

# Programme awk-----Introduction rapide

## Introduction

### *Historique*

Le nom de **awk** vient de ses trois concepteurs : Alfred V. Aho, Peter J. Weinberger et Brian W. Kernighan. La version originale fut créée en 1977 dans les laboratoires AT & TBell.

En 1985, une nouvelle version rend le langage plus puissant et implémente la possibilité de créer des fonctions.

En 1986, Paul Rubin et Jay Fenlason implémentent la version GNU de awk, avec les contributions de Richard Stallman et John Woods. C'est la naissance de **gawk**.

En 1988 et 1989, David Trueman et Arnold Robbins s'occupent de la compatibilité de awk/gawk jusqu'à aujourd'hui. Ils sont également responsables du suivi des bugs, des améliorations de performance, du support des standards et de l'implémentation de nouvelles propriétés.

### *Utilisations*

Le langage awk est très efficace dans la gestion de fichiers avec récupération d'informations et transformation des données. Il permet :

- la manipulation de fichiers texte en tous genres
  - le support de petites bases de données au format personnel
  - la génération de rapports
  - la validation/les tests de données
  - la production d'index et la conversion de documents
  - la création de scripts CGI
- etc.

### *Caractéristiques*

Une grande partie de la syntaxe a été empruntée au C : on dispose de fonction, d'instructions de contrôle (for, if...else, etc.). Malgré cela, awk reste un langage non typé, ce qui permet une plus grande souplesse.

### *Vocabulaire*

- Arrangement : détermine sur quels enregistrements doit être appliquée l'action. Synonyme : motif.
- Enregistrements : chaînes de caractères séparées par des retours chariots. En général, ce sont les lignes.
- Champs : chaînes de caractères séparées par des espaces. En général, ce sont les mots.

### *Principe*

Awk est en gros un langage de traitement de tableaux. Autrement dit, il est dédié à des

informations qui peuvent être groupées en enregistrements et en champs.

Le programme `awk` explore un ou plusieurs fichiers d'entrée à la recherche d'arrangements et il exécute des *actions* sur les enregistrements comportant ces arrangements.

Les arrangements-actions ont la forme :

*arrangement {action(s)}*

## Syntaxe

La syntaxe de `awk` est la suivante :

```
awk [-F] [-v var=valeur] 'programme' fichier
```

ou

```
awk [-F] [-v var=valeur] -f fichier-commande fichier
```

- L'argument `-F` doit être suivi du séparateur de champ (ici, `'|'`) :

```
-F:
```

- L'argument `-f` est suivi du nom du fichier qui contient les lignes de commandes. Cela évite de tout écrire entre les guillemets simples.

- L'argument `-v` définit une variable ('var' dans l'exemple) qui sera utilisée par la suite dans le programme.

Un programme `awk` possède la structure suivante: `arrangement {action}`. Quand il n'y a pas de critère c'est que l'action s'applique à toutes les lignes du fichier.

## Utilisation

Considérons, à titre d'exemple, le fichier de données `exemple.txt` :

```
helene 56 edu hcyr@sun.com
jean 32 ri jeanc@inexpress.net
julie 22 adm juliem@sympatico.ca
michel 24 inf michel@uqo.ca
richard 25 inf rcaron@videotron.ca
```

Si on tape :

```
$awk '{print}' exemple.txt
```

Le contenu du fichier exemple.txt s'affiche à l'écran.

La même chose, écrite différemment donnerait :

```
$awk '{print $0}' exemple.txt
```

La variable \$0 représente un enregistrement.

Il existe aussi \$1, \$2, ..., \$NF représentant chaque champ de l'enregistrement courant.

Le symbole \$ est utilisé comme spécificateur de colonne.

NF est le nombre de champs de l'enregistrement.

