AWK : le langage script de référence pour le traitement de fichiers

GNU/Linux Magazine
n°
Numéro
131

Mois de parution
octobre 2010

Auteurs
Par
Balima Damien

Lorsqu'il s'agit de l'utilisation d'un système UNIX, c'est un peu comme pour le cinéma : il faut connaître ses classiques. Grep, sed, vi, bash, etc., sont les incontournables et intemporels du genre, un peu comme les Lautner/Audiard du cinéma français. Awk est l'un de ces classiques...

1. Histoire d'Awk

Du nom de leurs auteurs Alfred Aho, Peter Weinberger et Brian Kernighan, Awk fut, en même temps que Sed, Bourne shell et tar, intégré dans la version 7 d'UNIX en 1979, par les laboratoires Bell. Awk fut par la suite continuellement intégré aux distributions d'UNIX avec, en 1985, une mise à jour importante de Awk qui donna le New Awk (ou Nawk). Par la suite, de nouvelles versions dérivées de Nawk apparurent, telles que Mawk (Mike's Awk), Gawk (Gnu Awk) ainsi que des versions

commerciales comme Motrice Kern Systems Awk (MKS Awk), Thompson Automation Awk (Tawk), Videosoft Awk (Vsawk), et d'autres versions plus spécifiques (Xgawk, Spawk, Jawk, Qtawk, Runawk).

Awk est depuis inclus dans la norme POSIX ; aussi, sur tout système UNIX qui se respecte, vous devriez trouver la commande **awk**. Pour vérifier la version de votre Awk, exécutez la commande suivante :

```
$ awk -W version
mawk 1.3.3 Nov 1996, Copyright (C) Michael D. Brennan
compiled limits:
max NF             32767
sprintf buffer      2040
```

Dans la suite de ce document, nous nous intéresserons à ce que Awk peut réaliser en tant que langage de script, dans les grandes lignes, afin de découvrir un autre type de langage assez exotique dans sa façon de procéder, mais qui offre des résultats remarquables, voire surprenants, et une bonne lisibilité, même pour un néophyte ; en espérant que vous serez également étonné par ce langage.

Les scripts de cet article sont testés sous mawk (version 1.3.3) et sous gawk (version 3.1.7).

2. Le langage awkien

2.1 Les concepts

Ce qui peut être déconcertant de prime abord avec Awk, ce sont ses concepts. Alors que dans la plupart des langages scripts, on dispose de la possibilité de faire à peu près tout et n'importe quoi, Awk dispose d'une structure de programmation qui évite certains aléas.

Pour illustrer ces propos, un concept purement awkien concerne l'ouverture des fichiers. En effet, bien qu'on puisse le faire, Awk se charge d'ouvrir les fichiers en lecture que vous mettez en paramètre, de les lire et de les fermer. D'autre part, ces fichiers ne sont pas altérés et en principe, on effectue une redirection en sortie des résultats affichés par les fonctions **print()** (ou **printf()**). Au moins un fichier en entrée est nécessaire pour lancer un script Awk. La procédure générale d'exécution est la suivante : placer en entrée les fichiers sources (ou connecter une sortie de commande par un tube), analyser les données par un script Awk et récupérer les résultats en sortie.

Ensuite, le langage Awk est conçu pour traiter les fichiers de données, et plus particulièrement les lignes de ces fichiers. Par défaut, Awk traite les lignes terminées par un retour-chariot et contenant des données séparées par un espace ou une tabulation. Ainsi, chaque ligne lue est traitée par le bloc principal d'instructions et un tableau interne contient les données séparées de la ligne en cours.

Cependant, certains fichiers ont des données regroupées sur plusieurs lignes plutôt que sur une seule. Dans ce cas, on parle d'enregistrement et l'on peut indiquer à Awk de traiter plusieurs lignes à la fois, jusqu'à la fin d'un enregistrement, en modifiant les variables **`FS`** (séparateur de champs) et **`RS`** (séparateur d'enregistrements).

