Introducció a AWK

PID_00270624

Guillem Lluch Moll

Temps mínim de dedicació recomanat: 3 hores

Universitat Oberta de Catalunya

Guillem Lluch Moll

Llicenciat en Matemàtiques per la Universitat Autònoma de Barcelona (UAB), màster en Programari Lliure per la Universitat Oberta de Catalunya (UOC) i màster en Llenguatges i Sistemes Informàtics (UNED). De llarga trajectòria docent, en diferents nivells i assignatures, actualment treballa com a coordinador TIC en un institut de secundària.

L'encàrrec i la creació d'aquest recurs d'aprenentatge UOC han estat coordinats pel professor: Julià Minguillón Alfonso (2020)

Primera edició: febrer 2020

Av. Tibidabo, 39-43, 08035 Barcelona
Realització editorial: FUOC

Índex

1. Funcionament i sintaxi

El programa **AWK**, a l'igual que Sed, funciona línia a línia. Però en lloc de veure cada línia com un tot, com un únic element, AWK considera que **les línies estan formades per camps**, camps separats per un o més espais, encara que aquest separador es pot canviar.

La sintaxi amb AWK consisteix en un patró i un procediment. Les dues són opcionals:

- Si no hi ha cap patró (però hi ha una o més accions), l'acció es porta a terme per a totes les línies.
- Si no hi ha cap procediment (però hi ha un patró), s'imprimeix les línies que coincideixen amb el patró.

Els patrons segueixen el format **ERE**.[1] Les comandes AWK s'escriuen entre cometes simples per a evitar conflictes amb la *shell*.

(1)Acrònim de *Extended Regular Expressions*.

A cada línia, AWK enumera els camps. Ens podem referir a aquests anteposant `$` al lloc que ocupa: així `$1` és el primer camp, `$2` el segon, etc. `$0` és tota la línia i `$NR`, l'últim camp.

Els procediments d'AWK són molt variats i inclouen les accions típiques dels llenguatges de programació estructurats (`if-else`, `for`, `while`, etc.). La seva sintaxi és molt similar a la de C.

> Podem entendre AWK com un llenguatge de programació en què la iteració línia a línia ve donada per defecte, automàticament.

AWK es pot ser servir escrivint les ordres en la línia de comandes o des d'un fitxer, emprant l'opció `-f`. Les accions es poden aplicar sobre un fitxer o es pot redirigir la sortida d'una comanda prèvia (com ara `echo` o `cat`). Exemples:

```
# Imprimeix la línia d'entrada amb la funció print. Els procediments es posen entre claus.
echo "field1 field2 field3" | awk '{print $0}'

field1 field2 field3
```

```
# S'aplica el patró de trobar una f literal. Com que no hi ha cap procediment, s'imprimeix.
echo "field1 field2 field3" | awk '/f/'

field1 field2 field3
```

```
# S'aplica el patró de trobar una c literal. No coincideix i, per tant, no es mostra res.
echo "field1 field2 field3" | awk '/c/'
```

```
# Es poden emprar més d'un procediment alhora, separats per ;.
# Imprimeix el primer i el tercer camp.
echo "field1 field2 field2bis" | awk '{print $1; print $3}'

field1
field2bis
```

```
# Es poden emprar més d'un argument a print separats per ,.
# Imprimeix el primer i el tercer camp en la mateixa línia, separats per espai.
echo "field1 field2 field2bis" | awk '{print $1,$3}'

field1 field2bis
```

```
# Per a concatenar es deixa un espai en blanc.
# Imprimeix el primer i el tercer camp en la mateixa línia, sense cap separació.
echo "field1 field2 field2bis" | awk '{print $1 $3}'

field1field2bis
```

Per a il·lustrar les explicacions, farem servir el fitxer «sales.csv» (generat a mockaroo.com/).

```
head -n6 sales.csv

id,first_name,last_name,email,gender,country,2017,2018,2019
1,Lulu,Chaloner,lchaloner0@reuters.com,Female,France,77.73,85.9,60.46
2,Vivianne,Feeney,vfeeney1@washingtonpost.com,Female,United Kingdom,10.22,51.01,49.88
3,Waring,Craythorne,wcraythorne2@merriam-webster.com,Male,France,3.7,35.21,72.35
4,Wakefield,Shepton,wshepton3@sphinn.com,Male,France,78.39,86.02,86.35
5,Beale,Myrtle,bmyrtle4@bloomberg.com,Male,United Kingdom,68.06,77.78,40.26
```

El primer que hem de fer és canviar el separador d'AWK per defecte. En lloc d'espai, volem la coma. Això es fa amb l'opció `-F` (en majúscules) d'AWK.

```
# Obté el nom. Per raons d'espai, es redirigeix la sortida i
# només es mostren els 5 primers noms.
awk -F, '{print $2}' sales.csv | head -n6

first_name
Lulu
Vivianne
Waring
Wakefield
Beale
```

Hi ha accions que només interessa fer un cop, no a cada línia. Amb AWK, podem realitzar accions no lligades a cap registre d'entrada **abans**, dins de la secció `BEGIN` o **després**, a la secció `END`. Tant una com l'altra són optatives.

```
# BEGIN s'empra molt per a assignar un valor a la variable FS,
# que és la que conté el separador dels camps.
# Abans de fer res, establim la coma com a separador dels camps.
# Per a cada entrada, imprimim el segon camp.
awk 'BEGIN {FS=","} {print $2}' sales.csv | head -n6

first_name
Lulu
Vivianne
Waring
Wakefield
Beale
```

```
# Exemple d'END.
# Abans de fer res, establim la coma com a separador dels camps.
# Per a cada entrada, imprimim el segon camp.
# Una vegada processades totes les línies, s'imprimeix un missatge que informa del final.
awk 'BEGIN {FS=","} {print $2} END {print "OK. Final"}' sales.csv |tail -n6
```

```
Sigismondo
Liuka
Tynan
Aylmer
Alexandr
OK. Final
```

