GAWK - um sumário

Este documento destina-se a servir de referência rápida na utilização do GAWK. Por isso mesmo trata-se de um documento conciso.

Opções da linha de comandos

A linha de comandos é constituida por opções para o **gawk**, pelo texto do programa em **awk** (se não for fornecido através da opção **-f**), e valores que se pretendam disponibilizar através das variáveis **ARGC** e **ARGV**, prédefinidas no **GAWK**:

```
gawk [ opções ]  -f ficheiro_source [--] ficheiro …
gawk [ opções ] [--] ´programa´ ficheiro …
```

As opções que o **gawk** aceita são as seguintes:

-F fs
--field-separator fs
: Usar **fs** como separador dos campos de entrada (o valor da variável prédefinida **FS**).

-f ficheiro_source
--file ficheiro_source
: Ler o programa de **awk** do ficheiro **ficheiro_source**, em vez do primeiro argumento da linha de comandos.

-mf=NNN
-mr=NNN
: A flag '**f**' define o número máximo de campos e a flag '**r**' define o numero máximo de registos. Estas opções são ignoradas pelo **gawk**, uma vez que este não possui nenhuns limites prédefinidos. Elas existem apenas para compatibilidade com versões antigas do **awk**.

-v var=val
--assign var=val
: Atribuir à variável **var** o valor **val** antes que a execução do programa se inicie.

-W traditional
-W compat
--traditional
--compat
: Usar o modo de compatibilidade no qual as extensões próprias do **gawk** estão desligadas.

-W copyleft
-W copyright
--copyleft
--copyright
: Imprime a versão curta da General Public License no "**stderr**". Esta opção pode desaparecer em versões futuras do **gawk**.

-W help
-W usage
--help
--usage
Imprime no "**stderr**" um sumário curto das opções disponíveis.

-W lint
--lint
Fornece avisos sobre construções de significado dúbio ou não portável.

-W lint-old
--lint-old
Fornece avisos sobre construções que não estão disponíveis na versão original do **awk** para Unix Version 7.

-W posix
--posix
Usa o modo de compatibilidade POSIX, no as extensões do gawk estão desligadas e se aplicam certas restrições adicionais.

-W re-interval
--re-interval
Permite que expressões de intervalos sejam usadas como expressões regulares.

-W source=ficheiro-programa
--source ficheiro-programa
Usa o texto de **ficheiro-programa** como source code do **gawk**. Esta opção permite que se misture programas na linha de comandos com programas em ficheiros, e é particularmente útil para misturar programas na linha de comandos com bibliotecas de funções.

-W version
--version
Imprime no "**stderr**" a versão do **gawk** que se está a utilizar.

-- Assinala o fim das opções. Isto é útil para permitir que certos argumentos para o **awk** possam começar com '-'.

Sumário da linguagem

Um programa em **awk** consiste numa sequência de zero ou mais instruções do tipo padrão-acção e definições de funções opcionais. Em cada instrução podemos omitir ou o padrão ou a acção.

```
padrão              { instruções da acção }
padrão
            { instruções da acção }

function    nome(lista de parâmetros)    { instruções }
```

O **gawk** lê primeiro o source code dos ficheiros de programa, se estes tiverem sido especificados, ou do primeiro argumento que não seja uma opção da lista de comandos. A opção '**-f**' poder ser usada várias vezes na linha de comandos. O **gawk** lê o texto de todos os ficheiros source, concatenando-os na ordem em que estes forem especificados. Isto é útil para construir bibliotecas

de funções sem ter de as incluir em cada novo programa de **awk** que as use. Para usar um biblioteca de funções num programa escrito na linha de comandos basta especificar '**--source ´biblioteca´**', e escrever o programa entre pelicas para que a shell não o processe.

A variável de ambiente **AWKPATH** especifica os directórios que devem ser pesquisados para encontrar ficheiros source que se definiram com a opção '**-f**'. Se a variável de ambiente **AWKPATH** não estiver definida, os directórios a pesquisar por defeito são o directório corrente e o **/usr/local/share/awk** (dependendo da maneira como o **gawk** tenha sido compilado e instalado). Se o nome do ficheiro tiver um '/' o **gawk** não o procura.