## Les variables prédéfinies

ARGC	Nombre d'arguments de la ligne de commande
ARGV	Tableau (de ARGV[0] à ARGV[ARGC]) des arguments de la ligne de commande
FILENAME	Nom du fichier sur lequel on applique les actions
FNR	Nombre d'enregistrements du fichier
NF	Nombre de champs de l'enregistrement courant
NR	Nombre d'enregistrements déjà lus
FS	Séparateur de champs (par défaut, un espace)
RS	Séparateur d'enregistrements (par défaut, '\n')
RLENGTH	Longueur de la chaîne trouvée
RSTART	Début de la chaîne trouvée

La commande awk suivante exécutée sur le fichier ci-dessus affiche le premier champ. Comme aucun arrangement n'a été spécifié, l'action {print \$1} s'applique à tous les enregistrements du fichier :

```
$awk '{print $1}' exemple.txt
```

```
helene  
jean  
julie  
michel  
richard
```

Notez la présence d'apostrophes pour protéger le programme de toute interprétation que pourrait faire la *shell* de certains caractères spéciaux.

L'arrangement `NR > 1` pourrait être spécifié pour sauter le premier enregistrement :

```
$awk 'NR>1 {print $1}' exemple.txt
```

```
jean  
julie  
michel  
richard
```

Le programme **awk** pourrait aussi afficher les deux premiers champs (champ 2, puis champ 1) et comporter une comparaison pour limiter la sélection aux seuls enregistrements dont le second champ est supérieur à 24 :

```
$awk '{if($2>24) print $2, $1}' exemple.txt
```

```
56 helene  
32 jean  
25 richard
```

Nous aurions pu obtenir le même résultat en spécifiant l'arrangement `$2 > 24`, au lieu d'inclure la commande `if` :

```
$awk '$2>24 {print $2, $1}' exemple.txt
```

Au lieu de la commande **print**, nous pourrions utiliser la commande **printf** qui permet le formatage. Elle fonctionne comme son équivalent dans le langage C. Les masques de formatage de la commande **printf** `%-10s` et `%6d` signifient respectivement

d'afficher une chaîne de caractères sur 10 colonnes et de la justifier à gauche, puis d'afficher un champ numérique sur 6 colonnes (justifié à droite par défaut) :

```
$awk '{if($2>24) printf("%-10s %6d\n", $1, $2)}'
exemple.txt
```

```
helene      56
jean        32
richard     25
```

## Les fonctions de traitement de chaîne

Il existe des fonctions préexistantes qui permettent de faire certains traitements sur les chaîne :

substr(s,p)	Extrait le morceau de la chaîne s commençant à la position p → Renvoie la sous-chaîne en question
substr(s,p,n)	Extrait le morceau de la chaîne s commençant à la position p et de longueur n → Renvoie la sous-chaîne en question
index(s,t)	Recherche la sous-chaîne t dans la chaîne s → Renvoie la position de la première occurrence de t, dans s, et 0 s'il n'y en a aucune
length(s)	Longueur d'une chaîne → Renvoie la longueur
match(s,n)	Effectue un test de reconnaissance du motif r sur la chaîne s → si s est reconnue, renvoie la position du premier caractère associé à la reconnaissance → sinon, renvoie 0
sub(r,s)	Remplace la première occurrence de r par s dans \$0 → Renvoie le nombre de substitution : 0 ou 1
sub(r,s,t)	Remplace la première occurrence de r par s dans la chaîne t → Renvoie le nombre de substitution : 0 ou 1

gsub(r,s)	Remplace toutes les occurrences de r par s dans \$0 → Renvoie le nombre de substitution : de 0 à n
gsub(r,s,t)	Remplace toutes les occurrences de r par s dans la chaîne t → Renvoie le nombre de substitution : de 0 à n

## Les blocs END et BEGIN

Normalement, awk exécute chaque bloc (délimités par des `{}`) de votre script une seule fois. Mais certaines situations nécessitent d'initialiser du code avant que awk ne parcourt le fichier en entrée. C'est à ça que sert le bloc BEGIN. Cette partie permet, par exemple, d'initialiser des variables, ou ouvrir des fichiers auxiliaires.

Dans le même ordre d'idée, le bloc END permet d'exécuter des commandes après que le fichier a été parcouru. Typiquement, c'est l'endroit idéal pour fermer les fichiers auxiliaires ouverts précédemment ou réaliser des statistiques. Bref, tout ce qui ne peut se faire qu'une fois toutes les données connues et traitées.