Quan hi ha diversos procediments AWK a executar, convé posar-ho en un fitxer i cridar-lo amb `-f`:

```
# Mostra el fitxer de l'script, que produeix el mateix resultat que l'anterior.
cat surnames
```

```
# Mostra el segon camp d'un fitxer separat per comes i imprimeix un missatge final.
BEGIN {FS=","}
{print $2}
END {print "OK. Final"}
```

Fixeu-vos que podem incloure comentaris en els *scripts* d'AWK. Per a aplicar aquest *script* a un fitxer, hem d'emprar `-f`:

```
# Apliquem l'script d'AWK surnames al fitxer sales.csv
awk -f surnames sales.csv | tail -n6

Sigismondo
Liuka
Tynan
Aylmer
Alexandr
OK. Final
```

2. Variables i arguments

Es pot tenir variables amb AWK, simplement assignant un valor al nom de la variable amb el símbol =. A més de les variables que es poden establir, hi ha unes quantes variables predefinides, que s'empren molt, com ara `FS`, que ja hem vist, que indica el separador dels camps.

Amb les **variables**, s'ha de vigilar ja que qualsevol variable no declarada prèviament està inicialitzada a 0 (que equival a fals) i a la cadena buida.

Amb AWK totes les variables tenen un valor com a cadena i com a número. AWK tria un dels dos valors, en funció del context. Les cadenes que no són numèriques (completament), tenen un valor numèric de 0.

```
# Ús d'una variable, france, no inicialitzada.
# Compta el nombre de línies en què apareix la paraula France.
# ATENCIÓ: com que france no està declarada, inicialment val 0.
awk '/France/ {france=france+1} END {print france}' sales.csv

9

# Ús d'una variable, país, inicialitzada.
# Compta el nombre de línies en què apareix la paraula France.
cat countries

# Compta el nombre de línies en què apareix France.
BEGIN {pais=0}
/France/ {pais=pais+1}
END {print pais}

awk -f countries sales.csv

9
```

2.1. Variables del sistema

Com ja hem dit, hi ha diverses variables que ja estan predefinides. `FS` serveix per a especificar el separador. Aquest separador no cal que estigui format per un únic caràcter, ja que pot estar format per un conjunt de caràcters. De fet, pot ser una expressió regular.

```
echo "inicio -> derecha -> arriba -> derecha" | awk 'BEGIN {FS=" -> "} {print $2,$3,$4}'

derecha arriba derecha
```

De la mateixa manera que hi ha un camp separador per a l'entrada, també n'hi ha un per a la sortida, `OFS`, per defecte, l'espai en blanc.

```
echo "inicio -> derecha -> arriba -> derecha" | awk 'BEGIN {FS=" -> "; OFS=", "} {print $2,$3,$4}'

derecha, arriba, derecha
```

Una altra variable del sistema útil és `NF` que correspon al recompte de camps per a la línia actual. Canviar-lo, sumant, crea més camps buits i disminuir-lo farà desaparèixer els camps més enllà de l'últim considerat.

```
awk 'BEGIN {FS=","} {print NF}' sales.csv |tail -n4

9
9
9
9
```

`RS` guarda el separador de línies, que per defecte és `\n`, és a dir, una nova línia. Però el podríem canviar per tractar registres de més d'una línia (o menys).

`NR` és el nombre de la línia, o millor dit, de registre actual.

```
awk 'BEGIN {FS=","} {print NR}' sales.csv |tail -n6

10
11
12
13
14
15
```

`FILENAME` conté el nom del fitxer sobre el qual estem treballant.

```
awk 'END {print FILENAME}' sales.csv

sales.csv
```

Per a controlar com es converteixen els números en cadenes, AWK ofereix `CONVFMT` i s'especifica amb la funció `printf` de C. Per defecte, AWK agafa fins a sis valors decimals.

Funció `printf` de C

Per a més informació, vegeu «Códigos de formato de printf/scanf».

La forma com es mostren els números es controla amb `OFMT`, també emprant la sintaxi de `printf` de C.

Per defecte, AWK considera que el separador decimal és el punt, independentment de *locale*. Perquè consideri el separador propi de l'idioma del sistema es pot emprar l'opció `-N`.

```
# Canviem el valor OFMT per tal que mostri 3 dígits després del separador decimal.
# total és un acumulador de la suma dels valors anteriors. En aquest cas, hauríem d'haver
saltat el primer registre.
awk 'BEGIN {FS=","; OFMT="%.3f"} {total+=$7; print total}' sales.csv

2017
2094.730
2104.950
2108.650
2187.040
2255.100
2291.260
2301.840
2382.590
2390.330
2422.300
2490.710
2497.630
```

```
2546.860
2596.070
```

2.2. Operacions

Les operacions matemàtiques permeses són les habituals:

Operador	Operació
+	Suma
–	Resta
*	Multiplicació
/	Divisió
%	Mòdul
^	Exponenciació
**	Exponenciació

Observeu que l'exponenciació es pot dur a terme amb qualsevol dels dos operadors especificats en la taula.