2.2 La lecture

Les instructions concernent une condition liée à un enregistrement par blocs définis par des accolades. Ces blocs d'instructions sont exécutés (selon leurs conditions) durant la lecture du fichier, à chaque enregistrement. De plus, deux blocs spéciaux permettent d'ajouter des instructions à exécuter avant la lecture du premier enregistrement (le bloc **`BEGIN`**) et après la lecture du dernier enregistrement (le bloc **`END`**).

Lorsqu'un enregistrement est lu, il est placé dans la variable **`$0`** (à ne pas confondre avec la variable **`$0`** de Bash et Perl). L'enregistrement est ensuite scindé en plusieurs parties dans un tableau interne, allant de **`$1`** à **`$NF`**, en utilisant **`FS`** comme séparateur de champs (à l'instar d'un **`split()`** automatique). En lignes de commandes, la suite de commandes Awk est à placer entre apostrophes. Par exemple :

```
$ awk '{print "ligne",NR,": ",$0," [nombre de champs:",NF,"]
[premier champ:",$1,"] [dernier champ:",$NF,"]"}'
./fichier.txt

ligne 1 : ceci est un fichier [nombre de champs: 4 ]
[premier champ: ceci ] [dernier champ: fichier ]

ligne 2 : qu'il est bien [nombre de champs: 3 ] [premier
champ: qu'il ] [dernier champ: bien ]
```

On remarque une particularité des scripts Awk : peu d'efforts pour un maximum de réconfort. Une ligne de code suffit pour afficher chaque ligne d'un fichier, le numéro de ligne, le nombre de champs, les séparer et les afficher. Awk peut lire plusieurs fichiers à la suite en les plaçant les uns à la suite des autres, et utiliser un ou plusieurs scripts Awk au lieu de lignes de commandes en utilisant des options **`-f`**. Exemple :

```
$ awk -f script1.awk -f script2.awk fichier1 fichier2
fichier3
```

2.3 L'affichage

L'une des fonctions les plus utilisées est certainement la fonction **print**, qui permet l'affichage de texte et variables, en les séparant par une virgule. Les champs sont ensuite séparés par la valeur de la variable **OFS** (un espace par défaut, voir l'exemple ci-dessus).

Soit un fichier source **fichier.txt**, qui sera repris dans les exemples suivants :

```
bob;adresse;12 rue de la mouette

alice;adresse;3 avenue du goéland

charles;adresse;24 rue de l'albatros
```

La fonction **printf()**, de syntaxe similaire à celle de C,permet d'afficher les données, de formater la mise en page et l'affichage des nombres. Exemple dans un bloc final (**END**), pour récupérer le nombre de lignes parcourues :

```
$ awk 'END {printf("| %-20s | %-12s |\n| %-20s | %-12i
|\n","fichier",FILENAME,"nombre de lignes",FNR)}'
fichier.txt

| fichier                    | fichier.txt   |

| nombre de lignes     | 3               |
```

Les variables **%s** et **%i** sont respectivement remplacées par une chaîne de caractères et un entier. En ajoutant **%-20s**, on place cette valeur dans un espace de 20 caractères en commençant par la gauche (**%20s** pour la droite). Le restant est rempli par des espaces, ce qui, dans l'exemple ci-dessus, aligne les tubes.

Les commentaires sont à placer après un caractère croisillon (**#**).

Les scripts peuvent renvoyer un code retour de sortie par la fonction **exit <code retour>**, par défaut égal à 0, et des commandes Shell peuvent être exécutées par la fonction **system()**, qui retourne le code de sortie de la commande.

```
$ awk 'NR==1 { system("ls -l "FILENAME) } { print $0 }'
fichier.txt

-rw-r--r-- 1 dams dams 104 juil. 24 18:55 fichier.txt

bob;adresse;12 rue de la mouette

alice;adresse;3 avenue du goéland
```

```
charles;adresse;24 rue de l'albatros
```

2.4 Les variables

Pour bien awker, il faut connaître les variables de base, au moins **FS**, **NR** et **RS**, utiles dans la plupart des scripts.

