

COMMENT UTILISER LA COMMANDE AWK AVEC DES EXEMPLES

AWK est une commande Linux puissante et langage de script à usage général conçu pour le traitement de texte avancé. Il est principalement utilisé comme outil de rapport et d'analyse. Il prend les données d'entrée, le transforme et envoie le résultat à la sortie standard. Ainsi, tout comme [la commande sed](#), elle est permet d'effectuer énormément de choses d'autant qu'elles intègrent beaucoup de fonctions.

Dans ce tutoriel, je vous propose un large éventails d'utilisations de la commande AWK avec de nombreuses exemples.

Cela vous permet d'aborder les bases d'AWK pour améliorer considérablement votre capacité à manipuler des fichiers texte sur la ligne de commande.

INTRODUCTION À LA COMMANDE AWK

Voici la syntaxe d'AWK :

```
awk pattern { action }
```

AWK peut traiter les fichiers de données textuels et les flux. Les données d'entrée sont divisées en enregistrements et champs que l'on peut traiter à l'aide de variables. La commande fonctionne sur un enregistrement à la fois jusqu'à la fin de l'entrée. Par exemple pour lire un fichier avec AWK :

```
awk '{print $3}' monfichier.txt
```

Comme toutes les commandes Linux, AWK peut aussi prendre un entrée la sortie d'une autre commande transmise par le caractère pipe.

```
who |awk '{print $1}'
```

De ce fait, on peut aussi lire la sortie d'un fichier par [la commande cat](#) :

```
cat monfichier.txt|awk '{print $3}'
```

Les enregistrements sont séparés par un caractère appelé séparateur d'enregistrement. Le séparateur d'enregistrement par défaut est le caractère nouvelle ligne, ce qui signifie que chaque ligne dans les données de texte est un enregistrement. Un nouveau séparateur d'enregistrement peut être défini à l'aide de la **variable RS**. Les enregistrements consistent en des champs séparés par le séparateur de champs. Par défaut, les champs sont séparés par une espace blanche, comprenant un ou plusieurs caractères d'onglet, d'espace et de nouveaux caractères.

Les champs de chaque enregistrement sont référencés par **le caractère Dollar (\$)** suivi d'un numéro de champ, en commençant par 1. Le premier champ est représenté avec \$1, le second avec \$2, etc.

Le dernier champ peut également être référencé avec **la variable spéciale NF**. L'ensemble de l'enregistrement peut être référencé avec \$0.

Les variables AWK

Elle offre de nombreuses variables dont voici les principales :

VARIABLES	DESCRIPTION
RS – <i>record separator</i>	Le séparateur d'enregistrement est le délimiteur utilisé pour diviser le flux de données en enregistrements. Le délimiteur par défaut est le caractère nouvelle ligne \n.
NS – <i>record number</i>	Correspond au numéro d'enregistrement de la sortie de la commande. Si vous utilisez le délimiteur standard pour vos enregistrements (nouvelle ligne), la correspondance avec le numéro de ligne d'entrée actuel.
FNR	Le numéro d'enregistrement d'entrée dans le fichier d'entrée en cours.
FS/OFS – <i>field separator</i>	FS définit le séparateur de champs des sorties AWK. Lorsque Awk imprime une sortie, il rejoindra les champs, mais cette fois, à l'aide de OFS. Séparateur au lieu du séparateur FS. Par défaut le séparateur est le caractère espace.
ORS – <i>output record separator</i>	Séparateur d'enregistrement de la sortie
NF – <i>number of fields</i>	Si vous utilisez le délimiteur standard "espace blanc" pour vos champs, cela représente le nombre de mots dans l'enregistrement en cours. Sinon il représente le dernier champs.
FNR	Nombre d'enregistrement d'un fichier
\$0	Représente l'entière ligne d'un texte
\$1 \$2 \$8	\$1 : Représente le première champs \$2 : Représente le second champs \$8 : Représente le 8e champs
FILENAME	Nom du fichier utilisé dans la commande

Les principales variables de la commande AWK

Les actions

Les actions AWK sont encadrées dans des accolades ({}) et sont exécutées lorsque le motif correspond.

