850
24 ford    explor  2003   25     9500
24 toyota  rav4    2002   180    750
25 chevy   impala  1985   85     1550
25 chevy   malibu  1999   60     3000
25 chevy   malibu  2000   50     3500
25 ford    taurus  2004   10     17000
25 honda   accord  2001   30     6000
26 ford    mustang 1965   45     10000
26 ford    thundbd 2003   15     10500
```

  - Nesse exemplo AWK prefixa o tamanho da linha em cada linha para ordenar posteriormente com o comando sort

NR (Número do Registro)

- A variável NR contém o número do registro (linha) da linha atual
  - Exemplo: selecionar todas as linhas com mais de 24 caracteres

```
$ gawk 'length > 24 {print NR}' cars
2
3
5
6
8
9
11
```

  - Outro exemplo: combinando com o operador de faixa para mostrar um grupo de linhas baseado no número da linha

```
$ gawk 'NR == 2 , NR == 4' cars
chevy   malibu  1999    60  3000
ford    mustang 1965    45  10000
volvo   s80     1998    102 9850
```

END

- O padrão **END** funciona de forma similar ao padrão **BEGIN**, mas executa a ação associada a esse padrão após processar a última linha

```
$ gawk 'END {print NR, "cars for sale." }' cars
12 cars for sale.
```

  – Mostra informações após processar todas as linhas, a variável NR retém o valor ao processar a última linha

Script Standalone

- É possível escrever um shell script chamando o gawk diretamente com os comandos que deseja executar

separar_demo

```
#!/bin/gawk -f

{
    if ($1 ~ /ply/) $1 = "plymouth"
    if ($1 ~ /chev/) $1 = "chevrolet"
    print
}
```

```
$ chmod u+rx separar_demo
$ ./separar_demo cars
plymouth    fury    1970 73    2500
chevrolet   malibu  1999 60    3000
ford        mustang 1965 45    10000
volvo       s80     1998 102   9850
ford        thundbd 2003 15    10500
chevrolet   malibu  2000 50    3500
bmw         325i    1985 115   450
honda       accord  2001 30    6000
ford        taurus  2004 10    17000
toyota      rav4    2002 180   750
chevrolet   impala  1985 85    1550
ford        explor  2003 25    9500
```

Variável OFS

- Pode-se alterar o valor do separador de campos de saída atualizando o valor da variável **OFS**

ofs_demo

```
BEGIN { OFS = "\t" }
{
    if ($1 ~ /ply/) $1 = "plymouth"
    if ($1 ~ /chev/) $1 = "chevrolet"
    print
}
```

```
$ gawk -f ofs_demo cars
plymouth    fury    1970    73  2500
chevrolet   malibu  1999    60  3000
ford        mustang 1965    45  10000
volvo       s80     1998    102 9850
ford        thundbd 2003    15  10500
chevrolet   malibu  2000    50  3500
bmw         325i    1985    115 450
honda       accord  2001    30  6000
ford        taurus  2004    10  17000
toyota      rav4    2002    180 750
chevrolet   impala  1985    85  1550
ford        explor  2003    25  9500
```

  - Esse exemplo usa tabulação ao invés de espaço como separador de campos de saída ao imprimir cada linha

printf

- O comando **printf** pode ser usado para refinar melhor a saída

printf_demo

```
BEGIN {
    print "                                        Miles"
    print "Make         Model           Year    (000)   Price"
    print "-------------------------------------------------------------"
}

{
    if ($1 ~ /ply/) $1 = "plymouth"
    if ($1 ~ /chev/) $1 = "chevrolet"
    printf "%-10s       %-8s    %2d     %5d     $ %8.2f\n",\
        $1, $2, $3, $4, $5
}
```

```
$ gawk -f printf_demo cars
                                        Miles
Make         Model        Year     (000)    Price
-----------------------------------------------------
plymouth     fury         1970        73    $  2500.00
chevrolet    malibu       1999        60    $  3000.00
ford         mustang      1965        45    $ 10000.00
volvo        s80          1998       102    $  9850.00
ford         thundbd      2003        15    $ 10500.00
...
```

Redirecionando a Saída

- Redirecionando as linhas com *chevy* para o arquivo **`chevfile`** e as linhas com *ford* para o arquivo **`fordfile`**

redirect_out