O **gawk** compila o programa para uma representação própria interna, e de seguida lê cada ficheiro do vector **ARGV**. Os valores iniciais do vector **ARGV** vêm dos argumentos da linha de comandos. Se na linha de comandos não se especificaram ficheiros o **gawk** lê o "*standard input*".

Se um "*ficheiro*" especificado na linha de comandos tem a forma '**var=val**', então é tratado como se fosse uma atribuição de uma variável: a variável **var** passa a ter o valor **val**. Se algum dos ficheiros tem o valor correspondente à string nula, então esse elemento da lista é ignorado.

Para cada registo na entrada, o **gawk** testa-o para ver se pode ser associado a qualquer padrão no programa em **awk**. Para cada padrão que possa ser associado, a acção correspondente é executada.

Variáveis e campos

As variáveis em **awk** não são declaradas; elas são criadas quando se usam pela primeira vez. Os seus valores podem ser números em vírgula flutuante ou strings. O **awk** também possui vectores; e podemos simular matrizes. Existem várias variáveis prédefinidas a que o **awk** atribui valores quando um programa é executado.

Campos

Quando cada linha de entrada é lida, o gawk separa a linha em campos, usando o valor da variável **FS** como o separador dos campos. Se **FS** for um único caracter , os campos serão separados por esse caracter. Senão **FS** também pode ser uma expressão regular. No caso especial en que **FS** é um único espaço, os campos podem estar separados por vários espaços e/ou tabs. Se **FS** for uma string vazia ("") então cada caracter individual no registo torna-se um campo separado. Note-se que o valor da variável **IGNORECASE** também afecta o modo como os campos são separados quando **FS** é uma expressão regular.

Cada campo na linha de entrada pode ser referenciado pela sua posição, **$1**,**$2**, etc. Para nos referirmos à linha toda podemos utilizar **$0**. O valor de um campo além de ser lido também pode ser modificado através de uma atribuição. O número do campo não precisa de ser uma constante, podendo ser uma variável:

```
n=5
print $n
```

imprime o quinto campo na linha de entrada. A variável **NF** contém o número total de campos na linha de entrada.

Referência a campos inexistentes (depois de **$NF**) devolvem uma string nula. No entanto atribuições a campos inexistentes (ex: **$(NF+2)=5**) criam esses campos com o valor atribuído, criam os campos intermédios com o valor

da string nula, e forçam a re-avaliação de **$0** com os campos a serem separados pelo valor de **OFS**.

Variáveis Existentes

As variáveis que o **gawk** possui internamente são:

ARGC O número de elementos em **ARGV**.

ARGIND O indíce em **ARGV** do ficheiro que está a ser processado. Quando o **gawk** está a processar ficheiros de dados é sempre verdade que '**FILENAME==ARGV[ARGIND]**'.

ARGV O vector dos argumentos da linha de comandos. O vector é indexado de 0 a **ARGC**-1. Para controlar os ficheiros usados como dados podemos alterar **ARGV** e **ARGC**. Um elemento com o valor nulo em **ARGV** é ignorado. Note-se que **ARGV** não inclui as opções do **gawk** ou o texto do programa quando este se encontra na linha de comandos.

CONVFMT O formato de conversão que o **gawk** usa ao converter números em strings.

FIELDWIDTHS
Uma lista de números separados por espaços, que descreve a largura dos campos, quando queremos manipular ficheiros com campos de largura fixa.

ENVIRON Um vector com as variáveis de ambiente. O vector é indexado pelo nome das variáveis, sendo cada elemento o valor de cada variável. Assim a variável de ambiente **HOME** será **ENVIRON["HOME"]**. Um valor possível será '**/home/arnold**'.
Ao alterar este array não mudamos o ambiente de execução de qualquer programa que o **gawk** execute através de redirecção ou da função **system**.
(Isto pode mudar em versões futuras do **gawk**).
Alguns sistemas operativos não possuem variáveis de ambiente, neste caso o vector **ENVIRON** será um vector vazio.

ERRNO A mensagem de erro do sistema quando acontece um erro no uso de getline ou close.