Une action peut avoir zéro ou plusieurs déclarations. Plusieurs états sont exécutés dans l'ordre dans lequel elles apparaissent et doivent être séparées par de nouvelles lignes ou des semi-colons (;).

Il existe plusieurs types d'instructions d'action qui sont soutenues dans AWK:

- Expressions, telles que l'affectation variable, les opérateurs arithmétiques, l'incrément et les opérateurs décréments.
- Déclarations de contrôle, utilisées pour contrôler le flux du programme (si, pendant, passer, etc.)
- Déclarations de sortie, telles que Imprimer et Printf.
- Déclarations composées, pour regrouper d'autres déclarations.
- Déclarations d'entrée, pour contrôler le traitement de l'entrée. Déclarations de suppression, pour supprimer les éléments de tableau.

On peut tout à fait effectuer plusieurs actions et instructions en les séparant par ; :

```
awk pattern { action }
```

La plupart du temps, on utilise **l'action print** qui permet d'afficher un champs en sortie en utilisant une variable.

Celle-ci prend les redirections de sorties comme les autres commandes.

Par exemple pour rediriger la sortie de l'action print d'AWK :

```
awk 'BEGIN { print "Hello, World !!!" > "/tmp/message.txt" }'
```

Enfin on peut aussi lui appliquer le pipe pour envoyer la sortie vers une autre commande :

```
awk 'BEGIN { print "hello, world !!!" | "tr [a-z] [A-Z]" }'
```

Enfin on sépare les actions par un point virgule ce qui permet d'effectuer plusieurs opérations dans une même commande AWK.

Par exemple ci-dessous, on effectue trois actions :

```
awk 'BEGIN { a = 50; b = 20; print "(a / b) = ", (a / b) }'
```

- On définit la valeur des variables a et b (soit deux actions),
- Puis on affiche le texte (a / b) avec print suivi par la valeur de l'opération 50 / 20

Cela affiche alors :

(a / b) = 2.5

AWK propose une gamme importantes de fonctions et opérateurs, vous trouverez une liste dans le tutoriel suivant :

[AWK : Utilisation et exemples fonctions, opérateurs et boucles](#)

COMMENT UTILISER LA COMMANDE AWK AVEC DES EXEMPLES

Dans ce tutoriel AWK, j'utilise deux fichiers employees.txt et employees.csv relativement identiques, simplement le format est différent.

```
debian@linux:~$ cat employees.txt
Riveau Alain      1919-05-22      2020-02-01      Directeur des operations      10
Cavalier Bart     1987-09-09      2020-09-01      Directeur des ventes          8
Stone Emma        1991-01-30      2021-01-02      Directrice des communication  7
Tremblay Guy      1962-02-02      2020-08-01      Vendeur 9
debian@linux:~$ cat employees.csv
Riveau Alain,1919-05-22,2020-02-01,Directeur des operations,10
Cavalier Bart,1987-09-09,2020-09-01,Directeur des ventes,8
Stone Emma,1991-01-30,2021-01-02,Directrice des communication,7
Tremblay Guy,1962-02-02,2020-08-01,Vendeur,9
debian@linux:~$ █
```

Extraire des colonnes

Pour extraire les colonnes d'un fichier, on utilise les variables champs.
Par exemple pour extraire la 3e colonne d'un fichier :

```
awk '{print $3}' employees.txt
```

```
debian@linux:~$ awk '{print $3}' employees.txt
1919-05-22
1987-09-09
1991-01-30
1962-02-02
debian@linux:~$ █
```

Pour afficher plusieurs colonnes par exemple la colonne 2 et 3 :

```
awk '{print $2 $3}' employees.txt
```

Pour rappel, sur Linux, il existe aussi **la commande cut** pour extraire les colonnes :

La commande cut Linux : utilisation et exemples

Changer le séparateur d'entrée

Par défaut le séparateur est le caractère espace, on utilise la **variable FS** pour changer ce dernier.