Per exemple:

```
awk 'BEGIN { a = 20; b = 3; print "(a ** b) = ", (a ** b) }'

(a ** b) = 8000

awk 'BEGIN { a = 117; b = 7; print "(a % b) = ", (a % b) }'

(a % b) = 5
```

També es poden emprar **operadors d'assignació**, molts típics en els llenguatges de programació similars a C:

Operador	Operació
++	Suma 1. Unari.
––	Resta 1. Unari.
+=	Assigna a la variable de l'esquerra el resultant de sumar-li el valor de la dreta.
–=	Assigna a la variable de l'esquerra el resultant de restar-li el valor de la dreta.
/=	Assigna a la variable de l'esquerra el resultant de dividir-lo pel valor de la dreta.
%=	Assigna a la variable de l'esquerra el resultant d'aplicar-li el mòdul pel valor de la dreta.
^=	Assigna a la variable de l'esquerra el resultant d'elevar-lo al valor de la dreta.

Operador	Operació
**=	Assigna a la variable de l'esquerra el resultant d'elevar-lo al valor de la dreta.

Per exemple:

```
awk 'BEGIN { a = 117; b = 7; a%=b ;print "a = ", a }'

a = 5

awk 'BEGIN {a++; a++; a++ ;print "a = ", a }'

a = 3

awk 'BEGIN { a = 117; b = 7; a-=b ;print "a = ", a }'

a = 110
```

Finalment, hi ha les **operacions lògiques**, booleanes, molt útils pel control de flux, que veurem una mica més endavant. Quan una expressió és certa, produeix com a resultat 1 i quan no ho és, 0.

Operador	Operació
<	Menor que
>	Major que
<=	Menor o igual que
>=	Major o igual que
==	Igual a
!=	Diferent de
~	Coincideix amb l'expressió regular
!~	No coincideix amb l'expressió regular
||	O
&&	I
!	Negació

Alguns exemples per a provar les expressions lògiques de la taula:

```
awk 'BEGIN {a=(6<8); print "a= ", a}'

a= 1

awk 'BEGIN {a=(6>8); print "a= ", a}'

a= 0

awk 'BEGIN {a=(6==8); print "a= ", a}'

a= 0
```

```
awk 'BEGIN {a=(6!=8); print "a= ", a}'

a= 1

awk 'BEGIN {a=("sur,este,norte"~".+,.+"); print "a= ", a}'

a= 1

awk 'BEGIN {a=("sur,este,norte"~".+;.+"); print "a= ", a}'

a= 0

awk 'BEGIN {a=(val==0 && val>=0 ); print "a= ", a}'

a= 1

# A val 1 pels registres parells o pel registre 13.
awk '{a= (NR%2==0 || NR==13); print "NR= ",NR,"a= ", a}' sales.csv

NR= 1 a= 0
NR= 2 a= 1
NR= 3 a= 0
NR= 4 a= 1
NR= 5 a= 0
NR= 6 a= 1
NR= 7 a= 0
NR= 8 a= 1
NR= 9 a= 0
NR= 10 a= 1
NR= 11 a= 0
NR= 12 a= 1
NR= 13 a= 1
NR= 14 a= 1
NR= 15 a= 0
```

2.3. Arguments i pas de paràmetres

Podem passar paràmetres a un *script* AWK posant el nom de la variable, el símbol igual i el valor que volem donar-li. Això s'ha de posar abans del nom dels fitxers sobre els quals volem que actuï. Com a exemple, vegem l'*script* següent:

```
cat countries_var2

# Compta el nombre de línies en què apareix la paraula especificada per nom_pais

$0 ~ country_name {country=country+1} # Si a la línia hi apareix el nom del país
END {print country_name,"has appered",country,"times"}
```

```
# Cridem l'script però li diem que ja tenim 10 aparicions anteriors
awk -f countries_var2 country=10 country_name=France sales.csv

France has appered 19 times
```

Si cal repetir molts cops la mateixa comanda, podem crear *scripts* AWK que es cridin des de Bash a fi de no haver d'assignar-los-hi explícitament els noms de les variables (això sí, l'ordre sí que serà important). És el que es coneix com a ***wrapper***, un embolcall.

Cal recordar aquí que, amb Bash, `$1` és el primer argument indicat a l'*script*, `$2` el segon i així consecutivament. Per a fer l'embolcall, no hem de confondre aquestes expressions amb els camps d'AWK. El que fem en el *wrapper* és dir-li que el primer argument bash anirà a `country` i el segon a `country_name`:

```
# Mostra el wrapper de bash. És una sola línia.
cat countries_wrapper

awk -f countries_var2 "country=$1" "country_name=$2" sales.csv

# Recordeu donar els permisos d'execució a countries_wrapper
# S'executa l'script amb la primera variable, per country, igual a 20 i la segona,
# per country_name valent France.
./countries_wrapper 20 France

France has appered 29 times
```

Una altra opció, si no hem d'interactuar amb la *shell*, és emprar a l'inici de l'*script* el *shebang* `#!/usr/bin/awk -f` i donar-li permisos d'execució. Podem indicar-li el paràmetres directament.

```
cat countries_shebang

#!/usr/bin/awk -f
BEGIN {print "BEGIN. pais=",pais}
END {print "END. pais=",pais}

# Executa l'script i li indica el valor de país.
./countries_shebang pais="United Kigdom" sales.csv

BEGIN. pais=
END. pais= United Kingdom
```

Aquí podem veure que un dels problemes que hi ha a l'hora de gestionar els arguments amb AWK, és que no estan disponibles fins que no s'ha llegit la primera línia del fitxer a processar, és a dir, no es poden emprar a `BEGIN`. La solució passa per emprar l'**opció `-v` davant de cada paràmetre** que indiquem.

```
#
./countries_shebang -v pais="United Kingdom" sales.csv

BEGIN. pais= United Kingdom
END. pais= United Kingdom
```

Per acabar aquest apartat, cal mencionar que AWK, igual que C o Java, fa ús de les variables `ARGC` i `ARGV`. Per a entendre-les, cal saber què són els *arrays*.