Variable	Description
ARGC	Nombre d'arguments de la ligne de commandes
ARGV	Tableau des arguments de la ligne de commandes
CONVFMT	Format de conversion des nombres en string (chaîne de caractères)
ENVIRON	Tableau associatif des variables d'environnement
FILENAME	Nom du fichier courant (et son chemin si précisé)
FNR	Numéro de l'enregistrement parcouru dans le fichier courant
FS	Séparateur de champs (par défaut les espaces, tabulations et retours-chariots contigus **`[ \t\n]+`**)
NF	Nombre de champs de l'enregistrement courant
NR	Numéro de l'enregistrement parcouru (tous fichiers confondus)
OFMT	Format de sortie des nombres
OFS	Séparateur de champs en sortie (un espace)
ORS	Séparateur d'enregistrement en sortie (une nouvelle ligne)
RLENGTH	Longueur du string trouvé par la fonction **`match()`**
RS	Séparateur d'enregistrement (une nouvelle ligne)

RSTART	Première position du string trouvé par la fonction **match()**
SUBSEP	Caractère de séparation pour les routines internes des tableaux (**\034**)

Les variables personnelles n'ont besoin ni d'être déclarées, ni d'être initialisées dans les blocs principaux. Il n'y a pas non plus de type à déclarer, elles sont auto-typées (attention toutefois aux mélanges entiers/caractères lors des opérations). Les champs du fichier **fichier.txt** précédent sont séparés par un point-virgule, aussi la variable de séparation de champs, **FS**, doit être modifiée avant la lecture du fichier, dans le bloc **BEGIN**. Les autres blocs d'instructions, qui ne sont pas précédés de **BEGIN**, **END**, ou de conditions, sont exécutés pour chaque enregistrement.

```
$ awk 'BEGIN{FS=";"} {print $3}' fichier.txt

12 rue de la mouette

3 avenue du goéland

24 rue de l'albatros
```

2.5 Les paramètres

Les paramètres sont indiqués par leurs noms suivis d'un caractère **=** et de leurs valeurs, avant les noms des fichiers. Ils sont ensuite récupérés dans le script par leurs noms, sans caractère additionnel, après le bloc **BEGIN** (si défini). Par exemple, le script **test.awk** suivant (l'opérateur tilde est une condition de contenu) :

```
BEGIN{FS=";"; OFS=" : "}

$1 ~ nom {print $1,$3}
```

```
$ awk -f test.awk nom=alice fichier.txt

alice : 3 avenue du goéland
```

2.6 Les chaînes de caractères

En plus de fonctions de base, Awk dispose également de fonctions dédiées aux traitements des chaînes de caractères (ou *string* en anglais), facilitant ce genre d'opérations. La liste de ces fonctions est la suivante :

Fonction de string	**Description**

`gsub(exp,sub,str)`	Substitue globalement par la chaîne `sub` chaque expression régulière `exp` trouvée dans la chaîne `str`, et retourne le nombre de substitutions. Si `str` n'est pas indiquée, par défaut `$0` est utilisé.
`index(str,st)`	Retourne la position du string `st` dans la chaîne `str`, ou 0 si non trouvé.
`length(str)`	Retourne la longueur de la chaîne `str`. Si `str` n'est pas indiquée, par défaut `$0` est utilisé.
`match(str,exp)`	Retourne la position de l'expression régulière `exp` dans la chaîne `str`, ou 0 si non trouvé. Affecte les valeurs aux variables `RSTART` et `RLENGTH` (cf. 2.4 Les variables).
`split(str,tab,sep)`	Sépare la chaîne `str` en éléments dans un tableau `tab` et en utilisant le séparateur `sep`. Si `sep` n'est pas renseigné, `FS` est utilisée par défaut.
`sprintf("format",exp)`	Retourne une chaîne au lieu de l'affichage vers la sortie standard, contrairement à `printf()`.
`sub(exp,sub,str)`	Comme `gsub()`, mais ne substitue par `sub` que la première expression `exp` trouvée dans `str`.
`substr(str,pos,long)`	Retourne une partie du string `str` commençant à la position `pos` et de longueur `long`. Si `long` n'est pas indiqué, `substr()` utilise tout le reste de `str`.