Par exemple, le séparateur du fichier /etc/passwd est le caractère :.
Ainsi pour extraire [le shell](#) d'un utilisateur Linux (dernière colonne), on utilise AWK de cette manière :

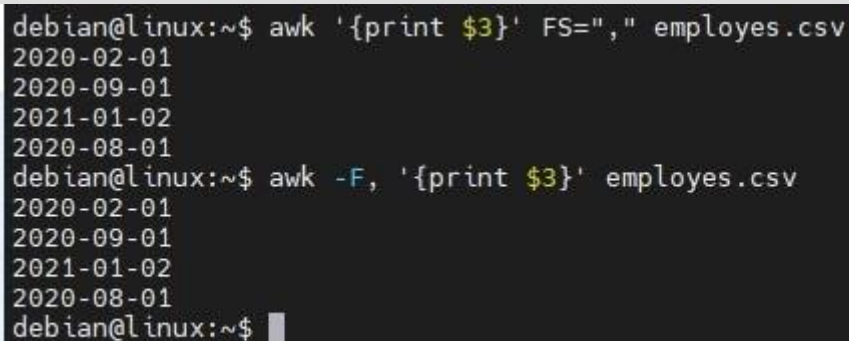
```
awk 'FS=":" {print $NF}' /etc/passwd
```

[Les fichiers CSV](#) utilisent, souvent comme séparateur, la virgule, ainsi pour extraire la seconde colonne d'un fichier CSV :

```
awk '{print $3}' FS="," employees.csv
```

On peut aussi utiliser l'option -F pour spécifier le séparateur :

```
awk -F, '{print $1}' employees.csv
```



```
debian@linux:~$ awk '{print $3}' FS="," employees.csv
2020-02-01
2020-09-01
2021-01-02
2020-08-01
debian@linux:~$ awk -F, '{print $3}' employees.csv
2020-02-01
2020-09-01
2021-01-02
2020-08-01
debian@linux:~$
```

Enfin si le séparateur est la tabulation, on demande alors à utiliser le caractère \t :

```
awk '{ print $1;}' FS="\t" employees.txt
```

Riveau Alain	1919-05-22	2020-02-01	Directeur des operations
--------------	------------	------------	--------------------------

```

debian@linux:~$ awk '{ print $1;}' employees.txt
Riveau
Cavalier
Stone
Tremblay
debian@linux:~$ awk '{ print $1;}' FS="\t" employees.txt
Riveau Alain
Cavalier Bart
Stone Emma
Tremblay Guy
debian@linux:~$ █

```

Changer le séparateur de sortie

On utilise la **variable OFS** pour changer le séparateur de sortie.

Par exemple, si le séparateur de votre fichier est l'espace, on peut le remplacer par une virgule pour avoir la structure d'un fichier CSV :

```
awk '{print $1,$2,$3}' OFS="," employees.txt
```

Pour remplacer le séparateur virgule par le point virgule :

```
awk '{print $1,$2,$3}' FS="," OFS=";" employees.csv
```

```

debian@linux:~$ awk '{print $1,$2,$3}' OFS="," employees.txt
Riveau,Alain,1919-05-22
Cavalier,Bart,1987-09-09
Stone,Emma,1991-01-30
Tremblay,Guy,1962-02-02
debian@linux:~$ awk '{print $1,$2,$3}' FS="," OFS=";" employees.csv
Riveau Alain;1919-05-22;2020-02-01
Cavalier Bart;1987-09-09;2020-09-01
Stone Emma;1991-01-30;2021-01-02
Tremblay Guy;1962-02-02;2020-08-01
debian@linux:~$ █

```

Ajouter du texte dans la sortie

On peut ajouter du texte dans l'action en l'encadrant par des guillemets.

Par exemple pour ajouter un espace entre chaque champs de sortie :

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

```

debian@linux:~$ awk '{print $1 " " $2 " " $3}' FS=, employees.csv
Riveau Alain 1919-05-22 2020-02-01
Cavalier Bart 1987-09-09 2020-09-01
Stone Emma 1991-01-30 2021-01-02
Tremblay Guy 1962-02-02 2020-08-01
debian@linux:~$ █

```

Pour ajouter une phrase dans la sortie AWK :

```
awk '{ print "Le premier champs:", $1}' employees.txt
```

```

debian@linux:~$ awk '{ print "Le premier champs:", $1}' employees.txt
Le premier champs: Riveau
Le premier champs: Cavalier
Le premier champs: Stone
Le premier champs: Tremblay
debian@linux:~$ █

