Commande awk

Table of Contents

La commande **awk** permet d'appliquer un certain nombre d'actions sur un fichier. La syntaxe est inspirée du C

syntaxe

```
awk [-Fs] [-v variable] [-f fichier de commandes] 'program' fichier
```

- **-F** Spécifie les séparateurs de champs
- **-v** Définie une variable utilisée à l'intérieur du programme.
- **-f** Les commandes sont lu à partir d'un fichier.

Le **program** est une suite d'action de la forme : motif { action } , le motif permet de determiner sur quels enregistrements est appliquée l'action. Si le motif existe dans l'enregistrement, l'action sera appliquée à la ligne .

Le motif peut etre :
* une expression régulière
`/expression regulière/`
`$0 ~ /expression regulière/`
`expression ~ /expression regulière/`
`expression !~ /expression regulière/`
* une expression BEGIN ou END
* une expression de comparaison: <, <=, == , !=, >=, >
* une combinaison des trois (à l'aide des opérateurs booléens || ou, && et, ! négation)
* une caractérisation des lignes
* motif1,motif2 : chaque ligne entre la premiere ligne correspondant au motif1 et la première ligne correspondant au motif2

- Par exemple, pour vérifier certains enregistrements du fichier `/etc/passwd`:

```
 awk 'BEGIN { print "Verification du fichier /etc/passwd pour ...";
              print "- les utilisateurs avec UID = 0 " ;
              print "- les utilisateurs avec UID >= 60000" ;
              FS=":"}
      $3 == 0 { print "UID 0 ligne "NR" :\n"$0 }
      $3 >= 60000  { print "UID >= 60000 ligne "NR" :\n"$0 }
      END   { print "Fin" }
' /etc/passwd
```

Résultat :

```
Verification du fichier /etc/passwd pour ...
- les utilisateurs avec UID = 0
- les utilisateurs avec UID >= 60000
UID 0 ligne 5 :
root:*:0:b:administrateur:/:/bin/sh
UID >= 60000 ligne 14 :
clown:*:61000:b:utilisateur en erreur:/home/clown:/bin/sh
Fin
```

- Par exemple, pour vérifier le libellé du groupe 20 dans `/etc/group`:

```
 awk 'BEGIN { print "Verification du fichier /etc/group";
              print "le groupe 20 s'appelle t-il bien users ? " ;
              FS=":"}
      $1 == "users" && $3 ==20 { print "groupe "$1" a le GID "$3" !" }
      END   { print "Fin" }
' /etc/group
```

Résultat :

```
Verification du fichier /etc/group
le groupe 20 s'appelle t-il bien users ?
groupe users a le GID 20 !
Fin
```

- Par exemple, pour afficher un interval de lignes avec leur numéro:

```
 awk 'NR == 5 , NR == 10 {print NR" : " $0 }' fichier
```

Imprime de la ligne 5 à la ligne 10 , chaque ligne précédée par son numéro

La lecture

Les instructions sont des 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 ]
```

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`

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).

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 (**#**).
Une action vide est représenté par **;**

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
```

L'instruction suivante est identique à print mais en utilisant un format

```
printf format , exp, exp` ou `printf (format,exp , exp )
```

Un format est une chaine de caractères et des constructeurs commencant par %

specifieur	**signification**
d	nombre decimal
s	chaine de caractères
specifieur	signification
-	expression justifiée à gauche
largeur	largeur d'affichage
.precision	longueur maximale d'une chaine de caracteres ou nombre de decimales

La sortie d'un print ou d'un printf peut être redirigée dans un fichier ou sur un pipe

Les noms de fichiers doivent être entre guillemets sinon ils sont considérés comme des variables

Exemple: le fichier fich.numerote contient le fichier fichier avec les lignes numérotées

```
awk ' { print NR " :" , $0 > "fich.numerote" } ' fichier
```

Exemple: le fichier fich.numerote contient le fichier fichier avec les lignes numérotées sur 3 caractères

```
awk ' { printf "%3d : %s " , NR , $0 > "fich.numerote" } ' fichier |
```