Grâce à ces blocs, nous pouvons ajouter un titre à notre liste en utilisant l'arrangement BEGIN pour lequel l'action s'applique avant la lecture des enregistrements (*la commande **awk** de cet exemple a été décomposée en plusieurs lignes ici afin de mieux l'illustrer, mais elle doit être entrée sur une même ligne*) :

```
$awk 'BEGIN {printf("Nom\t\tNuméro\n-----\n")}
      NR>=1 {if($2>24){printf("%-10s %6d\n", $1, $2)}}'
exemple.txt
```

Nom	Numéro
-----	
helene	56
jean	32
richard	25

Il ne reste plus qu'à faire le total des valeurs contenues dans le second champ et de les afficher à la fin. L'arrangement END s'applique à la fin de la lecture de tous les enregistrements .

Le programme de awk, devenant un peu long, peut être enregistré dans un fichier et son nom passé en paramètre en utilisant l'option **-f**. Si le programme est enregistré dans fichier liste.awk comme suit :

```

BEGIN {
    printf("Nom\t\tNuméro\n-----\n");
    total = 0;
}
NR>=1 {
    if($2>24) {
        printf("%-10s %6d\n", $1, $2);
        total += $2;
    }
}
END {
    printf("\t\t-----\n\t\t%6d\n", total)
}

```

La commande awk suivante faisant appel à ce fichier produirait le même résultat :

```
$awk -f liste.awk exemple.txt
```

```

Nom          Numéro
-----
helene       56
jean         32
richard      25
-----
              113

```

## Autres instructions du programme awk

Le langage de awk comporte les instructions suivantes :

```

if(condition) instruction [else instruction]

while(condition) instruction

for(expression; condition; expression) instruction

break

```

```
continue
# commentaires

next # saute les arrangements résiduels de cet
      # enregistrement

exit # saute le reste des enregistrements
```

Les expressions sont des entiers ou des chaînes de caractères et sont assemblées par les opérateurs +, -, \*, /, % et la concaténation (un espace). Les opérateurs de C: ++ --, +=, -=, \*=, /= et %= sont également valides.

Les variables peuvent être des scalaires, des tableaux (notation a[i]) ou des champs (\$1, \$2, etc.). Les constantes de chaînes de caractères sont délimitées par des guillemets ("").

La fonction length donne la longueur de l'argument qui lui est présenté ou de toute la ligne s'il n'y a pas d'argument.

## Particularité du langage

Une instruction ne doit pas forcément se finir par un ';' si elle est seule sur la ligne, mais obligatoirement dans le cas contraire (plusieurs instructions sur la même ligne) :

```
printf("%-10s %6d\n", $1, $2); total += $2;
```

mais

```
printf("\t\t-----\n\t\t%6d\n", total)
```

## Les fonctions

Voici un exemple très simple :

```
function Bonjour(Ajout) {
    print"Bonjour " Ajout
}

# Appel de la fonction
BEGIN {
    Bonjour("tout le monde")
}
```

```
}
```

## Conclusions

Awk n'est certainement pas aussi puissant que d'autres outils conçus dans des buts similaires, mais il a le grand avantage d'avoir la possibilité d'écrire de petits programmes adaptés à nos besoins, en très peu de temps.

Awk est très bien adapté pour les raisons qui ont conduit à sa construction : lire des lignes et agir en fonction de leur contenu.

Des fichiers comme `/etc/password` s'avèrent idéaux pour être traités et reformatés par Awk.

Mais le langage possède ses limites :

- il ne contient pas de débogueur
- la gestion des commandes externes est peu sûre ainsi que l'exploitation des codes retours et des erreurs.

## Ressources

- Sed & Awk (Nutshell handbook) par Dale Dougherty
- la FAQ : [www.faqs.org/faqs/computer-lang/awk/faq/index.html](http://www.faqs.org/faqs/computer-lang/awk/faq/index.html)
- §- un tutoriel : [sparky.rice.edu/~hartigan/awk.html](http://sparky.rice.edu/~hartigan/awk.html)
- le guide utilisateur de gawk : [www.gnu.org/software/gawk/manual/gawk.html](http://www.gnu.org/software/gawk/manual/gawk.html)
- et bien sûr : Google !