```

Enfin pour ajouter du texte avec un retour à la ligne, on utilise le caractère `\n` :

```
awk '{ print "Le premier champs:", $1 "\n" "Le second champs:", $2}'
```

```
employees.txt
```

```

debian@linux:~$ awk '{ print "Le premier champs:", $1 "\n" "Le second champs:", $2}' employees.txt
Le premier champs: Riveau
Le second champs: Alain
Le premier champs: Cavalier
Le second champs: Bart
Le premier champs: Stone
Le second champs: Emma
Le premier champs: Tremblay
Le second champs: Guy
debian@linux:~$ █

```

Recherche de texte et extraction de données

Si vous souhaitez n'afficher que des colonnes dont sa valeur est égale à une valeur spécifique, vous pouvez utiliser `if-else`.

```
awk '{ if($2 == "1919-05-22") print $0;}' FS="\t" employees.txt
```

```
Riveau Alain      1919-05-22      2020-02-01      Directeur des operations
```

```
10
```

```
awk '{ if($3 == "B6") print $0;}' geeksforgeeks.txt
```


AWK peut aussi effectuer une recherche de motif, cela est utile pour sortir des champs avec des valeurs spécifiques.

```
/motif/ { action }
```

Afficher le premier, second et dernier champs qui contiennent la valeur 1991.
Utile par exemple pour rechercher une date.

```
awk '/1991/ {print $1 " " $2 " " $NF}' employees.txt
```

```
debian@linux:~$ awk '/1991/ {print $1 " " $2 " " $NF}' employees.txt  
Stone Emma 7  
debian@linux:~$ █
```

Si l'on veut rechercher dans un champs en particulier, on peut le stipuler puis utiliser le caractère ~ pour la recherche de motif.

Par exemple pour afficher le champs 1, 2 et 3 où le 3e champs contient la valeur 1987 :

```
awk '$3 ~ /1987/ { print $1 " " $2 " " $3}' employees.txt
```

Au contraire, pour exclure un champs qui contient une valeur, on utilise le caractère !~ :

```
awk '$3 !~ /1987/ { print $1 " " $2 " " $3}' employees.txt
```

```
debian@linux:~$ awk '$3 ~ /1987/ { print $1 " " $2 " " $3}' employees.txt  
Cavalier Bart 1987-09-09  
debian@linux:~$ awk '$3 !~ /1987/ { print $1 " " $2 " " $3}' employees.txt  
Riveau Alain 1919-05-22  
Stone Emma 1991-01-30  
Tremblay Guy 1962-02-02  
debian@linux:~$ █
```

Enfin vous pouvez utiliser des opérateurs de relation (==,>=, <=,>, <,! =) Pour effectuer une comparaison numérique. Ici, nous effectuons un chiffre supérieur ou égal (>=) pour imprimer les lignes qui ont la valeur 22 ou plus dans le troisième champ, comme suit:

```
awk ' $3 >= 22 {print NR, $0 } ' fichier.txt
```

On peut aussi utiliser [les expressions régulières](#) de cette manière :

```
/regex motif/ { action }
```

Pour afficher le contenu d'un fichier dont les données de la première colonne commencent par la lettre A (avec la majuscule). Pour cela on utilise le **caractère ^** pour correspondre au début de la donnée :


```
awk '$1 ~ /^A/ {print $0}' employees.txt
```

On utilise le **caractère \$** pour tester la fin d'une donnée.

Dans les exemples suivants, nous combinons avec l'opérateur de match (~) pour imprimer toutes les lignes de la deuxième extrémité du champ avec le caractère «e», comme suit:

```
awk ' $2 ~ /e$/ {print NR, $0 } ' fichier.txt
```

Voici un autre exemple pour afficher le premier champs d'un fichier qui contient une date au format JJ-MM-AAAA :

```
awk '/([0-9]{2}-){2}[0-9]{4}/ {print $1}' employes.txt
```

Rechercher par une date au format AAAA-MM-JJ :

```
awk '/[0-9]{4}-([0-9]{2}-){2}/ {print $1}' employes.txt
```

```
debian@linux:~$ awk '/[0-9]{4}-([0-9]{2}-){2}/ {print $1}' employes.txt
Riveau
Cavalier
Stone
Tremblay
debian@linux:~$ █
```