Vegeu també

Tractarem les *arrays* a l'apart «*Arrays*» del present mòdul.

3. Control de flux

El **control de flux** fa referència a la possibilitat que algunes instruccions només s'executin si es dona una determinada condició o es repeteixi la seva execució mentre o fins que es doni una condició.

Més informació

Per a més informació, consulteu un manual de programació estructurada.

El control de flux amb AWK està totalment inspirat en el llenguatge C. Hi ha dos tipus de control de flux: les **bifurcacions** (el prototípic és `if-else`) i les **iteracions** amb `while`, `do-while` i `for`.

3.1. Bifurcació

En programació, no sempre ens interessa dur a terme totes les accions, sinó que molts cops ens interessa només si es compleix una o més condicions. La primera manera de fer és amb el **patrons**, ja que AWK permet limitar els procediments a executar només a aquella línia que compleix el patró.

```
# Mostra el número de línia que conté un dels països tractats.
cat bifurcacion_patron

/France/ {print "Linia", NR, "contiene France"}
/Spain/ {print "Linia", NR, "contiene Spain"}
```

```
awk -f bifurcacion_patron sales.csv

Linia 2 contiene France
Linia 4 contiene France
Linia 5 contiene France
Linia 7 contiene France
Linia 8 contiene Spain
Linia 9 contiene France
Linia 10 contiene Spain
Linia 11 contiene France
Linia 12 contiene France
Linia 13 contiene France
Linia 15 contiene France
```

Cal observar com les accions s'executen només quan es compleix el patró.

Una altra forma d'executar condicionalment les accions és amb `if` i, opcionalment, `else`. L'acció es durà a terme només si es compleix la condició. La sintaxi és:

```
if (expressió) { acció1 acció2 ...}
```

Amb l'`else` poden fer que si l'expressió de l'`if` no és certa, es duguin a terme una sèrie d'accions alternatives:

```
if (expressió) { acció1 acció2 ... } else
{ accio_alternativa1 accio_alternativa2 ... }
```

```
# Mostra els registres en què les ventes de l'any 2018 han baixat respecte del 2017.
# Si el cos només consta d'una instrucció i no hi else, no calen les claus.
awk 'BEGIN {FS=","} {if (NR!=1 && $7>$8) print $0}' sales.csv

6,Gill,Nelius,gnelius5@gravatar.com,Female,France,36.16,12.72,56.83
11,Liuka,Feedham,lfeedhama@bravesites.com,Female,France,68.41,49.67,55.89
13,Aylmer,Gabey,agabeyc@sphinn.com,Male,United Kingdom,49.23,19.6,74.39
14,Alexandr,Blackboro,ablackborod@spiegel.de,Male,France,49.21,47.91,93.8
```

Una altra forma d'execució opcional es fa amb l'operador condicional, que és l'interrogant. La sintaxi és:

```
expressió ? acció1 : acció2
```

Si l'expressió és certa (val 1), s'executa la primera acció, si és falsa (val 0), s'executa la segona. Aquesta forma no és gens recomanable si les expressions no són molt simples, ja que pot fer que el codi sigui molt difícil de seguir.

```
# Mostra els registres en què les vendes de l'any 2018 han baixat respecte del 2017.
# Si no han baixat escriu un missatge d'informació.
awk 'BEGIN {FS=","} {(NR!=1 && $7>$8) ? ms=$0 : ms="2018 more than 2017"; print ms}' sales.csv

2018 more than 2017
2018 more than 2017
2018 more than 2017
2018 more than 2017
2018 more than 2017
2018 more than 2017
6,Gill,Nelius,gnelius5@gravatar.com,Female,France,36.16,12.72,56.83
2018 more than 2017
2018 more than 2017
2018 more than 2017
2018 more than 2017
11,Liuka,Feedham,lfeedhama@bravesites.com,Female,France,68.41,49.67,55.89
2018 more than 2017
13,Aylmer,Gabey,agabeyc@sphinn.com,Male,United Kingdom,49.23,19.6,74.39
14,Alexandr,Blackboro,ablackborod@spiegel.de,Male,France,49.21,47.91,93.8
```

3.2. Iteracions

Les **iteracions** serveixen per repetir una o més accions segons una determinada condició.