FILENAME O nome do ficheiro de entrada corrente. Se não for especificado nenhum ficheiro na linha de comandos, então **FILENAME** será a string nula.

FNR O número do registo no ficheiro corrente.

FS O separador dos campos de entrada, um espaço por defeito.

IGNORECASE
A flag correspondente à distinção ou não entre maiúsculas e minúsculas, para operações com strings e para o processamento de expressões regulares. Se **IGNORECASE** tem um valor diferente de zero, então a associação de padrões nas regras, a separação de registos com **RS**, a separação de campos com **FS**, a associação de expressões regulares com '**~**' e '**!~**', as funções

internas **gensub**, **gsub**, **index**, **match**, **split** e **sub** ignoram a diferença entre maiúsculas e minúsculas quando fazem operaçoes com expressões regulares, e todas as comparaçoes de strings são feitas do mesmo modo.

NF O número de campos no registo corrente.

NR O número total de registos até ao registo corrente.

OFMT O formato de saída para números na instrução **print**, por defeito "**%.6g**".

OFS O separador dos campos na saída, um espaço por defeito.

ORS O separador dos registos na saída, por defeito uma nova linha.

RS O separador dos registos na entrada, por defeito uma nova linha. Se o valor de **RS** for a string nula então os registos estarão separados por uma linha em branco. Neste caso, o caracter correspondente a uma nova linha tamém actua como separador dos campos, independentemente do valor que a variável **FS** possa ter. O valor de **RS** pode ser uma string, funcionando neste caso como uma expressão regular, sendo o texto que se associe a essa expressão regular, o separador dos registos.

RT O texto que foi associado a **RS**, no caso de este ser uma expressão regular.

RSTART O indíce do primeiro caracter associado pela função **match**.

RLENGTH O comprimento da string associada pela função **match**; -1 se a associação não se efectou.

SUBSEP A string utilizada para separar indíces multiplos nos elementos de um vector, por defeito "**\034**".

Vectores

Os vectores são indexados por uma expressão entre parêntesis rectos. Os indíces dos vectores são sempre strings; os números são convertidos em strings se for necessário, observando as regras de conversão standard.
Se usarmos expressões múltiplas separadas por vírgulas dentro dos parêntesis rectos, então o indíce para o vector será uma string constituída pela concatenação das expressões individuais, convertidas em strings, separadas pelo caracter **SUBSEP**.
O operador especial **in** pode ser usado num contexto condicional para ver se um vector tem um índice com um certo valor.

```
if ( val in array )
        print  array[val]
```

Se um vector tem índices multiplos, então devemos usar **(i,j,...) in array**, para testar a existência de um elemento. O operador **in** também pode ser usado num ciclo **for** para varrer todos os elementos de um vector. Podemos apagar um vector ou os seus elementos individualmente com a função **delete**.

Tipos de dados

O valor de uma expressão em **awk** é sempre um número ou uma string. Alguns contextos (como os operadores aritméticos) necesstam de valores numéricos. Eles convertem as strings em números, interpretando o texto da string como um número. Se a string não se parecer com um número, o seu valor será zero. Outros contextos (tais como a concatenação) necesitam de strings. Eles convertem os números em strings através da função **sprintf**. Para forçar a conversão de uma string num número basta adicionar-lhe zero. Se a "string" já for um número, não ficará alterada. Para forçar a conversão de um valor numérico numa string, podemos concatenar ao seu valor a string nula.

As comparações são feitas numéricamente se ambos os operandos forem numéricos ou se um for numérico e o outro for uma string numérica. Senão um ou os dois operandos são convertidos em strings, e é feita uma comparação de strings. Os campos, a entrada de **getline**, **FILENAME**, os elementos de **ARGV**, os elementos de **ENVIRON**, e os elementos de um vector criado por **split** são os únicos items que podem ser strings numéricas. Constantes tais como **"3.1415927"** não são strings numéricas, são strings constantes. As variáveis não inicializadas possuem o valor **""** (a string nula ou vazia). Nos contextos onde é exigido um valor numérico isto é equivalente a zero.