```
/chevy/   {print > "chevfile"}
/ford/    {print > "fordfile"}
END       {print "done."}
```

```
$ gawk -f redirect_out cars
done.
$ cat chevfile
chevy   malibu  1999    60  3000
chevy   malibu  2000    50  3500
chevy   impala  1985    85  1550
$ cat fordfile
ford    mustang 1965    45  10000
ford    thundbd 2003    15  10500
ford    taurus  2004    10  17000
ford    explor  2003    25  9500
```

Juntando Tudo até Agora

- O programa a seguir gera um relatório sumarizado de todos os carros e somente de novos carros

Dica: embora não seja preciso, é bom declarar as variáveis no começo do programa antes de usar

summary

```
BEGIN {
        yearsum = 0 ;
        costsum = 0
        newcostsum = 0 ; newcount = 0
    }
    {
        yearsum += $3
        costsum += $5
    }
$3 > 2000 {
        newcostsum += $5 ; newcount ++
    }
END {
        printf "Average age of cars is %4.1f years\n",\
            2006 - (yearsum/NR)
        printf "Average cost of cars is $%7.2f\n",\
            costsum/NR
        printf "Average cost of newer cars is $%7.2f\n",\
            newcostsum/newcount
    }
```

```
$ gawk -f summary cars
Average age of cars is 13.1 years
Average cost of cars is $6216.67
Average cost of newer cars is $8750.00
```

Variável FS

- Ao invés de usar separadores de campo como espaços e tabulações é possível usar um delimitador diferente atualizando a variável **FS**

find_uid

```
BEGIN {
    FS = ":"
    saveit = 0
}

$3 > saveit {
    saveit = $3
}

END {
    print "Next available UID is " saveit + 1
}
```

```
$ gawk -f find_uid /etc/passwd
Next available UID is 247
```

  - Esse exemplo procura o próximo identificador livre do arquivo de usuários do sistema (/etc/passwd), separando os campo por :

if...else

- Usando **`if...else`** aninhados para gerar um relatório substituindo preço baseado no conteúdo do campo de preço

price_range

```
{
    if ($5 <= 5000)
        $5 = "inexpensive"
    else if (5000 < $5 && $5 < 10000)
        $5 = "please ask"
    else
        $5 = "expensive"

    printf "%-10s    %-8s    %2d    %5d    %-12s\n",\
        $1, $2, $3, $4, $5
}
```

```
$ gawk -f price_range cars
plym       fury       1970      73   inexpensive
chevy      malibu     1999      60   inexpensive
ford       mustang    1965      45   expensive
volvo      s80        1998     102   please ask
ford       thundbd    2003      15   expensive
...
```

Arranjos Associativos

- Usando arranjos associativos para contar a quantidade de carros no estoque de todos os fabricantes

manuf

```
#!/bin/bash

gawk '
    { manuf[$1]++ }
END {
    for (name in manuf)
        print name, manuf[name]
}
' cars | sort
```

```
$ chmod u+rx manuf
$ ./manuf
bmw 1
chevy 3
ford 4
honda 1
plym 1
toyota 1
volvo 1
```

É possível fazer genérico para qualquer campo em qualquer arquivo?

Arranjos Associativos

- Reescrevendo o programa anterior usando um campo genérico em um arquivo qualquer

items

```
#!/bin/bash

if [ $# != 2 ]; then
    echo "Usage: $0 field file"
    exit 1;
fi

gawk -v "field=$1" '
    { count[$field]++ }
END {
    for (item in count)
        printf "%-20s%-20s\n",\
            item, count[item]
}
' < $2 | sort
```

```
$ chmod u+rx items
$ ./items 1 cars
bmw                 1
chevy               3
ford                4
honda               1
plym                1
toyota              1
volvo               1
$ ./items 3 cars
1965                1
1970                1
1985                2
1998                1
1999                1
2000                1
2001                1
2002                1
2003                2
2004                1
```

ISSO É TUDO, PESSOAL!