Awk permet aussi d'extraire des données entre deux motifs de cette manière :

```
awk '/motif1/,/motif2/ { print $1 }' employe.txt
```

Extraire des données avec valeurs supérieures/inférieures à

On demande à afficher toutes les lignes dont la valeur du dernier champs est supérieure à 8.

```
awk '$NR > 8 {print $0 }' employes.txt
```

Pour afficher toutes les lignes dont la valeur du dernier champs est comprise entre 5 et 10 :

```
awk '$NR > 5 && $NR < 10 {print $0 }' employes.txt
```

Par exemple, ici, on extrait les adresses IP du journal de NGINX (1er champs) et on demande à awk de n'afficher que les valeurs supérieures à 50.

```
tail -10000 /var/log/nginx/access.log|awk '{print $1}'|sort -n|uniq -c|awk '$1  
> 50 {print $1 " " $2}'
```

Rechercher/Remplacer avec AWK

Le **rechercher/remplacer avec AWK** est possible grâce à la fonction gsub qui utilise une syntaxe similaire à [sed](#) s///g.

Ici on remplace le mot Ubuntu par Débian :

```
echo "J'aime Ubuntu"| awk '{ gsub(/Ubuntu/,"Debian"); print $0}'  
  
J'aime Debian
```

Dans cet autre exemple, gsub(/,/, " ") remplace la virgule par le caractère espace.

```
awk '{ gsub(/,/, " "); print $1 $2 $3}' FS=, employees.csv
```

```
debian@linux:~$ awk '{ gsub(/,/, " "); print $1 $2 $3}' FS=, employees.csv  
Riveau Alain 1919-05-22 2020-02-01 Directeur des operations 10  
Cavalier Bart 1987-09-09 2020-09-01 Directeur des ventes 8  
Stone Emma 1991-01-30 2021-01-02 Directrice des communication 7  
Tremblay Guy 1962-02-02 2020-08-01 Vendeur 9  
debian@linux:~$
```

Comment utiliser les règles BEGIN et END d'AWK

Une règle BEGIN est exécutée une fois avant que tout traitement de texte ne commence. En fait, il est exécuté avant que AWK ne lit même aucun texte.

Une règle END est exécutée après la fin du traitement. Vous pouvez avoir plusieurs règles de début et de fin, et ils seront exécutés dans l'ordre.

Cela est utile pour afficher un élément en début ou fin, par exemple lorsque l'on souhaite effectuer des opérations mathématiques pour afficher le résultat.

Par exemple, on peut utiliser BEGIN pour **afficher un texte en tout début de sortie AWK**.

```
who | awk 'BEGIN {print "Sessions actives"} {print $1,$4}'
```

OU encore il est possible d'**afficher le nombre de lignes d'un fichier** :

```
awk 'END { print "Fichier", FILENAME, "contient", NR, "lignes." }' employe.txt
```

Extraire des lignes d'un fichier

Par défaut le séparateur de sortie est le caractère nouvelle ligne.
Ainsi avec la commande AWK, on utilise la variable NR pour cibler les lignes d'un fichier.

Par exemple pour **extraire à partir de la seconde ligne** et donc supprimer l'en-tête d'un fichier, on demande à AWK d'afficher les lignes supérieures à 1 :

```
awk 'NR > 1' fichier.txt
```

Pour afficher un fichier à partir de la 10e ligne :

```
awk 'NR > 10' fichier.txt
```

On peut aussi **extraire un intervalle de lignes avec AWK**.

Par exemple pour extraire de la première ligne à la quatrième ligne d'un fichier :

```
awk 'NR>1 && NR < 4' fichier.txt
```

Afin d'**afficher les numéros de lignes d'un fichier**, on utilise la variable FNR :

```
awk '{ print FNR, $0 }' employes.txt
```