Padrões e Acções

Um programa em awk é composto de regras, cada uma das regras consistindo de um padrão seguido de uma acção. A acção encontra-se delimitada por '**{**' e '**}**'. O padrão pode não existir ou a acção pode não existir, mas não podem faltar os dois. Se o padrão não existir a acção é executada para todos os registos de entrada. Uma acção inexistente equivale a '**{ print }**', que imprime o registo de entrada.

Os comentários começam com o caracter '**#**' e terminam no fim da linha. As instruções podem ser separadas por linhas em branco. Normalmente uma instrução termina com o fim da linha, no entanto isto não é válido para linhas que terminem com '**,**','**{**','**?**','**:**','**& &**' ou '||'. As linhas que terminam em do ou else também são uma excepção. Em qualquer outro caso, a instrução também pode continuar na linha seguinte se a linha em que está colocada terminar com '\'.

Podemos colocar mais do que uma instrução na mesma linha, se separarmos as instruções com '**;**'.

Padrões

Os padrões em awk podem assumir uma das seguintes formas:

/*expressão regular*/
expressão relacional
padrão && *padrão*
padrão || *padrão*
padrão ? *padrão* : *padrão*
(*padrão*)
! *padrão*
padrão1, *padrão2*
BEGIN
END

BEGIN e **END** são dois padrões especiais que não são testados com o ficheiro de entrada. As acções de todos os padrões **BEGIN** são concatenadas, como se todas as instruções tivessem sido escritas numa única regra **BEGIN**. Elas são executadas antes que a entrada seja lida. Da mesma forma, todas as acções correspondentes às regras **END** são concatenadas e executadas quando a entrada de dados terminar (ou quando for executada a instrução exit). Os padrões **BEGIN** e **END** não podem ser combinados com outros padrões, e as suas acções não são opcionais, existindo obrigatóriamente.

Para padrões do tipo '**/expressão regular/**' , a acção associada é executada para cada linha de entrada que consiga ser associada à expressão. As expressões regulares são as mesmas do egrep sendo sumarizadas mais à frente.

Uma expressão relacional pode qualquer dos operadores definidos para as acções. Este tipo de expressões testa normalmente se certos campos podem ser associados a expressões regulares.

Como em C os operadores '**&&**','||', e '**!**' correspondem aos operadores lógicos '**and**', '**or**' e '**not**'. A avaliação das expressões é feita em "curto-circuito", como em C. Como na maioria das linguagens, os parêntesis podem ser usados para mudar a prioridade das expressões.

O operador '**?:**' é usado da mesma maneira que em C. Se o primeiro padrão fôr associado ao registo de entrada, então tenta-se associar o segundo o segundo padrão ao registo de entrada; senão tenta-se associar o terceiro. Só um dos padrões, entre o segundo e o terceiro é associado.

O padrão da forma '**padrão1,padrão2**' é um padrão de "intervalo". É associado a todas as linhas de entrada, desde a linha associada a '**padrão1**' até à linha associada a '**padrão2**', inclusivé. Um padrão de intervalo não pode ser usado como um operando para os operadores de manipulação de padrões.

Expressões regulares

As expressões regulares são do tipo usado no **egrep**. Elas são compostas pelos seguintes caracteres:

c	o caracter ***c*** (assumindo que ***c*** é um caracter sem um significado especial)
c	o caracter ***c*** (mesmo que ***c*** tenha um significado especial)
.	qualquer caracter excepto o '**newline**'
^	o início de uma linha ou de um campo
$	o fim de uma linha ou de um campo
[abc...]	qualquer dos caracteres **abc...** (classe de caracteres)
[^abc..]	qualquer caracter excepto **abc...** e '**newline**'
r1|r2	a expressão **r1** ou a expressão **r2**
r1r2	a concatenação da expressão **r1** com **r2**
r+	uma ou mais vezes a expressão **r**
r*	zero ou mais vezes a expressão **r**
r?	zero ou uma vez a expressão **r**
(r)	a expressão **r**

Acções

As instruções correspondentes às acções são escritas entre chavetas, '**{**' e '**}**'. As instruções são constituídas pelas atribuições, instruções condicionais e instruções de ciclos como em outras linguagens de programação. A linguagem em termos gerais é modelada a partir do C.