Les variables et expressions

Les variables prédéfinies

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

Variable	Signification	Valeur par défaut
ARGC	Nombre d'arguments de la ligne de commande	-
ARGV	tableau des arguments de la ligne de commnde	-
FILENAME	nom du fichier sur lequel on applique les commandes	-
FNR	Nombre d'enregistrements du fichier	-
FS	separateur de champs en entrée	“ ”
NF	nombre de champs de l'enregistrement courant	-
NR	nombre d'enregistrements deja lu	-
OFMT	format de sortie des nombres	“%.6g”
OFS	separateur de champs pour la sortie	“ ”
ORS	separateur d'enregistrement pour la sortie	“\n”
RLENGTH	longueur de la chaine trouvée	-
RS	separateur d'enregistrement en entrée	“\n”
RSTART	debut de la chaine trouvée	-
SUBSEP	separateur de subscript	“\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
```

Les opérations et affectations arithmétiques

- Les opérateurs arithmétiques sont les opérations usuelles : + - * / % (reste division entière) et ^ (puissance). Tous les calculs sont effectués en virgule flottante.
- La syntaxe de l'affectation : var = expression
- On peut aussi utiliser les operateurs +=, -=, *=, /=, %= et ^= (x+=y équivaut à x=x+y)

Les variables de champs

Rappel : Les champs de la ligne courant sont : $1, $2, ..., $NF

La ligne entière est $0

Ces variables ont les memes propriétés que Les autres variables. Elles peuvent etre reaffectées. Quand $0 est modifiées, les variables $1,$2 ... sont aussi modifiées ainsi que NF.Inversement si une des variables $i est modifiées, $0 est mise à jour.

Les champs peuvent etre specifiés par des expressions, comme $(NF-1) pour l'avant dernier champs.

Par exemple pour créer un nouveau fichier de mot de passe `/etc/passwd.new` en remplacant le shell `/bin/ksh` par `/bin/posix/sh`

```
awk 'BEGIN { FS=":" ;
             OFS=":"}
     $NF != "/bin/ksh" { print $0 }
     $3 == "/bin/ksh" && NF == 7  { $7 = "/bin/posix/sh" ;
                                    print $0 } '
     /etc/passwd > /etc/passwd.new
```

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
```

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 :

Nom des fonctions	signification
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.

Exemple pour numéroter les lignes du fichier

```
 awk '{ print NR " : " $0 }' fichier
```

Exemple pour afficher les actions executées lors du passage à l'état 2

```
 awk 'BEGIN { FS=":" ;
              OFS=":" ;
              print " Run Level 2 : Liste des actions "}
      $2 ~ /2/  { print "Keyword <<"$3">>, \n Tache <<"$4">>" }
      $2 == ""  { print "Keyword <<"$3">>, \n Tache <<"$4">>" }
' /etc/inittab > /etc/passwd.new
```

Les fonctions mathématiques

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

Fonction mathématique	Description
cos®	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®	Sinus de l'angle r (r en radians)

Fonction mathématique	Description
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()

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).

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
```

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
```

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. On peut utiliser des tableaux de chaines de caractères et de nombres à une dimension:

- Il n'est pas nécessaire de les déclarer.
- La valeur par défaut est “” ou 0.
- Les indices sont des chaines de caractères.

for et les tableaux

Comme les indices des tableaux sont des chaines de caractères, on ne peut pas determiner la taille d'un tableau. On doit donc utiliser la construction :

```
for (var in tableau)
```

Par exemple pour construire un index de cross reference:

```
awk ' NF > 0 {
for (i=1;i<=NF;i++) {
if ( $i ~ /^[0-9a-zA-Z][0-9a-zA-Z]*$/ ) {
index[$i] = index[$i] ";" NR "," i " " ;
index["total"]++ ;
}
}
}
END {
for ( x in index ) {
if ( x != "total" )
printf("%-20s\t%s\n",x,index[x]) | "sort -f "
}
x="total";
printf("%s mots detectés = %d\n",x,index[x]);
} ' fichier
```

simulations des tableau multidimensions: On ne peut pas utiliser des tableaux multidimentionnel, mais on peut les simuler en concatenant des composants des sous chaines avec le séparateur **SUBSEP**

Par exemple, pour afficher le fichier en commencant par la derniere ligne:

```
awk 'BEGIN { print "Mémorisation de votre fichier " FILENAME
SUBSEP=":"
}
{ for ( i=1 ; i <=NF ; i++ ) {
memfields[ NR , i ] = $i
}
}
END { for ( i in memfields ) {
print i ":" memfields[i] | "sort -n -t: "
}
print "Fin"
} ' fichier
```

Les expressions régulières

Les expressions régulières permettent d'affiner les conditions en utilisant des méta-caractères spécifiques (man 7 regex). 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).

Par exemple, pour vérifier les UID et GID dans le fichier `/etc/passwd`:

```
 awk 'BEGIN { print "Verification des UID et GID dans le fichier
/etc/passwd";
              FS=":"}
      $3 !~ /^[0-9][0-9]*$/ {print "UID  erreur ligne "NR" :\n"$0 }
      $4 !~ /^[0-9][0-9]*$/ {print "GID  erreur ligne "NR" :\n"$0 }
      END   { print "Fin" }
' /etc/passwd
```

Résultat :

```
Verification des UID et GID dans le fichier /etc/passwd
UID erreur ligne 14 :
clown:*:aaa:b:utilisateur en erreur:/home/clown:/bin:sh
GID erreur ligne 14 :
clown:*:aaa:b:utilisateur en erreur:/home/clown:/bin/sh
Fin
```