`tolower(str)`	Met en minuscules toute la chaîne `str` et retourne la nouvelle chaîne.
`toupper(str)`	Met en majuscules toute la chaîne `str` et retourne la nouvelle chaîne.

La concaténation de chaîne de caractères s'effectue sans espace ni caractère spécifique.

2.7 Les fonctions mathématiques

Awk dispose également de fonctions dédiées aux traitements numériques. Celles-ci sont les suivantes :

Fonction mathématique	Description
`cos(r)`	Cosinus de l'angle *r* (*r* en radians)
`exp(x)`	Exponentiel de *x*
`int(x)`	Valeur entière de *x*
`log(x)`	Logarithme de *x*
`sin(r)`	Sinus de l'angle *r* (*r* en radians)
`sqrt(x)`	Racine carrée de *x*
`atan2(y,x)`	Arc tangente de *y/x*
`rand()`	Nombre pseudo-aléatoire compris entre 0 et 1
`srand(n)`	Réinitialise la fonction `rand()`

2.8 Les fonctions personnelles

Les fonctions personnelles peuvent être définies dans le script Awk, en dehors des blocs d'instructions, ou dans un autre script qui sera également appelé par l'option `-f <script>`.

Une fonction se définit par le mot-clé `function` suivi du nom de la fonction et de ses paramètres. Par exemple :

```
{
        x=$1
```

```
        a=$2
        y=magauss(x,a)
        print "magauss(",x,";",a,") =",y
}
function magauss(x,a){
        pi=3.1415927
        return (1/(pi*a^2))^(1/4) * exp(x) * exp(-x^2/(2*a
^2))
}
```

```
$ echo "12 3" | awk -f test.awk
magauss( 12 ; 3 ) = 23.6772
```

Un tableau multidimensionnel peut également être passé en argument d'une fonction comme un simple tableau. De plus, les modifications faites dans la fonction sur un tableau passé en argument les modifient hors de la fonction également (les tableaux sont passés par références, les variables par valeurs).

2.9 Les conditions

Les conditions concernant l'enregistrement en cours de lecture peuvent être, de préférence, placées avant le bloc d'instructions. Dans ce cas, on n'indique pas l'instruction par défaut **IF**. À l'intérieur des blocs, les conditions sont de syntaxe généralement commune dans les langages scripts :

```
if (expression) {
 instruction
 ...
} [else {
 instruction
 …
}]
```

Dans le cas d'une seule instruction, les accolades peuvent être évitées. Plusieurs instructions sur une même ligne doivent être séparées par un point-virgule. L'opérateur réduit de condition **?:** est également fourni (si l'expression est vraie, l'instruction 1 est exécutée, sinon l'instruction 2 est exécutée) :

```
expression ? instruction_1 : instruction_2
```

Dans le script **test.awk** suivant, si l'enregistrement contient (opérateur tilde) « bob » et si le quatrième champ est égal à « adresse », il affiche le cinquième champ, sinon, si le second est égal à « adresse », cela affiche le troisième champ. On modifie la variable **FS** pour séparer les champs selon le caractère point-virgule.

```
BEGIN{FS=";"}
$0 ~ "bob"{
 if ($4 == "adresse") print $5
 else if ($2 == "adresse") print $3
}
```

```
$ awk -f test.awk fichier.txt
12 rue de la mouette
```

2.10 Les boucles

Les boucles **while**, **do...while** et **for** sont disponibles.

```
while(condition){
 instruction
 ...
}
```

Dans la boucle **do...while**, du fait que la condition soit vérifiée après les instructions, celles-ci sont exécutées au moins une fois.

```
do{
 instruction
 ...
}while(condition)
```

La boucle **for** est assez classique dans sa syntaxe.

```
for (initialisation; condition; incrémentation){
 instructions
 …
}
```

Il existe aussi une variante de la boucle **for** pour les tableaux à simple dimension.

```
for (variable in tableau){
instructions
 …
}
```

De plus, les boucles peuvent être complétées par des instructions **break** pour en sortir et **next** pour passer à l'étape suivante.