AWK ja itera automàticament sobre cada registre, típicament sobre cada línia, però això no sempre és suficient i, per tant, es pot emprar `while`, `do-while` i `for`.

```
# While analitza una condició i mentre sigui certa s'executen les accions.
# Cal observar que si la condició no es compleix de principi, no s'executarà cap acció.
cat while

BEGIN {
while (i<4){
    print i
    i++
```

```
    }
}

awk -f while

1
2
3

# Do-while executa una acció i mentre la condició sigui certa, la segueix executant.
# És molt similar a l'anterior, però, en aquest cas, les accions s'executaran un cop almenys.
cat do_while

BEGIN {
do{
    print i
    i++
    }
while (i<4)
}

awk -f do_while

1
2
3
```

Els `for` serveixen per iterar un nombre determinat de cops. Quan s'usa el `for`, hi ha la iniciació d'una variable, la condició que marca quan s'ha de deixar d'executar el cos del bucle, i l'increment o decrement que s'ha de fer després de cada iteració.

```
echo "Today is a beautiful day" | awk '{for (i=1; i<=NF; i++) print $i}'

Today
is
a
beautiful
day
```

4. *Arrays*

Així com en una variable podem guardar un valor, en un *array* en podem guardar molts que, des del punt de vista lògic, han de tenir alguna relació entre si. Cada element de l'*array* s'assigna al seu índex, que amb AWK es posa dins de claudàtors. Així, si tenim l'*array* `any2018`, `any2018[1]` és el valor de l'element 1 de l'*array*.

AWK permet ***arrays* associatius**, en què l'índex no és un número sinó una cadena. Encara més, per a AWK tots els índex són cadenes, és a dir, `any2018[1]` és el mateix que `any2018["1"]`.

```
awk 'BEGIN {a[1]=23; print a["1"]}'

23
```

```
# Suma totes les vendes de l'any 2018. Guarda tots els valors de 2018 (el camp 7) en un array
# i després el recorre sumant els seus valors.
cat array2018

# Suma totes les vendes de l'any 2018.

BEGIN {FS=","} # Establim el separador a ,

# Excepte pel primer registre,
# el 7è. camp (corresponent a 2018) es
# guarda en un array anomenat year2018.
{if (NR!=1) year2018[NR-1]=$7}

# Recorre l'array i se sumen els seus elements.
END{

    for (i=0;i<NR;i++){
        sum+=year2018[i]
    }
    print sum

}

awk -f array2018 sales.csv

579.07
```

Per a crear *arrays*, amb AWK hi ha `split()` que és de gran utilitat. La funció admet tres arguments:

- la cadena original a separar,
- el nom de l'*array* i
- el separador.

Retorna la longitud de l'*array* generat, no l'*array*.

```
# Agafem tot la línia, separem per / i ho guardem a date.
echo "01/07/2019" | awk '{n=split($0,date,"/"); print n,"camps: dia",date[1],"de",date[2],
"del",date[3]}'
```

```
3 camps: dia 01 de 07 de 2019
```

```
# Considerem només el primer camp, separem per guió i ho guardem a date.
echo "06-08-2020, cotxe, 15.620€" | awk 'BEGIN {FS=","}
{n=split($1,date,"-"); print n,"camps: dia",date[1],"de",date[2],"del",date[3]} '

3 camps: dia 06 de 08 de 2020
```

Els elements d'un *array* es poden esborrar, amb `delete`.

```
# Considerem només el primer camp, separem per guió i ho guardem a date.
# Llavors esborrem el primer valor.
echo "06-08-2020, cotxe, 15.620€" | awk 'BEGIN {FS=","}
{n=split($1,date,"-");delete date[1];print n,"camps: dia",date[1],"de",date[2],"del",date[3]} '

3 camps: dia de 08 de 2020
```

Amb `in` podem saber si un valor és un **índex** en un *array*:

```
awk 'BEGIN {a[1]="a1"; a[2]="a2"; print ("1" in a); print (1 in a); print (3 in a)}'

1
1
0
```

Hi ha dues variables del sistema que són *arrays*: `ARGV` i `ENVIRON`. El primer guarda els arguments amb què s'ha cridat AWK i el segon guarda les variables del sistema (que es poden consultar introduint `env` a la *shell*).

```
# Mostra el valor de DISPLAY del sistema.
awk 'BEGIN {print ENVIRON["DISPLAY"]}'

:0.0
```

Així com a `ARGV` hi ha els valors indicats com a arguments, a `ARGC` hi ha el nombre d'arguments indicats. `ARGV[0]` és `"awk"` mateix. Per exemple:

```
awk 'BEGIN {
 for (i = 0; i < ARGC ; ++i) {
 print "ARGV ", i, " = ", ARGV[i]
 }
}' argu1 argu2 argu3

ARGV 0 = awk
ARGV 1 = argu1
ARGV 2 = argu2
ARGV 3 = argu3
```

Quan executem AWK des d'un fitxer, `ARGV` ignora `-f` i el nom del fitxer.

5. Funcions

5.1. Funcions predefinides

AWK consta de diverses funcions predefinides. Les **numèriques** són:

Funcions	Operacions
`cos(x)`	Retorna el cosinus d'x.
`sin(x)`	Retorna el sinus d'x.
`atan2(y,x)`	Retorna l'arctangent d'y/x.
`exp(x)`	Retorna e elevat a x.
`int(x)`	Retorna x amb enter truncat.
`log(x)`	Retorna el logaritme, amb base e, d'x.
`sqrt(x)`	Retorna l'arrel quadrada d'x.
`rand()`	Retorna un nombre pseudoaleatori de l'interval [0,1).
`srand(x)`	Estableix una llavor per nombres aleatoris. Si no s'especifica, s'empra l'hora, dia i any actual.