Operadores

Os operadores em awk, são os seguintes, listados com precedência crescente:

= += -= *= /= %= ^=
: Atribuição. Pode-se usar a forma normal da atribuição ou a atribuição com operadores.

?: Uma expressão condicional como em C.

|| "OU" lógico.

&& "E" lógico.

~ !~ Associação de expressões regulares, não-associação.

< <= > >= != ==
: Operadores relacionais.

branco Concatenação de strings.

+ - Adição e subtração.

* / % Multiplicação, divisão e resto da divisão inteira.

+ - ! Operadores unários mais, menos e negação lógica.

^ Exponenciação (também se pode usar '**' e '**=')

++ -- Incremento e decremento (pré e pós)

$ Referência a campos.

Instruções de controle

if (*condição*) *instrução* [else *instrução*]

while (*condição*) *instrução*

do *instrução* while (*condição*)

for (*expr1* ; *expr2* ; *expr3*) *instrução*

for (*var* in *array*) *instrução*

break

continue

delete *array*[*index*]

exit [*expresssion*]

{ *instruções* }

Instruções de I/O

getline Ler **$0** do próximo registo de entrada; actualiza **NF**,**NR**, e **FNR**.

getline <ficheiro
Ler **$0** do próximo registo de **ficheiro**; actualiza **NF**.

getline var Ler **var** do próximo registo de entrada; actualiza **NF** e **FNR**.

getline var <fich
Ler **var** do próximo registo de **ficheiro**.

next Pára o processamento do registo corrente de entrada. O registo de entrada seguinte é lido, e o processamento recomeça com o primeiro padrão do programa em **awk**. Se entretanto o fim da entrada tiver sido alcançado, a(s) regra(s) correspondente(s) ao padrão **END** são executadas.

print imprime o registo corrente

print expr-list
imprime expressões

print expr-list > ficheiro
imprime expressões para um ficheiro

printf fmt, expr-list
formata e imprime

printf fmt, expr-list > ficheiro
formata e imprime para um ficheiro

Também são permitidas outras redirecções de entrada/saída. Para **print** e **printf** '**>> ficheiro**' adiciona ao ficheiro, enquanto '**| comando**' escreve para um pipe. De uma forma semelhante '**comando | getline** ' redirige a saida de **comando** para **getline**. A instrução **getline** devolve 0 no fim do ficheiro e -1 se acontecer um erro.

Sumário do printf

A instrução **printf** e a função sprintf aceitam as seguintes especificações de conversão de formatos:

%c Um caracter ASCII. Se o argumento utilizado for numérico, é tratado
como um caracter e é impresso. Senão o argumento é tratado como uma string e é impresso apenas o primeiro caracter dessa string.

%d Um número decimal (a parte inteira)

%i Um número inteiro (também).

%e Um número em vírgula flutuante na forma '**[-]d.dddddd[+-]dd**'.

%f Um número em vírgula flutuante na forma '**[-]ddd.dddddd**'.

%g Usar '**%g**' ou '**%f**' (escolher a forma mais curta) sem zeros à esquerda.

%o Um número em octal (inteiro)

%s Uma string.

%x Um número sem sinal em hexadecimal (inteiro).

%X Como '**%x**', mas usa 'A' a 'F' em vez de 'a' a 'f'.

Podem existir parâmetros adicionais entre o '**%**' e o caracter de controle:

- A expressão deve ser imprimida encostada à esquerda, dentro do campo.

largura
A expressão deve ser imprimida com esta largura. Se largura começa com um zero, então o campo será cheio com zeros. Senão será preenchido com espaços.

.num Um número que indica a largura máxima das strings ou os dígitos à
direita do ponto decimal.

Nomes especiais de ficheiros

Quando se faz redirecção de entrada/saída com **print** ou **printf** para um ficheiro, ou através do getline de um ficheiro, o **gawk** reconhece certos nomes especiais internamente. Estes nomes permitem o acesso a descritores de ficheiros abertos, herdados do processo pai do **gawk** (geralmente uma shell). Os nomes dos ficheiros são:

'/dev/stdin' A entrada standard.