Dans le script **test.awk** suivant, si l'enregistrement contient « bob » et si l'un des champs est égal à « adresse », il affiche le champ suivant. Si le champ « adresse » est à la fin, une ligne vide s'affiche à la place.

```
BEGIN{FS=";"}
$0 ~ "bob"{
 for (i=1; i<=NF; i++)
  if($i == "adresse") print $(i+1)
}
```

```
$ awk -f test.awk fichier.txt
12 rue de la mouette
```

2.11 Les tableaux

Comme les variables, les tableaux n'ont pas besoin d'être déclarés. De plus, les tableaux sont associatifs : une clé peut être associée à une valeur unique ; autrement dit, les tableaux peuvent être des vecteurs ou des maps. Ils se terminent par un chiffre 0. Les dimensions des tableaux multidimensionnels doivent être séparées par une virgule.

Les tableaux multidimensionnels associatifs peuvent être traités comme des tableaux à simple dimension séparée par la variable **SUBSEP** (avec une fonction **split()**, par exemple) dans les boucles **for**. Les conditions **if** peuvent être testées sur l'ensemble des dimensions entre parenthèses et avec le mot-clé **in**.

Dans les exemples ci-dessous, on utilise le fichier **fichier.txt** suivant :

```
bob;adresse;12 rue de la mouette ;ville;Igaluit
alice;adresse;3 avenue du goéland ;ville;Nuuk
```

```
charles;adresse;24 rue de l'albatros ;ville;Lima
```

On utilise les notions vues précédemment dans le script ci-dessous, avec un tableau à deux dimensions, deux paramètres et une fonction :

```
BEGIN{FS=";"}

{

          if (nom==0 || champ==0){

print "parametres nom=<nom> et champ=<champ> requis";

                  exit 1

         }

         remplir(tab)

}

function remplir(t){

         for(i=2;i<=NF;i++){

                  if($i=="adresse") t[$1,"adresse"]=$(i+1)

                  else if($i=="ville") t[$1,"ville"]=$(i+1)

}

}

END{

         if (nom!=0 && champ!=0)

  if( (nom,champ) in tab) print nom,champ,":",tab[nom,cham
p]

  else print nom,champ,"non trouvé"

}
```

```
$ awk -f test.awk nom="alice" champ="adresse" fichier.txt

alice adresse : 3 avenue du goéland

$ awk -f test.awk nom="bob" champ="ville" fichier.txt

bob ville : Igaluit
```

2.12 Les expressions régulières

Les expressions régulières permettent d'affiner les conditions en utilisant des méta-caractères spécifiques (*man 7 regex*). Il existe de nombreux documents traitant des expressions régulières, ne serait-ce que sur Internet, aussi nous n'en ferons qu'une brève description. On trouve fréquemment l'usage des caractères suivants : `^` pour signaler une position en début de ligne, $ pour une position en fin de ligne, `.` pour un caractère quelconque, les crochets `[` et `]` pour une plage de caractères, `+` pour indiquer au moins un caractère défini précédemment, et `*` pour un nombre quelconque ou nul de caractères. Sous Awk, ces expressions régulières, à la norme POSIX, sont à placer entre deux slashs `/` et ne prennent pas en compte les variables (ce qui peut être assez contraignant).

3. Démonstration

Un cas d'utilisation pratique est d'afficher les informations relatives aux processeurs et à la mémoire sous une page web. On peut utiliser tout un tas d'outils, de programmes et de langages pour cela ; voyons comment on pourrait le faire avec Awk.