Per exemple:

```
# Defineix la llavor i tria dos nombres aleatoris, que estaran entre 0 i 1.
# Mostra els nombres i la seva suma.
awk 'BEGIN {srand(10); x=rand(); y=rand(); print "x=",x,", y=",y,", suma=",x+y}'

x= 0.255219 , y= 0.898883 , suma= 1.1541
```

A l'hora d'elegir els nombre aleatoris, no sempre els volem de l'interval per defecte. Llavors caldria fer una projecció sobre l'interval desitjat. Per exemple, suposem que volem nombres aleatoris entre -3, inclòs, i 5, exclòs. Llavors, com que el nou interval té una longitud de 8, hem de multiplicar per 8 i, al resultat, restar-li 3, que és l'origen.

```
# Elegeix nombres aleatoris d'entre 0 i 1 i els projecta sobre [-3,5).
awk 'BEGIN {srand(10); x=rand(); y=rand(); newx= x*8 -3 ; newy= y*8 -3; print "x=",x,",
newx=",newx"\ny=",y,", newy=",newy}'

x= 0.255219 , newx= -0.95825
y= 0.898883 , newy= 4.19106
```

Per a **manipular cadenes**, hi ha:

Cadenes	Operacions
`gsub(r,s,t)`	Substitueix s per cada aparició de l'expressió regular r a t. Si no s'especifica t, s'empra `$0`. Torna el nombre de substitucions fetes.
`sub(s,p,t)`	Igual que `gsub` però només substitueix la primera ocurrència.
`index(s,t)`	Torna la posició t a s o zero si no existeix.
`lenght(s)`	Torna la llargada d's o de `$0` si no s'especifica s.
`match(s,r)`	Torna la posició d's, on comença l'expressió regular r, o 0 si no existeix. Aquesta funció estableix el valor de dues variables `RSTART` i `RLENGTH` que són allà on comença la coincidència (igual que el valor retornat) i la llargada.
`split(s,a,sep)`	Construeix un *array* a, a partir d's emprant el separador `sep`. Si `sep` no s'explicita, s'empra el valor d'`FS`.
`printf(cadena,expr)`	Funciona de la mateixa manera que la funció homònima de C i altres llenguatges. Es posa un cadena de caràcters amb uns símbols especials on aniran els valors de les variables. Per exemple, `sprintf("Hola %s", user)`, mostrarà `"Hola Pepito"` si la variable `user` és igual a «Pepito».
`substr(s,p,n)`	Torna la subcadena que va des de la posició p fins a l'n. Si n no s'especifica, torna la subcadena fins al final.
`tolower(s)` `toupper(s)`	Torna una cadena tot en minúscules o tot en majúscules.

```
# Mostra la primera posició d'"AG".
echo "ATAGGCTAGTAA" | awk '{pos=index($0,"AG"); print pos}'

3

# Substitueix "AG" per "AA".
echo "ATAGGCTAGTAA" | awk '{print $0; mutation=gsub("AG","AA"); print $0}'

ATAGGCTAGTAA
ATAAGCTAATAA

# Mostra on comença la coincidència amb el patró ".CTA" i la seva longitud.
echo "ATAGGCTAGTAA" | awk '{pos=match($0,".CTA"); print pos, RLENGTH}'

5 4

# Mostra on comença i acaba la coincidència amb el patró ".CTA".
echo "ATAGGCTAGTAA" | awk '{pos=match($0,".CTA"); printf("Des de %d fins a %d", pos,pos+RLENGTH)}'

Des de 5 fins a 9

# Mostra el fragment que coincideix amb el patró ".CTA".
echo "ATAGGCTAGTAA" | awk '{pos=match($0,".CTA"); print substr($0,pos,RLENGTH)}'

GCTA


# Transforma tots els caràcters a minúscules.
echo "ATAGGCTAGTAA" | awk '{print tolower($0)}'

ataggctagtaa
```

5.2. Creació de funcions

La creació de funcions pròpies segueix la sintaxi de C. Es comença amb la paraula reservada `function`, el nom i després els paràmetres (opcionals) entre parèntesi. El cos de la funció es posa entre claus. Amb `return` (opcional) podem fer que la funció torni un determinat valor.

```
# En aquest fitxer hi ha definides dues funcions que tradueixen dues paraules de l'anglès
al castellà o al català.
cat function

# Script que tradueix unes determinades paraules de l'anglès al castellà i català.

# Tradueix al castellà.
function spanish(moment){
    result="ERROR Moment of the day unknown"
    if (moment=="day") {result="dia"}
    if (moment=="night") result="noche"
    return result
}
# Tradueix al català.
function catalan(moment){
    if (moment=="day") {return "dia"}
    if (moment=="night") return "nit"
    return "ERROR Moment of the day unknown"
}

# Main
{
sp=spanish($0)
ca=catalan($0)
printf ("%s in spanish is %s and in catalan, %s\n",$0,sp,ca)
}

# Fitxer on s'aplicarà l'script anterior.
cat moments.txt

day
night
afternoon

# Resultat d'aplicar l'script awk function a moments.txt
awk -f function moments.txt

day in spanish is día and in catalan, dia
night in spanish is noche and in catalan, nit
afternoon in spanish is ERROR Moment of the day unknown and in catalan, ERROR Moment of the day unknown
```