Ou encore avec NR:

```
awk '{ print NR, $0 }' employes.txt
```

```
debian@linux:~$ awk '{ print FNR, $0 }' employes.txt
1 Riveau Alain 1919-05-22 2020-02-01 Directeur des operations 10
2 Cavalier Bart 1987-09-09 2020-09-01 Directeur des ventes 8
3 Stone Emma 1991-01-30 2021-01-02 Directrice des communication 7
4 Tremblay Guy 1962-02-02 2020-08-01 Vendeur 9
debian@linux:~$ awk '{print NR,$0}' employes.txt
1 Riveau Alain 1919-05-22 2020-02-01 Directeur des operations 10
2 Cavalier Bart 1987-09-09 2020-09-01 Directeur des ventes 8
3 Stone Emma 1991-01-30 2021-01-02 Directrice des communication 7
4 Tremblay Guy 1962-02-02 2020-08-01 Vendeur 9
debian@linux:~$
```

Supprimer les lignes blanches avec AWK

Pour **supprimer les lignes vides d'un fichier**, on demande à afficher toutes les lignes dont le séparateur d'enregistrement est vide.

Ainsi dans l'exemple ci-dessous, on affiche toutes les lignes du fichier avec '1' mais avec RS vide.

```
awk '1' RS='' Fichier
```

Compter le nombre de lignes

La fonction COUNT permet de compter le nombre d'éléments que l'on peut ensuite afficher grâce à END.

Pour **compter le nombre de lignes non vides d'un fichier** :

```
awk '/./ { COUNT+=1 } END { print COUNT }' employes.txt
```

Convertir du texte en minuscules en majuscules

On utilise la **fonction toupper** pour passer la sortie en majuscules.

Pour passer tout un fichier en majuscules :

```
awk '{print toupper($0)}' employes.txt
```

Par exemple pour passer la première colonne en majuscules :

```
awk '{print toupper($1) " " $2 " " $3}' FS=, employes.csv
```

```
debian@linux:~$ awk '{print toupper($1) " " $2 " " $3}' FS=, employes.csv
RIVEAU ALAIN 1919-05-22 2020-02-01
CAVALIER BART 1987-09-09 2020-09-01
STONE EMMA 1991-01-30 2021-01-02
TREMBLAY GUY 1962-02-02 2020-08-01
debian@linux:~$ █
```

Additionner les valeurs d'une colonne avec AWK

La fonction SUM permet de faire des sommes de valeurs et notamment de colonnes.
Par exemple, pour **additionner la valeur de la 5e colonne** :

```
awk '{ SUM=SUM+$5 } END { print SUM }' FS=, OFS=, employes.csv
```

Pour **afficher les valeurs de la colonne** :

```
awk '{ SUM=SUM+$5; print $5} END { print SUM }' FS=, OFS=, employees.csv
```

Enfin pour afficher le numéro de la ligne avec la valeur de la colonne et le total :

```
awk '{ SUM=SUM+$5; print "Ligne: "NR " "$5 } END { print "Le total est : "SUM }' FS=, OFS=, employees.csv
```

Ligne: 1 10

Ligne: 2 8

Ligne: 3 7

Ligne: 4 9

Le total est : 34

```
debian@linux:~$ awk '{ SUM=SUM+$5 } END { print SUM }' FS=, OFS=, employees.csv
34
debian@linux:~$ awk '{ SUM=SUM+$5; print $5} END { print SUM }' FS=, OFS=, employees.csv
10
8
7
9
34
debian@linux:~$ awk '{ SUM=SUM+$5; print "Ligne: "NR " "$5 } END { print "Le total est : "SUM }' FS=, OFS=, employees.csv
Ligne: 1 10
Ligne: 2 8
Ligne: 3 7
Ligne: 4 9
Le total est : 34
debian@linux:~$
```

LIENS

Comment afficher le contenu d'un fichier texte sur Linux

- [12 exemples pour utiliser la commande cat sur Linux](#)
- [6 exemples pour utiliser la commande tail sur Linux](#)
- [Comment utiliser la commande less sur Linux avec des exemples](#)
- [La commande cut de Linux : utilisation et exemples](#)
- [Comment utiliser la commande AWK avec des exemples](#)
- [Commande sed : utilisation et exemples](#)

- [12 exemples de commandes grep sur Linux](#)
- [La commande TR : utilisations et exemples](#)
- [La commande echo : utilisations et exemples](#)

Liste des commandes Linux

Source(s) : <https://linuxhandbook.com/awk-command-tutorial/>
<https://linuxize.com/post/awk-command/>
<https://www.howtogeek.com/562941/how-to-use-the-awk-command-on-linux/>