Les fichiers sont à placer en derniers paramètres et peuvent être traités différemment par les variables **`FILENAME`** et **`FNR`**. De plus, plusieurs scripts Awk peuvent être appelés par des options **`-f <script 1> [-f <script 2> ...]`**. Dans notre cas, ces fichiers sources sont les fichiers virtuels **`/proc/cpuinfo`** et **`/proc/meminfo`**. On nomme le script **`infoproc.awk`** et l'on redirige la sortie vers un fichier **`infoproc.html`** pour pouvoir l'afficher dans un navigateur web.

L'en-tête et la fin de page HTML sont codés dans le bloc **`BEGIN`** (qui est lu avant les fichiers sources) et **`END`** (lu à la fin), de façon à les afficher.

Le fichier virtuel **`/proc/cpuinfo`** contient des enregistrements (les informations d'un cpu) séparés par une ligne vide. Chaque champ des enregistrements est défini sur une ligne. Pour ce fichier, la variable de séparation d'enregistrement **`RS`** est donc égale à un saut de ligne(**`\n\n`**) et la variable de séparation des champs **`FS`** est égale à un retour-chariot (**`\n`**). Le fichier virtuel **`/proc/meminfo`** contient un seul enregistrement (la mémoire) et les champs sont séparés par un retour-chariot, aussi les variables **`RS`** et **`FS`** n'ont pas à être modifiées entre ces fichiers.

Le script pourrait être le suivant :

```
BEGIN{
```

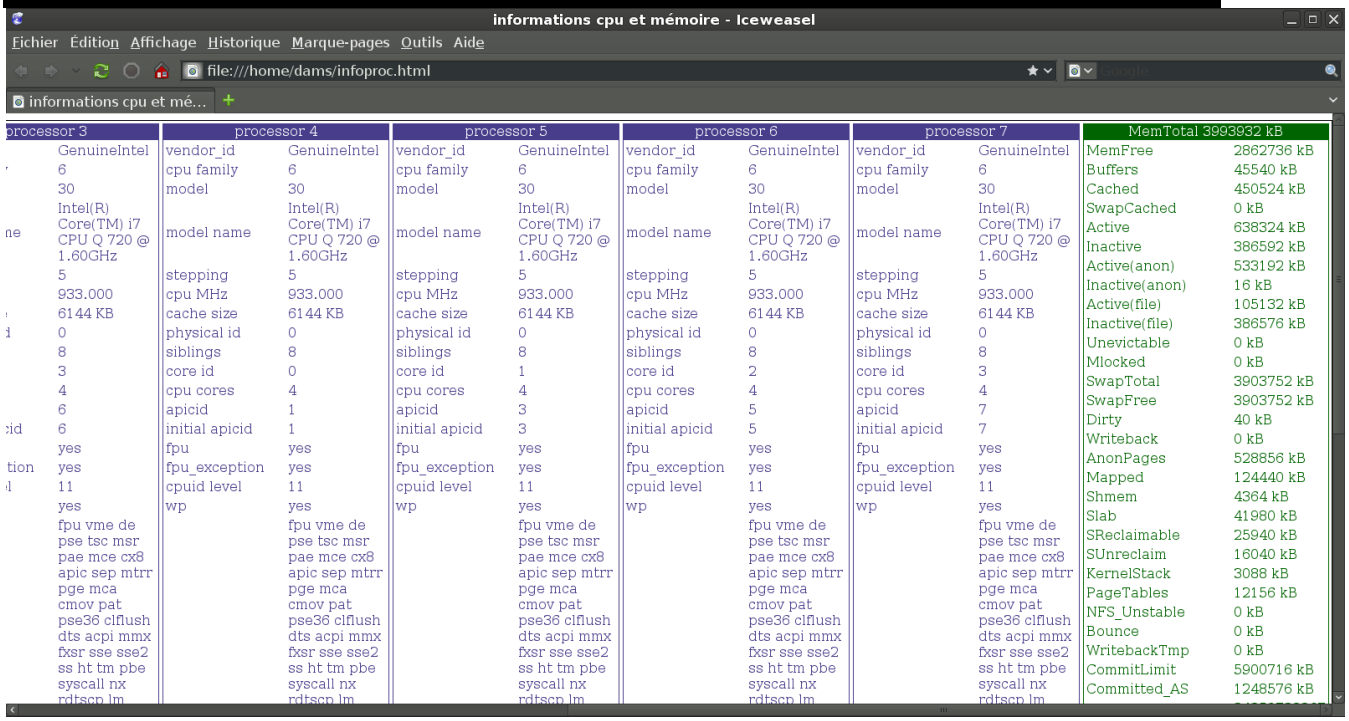
```
        RS="\n\n"       #separateur d'enregistrement

        FS="\n"         #separateur de ligne

        #en-tete et debut de page HTML

        print "<html>"

        print "<head>"

print "<title>informations cpu et m&eacute;moire</title>"

        print "<style type=\"text/css\">"

        print "table {border: solid thin; padding 10px; ma
rgin 5px}"

        print "table.proc {color: DarkSlateBlue; border-co
lor: DarkSlateBlue}"

        print "table.proc caption {color: white; backgroun
d: DarkSlateBlue; text-align: center}"

        print "table.mem {color: DarkGreen; border-color:
DarkGreen}"

        print "table.mem caption {color: white; background
: DarkGreen; text-align: center}"

        print "</style>"

        print "</head>"

        print "<body>"

        print "<table><tr>"

}

FILENAME ~ /cpuinfo$/ { print "<td valign=\"top\"><table c
lass=\"proc\">"}

FILENAME ~ /meminfo$/ { print "<td valign=\"top\"><table c
lass=\"mem\">"}

{

        for(i=1; i<=NF; i++){

                split($i,cpu,":")

                if(i==1) print "<caption>", cpu[1], cpu[2]
, "</caption>"
```

```
                else print "<tr><td>", cpu[1], "</td><td>"
, cpu[2], "</td></tr>"
}
        print "</table></td>"
}
END{
        #fin de page HTML
        print "</tr></table>"
        print "</body>"
print "</html>"
}
```