6. Lectura de fitxers amb `getline`

La comanda `getline`, sense cap altra valor, fa que es processi el registre següent i es deixin d'executar les accions posteriors a `getline`. Per exemple:

```
# Cada vegada que el primer camp és senar, s'executa getline i, per tant, no es fa res.
# Altrament, sí que s'executa el print.
awk 'BEGIN {FS=","} {if ($1%2==1) { getline}; print $0}' sales.csv

id,first_name,last_name,email,gender,country,2017,2018,2019
2,Vivianne,Feeney,vfeeney1@washingtonpost.com,Female,United Kingdom,10.22,51.01,49.88
4,Wakefield,Shepton,wshepton3@sphinn.com,Male,France,78.39,86.02,86.35
6,Gill,Nelius,gnelius5@gravatar.com,Female,France,36.16,12.72,56.83
8,Moss,Avo,mavo7@merriam-webster.com,Male,France,80.75,81.27,93.91
10,Sigismondo,Winterborne,swinterborne9@sciencedirect.com,Male,France,31.97,71.87,32.91
12,Tynan,Coen,tcoenb@businesswire.com,Male,France,6.92,10.44,34.3
14,Alexandr,Blackboro,ablackborod@spiegel.de,Male,France,49.21,47.91,93.8
```

`Getline` també s'utilitza **per a llegir altres fitxers**, a part del que s'aplica al propi *script*.

La sintaxi és `getline nom_variable < "nom_del_fitxer"`. En un exemple anterior hem vist com podíem traduir paraules concretes. És clar que el mètode emprat no és escalable, ja que les paraules equivalents les tenim en el mateix codi de les funcions. Amb l'ajuda de `getline` podem crear un fitxer amb parelles de paraules, llegir-lo des d'AWK i fer la traducció oportuna.

```
# Fitxer on hi ha parelles de paraules.
cat en-es.txt

day, día
night, noche
afternoon, tarde
morning, mañana
midday, mediodía
midnight, medianoche

# Script que cerca una paraula en un fitxer i retorna la seva parella.
cat translator

# Tradueix unes determinades paraules de l'anglès al castellà.

# Funció que tradueix MOMENT al castellà.
# Les parelles de paraules es guarden en un fitxer.
function translate(MOMENT){
    res="NOT FOUND" # Valor si no existeix la paraula
    while (getline words < "en-es.txt"){ # Llegeix el fitxer línia a línia
        split(words,word) # Posa els tokens en un array anomenat "word"
        if(word[1]==MOMENT) res=word[2] # Si el primer és igual al paràmetre, fet
    }
    close("en-es.txt") # Tanca el fitxer
    return res
}

# MAIN
{
print $0, translate($0)
}
```

```
awk -f translator moments.txt

day día
night noche
afternoon mediodía
```

7. El «problema» dels separadors

En molts fitxers csv, hi ha camps en què hi ha comes en el seu valor, és a dir, **comes que no són separadors de camps** sinó que formen part del propi camp. Per exemple, si un camp té la frase «uno, dos, tres», s'ha d'interpretar com un únic camp. Molts programes el que fan és posar entre cometes tots els camps de text per a evitar ambigüitats (i en els camps numèrics emprar el punt com a separador decimal). Observeu el fitxer csv següent:

```
cat separators.csv

3,"a good number, prime one",3b
1,"a really borring number",1b
2,"a couple is always a beautiful choice",2b
```

Amb el que hem vist fins ara, si volguéssim reordenar els camps, podríem intentar-ho amb:

```
awk 'BEGIN {FS=","} {print $3 " " $2,":",$1}' separators.csv

prime one" "a good number : 3
1b "a really borring number" : 1
2b "a couple is always a beautiful choice" : 2
```

Tal com s'ha explicat abans, aquest resultat no hauria de sorprendre. El segon camp s'ha partit en dos, ja que hi ha una coma.

La solució a aquest problema passa per canviar la manera com se separen els camps. Recordeu que amb `FS` triem el separador dels camps. Però ara el que s'ha de fer, en lloc de pensar en el separador, és crear una descripció, en forma d'expressió regular, de com són els camps i emprar la variable del sistema `FPAT`. El camps que ara tenim són de dues formes:

- Contingut entre cometes, format per una doble cometa, elements que no són doble cometa i doble cometa. L'expressió regular corresponent és `\"[^\"]+\"`.

- Contingut que no té comes, que és `[^,]+`.

Per tant, l'expressió regular que defineix com són els nostres camps és una o l'altra:

```
awk 'BEGIN {FPAT="(\"[^\"]+\")|([^,]+)"} {print $3 " " $2,":",$1}' separators.csv

3b "a good number, prime one" : 3
1b "a really borring number" : 1
2b "a couple is always a beautiful choice" : 2
```

8. Formatant la sortida

La intenció original dels creadors d'AWK va ser crear una eina per a fer informes de forma automàtica.

Amb molt poc d'esforç (i coneixement de `printf`) es poden fer documents amb una presentació força útil.

`printf`, a diferència de `print` no crea una nova línia automàticament després de la seva sortida:

```
awk 'BEGIN {print "one"; print "two"}'

one
two

awk 'BEGIN {printf "one"; printf "two"}'

onetwo
# Per a fer una nova línia, amb GNU/Linux, s'empra \n
awk 'BEGIN {printf "one\n"; printf "two"}'

one
two
```

La gran utilitat que té `printf` és que podem escriure uns símbols allà on volem que aparegui el valor d'una variable, fent així l'escriptura més còmode i més fàcil de mantenir. Allà on ha d'aparèixer el valor que volem, s'hi posa el símbol del tant per cent i una lletra que indica el tipus de valor que apareixerà: enter (d o i), cadena (s), etc.

printf/scanf

Podeu trobar un llistat dels símbols a «Códigos de formato de printf/scanf».

```
awk 'BEGIN {product="towels"; units="4"; printf("We need %d %s",units,product)}'

We need 4 towels
```