'/dev/stdout' A saída standard.

'/dev/stderr' A saída de erros standard.

'/dev/fd/*n*' O ficheiro aberto associado ao descritor **n**.

Estes nomes de ficheiros também podem ser usados na linha de comandos, como ficheiros de dados.

Funções numéricas

O **awk** tem as seguintes funções numéricas:

atan2(y,x) devolve o arcotangente de y/x em radianos.

cos(expr) devolve o coseno de uma expressão em radianos.

exp(expr) a função exponencial.

int(expr) trunca uma expressão para um número inteiro.

log(expr) a função logaritmo natural.

rand() devolve um número aleatório entre 0 e 1.

sin(expr) devolve o seno de uma expressão em radianos.

sqrt(expr) a função raiz quadrada.

srand(expr)
usa **expr** como a nova semente do gerador de números aleatórios. Se expr não for fornecida, usa o tempo do dia como semente. O valor de retorno é a semente anterior do gerador.

Funções de manipulação de strings

O **awk** tem as seguintes funções de manipulação de strings:

gsub(r,s,t) para cada substring que possa ser associada à expressão regular **r** na string **t**, substitui a string **s** e devolve o número de substituições. Se a string **t** não for fornecida, usa **$0**.

index(s,t) devolve o indíce da string **t** na string **s**, ou 0 se **t** não estiver contido em **s**.

length(s) devolve o comprimento da string **s**.

match(s,r) devolve a posição de **s** onde a expressão regular **r** ocorre, ou 0 se **r** não estiver presente, e actuliza os valores das variáveis **RSTART** e **RLENGTH**.

split(s,v,r) separa uma string **s** num vector **v** usando a expressão regular **r** como separador e devolve o número de campos. Se não for dado o valor de **r**, usa **FS**.

sprintf(fmt,expr-list)
formata a impressão de **expr-list** de acordo com **fmt**, e devolve a string resultante.

sub(r,s,t) funciona como **gsub**, mas substitui apenas a primeira ocorrência.

substr(s,i,n)
devolve a substring de **s** com **n** caracteres que começa na posição **i**. Se **n** não for dado, a função devolve a substring desde a posição **i** até ao fim da string.

tolower(str) devolve uma cópia da string **str** com os caracteres maiúsculos transformados em minúsculos. Os caracteres não-alfabéticos ficam inalterados.

toupper(str)
devolve uma cópia da string str com os caracteres minúsculos transformados em maiúsculos. Os caracteres não-alfabéticos ficam inalterados.

system(cmd-line)
executa o comando **cmd-line**, e devolve o "exit status" do comando.

Strings

As strings em **awk** são sequências de caracteres entre aspas("). Dentro da strings, como em C são reconhecidas certas sequências de caracteres:

\\ Um "backslash".
\a O caracter de "alerta"; normalmente o caracter ASCII BEL.
\b "Backspace".
\f "Formfeed".
\n "Newline".
\r "Carriage return".
\t "Tab" horizontal.
\v "Tab" vertical.
\xhex
O caracter representado pela sequência de dígitos hexadecimais a seguir ao '**\x**'.

\ddd O caracter representado por uma sequência de 1, 2 ou três dígitos em octal.

\c O caracter c.

As sequências de "escape" também podem ser usadas dentro de expressões regulares.

Funções

As funções em **awk** são definidas da forma seguinte:

function *nome*(*lista de parâmetros*) { *instruções* }

Os parâmetros fornecidos na chamada da função são usados para instanciar os parâmetros formais declarados na função. Os vectores são passados por referência e as outras variáveis são passadas por valor.

Se o número de parâmetros passados fôr menor do que o número de parâmetros declarados, os parâmetros adicionais recebem como valor a string nula. Os parâmetros adicionais funcionam então como variáveis locais.

Na chamada de uma função o "abrir parênteses" deve estar colocado logo a seguir ao nome da função, sem nenhum espaço intermédio. Isto acontece devido a uma ambiguidade sintática com o operador da concatenação.

A palavra **func** pode ser usada em vez de **function**.