34 lignes de code, commentaires et mise en page inclus, suffisent pour réaliser ce script, tout en étant lisible, clair et rapide à écrire.

On obtient le résultat suivant :

```
$ awk -f infoproc.awk /proc/cpuinfo /proc/meminfo >
infoproc.html; firefox infoproc.html
```



Conclusion

Disposant d'une syntaxe souple, robuste et compréhensible, le langage Awk a ainsi l'avantage d'être rapide à utiliser pour des traitements efficaces et des scripts allégés en lignes de code.

Comparé aux langages scripts plus récents, comme Perl, Php, Python ou Ruby, on peut cependant regretter que le langage Awk n'ait pas de bibliothèques supplémentaires, ni de module Apache, ce qui le limite – dans ses versions officielles – au traitement de fichier. Cependant, associé à un interpréteur de commandes comme Bash, les possibilités s'étendent et il devient possible, par exemple, d'effectuer des traitements de bases de données en récupérant la sortie des requêtes au préalable avant de la rediriger vers un script Awk.

Ainsi, Awk, de par sa relative simplicité et son efficacité, convient parfaitement pour l'écriture de « moulinettes », connecteurs et autres briques logicielles liées au traitement de données.

Pour en savoir plus

On trouve assez peu de livres traitant de Awk, comparé à ceux consacrés aux langages scripts plus récents. Cependant, *sed & awk*, deuxième édition, des auteurs Dale Dougherty et Arnold Robbins, est complet tout en étant didactique, aux éditions O'Reilly.

On trouve en ligne également le guide d'utilisation de gawk : *The GNU Awk User's Guide*, 3ème édition du livre *GAWK: Effective AWK Programming*, à l'URL http://www.gnu.org/manual/gawk/.

Un portail d'informations est aussi disponible à l'URL http://awk.info.

Le manuel (**`man awk`**) et la page d'information (**`info awk`**) sont également bien fournis et détaillés.