A part dels símbols, també es pot especificar l'amplada i l'alienació, entre el % i el símbol del tipus de valor. L'amplada s'estableix posant-hi el nombre de caràcters que es vol per a les cadenes. Per a limitar el nombre de decimals s'hi posa un punt just després del % i abans del símbol, per exemple:

```
awk 'BEGIN {units="4"; unit_price="3.987102"; printf("%d units cost %.2f",units,units*unit_price)}'

4 units cost 15.95
```

Per defecte, l'alineació és a la dreta. Per a canviar-ho, es posa un símbol menys (-) després del percentatge:

```
awk 'BEGIN {printf("Today I will see %-15s. ","John")}'

Today I will see John .

awk 'BEGIN {printf("Today I will see %15s. ","John")}'
```

```
Today I will see                                                                    John.
```

8.1. L'informe de vendes

Per a donar una idea de com fer un informe amb AWK, partirem del fitxer que hem anat fent servir i crearem un document per a poder analitzar millor la informació que hi tenim, mostrant la que ens interessa.

Farem una taula amb el nom del client, el que ha gastat cada any i la diferència amb l'any anterior. Després, farem les sumes per any.

A l'*script* hi ha dues funcions: una que imprimeix la capçalera i una altra que fa els càlculs apropiats i mostra la informació d'un client.

```
# Script awk per a fer l'informe.
cat informe

# Imprimeix la capçalera.
function print_head(){
    printf "\n\t\t\tSALES EVOLUTION 2017-2019\n\n"
    printf "Customer\t\t2017\t2018\tDif 2017\t2019\tDif 2018\n"
    printf "-----------------------------------------------------"
    printf "----------------------\n\n"
}

# Crea cada línia a partir del nombre i les ventes per any.
function process_customer(NAME, s2017, s2018, s2019){
    CUSTOMER_LENGTH=16 # Constant específica per al nombre de línies a la sortida.

    dif2017=s2018-s2017 # Diferència entre las vendes de l'any 2018 i 2017.
    difp2017=(dif2017/s2017)*100 # Diferència anterior, en percentatge.

    dif2018=s2019-s2018 # Diferència entre les vendes de 2019 i 2018.
    difp2018=(dif2018/s2018)*100 # Diferència anterior, en percentatge.

    # name, 2 2017, 3 2018, 4 dif2017, 5 2019, 6 dif 2019
    # Posa un o dos tabuladors segons la longitud del nombre.
    if (length(NAME)>CUSTOMER_LENGTH) printf ("%s\t",NAME)
    else printf ("%s\t\t",NAME)
    printf ("%.2f\t%.2f\t%.2f(%d%%)\t%.2f\t%.2f(%d%%)\n",s2017,s2018,dif2017,difp2017,s2019,
    dif2018,difp2018 )
}

BEGIN {FS=","; print_head()}

# MAIN
{
    if (NR!=1) { # Si no és la primera línia
        name=$2 " " $3
        process_customer(name,$7,$8,$9)
        t2017+=$7
        t2018+=$8
        t2019+=$9
    }


}

END {
    printf "-----------------------------------------------------"
    printf "----------------------\n"
    process_customer("Total:\t ",t2017,t2018,t2019) # NAME inclou un tabulador, per temes
    de presentació.
```

```
}
```

```
awk -f informe sales.csv

                              SALES EVOLUTION 2017-2019

Customer                    2017     2018    Dif 2017        2019    Dif 2018
-------------------------------------------------------------------------------------------

Lulu Chaloner              77.73    85.90    8.17(10%)      60.46    -25.44(-29%)
Vivianne Feeney            10.22    51.01    40.79(399%)    49.88    -1.13(-2%)
Waring Craythorne           3.70    35.21    31.51(851%)    72.35    37.14(105%)
Wakefield Shepton          78.39    86.02    7.63(9%)       86.35    0.33(0%)
Beale Myrtle               68.06    77.78    9.72(14%)      40.26    -37.52(-48%)
Gill Nelius                36.16    12.72    -23.44(-64%)   56.83    44.11(346%)
Kippy Marlen               10.58    69.81    59.23(559%)    50.30    -19.51(-27%)
Moss Avo                   80.75    81.27    0.52(0%)       93.91    12.64(15%)
Lona Garrison               7.74    74.90    67.16(867%)    14.06    -60.84(-81%)
Sigismondo Winterborne     31.97    71.87    39.90(124%)    32.91    -38.96(-54%)
Liuka Feedham              68.41    49.67    -18.74(-27%)   55.89    6.22(12%)
Tynan Coen                  6.92    10.44    3.52(50%)      34.30    23.86(228%)
Aylmer Gabey               49.23    19.60    -29.63(-60%)   74.39    54.79(279%)
Alexandr Blackboro         49.21    47.91    -1.30(-2%)     93.80    45.89(95%)
-------------------------------------------------------------------------------
Total:                    579.07   774.11    195.04(33%)    815.69    41.58(5%)
```

Bibliografia

Bibliografia bàsica

Dougherty, D.; Robbins, A. (1997). *Sed & Awk*. Massachusetts: O'Reilly Media.

Robbins, A. (2015). *Effective AWK Programming: Universal Text Processing and Pattern Matching* (4a. ed.). Massachusetts: O'Reilly Media.

Vidal Cortés, J. A. (2002). *El Lenguaje de Programación AWK/GAWK*. Madrid.

Referència

Es.cppreference.com (2012). «Códigos de formato de printf/scanf». cppreference.com. Disponible a: https://es.cppreference.com/w/cpp/io/c/printf_format.